Impact of Box-Cox Transformation on Machine-Learning Algorithms

Blum Luca[1*], Menon Carlo[2] and Elgendi Moe[2*]

[1]Department of Mathematics, ETH, Zurich, 8092, ZH, Switzerland.
[2]Department of Health Sciences and Technology, ETH, Zurich, 8008, ZH, Switzerland.

*Corresponding author(s). E-mail(s): lblum@student.ethz.ch; moe.elgendi@gmail.com;

Abstract

The effects of applying the Box-Cox transformation for classification tasks are studied. A consistent improvement of the accuracy is demonstrated. We show that the corresponding optimal parameter choice is dependent on the data as well as the classifier itself. Further an optimization procedure for finding suitable parameters is proposed. This procedure is tested on two real world datasets with promising results.

Keywords: Box-Cox transformation, power transformation, classification

1 Introduction

Feature transformation can improve the performance of a machine learning algorithm. Simple transformations can have already a significant impact on classification performance [1], [2]. Motivated by their findings, the impact of the Box-Cox transformation for classification tasks was studied.
Often Box-Cox is treated as a black-box and just used to increase the Gaussianity of the data. This can help in some special cases but we observed that transformations that do not maximize the Gaussianity of the data are often superior for classification accuracy. Additionally [2] have shown that Gaussianity of datasets is not critical and by letting the Box–Cox transform work

in operational ranges which do not necessarily correspond to an increase in Gaussianity, they have shown that class separability can be improved.

To our knowledge there was no research done for classification except [1] and [2]. Both have shown that transforming the data with a Box-Cox transformation can be beneficial. Additionally, [2] proposed an automatic procedure to get a optimal transformation. Their procedure relied on the optimization of statistical measurements like maximum likelihood or Fisher criterion. This paper generalizes the optimization procedure and makes it independent of statistical measurements.

2 The Box-Cox transformation

The original Box-Cox transformation is a one dimensional transformation with one parameter often called λ and is applied element-wise to a vector y [3] :

$$\text{Let } y \in \mathbb{R}^n \text{ and } \lambda \in \mathbb{R}$$
$$y_i^{(\lambda)} = \begin{cases} \frac{y_i^{(\lambda)}-1}{\lambda} & \text{if } \lambda \neq 0 \\ \ln(y_i) & \text{if } \lambda = 0 \end{cases}$$

Many different criteria for an optimal λ were proposed. The most used one was introduced by [3] in the original paper and is a maximum likelihood estimate (MLE). Other approaches used for example a Bayesian approach [4], robust estimators [5–7] or tried to iteratively maximize the Gaussianity [8]. The Box-Cox transformation is mostly studied for regression tasks. For multi-dimensional data $X \in \mathbb{R}^{n \times p}$ it is usually applied p times as 1-dimensional mapping to each column with different values for λ. Therefore the overall transformation is specified by a p-dimensional vector $\Lambda = [\lambda_1, \lambda_2, ..., \lambda_p]$. The optimization of the parameter vector Λ can be done in several ways. Naturally, one could optimize λ_i of the corresponding column X_i independently with a traditional criteria like MLE [3] or Bayesian [4]. This will be referred as *diagonal* setting.

$$\Lambda^* = \begin{bmatrix} \lambda_1^* \\ \lambda_2^* \\ ... \\ \lambda_p^* \end{bmatrix} = \begin{bmatrix} \arg\min_{\lambda_1} L(\lambda_1, X_1) \\ \arg\min_{\lambda_2} L(\lambda_2, X_2) \\ ... \\ \arg\min_{\lambda_p} L(\lambda_p, X_p) \end{bmatrix} \tag{1}$$

where $L(\cdot, \cdot)$ is a criteria that needs to be minimized. A simplification of this case is the *spherical* setting. There only a scalar value λ gets optimized and applied to every column.

$$\lambda^* = \arg\min_{\lambda} \sum_{i=1}^{i=p} L(\lambda, X_i) \tag{2}$$

The most general case is called *full* and optimizes

$$\Lambda^* = \begin{bmatrix} \lambda_1^* \\ \lambda_2^* \\ \dots \\ \lambda_p^* \end{bmatrix} = \begin{bmatrix} \arg\min_{\lambda_1} L(\Lambda, X) \\ \arg\min_{\lambda_2} L(\Lambda, X) \\ \dots \\ \arg\min_{\lambda_p} L(\Lambda, X) \end{bmatrix} \tag{3}$$

The work in [2] analyzed the *spherical* and *diagonal* setting. They showed that both of them are capable of improving the classification result although the *diagonal* case often gives a higher accuracy. This can be expected due to the higher number of parameters. Furthermore, they demonstrated that the choice of the optimization criteria depends on the classifier itself.

3 Motivation

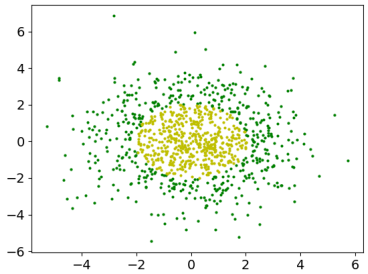
To demonstrate the influence of the Box-Cox transformation a stratified cross validation with 10 folds and 5 repetitions was executed on various artificial 2 dimensional binary classification tasks with varying Λ. For each direction $i \in \{1, 2\}$, λ_i was distributed evenly in the interval [-5, 5] with a spacing of 1. Hence 11×11 accuracy estimates were conducted. The accuracy measurements were carried out for different classifiers described in Table 1, as implemented in the Python library scikit-learn [9]. Unless otherwise stated, the default parameters have been used and if provided random seeds/states were set to 42.

Table 1: Evaluated classifiers: Details can be found in [9]

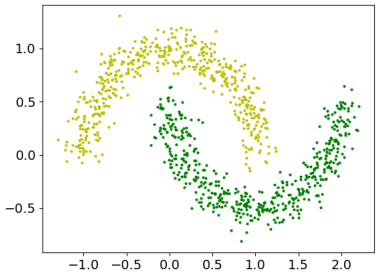
Classifier	Description
Linear	linear classifier with Perceptron loss and trained with SGD
Bayesian	Gaussian naive Bayes classifier
KNN	nearest neighbors voting with $k = 5$
SVC	C-Support Vector Classification with rbf kernel
NN	multi-layer Perceptron with 2 hidden layers, 10 neurons each, relu activation and cross entropy loss

Figure 1 shows the different datasets that were used to study the accuracy for different values of Λ.

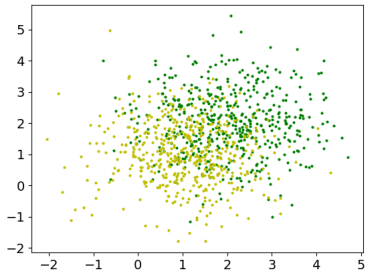
Figure 2 shows the accuracy measurements for the exhaustive grid exploration of Λ on the random classification dataset 1d. The corresponding



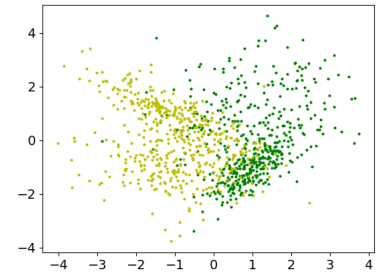
(a) Gaussian quantiles



(b) Interleaving half circles



(c) Isotropic Gaussian blobs



(d) Random dataset

Fig. 1: Artificial binary classification problems

pseudo-code is given in Algorithm 1. Before applying the Box-Cox transformation all datasets have been preprocessed with a range standardization between 1 and 2. This was done to show on the one hand exclusively the behavior of the Box-Cox transformation without influence of other effects and on the other hand the transformation needs positive data. Further the upper range bound ensures that the features do not explode when transformed with a larger Λ. Additionally, the results of the Box-Cox transformation were standard scaled before given to the classifiers.

