

Exercise 1: Classification

Florian Engl (12102619),
Lukas Sichert (12114770),
Kristof Dadic (12105475)

November 2024

Contents

| | | |
|---|-----------------|---|
| 1 | Datasets | 1 |
| 2 | Classifiers | 1 |
| 3 | Evaluation | 1 |
| 4 | Preprocessing | 1 |
| 5 | Hyperparameters | 3 |
| 6 | Conclusion | 9 |

1 Datasets

2 Classifiers

Bitte auf Whatsapp schauen wegen svc vs lsvc vllt auch bisl auf mathemat hintegründe eingehen

3 Evaluation

anmerken welche F1 score und wie der funktioniert haben eh alle getestet siehe whatsapp oder logfiles von test lsvc svc Man kann f1 macro auch auf nicht multiclass classification anwenden. und auch normalen f1 auf multiclass classification.

4 Preprocessing

We will do four different measures to properly preprocess the data. These are: data imputation, encoding of categorical variables, scaling of numerical values and dimensionality reduction. In this section we will look at each dataset and describe which tasks can be used on the dataset and how a specific task is performed. Then we will compare the different settings for all the models and determine their effectiveness using measures described in the previous section. In order to make this a meaningful comparison, we will use the default hyperparameters for all classifiers and we will split the data equally in the cross-validation.

Data leakage occurs when data is preprocessed before it was split into the training and test set. This way information from the test set is used to preprocess the training set. For example the MinMaxScaler uses the minimum and maximum values to scale these values properly. The values should only be computed on the training set and then be used on both training and test set. This can get rather difficult when using cross-validation. In order to avoid this, we will use the pipelines. Pipelines are a feature of scikit-learn that allow for the chaining of multiple transformers and estimators and also avoid data leakage. We will define a different pipeline for each preprocessing task and variant and chain these pipelines to compare the different settings.

Congressional Voting: As this dataset contains missing values we will need a strategy to impute these values. It is important to note that the missing values are all labeled as *'unknown'*. As all the values are categorical there are two possible strategies for data imputation. We can either use the most frequent value of

the respective attribute in the training dataset or we can treat the missing values as a separate category. We will use one-hot-encoding for the categorical values, except for the binary ones. Binary values will be encoded as 0 and 1. By using one-hot-encoding the two different treatments of the missing values will yield a different encoding. Still all the values remain binary, so no further scaling is needed. As the dataset is rather small, we will not use dimensionality reduction. This means that we will only compare the two different strategies for data imputation. In table 1 the results are visualized for the different classifiers. Here the right number in each cell is the average accuracy and the left number is the average F1 score.

| Data Imputation | Random Forest | LinearSVC | Ridge |
|-----------------|---------------|-------------|-------------|
| most frequent | 0.957/0.949 | 0.962/0.954 | 0.963/0.957 |
| nan is category | 0.955/0.947 | 0.955/0.947 | 0.960/0.953 |

Table 1: Comparison of data imputation for different classifiers for *Congressional Voting* dataset

For every classifier the most frequent value imputation strategy yields better results for both accuracy and F1 score. In this test we used 5 folds and 10 repetitions for the cross-validation.

Amazon Commerce Reviews: This dataset contains no missing values, therefore we don't need to impute any values. The target attribute has 50 different values, which will be encoded as $0, 1, 2, \dots, 49$. All the other values are numerical, therefore we will try different scaling strategies. We will try the most common scalers in the scikit-learn library, namely

- the MinMaxScaler which scales each feature to a given range, which is by default 0 to 1,
- the MaxAbsScaler which divides each feature by the maximum absolute values of the respective attribute,
- the StandardScaler which divides each feature by the standard deviation of the attribute,
- the RobustScaler which subtracts the median from each value and divides the result by the interquartile range of the respective attribute,
- the PowerTransformer which applies a power transformation to make the data more Gaussian-like,
- the QuantileTransformer which transforms the data to follow a uniform distribution.

The Transformers and the RobustScaler are known to be robust to outliers. We have also tested a method for dimensionality reduction called principal component analysis (PCA). This method is used to reduce the number of features by projecting the data onto a lower dimensional space. The `n_components=0.95` parameter determines, that our data should be projected onto the number of dimensions that explain 95% of the variance. Again the results are visualized in table ??.

In the above Experiment we have used 5 folds and 10 repetitions for the cross-validation. The best results are achieved by the PowerTransformer for all classifiers. However, only the ridge classifier benefited from the dimension reduction.

Census Income: This dataset has missing values, but only for categorical values. Those values are labeled with '?'. We will try the same two strategies as for the *Congressional Voting* dataset. The only exception is the 'education' column. This column is already encoded as the 'education-num' column and will therefore be dropped. For the numeric columns we tested the same numerical scalers, which we have tried for the *Amazon Commerce Reviews* dataset. Since the dataset only has 14 features, we will not use dimensionality reduction. Like for the previous datasets we will present the result in a table, where the first score in each cell is accuracy and the second number is the F1 score.

We used 5 folds and 5 repetitions in the cross-validation for this experiment, except for the random forest classifier. The random forest classifier was too slow, and we had to reduce the number of repetitions to 1. For this dataset the best preprocessing strategy was different for every classifier. The respective cells are marked in green in the table above. After these experiments we decided to use the StandardScaler combined with setting missing values as the most frequent category for the random forest classifier, the StandardScaler but with setting missing values as an extra category for the linear support vector classifier and the QuantileTransformer combined with setting missing values as an extra category for the ridge classifier.

