Exercise 1 Classification

e12045110 Maria de Ronde
e12040873 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. For our analysis we make use of the Python machine learning package Scikit-learn.

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

[1] 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 $\mathcal{O}(d|S|$ 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

[1] 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 [2] $\mathcal{O}(k|S|\log(|S|)d)$. The evaluation complexity is then only dependent on the maximum depth and the number of the trees [2] $\mathcal{O}(k$ 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)

[1] 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:

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

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 [1].

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

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

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. This method ensures that our results are comparable to give an overall view on the performance of all algorithms at the end of this report.

Kaggle: Amazon review

Dataset Description

The Amazon review dataset is used to predict the author of reviews. The reviews are translated into vectors. There are 50 classes, which represent authors of different reviews. The dataset contains 750 instances with 100002 vectors, a nominal attribute representing a unique ID, 10000 numerical vectors and a nominal vector representing the class.
In the figure 1 one can see how many instances belong to each class. We can see that the dataset is unbalanced. As some classes have 20 instances where other classes have only 10 instances.

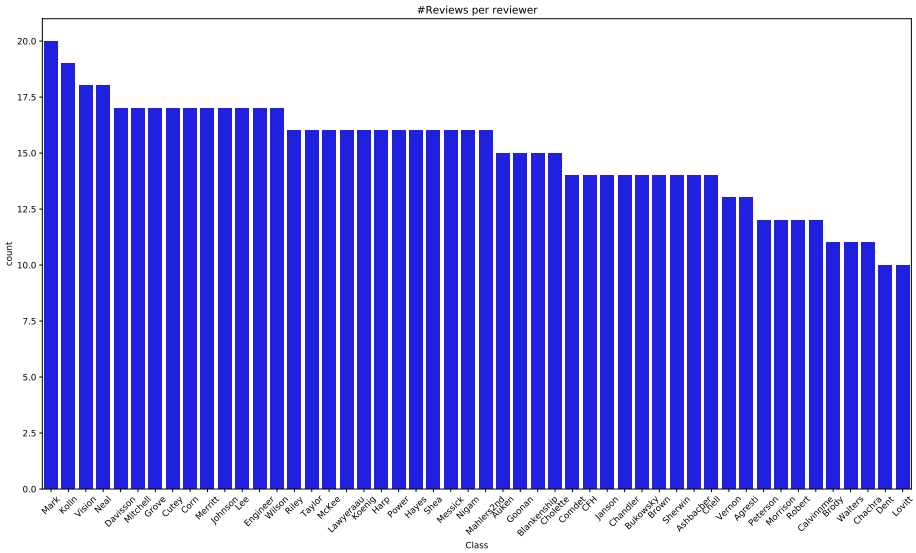

Figure 1: Instances per class

Pre-Processing

There are no missing values in the dataset. The unique ID has been deleted from the data set as has no relevance to the class. The text vectors have values differentiating of a mean of 250 occurrences per instance to 1 occurrence in the entire dataset. With 250 occurrences per instance it appears as though stop words have not been deleted from the dataset, however as we do not have the raw data available we cannot be certain.
We sort the vectors from highest number of total occurrences to the smallest number of total occurrences. Words which only occur in one instance can be seen as unique identifier to one author. By sorting the data we enable, to train our model leaving some of the first and last columns out and test whether this improve the model. The different scenarios taken into account can be seen in figure 3
In figure 2 one can see the distribution of the sum of the vectors. On average a word appears 309 times in all reviews, however there is a word which appears 187520 times, from 49 to 410 times in a review.
Two transformations to the dataset have been applied in order to flatten the weight of words which occur more often in one instance. The first transformation is the natural logarithm. The dataset will be transferred into $ln(X) + 1$. The one is added to each entry to ensure 0 entries stay 0. The second transformation is a binary translation wher $x_{i,j} = 1$ when $x_{i,j} > 1$. The scenarios will be ran for both transformations as well.

	Sum of vectors
count	10000.000000
mean	308.859700
std	2419.468303
min	0.000000
25%	9.000000
50%	21.000000
75%	220.000000
max	187520.000000

Figure 2: Distribution sum of vectors

	Scenario	Scenario description
0	10000	Select all 10000 attributes
1	8000	select the first 8000 attributes
2	6000	select the first 6000 attributes
3	50:10000	select the 50th till the 10000th attribute
4	50:8000	select the 50th till the 8000th attribute
5	50:6000	select the 50th till the 6000th attribute
6	100:10000	select the 100th till the 10000th attribute
7	100:8000	select the 100th till the 8000th attribute
8	100:6000	select the 100th till the 6000th attribute

Figure 3: Description of different scenarios

Parameter-Tuning

First we will split the data in two parts. One part, 90 %, for training and avlidation of the model with different parameters and the second part, 10%, to test the models afterwards. To extract the test set, the hold out strategy has been used.
To split the rest of the data in a training dataset, and a validation dataset, both, the hold out strategy and the cross validation using 10-fold have been used. For the hold out a division of 80% training and 20% validation has been The results of the model on the validation sets are used to tune the parameters on the model.
In order to set the scenarios and the parameters used we will make use of the average F1-score.
Perceptron:We determine the base parameters for the perceptron, alpha as 0.0005, eta as 1 penalty='none' and the max iterations are equal to 100. The results of the 9 scenarios for all tranformations can be found in figures 4 using both the hold out and cross validation strategy. All train sets of the transformed datasets havean F1-score of 100. For the non-transformed dataset a score of 100 was only obtained when high occurring words were deleted..
Comparing the F-1 of the test set of the different transformations binary transformation performs the worse in all scenarios. The logarithmic transformations performs better when we do not delete the words with high occurrences. After removing the top 50 words the F1-score of the X dataset is higher than the$ln(X) + 1$ dataset. This can be explained due to the fact that transforming the data into logarithmic values has the highest effect on the words which occur most often.
A clear difference can be seen between the performance of the hold out and the cross validation runs. The cross validation run has more stable result, where the hold out fluctuates more. Due to the high amount of classes, 50, and the relative low amount of instances it can happen that classes are not represented in the test set, some classes only appear 10 times in the entire dataset. The performance is sensitive to the chosen validation and train set. With using the cross validation the entire dataset is used (except the part left out for testing) and the influence of chance decreases. In order to tune the hyper parameters we will use the cross validation and look at the results of the validation set. Due to the high complexity of the data, and a relative little instances the classifiers can train the model to fit training data exactly, even with linear classifiers. Therefore the results on the training data cannot be compared.
For the parameter tuning of on the perceptron classifier scenario 3 of the non-transformed dataset is chosen as it has the best f1-score. The first parameter we tune is the maximum number of iteration maxiter. We test for values maxiter $\in \{2, 3, 4, 5, 10, 50, 100\}$. It can be seen that after 10 iterations the training set has a perfect fit, and the validation set has the highest performance. Therefore, we take $maxiter = 10$ for the next parameter tuning.
Next the model is trained for different learning rates ($eta0$), where $eta0 \in$