One can observe that the different heatmaps are not similar and hence the Box-Cox transformation is dependent on the classifier itself. For example $\Lambda = [-4, -3]$ gives the best performance for the neural network classifier but it is almost the worst for the SVC classifier. Also $\Lambda = [2, 5]$ is the best for the Bayesian classifier but the rather bad for the KNN classifier. This suggest that the optimization of the Box-Cox transformation is not only dependent on the data but also on the classifier. This observation was also made by [2].

Further the heatmaps show multiple local maxima. This means that the optimization is non-convex. Similar observation can be made for the other datasets and the corresponding heatmaps are provided in appendix A.

Algorithm 1 2D Accuracy Gridexploration

D: dataset (X, Y)
$\mathcal{L}_1 \leftarrow \{-5, -4, ..., 4, 5\}$
$\mathcal{L}_2 \leftarrow \{-5, -4, ..., 4, 5\}$
$repetitions \leftarrow 5$
$kfolds \leftarrow 10$ ▷ number of folds in crossvalidation
C: classifier
A: matrix to store accuracies
for each item λ_1 in $\mathcal{L}_1$ **do**
  for each item λ_2 in $\mathcal{L}_2$ **do**
    $a \leftarrow 0$ ▷ current accuracy
    for $rep = 1$ to $repetitions$ **do**
      for I_{train}, I_{test} in cvpartition(D, kfold) **do**
        $X^* = \text{boxcox}(X, [\lambda_1, \lambda_2])$

        $X_{train} \leftarrow X^*[I_{train}]$ ▷ split data
        $Y_{train} \leftarrow Y[I_{train}]$
        $X_{test} \leftarrow X^*[I_{test}]$
        $Y_{test} \leftarrow Y[I_{test}]$

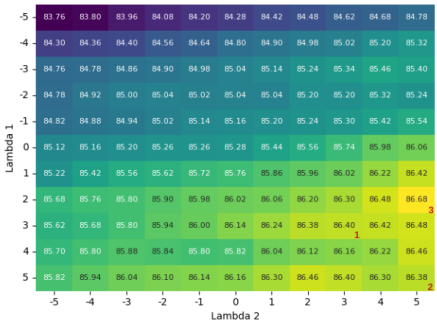
        $\text{Scaler} = \text{Standard\_Scaler}()$
        $X_{train} = \text{Scaler.fit\_transform}(X_{train})$ ▷ train model
        $C_t = \text{train}\,(C, X_{train}, Y_{train})$

        $X_{test} = \text{Scaler.transform}\ (X_{test})$ ▷ evaluate model
        $P = \text{predict}\,(C_t, X_{test})$
        $a \leftarrow a + \text{accuracy}\ (P, Y_{test})$
      end for
    end for
    $A[\lambda_1, \lambda_2] = \frac{a}{repetitions * kfolds}$
  end for
end for

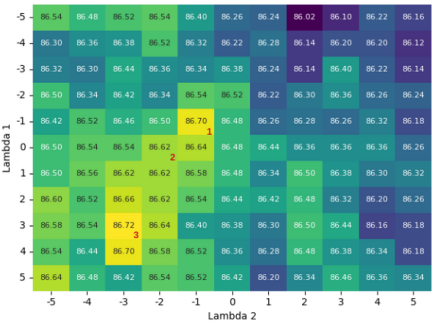
Finally one can see that the *full* optimization can give better results than the *spherical* and *diagonal* setting. The possible *spherical* configurations can be seen on the diagonal of the heatmap (e.g. $\Lambda \in \{[-5, -5], [-4, -4], ..., [5, 5]\}$. The diagonal can be illustrated by first fixing one direction $\lambda_i = 1$ and optimizing in the other direction and then vice versa. Possible optimal solutions for the *spherical*, *diagonal* and *full* optimization are indicated with corresponding numbers 1, 2 and 3. If there are multiple options for the optimal solution in one direction for the *diagonal* optimization, then the case that leads to the higher final optimization accuracy is used.

Table 2 summarizes the accuracy heatmaps for all four datasets in Figure 1. It shows the performance before applying the Box-Cox transformation and

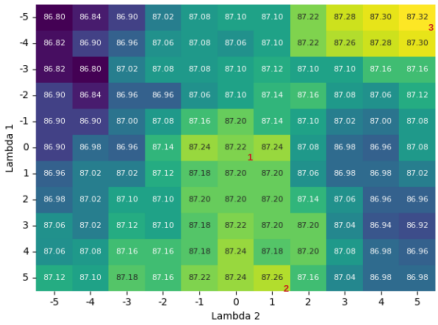
(a) Linear classificer



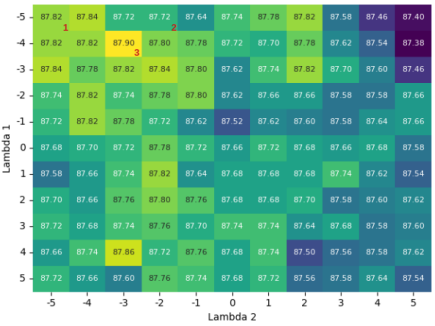
(b) Bayesian classifier



(c) KNN classifier



(d) SVC classifier



(e) NN classifier

Fig. 2: Accuracy heatmaps for random dataset. The numbers 1, 2 and 3 correspond to the optimal solution for the *spherical*, *diagonal* and *full* optimization. If there are multiple solutions then only one possibility is shown.

after applying the Box-Cox transformation with the best reported configuration of Λ. The numbers are rounded to the first decimal point. The accuracy before applying the Box-Cox transformation corresponds to a Box-Cox transformation with $\Lambda = (1, 1)$ because this does only shift the data by 1 in each direction and therefore does not influence the classification result.

Classifier	Accuracy before	Accuracy after	Improvement	Spherical	Diagonal
		Gaussian quantiles			
Linear	49.0	54.4	5.4	3.9	4.2
Bayesian	96.8	97.0	0.2	0.2	0.1
KNN	96.7	96.6	0.6	0.3	0.4
SVC	98.8	99.2	0.5	0.1	0.2
NN	98.9	99.1	0.2	0.0	0.2
		Interleaving half circles			
Linear	83.5	85.5	2.0	0.5	-0.3
Bayesian	87.4	88.3	0.9	0.0	0.8
KNN	100.0	100.0	0.0	0.0	0.0
SVC	99.7	99.7	0.1	0.0	0.1
NN	99.9	100.0	0.0	0.0	0.0
		Isotropic Gaussian blobs			
Linear	68.1	70.8	2.7	2.0	2.7
Bayesian	76.1	76.4	0.3	0.2	0.3
KNN	74.6	75.2	0.6	0.4	0.6
SVC	76.1	75.7	0.4	0.4	0.4
NN	76.0	76.4	0.5	0.4	0.3
		Random dataset			
Linear	77.3	80.5	3.2	1.8	-0.3
Bayesian	85.9	86.6	0.8	0.5	0.5
KNN	87.2	87.7	0.4	0.4	0.3
SVC	87.2	87.3	0.1	0.0	0.1
NN	87.7	87.9	0.2	0.1	0.0

Table 2: Accuracy of different classifiers before and after applying Box-Cox transformation

One can see that the linear classifier benefits the most from the Box-Cox transformation. Nevertheless, the other classifiers also benefit unless the classification result is almost perfect before applying the transformation (KNN and NN in the interleaving half circles dataset). Thus Box-Cox transformation can consistently improve the classification result.

