Paolo Nicoli 833311
Andrea Sassanelli 835119

Text Mining and Search Project

Text Classification on 20newsgroup dataset

Introduction

This project compares the performances of various algorithms on a single-label multi-class classification task on the 20newsgroup dataset. To this end a python program was developed, including data loading functions and parametrized routines to train and evaluate different algorithms.

Dataset Description

The 20newsgroup dataset [1] is a collection of approximately twenty thousand documents extracted from Usenet newsgroups. The dataset is partitioned in order to group together documents from the same newgroups and presents 20 such categories, hence the name.

Newsgroups used to follow a strict thematic taxonomy which also is present in the dataset classes. Not all top-level taxa have equal representation in the dataset though, leading to unbalanced classes while using only top level classification.

Top	Fully qualified group name	Docs	Total
alt	alt.atheism	480	480
comp	comp.graphics	584	2936
	comp.os.ms-windows.misc	591	
	comp.sys.ibm.pc.hardware	590	
	comp.sys.mac.hardware	578	
	comp.windows.x	593	
misc	misc.forsale	585	585
rec	rec.autos	594	2389
	rec.motorcycles	598	
	rec.sport.baseball	597	
	rec.sport.hockey	600	
sci	sci.crypt	595	2373
	sci.electronics	591	
	sci.med	594	
	sci.space	593	
soc	soc.religion.christian	599	599
talk	talk.politics.guns	546	1952
	talk.politics.mideast	564	
	talk.politics.misc	465	
	talk.religion.misc	377	

It must be noted that the documents in the dataset are full Usenet articles comprehensive of headers, quoted text and author signatures. This peculiarity needs to be properly addressed during the analysis since group-specific headers and signatures can bias the models.

Headers

Each document starts with a section containing various post attributes and metadata called headers, as defined in the Network News Transfer Protocol specifications [2]. Each header follows a key:value syntax, allowing for consistent pattern matching using regular expressions.

Quoted Text

Since Usenet newsgroups were often avenues of debate many posted articles contained rebuttals and replies to previous posts. In order to facilitate communication when writing rebuttals authors often copied part of the original post in their replies. While not explicitly codified in the standard, it became common practice to identify this quoted text by prepending each line with a “>” character. As with the headers this syntactic rule allows for pattern matching using regular expressions.

Signatures

Some authors personalized their posts by appending customized signatures. Since there is no consistency in the signature syntax a heuristic approach is used for their identification, flagging the last part of a document if separated by spaces of sequences of hyphens.

Train-Test splitting

The dataset provides a precomputed split between training and testing samples using a temporal criterion. This version of the dataset also removes some of the headers (Xref, Newsgroups, Path, Followup-To, Date) to partially mitigate their bias. We chose this version to simulate an online system deployed in production after training.

Pre-processing

The documents were pre-processed by applying tokenization, case folding and stemming. Additionally any token that contained non-alphabetical characters was removed, as were all stopwords occurrences.

The aforementioned peculiarities of the dataset were also addressed by implementing a custom removal step for headers, quotes and signatures of the documents.

Pre-processing pipeline:

1. Head|Quote|Signature removal
2. Tokenization
3. Case folding
4. Non-alphabetical pruning
5. Stopwords removal
6. Stemming

Implementation

For most of the pre-processing the `nltk` library was used: for tokenization a combination of the Treebank and Punkt algorithms [3] was applied; for stemming an implementation of the Porter algorithm [4] ; for stopwords removal the list provided by the library was used to identify and drop stopwords tokens after the normalization steps.

To remove headers, quotes and signatures custom functions were defined using as reference those implemented in the `scikit-learn` library [5]. These functions rely on line-level structures to identify what to remove; for this reason they are defined to operate on raw documents, splitting on newline characters to apply filtering criteria and concatenating the resulting lines into a new processed document. This arrangement allows for modularity in the filtering process, operated through a tuple of Boolean values.