Diabetes: This dataset does not have any missing values except for the target attribute. Therefore, we don't need to consider data imputation. As the dataset is very low dimensional, we will not use any dimensionality reduction. So we will only compare different scaling methods. We have used the already introduced scaling methods and got the result shown in table 3. In the end we decided to use the scalers marked in green for the

| Scaler | Data Imputation | Random Forest | LinearSVC | Ridge |
|---------------------|-----------------|---------------|-------------|-------------|
| StandardScaler | most frequent | 0.857/0.677 | 0.852/0.655 | 0.841/0.607 |
| MaxAbsScaler | most frequent | 0.857/0.676 | 0.851/0.654 | 0.841/0.607 |
| None | most frequent | 0.857/0.676 | 0.800/0.379 | 0.830/0.567 |
| MinMaxScaler | most frequent | 0.857/0.676 | 0.852/0.654 | 0.841/0.607 |
| RobustScaler | most frequent | 0.857/0.676 | 0.851/0.649 | 0.830/0.561 |
| QuantileTransformer | most frequent | 0.857/0.676 | 0.845/0.645 | 0.842/0.621 |
| PowerTransformer | most frequent | 0.852/0.665 | 0.845/0.645 | 0.842/0.621 |
| MaxAbsScaler | '?' is category | 0.857/0.676 | 0.853/0.659 | 0.841/0.610 |
| RobustScaler | '?' is category | 0.857/0.676 | 0.852/0.658 | 0.841/0.610 |
| None | '?' is category | 0.857/0.676 | 0.800/0.379 | 0.830/0.568 |
| StandardScaler | '?' is category | 0.857/0.676 | 0.853/0.659 | 0.841/0.610 |
| MinMaxScaler | '?' is category | 0.857/0.676 | 0.853/0.658 | 0.841/0.610 |
| QuantileTransformer | '?' is category | 0.857/0.676 | 0.847/0.649 | 0.844/0.627 |
| PowerTransformer | '?' is category | 0.852/0.666 | 0.846/0.645 | 0.843/0.623 |

Table 2: Comparison of different numeric and categorical transformers for the *Census Income* dataset

| Scaler | Random Forest | LinearSVC | Ridge |
|---------------------|---------------|-------------|-------------|
| MinMaxScaler | 0.767/0.592 | 0.770/0.553 | 0.768/0.542 |
| MaxAbsScaler | 0.767/0.590 | 0.771/0.553 | 0.769/0.543 |
| RobustScaler | 0.766/0.589 | 0.770/0.560 | 0.770/0.555 |
| None | 0.766/0.588 | 0.771/0.559 | 0.770/0.555 |
| StandardScaler | 0.766/0.589 | 0.770/0.560 | 0.770/0.555 |
| PowerTransformer | 0.766/0.592 | 0.763/0.564 | 0.761/0.554 |
| QuantileTransformer | 0.766/0.592 | 0.763/0.564 | 0.761/0.554 |

Table 3: Comparison of different scalers for the *Diabetes* dataset

respective classifiers.

5 Hyperparameters

Now we want to take a closer look at the hyperparameters of our models. In a first step we want to optimize the hyperparameters for the different models and datasets. In a second step we will analyze how sensitive the models are to changes in the hyperparameters. Unfortunately we will not be able to find the optimum for all hyperparameters, as scikit-learn offers a wide range, so we are going to focus on the most important ones.

For tuning we want to use Bayesian Optimization from the scikit-optimize package. Bayesian Optimization is a probabilistic approach to finding the minimum or maximum. This means that it builds a probabilistic model of the function and uses this model to decide on where to evaluate in the next step. In our case the function is given by the performance measure.

Bayesian Optimization does not only take a function as input but a search space as well, which is why we need to decide on an appropriate range of values for the parameters as well. Moreover, because we chose two different performance measures for each dataset, we will have to do the optimization twice and then compare results to get the overall best.

Random Forest Classifier: The random forest classifier has a lot of hyperparameters and takes a comparatively long time to train and evaluate. That being the case we decided to focus the following five parameters to keep tuning times at a reasonable level. These are:

- `n_estimators`: This determines the number of trees in the forest.
- `max_depth`: This gives a maximum on the depth of the trees. If we set this to `None`, the trees will expand fully until each leaf has less than `min_samples_split` instances.
- `min_samples_split`: Is the minimum number of instances necessary for an inner node to split.
- `min_samples_leaf`: Is the minimum number of instances necessary for a leaf node. This means a split will

only happen if both leaf nodes have at least `min_samples_leaf` instances afterwards.

- `max_features`: This gives the number of features that are being randomly selected from all features at each node to decide on the best split. Possible values are integers or floats and 'sqrt', 'log2', 'None' or 'auto'.

For the big datasets 'Amazon Commerce Reviews' and 'Diabetes' we will use the search space:

`n_estimators` $\in \{100, \dots, 2000\}$, `max_depth` $\in \{\text{None}, 10 \dots, 100\}$,
`min_samples_split` $\in \{2, \dots, 100\}$, `min_samples_leaf` $\in \{1, \dots, 5\}$,
`max_features` $\in \{\text{'sqrt'}, \text{'log2'}\}$.

Here we didn't consider 'None' as a value for `max_features` because it resulted in a runtime that was way too long for the scope of this project. Moreover it is empirically known, that the other results usually give better values for classification. Nonetheless on the other two datasets we could test `max_features` = 'None' and could set the number of trees up to 3000, as these are significantly smaller datasets. For the other parameters we used the same search space as for the big datasets. In table 4 and 5 we summarized the results of our tuning process by giving the best values for the respective performance measures.

As already discussed we used the F1 score as a second metric for all datasets excepts the 'diabetes' dataset for which we used recall instead. It is also important to note that on the multiclass dataset 'Amazon Commerce Reviews' we used the macro averaged F1 score and on the others the standard weighted one.

| Dataset | estimators | depth | split | leafs | features | accuracy |
|----------------------|------------|-------|-------|-------|----------|----------|
| congressional voting | 1963 | 40 | 20 | 4 | None | 0.963 |
| Amazon Reviews | 2000 | 40 | 2 | 1 | sqrt | 0.728 |
| Census Income | 100 | 30 | 20 | 1 | sqrt | 0.865 |
| diabetes | 3000 | 40 | 2 | 4 | sqrt | 0.773 |

Table 4: Tuned hyperparameters for random forest classifier and accuracy

| Dataset | estimators | depth | split | leafs | features | F1/recall |
|----------------------|------------|-------|-------|-------|----------|-----------|
| congressional voting | 1963 | 40 | 20 | 4 | None | 0.957 |
| Amazon Reviews | 2000 | 40 | 2 | 1 | sqrt | 0.728 |
| Census Income | 2000 | 40 | 20 | 1 | sqrt | 0.686 |
| diabetes | 3000 | 70 | 6 | 4 | None | 0.623 |

Table 5: Tuned hyperparameters for random forest classifier and F1 score/recall

We got the same results tuning for accuracy and for the F1 score on the first two datasets. On the last two datasets we calculated the second performance measure with the best parameters for accuracy and got a F1 score of 0.685 for the Census Income dataset and a recall of 0.647 for the diabetes dataset, so we will keep the parameter values for the best accuracy.