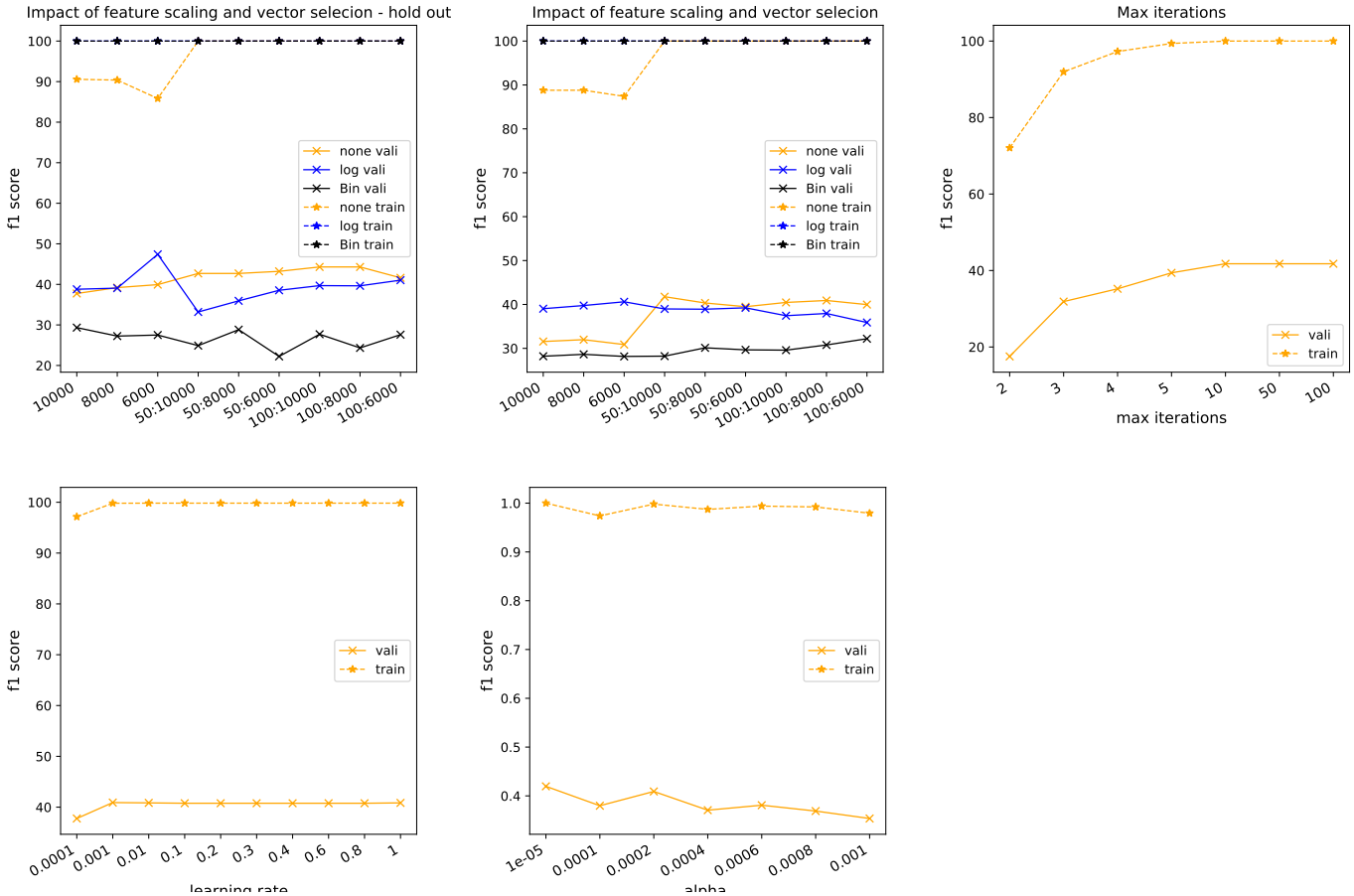

Figure 4: Parameter tuning perceprton

$\{1^{-4}, 1^{-3}, 0.01, 0.1, 0.2, 0.3, 0.4, 0.6, 0.8, 1\}$. The model is only influences slightly by the different learning rates.The highest performance was obtained wit a learning rate of 0.01.
Finally a regularization term has been introduced.We ram α was set to be $\in \{1e^{-4}, 1e^{-3}, 0.01, 0.5, 1\}$. The f1-score for both the training set as well as the validation set dropped rapidly after 1^{-3}. Therefore, a second run with $\alpha \in \{1e^{-5}, 1e^{-4}, 0.0002, 0.0004, 0.0006, 0.0008, 1e{-}3\}$ has been done. For $\alpha = 0.00001$, the validation performs best with an f1 score of 42.0%. We run our final model for the test set with the best parameters settings from the parameter tuning. The parameters are as follows maximum number of iterations is 10, $eta0 = 0.0011$, the regularization term is l1, with an α of 0.00001 against the test set. This results in an accuracy of 53.3% and a f-1 score of 49.8%.
Random forest: The second classifier is the random forest classifier. The base parameter settings are the following:$ccp\alpha = 0$, $maxleaf = 50$, $maxdepth = 10$ and $numberofestimators = 100$. The results of the different scenarios for hold out and cross validation can be found in the figures 5. The best accuracy is obtained scenario 2 on the natural logarithmic dataset with an f1-score of 40.15% in the cross validation. For the first scenarios the logarithmic transformation performs best. Starting at scenario 4 the performances come closer together and there is no transformation which out performs the other.
The following hyper parameters have been tested: number of estimators, the maximum depth, maximum leafs and the complexity cost parameter, ccp alpha. Starting with the number of estimators, the number of trees, given n estimators $\in \{1, 5, 10, 50, 70, 100, 200\}$.
Both the train set and the validation set achieve an higher performance, for all scoring methods, whenever the

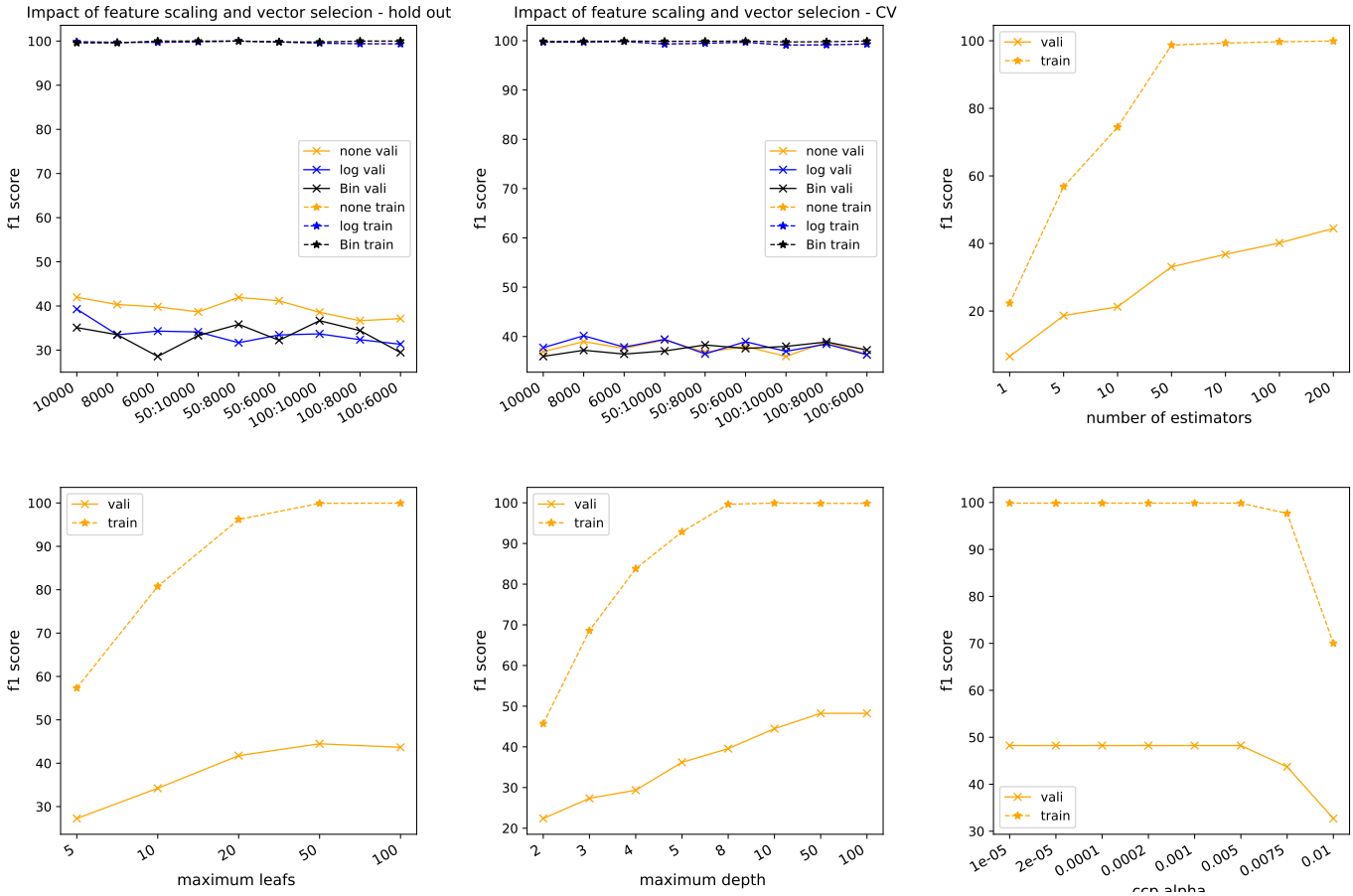

Figure 5: Parameter tuning random forest

number of estimators increases. The number of estimators is set to 200.
The second hyper parameter is the maximum number of leafs per branch, we test $maxleafs \in \{5, 10, 20, 50, 100\}$ The accuracy increases when the number of leafs increase to 50, with 100 all performances increase for the training set, but not for the test set. The model is over fitting. The maximum number of leafs is set to 50.
After the number of trees and the leafs per branch, the depth of each tree is determined. where $maxdepth \in \{2, 3, 4, 5, 8, 10, 50, 100\}$. The f1-score for the training set is highest with a maximum depth of 10. The f1-score for the validation set is highest with a maximum depth of 50 or a 100, the same scores are obtained. The maximum depth is set to 50.
Last, we tune the minimal cost complexity pruning parameter. We test for $ccp\alpha \in \{1e-5, 2e-5, 1e-4, 2e-4, 1e-3, 5e-3, 75e-4, 1e-2\}$. It can be seen that for $ccp\alpha < 0.005$ the pruning does not cost. However, for $ccp\alpha > 0.005$ both the train and the test have reduced scores. Therefor, we will set $ccp\alpha = 0.005$. The final model is now tested on the validation data with the following parameter settings: $nestimators = 200$,$max_leaf = 50$, $max_depth = 50$ and $ccp\alpha = 0.005$. This results in an f1-score of 51.8%.
Naive Bayes: The third classifier is the multinominal Naive Bayes classifier. For the base run we set α equal to 1. The nine scenarios are run for both hold out and cross validation. A clear pattern can be seen in the result. The accuracy of the non transformed dataset clearly out performs the logarithmic and the binary transformations. Furthermore it can be seen that including less vectors improves the model, both the first and the last vectors from the dataset. As the best results are obtained in scenario 8, additional scenarios are ran excluding additional vectors from the beginning and the end.

First 5 additional scenarios are created excluding more vectors in the end. The best results are obtained with scenario 11, and 12. Another 10 scenarios are considered leaving the first 150, 200, 250, 300 and 350 vectors out until the 4000th and the 4500th vector. The accuracy can be found in table 6. the best accuracy is obtained in scenario 250 closely followed by scenario 22.
The first 9 scenarios have been ran for different smoothing priors. We ran the model for alpha's $\in \{1e{-}3, 1e{-}2, 1e{-}1, 0.5, 1, 10, 100, 1000\}$. All results are equal for all alphas.

