DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Informatics

Integration of Prior Knowledge for Regression and Classification with Sparse Grids

Lukas Krenz

DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Informatics

Integration of Prior Knowledge for Regression and Classification with Sparse Grids

Integration von Vorwissen für Regression und Klassifikation mit dünnen Gittern

Author:	Lukas Krenz
Supervisor:	Univ.-Prof. Dr. Hans-Joachim Bungartz
Advisor:	Valeriy Khakhutskyy, M.Sc.
Submission Date:	September 15, 2016

I confirm that this bachelor's thesis in informatics is my own work and I have documented all sources and material used.

Munich, September , 2016 Lukas Krenz

Acknowledgments

'Yes, it's pretty dark in here,' said the professor. 'But there are miracles that can only occur in the dark.'
'Where's the exit, please?'
'You don't want to try out my oracle?'
'Er, to be honest: no. I'm not feeling too good and I've had it up to here with this fairground hocus-pocus.'
Nightingale's eyes flashed and something inside his head crackled ominously. 'Hocus-pocus?' he hissed. 'This isn't hocus-pocus, it's scientific exactitude!'

(Walter Moers - Rumo and His Miraculous Adventures)

Darkness is the natural space for even small scientific works. But next to darkness, just around the corner, lingers the light of social interaction.

I want to thank Prof. Bungartz for the chance to work on this exciting topic. Special thanks goes to my advisor Valeriy, who managed to steer me in the right direction and always had time to answer any of my questions. His valuable feedback not only improved the quality of this thesis, but also improved my knowledge of scientific writing.

My gratitude extends to my friends and family, who always managed to motivate me. A honorary mention goes to my friend William, who did the (mostly) thankless task of proof-reading this thesis. Of course, only I am to blame for all errors left.

Without any of the mentioned people, this thesis might have been impossible. In any case, it would have been a far less enjoyable experience.

Abstract

This thesis discusses different ways of imposing prior knowledge about datasets on the sparse grid model for supervised learning. We introduce a Tikhonov regularization method that uses information about the smoothness of the function we want to approximate. We also present the sparsity-inducing penalties lasso, elastic net, and group lasso. The different regularization approaches are compared with the standard ridge regularization. Because some regularization penalties are not differentiable, we discuss the fast iterative shrinkage-thresholding algorithm and show how it can be used in conjunction with our added regularization methods. Furthermore, we modify the grid generation procedure. The first discussed method is generalized sparse grids, which allows us to control the granularity of the grid. The second method is interaction-term aware sparse grids, which are used to construct smaller and more efficient grids for image recognition problems. All methods were implemented with the *SG++* library and showed promising results for both artificial and real-world datasets.

Zusammenfassung

Diese Bachelorarbeit diskutiert verschiedene Wege, vorheriges Wissen über Datensätze auf das Lernen mit dünnen Gittern zu übertragen. Wir führen die Tikhonov Regularisierung ein, die Informationen über die Glattheit der Funktion, die wir approximieren wollen, benutzt. Wir präsentieren außerdem die Regularisierungsmethoden Lasso, Elastic Net und Group Lasso, die alle zu dünnbesetzten Lösungsvektoren führen. Die verschiedenen Methoden werden mit der klassischen Ridge-Regularisierung verglichen. Weil einige Funktionale nicht differenzierbar sind, diskutieren wir den "Fast Iterative Shrinkage Thresholding" Algorithmus, und zeigen, wie er mit den neu hinzugefügten Regularisierungsmethoden benutzt werden kann. Desweiteren modifizieren wir den Gittergenerierungsalgorithmus. Die erste Methode heißt verallgemeinerte dünne Gitter und erlaubt uns, die Granularität der Gitter zu verändern. Die zweite diskutierte Methode sind Gitter, die Wissen über Interaktionsterme benutzen, was nützlich für bildverstehende Verfahren ist. Alle Methoden wurden mit Hilfe der *SG++* Bibliothek implementiert und zeigten vielversprechende Ergebnisse für künstliche und echte Datensätze.

Contents

CHAPTER 1

Introduction

Machine Learning

The importance of machine learning and statistics cannot be overstated. They both supply techniques that are useful for data analysis and forecasting that power many scientific achievements. Data is becoming more and more important for companies as well, for example to predict future business outcomes, to generate more efficient advertisement, and so on. Many usage scenarios have one thing in common: They are complicated problems with a vast supply of data.

Sparse Grids

Throughout this thesis we will discuss supervised learning methods that are based on the sparse grid methodology. Sparse grids is a family of closely-related discretization techniques that originates from numerical partial differential equations. It approximates functions with many simple basis functions. In comparison to a full grid, sparse grids represent a rather economical approach. They scale far better for higher dimensions, which is even more critical for machine learning than it is for other numerical problems. This is because while a ten-dimensional differential equation might be called high-dimensional, a ten-dimensional dataset is merely low-dimensional. In other words, sparse grids break the curse of dimensionality to some degree [BG04]. For some problems, this is not enough. Very-high dimensional problems still suffer from the same malediction. Fortunately, modern sparse grids research is concerned with economic solutions for challenging problems.

While techniques such as adaptive grids [**spatAdaptGrid**] do not manage to turn the malediction into a benediction, they still mitigate some of the torment. Sparse grids are ideally suited for complicated problems, as they scale well with larger data and because they can solve arbitrarily complex problems through a combination of their solid theoretical foundation and the continuous development of new refinement strategies. This eclectic approach leads to the effect that while they might not be the best strategy for a specific problem, they are a very robust strategy, which can be used for a plethora of diverse tasks.

Sparse grids have a long history consisting of a vast array of theoretical arguments. While they are thoroughly studied from a functional theory background, discussions about their statistical interpretation are rather new [**sparse-parsimony**]. In this thesis we will focus on their statistical properties by trying to fit them into an intuitive and

simple statistical framework.

Prior Knowledge

The goal of this thesis is to introduce and evaluate different ways of integrating prior knowledge into our data mining procedure. In this context, prior information corresponds to something akin to the standard Bayesian prior. This prior represents assumptions, which can be either drawn from the dataset or from inherent properties of the sparse grid methods. Following the dogma that more data is always better than fewer data, more prior knowledge can always be helpful. This is only natural, because it is always easier to create useful models by starting with more (correct) assumptions about the true model.

The thesis starts with a chapter explaining the needed preliminary mathematical techniques and especially the fundamentals of the sparse grid framework. We then look at different ways of including prior information into this learning method. We make the following contributions:

- In **??** we evaluate different regularization methods that help us to impose smoothness constraints on our model and allow us to simplify our models. We begin with a discussion of regularization theory and then segue into an analysis of two fundamental methods: Tikhonov regularization and sparsity-inducing penalties. The regularization methods represent different assumptions. We present two methods that show promising results under only mild conditions, and methods that are better suited for problems where we can make stronger assumptions about the data. The chapter also contains a discussion of an alternative state-of-the-art solver for regularized linear systems that is able to handle the newly added methods.

- In **??** we evaluate different grid construction methods. In contrast to the chapter before, we use our prior knowledge to modify the grid generation process, which is more efficient than the regularization approach, but also requires stronger assumptions than the regularization approach. We first discuss a generalized form of sparse grids that we can use to modify the granularity of the generated grid. Secondly, we introduce interaction-term-aware sparse grids that allow us to construct grids that only include a subset of all possible interactions. The methods discussed in that chapter enable us to tackle very-high dimensional problems that are impossible or inefficient for regular sparse grids.

We will use multiple artificial and real-word datasets. To improve the performance of our method, we used some pre-processing steps. They are all documented in **??**.

CHAPTER 2

Sparse Grids & Learning

2.1 Sparse Grids

Sparse grids is a discretization technique that originates from numerical partial differential equations. Even though they can be used for a diverse set of problems, we will only build up the exact amount of theory that is useful for our further study.

This chapter starts with a short description of the adaptive sparse grid technique and then progresses to our application of choice: supervised learning. The discussion of sparse grids follows [**spatAdaptGrid**; **sparse-parsimony**; BG04].

2.1.1 Basis functions

1-Dimensional Basis

As our first building stone we define the "mother hat" function

$$h(x) = \max(1 - |x|, 0).$$

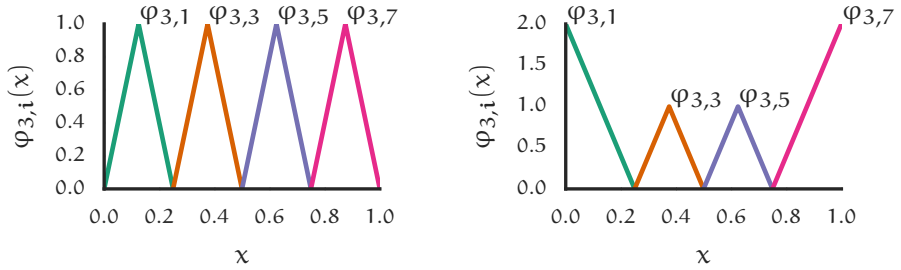
We now define a one-dimensional linear basis function for a level l and an odd index $i \leqslant 2^l - 1$ by

$$\varphi(x)_{l,i} = h(2^l x - i). \tag{2.1}$$

The basis function defined by the former equation are called the linear basis functions [BG04]. These basis functions assume that the function we want to approximate is zero on the boundaries. This is why we use the similarly constructed "modified linear basis functions", as defined by Pflüger in [**spatAdaptGrid**]:

$$\varphi_{l,i}(x) = \begin{cases} 1 & \text{if } l = 1 \wedge i = 1, \\ \begin{cases} 2 - 2^l x & \text{if } x \in [0, 2^{1-l}], \\ 0 & \text{otherwise,} \end{cases} & \text{if } l > 1 \wedge i = 1, \\ \begin{cases} 2^l x - i + 1 & \text{if } x \in [1 - 2^{1-l}, 1], \\ 0 & \text{otherwise,} \end{cases} & \text{if } l > 1 \wedge i = 2^l - 1, \\ h(2^l x - i) & \text{otherwise.} \end{cases}$$

They are identical to the linear basis functions (2.2), except on the boundaries, where they use extrapolation. Note that the modified linear basis is constant for level one. A visualization of all one dimensional basis functions for level three can be seen in **??**. Note that they are placed on a regular one-dimensional grid.



(a) Linear basis (b) Modified linear basis

Figure 2.1: One dimensional basis functions for level three.

We define the useful shorthand notation of multi-indices, which represent a collection of indices. Arithmetical functions act on multi-indices element-wise, as does the relation

$$\boldsymbol{\alpha} \leqslant \boldsymbol{\beta} \iff \forall i\colon \alpha_i \leqslant \beta_i.$$

The l_1 and the maximum-norm are defined for multi-indices by

$$|\boldsymbol{\alpha}|_1 = \sum_{1 \leqslant i \leqslant d} \alpha_i, \; |\boldsymbol{\alpha}|_\infty = \max_{1 \leqslant i \leqslant d} \alpha_i.$$

We use $\mathbf{1}$ and $\mathbf{2}$ as a short-hand for $(1, \ldots, 1)$ and $(2, \ldots, 2)$ respectively.

We can then construct the d-dimensional basis functions with the tensor product construction

d-Dimensional Basis

$$\varphi_{\mathbf{l},\mathbf{i}}(\mathbf{x}) = \prod_{k=1}^{d} \varphi_{l_k,i_k}(x_k),$$

where $\mathbf{l}$ corresponds to the levels and $\mathbf{i}$ to the indices used by the basis function [BG04].

2.1.2 Full & Sparse Grids

Using the index-set **[sparse-parsimony]**

Hierarchical Subspaces

$$G_l = \{(\mathbf{l}, \mathbf{i}) \in \mathbb{N} \times \mathbb{N} \mid 1 \leqslant i_t \leqslant 2^{l_t} - 1,\ i_t \text{ odd}, 1 \leqslant t \leqslant d\},$$

the d-dimensional basis functions span the hierarchical subspaces

$$W_l = \operatorname{span}\{\phi_{l,i} \mid (\mathbf{l}, \mathbf{i}) \in G_l\}.$$

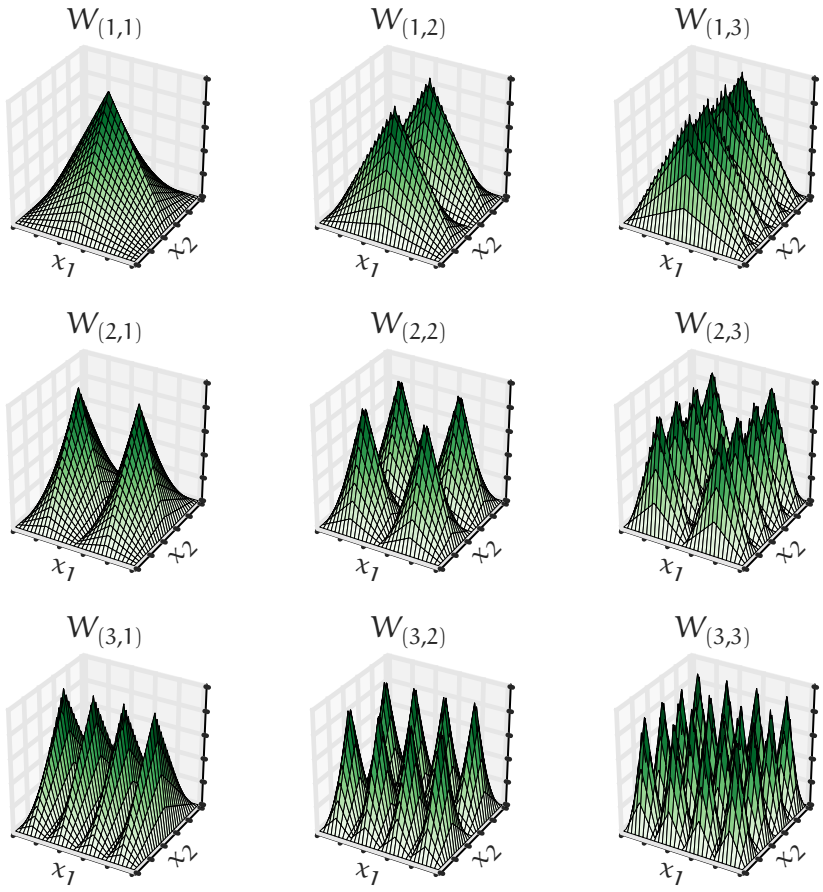


Figure 2.2: Hierarchical two-dimensional subspaces up to level three. With standard linear basis functions.

?? shows all two-dimensional subspaces up to level three. Note that the basis functions are placed on a regular grid.

Using those subspaces, we can create the set of grid points $G_n^{-\infty}$ of the full grid for a level n and its corresponding approximation space $V_n^{-\infty}$ *Full Grid*

$$G_n^{-\infty} = \bigcup_{|l|_\infty \leqslant n} G_l,$$
$$V_n^{-\infty} = \bigoplus_{|l|_\infty \leqslant n} W_l.$$

The number of basis functions of the full grid approximation space $|V_n^{-\infty}|$ is in $O(2^{nd})$.

We can represent every function $f(\mathbf{x})$ in $V_n^{-\infty}$ by

$$f(\mathbf{x}) = \sum_{g \in \mathbf{G}_l} \alpha_g \varphi_g(\mathbf{x}), \tag{2.2}$$

as a sum over all grid points that is weighted by the so called hierarchical coefficients or surpluses α_g.

Function Space

Let $\Omega = [0,1]^d$ represent a d-dimensional interval. We now consider functions $f\colon \Omega \to \mathbb{R}$ with bounded weak mixed second derivatives

$$D^{\mathbf{k}} f = \frac{\partial^{|\mathbf{k}|_1}}{\partial x_1^{k_1} \cdots \partial x_d^{k_d}} f,$$

where $\mathbf{k}$ is a d-dimensional multi-index. In other words, we consider functions that are sufficiently smooth. These functions form the Sobolev space [**sparse-parsimony**]

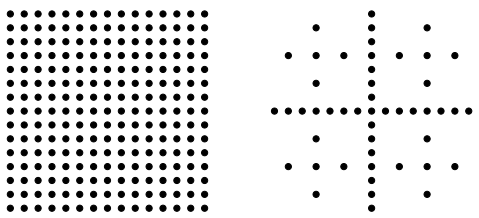
$$H_2^{\text{mix}}(\Omega) = \{f\colon \Omega \to \mathbb{R} \mid D^{\mathbf{k}} f < \infty, |\mathbf{k}|_\infty \leqslant 2, f|_{\partial\Omega=0}\}.$$

Sparse Grid

For this function space sparse grids is a discretization method that represents a reasonable trade-off between accuracy and efficiency. By exchanging the l_∞ with the l_1-norm we get

$$\begin{aligned} G_n^0 &= \bigcup_{|l|_1 \leqslant n+d-1} G_l, \\ V_n^0 &= \bigoplus_{|l|_1 \leqslant n+d-1} W_l, \end{aligned} \tag{2.3}$$

which correspond to the grid point set and the approximation space of sparse grids respectively [BG04]. Again, we can split every function $f(\mathbf{x}) \in V_n^0$ into a weighted sum over all basis functions. A sparse grid for a level n has only $|G_n^0| \in O(2^n n^{d-1})$ grid points. **??** shows a visualization of both grid types for a two-dimensional grid with level four.



(a) Full Grid (b) Sparse Grid

Figure 2.3: Grid visualization for 2-dimensional grids with level four.