Search space was different for different datasets.

Linear Support Vector Classifier: For support vector machines the arguably most important hyperparameter, apart from the tpe of kernel used, is the regularization parameter `C`. The Kernel in our case is always linear which leaves `C`. In addition we considered three other parameters, we thought to be interesting. The considered hyperparameters are:

- `C`: The so called regularization parameter. Its inverse is the strength of the regularization, which is why it has to be greater than 0. A strong regularization means a big loss if the decision boundary small.
- `fit_intercept`: It can be set to 'True' or 'False'. If set to 'True' the model will calculate the intercept for the decision boundary.
- `class_weight`: This parameter decides how much the model should take into account classifying differently sized classes. In unbalanced datasets for example it can be useful to give a bigger weight to correctly classifying the minority class in order to prevent the model overly predicting the majority class. We only considered the values 'balanced' and 'None' but you could assign specific weights for each class. When set to 'balanced' the weights are inversly proportional to class sizes.
- `dual`: Takes the values 'True' or 'False' and decides whether the primal or the dual problem is solved.

Generally the dual Problem is faster especially for high dimensional data, but as we will see sometimes you get better results with the primal problem.

As already discussed, the influence of the `class_weight` will be especially interesting for our imbalanced datasets. For the linear support vector classifier we used the following search space on all four datasets:

$$C \in [10^{-6}, 10^6], \text{fit_intercept} \in \{\text{True}, \text{False}\}, \\ \text{class_weight} \in \{\text{'balanced'}, \text{'None'}\}, \text{dual} \in \{\text{True}, \text{False}\}.$$

Again we summarized the results of our optimization process in table 6 and 7.

| Dataset | C | fit_intercept | class_weight | dual | accuracy |
|----------------------|---------|---------------|--------------|-------|----------|
| congressional voting | 1.150 | True | balanced | False | 0.963 |
| Amazon Reviews | 0.00946 | True | balanced | False | 0.748 |
| Census Income | 679.520 | True | None | False | 0.853 |
| diabetes | 2.353 | True | None | False | 0.771 |

Table 6: Tuned hyperparameters for linear support vector classifier and accuracy

| Dataset | C | fit_intercept | class_weight | dual | F1/recall |
|----------------------|--------|---------------|--------------|-------|-----------|
| congressional voting | 1.241 | True | None | False | 0.955 |
| Amazon Reviews | 0.0093 | True | balanced | False | 0.725 |
| Census Income | 0.274 | True | balanced | False | 0.677 |
| diabetes | 1e-06 | False | balanced | False | 0.838 |

Table 7: Tuned hyperparameters for linear support vector classifier and F1 score/recall

The results show, that the computation with the primal problem outperformed the dual problem in every case. However it should be noted that for the 'Amazon Commerce Reviews' dataset we got the same accuracy and F1 score for dual computation. Therefore we will choose the dual computation as it is faster.

Later we will see that small changes in C do not vary the result too much, so we choose $C=0.00946$ for 'Amazon Commerce Reviews' and $C=1.2$ for 'congressional voting'. Taking the other values from table 6 for 'congressional voting' we get an F1 score of 0.951 which is pretty good and we will use these values.

For the other two datasets finding the right values for optimizing both performance measures at the same time is harder because they differ quite a lot. The 'Census Income' dataset is imbalanced, which is why we would usually prefer the values in table 7. This gives us an accuracy of 0.805. But using the values from table 6 the F1 score is 0.659312, which is less of a difference to the optimal value than our loss in accuracy would be otherwise so we choose these values.

To the 'diabetes' dataset we will get back later in the sensitivity analysis because, as we will see, matters are more complicated for this dataset.

Ridge Classifier: As for the linear support vector classifier the arguably most influential parameter is the regularization parameter called `alpha`. For the ridge classifier the regularization punishes high weights which again helps to prevent overfitting. `alpha` for the ridge classifier correspond to the value $\frac{1}{2C}$ and therefore has to be positive.

We also considered two other parameters, which we already encountered when tuning the linear support vector classifier, namely `fit_intercept` and `class_weight`. The search space we considered is:

$$\text{alpha} \in [0, 100], \text{fit_intercept} \in \{\text{True}, \text{False}\}, \text{class_weight} \in \{\text{'balanced'}, \text{'None'}\}.$$

Once more table 8 shows the tuned parameters for the accuracy and table 9 for the F1 score/recall.

For 'congressional voting' and 'Amazon Commerce Reviews' the values in table 8 and table 7 coincide, so those are the best settings.

For the 'Census Income' dataset testing the values for the best accuracy gave us a F1 score of 0.626 and vice versa we got an accuracy of 0.795. In this case the loss in accuracy is comparable to the loss in the F1 score so we choose the values from table 9. Doing this we take into account that the dataset is imbalanced and prevent the model from exploiting the different class sizes.

| Dataset | alpha | fit_intercept | class_weight | accuracy |
|----------------------|--------|---------------|--------------|----------|
| congressional voting | 0 | True | None | 0.963 |
| Amazon Reviews | 45.32 | False | None | 0.754 |
| Census Income | 63.876 | True | None | 0.844 |
| diabetes | 0 | True | None | 0.771 |

Table 8: Tuned hyperparameters for ridge classifier and accuracy

| Dataset | alpha | fit_intercept | class_weight | F1/recall |
|----------------------|--------|---------------|--------------|-----------|
| congressional voting | 0 | True | None | 0.963 |
| Amazon Reviews | 45.32 | False | None | 0.73 |
| Census Income | 11.333 | True | balanced | 0.666 |
| diabetes | 0 | False | balanced | 0.797 |

Table 9: Tuned hyperparameters for ridge classifier and F1 score/recall

Again we will revisit the 'diabetes' dataset later in the sensitivity analysis to analyze the best parameter setting for overall performance.

