Exercise 1 Classification

e12045110 Maria de Ronde
e12045110 Quentin Andre
e11921655 Fabian Holzberger

April 25, 2021

Introduction

In this project we analyze the performance of three traditional classification algorithms on 4 significantly different datasets. We aim to show which algorithm performs best on a specific dataset and further if one algorithm outperforms the others for all datasets. In the next section we first describe the applied algorithms and then the performance metrics that we have chosen. Then a detailed analysis of all algorithms on each dataset is done, which concludes with the final performance comparison.

Applied Algorithms

Assume $\hat{x} \in \mathbb{R}^d$ is a data sample, for that we want to predict a the label $\hat{y} \in \{0, 1\}$. For our algorithms we use a finite dataset $S \subset \mathbb{R}^d$ with $|S| = N$ samples.

Perceptron

[2] We use the following linear rule for classification:

$$f(x) = \begin{cases} 1 & \text{if } (w, x) + b > \theta \\ 0 & \text{else} \end{cases} \tag{1}$$

where $(\cdot, \cdot)$ is the scalar product, $w \in \mathbb{R}^d$ is a weight vector, $b \in \mathbb{R}$ is a bias and $\theta \in \mathbb{R}$ a threshold. By the value we obtain for evaluating $f(\hat{x})$ we are able to predict the label $\hat{y}$, that is if it is in class 0 or in class 1. The weight w and bias b have to be learned by an iterative algorithm with complexity $\mathcal{O}(d|S| \text{ maxiter})$ where maxiter is the maximum number of iterations we perform to fit the weight and bias. After the evaluation can be done in $\mathcal{O}(d)$ since only the scalar product has to be evaluated.
The Perceptron is by that a very cheap classifier in therms of training and evaluation. We expect no overfitting since the decision boundaries are linear.

Random Forrest

[2] This classifier is based on the decision tree algorithm. The difference is that an ensemble of k decision-trees is constructed by a construction algorithm of choice, applied on the training set S. The trees are constructed by generating a random parameter θ from some distribution that then further creates the random subset $S_\theta \subset S$ that we use to build a tree. After constructing all trees, we classify $\hat{x}$ by evaluating all trees and then voting by majority.
To construct a random forest from S we have the complexity [1] $\mathcal{O}(k|S|\log(|S|)d)$. The evaluation complexity is then only dependent on the maximum depth and the number of the trees [1] $\mathcal{O}(k \text{ maxdepth})$. The random

forest algorithm is capable of building a complex decision boundary while weakening the overfitting, often observed when using only a single decision tree. Compared to the other algorithms used in this project it is in training and evaluation the most expensive algorithm.

Naive Bayes (Multinomial Bayes)

[2] W.l.o.g. assume $x \in \{0,1\}^d$. We assume that the label and features x_i are independent of each other such that we can calculate the probability of sample x having label y as:

$$\mathcal{P}(X = x|Y = y) = \prod_{i=1}^{d} \mathcal{P}(X = x_i|Y = y) \tag{2}$$

Together with the probability $\mathcal{P}(Y = y)$ of label y occurring we can formulate the binary classifier:

$$g(x) = \max_{y\in\{0,1\}} \mathcal{P}(Y = y) \prod_{i=1}^{d} \mathcal{P}(X = x_i|Y = y) \tag{3}$$

And, therefore, as in the Perceptron algorithm the binary function g classifies $\hat{x}$ by $g(\hat{x})$. Naive Bayes has a training complexity of $\mathcal{O}(|S|d)$ and a evaluation complexity of $\mathcal{O}(cd)$ where c is the number of classes we predict. It is by that an easy to implement and fast algorithm that yields good results when applied on natural language processing, as two of our datasets are.