2.1.3 Adaptivity

Sparse grids work best if the function adheres to our smoothness assumptions. Functions that are not well behaved, such as functions that are not smooth in some parts of their domain, are more difficult to handle. **spatAdaptGrid** devised an adaptive technique that helps us to approximate challenging functions in [**spatAdaptGrid**]. Instead of relying only on the theoretically optimal results of sparse grids for functions in H^2_{mix}, he described an optimization process to refine grids so that they adapt to the circumstances of the problem.

Because the ideal grid is computationally hard to calculate, a greedy algorithm that approximates the optimal solution is used [**spatAdaptGrid**]. The idea is quite simple: we create new grid points that are likely to capture additional information, close to existing points. An example can be seen in **??**. For a more elaborate discussion we refer the interested reader to [**spatAdaptGrid**].

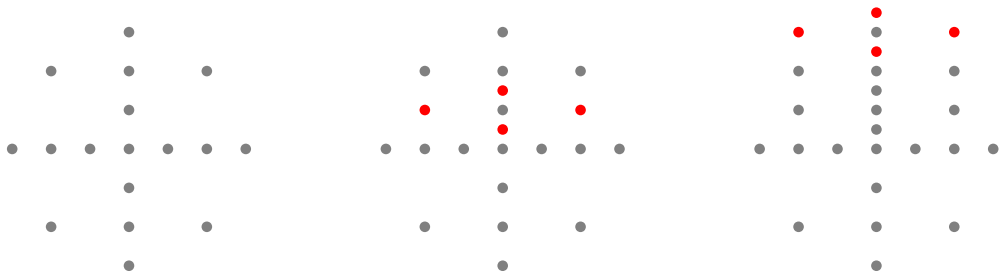


Figure 2.4: We start with a small 2-dimensional grid with level 3. The first picture shows the standard grid, the other two show one adaptivity step each. The red points are the points created by the adaptivity step.

2.2 Learning with Sparse Grids

Supervised Learning

Let the set

$$T = \{(\mathbf{x}_i, y_i)\}_i^n \subset [0,1]^d \times \mathbf{y}$$

be a set of labelled examples with $\mathbf{x} = x_1, x_2, \ldots, x_d$ as predictors and $\mathbf{y}$ as the target variable. Predictors that are not in $[0,1]$ need to be scaled. This set represents our dataset. The goal is not interpolation, as we do not want to find a function that fits the examples exactly. We rather want to find an approximation of our function that generalizes well, i.e. a function that captures the structure of the training data and can be used to predict the target variable for different, yet unseen data points.

We differentiate between

Regression if we want to predict a continuous y,

Classification if we want to predict a discrete value, for example a class.

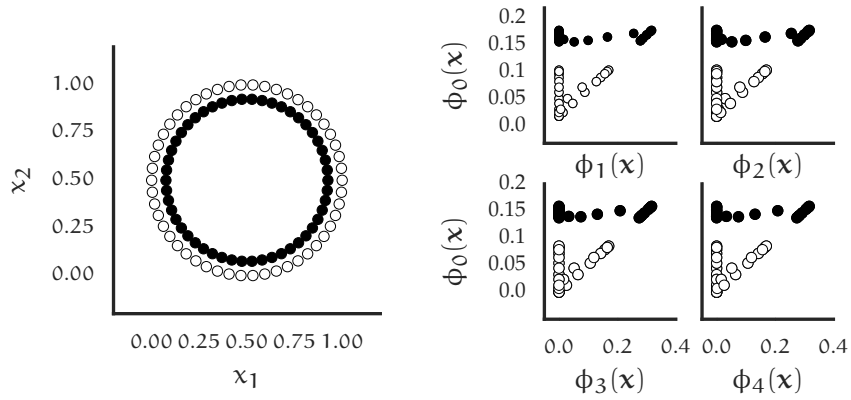
In this thesis we will mostly see examples of regression, the last chapter contains an example for a high-dimensional classification problem.

Regression

Let $\boldsymbol{\phi}(\boldsymbol{x})\colon \mathbb{R}^d \to \mathbb{R}^{m\times 1}$ denote a vector valued function consisting in all m d-dimensional basis functions evaluated at a point $\boldsymbol{x}$ with an associated weight vector $\boldsymbol{\alpha} \in \mathbb{R}^{1\times m}$. We can then express our prediction for y as

$$\hat{y}(\boldsymbol{x}) = \sum_{j=1}^{m} \alpha_j \varphi_j(\boldsymbol{x}) = \boldsymbol{\alpha}^T \boldsymbol{\phi}(\boldsymbol{x}),$$

which is a weighted sum of the basis functions, closely related to **??**.



(a) Original data. (b) Sparse grid perspective of the data.

Figure 2.5: Feature transformation for the circle dataset using a level two grid with standard linear basis.

Feature Transformation

We can view the sparse grid model as a feature map that transforms the originally d-dimensional dataset into a new m-dimensional dataset, where one basis function represents one dimension of the sparse grid approximation space [**sparse-parsimony**]. A simple example is shown in **??**. We can see that the original dataset is not linearly separable, while the sparse grid representation is. Usually we have more dimensions m

than original dimensions n. We then define the model matrix $\mathbf{\Phi} \in \mathbb{R}^{n \times m}$ as

$$\mathbf{\Phi}(\mathbf{x}) = \begin{bmatrix} \phi_1(x_1) & \phi_2(x_1) & \dots & \phi_m(x_1) \\ \phi_1(x_2) & \phi_2(x_2) & \dots & \phi_m(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(x_n) & \phi_2(x_n) & \dots & \phi_m(x_n) \end{bmatrix},$$

where each row corresponds to one datum of the dataset and each column is one of m new features.

Optimization

We can then perform linear regression using the matrix $\mathbf{\Phi}$ as our design matrix. This perspective is useful, because it relates classical statistical methods with sparse grid learning and thus allows us to share common results. The optimization goal is then given by

$$\min_{\boldsymbol{\alpha}} \|\mathbf{\Phi}\boldsymbol{\alpha} - \mathbf{y}\|_2^2 + n\lambda \mathcal{S}(\boldsymbol{\alpha}), \tag{2.4}$$

where $\mathcal{S}(\boldsymbol{\alpha})$ is a regularization operator and λ is a constant. The regularization parameter λ is scaled by the number of data points n. This least-square problem can be solved for differentiable $\mathcal{S}$ by a gradient-based solver.

Adaptivity can be integrated into this process by performing a refinement step after solving the optimization step, iterating until a satisfactory performance is achieved. We select the points that should be refined by calculating the mean squared error (MSE) for the model and then refine the grid, starting with the points with the highest contribution to the error.

Classification

A binary classification problem can be transformed into a regression problem by relaxing the target $\mathbf{y}$. We set $y_i = 0$ for each data point when the datum belongs to the first class and $y_i = 1$ if it does not. The prediction of the model can then be interpreted as a certainty that the datum is a member of the class.

To solve a multi-class-classification problem, one-vs-all classification is used, for which we transform the problem into multiple binary classification problems. We predict the target for an unseen data point by calculating the results for each binary estimator and then report the class label of the learner that returned the largest certainty.

This implies that all methods that improve the performance of the regression procedure also very likely lead to better classification results because we are performing classification via regression.

2.3 Software

Throughout this thesis we will use the following libraries:

SG++ is a sparse grid toolbox implemented in *C++*. This library was used for all experiments and contains all needed methods to recreate our experiments. Every

method described in the following chapters was integrated into *SG++* [**spatAdaptGrid**].

Scikit-Learn is a Python machine learning library. We used it to implement cross-validation procedures and to perform grid-searches for hyper-parameter tuning [**software-sklearn**].

BayesOpt is an implementation of Bayesian optimization procedures. It was used for hyper-parameter search as well [**software-bayesOpt**].

Matplotlib is a popular Python plotting library with which we created all graphics [**software-matplotlib**].

CHAPTER 3

Regularization Methods

During the discussion of **??** we neglected the regularization operator $\mathcal{S}\colon \mathbb{R}^d \to \mathbb{R}$. In this chapter we focus on this operator exclusively. We start with a short introduction to statistical regularization theory and then compare two groups of regularization methods: Tikhonov regularization and sparsity-inducing penalties.

We will use the l_p-norms for vectors

$$\|\boldsymbol{\alpha}\|_p = \left(\sum_{a \in \boldsymbol{\alpha}} |a|^p \right)^{\frac{1}{p}},$$
$$\lim_{p \to \infty} \|\boldsymbol{\alpha}\|_p = \max_{a \in \boldsymbol{\alpha}} a$$

in this chapter.

3.1 Regularization Theory

Bias-Variance trade-off

Regularization helps us to train models that not only fit the data but also generalize well. To understand how regularization works, it is helpful to decompose our error functional. Our model assumes that there is a relation between the predictors $\mathbf{x}$ and the target variable y that can be expressed as

$$y = f(\boldsymbol{\alpha}) + \varepsilon,\ \varepsilon \sim \mathcal{N}(0, \sigma^2),$$

where $f(\boldsymbol{\alpha})$ is the function we want to approximate and ε is normally distributed noise. Consider the expected prediction error at a point $\mathbf{x}$ **[esl]**:

$$\text{Pred. error}\,(\mathbf{x}) = \mathbb{E}\left[\left(y - \hat{f}(\mathbf{x})\right)^2\right] = \text{Bias}\left[\hat{f}(\mathbf{x})\right]^2 + \text{Var}\left[\hat{f}(\mathbf{x})\right] + \sigma^2 \tag{3.1}$$

with

$$\text{Bias}\left[\hat{f}(\mathbf{x})\right] = \mathbb{E}\left[\hat{f}(\mathbf{x}) - f(\mathbf{x})\right],$$
$$\text{Var}\left[\hat{f}(\mathbf{x})\right] = \mathbb{E}\left[\hat{f}(\mathbf{x}) - \mathbb{E}\left[\hat{f}(\mathbf{x})\right]^2\right],$$

where $\hat{f}(\mathbf{x})$ denotes our approximation of the real function $f(\mathbf{x})$. We call **??** the bias-variance trade-off. This equation splits the error into three different parts:

Bias is the error caused by assumptions the model makes,

Variance is the fluctuation of the model around the mean,

Irreducible Error (σ) is the error caused by noise that is inherent to the relation between the predictors and the target variable.

Occam's razor

Following the principle of Occam's razor—parsimonious models are better—all regularization methods penalize complexity. Using regularization leads to smaller and simpler models, which increases the bias. Of course, increasing this part of the error term makes no sense, if we would not get a payoff. Regularization decreases the variance, thus making our model more robust. In this chapter we consider regularization methods that have both a scaling parameter, that controls the strength of our simplicity assumptions, and (sometimes) parameters that we use to modify the kind of our assumptions. We call the parameter that controls the regularization strength λ.

Finding λ

The choice of this parameter is important: If we increase λ, we exchange more bias for variance. This is why **??** implies a trade-off—we cannot have the cake and eat it too! We usually find the ideal λ by a more-or-less intelligent trial and error process. To do this, we train a predictor for each λ we want to consider on a subset of our data (called the training set), and test its performance using cross-validation (CV).

Prior Information

Another way to reason about regularization is as a method to encode our assumptions directly into the training process. Many regularization methods—and all mentioned in this chapter—can be viewed from a Bayesian perspective, which gives an intuitive explanation of the effect of our assumptions.

The choice of the regularization functional is crucial. We do not know which one performs best without training a model. Sometimes we can encode knowledge about the dataset structure but this is often quite difficult. Each method encodes different assumptions but all have in common that they decrease the model complexity.

3.2 Tikhonov Regularization

Tikhonov regularization is one of the most commonly used regularization methods for ill-posed problems. It is also widely used to regularize regression, which leads to solutions with a larger bias but with a smaller variance. We will show the general form of this penalty first and then adapt the penalty to sparse grid learning.

3.2.1 Theory

We can use Tikhonov regularization by setting the regularization penalty S in **??** to

$$S = \|\Gamma\alpha\|_2^2, \tag{3.2}$$

where Γ is a linear operator. The overall optimization goal is then given by

$$\min_{\alpha} \|\Phi\alpha - y\|_2^2 + n\lambda\|\Gamma\alpha\|_2^2. \tag{3.3}$$

We can also view this problem in the constraint minimization form

$$\begin{aligned} &\text{minimize} && \|\Phi\alpha - y\|_2^2 \\ &\text{subject to} && \|\Gamma\alpha\|_2^2 \leqslant l, \end{aligned} \tag{3.4}$$

for a certain constant l. We can see from this formulation that Tikhonov regularization forces our scaled weights α to lie inside a d-dimensional sphere with a diameter of length l. There is an one-to-one correspondence between l in **??** and λ in **??**. Both variables determine the constraints. Tikhonov regularization fits into a Bayesian framework. We can interpret it as a multivariate-Gaussian prior on the weights with zero mean and covariance matrix Γ^{-1} [**stat-inverse**]. The prior is thus distributed as

$$\alpha \sim \mathcal{N}(0, \Gamma^{-1}). \tag{3.5}$$

A common choice for Γ is the identity matrix as proposed in [**spatAdaptGrid**]. This corresponds to a penalty on the summed squared values of the weights. It is a Gaussian prior with the identity matrix as its covariance matrix. In statistics, this method is called ridge regression [**esl**].

Diagonal Matrix

The identity Tikhonov matrix assumes that all weights are distributed with the same variance. Luckily, we can do better for the sparse grid method. Following the assumption of **??**, we can express every function $f \in H_2^{mix}$ as a weighted sum of basis functions. For these functions the following upper bound on the hierarchical coefficients $\alpha_{l,i}$ holds:

$$|\alpha_{l,i}| \leqslant 2^{-d-2|l|_1} \cdot \|D^2 f\|_\infty \in O\left(2^{-2|l|_1}\right), \tag{3.6}$$

where the differential operator norm only depends on the function f and neither on the dimension nor the level of the basis functions [BG04]. Because the dimension is constant for a given grid we can safely exclude it.

This fact can be used to implement a regularization method that is better suited for functions in H_{mix}^2. We impose an improved prior on the weights using Tikhonov regularization with the matrix

$$\Gamma_{i,i} = c^{|l|_1 - d}, \tag{3.7}$$

for a constant c [**sparse-parsimony**]. For $c = 4$ this corresponds to a prior on the variance of the weights that is identical to the upper bound given by **??** up to a multiplicative constant. A different c can be used as well, we can either treat c as an inherent property of the method or as an additional hyper-parameter. We use the dimension d as a normalizing factor, this way the prior corresponds to the series $(1, 1/4, 1/16, \ldots)$. The resulting prior is depicted by **??** for a two dimensional grid.

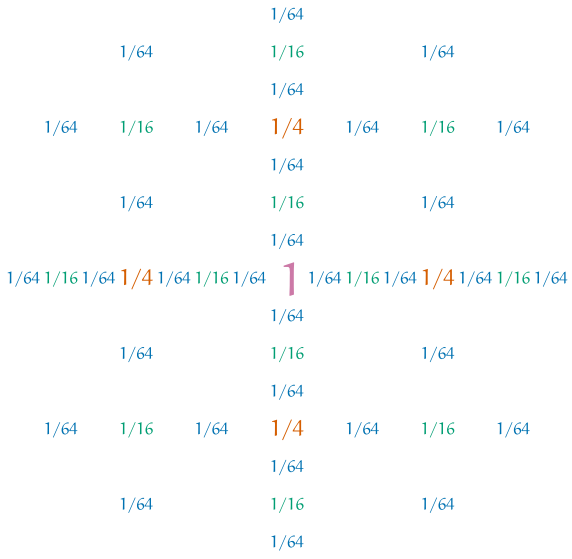


Figure 3.1: Prior generated by **??** for a two dimensional grid with level four. Each number is centered on a grid point and corresponds to the prior for that particular weight.

3.2.2 Implementation

Each regularization method that can be used with the conjugated gradient solver is implemented in the *SG++* library by specializing the base class *OperationMatrix* that offers a method called MULT, which accepts a weight vector and returns a scaled version of the weights. The *OperationMatrix* for the standard ridge regularization returns its input weights unchanged.

For our implementation we created a class *OperationDiagonal* that inherits from *OperationMatrix*. The constructor accepts an argument that allows the specification of the exponent base c. Its implementation of MULT multiplies each input weight with the corresponding inverse prior. We calculate the multiplier for each weight during the first call of the multiplication method and cache it until the grid size changes. This means that we only need to perform this calculation once per refinement step. We determine the multiplier for each grid point by simple iterating over all existing grid points and save the result of **??**. The cost of this operation is in $O(n)$ and is thus negligible.

The class *OperationDiagonal* can then be used in the same way as the implementation of the ridge regularization.

3.2.3 Results & Discussion

To prove the effectiveness of the diagonal method, we first show empirically that **??** holds for a simple function in H^2_{mix}. We then present results that indicate that our proposed regularization functional shows improved results for an artificial dataset, for which the upper bound for the surpluses holds by construction. Finally we show benchmark results for two real-world datasets, comparing the identity matrix with the diagonal operator.