For the tuning and testing processes, we always used repeated stratified 5 fold crossvalidation. On the congressional voting and the diabetes dataset we used 10 repetitions and on the other two datasets we used 5 repetitions except for the random forest classifier with which we did only one repetition. This was due to the otherwise very long runtime for the large datasets.

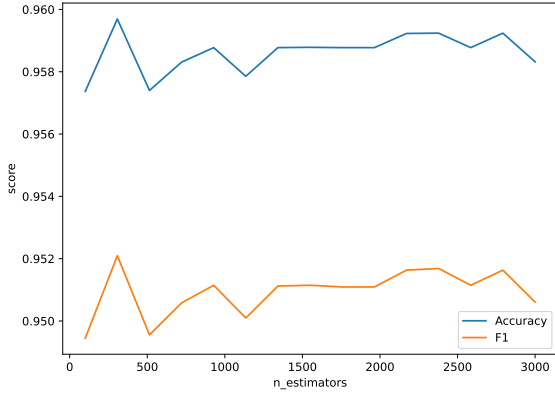
The next step in analysing the influence of hyperparameters on our classifiers is the sensitivity analysis. This means we will take a look at how changing a single parameter value influences the performance of the model. We will do this for the hyperparameters we tuned in the previous section and will always change one while keeping the others at their optimal values, which we found earlier. We are going to achieve this by plotting the effectiveness measures against the parameter values.

However for the 'diabetes' Dataset in combination with the linear support vector classifier and the ridge classifier we will take a closer look and try all combinations of `fit_intercept` and `class_weight` and vary the parameters `C` and `alpha` because we still need to decide on the overall best setting.

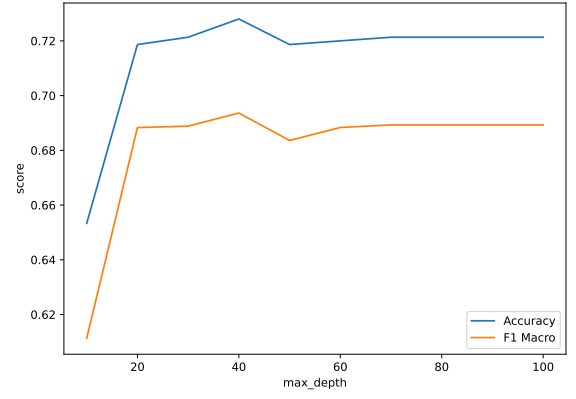
Random Forest Classifier Sensitivity: We first want to find out how much each of the 5 considered hyperparameters influences the effectiveness of the classifier. We found that the sensitivity highly depends on the dataset:

- `n_estimators`: On the datasets 'Census Income' and 'diabetes', the parameter changed almost nothing. However for 'Amazon Commerce Reviews' we got a clear Trend, that both measures increased with the number of trees first rather quickly and then more slowly. For the 'congressional voting' dataset the parameter seemed to influence the effectiveness but we no clear trend could be identified.
- `max_depth`: Varying this parameter only changed the effectiveness for the 'Amazon Commerce Reviews' and the 'Census Income' dataset. Here we could see that the accuracy increased with the depth of the trees at first and then stagnated, but the increase in accuracy for 'Census Income' was almost negligible and the increase in the F1 score was also way smaller than for 'Amazon Commerce Reviews'.
- `min_samples_split`: For the 'Census Income' and the 'diabetes' datasets varying this parameter hardly changed the effectiveness scores. On 'Amazon Commerce Reviews' both scores decreased with higher values whereas for the 'congressional voting' dataset both values decreased to a minimum at `min_samples_split=5` and then increased to the original value again.
- `min_samples_leaf`: This parameter did not really influence the performance for the 'diabetes' and 'Census Income' datasets. For the other two, small values fared better which fits in with our optimized values.
- `max_features`: 'sqrt' consistently outperformed 'log2' by quite a big margin on the 'Amazon Commerce Reviews' dataset. For the 'Census Income' it did not make much of a difference. On our smaller and less dimensional datasets None led to the best results except for the accuracy of 'diabetes' where 'sqrt' performed a little better.

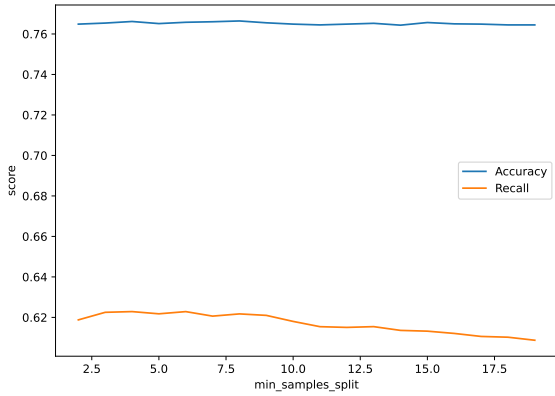
To underline our conclusions we want to give a few exemplary plots in figure 1 below.



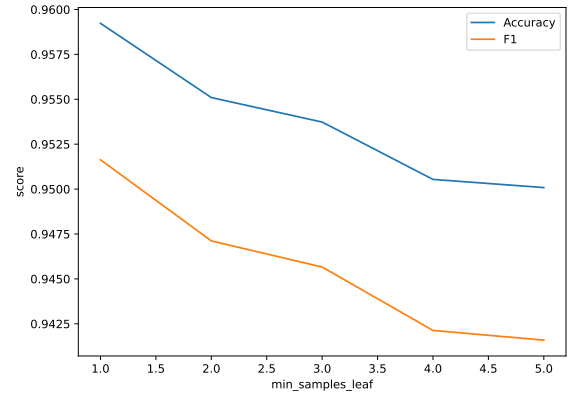
(a) Parameter `n_estimators` for dataset 'congressional voting'



(b) Parameter `max_depth` for dataset 'Amazon Commerce Reviews'



(c) Parameter `min_samples_split` for dataset 'diabetes'



(d) Parameter `min_samples_leaf` for dataset 'congressional voting'

Figure 1: Sensitivity analysis of random forest classifier

Linear Support Vector Classifier Sensitivity:

The parameter `C` had significant influence on the performance for every dataset except the 'Census Income' one, where only the F1 score increased at first and then also stagnated.