After the Head|Quote|Signature filtering takes place the pre-processing pipelines continues with the application of tokenization, which from the (reconstructed) document produces a list of tokens. On this list pruning of non-alphabetical elements is applied to remove punctuations, special characters and numbers and decrease the noise inherent from an informal communication channel like newsgroup posts. A blacklist of stopwords is applied during the same step to reduce time complexity. Lastly a stemming procedure is applied to the pruned list of tokens to further reduce the variability of the terms.

As the pre-processing pipeline is applied document-wise during dataset loading the final output is an ordered list of tuples, one for each document, each composed by the Document ID, its Newsgroup class and its extracted list of terms.

While this is not the most efficient implementation possible it was preferred because it allowed to explore and implement each step of the procedure instead of fully relying on opaque library methods.

Representation

The representation chosen for the classification task is a Document-Term matrix with Tf-Idf weights. In order to decrease the sparsity of the resulting matrix a lower limit is applied to the document frequency of included terms. To reduce noise from low-information terms un upper limit is also applied, removing terms present in more than the allowed fraction of documents.

A coarse parameter sweep was conducted to identify an optimal configuration: the minimal document frequency has the stronger impact on the number of terms and matrix sparsity, while the upper limit does not have any impact on either sparsity or term frequency for the values explored as not term reaches the explored cutoff values.

Min DF (docs)	Max DF (frac)
1	0.75
10	0.80
20	0.90

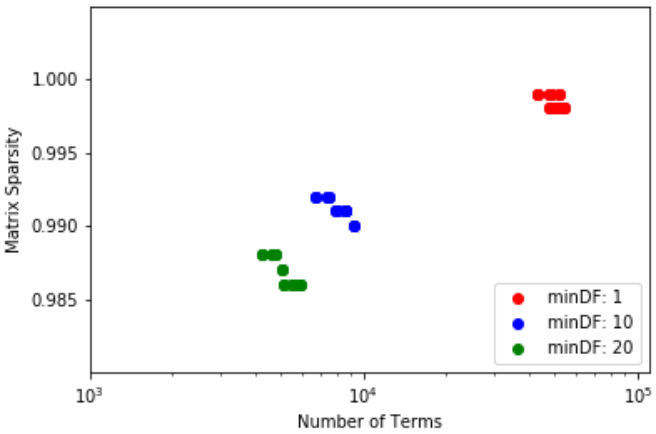


***Figure 1**: Scatterplot of matrix sparsity vs number of terms. The points refer to representations with different filtering flags. MaxDF values are not reported as their points perfectly overlap*

Implementation

To compute the Document-Term matrix and its Tf-Idf weights objects from the `scikit-learn` library [6] were used, fitted only on the training sample and applied to the testing sample as-is in order to simulate the performances of an online system.

As the objects used to build the Document-Term matrix operate on whole documents the list of tuples produced during pre-processing were reassembled into pseudo-documents composed only by the pre-processed terms. While again this is not an efficient implementation it allowed for hands-on experience with each pre-processing step.

Models

Four types of classification models were investigated:

- Multinomial Naïve Bayes [7]
- Support Vector Machine [8]
- K Nearest Neighbours [9]
- MultiLayer Perceptron [10]

The hyperparameter configurations were not investigated, using default configurations while instead focusing the analysis on the classifier performances on different pre-processing scenarios.

As evaluation metrics for the models accuracy on the testing sample and training time were used.

Implementation

For model implementation the `scikit-learn` library was used, leveraging its standardized framework. Since each algorithm uses the same calls for training and evaluation and since the