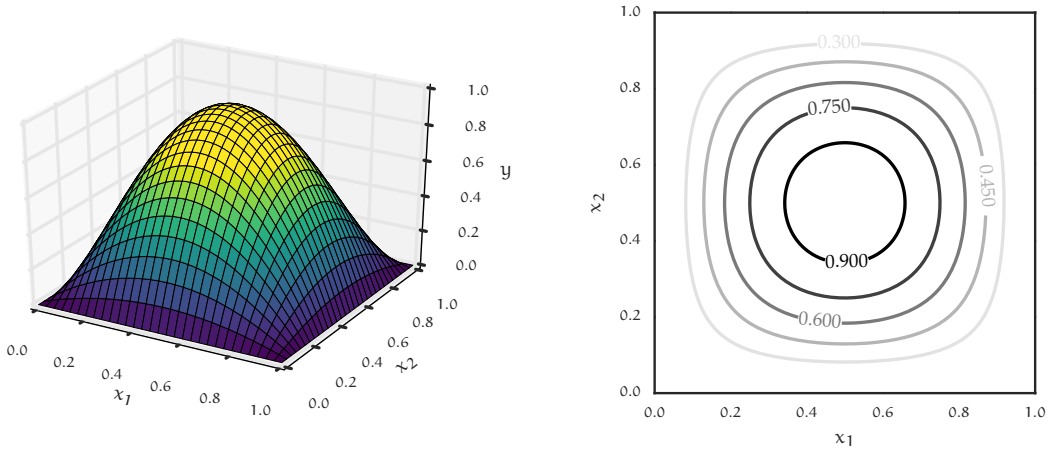


Figure 3.2: Surface and contour plot of the inverse parabola function **??**

Inverse Parabola

We first consider the inverse parabola

$$f(x_1, x_2) = 16x_1(1 - x_1)x_2(1 - x_2). \tag{3.8}$$

We created a two dimensional grid with standard linear basis functions and level three that had 17 grid points. The construction also works for a different choice of level. The

dataset was then created by setting the features x_1, x_2 equal to the coordinates of one grid point each, the target $\mathbf{y}$ was then calculated using **??**. The sparse grid regression model trained on this dataset and the aforementioned grid was able to recover the target perfectly, with a mean squared error smaller than the machine epsilon. More interestingly, the calculated weights were identical to our prior. Note that we did not perform regression but rather interpolation. While this does not prove that this prior holds, it is a simple example for this construction.

Recovering Weights

But does our prior improve the results for a function, when we include our prior knowledge about the weights bound? To test this hypothesis, we constructed another artificial dataset. We first created a two dimensional sparse grid learner with level 3 and sampled its weights from the normal distribution $\boldsymbol{\alpha} \sim \mathcal{N}(0, \boldsymbol{\Gamma}^{-1})$, which corresponds to the prior **??**. The operator $\boldsymbol{\Gamma}$ is our diagonal matrix from **??**. Let $\mathbf{X} \in \mathbb{R}^{n \times 2}$ be our design matrix, where each row is drawn from a two-dimensional uniform distribution. We then created our target vector $\mathbf{y}$ by predicting the result of $\mathbf{X}$ using the constructed model. Right now, this gives a trivial regression problem and to show that the diagonal matrix yields better results, we need to add some noise to $\mathbf{y}$. Let σ denote the standard deviation of $\mathbf{y}$. We then crafted different variants of our dataset by adding normal noise to the target variable with mean zero and standard deviation $s\sigma$, for some values of s. The signal-to-noise-ratio (SNR) of the modified target is then given by $1/s$.

SNR	Exponent Base	λ	Weights-RMSE	CV-RMSE	σ_{noise}
4.0	4.0	4.7149×10^{-5}	0.062 738	0.274 558	0.262 597
2.0	4.5	1.4563×10^{-4}	0.080 263	0.545 097	0.525 194
1.0	6.5	2.5595×10^{-4}	0.095 696	1.083 762	1.050 387
0.5	4.5	2.4421×10^{-3}	0.109 284	2.159 250	2.100 775

Table 3.1: Combinations of λ and exponent base for different SNRs that achieved the best root-mean-square-error (RMSE).

We performed a grid search[1] for sparse grid estimators, testing on a grid consisting of different choices for λ and the exponent bases in the interval $[2, 10]$ with stepsize 0.5. The results for the best parameters can be seen in **??**. Note that we achieved the best cross validations errors with exponent bases that are close to four. There is one outlier, the SNR $= 1$ case, which can be explained by the very noisy data. The regularization parameter λ and the RMSE for the weights decrease for a higher SNR, as expected. All results are close to the theoretical optimal error, which is equivalent to the variance of the noise. Those results indicate that our proposed regularization method improves

[1] We used a Monte-Carlo cross-validation method with ten iterations and a 9:1 train-validation split for this experiment, in contrast to the usual 10-Fold method, to compare the different estimators.

performance for a dataset, which can be approximated by a model with surpluses that follow the upper bound.

Concrete Dataset

So far we have only seen examples for artificial datasets, which adhered to our assumptions by construction. We now introduce our first real-world dataset, the concrete dataset. For this dataset our goal is to predict the compressive strength of concrete, using the recipe of its mixture and its age in days. The recipe consists of seven quantitative predictors, all given in the unit kilogram per thousand liters. Concrete consists of the ingredients cement, blast furnace slag, fly ash, water, superplasticizer, coarse aggregate, and fine aggregate. The dataset was donated to the UCI machine learning repository [**datasets-uci**] by **datasets-concrete** and was first published in their paper [**datasets-concrete**]. It consists in 1030 instances altogether. We split the data and used 80% for training and the other 20% solely as testing data.

We performed a grid search using a learner with level four for the diagonal regularization with fixed exponent bases $c = 4$, and $c = 1$, which correspond to the diagonal and the identity matrices respectively. Each learner performed five adaptivity steps, each refining three grid points. The performance of the estimators was estimated using a standard 10-fold cross validation procedure.

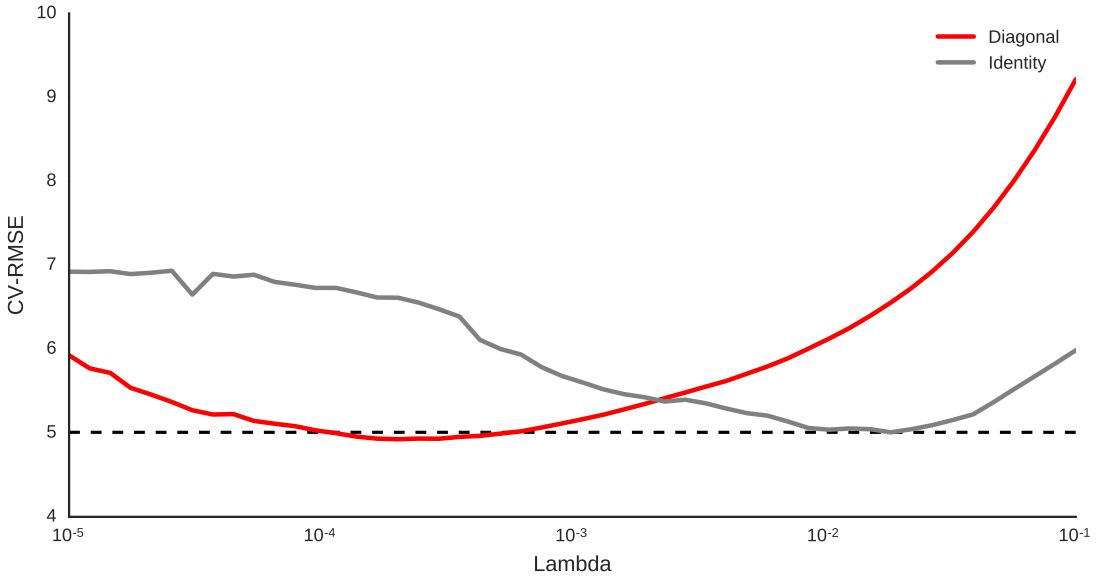


Figure 3.3: Results for the concrete dataset obtained with estimators with level four for two different Tikhonov matrices.

The results are shown in **??**. We can see that our method resulted in better results than the identity regularization, although the difference between the two methods was

small.

Power Plant Dataset

We also tested the performance for the power plant dataset, for which a visualization is given by **??**. The target variable of this dataset is the hourly energy output for a combined cycle power plant. To predict this target, we use the temperature, the ambient pressure, the relative humidity, and the exhaust vacuum as predictors. This dataset appeared first in [**datasets-powerplant**] and was donated to the UCI machine learning repository [**datasets-uci**] by **datasets-powerplant** It consists in 9568 instances, which were split into a training and testing dataset at a ratio of 8:2.

We then performed a grid search over an grid of lambdas in the interval $[10^{-10}, 10^{-1}]$ for learners with level 5, again using ten-fold cross validation. The grids were refined five times, refining three points for each adaptivity step. The results can be seen in **??**. Note that this figure only shows the results in a small interval, all values of λ that were larger than the values shown resulted in far larger errors. Again, we can see that the diagonal regularization improved the RMSE by small margin of roughly 0.03. Note that this improvement is larger than the overall-effect of the identity regularization, which implies that, in contrast to the ridge regularization, the diagonal matrix achieved significant improvements over the non-regularized regression.

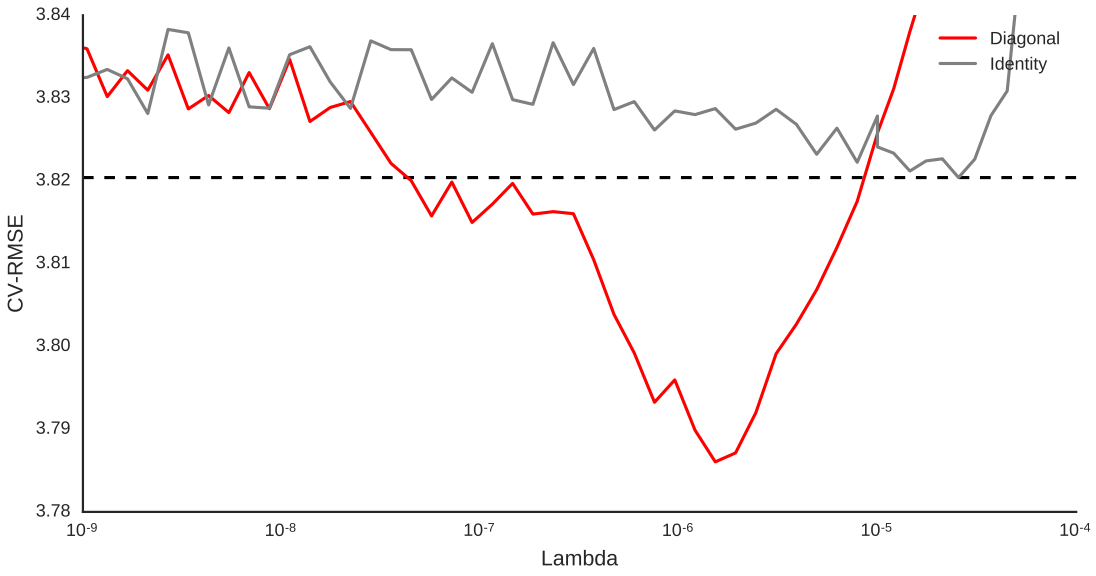


Figure 3.4: Identity vs. diagonal matrix, for the power plant dataset with learners of level five.

We can conclude from these results that the diagonal regularization method is able to result in better outcomes if the datasets adhere to the assumptions of the method. The

tests on real-world datasets showed that our proposed method is a solid alternative to the standard ridge regularization penalty, increasing the performance by a small margin for a negligible additional performance cost.

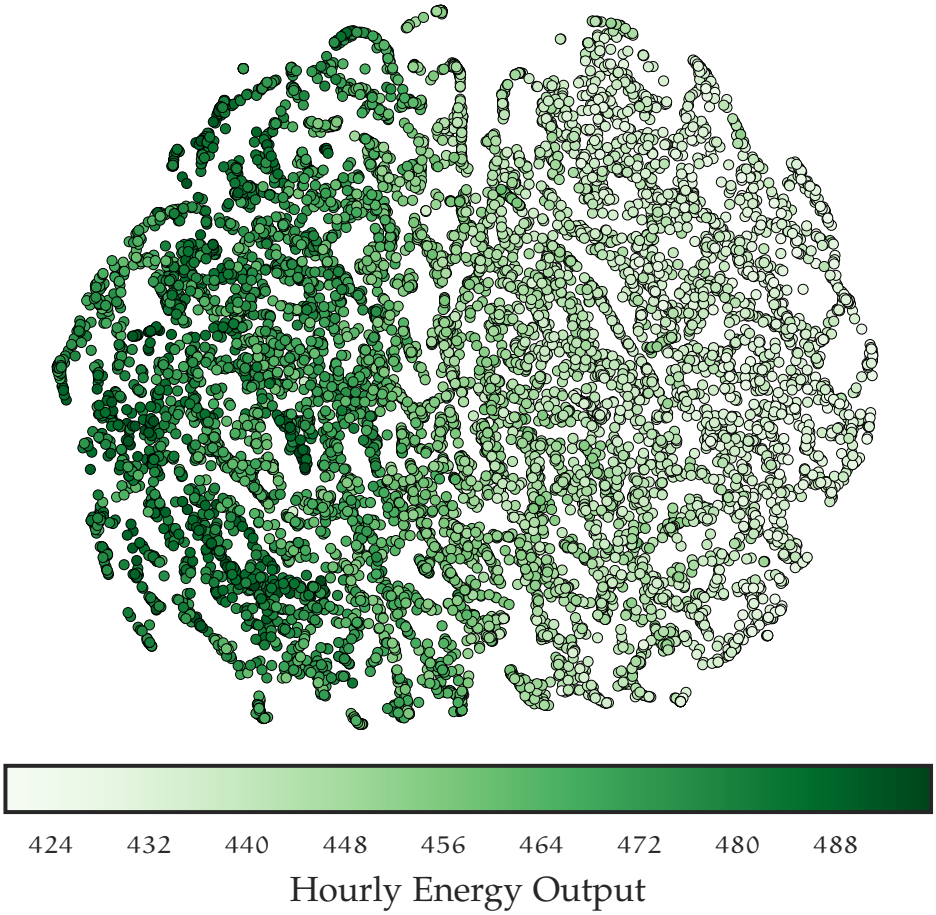


Figure 3.5: 2-dimensional visualization of the power plant dataset. This visualization was generated using the t-Distributed Stochastic Neighbor Embedding (t-sne) algorithm [**tsne**]. Data points close to each other in the original, multi-dimensional dataset are also close in this two-dimensional representation. The target values are not considered for this calculation, they are only used to determine the color of the points. The axes contain no useful information and are therefore omitted.

3.3 Sparsity-Inducing Penalties

We have seen different variations of Tikhonov regularization. All regularization methods so far have one thing in common: they used the squared Euclidean-norm. In this section we will look at three different methods, which all use different norms that induce sparsity. Sparsity in this context means that some entries of the weight vector $\boldsymbol{\alpha}$ are exactly zero.

Lasso

A first simple method is the so called lasso[2], first published by Tibshirani in [**lasso**]. We can represent this procedure in a form similar to **??** for a constant l:

$$\begin{aligned} &\text{minimize} && \|\boldsymbol{\Phi}\boldsymbol{\alpha} - \mathbf{y}\|_2^2 \\ &\text{subject to} && \|\boldsymbol{\alpha}\|_1 \leqslant l, \end{aligned}$$

which we can also cast into the more convenient Lagrangian representation

$$\mathcal{S} = \|\boldsymbol{\alpha}\|_1.$$

We can see from the constraint form that the lasso only accepts solutions that are inside a d-dimensional hyper-cube centered on the origin. **??** compares the constraint regions of the ridge and lasso regularization. Let $\hat{\boldsymbol{\alpha}}$ denote the non-regularized least squares solution. All values of $l \geqslant \|\hat{\boldsymbol{\alpha}}\|_1$ shrink the predictors, some weights can be exactly zero [**lasso**].

Laplace Prior

We can also view this smoothness function from a Bayesian perspective. In this context, lasso regularization can be seen as a Laplace prior on the weights with zero mean. A visualization of the prior for ridge and lasso regularization can be seen in **??**. We can see that the Laplace prior puts more of its weight at zero and on its tails than the normal distribution, which implies that solutions are more likely to be exactly zero or larger than for the ridge estimate [**lasso**].

Relation to Best-Subset

The lasso is an instance of a large family of regularization functionals, where the penalty is realized as the l_p norm of the weights. We define the "l_0-norm" $\|\boldsymbol{\alpha}\|_0 = |\{a \in \boldsymbol{\alpha} \mid a \neq 0\}|$ as the cardinality of the support of $\boldsymbol{\alpha}$, i.e. the number of entries of a vector that are not zero. The name "l_0-norm" reflects that it is similar to an l_p norm but is not a proper norm itself. The l_0-penalty corresponds to the standard best-subset feature-selection method. Because the l_p norms are only convex for $p \geqslant 1$, the lasso can be interpreted as the best convex approximation of the best-subset method [**sparse-learning**; **lasso**].

Elastic Net

After we convinced ourselves that the lasso indeed leads to sparse solutions, we will discuss its weaknesses. An important modification of the lasso is the elastic net penalty,

[2] In the original paper [**lasso**] the name lasso was introduced as an acronym for "least absolute shrinkage and selection operator". We use the term in a more metaphorical manner, where the lasso stands for an actual rope used to catch cattle, or in our case predictors. See also [**sparse-learning**].