Setting `class_weight` to 'balanced' improved the F1 score/recall for every dataset as expected but also improved the accuracy for the 'Amazon Commerce Reviews' and the 'congressional voting' dataset. On the other datasets it decreased the accuracy, so there is no overall conclusion as to what setting to use as it depends not only on the dataset but also on the preferred performance measure.

For the parameter `fit_intercept`, we found that setting the intercept to 'True' yielded the best results for accuracy and F1 score across all datasets. When considering the recall on the 'diabetes' dataset, `fit_intercept=False` performed slightly better, but the overall best choice was again False, as we will see later.

The parameter `dual` gave better results for accuracy and recall when set to 'False' for the 'Census Income' dataset. On the other datasets it did not make much of a difference, only the accuracy on the 'diabetes' dataset was slightly better when setting `dual` to 'False'.

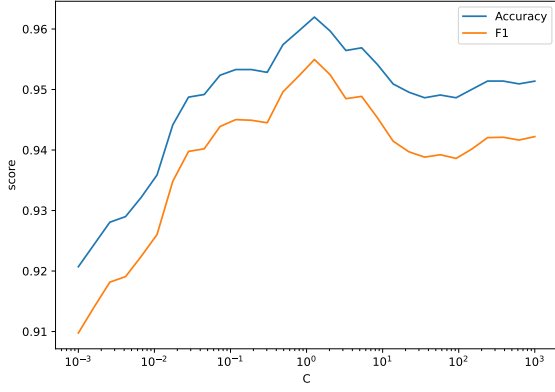
We still have to analyse the sensitivity plots for the dataset diabetes to figure out what parameter values are the best overall.

In figure 3 we can see, that for very small values of `C` the two scores diverge heavily, which means the `C` value found in table 7 is not suitable for optimizing accuracy and recall at the same time.

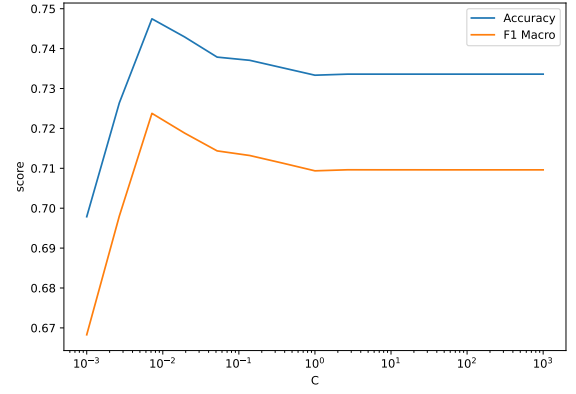
To get the best recall with still decent accuracy we choose `class_weight='balanced'` and `fit_intercept='True'` and `C=0.0452`. This gives us an accuracy of 0.747 and a recall of 0.734.

Ridge Classifier Sensitivity:

Interestingly we found that for the ridge classifier the parameter `alpha` hardly influenced the effectiveness scores of our models. While the measures were nearly left untouched by variation of `alpha` for the 'Census Income' and the 'diabetes' dataset, the 'Amazon Commerce Reviews' dataset showed a little variation in accuracy and



(a) Parameter C for dataset 'congressional voting'



(b) Parameter C for dataset 'Amazon Commerce Reviews'