Additionally one can see that *spherical* optimization mostly does not achieve the same improvements as the *full* optimization. This can be expected because the lower number of parameters. In contrast, *diagonal* optimization

can even result in worse accuracies. This can be seen for example for the linear classifier in the interleaving half circles dataset. Fixing one direction and optimizing for the other direction gives an increase in accuracy (fixing $\lambda_1 = 1$ leads to $\lambda_2 = -1$ with an improvement of 1.26% and fixing $\lambda_2 = 1$ leads to $\lambda_1 = 2$ with an improvement of 0.72 %). But then combining the independent results leads to $\Lambda = [2, -1]$ and a loss of accuracy of -0.3 %. This can get arbitrary bad because it is unknown in advance what the combination of the independent optimizations leads to.

4 Model and Optimization

The previous section showed that the *full* optimization leads to the best improvements. Further it was demonstrated that the optimization is dependent on the classifier. Therefore we propose a procedure for a classifier-dependent multi-dimensional non-convex optimization. First the general setup is described. Following a naive optimization is introduced. This was used as a baseline but suffers from the curse of dimensionality. Next a iterative optimization is described that solves the dimensionality problem. Subsequently various techniques to improve the iterative procedure are presented.

The general setup that was used with different optimization techniques is given by a training function and a predicting function. The training procedure is given in Algorithm 2. It requires the features, the corresponding class labels, a classifier and an optimization procedure for Λ. It first scales the data into the range $[1, 2]$ to ensure that the features are positive so that the Box-Cox transformation can be applied, and to ensure that the features do not explode at larger Λ. Then an optimization procedure is applied to find suitable values for Λ. As described in the previous section this is dependent on the classifier itself. Next the Box-Cox transformation is applied to the features with the optimized Λ. Then the data is standard scaled to help classifiers that depend on a distance measure. Finally the classifier is trained.

The predicting procedure is presented in Algorithm 3. It requires the features, Λ that was optimized during training, a fitted classifier, a fitted min-max scaler and a fitted standard scaler. First the method min-max scales the data, then applies the Box-Cox transformation with the given Λ, then uses standard scaling and finally predicts the labels with the given classifier.

The first optimization procedure is a gridsearch. This means that a set of possible values for every λ_i is specified. Then the optimization tries all combinations. This is an exhaustive search and runs in polynomial time $O(L^p)$ where L is the number of possible values and p is the number of features. Therefore gridsearch suffers from the curse of dimensionality. For example trying 10 values for 10 features already needs 10 billions evaluations. Therefore this gets quickly infeasible. Nevertheless it can be used as a baseline for lower dimensional datasets. Pseudo-code for this method for the 2-dimensional case is given in Optimization 1 and can directly be used as optimization for training in Algorithm2.

Algorithm 2 fit(X, Y, C, O)

X: features
Y: corresponding class labels
C: classifier to optimize for
optimization: procedure for optimizing Λ
M: min-max scaler into the range $[1, 2]$
S: standard scaler

$X_M \leftarrow M.fit\_transform(X)$ ▷ fit min-max scaler and apply it
$\Lambda \leftarrow$ optimization(X_M, Y, C) ▷ find optimized Λ
$X_B \leftarrow$ boxcox(X_M, Λ) ▷ apply Box-Cox transformation
$X_S \leftarrow S.fit\_transform(X_B)$ ▷ fit standard scaler and apply it
C.train (X_S, Y) ▷ train classifier

return Λ, C, M, S

Algorithm 3 predict(X, Λ, C, M, S)

X: features
Λ: optimized parameters of Box-Cox transformation
C: trained classifier
M: fitted min-max scaler
S: fitted standard scaler

$X_M \leftarrow M.transform(X)$ ▷ apply fitted min-max scaler
$X_B \leftarrow$ boxcox(X_M, Λ) ▷ apply Box-Cox transformation
$X_S \leftarrow S.transform(X_B)$ ▷ apply fitted standard scaler
$Y \leftarrow C$.predict (X_S) ▷ predict labels with trained classifier

return Y

To solve the dimensionality problem of gridsearch, we propose an iterative optimization. First an initial point $G \in \mathbb{R}^p$ for Λ is specified. Then starting from this point, all directions except for one gets fixed. The not fixed direction gets optimized with a $1-$dimensional gridsearch. Therefore a set of candidate values for the search needs to be defined. Comparing the possible values and selecting the one that gives the highest improvement leads to the optimization in the first direction. Then the next direction gets unfixed and all other directions get fixed. Again, the best value is selected with a 1-dimensional gridsearch. This procedure is repeated until all directions are optimized once. This will be referred as one *epoch*. After that, one can restart the same procedure, starting with the previously found optimized solution instead of the initial point G. Pseudocode for this iterative optimization is given in Optimization 2 and will be referred as *Basic*. The advantage of this method is that

Optimization 1 Gridsearch(X, Y, C)

X: features
Y: corresponding class labels
C: classifier to optimize for
S: standard scaler
Λ_{opt}: optimized Box-Cox parameters
$grid \leftarrow \{-5, -4, .., 4, 5\}$ ▷ candidate values for each direction
$A \leftarrow 0$ ▷ best accuracy obtained during search

for each λ_1 in $grid$ **do**
  for each λ_2 in $grid$ **do**
    $\Lambda_{tmp} \leftarrow [\lambda_1, \lambda_2]$
    $X_B \leftarrow \text{boxcox}(X, \Lambda_{tmp})$ ▷ apply Box-Cox transformation
    $X_S \leftarrow S.fit\_transform(X_B)$ ▷ fit standard scaler and apply it
    $C.\text{train}\,(X_S, Y)$ ▷ train classifier
    $P \leftarrow C.\text{predict}\,(X_S)$ ▷ evaluate classifier
    $A_{tmp} \leftarrow \text{accuracy}(P, Y)$ ▷ evaluate accuracy

    if $A_{tmp} > A$ **then** ▷ update Λ_{opt} if accuracy is improved
      $A \leftarrow A_{tmp}$
      $\Lambda_{opt} \leftarrow \Lambda_{tmp}$
    end if

  end for
end for
return Λ_{opt}

it scales linearly $O(epochs \cdot p \cdot gridsize)$ in the number of features p, where gridsize denotes the number of points used for the 1-dimensional gridsearch. This procedure has three hyperparamters which can influence the result (initial starting point G, number of epochs and the grid).

In Figure 3a there is an example why multiple epochs can be beneficial. Starting from initial point $G = [1, 1]$ and first optimizing vertically in λ_1 direction and then horizontally in λ_2 direction will give an optimal value of 2 for $\Lambda = [2, 2]$. If another epoch and thus another optimization in both directions is added, the global optimal solution 3 at $\Lambda = [3, 2]$ is found. Therefore multiple epochs can help to find a better optimization result.