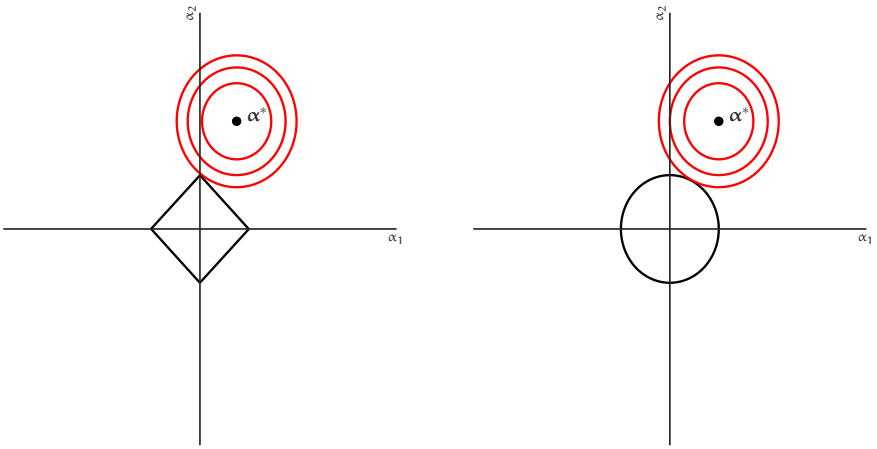


Figure 3.6: Constraint regions for lasso and ridge regularization respectively. The red contours correspond to a l_2^2 loss function with optimal solution $\boldsymbol{\alpha}^*$. The figure is inspired by figure 3.11 of [**esl**].

first introduced in [**elasticnet**] by **elasticnet** Let n denote the number of data points and p the number of grid points. The lasso does not show good results in the following situations [**elasticnet**]:

- In the $(n < p)$-case, the lasso selects at most n predictors.
- When the predictors are highly correlated, the lasso selects one of the predictors at random or shows otherwise unstable behaviour. If two variables are identical, the lasso solution is not unique [**sparse-learning**].
- The Tikhonov regression shows better practical results than the lasso in the correlated case.

All those reasons indicate that the lasso is not a good choice for some problems. The Tikhonov regression is no direct competitor, because it does not lead to sparse solutions.

The elastic net solves those problems by adding some ridge regularization to the lasso penalty. It is a compromise between both methods and is given by

$$\lambda \mathcal{S}(\boldsymbol{\alpha}) = \lambda \|\mathbf{x}\|_2 + \gamma \|\mathbf{x}\|_1, \tag{3.9}$$

where λ and γ are two independent parameters [**elasticnet**]. Although this form is more convenient for some of the following calculations, it does not yield a nice interpretation.

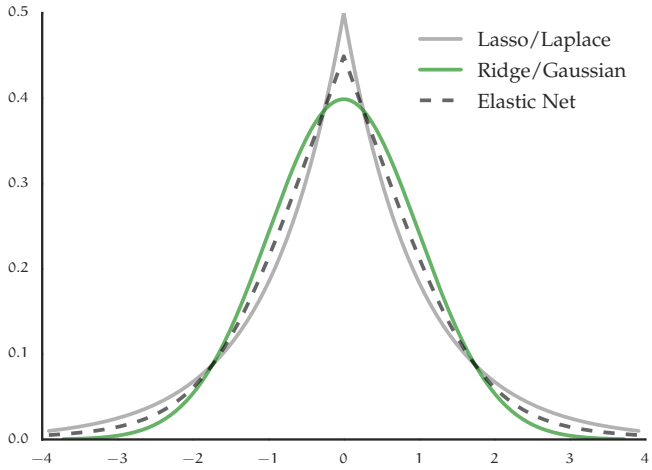


Figure 3.7: Plot of priors for ridge, lasso and elastic net.

This is why we re-parametrize the previous equation as

$$\lambda \mathcal{S}(\boldsymbol{\alpha}) = \lambda_1 \left((1 - \lambda_2) \|\mathbf{x}\|_2 + \lambda_2 \|\mathbf{x}\|_1 \right),$$

where λ_1 determines the overall regularization effect and λ_2 controls the relative influence of the lasso term. From this equation we can recover both the ridge and the lasso regularization, by setting λ_2 to 0 or 1 respectively. The advantage of the method is then obvious. While the lasso part of the penalty enforces sparsity, the ridge part shrinks correlated variables together, thus stabilizing the feature selection [**elasticnet**]. A visualization of the corresponding prior, a mixture of an Gaussian and a Laplace prior, can be seen in **??**.

Group Lasso

Another useful generalization of the lasso is the group lasso that shrinks groups of weights at the same time. Either all members of a group are selected, or none at all. It was first developed by **grouplasso** in [**grouplasso**], then extended to the logistic regression method in [**grouplasso-logistic**]. **grouplasso-generalizations** discussed some generalizations of the method in [**grouplasso-generalizations**] and a discussion of the statistical properties is offered by [**grouplasso-benefit**]. The discussion here follows [**sparse-learning**], unless otherwise noted.

Let $\mathcal{P}$ denote a partition of $\boldsymbol{\alpha}$, i.e. a set of disjoint subsets whose union is $\boldsymbol{\alpha}$. We can then define the group lasso as

$$\mathcal{S}(\boldsymbol{\alpha}) = \sum_{\mathbf{p} \in \mathcal{P}} \left(\sqrt{|\mathbf{p}|} \right) \|\mathbf{p}\|_2,$$

where $|\mathbf{p}|$ denotes the size of a group $\mathbf{p}$ and is used as a weighting factor. We can choose to weight the groups differently but the square root of the cardinality is a useful factor that is simple to calculate and works well in practice. If we would not include this factor larger groups would be more likely to be included in the final model [**sparse-learning**].

Order of grid points

Let

$$\text{order}(\mathbf{p}) = |\{i \mid p_i \neq 0.5\}|$$

denote the cardinality of the support for a grid point with coordinates $\mathbf{p}$. A modified linear basis function with coordinate 0.5 for a dimension is constant with respect to this dimension. For example, the bias term is constant for all dimensions and has therefore order zero. The points of order one correspond to all basis functions that are constant for all but one dimension, and so on. We call all grid points with order larger than one interaction terms, because they model the interaction between different dimensions.

Grouping Grid Points

We then partition our weights into groups consisting of all terms of the same order. This grouping corresponds to the original predictors, including the interactions between them. **??** shows a possible algorithm which results in our chosen partition. We use the fact that all grid points have a unique sequence number, which we can use to refer to a specific basis function. This way of grouping variables and its usefulness for the group lasso is also discussed in [**sparse-parsimony**]. Note that we can recover the original lasso penalty by choosing partitions of size one.

It can be shown that the group lasso penalty performs better than the lasso regularization if the group structure is evident in the data. If the structure is not contained in the data, the lasso shows stronger theoretical results. For a more elaborate discussion of these and more results of the group lasso, we refer to [**grouplasso-benefit**].

Non-differentiable Penalties

All discussed sparsity-inducing penalties are not differentiable at zero. We were able to solve the Tikhonov regularization method using a standard conjugated gradient scheme. This is not possible for the methods presented in this section, because we cannot rely on gradient information any more. This is why we need to solve these problems using a gradient-free optimization procedure. Because we can still profit from the structure of our problem, we do not have to use a black-box-optimization algorithm. We present a solver for least-squares problems with sparse regularization in the following section.

3.3.1 Proximal Methods

There are many solvers for the lasso and related methods. In this section we present the Fast Iterative Shrinkage Tresholding Algorithm (FISTA), first introduced in [BT09]. This solver offers a good compromise between flexibility and performance and we can use it to solve all presented sparse regularization methods. The discussion of FISTA

Algorithm 1 Group Lasso: Group

Input: gridStorage that contains all grid points, weight vector $\boldsymbol{\alpha}$.

```
1:  function GROUP(gridStorage, α)
2:      curGroup ← 0
3:      groups ← HashMap<vector<bool>, int>()
4:      groupVec ← vector<bool>()
5:      for point ∈ grid do
6:          usedDims ← vector<bool>(false,..., false)
7:          for curDim ∈ {0, 1, ..., d} do
8:              coordinate ← GETCOORDINATE(point, curDim)
9:              usedDims[curDim] ← coordinate ≠ 0.5
10:         if usedDims ∈ groups then
11:             groupVec[GETSEQNUMBER(point)] ← groups[usedDims]
12:         else
13:             groupVec[GETSEQNUMBER(point)] ← curGroup
14:             curGroup ← curGroup + 1
15:     return groups, groupVec
```

follows [BT09], the description of the proximal operators follows [PB14]. FISTA is able solve problems of the form

$$\min_{\boldsymbol{\alpha}} F(\boldsymbol{\alpha}) = f(\boldsymbol{\alpha}) + g(\boldsymbol{\alpha}), \tag{3.10}$$

i.e. minimizing functions that can be expressed as a sum of a convex, smooth function $f(\boldsymbol{\alpha})$ and another convex, possibly non-smooth function $g(\boldsymbol{\alpha})$. It is required that $f(\boldsymbol{\alpha})$ has a Lipschitz-continuous gradient. This means that the following condition holds for all possible vectors $\mathbf{x}$ and $\mathbf{y}$ for some positive constant L:

Lipschitz constant

$$\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\| \leqslant L\|\mathbf{x} - \mathbf{y}\|. \tag{3.11}$$

We call the smallest possible value of L the Lipschitz constant.

For our goal given by **??** we set $f(\boldsymbol{\alpha})$ to

$$f(\boldsymbol{\alpha}) = \frac{1}{2}\left\|\boldsymbol{\Phi}\boldsymbol{\alpha} - \mathbf{y}\right\|_2^2,$$

such that the gradient of f is given by

$$\nabla f(\boldsymbol{\alpha}) = \boldsymbol{\Phi}^T\left(\boldsymbol{\Phi}\boldsymbol{\alpha} - \mathbf{y}\right).$$

The Lipschitz constant for the gradient of f is

$$L_{\nabla f} = (\sigma_{max}(\Phi))^2, \tag{3.12}$$

where σ_{max} corresponds to the maximum singular value [BT09]. Additionally we set g to

$$g(\alpha) = n\lambda\delta(\alpha).$$

We now define the Moreau envelope of a function $g(\alpha)$, which is given for any $\lambda \in (0, +\infty)$ by *Moreau envelope*

$$M_g(\alpha, \lambda) = \inf_x \left\{ g(x) + (1/(2\lambda))\|x - \alpha\|_2^2 \right\}. \tag{3.13}$$

It is a regularized, smooth version of our function $g(\alpha)$ that has the same minimum as $g(\alpha)$. This implies that every point that minimizes $M_g(\alpha, \lambda)$ also minimizes our original function $g(\alpha)$ [PB14]. Finally, we are able to define the proximal operator that returns the infimum point of **??** by

$$\mathbf{prox}_g(\alpha, \lambda) = \operatorname*{argmin}_x \left\{ g(x) + (1/(2\lambda))\|x - \alpha\|_2^2 \right\}. \tag{3.14}$$

The proximal operator can be viewed as a gradient step with stepsize λ on the Moreau envelope $M_g(\alpha, \lambda)$ *Proximal Operator*

$$\mathbf{prox}_g(\alpha, \lambda) = \alpha - \lambda \nabla M_g(\alpha, \lambda).$$

This identity follows by rewriting **??** in terms of the proximal operator and calculating the gradient [PB14]. We can use **??** to minimize $M_g(\alpha, \lambda)$ and thus also for optimizing $g(\alpha)$. In the most general case **??** would imply the need to solve a convex optimization problem. Fortunately we can find closed form solutions for many functions. Consider for example $g(\alpha) = 0$. In this case, the Moreau envelope and the proximal operator are trivial and given by

$$M_0(\alpha, \lambda) = \inf_x \left\{ 0 + 1/(2\lambda)\|\alpha - x\|_2^2 \right\},$$
$$\mathbf{prox}_0(\alpha, \lambda) = \alpha.$$

It is obvious that the minimum of $M_g(\alpha, \lambda)$ is equivalent to the minimum of $g(\alpha)$, the proximal operator corresponds to a gradient step on M_g.

We are now going to develop a minimizer for our composite goal that resembles a majorization-minimization algorithm. To do this, we first define an upper-bound of $F(\alpha)$ (*majorizing*) that we are then going to minimize (*minimization*) [PB14].

We first give a regularized linearization of $f(\alpha)$ at an arbitrary but fixed point y for an $L > 0$: *Upper Bound*

$$\hat{f}_L(\alpha, y) = f(y) + \langle \alpha - y, \nabla f(y) \rangle + L/2\|\alpha - y\|_2^2, \tag{3.15}$$

where the angle brackets $\langle x, y \rangle = y^\intercal x$ represent the inner product. The first two terms are given by the first order Taylor expansion of $f(\alpha)$ at the point y, the last term can be interpreted as a trust-region or regularization that punishes large deviations from y [PB14]. We then combine this linearization with our second function to archive an upper-bound of $F(\alpha)$:

$$Q_L(\alpha, y) = f(y) + \langle \alpha - y, \nabla f(y) \rangle + L/2 \|\alpha - y\|_2^2 + g(\alpha). \tag{3.16}$$

We can see from **??** that $Q_L(\alpha, y)$ is an upper-bound of $F(\alpha)$ if L is equal to or greater than the Lipschitz constant of $\nabla f(\alpha)$.

Fixed-point minimizer

The minimizer for this approximation is then given as the fixed-point equation

$$\begin{aligned} \pi_{g(\alpha)}(\alpha^*, L) &= \underset{x}{\operatorname{argmin}} \{Q_L(x, \alpha^*)\} \\ &= \mathbf{prox}_g(\alpha^* - L^{-1} \nabla f(\alpha^*), L^{-1}) \\ &= \mathbf{prox}_g(\alpha^* - L^{-1} \Phi^\intercal (\Phi \alpha^* - y), L^{-1}), \end{aligned} \tag{3.17}$$

where α^* denotes the optimal solution and L is the Lipschitz constant of ∇f given by **??** [BT09]. In this equation L is used to determine the optimal stepsize. This minimizer is called proximal gradient algorithm (or proximal-splitting) in the literature, because we first perform a gradient step on $f'_L(\alpha)$ given by **??** and then a proximal step on $g(\alpha)$ [PB14]. Using **??** repeatedly on a point will result in the fixed-point, i.e. the minimum of the upper bound, and thus also in the minimum of our original goal [PB14].

Algorithm 2 Iterative Shrinkage Tresholding Algorithm (ISTA) [BT09]

Input: Lipschitz constant L of ∇f, regularization parameter λ

```
1: function ISTA(L, α)                          ▷ α is an initial guess
2:     while not converged do
3:         α ← π_{g(α)}(α, L)
4:     return α
```

?? is all we need to define the simple iterative scheme called Iterative Shrinkage Tresholding Algorithm (ISTA), which is presented by **??**. Originally the name ISTA was only used for solving the lasso problem, but is now used for the more general algorithm as well. This iterative scheme is identical to the standard gradient descent algorithm for our trivial function $g(\alpha) = 0$.

Proximal Operators for Regularization

So far we have only seen the proximal operator of a very simple function. Of course, this is not yet satisfactory. The goal of this chapter is to describe a solver for non-differentiable penalties. Fortunately, closed form solutions for the other needed

proximal operators exist as well:

$$\begin{aligned}
\text{for Lasso} \qquad & \left(\mathbf{prox}_{\lambda\|\boldsymbol{\alpha}\|_1}(\boldsymbol{\alpha}, t)\right)_i = [\alpha_i - t\lambda]_+ - [-\alpha_i - t\lambda]_+, \qquad (3.18)\\
\text{for Ridge} \qquad & \left(\mathbf{prox}_{\lambda\|\boldsymbol{\alpha}\|_2^2}(\boldsymbol{\alpha}, t)\right)_i = (\alpha_i/(1+2t\lambda)),\\
\text{for Elastic Net} \qquad & \mathbf{prox}_{\lambda\|\boldsymbol{\alpha}\|_1+\gamma\|\boldsymbol{\alpha}\|_2^2}(\boldsymbol{\alpha}, t) = (1/(1+2t\gamma))\left(\mathbf{prox}_{\lambda\|\boldsymbol{\alpha}\|_1}(\boldsymbol{\alpha}, t)\right)\\
\text{for Group Lasso} \qquad & \left(\mathbf{prox}_{\lambda\sum_{p\in\mathcal{P}}\sqrt{|p|}\|\mathbf{p}\|_2}(\boldsymbol{\alpha}, t)\right)_{\mathbf{p}} = \left[1 - \left(\lambda t\sqrt{|\mathbf{p}|}\right)(\|\mathbf{p}\|_2)^{-1}\right]_+ \mathbf{p},
\end{aligned}$$

where $(x)_+ = \max(x, 0)$ denotes the positive part of x and t is a stepsize. The regularization parameters depend on the function $g(\boldsymbol{\alpha})$. We omit the derivations for the sake of brevity, the interested reader refers to the survey paper [PB14]. By setting the regularization parameter λ equal to zero we again recover the gradient minimization method.