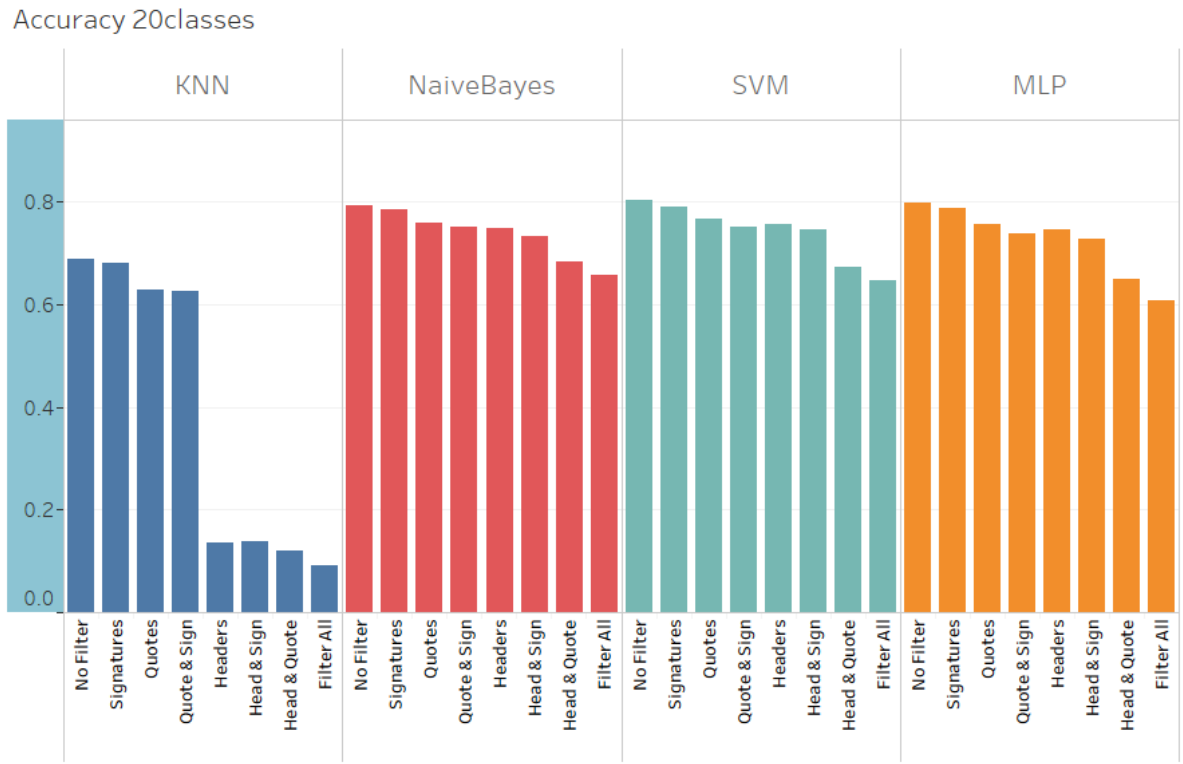
python language allows expansion of dictionaries into keyword arguments through the unpacking operator a fully parametrized execution of was possible.

The main routine is thus a nested for-loop iterating over each desired value for the pre-processing parameters with, at its core, another loop iterating on the keys of a “schedule” dictionary where the models and their relative hyperparameters are stored. This allows to run the pre-processing pipeline once and evaluation of all the models on the same resulting representation. Outputs from the models are written to file, together with the pre-processing configuration to enable offline analysis.