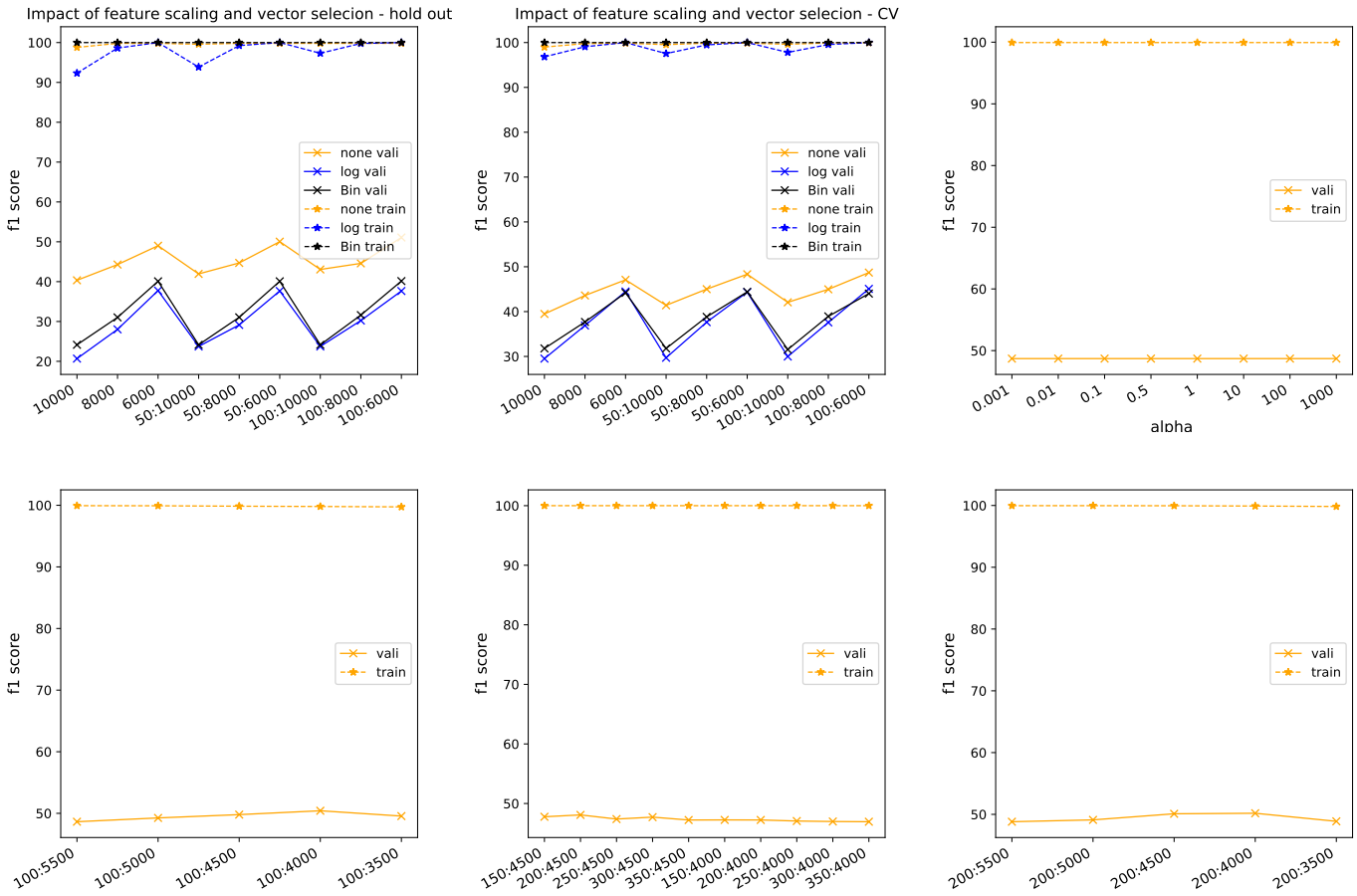

Figure 6: Parameter tuning Naive Bayes.

Classification Dataset: Kaggle Congressional Voting (link to dataset)

Dataset Description

The Congressional Voting dataset is used for classification, it is a prediction on the party (republican or democrat) for politicians of the congress. It consists of 218 samples and 17 attributes where 15 attributes are given as string values (but corresponding to booleans) and 1 further as numerical value, the id of the row, which is not relevant so we drop it during the preprocessing step. This dataset is quite small but with quite big dimension. The label is an ordinal value, "democrat" or "republican", and during the preprocessing step we transform it to respectively 0 and 1 value, and we replace "y" and "n" by 1 and 0. Hence, we have now only binary values after preprocessing. The repartition of the samples between the two parties and the repartition of samples among features are shown in fig 7.

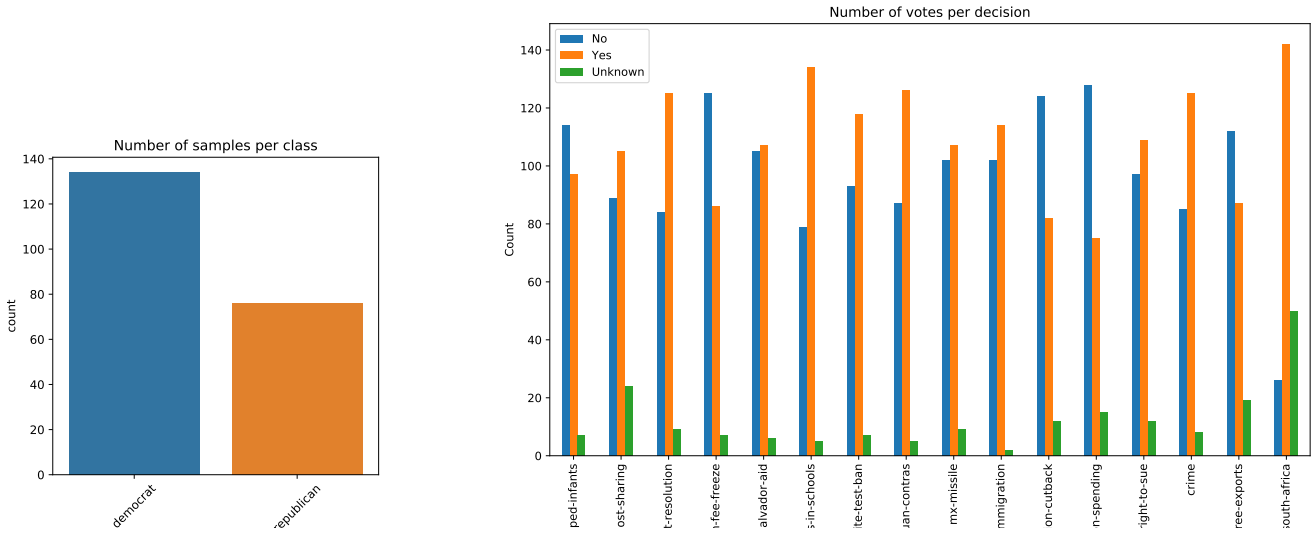

Figure 7: Description of dataset

We can see that there are some missing values, and we have the repartition of missing values among samples with the following table:

Number of samples	0	1	2	3	4	5	6	9	14	15
Missing values	118	61	21	10	1	2	2	1	1	1

Figure 8: Number of samples per number of missing values

We can notice that the feature "export-administration-act-south-africa" is the most incomplete one, with 23% missing values, the other features having less than 11% missing values.
We have to notice about this dataset that the false negatives and false positives show no theoritical difference: we do not have to privilege one over the other. So the performance by class should be barely the same. What is more, we can see that the training dataset is inbalanced, so we will use balanced indicators (such as balanced accuracy and f1 score), and make sure that our precision and recall are close.

Pre-Processing

The threshold for the minimum number of known values to conserve a row will be a tuning parameter for the algorithms. We choose to keep every column in order to avoid a loss of information. To imputate the missing values, we tried with a multivariate feature imputation or a KNN imputer, but the results were better with a random method (at least 15% of decrease in accuracy and f1 score for each algorithm with non-random methods). So we use random, however it means that we have to make means for every metrics with several preprocessed trainsets because we have to avoid an overfitting on a specific random configuration.

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 we realize two parameter tunings per algorithm, one with the grid search provided by scikit-learn and another manually one parameter after the other, and this one is made with regard to the performance on the validation set. We try to optimize the trainset f1 score on first instance in

order to introduce a too big bias because of the validation set, and we will use the testset provided by Kaggle as final testset. We select parameter values by the cross validation performance, since we assume this performance if more stable when compared to the holdout validation. The main metric we use for tuning is the macro averaged f1 score, since we have an imbalanced dataset and recall and percision should be close because we have no theoritical difference between mislabelling a democrat or a republican.

Perceptron: First, we test the different preprocessing methods with the optimal parameters found with grid search. We launch it 5 times to prevent a too large variation because of the random of the preprocessing. We have a range for thresholds of known values between 10 and 14 because we want to avoid having too less known values per sample (too large part for random values) or avoid the deletion of too much data because of the unknown values. We can see that the relevant imputation is the random imputation, and the optimal threshold is 13 (9) and we have the confirmation that random imputation is way better and now we will work only with random imputation. We take the most frequent parameters among the 5 algorithms we have for the optimal preprocessing configuration.

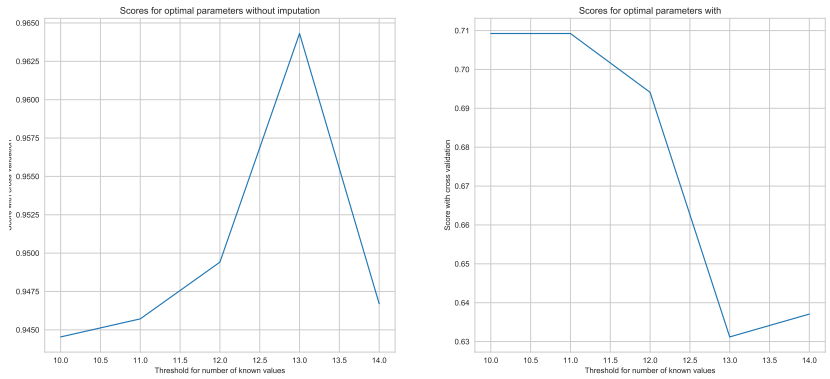

Figure 9: Optimize preprocessing parameters with grid search

With a f1 score of 0.95 with cross validation on trainset and a f1 score of 0.83 for validation set, have a big overfitting with grid search. For manually search, the preprocessing is run 10 times, because the operations are less costly so it takes a reasonable time. First we optimize the threshold for missing values (11), and the best one seems to be a threshold of 13, because with 6-8 as threshold we are too sensible to random values and with a threshold of 13 we have one of the best accuracy for trainset and no difference between accuracies of the two datasets. As we could have imagined, when we drop every sample with missing values, we have a strong overfitting, so a threshold of 15+ is not optimal. For the maximum number of iterations, we choose 18 because the f1 score starts to be stable. For the penalty parameter, it can change between runs but most of the time having no penalty is among the best choices. 0.001 seems to be the best stopping criterion and have the best f1 score for the trainset. For the learning rate it does not seem to exist a best choice, so we take 0.05, and finally we can see that no early stopping is better for both datasets. We finally obtain a cross validated f1 score of 0.91 and a f1 score on validation set of 0.94, so it does not overfit anymore and we did not lose a lot for the trainset.