Minimizing Lasso

These proximal operators can be used to define a minimizer for a non-smooth function g. For example, combining **??** with the proximal operator for the lasso functional $g(\boldsymbol{\alpha}) = n\lambda\|\boldsymbol{\alpha}\|_1$ given by **??** results in the minimizer

$$\begin{aligned}
\pi_{n\lambda\|\boldsymbol{\alpha}\|_1}(\boldsymbol{\alpha}^*, L) &= \mathbf{prox}_{\lambda\|\boldsymbol{\alpha}\|_1}(\boldsymbol{\alpha}^* - L^{-1}\nabla f(\boldsymbol{\alpha}^*), L^{-1})\\
&= \left[\left(\boldsymbol{\alpha}^* - L^{-1}\nabla f(\boldsymbol{\alpha}^*)\right) - n\lambda L^{-1}\right]_+ - \left[-\left(\boldsymbol{\alpha}^* - L^{-1}\nabla f(\boldsymbol{\alpha}^*)\right) - n\lambda L^{-1}\right]_+,
\end{aligned}$$

again given as a fixed-point iteration. In this equation the function $[x]_+$ is applied element-wise on its input vector. We can then use this minimizer with **??** to compute a solution to the lasso problem. The fixed-point equations for the lasso and elastic net regularization are computed analogously.

Nesterov's accelerated gradient descent

ISTA always converges to the global maximum, but only does so linearly [BT09]. To overcome this problem, Beck and Teboulle combined the ISTA algorithm with the accelerated gradient descent algorithm discovered by Nesterov. Nesterov's accelerated gradient descent is closely related to the ordinary gradient descent algorithm. The first step is identical, each following step carries some momentum of the step before, thus stabilizing the procedure. It is an optimal first-order optimization schema, i.e. one that cannot be improved asymptotically. It achieves quadratic convergence. This property is retained when combined with the proximal-splitting procedure **??**, the result is called FISTA [BT09]. Each step of FISTA evaluates the gradient and the proximal operator once, just as ISTA does. This means that the accelerated algorithm has a comparable cost for each iteration.

Linesearch

Another problem with **??** is its dependence on the Lipschitz constant of ∇f to determine the optimal stepsize. For our choice of f, the best constant L is given by **??**. To avoid this expensive calculation, we use a backtracking line search to determine a suitable stepsize. In this line search procedure we use **??** as an upper bound for **??**. We

do this by iterating and finding the smallest L for which **??** is an upper bound. This always results in the Lipschitz constant [BT09]. It is then straightforward to derive **??**.

Algorithm 3 Linesearch [BT09]

Input: $L > 0, \eta > 1, \boldsymbol{\alpha}$

```
1: function LINESEARCH(α, L)
2:     i ← 0
3:     do
4:         L ← η^i L
5:         prox ← π_α(α, L)
6:         i ← i + 1
7:     while F(prox) < Q_L(prox, α)
8:     return prox and L          ▷ Also return prox to avoid duplicate calculations.
```

We need to evaluate the line search once for each iteration step. It is possible that this procedure finds a non-optimal L, i.e. an L that is larger than the Lipschitz constant. This leads to a smaller stepsize, which is not a problem in practice because our optimization procedure still converges, although slower than possible. We have to take that into consideration for our choice of linesearch parameters. Usual values are $L = 0.5$ and $\eta = 2$ [**fista-backtracking**]. Using **??**, we can finally present an optimal first order optimization algorithm, shown by **??**.

It is of course possible to use a constant stepsize like in **??**. To do this, replace the line search with the minimal value of L and calculate $\pi(\boldsymbol{\alpha}, L)$ directly. We can also integrate the linesearch into the ISTA algorithm, by replacing the fixed L with a call to the linesearch subroutine. A comparison of the practical speed of ISTA and FISTA with constant stepsize can be seen in **??**.

An alternative backtracking scheme for FISTA is offered in [**fista-backtracking**]. It offers the same asymptotic convergence speed, but shows practical improvements for some minimization problems. A discussion of FISTA and other solvers for sparse grids with sparsity-inducing penalties is contained in [**sparse-parsimony**].

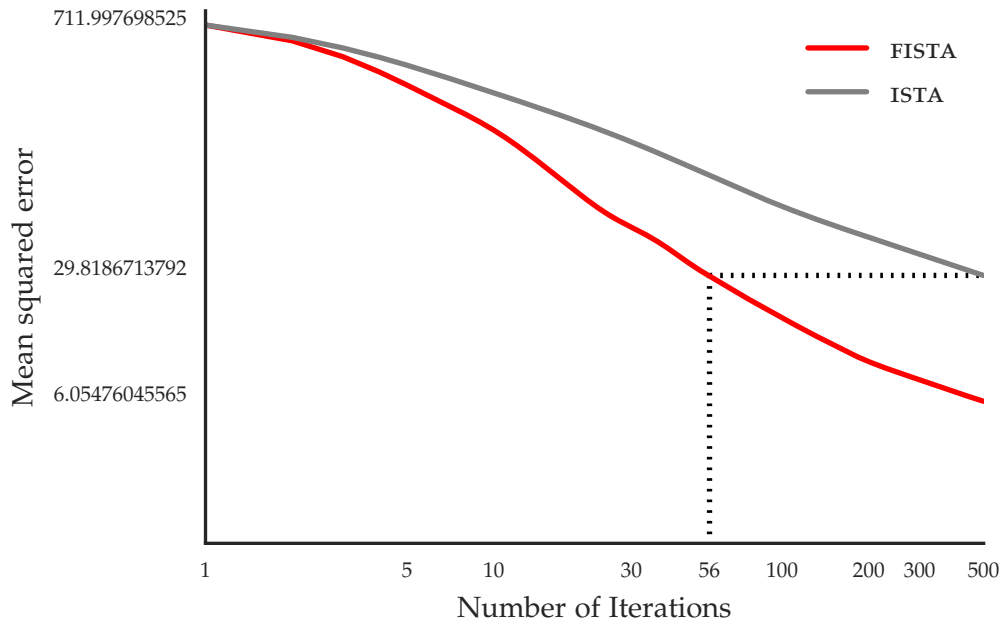


Figure 3.8: Comparison of FISTA and ISTA, both with constant stepsize. The figure shows the training MSE for 500 iterations with $\lambda = 0.1$ for the training set of the concrete data set. The dotted lines indicate the best value reached by ISTA. Note that FISTA is able to return a better result after only 56 iterations — compared to the 500 needed by ISTA.

Algorithm 4 Fast Iterative Shrinkage Tresholding Algorithm (FISTA) [BT09]

Input: Initial guess for Lipschitz constant L of ∇f, regularization parameter λ

function FISTA(L, α) ▷ α is an initial guess for α^*.
  $y \leftarrow \alpha$
  $t \leftarrow 1$
  while not converged **do**
    $\alpha_{\text{before}} \leftarrow \alpha$
    $\alpha, L \leftarrow$ LINESEARCH(y, L) ▷ Linesearch returns $\pi_L(y)$ and the used L.
    $t_{\text{before}} \leftarrow t$
    $t \leftarrow 1/2(1 + \sqrt{1 + 4t^2})$
    $y \leftarrow \alpha + (t_{\text{before}} - 1)\, t^{-1}\, (\alpha - \alpha_{\text{before}})$
  return α

3.3.2 Implementation

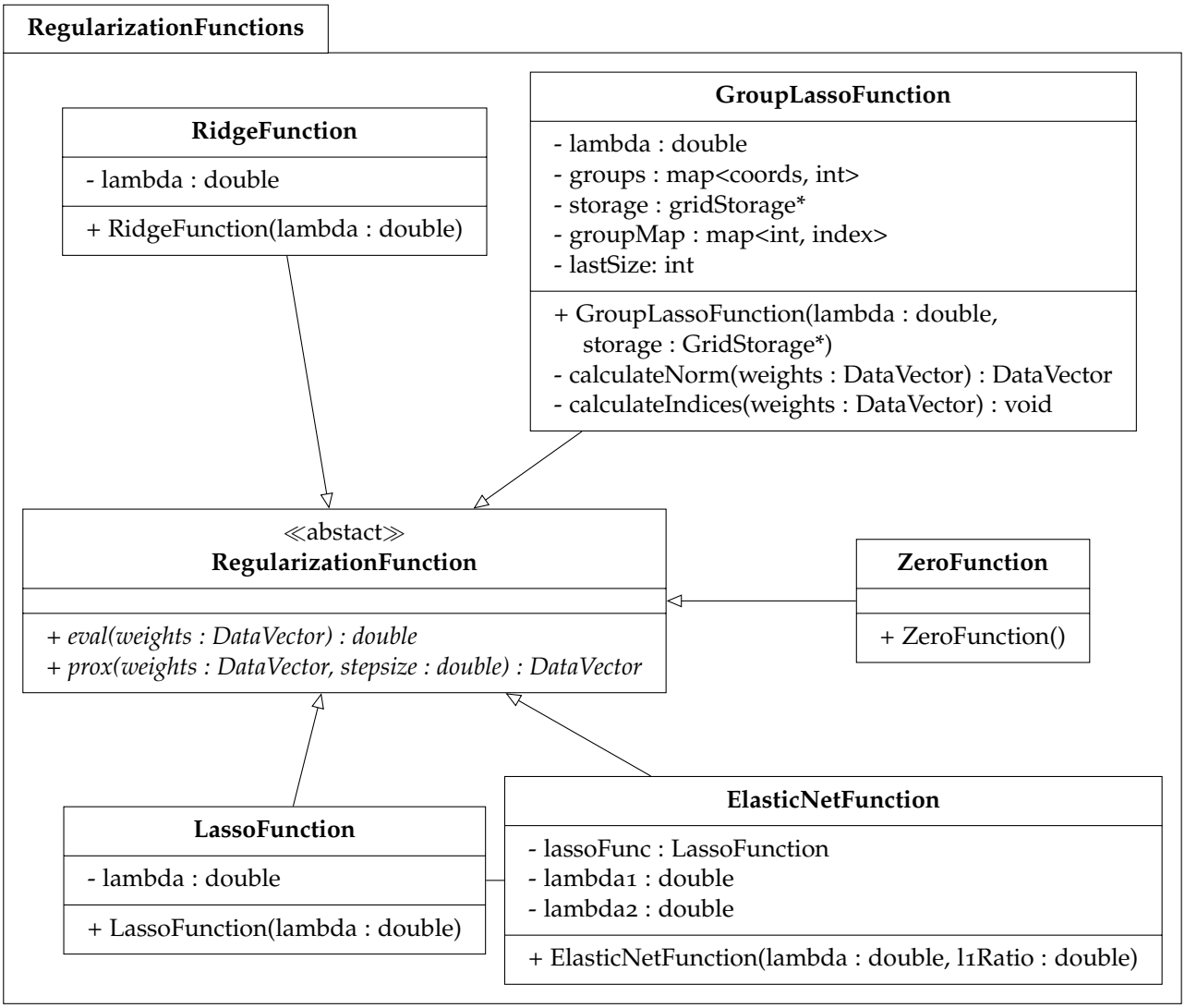


Figure 3.9: UML-class diagram of the implementation of the regularization functions.

As seen in **??** we define a base class RegularizationFunction that offers the two methods EVAL and PROX, which calculate $g(\alpha)$ and its proximal operator respectively. Every parameter that is needed for each functional is passed during construction, this way we achieve a great amount of flexibility. Classes that use the regularization functions do not have to be concerned about the definition or the parameters of the functionals, they can treat them as a black box. We offer an implementation of the ridge,

the lasso, the elastic net and the group lasso function, which resemble the definitions outlined in this chapter. The group lasso function uses the following subroutines:

calculateIndices that partitions our weights into groups that share the same order. This method is only called when the grid size changes, so only once during the first solver iteration. When the grid size changes, e.g. due to an adaptivity process, the groups are automatically recalculated. The results of this operation are stored in the vector *groups* and the map *groupMap*.

calculateNorm is called once per evaluation of PROX and EVAL. It calculates the group norms and the group size.

The ElasticNetFunction uses the *LassoFunction* to calculate the l_1 part of its evaluation. Additionally we implemented a *ZeroFunction* that can be used, when no regularization is desired.

Our solver FISTA is then implemented as a class with one template argument: the proximal operator. We therefore have to first create a *RegularizationFunction* and then a *Fista* class. Even though this might seem inconvenient, it allows the compiler to inline all calls to both the function evaluation and the proximal operator, which get called once per iteration. We defined a base class *FistaBase* that allows us to hold a pointer not only to a specialized Fista object, but also to one where we do not know the used *RegularizationFunction*.

FISTA itself is split into various subroutines, that allow a separation of concerns. This leads to a very clean implementation that closely represents **??** in combination with **??**. All calls to the subroutines were inlined automatically, at least for the used GCC-compiler with the highest optimization settings. The *Fista* class has two public methods. We use the first one, called SOLVE, to solve a specific problem instance and the second one, called GETL, to get the last estimated Lipschitz constant. This is useful to avoid a costly grid search after refining the grid, because the Lipschitz constant is usually larger for a larger grid.

3.3.3 Results & Discussion

Friedman1 Dataset

We begin this section by analyzing the implicit feature selection performed by all discussed methods. To do this, it is helpful to use an artificial dataset. In our case we use the Friedman1 dataset, first published in [**datasets-friedman**]. Let $\mathbf{x} = (x_1, \ldots, x_{10}) \in \mathbb{R}^{10}$ be a uniformly distributed vector. We use $\mathbf{x}$ as our predictors, and define

$$y = 10 \sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5 + \varepsilon, \tag{3.19}$$

with $\varepsilon \sim \mathcal{N}(0, 1)$ as additional Gaussian noise. Despite its simplicity, this dataset is very useful for evaluating the regularization methods discussed in this section. Its advantages are:

- The dataset is inherently sparse because five features are not correlated with the response and are left entirely unused.
- It is completely additive with the exception of the $x_1 \times x_2$ interaction term.
- The importance of each variable is directly visible from the definition, which allows us to discuss the selection order. We can see that the contribution of the x_4 term is the largest, followed by x_5, x_3, and finally x_1, x_2 and their interaction.
- We know that the optimal RMSE is equal to the standard deviation of the noise and is hence 1.0. The dataset can therefore be used as a sanity check.

Using a method discussed in [**regularizationpaths**], we can calculate the value of λ for which all weights are exactly zero. The maximum λ is given by

$$\lambda_{max} = \frac{\max_i |\langle \Phi_i, y \rangle|}{\lambda_2 n},$$

where the index i denotes the ith column of the matrix and λ_2 is the amount of l_1 regularization. For the group lasso penalty we have to consider the group structure as well. In practice, the bias term is often the group with the largest weights, in which case we can use the same formula. This is true for the Friedman1 dataset. We then construct a logarithmic grid from λ_{max} to $\lambda_{min} = \varepsilon \lambda_{max}$, where ε is set to 0.001. It is more efficient to start the path with all weights set to zero, see [**regularizationpaths**] for a more advanced discussion.

We calculated regularization paths for the lasso, the elastic net (with $\lambda_2 = 0.3$) and the group lasso using an estimator with level 2 for the Friedman1 dataset.

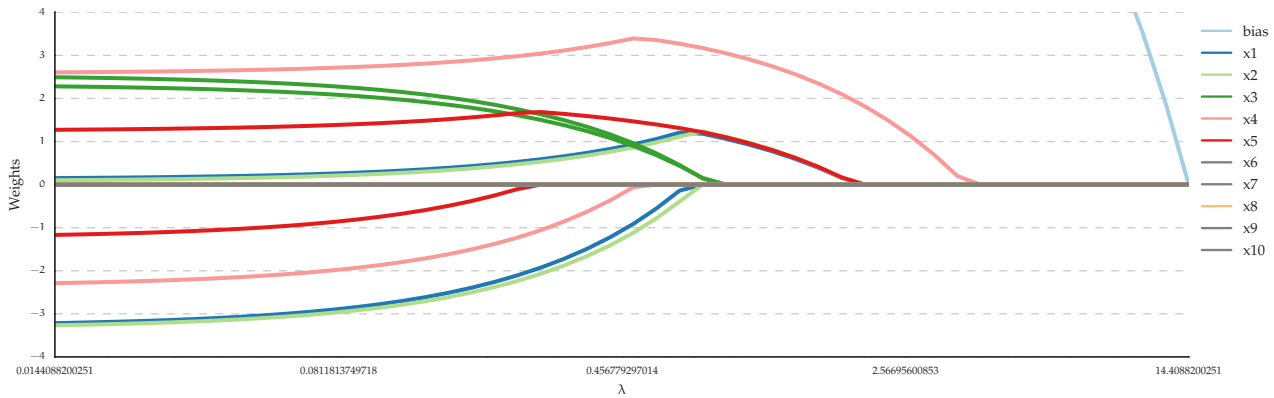


Figure 3.10: Regularization path for the lasso

Lasso path

?? shows the path for the lasso, which first integrated the bias, then one x_4 point, then one each of x_1, x_2, and one x_5 point. The only unneeded term included at λ_{max} was a x_8 point with a very small weight. This agrees roughly with the importance of the terms. Note that it did not select all grid points of a group at the same time. This has the effect that the inclusion of the second term of the same group often leads to a weight decrease of the first included term. In other words the magnitude of a weight might decrease for a smaller regularization parameter.

