ML - Exercise 2: Classification

TU Vienna, 2019 WS
Schafellner Jakob 1404727,
Quotschalla Moritz, 1321554,
Beyweiss Raphael, 1525113

Datasets

Amazon (Kaggle)

The Amazon dataset comes from one of the Kaggle in class competitions and it's the biggest of our datasets. It has 750 rows and 10001 features. Since this dataset provides much more features than samples, it's easy to run into overfitting problems.
The target variable is called 'Class' and it contains labels which need to be encoded. The target variable is balanced within our training set.

Breast Cancer (Kaggle)

The Breast Cancer dataset is the small dataset from the Kaggle in class competitions. It has 285 samples, 30 features and a binary target variable, which either says that there was a recurrence-event or not. The features are all properties of the original found cancer.

Cardiotocography

The Cardiotocography dataset is our first own dataset. The dataset has 2126 samples, 21 features and two target variables. One defines 3 classes for the fetal heart rate, which is just a basic classification into normal, suspect and pathologic. The other one defines 10 classes, which is obviously more detailed but is also not directly mapable to the 3 other classes. The features are all statistical features of the cardiotocography.

Heart Disease

The Heart Disease dataset is our second own dataset. It has 303 instances, 13 features and a binary target variable, distinguishing the presence of heart disease in a patient. The features are basic data like age and sex and health data concerning mainly the state of the heart of a patient.

Scikit Learn

For this exercise we used the library Scikit Learn. On the one hand we chose this library because we wanted to use Python and on the other hand because the library has a very intuitive Api which makes it very easy to try out and work with.
In order to get the most out of this simple API and to be able to try many different approaches as fast as possible, we decided to develop with Jupyter Notebooks. This allows us to execute code in small blocks while the state of the program remains available.

In the next paragraphs we describe functionalities of Scikit Learn which helped us a lot during this task.

Pipeline

On the one hand, we have used the pipeline functionalities of Scikit Learn. This made it possible to perform different steps one after the other, whereby the output of one step always acts as input for the next. Different transformations of the data work differently well for different types of machine learning techniques. Here the pipeline allows a quick testing of different preprocessing steps without changing much code. You just swap one step in the pipeline and the rest works as usual. This way of working also allows a good comparison of the results if the model remains the same and only the preprocessing steps change. Also, post processing steps could be done very handy if you want to make sure that the output of the results is always in a consistent form.

GridSearchCV

In order to make the hyperparameter tuning of the model clearer we have used the function GridSearchCV of Scikit Learn for this task. Here different parameters can be defined in the form of lists and these are then tested and evaluated in all possible combinations. In order to make the results more relevant, a triple cross validation is used by default.
We have also worked with triple cross validation.

Classifiers

We chose five different classifiers because we were very interested in how the results of the different models differ.
In the next paragraphs we will describe the used techniques roughly and explain why we have decided for these techniques.

Logistic Regression

As the first model we decided for Logistic Regression. The functionality of Logistic Regression is easy to understand and we wanted to use this model to find out how sophisticated the classification of different datasets is and how much information we can get

from it with a more simple model. Furthermore, when using complicated models, things can go wrong and we think it's a good idea to check the basic functionality with a simpler model in order to also examine less potential error sources in case of an error.

KNN-Classifier

Next we have decided for a KNN model. The idea behind KNN is easy to understand and the results of this classifier may be very good. The hyper parameter tuning is limited in this case to the number of neighbours considered. Thus a fast processing is also possible with large datasets that don't have to try so many different parameters.

Random Forest Classifier

We choose this classifier because it worked out pretty good in the first exercise and we wanted to show, how well it could handle classification tasks.

GradientBoostingClassifier

We choose this classifier because it worked out pretty good in the first exercise and we wanted to show, how well it could handle classification tasks.

MLP-Classifier

We choose this model because we were interested in a comparable measurement of a deep learning approach.

Metrics

In order to determine whether our predictions deliver meaningful results, we need a way to determine how good these predictions are. If you have classification problems, you could diffuse the number of correctly classified samples by the number of all samples to see how many percent of all samples are correctly classified. This number often has little validity. If, for example, there are significantly more samples of a class, a model that always predicts this class would classify a high percentage of all samples correctly.
As an example, one could imagine a spam filter. Since on average more normal mails are received than spam mails, spam filter that never classifies a mail as spam would correctly classify a relatively high percentage of all mails. Since this does not bring any benefit, we have to think about other evaluation methods.
The following paragraphs describe the metrics used by us in the course of our work.

Accuracy

As described before accuracy can be calculated by using the number of correct predictions and dividing it by the total number of predictions.

Confusion Matrix

A confusion matrix is a rather popular way to determine if a classification model did a good job. It's basically a matrix, representing the predictions and the actual classes.
Each row of the matrix represents the instances in a predicted class while each column represents the instances in an actual class. The name stems from the fact that it makes it easy to see if the system is confusing two classes (i.e. commonly mislabeling one as another).