Algorithm	Threshold	Max iter	Penalty	Stop criterion	Learning rate	Early stop	f1 train	f1 test
Grid search	13	10	None	10^{-3}	0.1	No	0.95	0.83
Manually search	13	18	None	10^{-3}	0.05	No	0.91	0.94

Figure 10: Abstract of parameters and performance for perceptron algorithms

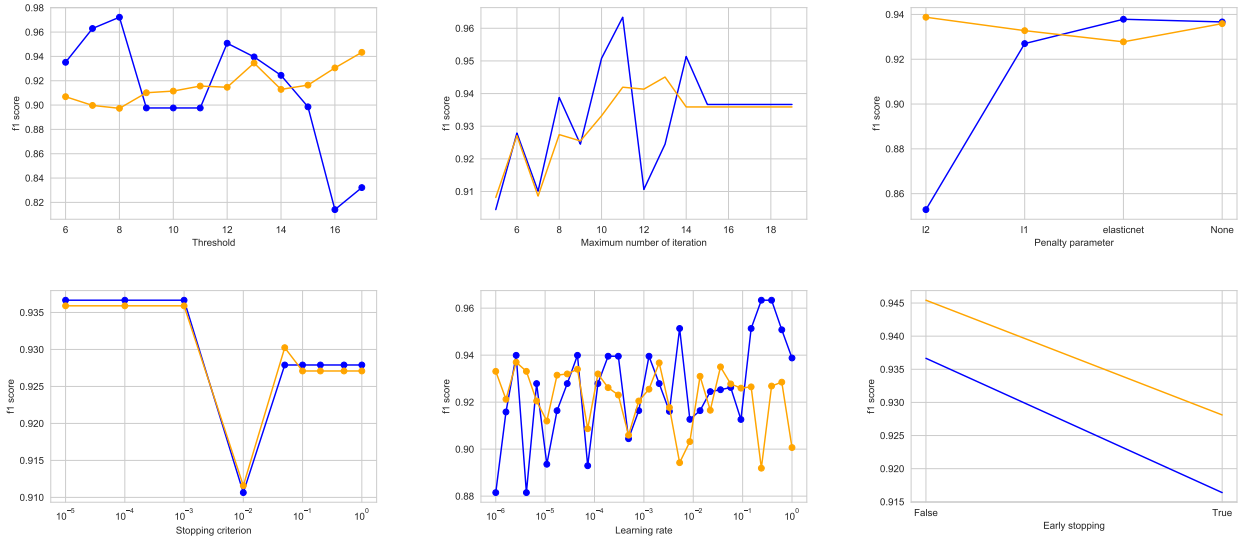

Figure 11: F1-score of Perceptron for different pre-processing parameters and model-parameters

Random forest: Again, we test the different preprocessing methods with the optimal parameters found with grid search. This time we have an optimal threshold of 13 (12).

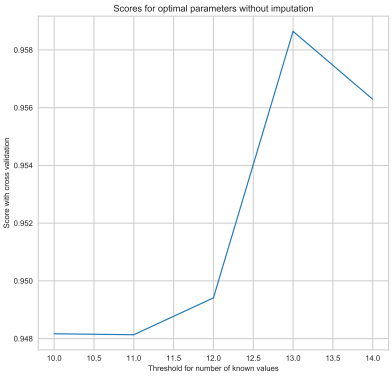

Figure 12: Optimize preprocessing parameters with grid search

Algorithm	Thres.	Criterion	Min leaf	Pruning complexity	Min split	f1 train	f1 test
Grid search	13	entropy	1	10^{-4}	2	0.92	0.94
Man. search	13	entropy	1	10^{-3}	0.04	0.92	0.97

Figure 13: Abstract of parameters and performance for random forest algorithms

With a f1 score of 0.92 with cross validation on trainset and a f1 score of 0.94 for validation set, it seems that the result is very great. Now we optimize the threshold for missing values (15), and we choose 13 because it is one of the best accuracy for trainset and we are not overfitting. What is more, we want to maintain the more data we can. We can see also here that deleting too many samples leads to overfitting. For the criterion, we can choose entropy or Giny without a real difference. Then for maximum number of samples on a leaf, the best choice is 1, to have the best accuracy on train set and no overfitting (fractions are not a number of leaf but are multiplied by the total number of leafs). Therefore, we take 0.001 for the complexity parameter used for Minimal Cost-Complexity Pruning because it is the optimum for the trainset, we have no overfitting and it is relatively stable. Finally, we optimize the mimum samples to split a node by taking 0.04 for value, because it is the optimum for the trainset and we have no overfitting. We finally obtain a cross validated f1 score of 0.92 and a f1 score on validation set of 0.97, so we can think that we were lucky with the random configurations,

or that the minimum sample for splitting made a great difference (we could not test it with the grid search because it was already too much time consuming).

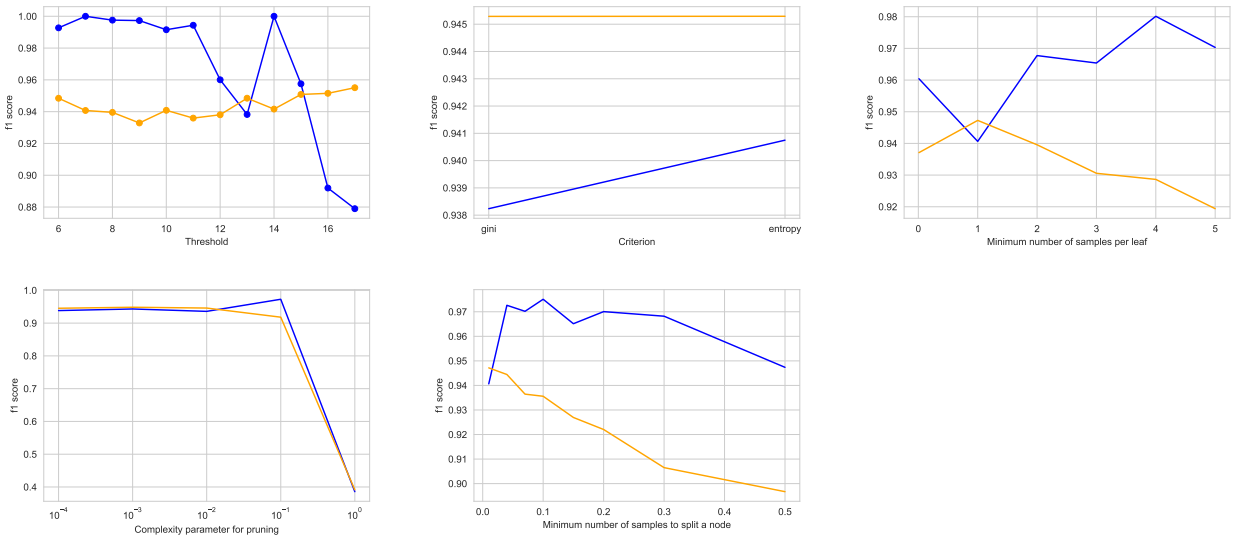

Figure 14: F1-score of Random forest for different pre-processing parameters and model-parameters

Naive Bayes: Finally, we test the different preprocessing methods with the optimal parameters found with grid search. This time we have an optimal threshold of 12 (15).

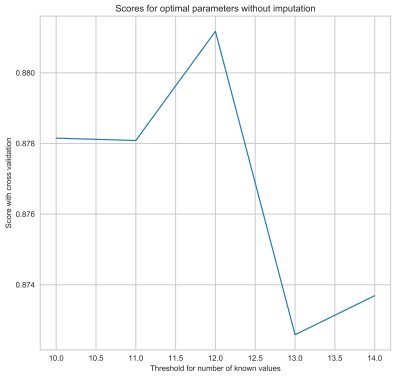

Figure 15: Optimize preprocessing parameters with grid search

Algorithm	Threshold	Laplace smoothing	f1 train	f1 test
Grid search	12	0.01	0.84	0.82
Manually search	13	10	0.87	0.88

Figure 16: Abstract of parameters and performance for naive Bayes algorithms

We can see that, even if we don't have any overfitting, the algorithm has not a very great f1 score, 0.84 with cross validation. Here the problem is mainly the percision (on validation test overall, with a percision of 0.78) and it could be problem coming from an imbalanced dataset. Now we optimize the threshold for missing values (17), and we can see that the best f1 score, that the grid search took, is exactly in the trough for the validation set. So these time we choose 15 because it is one of the best accuracy for trainset and we are not overfitting, taking the risk to delete a lot of data. Then we optimize the Laplace smoothing parameter with 10, which seems stable and has the best f1 score for both datasets. We finally obtain a cross validated f1 score of 0.87 and a f1 score on validation set of 0.88, so we can think that we were lucky with the random configurations for the trainset, but for the validation set it was better to take less samples into account.

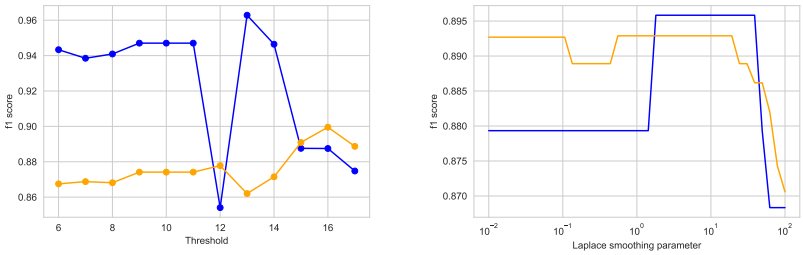

Figure 17: F1-score of Naive Bayes for different pre-processing parameters and model-parameters

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 18: 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 19 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.