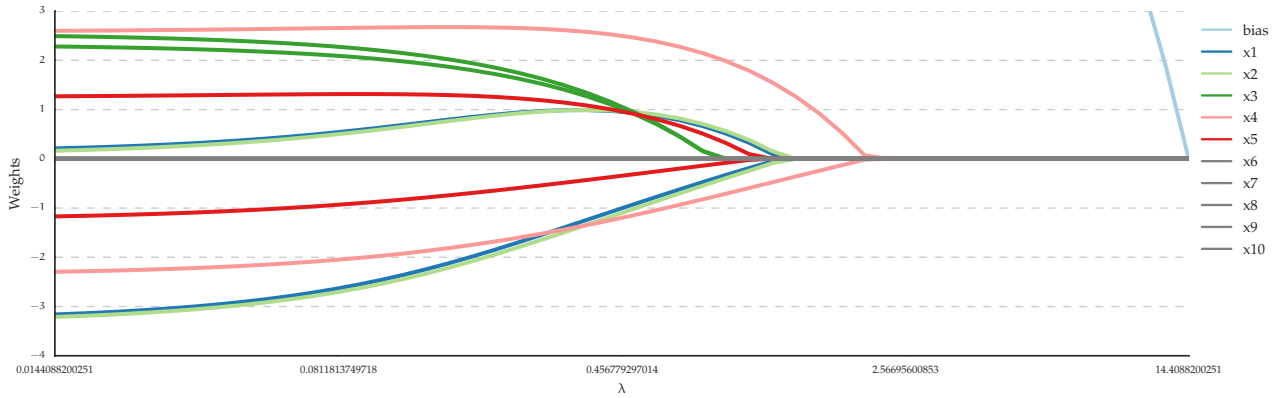


Figure 3.11: Regularization path for the group lasso

Group Lasso Path

The regularization path of the group lasso is depicted by **??**. It performed a more stable grid point selection and all grid points corresponding to the same group were either in the model or not. There was no sparsity on a grid point level but rather on the group level. As with the lasso, the x_4 terms were chosen first, followed by the x_1, x_2 terms at roughly the same time, the final chosen terms were the x_5 and x_3 ones. It did not choose any unneeded point, even at the minimum λ. Interestingly, the weights at λ_{max} for both the grouped and standard lasso were very similar.

Elastic Net Path

The elastic net path, shown by **??**, also selected the bias first. It then selected one point of $x_1, x_2, x_4,$ and x_5 each. One x_3, x_6, x_7, x_8, x_9 and x_{10} terms were selected next, which are all irrelevant features, with the exception of the x_3 point. For the highest chosen λ all grid points were selected. We can see that the elastic net does not perform proper feature selection when the ridge regularization dominates the lasso penalty. The same reasons lead to the effect that the value of most weights were larger than in the lasso estimate, which agrees with the weight prior. The result would look drastically different if a value of λ_2 close to one is chosen, which then approaches the lasso penalty.

Using the same method we calculated the regularization paths for the lasso and the grouped lasso again for the Friedman1 dataset, but this time using a level three

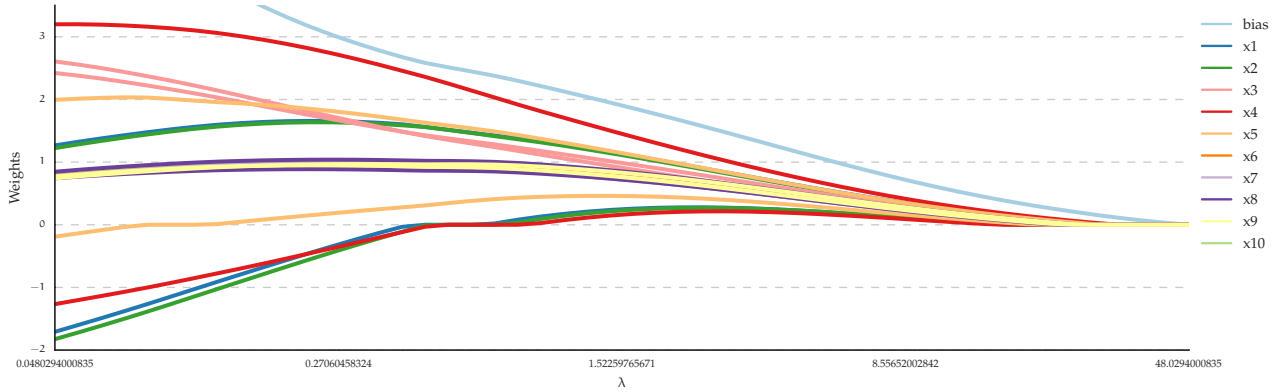


Figure 3.12: Regularization path for the elastic net with $\lambda_2 = 0.3$

estimator. The results are depicted in **??**. We can see that while the group norms were similar, the grid point selection of the group lasso was more stable and selected points on a group level.

Concrete Results

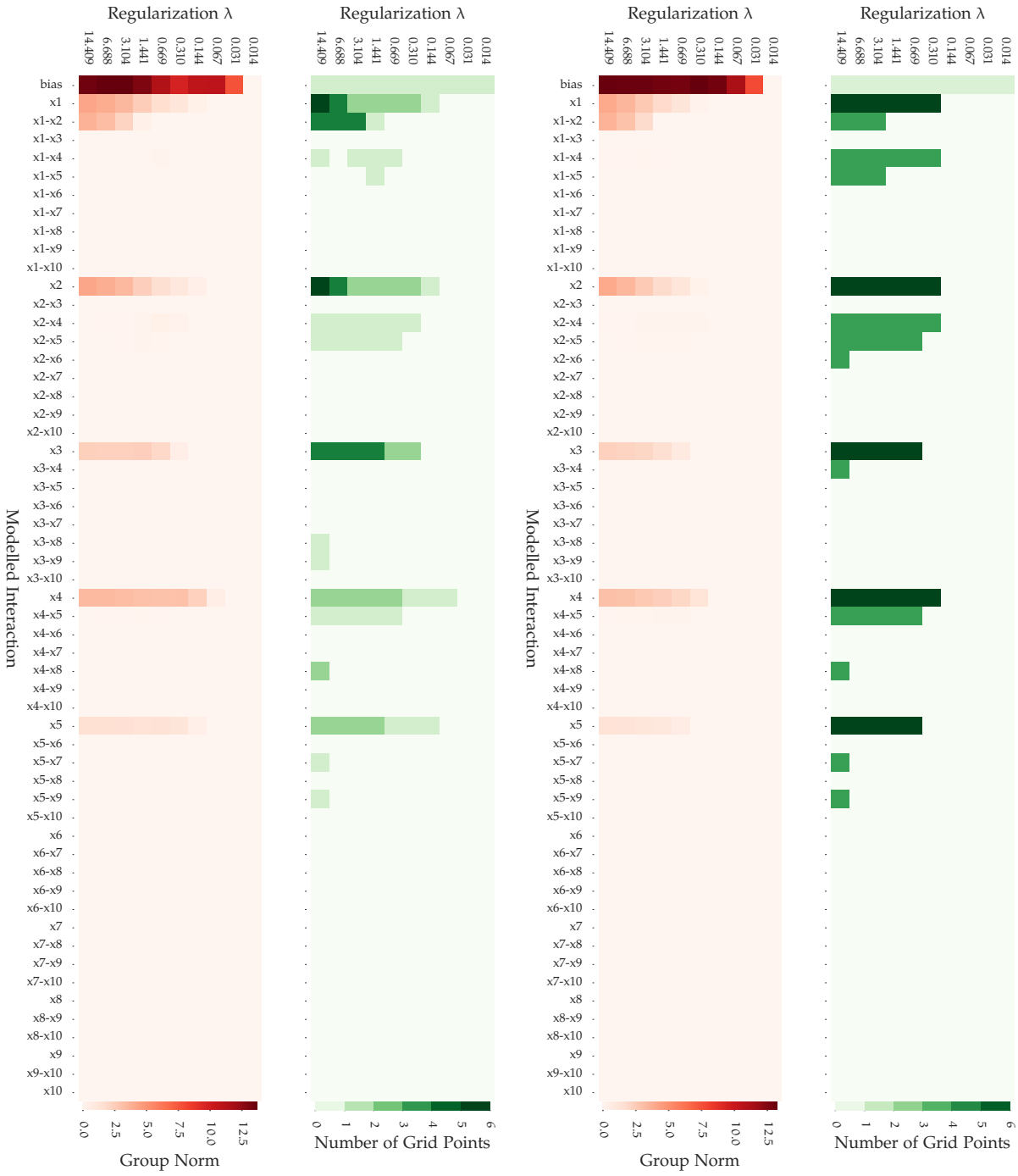
We saw some results that indicated that the feature selection works. The Friedman1 dataset is not a good choice to discuss the real-world performance of our proposed penalties, because it is too simplistic. This is why we return to the concrete dataset. Results for the this dataset can be found in **??**. We performed a grid search over $\lambda \in [10^0, 10^{-1}, \ldots, 10^{-4}]$ again using five refinement steps over three points each.

For the level four grid, the group lasso performed best, followed by the elastic net with $\lambda_2 = 0.95$ and the lasso penalty. We can see that we did not profit from a larger amount of ridge shrinkage, even though a small ridge portion helped to stabilize the result. The results indicate that sparsity on a group level was better than sparsity on a basis-function level. Note that the testing-RMSE was best for the lasso. We have seen that for a medium-sized grid all sparsity-inducing penalties showed solid results.

Does this situation change for a larger grid? The results for the level five grid showed the same tendency. We can see that all methods achieved better results for the larger grid, only the order of the methods is changed. For this experiment the lasso showed the best results, followed by the elastic net with $\lambda_2 = 0.95$ and the group lasso. This makes intuitive sense: A larger grid needs more sparsity and sparsity on the group level is not enough any more. The result for the elastic net is interesting, because we had more grid points than data points. We would expect it to outperform the l_1-regularization, because it should show a more stable feature selection in this case. All sparsity-inducing penalties showed promising results for the larger grid as well.

Akaike information criterion

Instead of comparing the average cross-validation error, we can also use the Akaike



(a) Lasso

(b) Group Lasso

Figure 3.13: Regularization paths for the Friedman1 dataset with level three. Each figure shows two heatmaps. The first one shows the l_2 norm of the group terms, the second one presents the number of selected terms.

Level	Reg. Method	λ	CV-RMSE	Train-Grid	Train-RMSE	Test-RMSE
4	Group Lasso	1×10^{-2}	4.774	1385	3.133	4.032
4	Elastic Net (λ_2 = 0.95)	1×10^{-2}	4.868	1382	2.902	3.869
4	Lasso	1×10^{-2}	4.884	1382	2.911	3.850
4	Elastic Net (λ_2 = 0.5)	1×10^{-2}	4.927	1384	3.077	4.046
4	Ridge	2×10^{-2}	5.007	1470	3.120	4.198
4	Elastic Net (λ_2 = 0.05)	1×10^{-3}	5.060	1367	2.483	3.813
5	Lasso	1×10^{-2}	4.582	6632	2.470	3.737
5	Elastic Net (λ_2 = 0.95)	1×10^{-2}	4.594	6632	2.460	3.742
5	Group Lasso	1×10^{-2}	4.650	6694	2.861	3.955
5	Elastic Net (λ_2 = 0.5)	1×10^{-2}	4.669	6633	2.400	3.844
5	Ridge	2×10^{-2}	4.709	6650	2.286	4.184
5	Elastic Net (λ_2 = 0.05)	1×10^{-2}	4.730	6648	2.297	4.085

Table 3.2: Results for the concrete dataset using different penalties. Entries are ordered by cross validation error.

information criterion (AIC) [**aic**]. It is another way of comparing the relative quality of various models. We can calculate it directly from the MSE

$$\text{AIC}(\text{DF}, \text{MSE}) = 2\,\text{df} + n \ln(\text{MSE}) + c, \tag{3.20}$$

where DF corresponds to the degrees of freedom of the model and n to the number of data points. The constant c depends only on the data and can be safely omitted, because we only compare models for a specific dataset [**esl**]. Note that the AIC is asymptotically equivalent to leave-one out cross-validation [**aic-asymp**].

Effective Degrees of Freedom

We use unbiased estimates of the true effective DFS for the identity matrix and the lasso regularization. We start with the surprisingly simple solution for the lasso penalty

$$\text{df}(\boldsymbol{\Phi}, \boldsymbol{\alpha}) = \text{rank}(\boldsymbol{\Phi}_{\mathcal{A}(\boldsymbol{\alpha})})$$

with

$$\mathcal{A}(\boldsymbol{\alpha}) = \{a \in \boldsymbol{\alpha} \mid a \neq 0\},$$

where $\boldsymbol{\Phi}_{\mathcal{A}(\boldsymbol{\alpha})}$ denotes all columns of $\boldsymbol{\Phi}$ that are in the so-called active set $\mathcal{A}$ and $\text{rank}(\boldsymbol{\Phi})$ corresponds to the rank of the model matrix $\boldsymbol{\Phi}$ [**lasso-df**]. If the model matrix $\boldsymbol{\Phi}$ has full column rank the DFS are given by the cardinality of the active set [**lasso-df**]. The solution for the identity matrix regularization is given by

$$\text{df}(\boldsymbol{\Phi}, \lambda) = \sum_i \frac{\sigma_i^2 - \lambda}{\sigma_i^2},$$

where σ_i represents the ith singular value of Φ [**esl**]. For the sake of brevity, the formulas for our other used penalties are omitted. Unbiased estimates for the elastic net can be found in [**lasso-df**] and for the group lasso in [**grouplasso-df**]. The formula for the diagonal Tikhonov regularization can be derived easily from the general formula presented for example in [**esl**].

Level	Reg. Method	λ	Train-Grid	DF	Train-RMSE	AIC	Test-RMSE
5	Ridge	1.96×10^{-2}	6650	716.7	2.286	2796.22	4.184
4	Lasso	1.00×10^{-2}	1382	518.0	2.911	2797.09	3.850
4	Ridge	1.84×10^{-2}	1470	558.0	3.120	2991.22	4.198
5	Lasso	1.00×10^{-2}	6632	754.0	2.470	2997.94	3.737

Table 3.3: Comparison of AIC for the concrete dataset. All results are ordered by AIC in increasing order.

?? shows the AIC for the best found ridge and lasso models for both level four and five for the concrete data set. All estimators used five refinements on three points each. We can see that the number of degrees of freedom did not increase linearly with larger grid size. This is because we only used about one thousand training data points, which was the limiting factor, rather than the number of grid points. While the ridge penalty used more degrees of freedom for the grid with level four than the lasso, the situation was reversed for the level five grid. The AIC for both methods was comparable, it was smaller for both ridge models by a very small margin.

We can see that all regularization methods managed to perform well for a large grid that has roughly six times more grid points than data points. The CV-errors achieved were better than the best results for both tested Tikhonov regularization methods.

CHAPTER 4

Grid generation

We will discuss methods in this chapter that allow us to impose our prior knowledge directly on the grid generation process. This is a stark contrast to the methods in the previous chapter, in which we applied our constraints during the learning process. Even though these methods work well even with limited prior knowledge, they have one major drawback: We *always* have to generate a complete sparse grid. This was not a problem for low to mid-dimensional datasets, but is a limiting factor for higher dimensional problems. To tackle very-high dimensional problems, we need to be able to influence the grid generation.

High dimensional problems

We will discuss two modifications of the grid generation algorithm in this chapter. The first one, generalized sparse grids, allows us to create grids with variable granularity. The second method, interaction-term aware sparse grids, uses knowledge about the importance of interaction-terms to create grids that are both smaller and more effective.

4.1 Generalized Sparse Grids

Generalized sparse grids allow us to create grids with a variable granularity. They can be used to create smaller grids while retaining the approximation error, given some additional smoothness constraints. They were first developed by **optimizedApproxSpaces** in [**optimizedApproxSpaces**]. A discussion about their usefulness for machine learning can be found in [**sparse-reconstruction**] and in [**sparse-parsimony**].

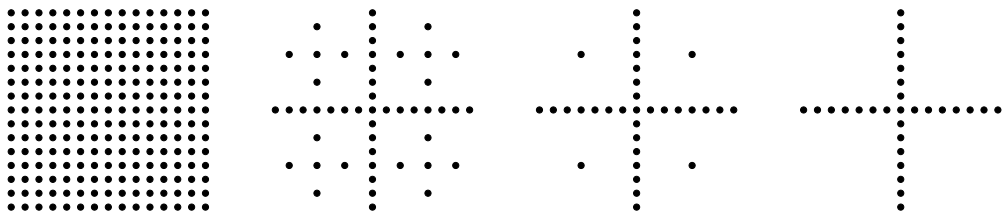
4.1.1 Theory

Despite the fact that the generalized sparse grid technique originates from an elaborate functional analysis argument, they can be stated by two simple formulas. We only need to change **??** to generalize our grid. The set of grid points for the generalized sparse

grid G_n^T of level n and its corresponding approximation space V_n^T is described by

$$G_n^T = \bigcup_{\substack{|l|_1 - T|i|_\infty \\ \leqslant n+d-1-Tn}} G_l, \qquad (4.1)$$
$$V_n^T = \bigoplus_{\substack{|l|_1 - T|i|_\infty \\ \leqslant n+d-1-Tn}} W_l.$$

The constant T chosen in the interval $(-\infty, 1]$ governs our choice of sub-spaces and thus also the granularity of the grid. Setting T to zero recovers the standard sparse grid, higher values approaching one transform the grid to the form seen in **??**. The limit $T \to -\infty$ corresponds to a full grid. Note that even though the value of T is continuous, it acts as a discrete operator. This is why the actual value of T does not matter, different values can result in the same grid. An example for a two-dimensional grid with level four can be seen in **??**.