Figure 2: Sensitivity analysis of linear support vector classifier

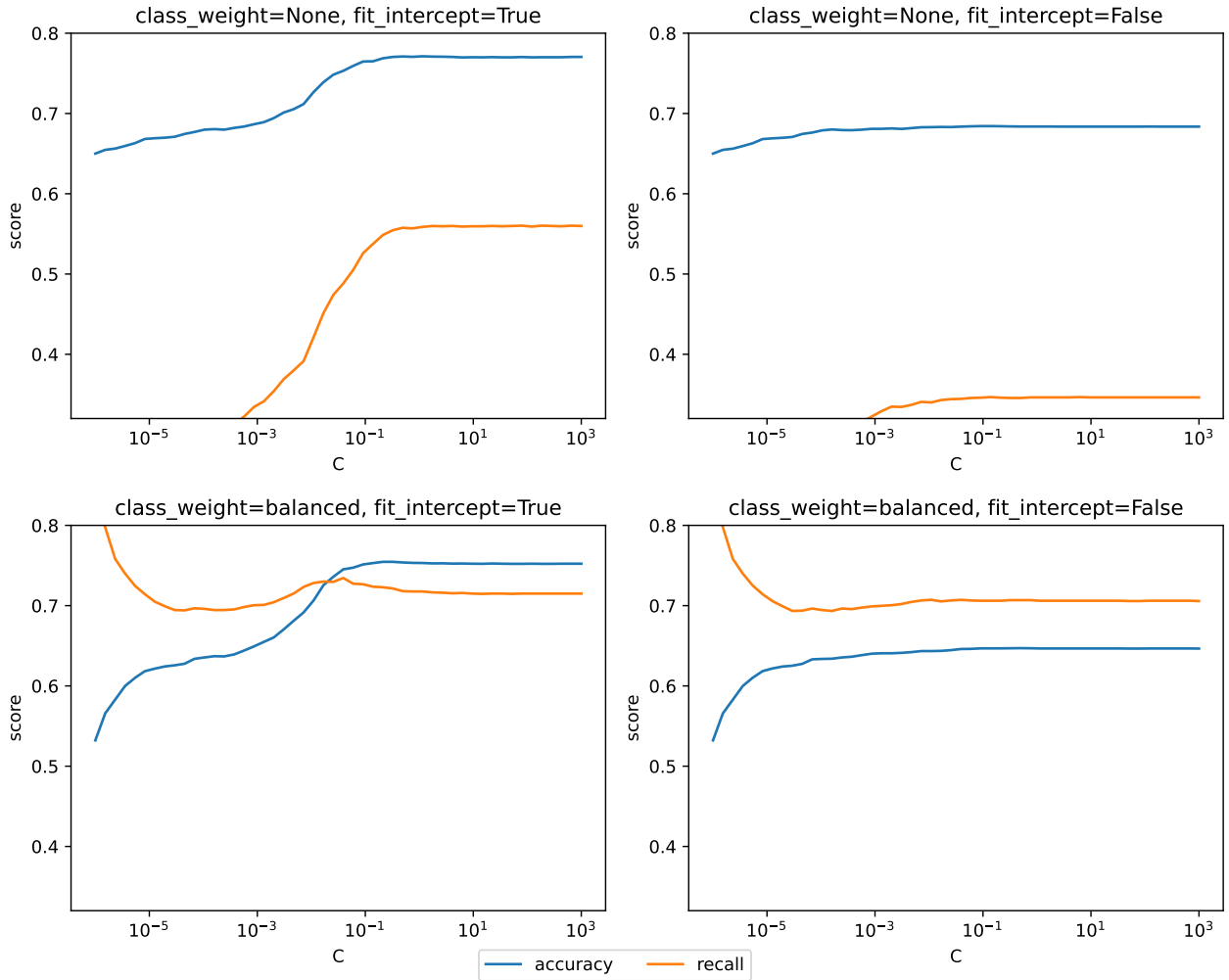


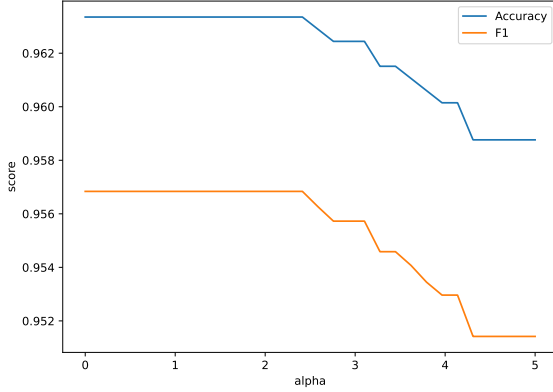
Figure 3: Sensitivity of Parameter C for datasets 'diabetes'

F1 score and the 'congressional voting' showed a decrease in both metrics as one can see in figure 4 but the difference was not really significant considering the scaling of the plot.

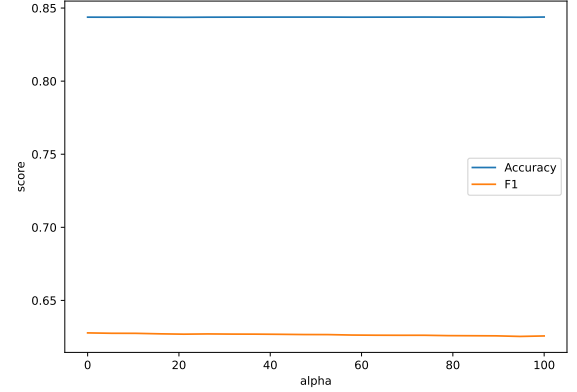
Setting `class_weight` to 'balanced' improved the F1 score/recall for the 'Census Income' and the 'diabetes' dataset while lowering the accuracy as one could expect due to the bias variance trade off. For the other two datasets this hyperparameter did not really influence performance at all.

Finally we want to analyse the parameter `fit_intercept`. This parameter had no significant influence on the dataset 'Census Income' and only slightly increased accuracy and F1 score when set to 'True' for 'congressional

voting'. For 'Amazon Commerce Reviews' interestingly setting it to 'False' increased the scores of both metrics by about .5 percentage points. The biggest influence that we could observe was when testing it on the 'diabetes' Dataset. Here `fit_intercept=False` yielded a pretty significant increase in recall but a decrease in accuracy as one can see from figure 5.



(a) Parameter `alpha` for dataset 'congressional voting'



(b) Parameter `alpha` for dataset 'Census Income'

Figure 4: Sensitivity analysis of ridge classifier

Once more we are not done yet, as we still need to decide on the best parameter setting for the diabetes dataset. We have already established, that changing `alpha` does not really influence the performance of the model, so we the more important question is what combination of the other two parameters works best.

By looking at figure 5 we can see that we have to set `class_weight` to 'balanced' and `fit_intercept` to 'False' when favoring recall over accuracy because this gives us the highest recall and in the other cases the recall was either a lot worse or the accuracy only increased marginally. For this combination of `fit_intercept` and `class_weight` the best value for `alpha` is 0 and we get an accuracy of 0.737 and a recall of 0.797.

6 Conclusion

The goal of this section is to compare the classifiers for each dataset and give a general overview on the performance of the tested classifiers. To make this comparison fair, we ensured to use exactly the same splits within each dataset in the cross-validation. This is achieved by always using the same random state when creating the splits. Of course we applied the most successful preprocessing steps as discussed in section 4 and the hyperparameter settings established in section 5, which ensures that each classifier performs as good as possible. This way we can accurately state which classification algorithm performed best for each dataset.