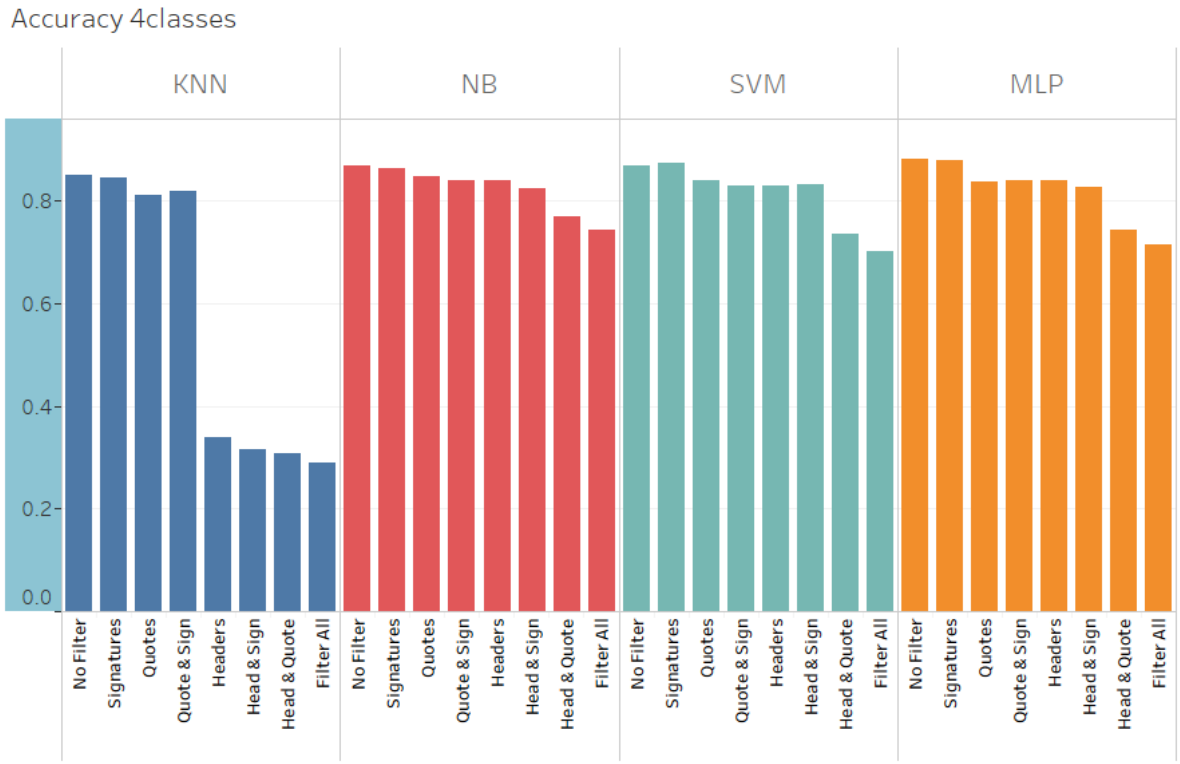
Results

Two sweeps over pre-processing filter flags were conducted, exploring the performances of the classifiers for two target problems: the classification over all 20 newsgroup classes and over 4 classes belonging to different taxa.

The results present a strong evidence that more pre-processing filtering degrades classifier performances, independently from the number of target classes. This effect is well documented in literature as newsgroup metadata contained in the headers and user specific tokens like mail addresses and signatures act as strong class indicators due to user habit but are not properly representative of the topics discussed in the newsgroups.
This phenomenon is particularly evident in the KNN classifier which completely fails as soon as header filtering is introduced.



Another result worth mentioning is how reducing the scope of the classification from 20 classes to just four improves performance for all classifiers, allowing all models to break the 80% accuracy barrier at least for scenarios with low filtering, but still does not provide a strong advantage to any model against the Naïve Bayes which retains the overall advantage. Its near competitor would be the MultiLayer Perceptron but its slim accuracy improvement (0.004) cannot justify the training time increase of over 350 times from an average 0f 0.1 seconds to over 35 seconds.



Conclusions

On simple classification tasks against out-of-the-box models the Naïve Bayes algorithm retains first place. While further analysis can be conducted by tuning hyperparameter configurations the advantage remains for fast prototyping and Pay-As-You-Go implementations.

As reported in literature pre-processing filtering has a strong impact on classification tasks for this dataset and it should be taken into account when developing online systems that may rely on metadata.

References

[1] http://qwone.com/~jason/20Newsgroups/.

[2] https://tools.ietf.org/html/rfc3977.

[3] https://www.nltk.org/api/nltk.tokenize.html.

[4] https://www.nltk.org/api/nltk.stem.

[5] https://scikit-learn.org/0.19/datasets/twenty_newsgroups.html.

[6] https://scikit-learn.org/stable/modules/classes.html#module-sklearn.feature_extraction.text.

[7] https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html.

[8] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html.

[9] https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html.

[10] https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html.