Precision

Precision is another way of measuring the quality of a model. It can be calculated by dividing the number of true positives by the sum of true positives and false positives.
Immediately, you can see that Precision talks about how precise/accurate your model is out of those predicted positive, how many of them are actual positive.
Precision is a good measure to determine, when the costs of False Positive is high. For instance, email spam detection. In email spam detection, a false positive means that an email that is non-spam (actual negative) has been identified as spam (predicted spam). The email user might lose important emails if the precision is not high for the spam detection model.

Recall

Recall can be calculated by dividing the number of true positives by the sum of true positives and false negatives. Recall actually calculates how many of the Actual Positives our model capture through labeling it as Positive (True Positive). Applying the same understanding, we know that Recall shall be the model metric we use to select our best model when there is a high cost associated with False Negative.
For instance, in fraud detection or sick patient detection. If a fraudulent transaction (Actual Positive) is predicted as non-fraudulent (Predicted Negative), the consequence can be very bad for the bank.
Similarly, in sick patient detection. If a sick patient (Actual Positive) goes through the test and predicted as not sick (Predicted Negative). The cost associated with False Negative will be extremely high if the sickness is contagious.

F1 Score

The F1 score is very popular and can be calculated by the following formula:

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

F1 Score is needed when you want to seek a balance between Precision and Recall. What is the difference between F1 Score and Accuracy then? We have previously seen that accuracy can be largely contributed by a large number of True Negatives which in most business circumstances, we do not focus on much whereas False Negative and False Positive usually has business costs (tangible & intangible) thus F1 Score might be a better measure to use if we need to seek a balance between Precision and Recall AND there is an uneven class distribution (large number of Actual Negatives).

Report on work

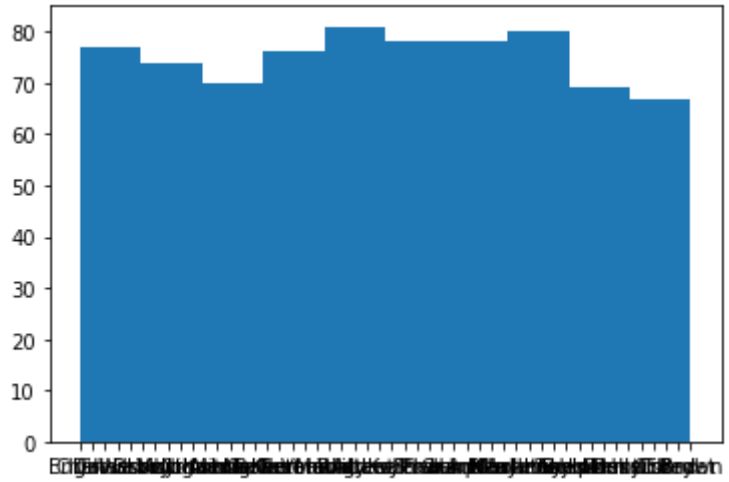
Mentioned scores are calculated by using a hold out test validation set.

Amazon

As already mentioned in the datasets section, the Amazon dataset was part of a kaggle inclass competition and it's the biggest dataset that we've used.
The speciality of this dataset is that it contains a lot more features than samples what might become a problem for our models (750, 10001). Datasets with a lot more features are prone to cause overfitting problems.

First Approach

We took a look at the dataset and realized that there are no missing values. We won't need to impute data.
The next step was to analyze the target variable. We had to predict a class. The different classes seem to be rather balanced in this dataset.



This plot shows the histogram of the classes in the amazon dataset. Since the classes are balanced it is ok to use accuracy as our scoring metric.

Since the classes are labels we need a way to encode them because our machine learning models can only work with numeric input.
We used the LabelEncoder from sklearn which transforms our class label into a unique numbers that work with our machine learning models. After the prediction it's possible to inverse this transformation to get the corresponding labels back.

After these preprocessing steps we wanted to get a feeling for the challenge and we wanted to evaluate how well, a rather basic model would perform. Therefore we used the logistic regression on this dataset which lead to 0.55 accuracy.

We could further improve accuracy by using a robust scaling method before, to 0.64 accuracy.
This was our final score for our logistic regression model, even with exhaustive hyperparameter tuning, we couldn't get better results.

Logistic Regression

Score history:

description	accuracy	marco avg	weighted avg
first try	0.55	0.56, 0.54, 0.53	0.57, 0.55, 0.54
with robustScaler	0.64	0.70, 0.63, 0.63	0.68, 0.64, 0.62
with accuracy scoring	-	-	-

Second Approach

We had our baseline from our logistic regression model and we wanted to try out other models to see how well they would work, in comparison to this baseline.
Next we chose a random forest model. This model performed pretty good in our last exercise and we wanted to investigate if it would also perform well in this exercise.

Our random forest model gave us 0.58 accuracy. We managed to get this score up to 0.63 and our Kaggle score for this submission resulted in 0.72 accuracy.
Random forest models are known to handle datasets with more features than samples quite good. It's very important to unseen data to evaluate the model's score since it will perform too good for seen data.

accuracy			0.63	225
macro avg	0.70	0.61	0.62	225
weighted avg	0.69	0.63	0.62	225

Approaches to reduce number of features

Reducing the number of features results in a easier model, that could be trained faster and it should also generalize better. Unfortunately we couldn't improve our results with this techniques.