Another useful improvement is to restart the optimization with another initial point. Figure 3b illustrates the problem. Starting at $G = [1, 1]$ and optimizing first vertically and then horizontally will result in $\Lambda_{opt} = [2, 1]$. This can not be further optimized with the given iterative method. Unfortunately there is a better solution at $\Lambda = [1, 2]$. If for example the optimization would have started at $G = [2, 2]$ the global optimal solution could be attained. Therefore it can be beneficial to restart the optimization procedure with multiple

Optimization 2 Basic(X, Y, C)

X: features
Y: corresponding class labels
C: classifier to optimize for
p: number of features/directions
S: standard scaler
G: initial starting point
$\Lambda_{opt} \leftarrow G$ ▷ optimized Box-Cox parameter
$grid \leftarrow \{-5, -4, .., 4, 5\}$ ▷ candidate values for each direction
$epochs \leftarrow e$ ▷ number of epochs
$A \leftarrow 0$ ▷ best accuracy obtained during search

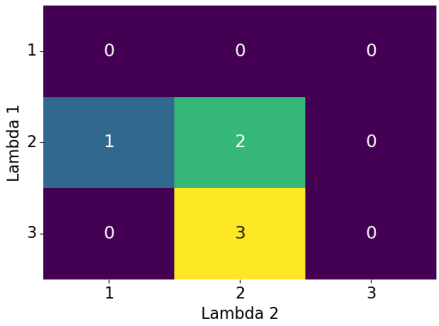
for $epoch = 1$ to $epochs$ **do**
  for $dir = 1$ to p **do**
    $\Lambda_{tmp} \leftarrow \Lambda_{opt}$
    for each λ_i in $grid$ **do**
      $\Lambda_{tmp}[dir] \leftarrow \lambda_i$ ▷ change one direction
      $X_B \leftarrow \text{boxcox}(X, \Lambda_{tmp})$ ▷ apply Box-Cox transformation
      $X_S \leftarrow S.fit\_transform(X_B)$ ▷ fit standard scaler and apply it
      $C.\text{train}\,(X_S, Y)$ ▷ train classifier
      $P \leftarrow C.\text{predict}\,(X_S)$ ▷ evaluate classifier
      $A_{tmp} \leftarrow \text{accuracy}(P, Y)$ ▷ evaluate accuracy

      if $A_{tmp} > A$ **then** ▷ update Λ_{opt} if accuracy is improved
        $A \leftarrow A_{tmp}$
        $\Lambda_{opt} \leftarrow \Lambda_{tmp}$
      end if

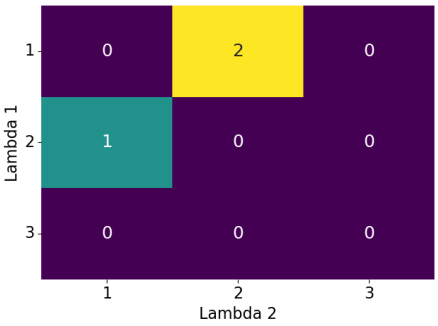
    end for
  end for
end for
return Λ_{opt}

initial points. Corresponding modifications to the *Basic* optimization can be found in Optimization 2.1. It introduces *shift_epoch* as a new hyperparameter that determines after how many *epochs* a new starting point G is generated.

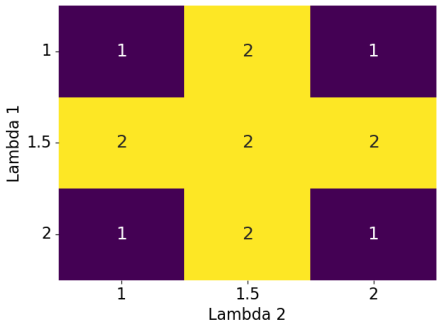
The previous problem could also be solved by changing the order of directions of optimization. So far the directions got optimized numerically, meaning that first λ_1 is optimized, then λ_2 and so on. Starting in Figure 3b again at initial point $G = [1, 1]$, instead of first optimizing in vertical direction, optimization is done first in horizontal direction. This directly finds the global solution. Hence, shuffling the order of directions for optimization can also be helpful. The corresponding changes to *Basic* can be found in Optimization 2.2.



(a) multiple epochs



(b) multiple starting points and shuffling optimization order



(c) finer grid

Fig. 3: Motivations for improvements to iterative method

Again, there is a new hyperparameter *shuffle_epoch* that determines after how many *epochs* the optimization order gets shuffled.

Lastly, it might be possible to find a better solution if the gridsearch is denser. Figure 3c demonstrates the problem. If the grid only uses integer values then it is impossible to find one of the global optimal solutions = 2. Hence the grid should be refined to 0.5 increments. Unfortunately this doubles the computational demand. Another refinement may further improve the result but increases the computational demand even more. One solution to circumvent the increasing computational costs, is to use local refinement. This means that the grid gets locally denser but also smaller. *Basic* uses the same global grid for every 1-dimensional gridsearch (e.g. $\{-5, -4, ..., 4, 5\}$). To get a finer grid but with the same number of points, the grid needs to be attached locally to the current Λ_{tmp}. Since the number of grid points should remain the same and the grid gets denser, the grid spans a smaller range of values. For example starting with the grid $\{-5, -4, ..., 4, 5\}$ and then doubling the resolution leads to the following grid $\{-2.5, -2, ..., 2, 2.5\}$. Both of them have the same number of points. Instead of testing globally if any $\lambda_i \in \{-5, -4, ..., 4, 5\}$ improves the

Optimization 2.1 Shift(X, Y, C)

```
...                                                    ▷ same as Basic
shift_epoch ← s                  ▷ number of epochs until new starting point
shift ← False                    ▷ Boolean flag to indicate a new starting point

for epoch = 1 to epochs do

    if epoch mod shift_epoch == 0 then
        G ← generate_initial_point()
        shift ← True
    end if

    if shift then
        Λ_tmp ← G
    else
        Λ_tmp ← Λ_opt
    end if

    for dir = 1 to p do
        for each λ_i in grid do
            ...                                            ▷ same as Basic
            if A_tmp > A then          ▷ update Λ_opt if accuracy is improved
                A ← A_tmp
                Λ_opt ← Λ_tmp
                shift ← False
            end if

        end for
    end for
end for
return Λ_opt
```

result, the current optimal solution in this direction is used and then the refined grid is attached to it. Therefore it is tested if any $\lambda_i \in \{\lambda_{tmp,i} - 2.5, \lambda_{tmp} - 2, ..., \lambda_{tmp,i}+2, \lambda_{tmp,i}+2.5, \}$ improves the accuracy. In order to take advantage of both global and local optimization, the global search is used at the beginning of the optimization to capture the full search domain. After some epochs a local refinement is used to get a finer search space. With this modification the computational cost stays the same. Additionally it allows for more and finer candidate values that can result in an improvement. Incorporating this method to *Basic* is shown in Optimization 2.3. As before, an additional hyperparameter *finer_epoch* is introduced to specify after how many *epochs* the grid gets refined.