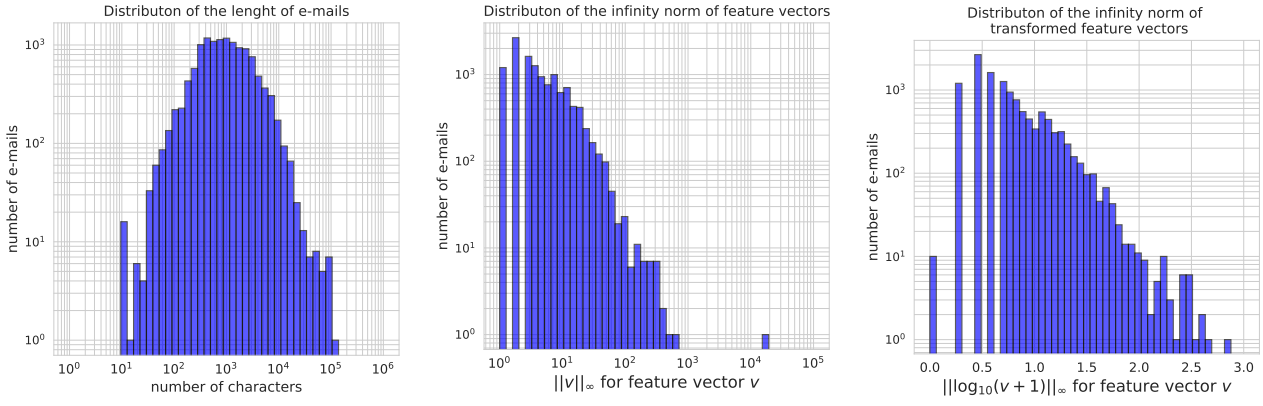

Figure 19: left: Distribution of email lenghts, middle: Distribution of maximum norm in extracted word vectors with lenght 8000, right: Distribution of maximum norm in extracted word vectors with lenght 8000 after removing outlyers and applying logarithmic transformation

Pre-Processing

In the dataset are 213 duplicate emails that we first remove and then perform 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 integers. First we create a list of all words and count their occurrences in all emails. Then we take the N most common words and count the occurrences 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 except 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 19 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 $||\cdot||_\infty$ distribution, that is bounded by the maximum magnitude. 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 19 right.

Parameter-Tuning

In figure 19 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 $||v||_\infty > 10^4$. Outliers with $||v||_\infty > 10^3$ are therefore removed in the testset. Additionally we apply the logarithmic transformation $\log_{10}(x+1)$ to all the elements of a vector and obtain a $||\cdot||_\infty$ distribution, that is bounded by the maximum magnitude. 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 19 right.
Perceptron: We compare cross-validation with 10 splits to a holdout-validation. In figure 20 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 otherwise. Binary and logarithmic scaling have the best performance in the cross-validation, where the logarithmic scaling has slightly better performance on the training set. Therefore, we choose the logarithmic scaling for the dataset. In figure 20 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 learning rate in figure 20 top right. Here the optimal learning rate is 0.1 if we consider the cross-validation performance on the validation set. Note that the holdout performance is not optimal for this value. The influence of the tolerance can be seen in 20 bottom left. For values lower than 0.001 it has a constant f1-score, that drops for the cross-validation on the validation set when we increase it. We can therefore fix the tolerance at 0.001 and investigate the maximum number of iterations in 20 bottom middle. Surprisingly a low number of iterations causes the f1-score to increase. We fix the maximum iteration for this purpose to 10.

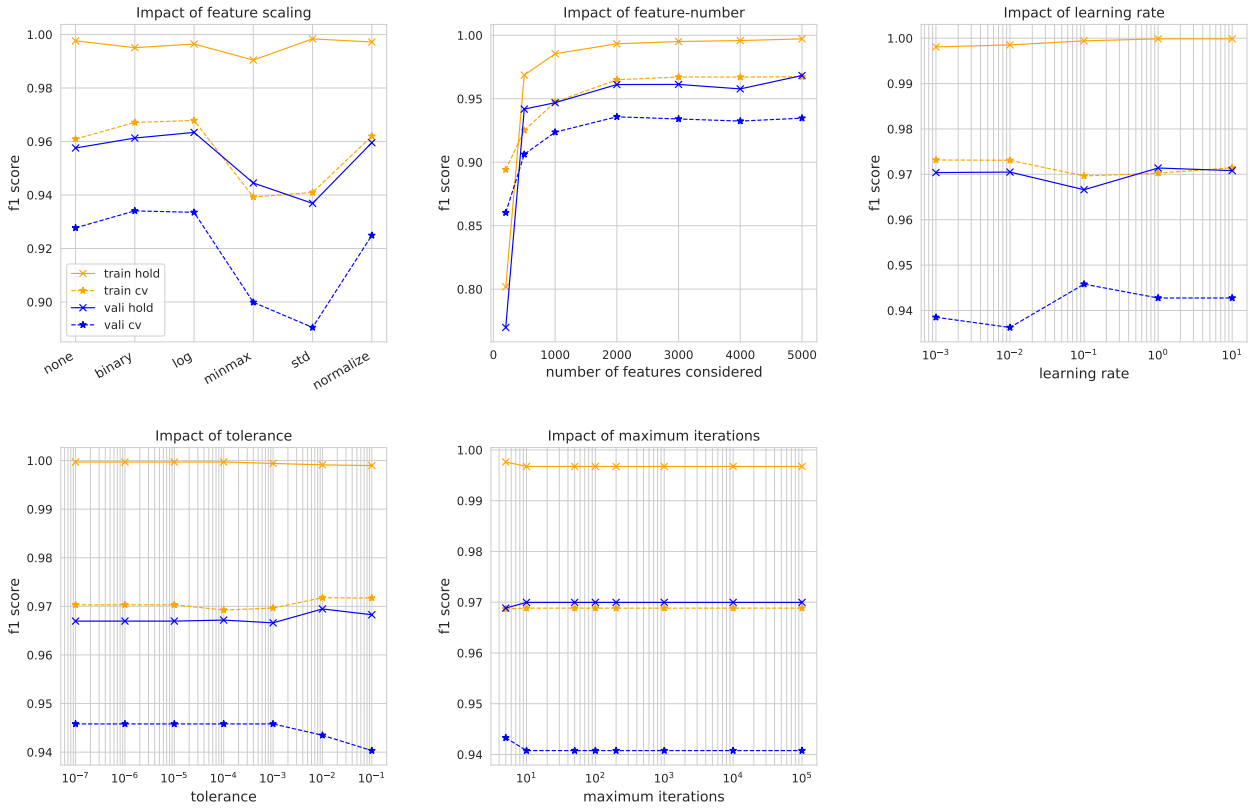

Figure 20: f1-score of Perceptron for different pre-processing parameters and model-parameters

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: 10^{-6}. The f1-score on the testset is for these parameters: 0.97 for the holdout validation and 0.95 for the cross-validation.

Random forest: Since the random forest 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.

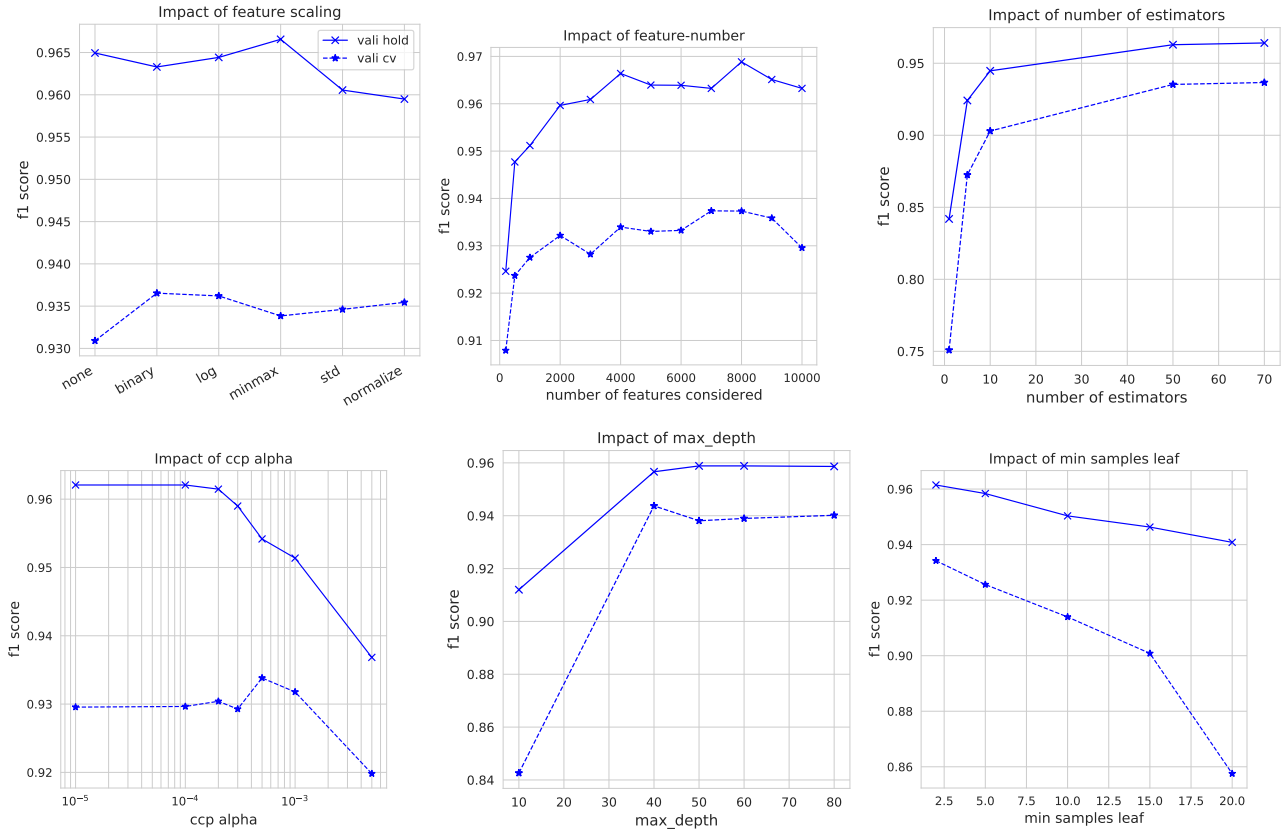

Figure 21: f1-score of random forest for different pre-processing parameters and model-parameters