In table 10 we summarized the top performance measures for each dataset and classifier. We also compared the best result for accuracy with the accuracy of the "Dummy classifier", which is shown in table 11. The "Dummy classifier" always predicts the majority class, so it can be used to establish a baseline. If the accuracy of a classifier is better than this baseline, then that classifier must have learned something. We can only use accuracy as a metric for the "Dummy classifier", because the F1 score is not defined as you would divide by zero in the calculation.

| | Congressional Voting | | Amazon Reviews | | Census Income | | Diabetes | |
|---------------|----------------------|-------|----------------|-------|---------------|-------|----------|--------|
| | Accuracy | F1 | Accuracy | F1 | Accuracy | F1 | Accuracy | recall |
| Random Forest | 0.959 | 0.952 | 0.649 | 0.630 | 0.866 | 0.684 | 0.766 | 0.623 |
| LinearSVC | 0.960 | 0.952 | 0.648 | 0.629 | 0.805 | 0.677 | 0.645 | 0.706 |
| Ridge | 0.963 | 0.957 | 0.699 | 0.684 | 0.844 | 0.628 | 0.737 | 0.795 |
| Dummy | 0.613 | - | 0.020 | - | 0.759 | - | 0.653 | - |

Table 10: 5-fold cross-validation results for each dataset and classifier

The scores in table 10 are computed using repeated stratified 5-fold cross-validation. We chose 10 repetitions for the datasets "congressional voting" and "diabetes" and 5 repetitions for the other two datasets (due to runtime). From this table we can conclude that on the dataset 'congressional voting' and 'Amazon Commerce Reviews' dataset the ridge classifier performed the best. For the dataset 'Census income' we can see that the linear support vector classifier is performing best with respect to both performance measures. Lastly, for the diabetes dataset there is no clear choice. But if we keep our preferences the same as in the last few sections, we can conclude that either the linear support vector classifier or the ridge classifier are the best choice for this

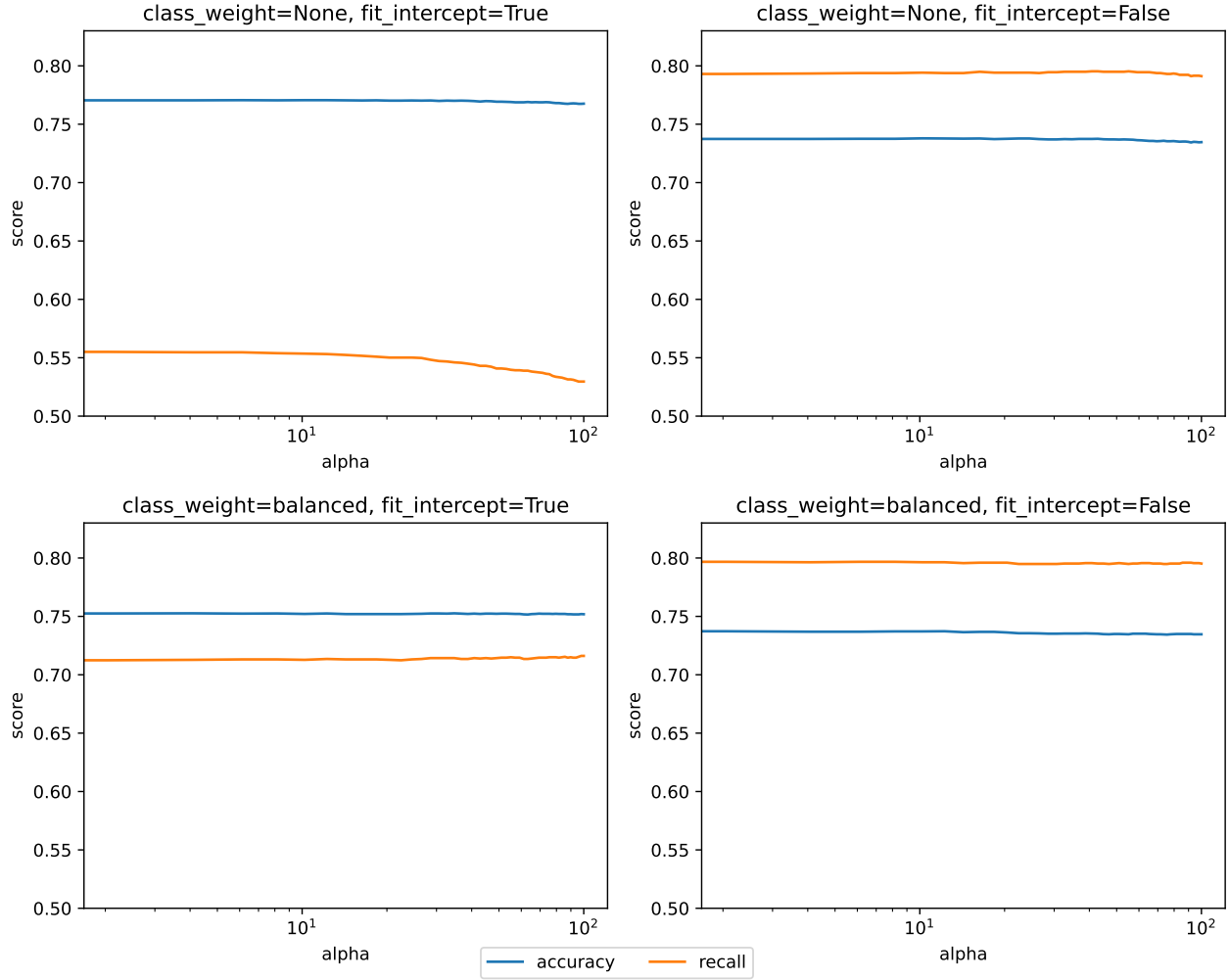


Figure 5: Sensitivity of Parameter α for datasets 'diabetes'

dataset.

At this point we also want to compare cross-validation to the hold-out method. The hold-out method works by taking 70% of the data as the training set and the remaining 30% as the test set. Then on the test set an evaluation is done. The cross-validation has already been introduced in a previous section.

The scores for the hold-out method highly depend on the split and the model chosen as seen in figure ??, where the box-plots are shown for the '*Amazon Commerce Reviews*' dataset.W

In contrast, for the dataset '*Census Income*' the hold-out scores are almost the same as the mean of the cross-validation, which is to be expected because this dataset has a lot of instances. The box-plots for this dataset (figure 9) are also comparatively very narrow, meaning that the scores calculated during cross-validation vary very little i.e. the variance is really low.