At first we tried to reduce features with a variance of zero. If a feature value won't have different values in different rows it won't be very important to predict the class of the rows. Our next try was to use a *l2* penalty to reduce the feature number. Our third and last approach was to use PCA components as features. We didn't find a model that worked better with these approaches than our random forest model.

Approaches with multiple models

We tried to ensemble multiple similar performing models and hoped for a slight boost in our performance. We combined our random forest model with our logistic regression model and we also tried these two combined with a support vector machine. We used a voting classifier which evaluates the class based on majority voting. We also tried to evaluate the class by summing up the probabilities of each class and predict the class with the highest sum.
The ensemble approach didn't lift our accuracy. It stayed the same or decreased slightly.

Other models

GradientBoostingClassifier:

accuracy			0.36	225
macro avg	0.44	0.32	0.31	225
weighted avg	0.47	0.36	0.34	225

MLP (basic setup):

accuracy			0.14	225
macro avg	0.15	0.13	0.12	225
weighted avg	0.16	0.14	0.13	225

KNN-Classifier:

accuracy			0.06	225
macro avg	0.05	0.05	0.03	225
weighted avg	0.05	0.06	0.03	225

Breast Cancer

In total there are 30 features and 285 samples. All features are float values with different ranges, some smaller than 1 and some at range of a couple hundreds, which implies that for some methods scaling is necessary. The class is a binary variable with a ⅔ : ⅓ distribution.

There were no missing values and as all features are already float values no further preprocessing except from scaling was necessary.

In the first step we tried again the logistic regression model, which performed relatively poor directly. But when we tried to use a scaler as a preprocessing step, the model worked quite well. Also the test partition had an accuracy of 100%, which basically sounds nice but it can also indicate that the model overfits to the train partition of the dataset.

Logistic Regression

Score history:

description	accuracy	marco avg	weighted avg
first try	0.34	0.17, 0.50, 0.25	0.11, 0.34, 0.17
with MinMaxScaler	0.98	0.98, 0.97, 0.97	0.98, 0.98, 0.98
with RobustScaler	1.00	1.00, 1.00, 1.00	1.00, 1.00, 1.00
-	-	-	-

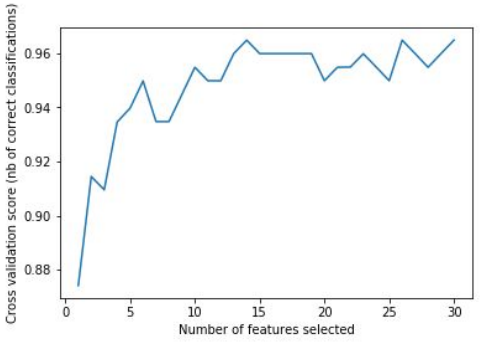
As the next model we tried the knn-model, which performed not that bad without any scaling, but obviously with scaling performed way better, but we did not reach the 100% accuracy mark as with the other models, which indicates that the model is basic enough to not overfit the dataset. This intuition was confirmed by the kaggle competition where this model performed best for us.

KNN-Classifier

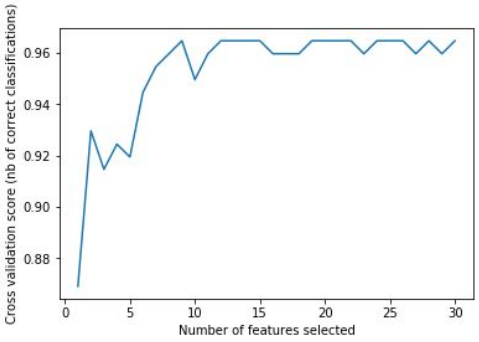
Score history:

description	accuracy	marco avg	weighted avg
first try	0.83	0.82, 0.78, 0.79	0.82, 0.83, 0.82
with MinMaxScaler	0.98	0.98, 0.97, 0.97	0.98, 0.98, 0.98
-	-	-	-

The third used model we used was random forest, which performed very well from the beginning and did not need any scaling as preprocessing step. But again the model reached the 100% accuracy mark for the test partition, and there was as well most likely overfitting. To reduce this effect we tried to drop features with recursive feature elimination. We tried this but did not exceed the simpler model knn. Also it did not seem to have very stable results as seen in the plot, it looks like the best feature combination is just a lucky guess for the given dataset.

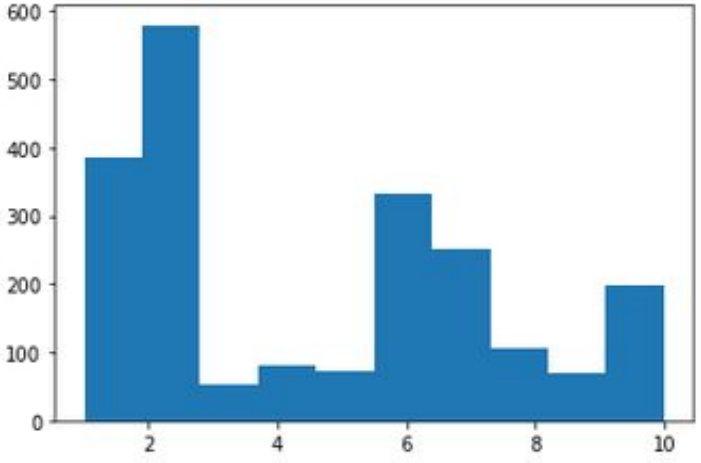


The last model we tried was gradient boost, which performed very similar to the random forest model. As well we tried to reduce the overfitting effect by recursive feature elimination, but the model was still not better than knn. Here the feature elimination looks more stable and it seems that it is possible to reduce the feature set by around ⅔, Which again implies that it is very likely that the model is overfitting.