(a) $T = -\infty$ (b) $T = 0.0$ (c) $T = 0.5$ (d) $T = 1.0$

Figure 4.1: Grid visualization for 2-dimensional grids with different Ts

The dimension of a generalized sparse grid space with level n and constant T can be described by

$$|V_n^T| \leqslant \begin{cases} d2^n & T = 1, \\ O(2^n) & T \in (1/n, 1), \\ O(2^n n^{d-1}) & T \in [0, 1/n], \\ O(2^{\frac{T-1}{T/d-1}n}) & T < 0. \end{cases}$$

We can see that the special cases $T = 0$ and $T \to \infty$ are covered [**optimizedApproxSpaces**]. While it is possible to state a formula for the approximation error, it is quite hard to apply it to machine learning problems, as the smoothness properties of real world datasets are unknown. This means that it is hard to decide whether generalized sparse grids are useful for a given problem without generating a model. We can view the generalized sparse grids from a different perspective as well. A higher value of T decreases

the number of higher-order interaction terms while the number of basis functions that only model one feature is unchanged. The effect for a grid with dimension 4 and level 5 can be seen in **??**. Note that the grid for $T = 1$ does not contain any interaction

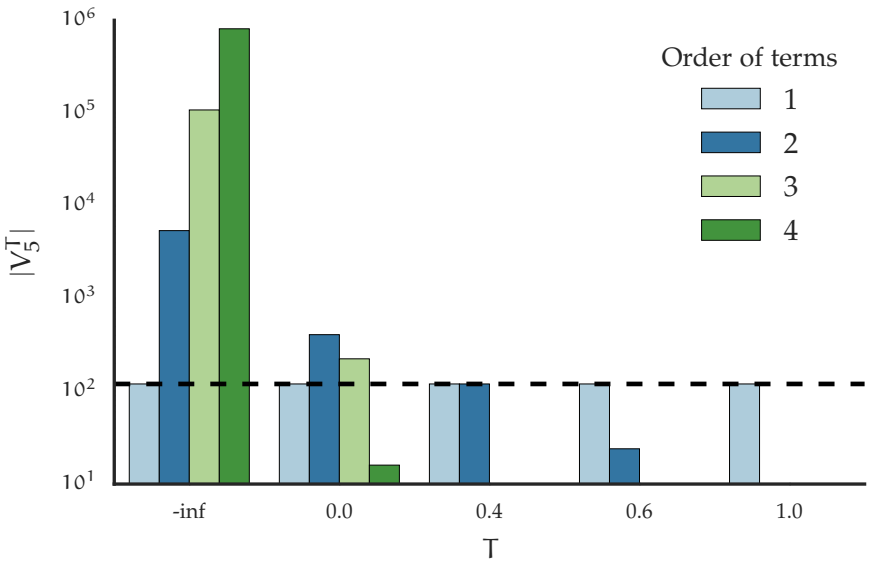


Figure 4.2: Number of terms of each order for a grid with dimension 4 and level 5. The bias term is not included in the graphic, it is contained in all grids.

terms. Smaller values of T increase the number of interaction terms until the full grid is reached. Because the importance of feature interactions is hard to judge for most problems, this heuristic is also difficult to apply.

4.1.2 Implementation

The implementation is quite simple. It was possible to modify the already existing grid generation algorithm to include the changed inclusion criterion described by **??**. The resulting algorithm is given by **??**, lines 11–15 correspond to our changes. We used the functions CreatePoint, which generates a point for a given level and index, and CreatePointAt, which also generates a point, but this time as a child of another point. The distinction between both methods is not important here, because our algorithms do not rely on the hierarchical structure of the grid. Our implementation is a direct translation of the pseudo-code.

Algorithm 5 Generalized Sparse Grid Generation

Input: Number of dimensions, level n and granularity T

```
1: function GridGeneration(dimensions, n, T)
2:     for 0 ⩽ d < dimensions do
3:         CreatePoint(d, 1, 1)
4:     for (l, i) ∈ {(l, i) | 1 ⩽ l ⩽ n ∧ 1 ⩽ i < l^2, i odd} do
5:         CreatePoint(0, l, i)                    ▷ 1d-grid points for first dimension
6:     for d < dimensions do
7:         for p ∈ GetAllGridPoints do
8:             levelSum ← p.getLevelSum - 1
9:             levelMax ← p.getLevelMax
10:            l ← 1
11:            while max(l, levelMax) ⩽ n do
12:                left ← (l + levelSum) − (T · max(l, levelMax))
13:                right ← (n + dimensions − 1) − (T · n)
14:                if left > right then
15:                    break
16:                for 1 ⩽ i < 2^l, i odd do
17:                    CreatePointAt(d, l, i, p)
18:                l ← l + 1
```

4.1.3 Results & Discussion

Generalized grids work well in theory. To show their practical performance, we tested two things that we will discuss in this section:

- In what way does the grid parameter T influence the regularization parameter λ?
- Can we achieve a better performance with generalized sparse grids compared to standard grids using a comparable number of grid points?

Friedman1

We performed 25 iterations of a Bayesian hyper-parameter search for λ for the Friedman1 dataset with identity regularization and a generalized sparse grid for different Ts and level four. Each learner performed five adaptivity steps refining three points each. The results can be seen in **??**. In this case, the best results were achieved for $T = 1$ and the smallest grid size, all other parameters overfit the data. This happened because we used a small version of the Friedman1 dataset—it being an artificially created dataset, it would be easy to create more samples and then fit an arbitrarily large model. We can see from this example that generalized sparse grids allowed us to use a

T	λ	CV-Grid	CV-RMSE	Train-Grid	Train-RMSE	Test-RMSE
−0.5	2.2762×10^{-10}	5541.3 ± 18.5	1.246	5547	0.823	1.226
0	1.4539×10^{-4}	2278.8 ± 3.3	1.196	2277	0.845	1.179
0.5	7.7081×10^{-5}	640.8 ± 14.1	1.051	651	0.959	1.028
1	1.0432×10^{-4}	391.2 ± 7.2	1.031	395	0.976	1.015

Table 4.1: Best results and used λ for different Ts for the Friedman1 dataset and an estimator with level four. The CV-Grid sizes are reported with their standard deviation.

T	λ	CV-Grid	CV-RMSE	Train-Grid	Train-RMSE	Test-RMSE
−0.4	1.9276×10^{-2}	8468.7 ± 20.6	4.703	8470	2.275	4.215
0	1.9622×10^{-2}	6678.3 ± 27.1	4.709	6650	2.286	4.184
0.5	6.2935×10^{-3}	1140.4 ± 23.9	4.771	1180	2.664	3.797
0.6	1.2700×10^{-2}	712.7 ± 24.2	4.781	685	3.398	4.308
1	1.0149×10^{-2}	517.9 ± 23.3	4.929	516	3.628	4.508

Table 4.2: Best results and used λ for different Ts for the concrete dataset and an estimator with level five. The optimal RMSE is 1.0.

higher level, which corresponds to a larger amount of grid points with order one than with standard sparse grids. Additionally the combination with adaptivity allowed us to start with an estimator that only modelled few interaction terms. Needed interaction terms were then created during refinement.

The parameter λ describes the amount of regularization per grid point. We can see no trend in the results for the Friedman1 dataset for λ. Only the value for $T = -0.5$ was smaller by some orders of magnitude. A possible reason for that might be the same reason the sparse grid with $T = 1$ showed the best result: Except for the $(x_1 \times x_2)$ interaction, the Friedman1 dataset has no qualitative interactions, i.e. interactions that are not inherently additive in effect. This implies that the additional interaction points for larger grids would have a relatively low surplus, even for an estimator fit without regularization. The higher-order grid points thus need a smaller amount of regularization than the first-order terms, which explains the small λ for the largest grid. All other results had regularization parameters that were of similar order, the differences were significant. The Friedman1 results demonstrated that generalized grids can help us to use grids with a level that would lead to severe overfitting for normal sparse grids.

We used a similar experiment for the concrete dataset, this time performing 45 *Concrete*

Bayesian search iterations and using estimators with level five. For this dataset the chosen level did not lead to overfitting even for the highest value of T and we can therefore use it to discuss the trade-off between approximation accuracy and grid size our proposed method makes. The results can be seen in **??**.

Again, the values of λ did not change significantly for different Ts. Note that there was a correlation between grid sizes and the errors: larger grids performed better, at least for the cross validation and training error metrics. A further increase of the level of the approximation space could soon lead to overfitting. Even for our chosen level, we have more grid points than training examples in the $T \geqslant 0.5$ cases, which lead to an underdetermined linear system. We can also see that the decrease in error between the largest grid and the standard sparse grid was small, considering the amount of additional grid points needed. Note that the estimator with $T = 0$ had a higher CV- and train-RMSE than the learner with $T = -0.5$, but a lower testing error. This and the fact that the differences are small lead to results for which it is hard to decide, which choice of T performed best. It is therefore reasonable, to choose the simpler model. Because of that we can see that the trade-off between error and discretization cost, which the generalized sparse grids make, worked. They used fewer grid points to achieve similar errors.

If we compare the performance of the generalized sparse grid with level five and $T = 0.5$ with the standard sparse grid of level four with the same adaptivity settings, we can see that the generalized grid performed better than the standard grid, even though they both used a similar number of grid points. The standard grid with level four needed 1470 grid points for a CV-RMSE of 5.007, which means that it needed more grid points for a worse performance.

Diagonal Regularization

As an additional result, generalized sparse grids combined with the diagonal matrix regularization penalty showed further potential. An example for this can be seen in **??**, which depicts a grid search similar to the one in **??**, only using a level five grid with $T = 0.5$ instead of a level four grid with $T = 0$.

We can conclude that our proposed method can improve the performance without additional cost. Because the smoothness constraints for the generalized grids are stronger than the assumptions of the standard grid, this result does not have to hold for all datasets. Even though its performance depends on the relevance of the higher-order-terms, it is hard to predict whether our proposed method will show good results. The reason for this is that it is difficult to tell if interactions are relevant for a given dataset in the general case. Therefore it seems useful and necessary to build multiple models with different grid types and thus include the granularity of the grid into the model selection process.

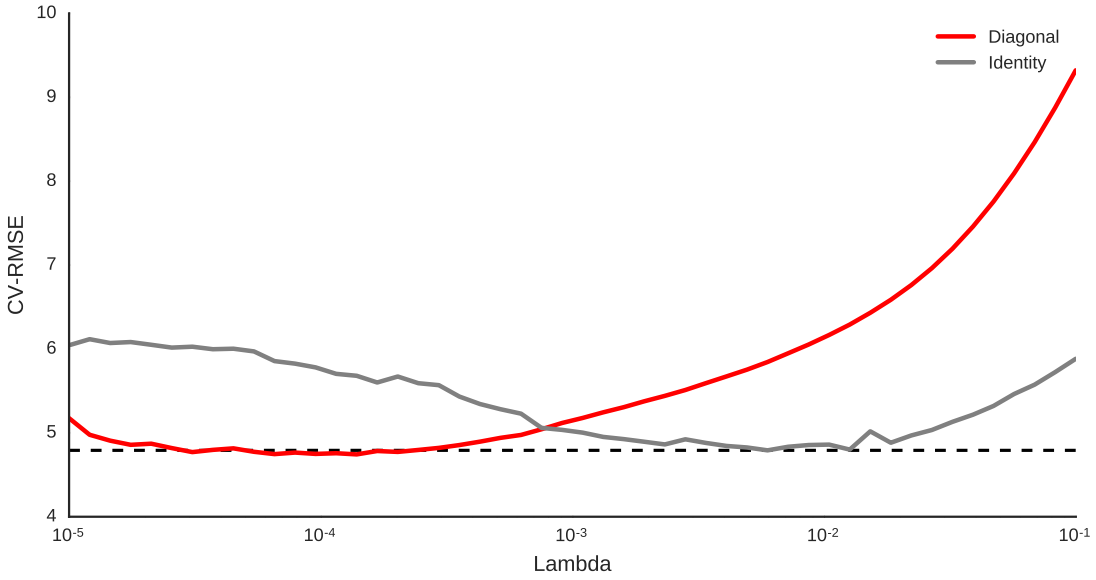


Figure 4.3: Obtained results for the concrete dataset with estimators for level five and $T = 0.5$.

4.2 Interaction-Term Aware Sparse Grids

As previously noted, sparse grids not only include grid points, which model an original feature, but also interaction points. We have seen that sparsity-inducing penalties perform automatic feature selection. In some cases we are able to make an educated guess, which interactions are relevant, before actually training a model. We will introduce a method in this section that allows us to create interaction-term aware sparse grids, i.e. grids that only contain a subset of all possible interaction terms. We will also present an application for this method: image recognition. A discussion of this method can be found in [**sparse-parsimony**].

4.2.1 Theory

No. of Interaction Terms

We can calculate the number of included terms for a d-dimensional dataset modelled by an sparse grid of level l using simple combinatorics:

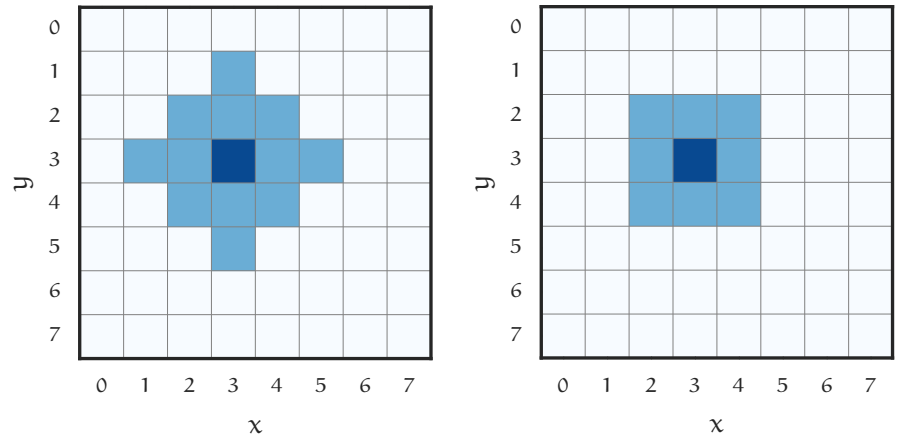
$$\text{count-terms}(d, l) = \sum_{k=0}^{\max(d,l-1)} \binom{d}{k}.$$

Because the number of grid points is directly related to the number of chosen interaction terms, it is clear that the standard sparse grid technique is computationally expensive

or infeasible for very high dimensional problems. This means that we have to restrict ourselves to a small level and are therefore limited to lower-order terms. If we only include interactions between some variables we can use larger levels without increasing the number of interaction terms to an intractable number.

Nearest Neighbors

An example where this technique is useful is image recognition. Assume we have a 2-dimensional picture of which each pixel corresponds to a feature. We make the following assumption: Interactions between pixels that are close to each other spatially are more important than interactions between pixels that are further away from each other.



(a) $d(\mathbf{a},\mathbf{b})_{\|\cdot\|_1} \leqslant 2$ (b) $d(\mathbf{a},\mathbf{b})_{\|\cdot\|_2} \leqslant \sqrt{2}$

Figure 4.4: Nearest Neighbors for two different metrics. The darkest point is the center of the neighborhood, the other colored points are its neighbors.

Let d be a metric that measures the distance between two pixels. Examples for widely-used metrics are

$$\begin{aligned} d(\mathbf{a},\mathbf{b})_{\|\cdot\|_2} &= \sqrt{(a_1-b_1)^2+(a_2-b_2)^2}, \\ d(\mathbf{a},\mathbf{b})_{\|\cdot\|_1} &= |a_1-b_1|+|a_2-b_2|, \end{aligned} \tag{4.2}$$

which are called the Euclidean and the Manhattan distance respectively. A visualization of both metrics can be seen in **??**. We can then calculate the nearest neighbors of each pixel by iterating over all other pixels and checking whether the metric is below a certain threshold. To calculate all neighbors, we simply iterate over each pixel. This is not an asymptotically optimal algorithm; we pay $O(n^2)$ for all features. Our used method is given by **??**. Even though more efficient algorithms exist, our method is good enough, because we only have to calculate the neighbors once per dataset, which is negligible

compared to the cost of training the actual model. After calculating the neighbors, we can generate the interactions from them. They are given by all $i \leqslant \max(d, l-1)$ long combinations of all possible neighbors for each pixel, where i is an arbitrarily chosen value. This means that it is possible to create grids with a high level without some higher-order terms. For example, it is possible to create a grid with level six, but only use the interactions up to order four. In fact, it can be useful to start with a low-level grid but to use higher-order interaction terms during refinement. This recovers the usual behavior of non-interaction-term aware sparse grids.

Algorithm 6 Nearest Neigbors

Input: Set of all pixels p, distance metric $d\colon (\mathbb{R}^2, \mathbb{R}^2) \to \mathbb{R}$, threshold t

```
1: function NearestNeighbors(p, d, t)
2:     neighbors ← vector<vector<int>>()
3:     for a ∈ p do
4:         curNeighbors ← vector<int>()
5:         for b ∈ p do                    ▷ Each pixel is its own neighbor in our case.
6:             if d(a, b) ⩽ t then
7:                 append(curNeighbors, b)
8:         append(neighbors, curNeighbors)
9:     return neighbors
```

4.2.2 Implementation

Similar to the implementation of the generalized sparse grids we need to make some small adjustments to the grid generation algorithm described in **??**. We pass an additional parameter to the function GridGeneration that determines the interaction terms we want to integrate into the model. This parameter is a list of interactions, each interaction is modelled as a list of dimensions that should interact with each other. The list is then converted to a hash set that stores one boolean vector for each interaction. Each entry of this vector is true if the dimension should be used and false otherwise. For example, a $(x_1 \times x_2)$ interaction for a 3-dimensional dataset is modelled as the vector $(1, 2)$ and the corresponding boolean vector is then given by (*true*, *true*, *false*).