We also want compare the efficiency of the classifiers. For this we used the runtime of the hold-out method calculation from above. We measured both the time for training (including preprocessing) and the time it took the test the model. Those values are displayed in table 11.

| Model | Census | | Amazon | | Congress | | Diabetes | |
|--------------|--------|------|--------|------|----------|------|----------|------|
| | Train | Pred | Train | Pred | Train | Pred | Train | Pred |
| LinearSVC | 0.17 | 0.02 | 0.51 | 0.02 | 0.00 | 0.00 | 0.00 | 0.00 |
| Ridge | 0.09 | 0.02 | 7.91 | 0.20 | 0.00 | 0.00 | 0.00 | 0.00 |
| RandomForest | 126.59 | 1.63 | 12.92 | 0.10 | 1.00 | 0.03 | 4.79 | 0.09 |

Table 11: Runtime of models for different datasets

On the contrary, the linear support vector classifier and ridge classifier performed similar on the first glance. The results from '*Amazon Commerce Reviews*' and '*Census Income*' might suggest, that the linear support

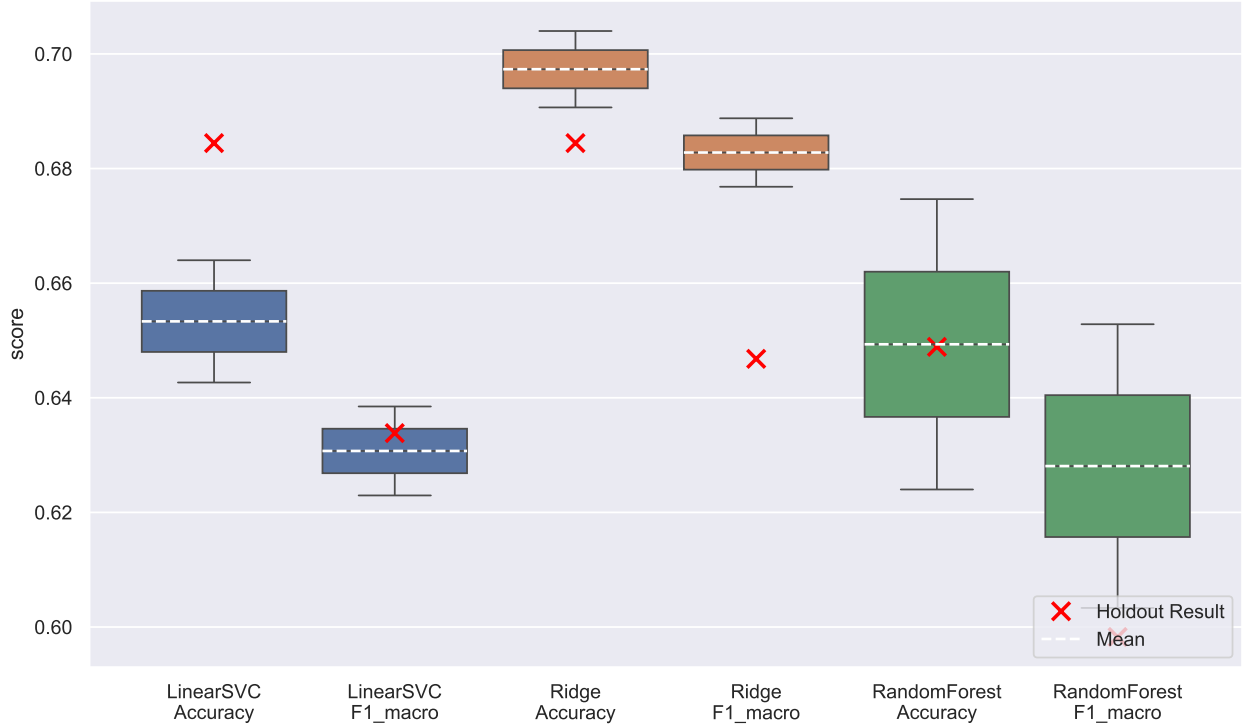


Figure 6: Model Comparison for 'Amazon Commerce Reviews' dataset

vector classifier is faster for high dimensional datasets, whereas the ridge classifier is quicker with a high number of instances. We found out that the parameter dual of the linear support vector classifier has a great influence on its runtime. This parameter describes whether the algorithm should solve for the primal or dual optimization problem. Let n be number of samples and d the number of dimensions of a dataset. The dual formulation requires the computation of a $n \times n$ matrix, whereas in the primal formulation a $d \times d$ matrix is calculated. Therefore the primal formulation is faster for datasets, where $n \gg d$, which was especially noticeable for the "Census Income" dataset. Also on the 'Diabetes' dataset as it is very low dimensional the dual procedure is faster. The results are shown in table 12.

| Dataset | Normal | Dual |
|----------|--------|-------|
| Census | 0.198 | 1.43 |
| Amazon | 2.27 | 0.648 |
| Census | 0.005 | 0.007 |
| Diabetes | 0.004 | 1.25 |

Table 12: Comparison of Dual and Normal Ridge classifier training time for different datasets

After all experiments our final conclusion is that no classifier we used outperformed the other across all datasets. The results in table 10 imply that the effectiveness of a classifier is highly dependent on the dataset. What we can establish is that the linear support vector classifier was never the worst in both metrics. Therefore it seems to be a very versatile classifier. In comparison the success for random forests varied quite a lot for different datasets and effectiveness measures. This could be due to the fact, that decision trees generally tend to overfit the model. As for efficiency, table 11 highlighted a big weakness of the random forest classifier, namely that it's comparatively very slow. Summing up section ??, choosing the right preprocessing steps, especially the scaler and dimension reduction, influences the results notably, though in varying magnitude for different classifiers and datasets. Here random forest seems to be the least influenced by the choice of scaler as seen in tables 2 and 3. The sensitivity analysis in section ?? showed how much of an impact even a single hyperparameter can have on the overall performance of a model. However, comparing the results of section 4 with the ones in section 5 and table 11, it is clear that tuning the parameters hardly influenced the result for the "congressional voting" dataset. In contrast, the recall score for the "diabetes" dataset and both effectiveness scores for the "Amazon Commerce Reviews" improved a lot after the parameter tuning. For the dataset "Census Income" only the support vector classifier improved after optimizing the F1 score. As a summary of this whole project we

would say that the best strategy for a classification problem majorly depends on the dataset in the beginning. Once a classifier is chosen everything else depends on the combination of the two including the importance of parameter tuning and preprocessing as well as the overall success.