The random forest algorithm is usually not affected in its performance by scaling. Nevertheless, 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 forest 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 performance 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 length and might overfit largely. In bottom left we perform cost-complexity pruning on the trees. Larger values of the parameter will post-prune more tree branches. At $5 \cdot 10^{-3}$ we reach an optimum in the cross-validation f1-score, which doesn't differ significantly from the other performances achieved 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 general 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 an optimum for a maximal tree depth of 15. The last parameter to prevent overfitting we investigate is the minimum number of samples 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 don't 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 evaluating the performance also on the testset and doing 10 validation steps in the cross-validation. In Naive Bayes the scaling is generally not an issue. If we nevertheless apply the different scaling methods (only the ones that produce positive ranges) we observe the behaviour shown in figure 22 left. Here 6000 features were extracted, and 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 22 middle. Here for the cross-validation on the validation set the f1-score is monotonically increasing. Since the performance increase is not significant after using more than 6000 features we use that number of features for this algorithm. The Laplace smoothing parameters influence is plotted in figure 22 right. The impact of this parameter on the performance is not large but we can identify an optimum at a value of 1.

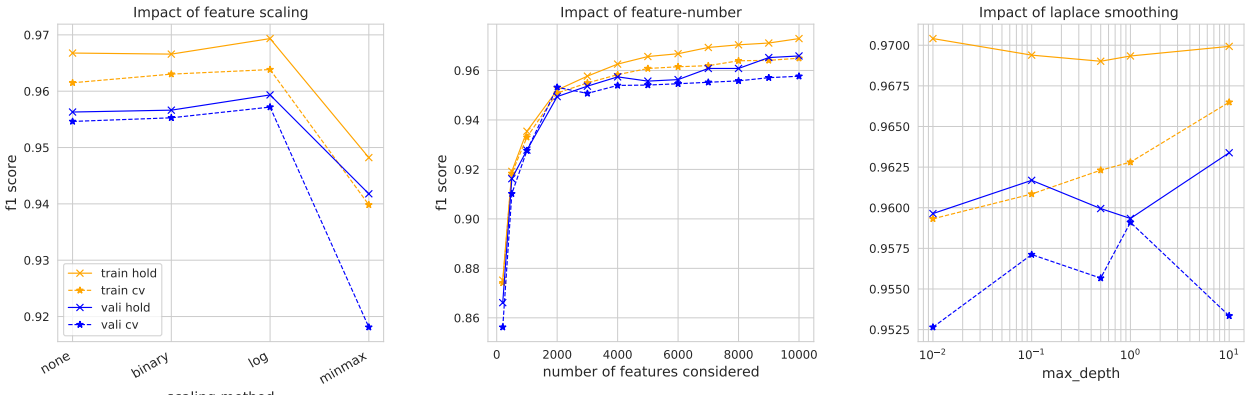

Figure 22: F1-score of Naive Bayes for different pre-processing parameters and model-parameters

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 cross-validation executed on the testset.

Bridges Dataset(link to dataset)

Dataset Description

In this dataset with a size of 108 smaples, a collection of attributes of briges in Pittsburgh Pennsylvania is presented. The task is to predict the type of bridge by the given attributes. The the attributes are summarized in table 23, where we can see that most of them are nominal and lenght and span can be identified as ordinal.

attribute	propertie	attribute	propertie
1. river	3 nominal values	7. clear-g	2 nominal values
2. location	52 nominal values	8. t-or-d	2 nominal values
3. erected	4 nominal values	9. material	3 nominal values
4. purpose	4 nominal values	10. span	ordinal, short, medium, long
5. length	ordinal, short, medium, long	11. rel-l	3 nominal values
6. lanes	4 nominal values	12. type	6 nominal values

Figure 23: attributes for the bridges dataset

In figure 24 left we see the distribution of the type of bridges that occur in our data. This trainset is imbalanced, with a bit less than the half of the samples which are labellized as simple truss... we can already imagine that our results will not be very accurate according to the small amount of samples and this very imbalanced dataset (we only have around 10 samples per label in most of cases).

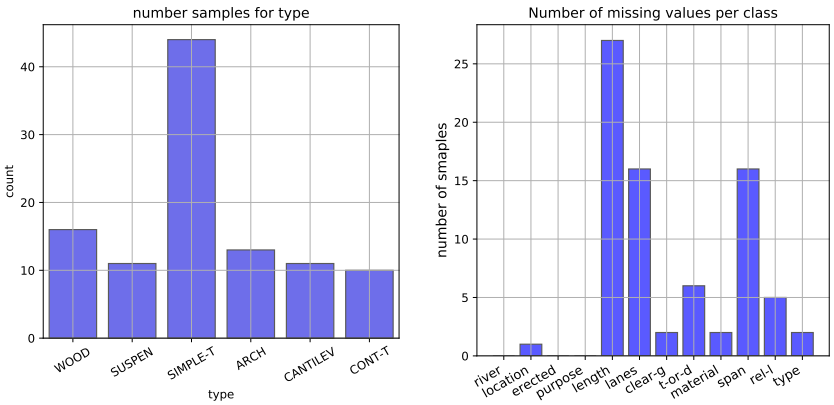

Figure 24: left: distribution of the target value over the dataset, middle: distribution of misssing values

Next in figure 24 middle we show the missing values in the dataset. We can see that there are some missing values, and we have the repartition of missing values among samples with the following table insert table. We can notice that the feature "lenght" is the most incomplete one, with 25

Pre-Processing

Parameter-Tuning

We will train our classifers using a cross validation of 3 holds. We use a 3 holds to stiill have a big enough sample for the testing. The pareameter tuning is based on the training results, as the dataset is too small to split separate for validation and testing. Before we can train the models we will impute for the missing values. HOW DO WE DO THIS!!!!! **Perceptron** :In order to find the best paramameter settings for the perceptron classifier we made use of the grid search. The following parameter values are considered in the grid search: eta0 $\in \{0.1, 0.2, 0.3, 0.4\}$, early stopping which is true or false, the maximum number of iterations with maxiter $\in \{10, 20, 30, 100\}$.The tol (IS THIS ALPHA???) $\in \{1e-4, 1e-3, 1e-2, 1e-1\}$ and the penalty possibilities l2, l1l elasticnet, or no penalty. We ran the grid search 5 times. In **??** can be seen that the results are unstable. In five runs we got up to three different paramters. Therefore, a manual search per parameter has been done as well. **Random forest** **Naive Bayes**

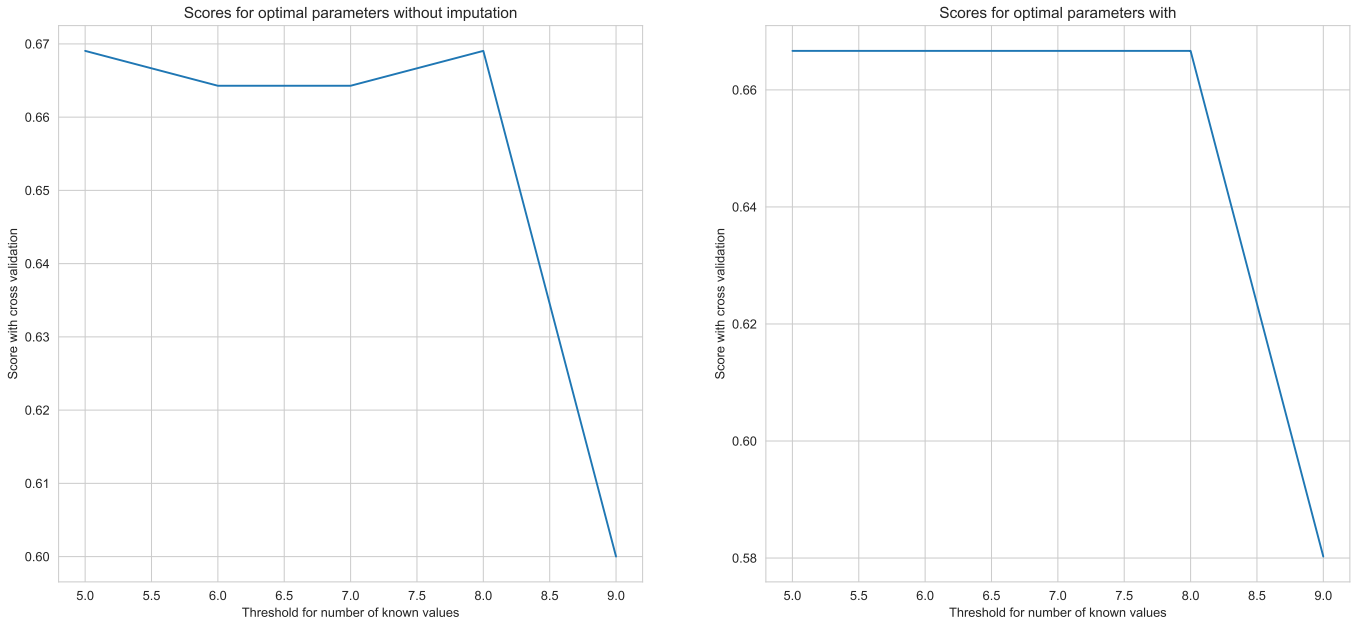

Figure 25: a

Conclusion

In this project we investigated the performance of the algorithms: Perceptron, Random Forest and Naive Bayes on four different Datasets. To give insight into the overall performance of the algorithms, we calculate the f1-score for all in table **??**. We see that the Perceptron had the best performance on the Bridges dataset and on the Voting dataset where it shares its f1-score with the Random Forest algorithm. The Random Forest Algorithm is on one dataset the best. Naive Bayes performs best on the Amazon and Email datasets. For the mean performance over all datasets, we get the same ranking such that the best algorithm over the datasets is Perceptron followed by Naive Bayes and Random forest on the third place. In the introduction chapter we already stated that Random Forrest has the highest algorithmic complexity making it, not a good choice for large datasets. This was also observed in our analysis in therms of practical running time. Perceptron and Bayes have a linear complexity in the trainingset size and even constant evaluation time for bounded dimension of the samples. Therefore, they are a good choice and allow us to build good classifiers aswell as shown in this project.
Additionally the best parameters we obtained in our analysis are summarized in the figure 32 for the perceptron, in 33 for Random forest and in 34 for Naive Bayes. Note that the values "def" mean that we adapted the default values given in sklearn.