Performance Metrics

This chapter summarizes the metrics that are used for performance evaluation of the algorithms in this project. The definitions can be found in [2].

$$\text{Percision } = \frac{tp}{tp+fp}, \quad \text{Recall } = \frac{tp}{tp+fn} \tag{4}$$

$$\text{Accuracy } = \frac{tp+tn}{tp+tn+fp+fn}, \quad \text{F1 } = \frac{2}{\text{Percision}^{-1}+\text{Recall}^{-1}} \tag{5}$$

For the comparison of the algorithms, we use the F1-score since it can be applied in a meaningful way for many scenarios. This is not the case for precision and recall. When comparing between the F1 scores of multiclass classifiers, we will calculate the macro averages F1 score.

Amazon Reviews Dataset

Dataset Description

Pre-Processing

Parameter-Tuning

Performance-Analysis

Congressional Voting Dataset

Dataset Description

Pre-Processing

Parameter-Tuning

Performance-Analysis

Email Spam Dataset(link to dataset)

Dataset Description

The task of the email spam dataset is to predict if an email is spam or not. The link above contains three datasets from which we choose two, namely the `lingSpam.csv` and `enormSpamSubset.csv` for our project, since they have no missing values and the same layout. The dataset contains 12604 emails where 43.11

Index	Body	Label
100	Subject: inexpensive online medication here pummel wah springtail cutler bodyguard we ship quality medications overnight to your door !...	1
6006	Subject: organizational changes we are pleased to announce the following organizational changes : enron global assets and services in order to increase senior management focus on our international businesses...	0

Figure 1: Structure of the Email-Spam Dataset

Every email has a binary target-label assigned, such that a 0 marks non-spam and a 1 marks spam emails. In figure 2 the distribution of the characters per email is shown. We see that most emails have a lenght in the range of 100 to 10.000 characters.

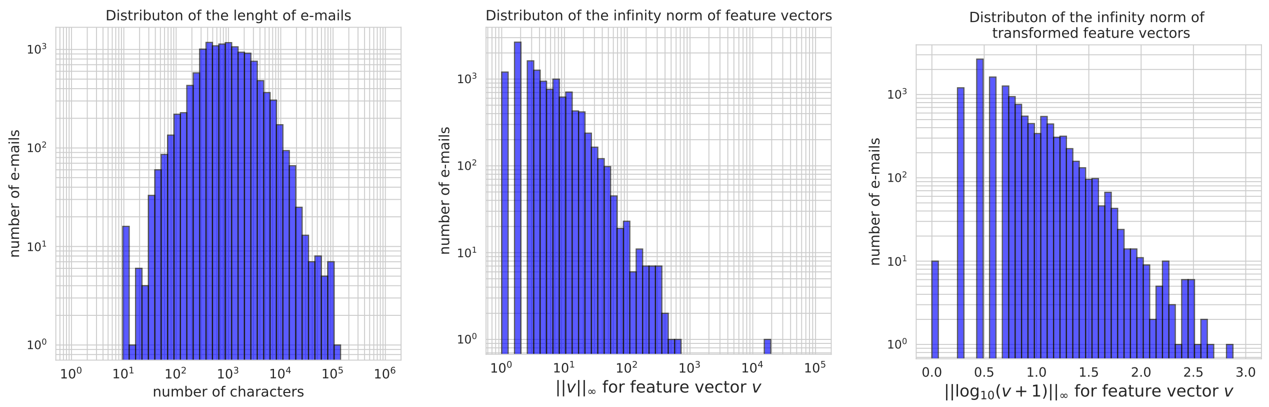

Figure 2: left: Distribution of e-mail lenghts, middle: Distribution of maximum norm in extracted word vectors with lenght 8000, right: Distribution of maximum norm in extracted word vectors with lenght 8000 after removing outlyers and applying logarithmic transformation