Optimization 2.2 Shuffle(X, Y, C)

```
...                                                    ▷ same as Basic
dir_order ← [1, 2, ..., p]                  ▷ order of directions for optimization
shuffle_epoch ← h                             ▷ number of epochs until the order
                                                    ▷ of direction gets shuffled

for epoch = 1 to epochs do

    if epoch mod shuffle_epoch == 0 then
        shuffle(dir_order)
    end if

    for each dir in dir_order do
        ...                                            ▷ same as Basic
    end for
end for
return Λ_opt
```

Optimization 2.3 Finer(X, Y, C)

```
...                                                    ▷ same as Basic
finer ← 0.5                                            ▷ refinement of grid
finer_epoch ← f                    ▷ number of epochs until the grid gets finer
global = 1                           ▷ use global gridsearch at the beginning

for epoch = 1 to epochs do

    if epoch mod finer_epoch == 0 then
        global = 0
        grid ← finer * grid          ▷ element-wise scale each element in grid
    end if

    for dir = 1 to p do
        Λ_tmp ← Λ_opt
        candidates ← grid + (1 − global) * Λ_opt[dir]
        for each λ_i in candidates do
            ...                                        ▷ same as Basic
        end for
    end for
end for
return Λ_opt
```

5 Results

Following the proposed optimization procedure is applied to different real word datasets. The examined classifiers are given in Table 1. All results are measured with 10-fold stratified crossvalidation and 5 repetitions. To test the proposed method various settings for the hyperparameters were used. The setup is given in Table 3. Optimization in one direction was done evenly spaced over the interval $[-5, 5]$ and gridsize corresponds to the number of gridpoints (e.g. gridsize of 11 gives the set $\{-5, -4, ..., 4, 5\}$ as candidate values). *Basic* is just the iterative optimization without any further improvements. *Shift*, *Shuffle* and *Finer* show exclusively the influence of restarting the optimization with a new starting point, changing the order of directions or refining the optimization grid. *Combined 1* and *Combined 2* demonstrates how these improvements to the *Basic* optimization behave in combination. The initial starting point $G \in \mathbb{R}^p$ for the iterative optimization was chosen in each direction as the maximum likelihood estimate.

Name	gridsize	epochs	shift_epoch	shuffle_epoch	finer_epoch
Basic	11	4	4	4	4
Shift	11	8	4	8	8
Shuffle	11	8	8	2	8
Finer 1	11	8	8	8	4
Combined 1	11	16	8	2	4
Combined 2	21	16	8	2	4

Table 3: Hyperparameter setting

5.1 Sonar Dataset

The sonar dataset has 207 samples and 60 features. The labels are binary and indicate whether the sonar signal was reflected by a rock or metal. The results from the repeated crossvalidation is given in Table 4.

There is an improvement in accuracy for all classifier except the neural network. Especially the Bayesian classifier improved on average by 7.9%. In contrast the neural network decreases by -0.4% on average. The influence of the changes to the basic iterative Optimization 2 with *Shift*, *Shuffle* and *Finer* can be seen well for the linear and KNN classifier. *Shift* does restart the optimization with a new initial point and it seems to decrease the accuracy for the linear classifier but slightly increase it for KNN. In contrast shuffling the order of the directions during optimization did not result in an advantage for a classifier. Refining the grid after some epochs also did not provide an increase in accuracy compared to the basic optimization. Combining these methods into one optimization can sometimes decrease the performance (linear and NN) and sometimes increase the performance (Bayesian). Interestingly, the

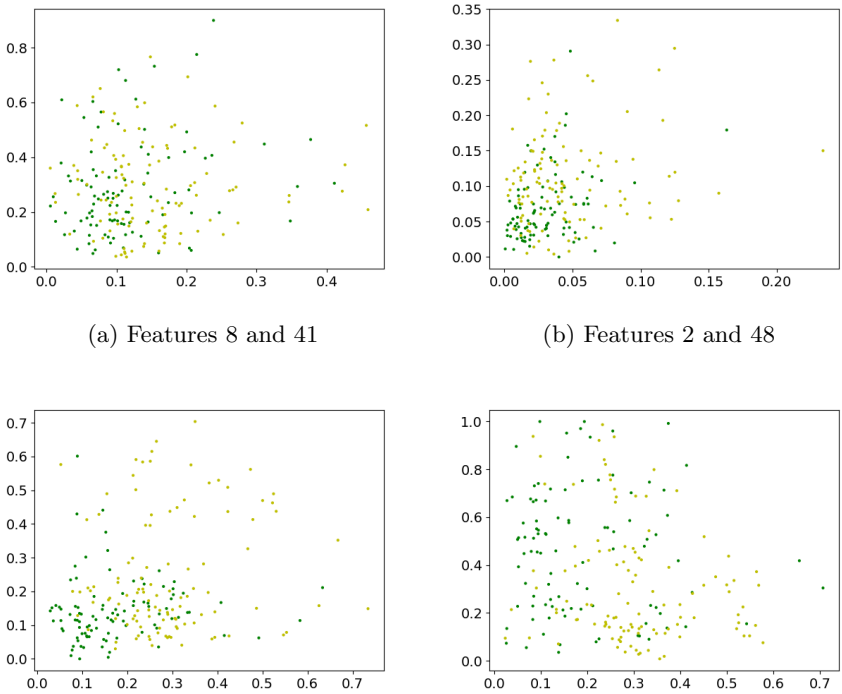
	Linear	KNN	Bayesian	SVC	NN
Base accurcay	75.195	81.343	67.7	84.052	84.024
Basic	1.162	1.824	7.919	2.286	-0.386
Shift	0.981	1.829	7.919	2.286	-0.386
Shuffle	1.162	1.824	7.919	2.286	-0.386
Finer	0.876	1.824	7.919	2.286	-0.386
Combined 1	1.257	1.452	7.538	2.286	-0.386
Combined 2	0.095	1.824	7.910	2.286	-0.486

Table 4: Improvement in accuracy for different optimization settings on the sonar dataset

influence of the combined optimization is better for the linear classifier and the neural network when the grid for Λ is sparser. The opposite can be observed for the KNN and Bayesian classifier. Lastly, all hyperparameter settings achieved exact the same improvement for the SVC.

Additionally, a 2-dimensional feature study was performed. Two 2-dimensional datasets 4a and 4b were created by extracting two random features from the sonar dataset. Additionally, with a chi-square test the ranks of the features were calculated. Then a dataset with the two highest ranking features 4c and a dataset 4d with the third and fourth highest ranking features was build. Further the performance of a gridsearch was measured and can serve as baseline. The datasets are shown in Figure 4 and the results are given in Table 5.

The iterative method could deliver an improvement of the base accuracy in 15 out of 20 cases for at least one hyperparameter setting. Only the accuracy for the KNN classifier for features $[8, 41]$ and $[11, 45]$, Bayesian classifier for features $[12, 36]$, SVC classifier for features $[8, 41]$ and the neural network for features $[11, 45]$ could not be improved. Using the hyperparameter setting that resulted in the lowest loss for each case, the highest decrease in accuracy is only -1.543% (Bayesian classifier for features $[12, 36]$). In contrast the best hyperparameter setting could achieve a gain of 12.824% (linear classifier for features $[12, 36]$). Comparing the proposed iterative method to gridsearch, the iterative optimization could achieve better results for all classifiers for features $[8, 41]$ for at least one hyperparameter setting. Further it provided gains for the linear, KNN and neural network for features $[2, 48]$, Bayesian, SVC and neural network for features $[11, 47]$ and linear, KNN and neural network for features $[12, 36]$. Hence in 14 out of 20 cases the iterative optimization resulted in a better performance than gridsearch. The gain in accuracy for the linear classifier are always positive. This holds also for the Bayesian classifier except for features $[12, 36]$. The KNN classifier fluctuates around 0. Sometimes the iterative method can not achieve any improvement for all tested hyperparameters setting (features $[8, 41]$ and $[11, 45]$) and sometimes it is able to improve the result. For the SVC there is always at least one hyperparameter choice that leads to an improvement except for features $[8, 41]$. The same can be observed



(a) Features 8 and 41

(b) Features 2 and 48

(c) Features 11 and 45

(d) Features 12 and 36

Fig. 4: 2-D datasets extracted from sonar dataset

for the neural network except for features [11, 45]. Nevertheless, there it results in a smaller loss of accuracy than gridsearch for the *Finer* and *Combined 2* case. For all classifiers and datasets there is often a set of hyperparameters that improves the result of the *Basic* optimization. Additionally, the influence of *Shift* and *Finer* compared to *Basic* is sometimes positive and sometimes negative. This can vary between classifiers applied to the same dataset as for example for features [8, 41] *Finer* increases the accuracy for the linear classifier but decreases the accuracy for the KNN classifier. Further, it can vary between datasets but the same classifier as for example the performance of the linear classifier is increased for *Shift* for features [8, 41] but decreased for features [2, 48]. The same observation can be made for *Combined 1* and *Combined 2*. *Shuffle* does not influence the accuracy compared to *Basic*.

