Exercise 1

Your Name

November 11, 2024

Abstract

This is the abstract of the document. It provides a brief summary of the content.

Contents

1 Introduction

This is the introduction section. Here you can introduce the topic of your document.

2 Methodology

This section describes the methodology used in your work.

2.1 Subsection Example

This is an example of a subsection.

3 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 minimun 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 tow 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.

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.

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* datasetW

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. . . ,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 hat 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

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

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	**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.

4 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 rnage 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.

5 Discussion

This section discusses the implications of your results.

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 4 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	0.973	0.966	0.708	0.667	0.805	0.677	0.766	0.623
LinearSVC	0.969	0.961	0.742	0.710	0.866	0.684	0.736	0.796
Ridge	0.968	0.959	0.743	0.706	0.844	0.626	0.737	0.795
Baseline	0.624	-	0.031	-	0.761	-	0.651	-

Table 4: Best results for each dataset and classifier

The scores in table **??** 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'* the random forest classifier performed the best. If we again prefer F1 score for the dataset *'Amazon Commerce Reviews'*, then the linear support vector classifier gives the best results. 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 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 chosen as seen in figures **??** and **??**, where the box-plots are shown for the datasets *'congressional voting'* and *'Amazon Commerce Reviews'*. 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 **??**.

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 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 × n matrix, whereas in the primal formulation a d × d matrix is calculated. Therefore the primal formulation is faster for datasets, where n ¿ d, which was especially noticeable for the "Census Income" dataset. Also on the *'Diabetes'* dataset as it is very low dimensional the dual prosedure is faster. The results are shown in table 5.

Dataset	**Not Dual**	**Dual**
Census	0.198	1.43
Amazon	2.27	0.648
Census	0.005	0.007
Diabetes	0.004	1.25

Table 5: Comparison of Dual and Not Dual values for different datasets

References

[1] Author, *Title of the Book*, Publisher, Year.

[2] Author, *Title of the Article*, Journal, Volume, Page numbers, Year.