Pre-Processing

In the dataset are 213 duplicate emails that we first remove and then peform the train/test-split where the trainset size is 20% of the original dataset. Next we apply the Bag of Words feature extractor to each email. The algorithm converts every email to a vector $v \in \mathbb{Z}^N$ of intergers. First we create a list of all words and count their occurences in all emails. Then we take the N most common words and count the occurences of the most common words in every email to get v. Before applying the Bag of word extractor we pre-process emails by the following steps:1. remove links (http...), 2. remove all characters exept alphabetical chars and numbers, 3. convert uppercase to lowercase, 4. split text-bodys into separate words, 5. lemmatize all words, 6. remove stopwords. For the steps 5., 6. we use nltk python package. By the preprocessing we reduce the number of distinct words from 126019 to 103759 words.
In figure 2 middle we see the distribution of the maximum norm of the extracted vectors. One can identify that the maximum norm spans several orders of magnitude from 0 to 10^5. Especially there is only one vector v with $||v||_\infty > 10^4$. Outlyers with $||v||_\infty > 10^3$ are therefor removed in the testset. Additionally we apply the logarithmic transformation $\log_{10}(x+1)$ to all the elements of a vector and obtain a $||\cdot||_\infty$ distrubution that is bounded by the maximum magnithude. Note that we add 1 to all components of a vector since this component is 0 after the logarithmic transformation. The distribution after the transformation is shown in figure 2 right.

Parameter-Tuning

For the parameter Tuning we further split the trainingset into a 20% validation and 80% trainingset. The model fitting is done on the trainingset and the parametertuning with regard to the performance on the validation set. We select parametervalues by the cross validation performance, since we assume this performance if more stable when compared to the holdout validation, that might be to optimistic ore pessimistic. Since the task is to distinguish spam and non-spam emails and we assume that misslabeling important non-spam emails is just as importnant as not misslabeling potentially harmfull spam emails we make use of the F1-score as performance metric. This further makes sense since our dataset is relatively balanced as stated above.
Perceptron: We compare cross-validation with 10 splits to a holdout-validation. In figure 3 top left we see the influence of the scaling method on the f1-score of the perceptron algorithm. The binary scaling means that the transformed vectors have value 1 in a component if the original vectors component is nonzero and 0 otherwhise. Binary and logarithmic sclaing have the best performance in the cross validation, where the logarithmic scaling has slightly better performance on the trasining set. Therefore we choose the logarithmic

scaling for the dataset. In figure 3 top middle the influence of the extracted features on the perceptron algorithm with default parameters is shown. The performance for the perceptron is optimal for 2000 features since the f1-score saturates for this feature number. Therefore we adapt this number of features and investigate the learing rate in figure 3 top right. Here the optimal learing rate is 0.1 if we consider the cross-validation performance on the validaton set. Note that the holdout performance is not optimal for this value. The influence of the tolarance can be seen in 3 bottom left. For values lower than 0.001 it has a constant f1-score that dropt for the cross validation on the validation set when we increase it. We can therefore fix the tolarance at 0.001 and investigate the maximum number of terations in 3 bottom middle. Surprisingly a low number of iterations causes the f1-score to increase. We fix the maximum iteration for this purpose to 10.