5.2 Breast Cancer Dataset

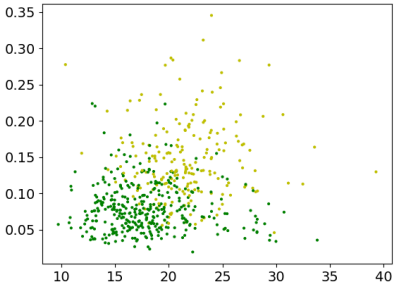
This dataset consists of 569 samples and 30 features. It is a binary classification problem that distinguishes between benign and malignant fine needle aspirate

(FNA) samples. The influence of the iterative method on the accuracy is given in Table 6.

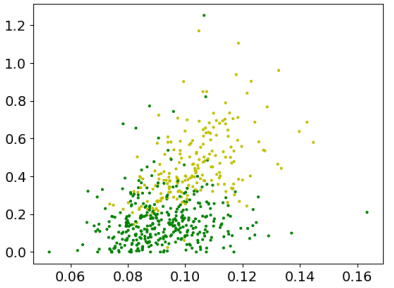
One can make similar observation as for the sonar dataset. There is an improvement for all classifiers except the neural network. Again the Bayesian classifier improves the most and the influence of the different hyperparameter setting on the SVC is almost static. In comparison to *Basic*, *Shift* is beneficial for the linear classifier and the neural network but harms the Bayesian, SVC and KNN classifier. Further, extending the *Basic* optimization with *Shuffle* has a small negative effect on the Bayesian classifier but does not change the accuracies of the other classifiers. *Finer* can improve the result for the linear classifier but does not change it for the other classifiers. *Combined 1* can consistently get better results than the *Basic* optimization except for the Bayesian classifier. In contrast, *Combined 2* can only improve the neural network.

Again, a 2-dimensional feature study was performed. Two 2-dimensional datasets 5a and 5b were created by randomly selecting two features from the breast cancer dataset. Also, a dataset 5c with the two highest ranking features and a dataset 5d with the third and fourth highest ranking features was created by using the ranks of a chi-square test. The datasets are given in Figure 5. The results of the iterative method are shown in Table 7. Additionally, a gridsearch was executed to get a baseline.

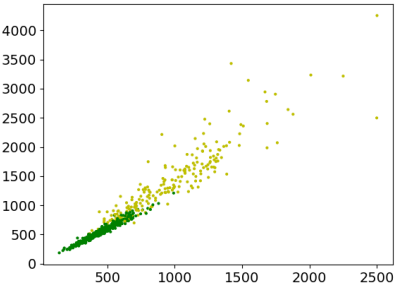
The iterative method resulted in an improvement of the base accurcay in 13 out of 20 cases for at least one hyperparameter setting. The highest loss of the best hyperparameter setting is -0.387%. It was realized by the neural network for features $[14, 23]$. In contrast the highest gain of 9.412% was achieved by the linear classifier for features $[5, 27]$. Compared to gridsearch, the iterative method is able to get the same or better performance in 13 out of 20 cases for at least one hyperparameter setting. The linear classifier always benefits from the proposed optimization. Bayesian classifier also could achieve consistent improvements except for features $[4, 24]$. The KNN classifier performance improved only in half of the cases (features $[2, 6]$ and $[5, 27]$) as did that of the neural network (features $[4, 24]$ and $[2, 6]$). As already observed in the experiment with the sonar dataset, *Shift* and *Finer* can both increase and decrease the performance compared to the *Basic* optimization. The influence varies from classifier to classifier and from dataset to dataset. For example for features $[5, 27]$ the linear classifier does not benefit from the *Shift* but the Bayesian classifier does. In contrast, the linear classifier for features $[2, 6]$ has experienced an increase in accuracy. *Shuffle* did not further influence the performance of the *Basic* method. Also, *Combined 1* and *Combined 2* have a varying influence depending on the classifier and the dataset.



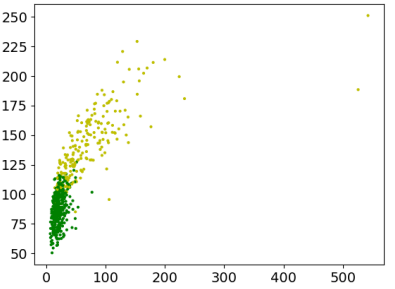
(a) Features 2 and 6



(b) Features 5 and 27



(c) Features 4 and 24



(d) Features 14 and 23

Fig. 5: 2-D datasets extracted from breast cancer dataset

	Linear	KNN	Bayesian	SVC	NN
Features 8 and 41					
Base accurcay	53.576	54.09	57.39	62.224	63.267
Gridsearch	7.081	-0.924	3.476	-1.138	-1.243
Basic	7.595	-1.195	3.481	-0.195	0.100
Shift	7.695	-0.529	3.681	-1.148	-0.090
Shuffle	7.595	-1.195	3.481	-0.195	0.100
Finer	8.657	-1.200	3.386	-0.386	-0.376
Combined 1	9.314	-2.252	3.767	-0.186	-0.376
Combined 2	6.638	-1.005	4.457	-0.181	-0.476
Features 2 and 48					
Base accurcay	55.948	64.862	62.59	66.538	66.719
Gridsearch	11.562	0.090	5.224	1.271	-0.086
Basic	12.638	-0.014	4.633	0.305	0.105
Shift	12.167	0.371	4.633	0.305	0.105
Shuffle	12.638	-0.014	4.633	0.305	0.105
Finer	12.724	0.567	4.448	0.800	0.105
Combined 1	12.443	0.376	4.638	0.605	0.200
Combined 2	11.762	0.567	4.448	0.505	-0.276
Features 11 and 45					
Base accurcay	63.995	71.371	64.805	73.486	72.995
Gridsearch	9.495	-0.186	8.886	-0.095	-1.162
Basic	8.838	-0.957	9.462	0.090	-1.257
Shift	8.933	-0.957	9.367	-0.100	-1.638
Shuffle	8.838	-0.957	9.462	0.090	-1.257
Finer	9.124	-0.567	9.562	0.190	-1.152
Combined 1	8.738	-0.567	9.562	0.095	-1.724
Combined 2	9.314	-0.467	9.948	0.190	-1.148
Features 12 and 36					
Base accurcay	59.51	70.443	70.705	68.876	69.824
Gridsearch	12.267	0.681	-0.695	1.833	0.386
Basic	12.133	1.733	-1.543	0.767	1.062
Shift	11.938	1.733	-1.543	0.762	0.881
Shuffle	12.133	1.733	-1.543	0.767	1.062
Finer	12.443	1.924	-1.929	0.771	0.686
Combined 1	12.824	2.019	-2.124	0.867	0.971
Combined 2	12.748	2.010	-1.824	0.867	1.648