Cardiotocography

There are 21 features and 2126 samples. All features are either float values or integer values with different ranges without missing values. There are two possible class types, one with 3 classes and one with 10 classes. We opt for the one with 10 classes as it is “more” different from the other dataset. The classes are not uniformly distributed, therefore we used F1 score.



The first model was again the logistic regression model, which performed initially not that bad as there are 10 classes. But again it performed way better with scaling. Again robust scaling worked better than min max scaling.

Logistic Regression

Score history:

description	accuracy	marco avg	weighted avg
first try	0.67	0.69, 0.54, 0.55	0.68, 0.67, 0.66
with MinMaxScaler	0.75	0.71, 0.58, 0.60	0.75, 0.75, 0.73
with RobustScaler	0.79	0.73, 0.68, 0.69	0.79, 0.79, 0.78
-	-	-	-

The knn model worked surprisingly well without scaling, even tho the ranges of the features are very different, the only plausible reason we could think of is that there are many features where the correlation value is pretty high, so most of the features got indirectly almost the same range. But anyways scaling improved the results a bit.

KNN-Classifier

Score history:

description	accuracy	marco avg	weighted avg
first try	0.75	0.4, 0.68, 0.70	0.74, 0.75, 0.74
with RobustScaler	0.77	0.74, 0.71, 0.72	0.77, 0.77, 0.77
with MinMaxScaler	0.78	0.77, 0.69, 0.71	0.78, 0.78, 0.78
-	-	-	-

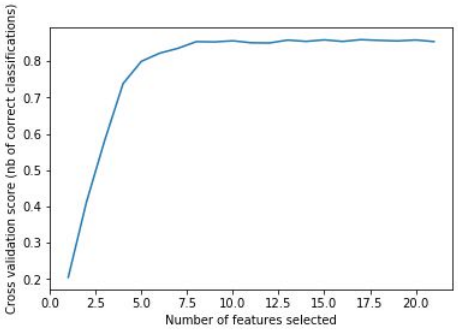
The random forest and gradient boosting models had the best results for the dataset, again we tried to improve the results with feature dropping. While this improved the results for random forest, the results for the gradient boost model got worse. Also the best feature set seem to be anyways reducible by 4-5 features as both models can be reduced by the same features, and probably can be reduced even further as the plots imply, since after 8-10 features there are only very small changes in the results.

Random Forest

Score history:

description	accuracy	marco avg	weighted avg
first try	0.86	0.84, 0.80, 0.82	0.86, 0.86, 0.86
recursive features selection	0.87	0.86, 0.83, 0.84	0.87, 0.87, 0.86
-	-	-	-

```
Optimal number of features : 17
Optimal features:
 Index(['LB', 'AC', 'FM', 'UC', 'DL', 'DP', 'ASTV', 'MSTV', 'ALTV', 'MLTV',
        'Width', 'Min', 'Max', 'Mode', 'Mean', 'Median', 'Variance'],
       dtype='object')
```

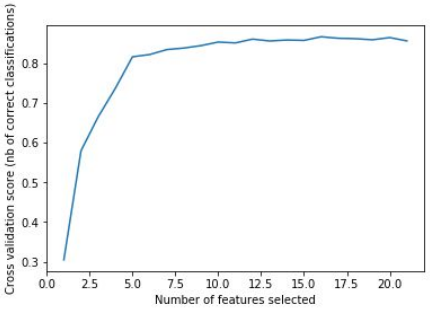


Creating a GradientBoostingClassifier

Score history:

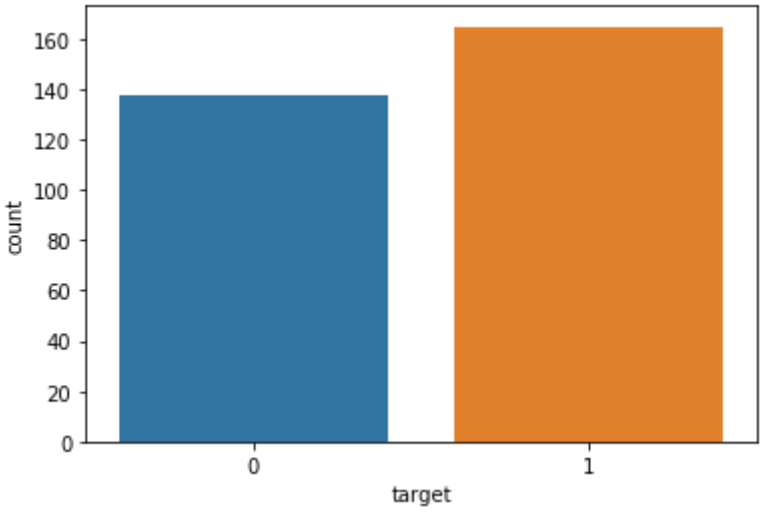
description	accuracy	marco avg	weighted avg
first try	0.86	0.85, 0.80, 0.81	0.86, 0.86, 0.86
recursive features selection	0.85	0.83, 0.79, 0.80	0.85, 0.85, 0.85
-	-	-	-

```
Optimal number of features : 16
Optimal features:
 Index(['LB', 'AC', 'UC', 'DL', 'DP', 'ASTV', 'MSTV', 'ALTV', 'MLTV', 'Width',
        'Min', 'Max', 'Mode', 'Mean', 'Median', 'Variance'],
       dtype='object')
```