Dataset	Perceptron	Random Forrest	Naive Bayes
Amazon	0.50	0.52	0.59
Email	0.95	0.93	0.96
Voting	0.94	0.94	0.88
Bridges	0.57	0.45	0.5
mean	0.74	0.71	0.73

Figure 32: Best f1-scores obtained on the testsets

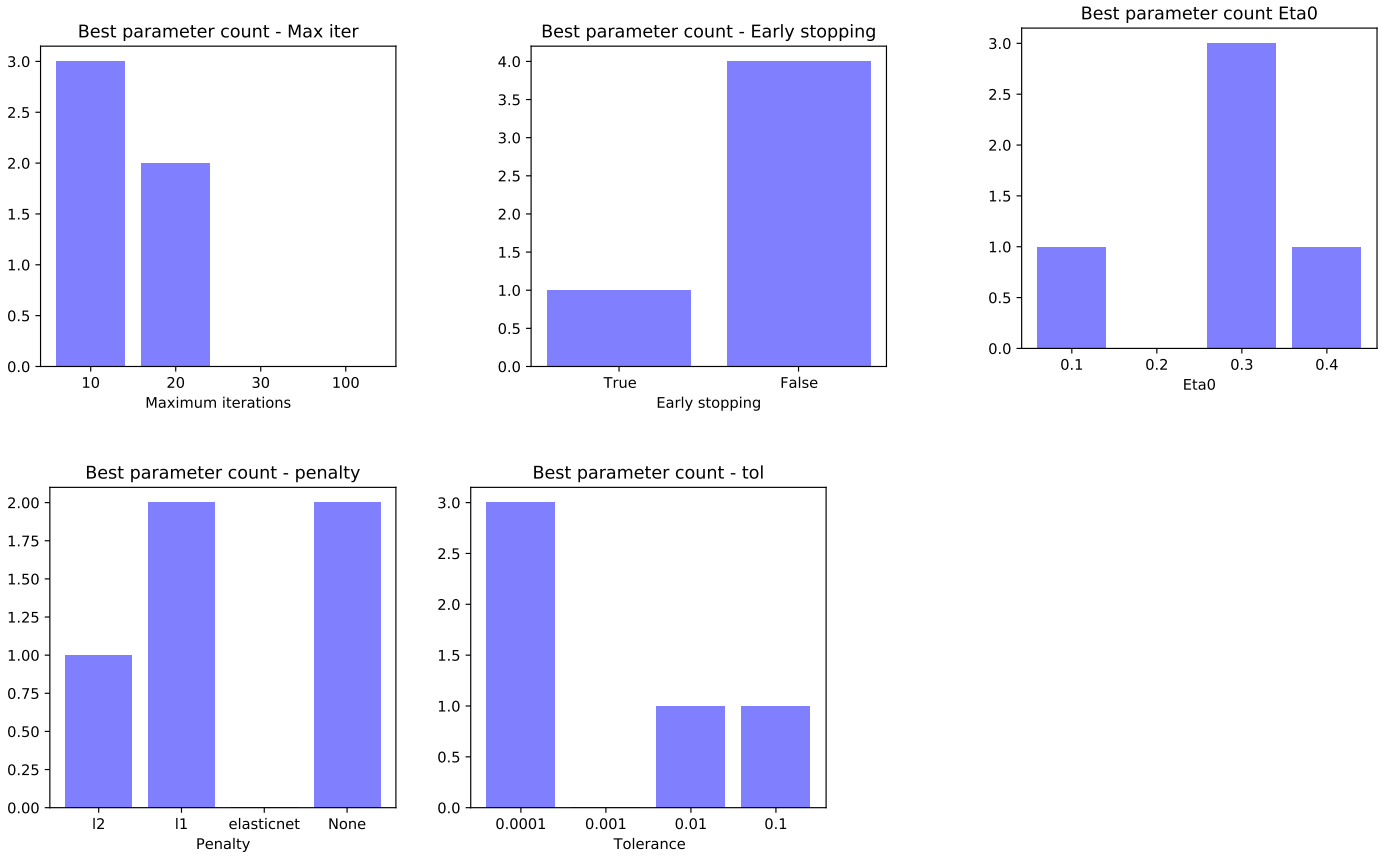

Figure 26: a

References

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

[2] 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.