Table 5: Improvement in accuracy for different optimization settings on 2-dimensional subsets of the sonar dataset

	Linear	KNN	Bayesian	SVC	NN
Base accurcay	96.487	96.838	93.289	97.539	98.103
Basic	0.175	0.245	1.371	0.211	-0.598
Shift	0.281	0.139	1.230	0.175	-0.528
Shuffle	0.175	0.245	1.336	0.211	-0.598
Finer	0.316	0.245	1.371	0.211	-0.598
Combined 1	0.351	0.245	1.125	0.211	-0.528
Combined 2	-0.106	0.210	1.336	0.211	-0.563

Table 6: Improvement in accuracy for different optimization settings on the breast cancer dataset

	Linear	KNN	Bayesian	SVC	NN
Features 2 and 6					
Base accurcay	75.61	79.615	81.622	82.678	83.66
Gridsearch	7.064	0.176	1.019	0.736	-0.175
Basic	6.187	0.034	0.635	0.455	-0.140
Shift	6.293	0.034	0.845	0.490	-0.175
Shuffle	6.187	0.034	0.635	0.455	-0.140
Finer	6.222	0.068	0.635	0.490	-0.105
Combined 1	6.117	0.141	0.705	0.666	-0.175
Combined 2	6.362	0.210	0.669	0.596	0.036
Features 5 and 27					
Base accurcay	76.007	84.538	84.677	85.736	86.754
Gridsearch	8.566	0.072	-0.034	0.138	-0.175
Basic	8.919	0.071	-0.034	0.383	-0.317
Shift	8.779	-0.034	0.002	0.384	-0.422
Shuffle	8.919	0.071	-0.034	0.383	-0.317
Finer	9.200	0.036	-0.174	0.489	-0.457
Combined 1	9.095	0.072	-0.315	0.419	-0.457
Combined 2	9.412	0.177	0.142	0.383	-0.457
Features 4 and 24					
Base accurcay	90.468	92.864	90.893	92.439	92.829
Gridsearch	1.553	-0.211	0.140	-0.281	0.212
Basic	1.694	-0.140	-0.246	-0.175	0.177
Shift	1.870	-0.140	-0.387	-0.246	0.212
Shuffle	1.694	-0.140	-0.246	-0.175	0.177
Finer	1.519	-0.246	-0.246	-0.175	0.387
Combined 1	1.519	-0.175	-0.386	-0.211	0.352
Combined 2	1.482	-0.422	-0.457	-0.211	0.211
Features 14 and 23					
Base accurcay	88.435	90.439	90.966	92.407	92.057
Gridsearch	3.620	-0.315	0.385	0.528	-0.773
Basic	3.409	-0.386	0.349	-0.350	-0.387
Shift	3.620	-0.421	0.419	-0.421	-0.703
Shuffle	3.409	-0.386	0.349	-0.350	-0.387
Finer	3.516	-0.352	0.384	-0.350	-0.387
Combined 1	3.481	-0.352	0.419	-0.421	-0.423
Combined 2	3.125	-0.387	0.385	-0.351	-0.457

Table 7: Improvement in accuracy for different optimization settings on 2-dimensional subsets of the breast cancer dataset

6 Discussion

With a grid exploration we have shown that the Box-Cox transformation is able to consistently improve the accuracy. This behavior was also observed by [2]. Also according to their results, we have demonstrated that the optimization depends on the classifier. Further, we have observed that a *full* optimization leads to higher improvements. Therefore a suitable optimization was introduced. This method was further improved to be able to handle different problems during optimization. For two real world datasets we have demonstrated that the proposed procedure is able to achieve improvements in accuracy. Furthermore, it could be shown that this optimization is superior to gridsearch in the majority of cases. We suspect that the iterative procedure introduces some implicit regularization. Gridsearch is likely to overfit to the training data, whereas the iterative method might not be able to find the global solution on the training set and hence suffers less from overfitting. Additionally, the proposed optimization scales linearly even with the ability to support finer grids. The real world dataset studies have shown that the hyperparameter setting is dependent on the data itself and the classifier. Further, restarting the optimization with multiple starting points and refining the grid were able to influence the result. Shuffling the optimization order did not have a meaningful impact.

As future work, an extensive application of the method to various datasets could be used to test the abilities of the optimization. Furthermore, the influence of the hyperparameters could be analyzed. In addition, the optimization could possibly be improved by, for instance, replacing the 1-dimensional grid search with another 1-dimensional optimization. Although the Box-Cox transformation has been shown to increase accuracy, it remains unclear whether it is also able to push performance beyond the state-of-the-art. Finally, the framework was designed with a general train-predict functionality that is often used in machine learning. Therefore our method could also be applied to other tasks such as regression.

References

[1] Liang, Y., Hussain, A., Abbott, D., Menon, C., Ward, R., Elgendi, M.: Impact of data transformation: An ecg heartbeat classification approach. Frontiers in Digital Health **2**, 53 (2020). https://doi.org/10.3389/fdgth.2020.610956

[2] Bicego, M., Baldo, S.: Properties of the box–cox transformation for pattern classification. Neurocomputing **218**, 390–400 (2016). https://doi.org/10.1016/j.neucom.2016.08.081

[3] Box, G.E.P., Cox, D.R.: An analysis of transformations. Journal of the Royal Statistical Society. Series B (Methodological) **26**(2), 211–252 (1964)

[4] Sweeting, T.J.: On the choice of prior distribution for the Box-Cox transformed linear model. Biometrika **71**(1), 127–134 (1984) https://academic.oup.com/biomet/article-pdf/71/1/127/1000994/71-1-127.pdf. https://doi.org/10.1093/biomet/71.1.127

[5] Carroll, R.J., Ruppert, D.: Transformations in regression: A robust analysis. Technometrics **27**(1), 1–12 (1985) https://www.tandfonline.com/doi/pdf/10.1080/00401706.1985.10488007. https://doi.org/10.1080/00401706.1985.10488007

[6] Lawrance, A.J.: Regression transformation diagnostics using local influence. Journal of the American Statistical Association **83**(404), 1067–1072 (1988) https://www.tandfonline.com/doi/pdf/10.1080/01621459.1988.10478702. https://doi.org/10.1080/01621459.1988.10478702

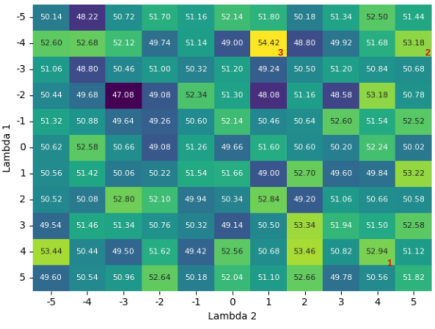
[7] Kim, C., Storer, B.E., Jeong, M.: Note on box-cox transformation diagnostics. Technometrics **38**(2), 178–180 (1996) https://doi.org/10.1080/00401706.1996.10484463. https://doi.org/10.1080/00401706.1996.10484463

[8] Vélez, J.I., Correa, J.C., Marmolejo-Ramos, F.: A new approach to the box–cox transformation. Frontiers in Applied Mathematics and Statistics **1**, 12 (2015). https://doi.org/10.3389/fams.2015.00012

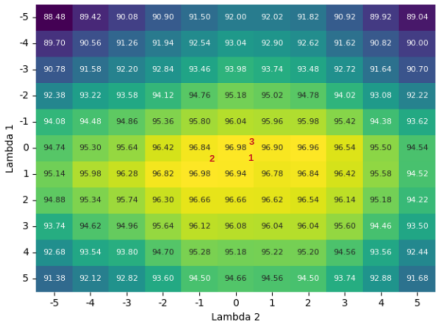
[9] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)

Appendix A Additional grid exploration heatmaps for artificial datasets

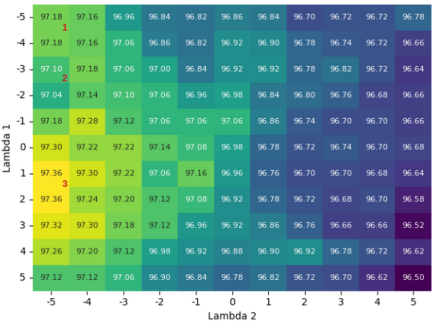
Following the heatmaps of the grid exploration for the *Gaussian quantiles* 1a, *interleaving half circles* 1b and *isotropic Gaussian blobs* 1c. The optimal solution for the *spherical*, *diagonal* and *full* optimization is indicated with corresponding numbers 1, 2 and 3. If there are multiple solutions then only one possibility is given.