Heart Disease

This dataset has 14 features and 303 samples. Except one (float) all features are integer values with different ranges but not missing values. As there were six categorical variables, we also tested the effect of one hot encoding on the results.
The classification is defined with a binary variable, which is evenly distributed. Therefore, over-/undersampling was not necessary.



First we used a logistic regression model, which initially performed already well withoud scaling or one-hot encoding. Using a scaler only improved the results marginally. Performing one-hot encoding before also optimized the performance slightly.

Logistic Regression

Score history:

description	accuracy	macro avg	weighted avg
first try	0.75	0.75, 0.74, 0.74	0.75, 0.75, 0.75
with MinMaxScaler	0.76	0.76, 0.75, 0.75	0.76, 0.76, 0.76
with RobustScaler	0.76	0.76, 0.75, 0.75	0.76, 0.76, 0.76
one-hot encoded	0.77	0.77, 0.77, 0.77	0.77, 0.77, 0.77
with MinMaxScaler	0.78	0.78, 0.78, 0.78	0.78, 0.78, 0.78
with RobustScaler	0.79	0.79, 0.79, 0.79	0.79, 0.79, 0.79

The second approach was the knn model, which worked initially worse than the logistic regression model. Using a scaler had much more positive effect here and notably improved the result. Performing one-hot encoding before only had minor impact without scaling, but improved noticeable with a RobustScaler and even decreased with a MinMaxScaler.

KNN-Classifier

Score history:

description	accuracy	macro avg	weighted avg
first try	0.68	0.68, 0.67, 0.68	0.68, 0.68, 0.68
with MinMaxScaler	0.79	0.79, 0.78, 0.79	0.79, 0.79, 0.79
with RobustScaler	0.79	0.79, 0.78, 0.79	0.79, 0.79, 0.79
one-hot encoded	0.69	0.69, 0.69, 0.69	0.69, 0.69, 0.69
with MinMaxScaler	0.77	0.77, 0.77, 0.77	0.77, 0.77, 0.77
with RobustScaler	0.82	0.83, 0.82, 0.82	0.83, 0.82, 0.82

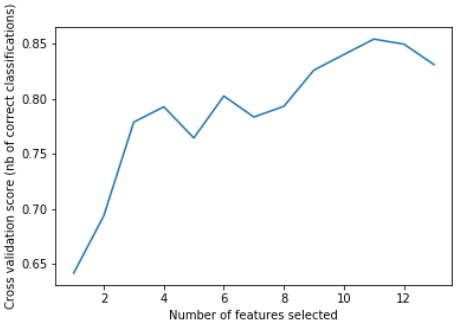
The random forest model had similar results. To further improve the score we applied recursive features selection and again one-hot encoding. While one-hot encoding had only minor impact, the recursive features selection was more effective and achieved one of the best results with 11 features out of 13.

Random Forest

Score history:

description	accuracy	macro avg	weighted avg
first try	0.77	0.77, 0.76, 0.76	0.77, 0.77, 0.77
recursive features selection	0.81	0.81, 0.81, 0.81	0.81, 0.81, 0.81
one-hot encoded	0.79	0.80, 0.78, 0.78	0.80, 0.79, 0.79

```
Optimal number of features : 11
Optimal features:
 Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'thalach', 'exang', 'oldpeak',
        'slope', 'ca', 'thal'],
       dtype='object')
```



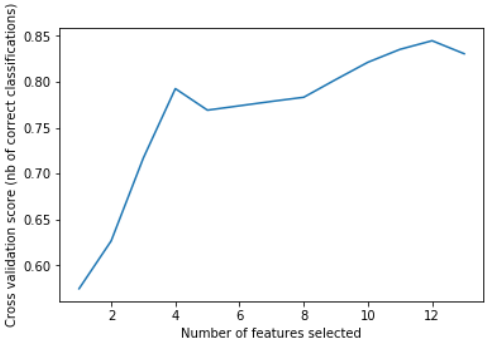
At last, the gradient boosting model performed a bit worse than the random forest model. Again we applied recursive features selection and one-hot encoding and reached similar improvements as for the random forest model. For the gradient boosting model the best results were with 12 features.