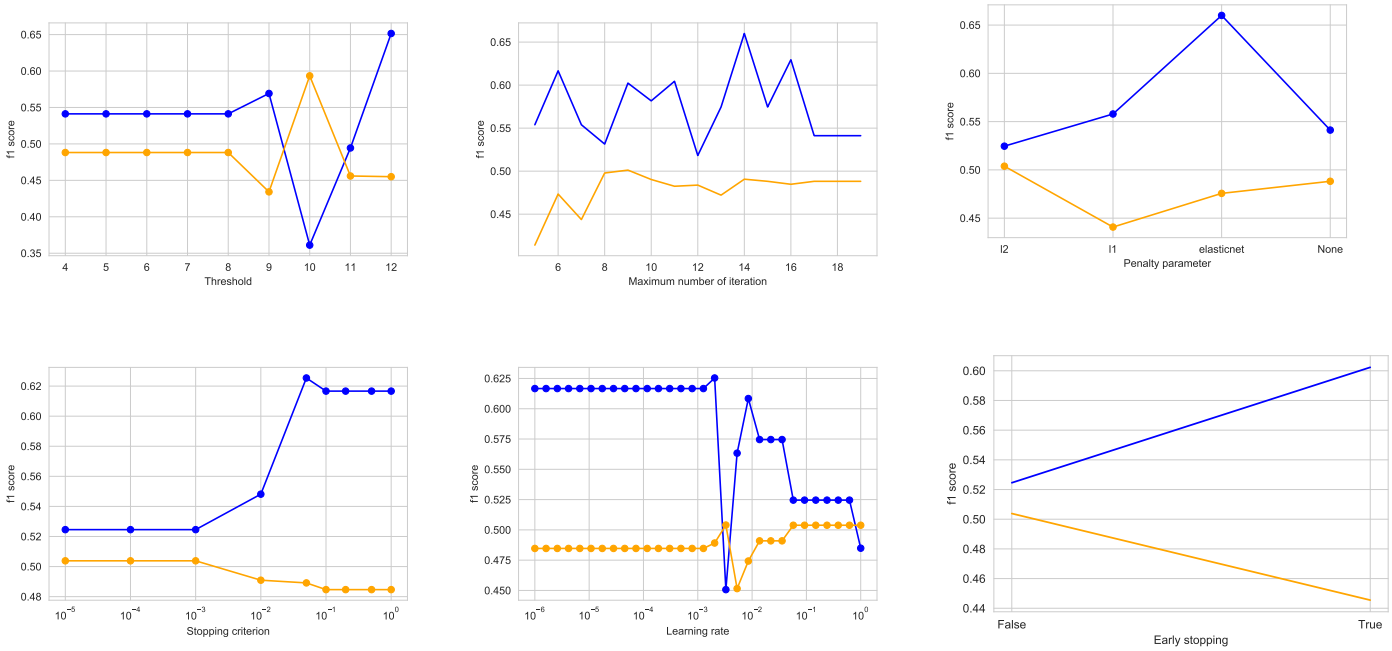

Figure 27: a

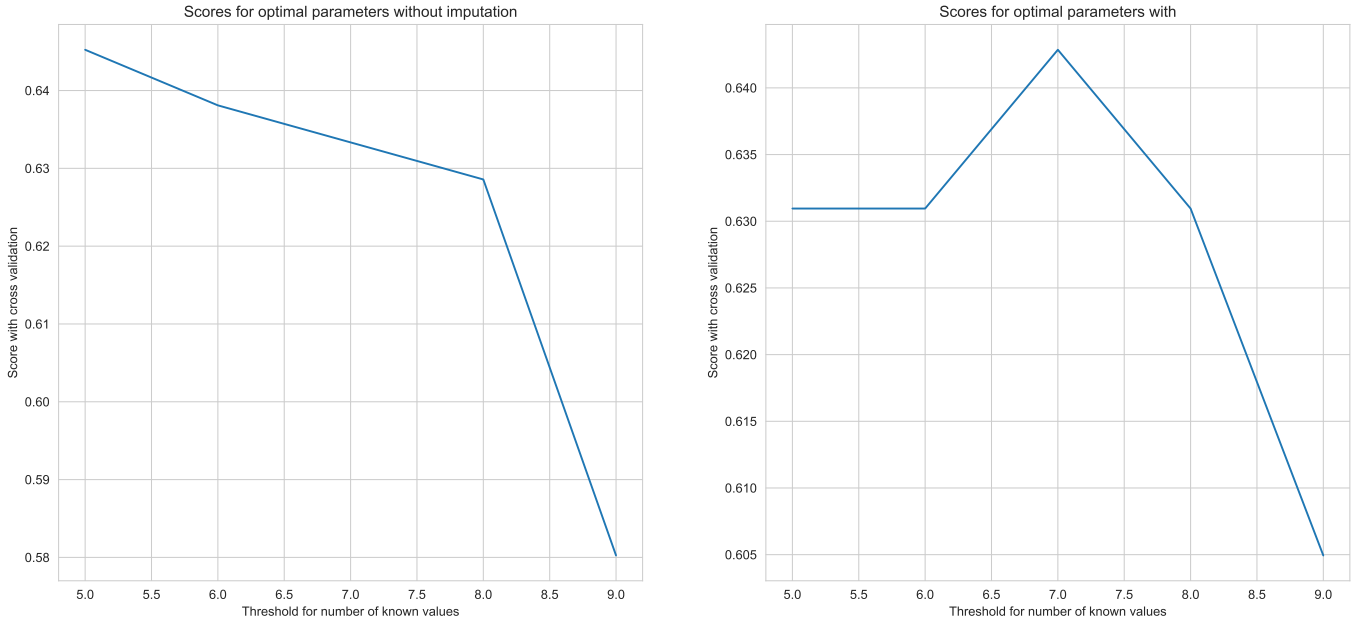

Figure 28: a

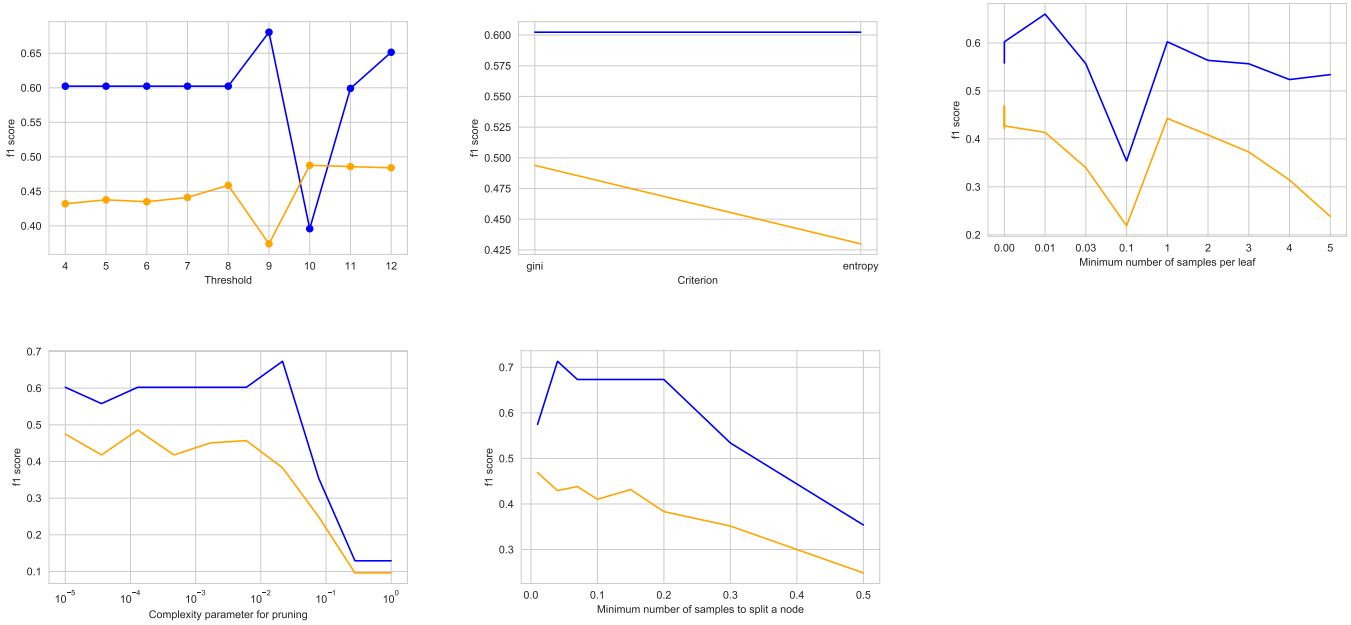

Figure 29: a

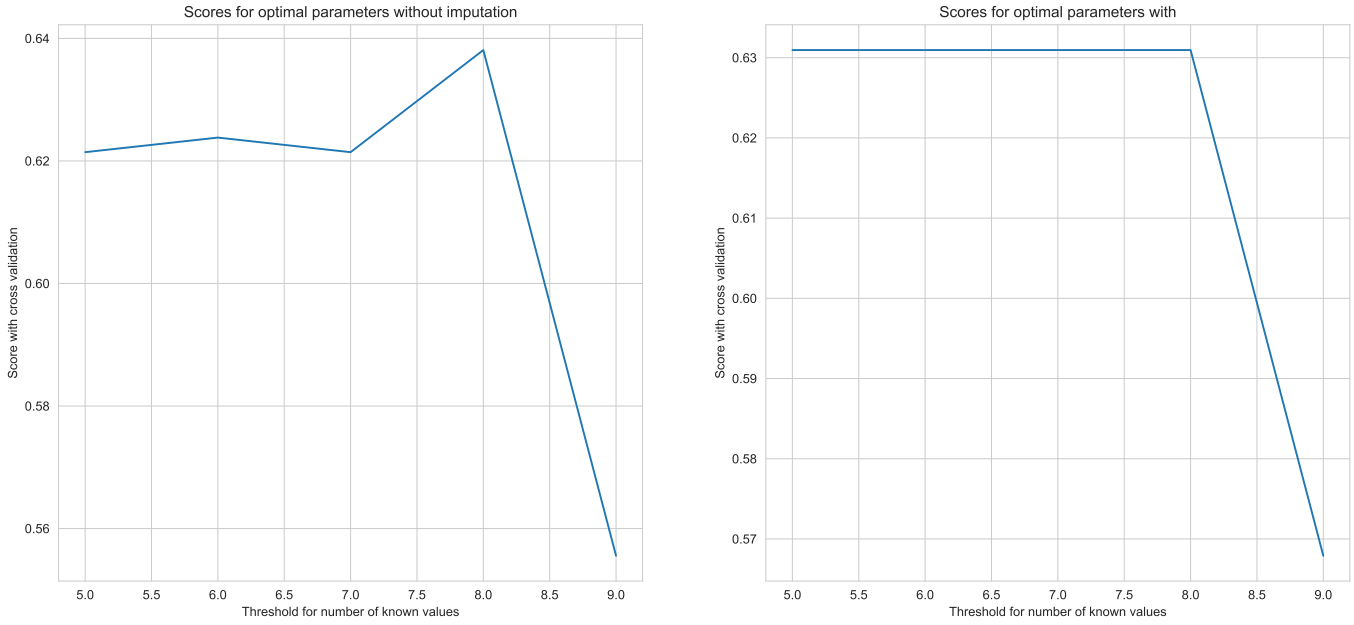

Figure 30: a

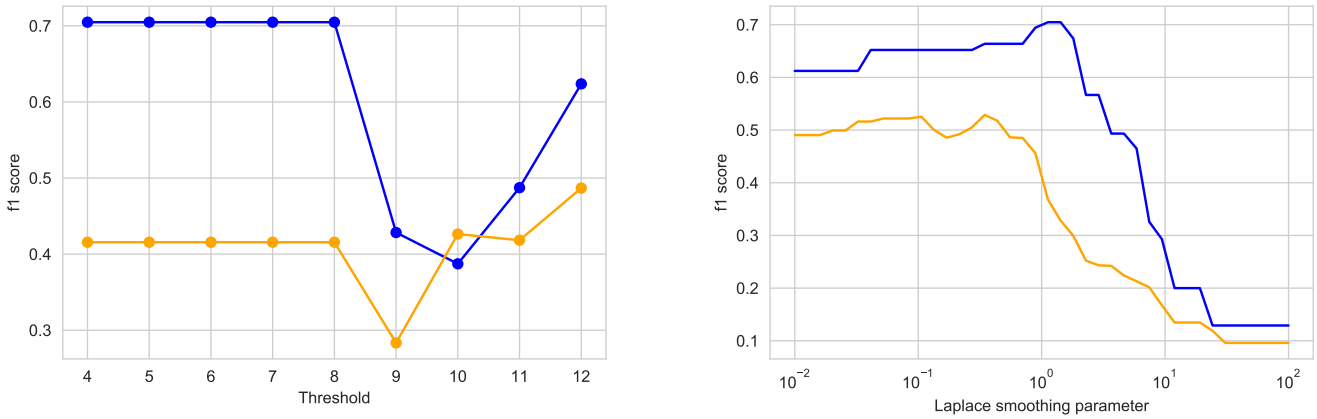

Figure 31: a

Dataset	Threshold	Max iter	Penalty	Stop criterion	Learning rate	Early stop
Amazon	def	10	l1	def	10^{-5}	l1
Email	def	10	None	def	0.001	No
Voting	13	18	None	0.001	0.05	No
Bridges	10	9	l2	0.001	0.005	

Figure 33: Best parameter configurations for Perceptron classifier with respect to the considered datasets

Dataset	Threshold	Nr. estimators	Max depth	Max leafs	Min samples p leaf	ccp alpha	criterion
Amazon	def	200	50	50	def	0.005	entropy
Email	def	70	60	def	def	0.005	entropy
Voting	13	100	def	1	def	0.001	entropy
Bridges	def	100	def	def	1	0.005	gini

Figure 34: Best parameter configurations for Random Forest classifier with respect to the considered datasets

Dataset	Threshold	alpha
Amazon	def	0.5
Email	def	1
Voting	16	10
Bridges	12	0.6

Figure 35: Best parameter configurations for Naive Bayes classifier with respect to the considered datasets