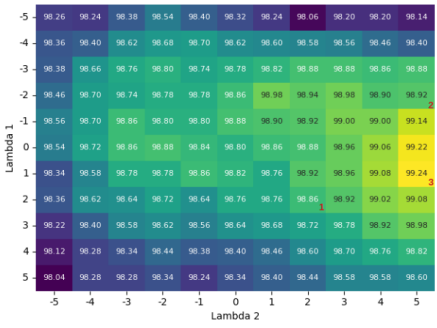
(a) Linear classificer



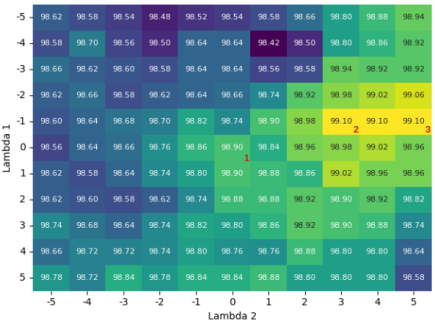
(b) Bayesian classifier



(c) KNN classifier

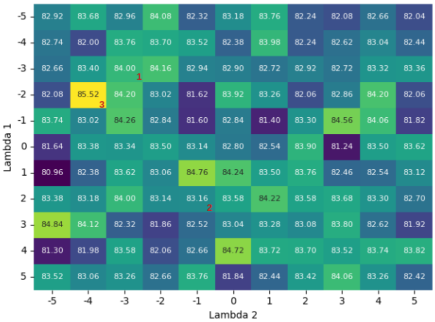


(d) SVC classifier

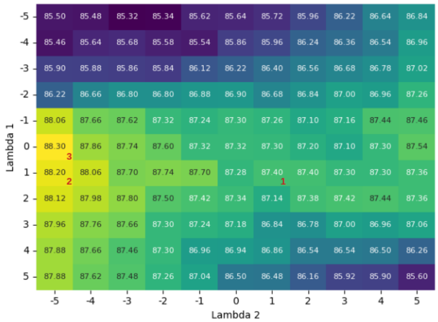


(e) NN classifier

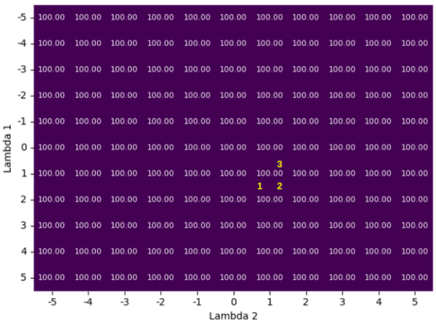
Fig. A1: Accuracy heatmaps for Gaussian quantiles



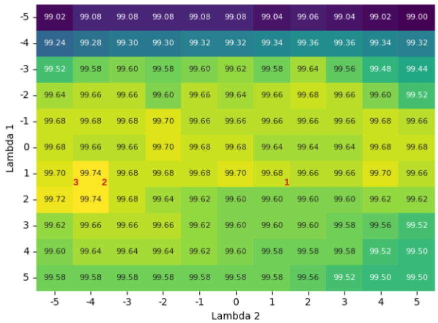
(a) Linear classificer



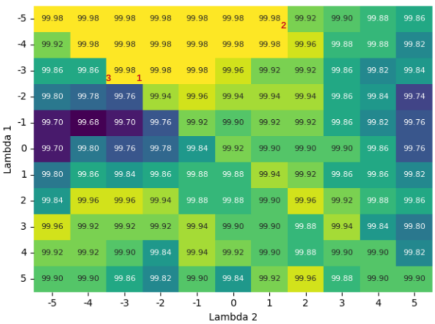
(b) Bayesian classifier



(c) KNN classifier

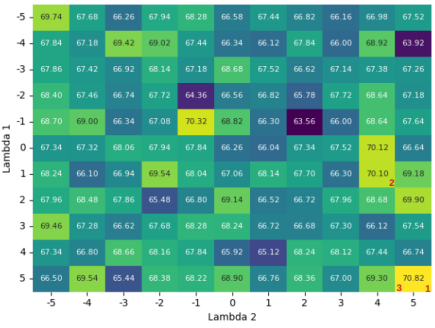


(d) SVC classifier

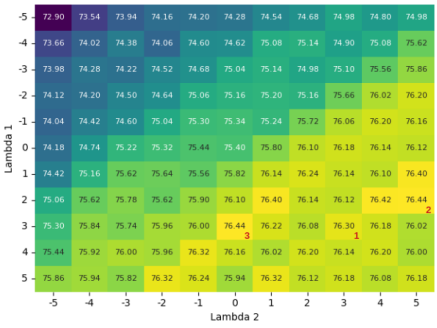


(e) NN classifier

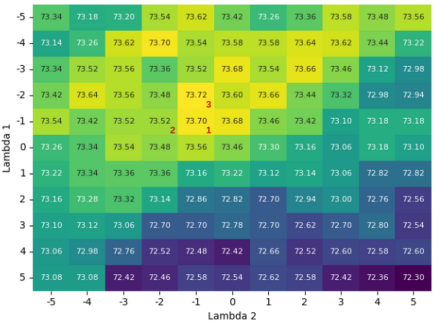
Fig. A2: Accuracy heatmaps for interleaving half circles



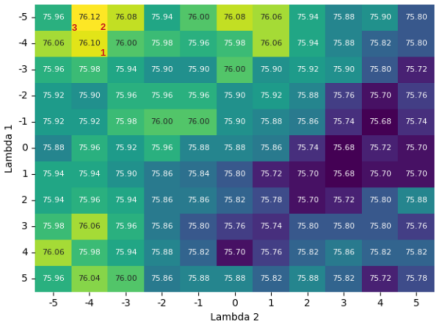
(a) Linear classificer



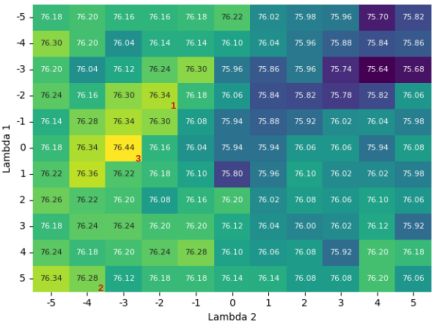
(b) Bayesian classifier



(c) KNN classifier



(d) SVC classifier



(e) NN classifier

Fig. A3: Accuracy heatmaps for isotropic Gaussian blobs