Creating a GradientBoostingClassifier

Score history:

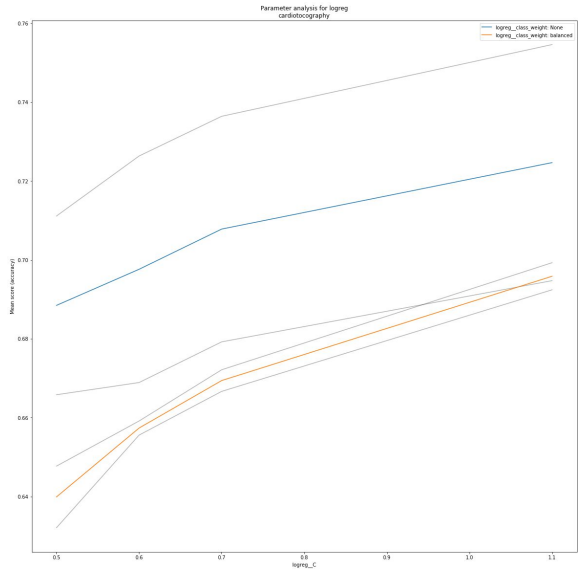
description	accuracy	macro avg	weighted avg
first try	0.76	0.76, 0.75, 0.75	0.76, 0.76, 0.76
recursive features selection	0.79	0.79, 0.79, 0.79	0.79, 0.79, 0.79
one-hot encoded	0.78	0.78, 0.77, 0.77	0.78, 0.78, 0.78

```
Optimal number of features : 12
Optimal features:
 Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'restecg', 'thalach', 'exang',
        'oldpeak', 'slope', 'ca', 'thal'],
       dtype='object')
```

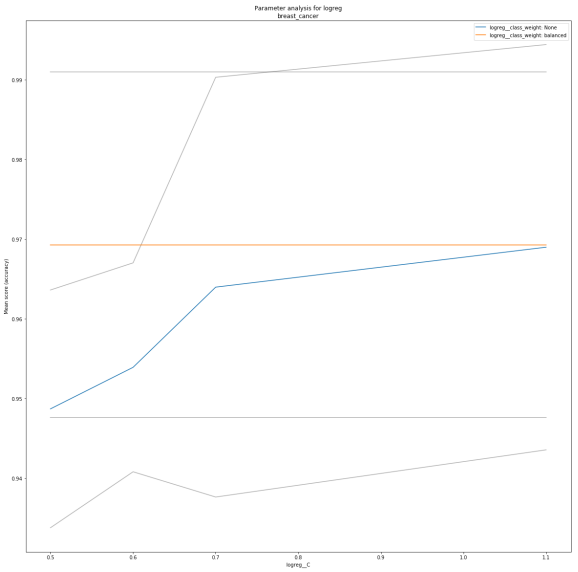


Parameter analysis

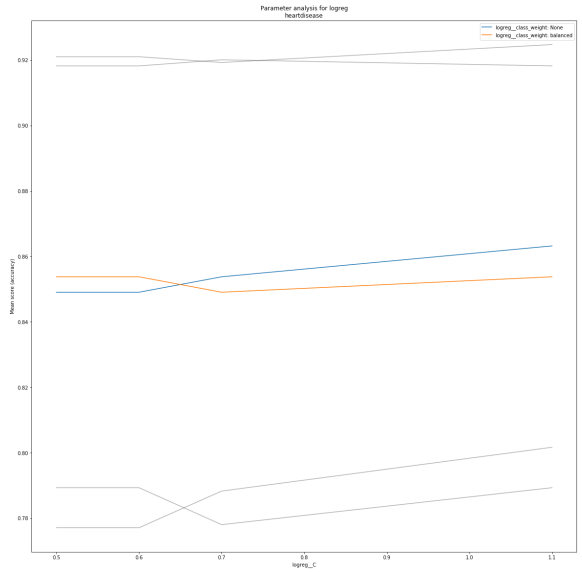
Logistic Regression



Clearly you can observe that by increasing the C parameter the results get better, also no class weight adaption outperforms balanced weights.

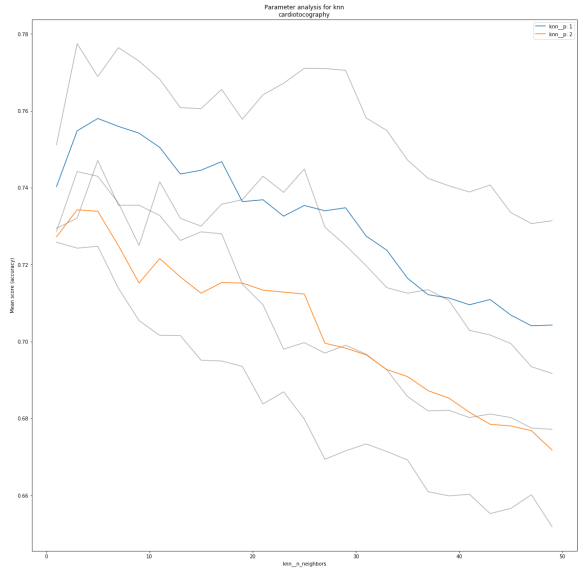


For this dataset if the class weights are balanced there is no change at all for changes of the parameter C. If the class weights are not balanced, again with higher parameter C the results are improved, but at most reach the level of balanced class weights.