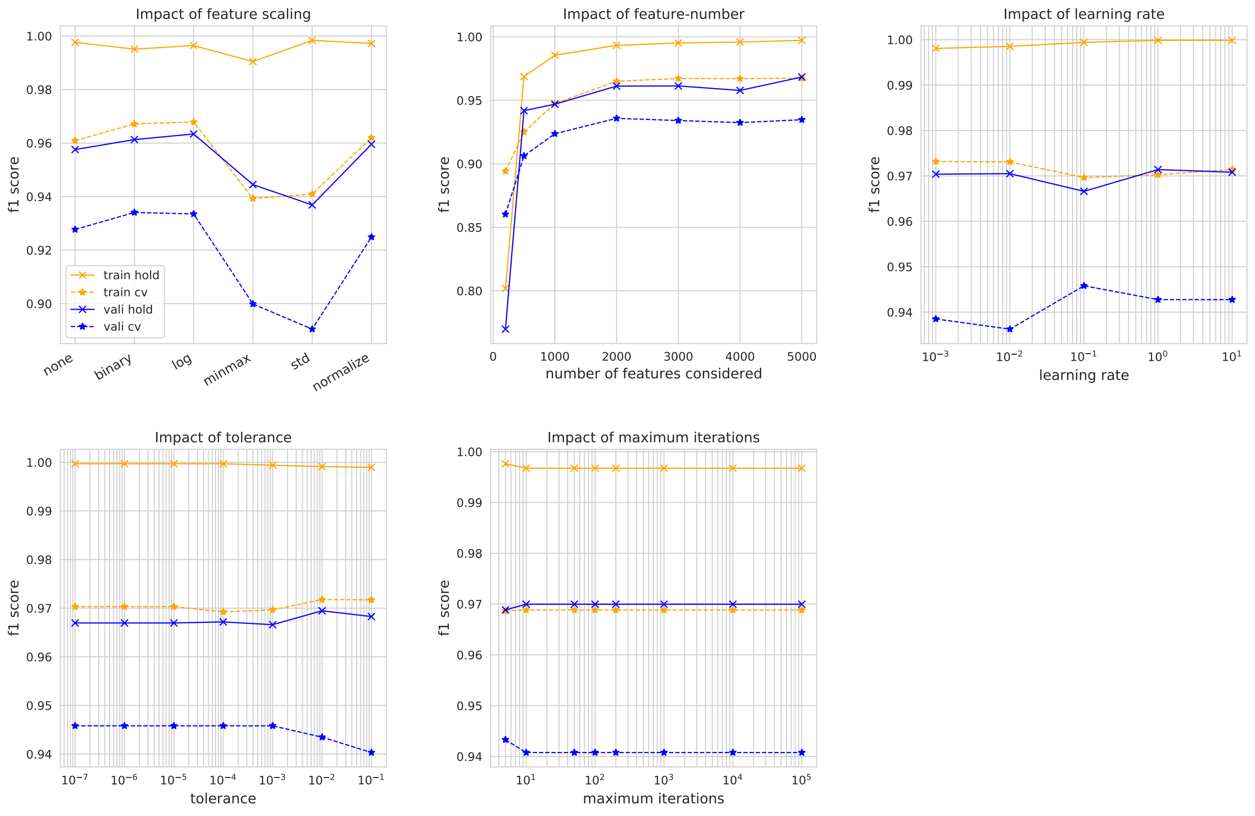

Figure 3: F1-score of Perceptron for different pre-processing parameters and model-parameters

For the perceptron the cross validation yields always a more pessimistic performance when compared to the holdout-validation. As optimal parameters we have chosen: scaling method: logarithmic, extracted features: 2000, learning rate: 0.001, tolerance: $1e-6$. The f1-score on the testset is for this parameters: 0.97 for the holdout validation and 0.95 for the cross validation.

Random forrest: Since the random forrest has much larger runtime compared to the other algorithms in this project we will not evaluate its performance on the training set and further set the number of validation steps in the cross validation to 4.