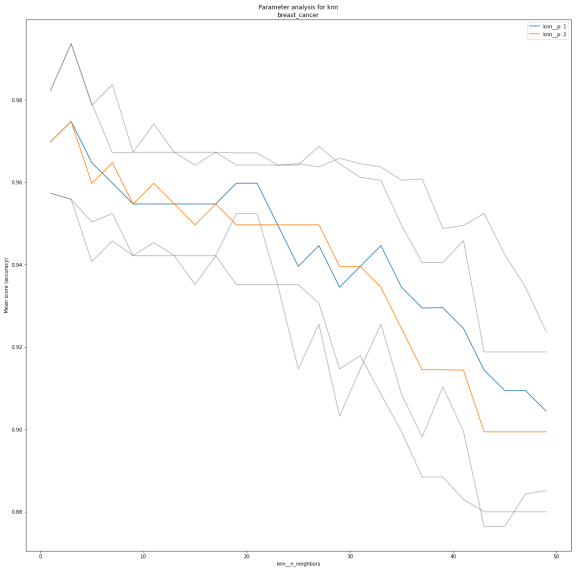


Here as well increasing C increases the results, while for low C values no balanced class weights starts better, for higher C value ranges balanced class weights outperform

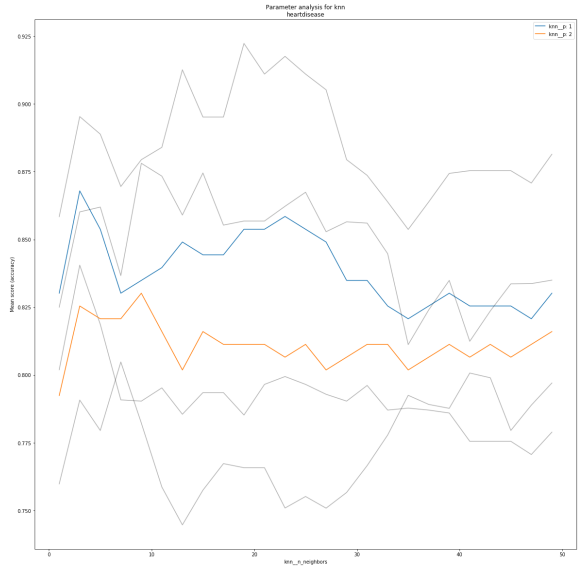
k-Nearest Neighbors



For the first few k values the results are improved but after reaching the peak increasing k is decreasing the results. The distance function p1 outperforms p2.

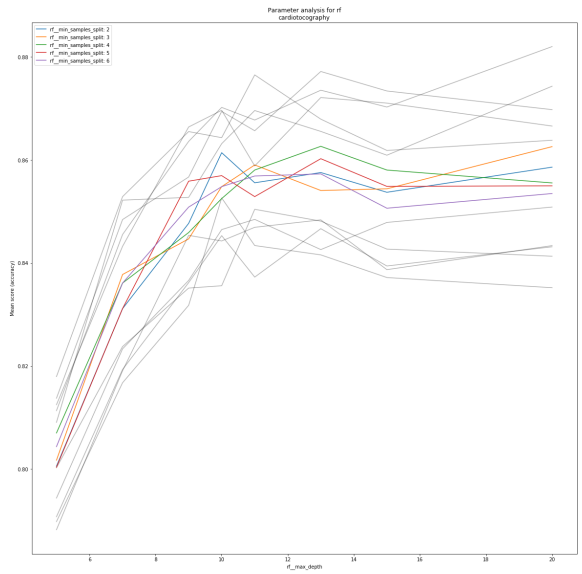


Here as well the after reaching the peak, increasing k is decreasing the results. But in contrast here no distance function performs better than the other.

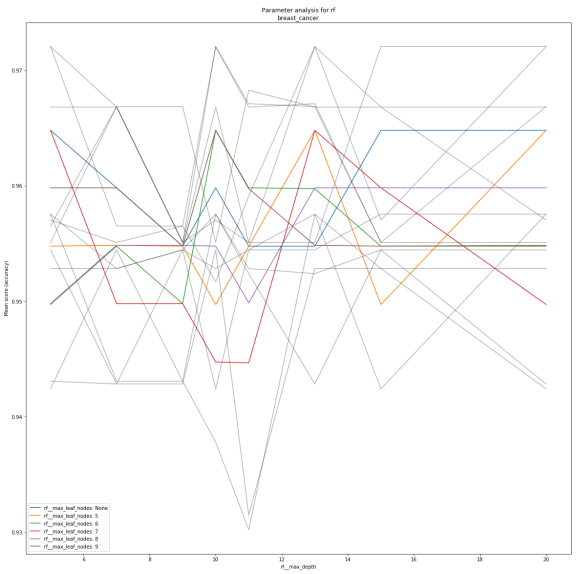


With this dataset increasing k after the peak, does not decrease the results so obvious as for the other datasets, but decreasing pattern is still observable.