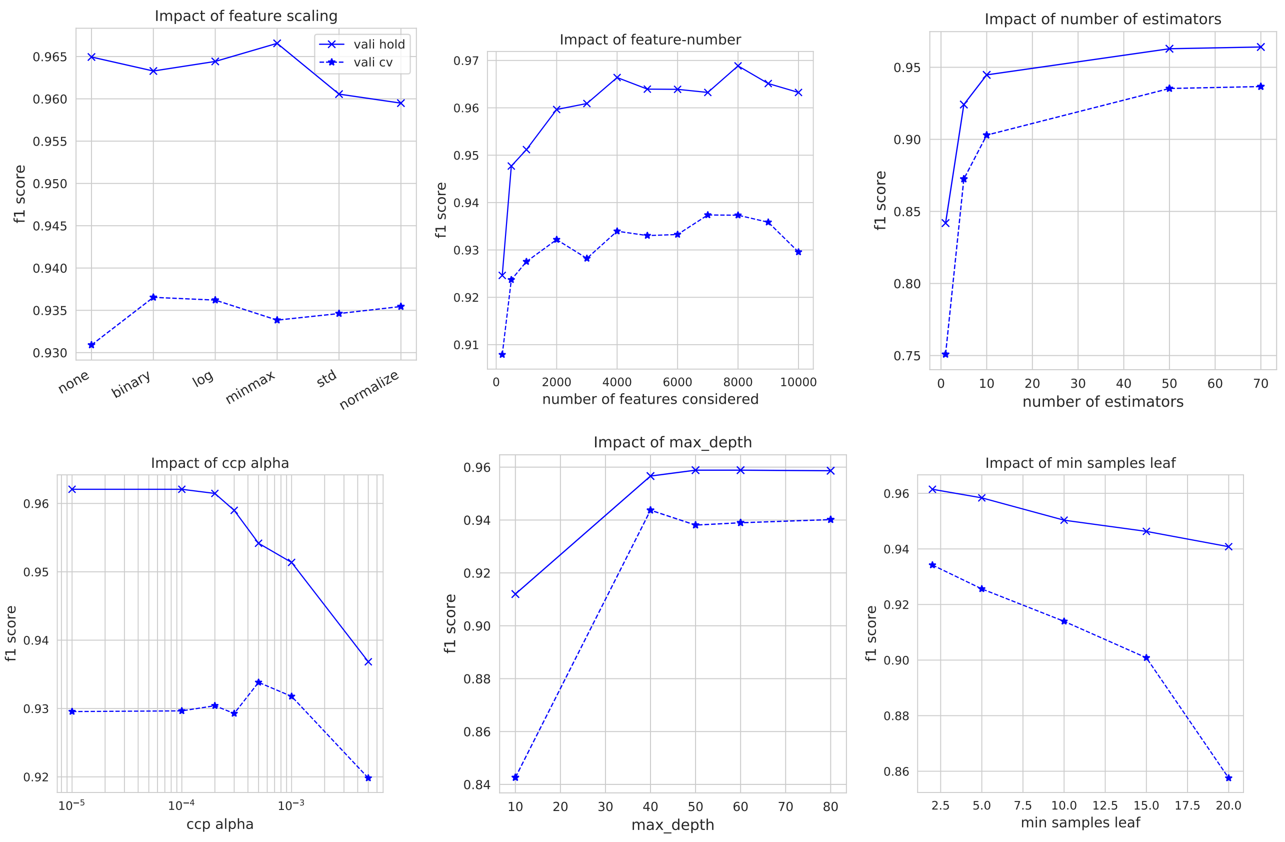

Figure 4: F1-score of Random-Forrest for different pre-processing parameters and model-parameters

The random forrest algorithm is usually not affected in its performance by scaling. Nevetheless we inspect the influence of the scaling parameters in at top left with 8000 features extracted. We see that no scaling gives even for this algorithm less performance. Since the difference for the cross-validation is small we keep the unscaled data for this algorithm. When considering the impact of the number of extracted features in top middle the f1-score in the cross validation improves till 7000 features are reached and then drops again. Therefore to decrease the runtime as much as we can we select 7000 features even if the holdout f1-score here is clearly lower. Another important parameter for performance and runtime in the random forrest is the number of trees that are build. The performance of this parameter can be seen in top right. The performance increases when we build more decision trees but at 50 trees this preformance increase is not significant anymore. Therefore to decrease the runtime we select 50 trees for further investigations but 70 trees when stating the optimal parameters below. So far the decision trees are grown to full lenght and might overfit largly. In bottom left we perform cost-complexity pruning on the tree. Larger values of the parameter will post-prune more tree branches. At $5e-3$ we reach an optimum in the cross validation f1-score which doesent differ significantly from the other performances achived for different values of this parameter. We still want to select parameter as large a possible to simplify the tree and prevent overfitting for the generqal case such that we select $5 \cdot 10^{-3}$ for this parameter. We can also prevent the tree from overfitting by bounding its depth. The impact of this method is shown in bottom right. Here we obtain a optimum for a maximal tree depth of 15. The last parameter to prevent overfitting we investigate is the minimum number of spamples required to split a leaf into a new branch. This parameters performance impact can be seen in bottom right. Here increasing the parameter values drastically decreases the f1-score and therefore we dont make use of it.
In our analysis we obtained the best performance with :scaling method: none, extracted features: 8000, cost complexity pruning parameter $5 \cdot 10^{-3}$, maximum depth: 60, number of trees: 70. By that we obtain on the