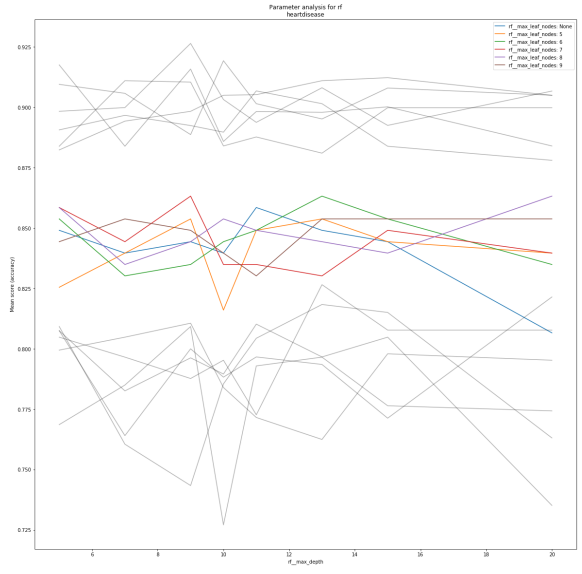
Random Forest



With this dataset you can see that increasing the max depth has in the low ranges a steep curve but in the higher ranges it is flattening out. For the min samples split parameter there is no value which generally outperforms the others

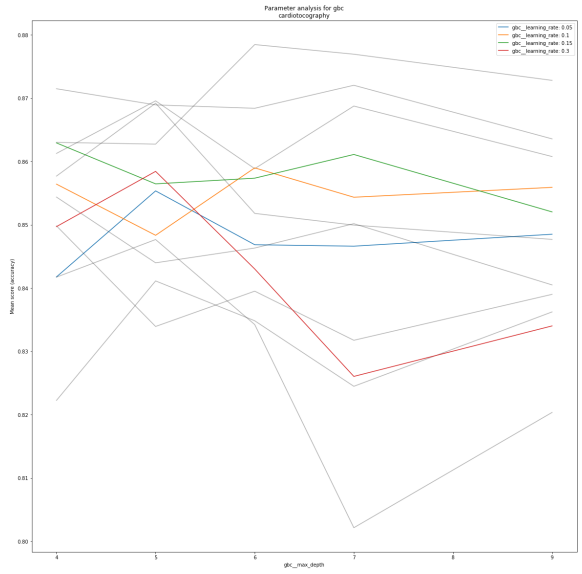


No clear pattern is here observable, which indicates that the used parameter ranges are already in the flatten out part.

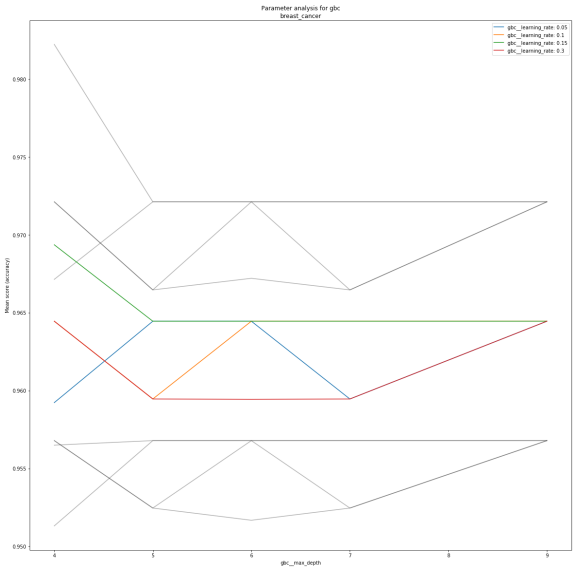


No clear pattern is here observable, which indicates that the used parameter ranges are already in the flatten out part.

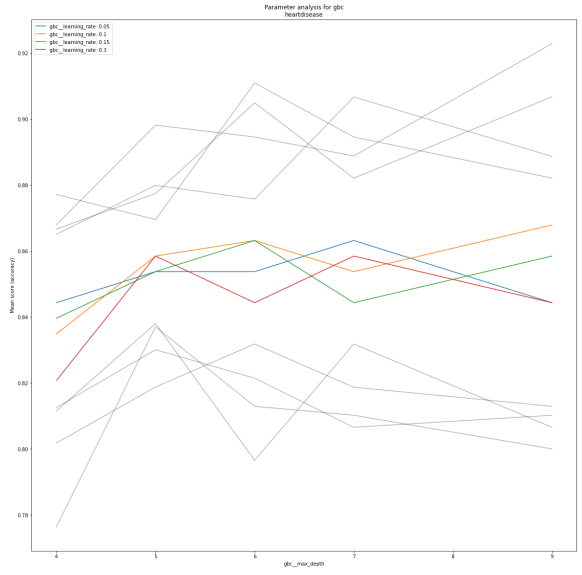
Gradient Boost



We have already seen for the regression task that for increasing the max depth for in the low values range the curve is very steep and in the higher values range the curve is flattening out. Since we can't observe this here, it is an indication that the parameter range seen here is already in the flatten out part. Also we observed in the last task that higher learning rate performs worse in contrast to lower learning rate for higher max depth und better for lower max depth, which is also observable here.



Here none of the patterns really observable, but this can come from the dataset, which is very small and easily overfittable.



Here for the lower max depth values it

indicates a steep curve, which is in the higher ranges flattening out. Also here the lower learning rate tends to get better for higher depth, while higher learning rate tends to decrease.	