testset a f1-score of 0.93 for holdout and cross-validation with 5 validation steps.
Naive Bayes: Since this algorithm has a much smaller runtime we again switch to evaluationg the peformance also on the testset and doing 10 validation steps in the cross-validation. In Naive Bayes the scaling the scaling is generally not an issue. If we nevetheless apply the different scaling methods (only the ones that produce positiv ranges) we observe the bahaviour shown in figure left. Here 6000 features were extracted and the the logarithmic scaling is having the best performance wherein the minmax scaling is performing significantly worse than the other methods. We choose logarithmic scaling and proceed with evaluating the impact of the number of extracted features in figure middle. Here for the crossvalidation on the validation set the f1-score is monotonically increasing. Since the performance increase is not significant after usding more than 6000 features we use that number of features for this algorithm. The laplace smoothing parameters influence is plotted in figure right. The impact of this parameter on the performance is not large but we can identify an optimum at a value of 1.

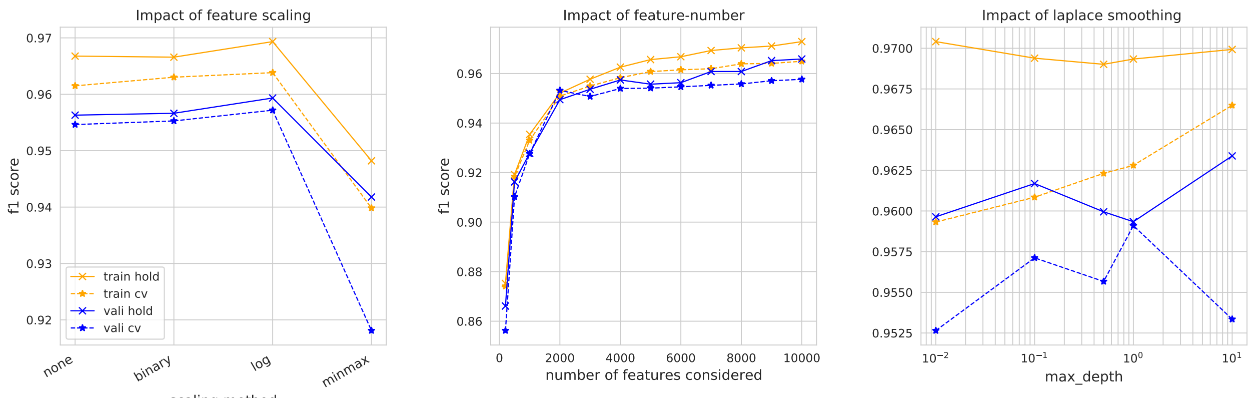

Figure 5: F1-score of Naive Bayes for different pre-processing parameters and model-parameters

The best parameters for the Naive Bayes algorithm are by that: :scaling method: logarithmic, extracted features: 6000, smoothing parameter: 1. This gives a 0.97 f1-score on for the holdout validation and 0.96 for the corssvalidation executed on the testset.

Performance-Analysis

Bridges Dataset(link to dataset)

Dataset Description

Pre-Processing

Parameter-Tuning

Performance-Analysis

Conclusion

References

[1] I. W. Tsang, J. T. Kwok, P.-M. Cheung, and N. Cristianini, “Core vector machines: Fast svm training on very large data sets.,” *Journal of Machine Learning Research*, vol. 6, no. 4, 2005.

[2] S. Shalev-Shwartz and S. Ben-David, *Understanding machine learning: From theory to algorithms.* Cambridge university press, 2014.