We then only needed to modify the function createPointAt (called in line 17). Each possible new point is encoded in the same manner as the interactions: as a boolean vector. Before creating the grid point, we check whether the new grid point models a desired interaction. This was implemented by checking if the encoded coordinates are contained in the hash set. Only after a successful check, the grid point is actually

created.

We had to make the same adjustments to the adaptivity procedure. Our implementation is simple, because we leverage the existing adaptivity procedure of the *SG++* implementation. We implemented a class *HashRefinementInteraction* that inherits from the base class *HashRefinement*. This base class implements the actual refinement procedure, the subclass leaves all but one function unchanged. The function CREATEGRIDPOINT is in called in the base class to create the new grid points. We modified this function in a straight forward way: Before each point is added to the model, we perform the same check as in the method CREATEPOINTAT.

This implementation strategy is efficient due to the constant usage of optimized data structures. It costs us $O(1)$ operations to check if an element is contained in the set, which is both asymptotically optimal and efficient in practice. The boolean vector in the *C++* standard library is implemented as a bitfield, which results in a lower space overhead.

The neighborhood generation follows the description of this chapter. We implemented a method CALCULATEINTERACTIONS, that first uses **??** and then returns all resulting interactions. It accepts a set of points, a metric, a threshold and a maximum-order as its arguments.

4.2.3 Results & Discussion

Optical Digits

To check the validity of the interaction-term aware sparse grids and of the nearest neighbor approach for images, we used a version of the classical MNIST-dataset, obtained from the UCI machine learning repository [**datasets-uci**]. The goal of this dataset is to use hand drawn pictures of single digits to classify the depicted digit. Our version of the dataset is composed of 64-features, each one representing one gray-scale pixel in the range 0–15. A visualization of the digits can be seen in **??** and a two-dimensional representation of the dataset is depicted in **??**.

Trying to construct a sparse grid for such a highly dimensional dataset is possible for small levels and becomes highly intractable for larger levels. This is why we need to exclude most of the interaction terms. We used **??** to select the neighbors for each pixel, and only included the resulting interactions in our grid. This was done using the Euclidean distance, given by **??**, with a threshold of $\sqrt{2}$, which leads to 3×3 neighborhoods, as shown in **??**. A comparison of the resulting grid sizes for the used grid and the standard sparse grid is shown in **??**. Note that we have more basis functions than training points for a level greater than two for the standard grid. The usage of interaction-term aware grid generation delays that development to level three.

Subsampled Dataset

Because we have to train one regression learner for each class it takes a rather long time to generate a model, which in practice meant that a grid search for the regularization parameter was infeasible. This is why we tried to find an approximation

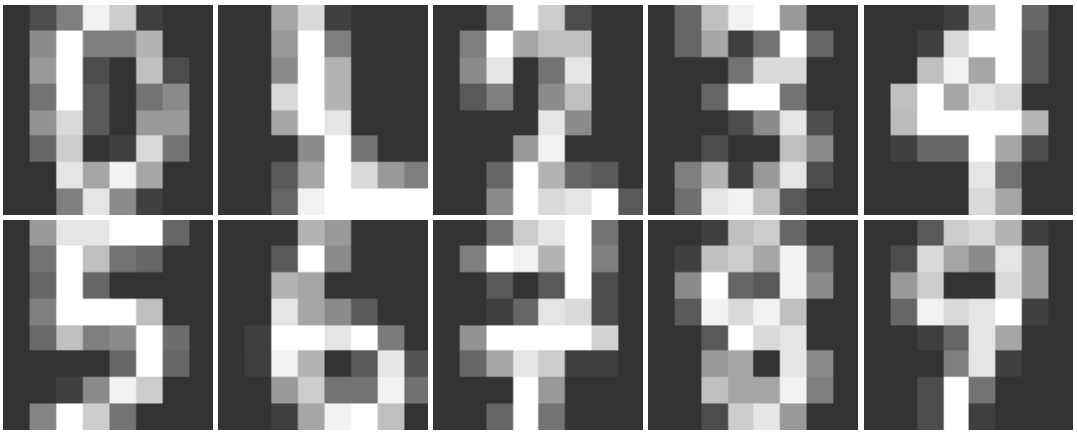


Figure 4.5: Some examples for digits

by using only a three-class subset of the dataset for this purpose. We selected all entries for the digits 2, 7 and 9. The smaller dataset comes with two advantages. Firstly, we only had to use three regression models for each classification task and secondly, the training of each model was considerably faster, because the sparse grid model scales linearly with the number of training points. The downside is of course that the estimated best λ is only a crude approximation of the optimal one. Because our model assumes that each binary sub-classification model uses the same hyper-parameters, there is some variance of the best-parameter to consider. The discrete nature of the decision problem also allows us some leeway. Altogether our approximation of the hyper-parameter might not be entirely optimal, but it is sufficient.

Estimating λ

Finally we performed a grid search for λ using a ridge regularized model with level three and the aforementioned choice of interaction terms. We used a three-fold stratified cross validation metric to compare the models. The best learner achieved an CV-accuracy of 100% on this subset with $\lambda = 0.1$. Of course, this is not a good estimate for the error on the complete dataset but still delivers a solid estimate for the regularization parameter. The results also showed that the choice of λ does not influence the validation accuracy heavily, as a learner with $\lambda = 10^{-12}$ achieved a CV-accuracy of about 99.91%. We got the same estimate for a larger (5×5) neighborhood.

Models

We created models for level two and three, and compared different interaction-term inclusion criteria. All of our models used $\lambda = 0.1$ and no adaptivity. The results can be seen in **??**. Because a sparse grid for level two does not contain any interaction terms, it is not useful to apply our proposed modified grid generation algorithm here. A standard sparse grid learner achieved an accuracy of 92.77% with only 129 grid points. Increasing the level to three and including all interactions in a 3×3 grid resulted in a

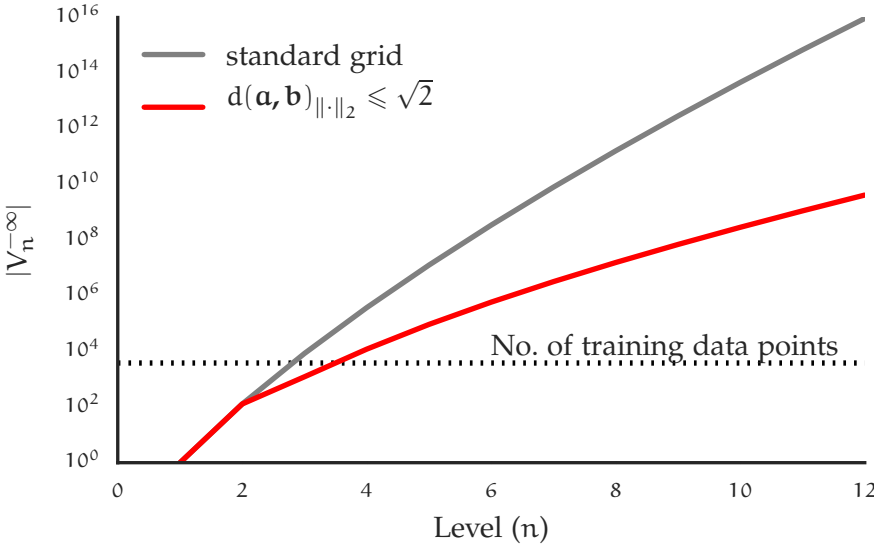


Figure 4.6: Comparison of grid sizes with standard and interaction-term aware sparse grids for the optical digits dataset.

Sparse Grid Method	Level	Neighbors	Train-Grid	Test-Accuracy[%]
Standard	2	all	129	92.77
Interaction-Aware	3	$d(\mathbf{a},\mathbf{b})_{\|\cdot\|_2} \leqslant \sqrt{2}$	1225	97.33
Adaptive [**spatAdaptGrid**]	2	all	1760	97.74
Interaction-Aware	3	$d(\mathbf{a},\mathbf{b})_{\|\cdot\|_2} \leqslant 2\sqrt{2}$	2569	97.83
Standard	3	all	8449	98.22

Table 4.3: Accuracy of sparse grids models for the optical digits dataset.

test accuracy of 97.33% percent and 1225 grid points. **spatAdaptGrid** used a learner that started with a level two grid and then performed aggressive refinement until a grid with 1760 grid points was reached [**spatAdaptGrid**]. This adaptive grid resulted in an accuracy of 97.74%. Our final estimator used a grid with 2569 grid points with a level three interaction-term aware grid, where we included all pair-wise interaction between pixels that are inside a 5×5 grid. This model achieved an accuracy of 97.83%. Additionally, we trained a standard sparse grid learner with level three, which used 8449 grid points and reached an accuracy of 98.22%.

We saw that our model improved Pflüger's result without using adaptivity. Because we only used a small amount of training data for such a highly-dimensional datasets, refinement steps starting from level three soon lead to severe overfitting. Nonetheless,

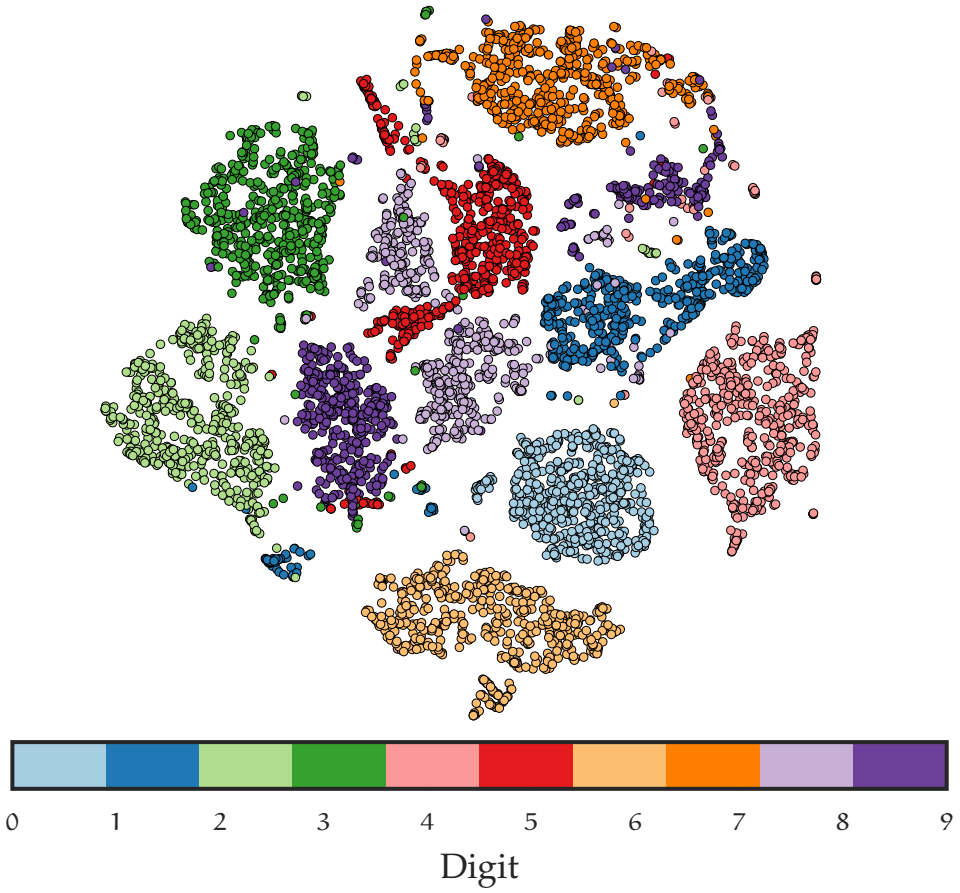


Figure 4.7: T-SNE [**tsne**] plot of the optical digits dataset.

our method managed to reach promising results that were highly competitive with the reference model and beat the level two model by a large margin. It can be expected that our proposed method would be able to improve its performance further if we used more training data or a stricter regularization method, such as one of the sparsity-inducing penalties described in **??**. In comparison to the largest model, our learner is a reasonable trade-off between accuracy and cost. We cannot use a standard sparse grid for even higher dimensional datasets, this is why we need to use a simpler model such as interaction-term aware sparse grids or adaptive grids.

CHAPTER 5

Conclusion

Over the course of this thesis we discussed many useful ways to use prior knowledge to enhance the capabilities of the sparse grid model for machine learning. The main results can be summarized as follows:

- We presented an improved Gaussian prior in **??**. The diagonal Tikhonov functional used only information about the smoothness of the dataset, which applies to many real world data sets and is therefore a rather mild assumption. After we convinced ourselves that the prior indeed improved the performance for data, which adhered to our assumptions, we tested the penalty on non-artificial datasets.

  We achieved solid results for the concrete and the power plant datasets. In case of the power plant dataset, we have been able to double the effect regularization had on the RMSE.

- In **??** we presented three sparsity-inducing penalties: the lasso, the elastic net and the group lasso. All methods used different prior knowledge. The lasso resulted in a Laplace prior, which the elastic net combined with a Gaussian prior to achieve better theoretical performance for correlated predictors. We also showed an adaptation of the group lasso to sparse grids and presented a way to group the grid points. This allowed us to impose sparsity on a original feature level, and not on a grid point level.

  We were able to improve upon the performance of the ridge regularization for the concrete dataset using high levels. The resulting degrees of freedoms were similar for both the lasso and the ridge regularization for the concrete dataset.

  Finally we implemented and discussed an optimal first-order solver FISTA that was able to optimize the non-differentiable penalties.

- We showed in **??** that generalized sparse grids can be applied to machine learning problems. This method can be used to control the granularity of the grid and thus helped us to use a larger level. More importantly, we presented intuition why the method works for data mining problems: Generalized grids change the number

of interaction points, while keeping the number of basis functions that model only one feature constant. We were able to show that this method helped us to avoid overfitting for the Friedman1 dataset.

The results for the concrete dataset were twice-fold useful. Firstly, we showed that a grid that is closer to a full grid can achieve better results than an ordinary sparse grid. Secondly, we showed that a generalized sparse grid can still retain most of its accuracy while vastly reducing the number of grid points.

- The effectiveness of interaction-term aware sparse grids was shown in **??**. This technique allowed us to specify which interactions we want to include in the model. We showed how this can be used to decrease the number of grid points and thus allowed us to cope with a 64-dimensional dataset.

  The relevant interactions were chosen by a nearest-neighbor approach, where we selected interactions only between pixels that were spatially close to each other. The results for the optical digits dataset were competitive with an adaptive sparse grid and presented an useful trade-off between error and cost.

We have seen that it is indeed useful to use information about datasets to improve the performance of the sparse grids technique. In some cases, our prior knowledge only consisted of very mild assumptions that should apply to most datasets. These methods can therefore be used for all-purpose learning and are not limited to a certain problem category. All methods lead to competitive results and were able to create more efficient grids.

The methods contained in this thesis are only a small subset of possible ways how prior knowledge can improve the performance of learners. Nevertheless, they represent a useful selection of flexible methods.

APPENDIX A

Datasets

In this appendix, we look at all used datasets and document the pre-processing steps. We did not perform any pre-processing for datasets that were created "on the fly", as they were created with the sparse grid learner in mind.

Each feature of each dataset was scaled to the range $[0, 1]$ with the so called min-max scaler

$$\text{scale}(\boldsymbol{x}) = \frac{\boldsymbol{x} - \min(\boldsymbol{x})}{\max(\boldsymbol{x}) - \min(\boldsymbol{x})}.$$

Because some features of some datasets are distributed in a non-optimal way, we performed a Box–Cox transformation for them

$$\text{Box–Cox}(x_i) = \begin{cases} \ln(x_i) & \text{for } \lambda = 0, \\ x_i^\lambda - 1 & \text{otherwise,} \end{cases}$$

where λ is a parameter that was estimated by maximizing the log-likelihood [**box-cox-trans**]. Note that we need to shift the data before applying this transformation by a small positive number because $\ln(0)$ is undefined.

We used the following datasets:

Name & Reference	No. of predictors	No. of instances	Scaled	Box–
Friedman1 [**datasets-friedman**]	10	10 000	✗	?
Concrete [**datasets-concrete**; **datasets-uci**]	8	1030	✓	?
Power Plant [**datasets-powerplant**; **datasets-uci**]	4	9568	✗	?
Optical Digits [**datasets-uci**]	64	5620	✓	✗

Table A.1: Datasets used for this thesis.

The used parameters for the Box–Cox transformation can be found in the following tables.

Predictor	λ
Water	0.806 605
Fly Ash	−1.994 271
Fine Aggregate	1.605 275
Blast Slag	−2.718 958
Age	−7.246 132
Coarse Aggregate	1.000 000
Compressive Strength	1.000 000
Cement	−0.683 939
Superplasticizer	−1.491 446

Table A.2: Box–Cox parameters for the concrete dataset.

Predictor	λ
x_1	0.737 723
x_2	0.806 982
x_3	0.738 149
x_4	0.659 072
x_5	0.886 650
x_6	0.781 566
x_7	0.737 074
x_8	0.775 896
x_9	0.803 518
x_{10}	0.817 283

Table A.3: Box–Cox parameters for the Friedman1 dataset.

Predictor	λ
Ap	0.000 000
V	−0.439 899
Rh	2.460 495
At	1.335 384
Pe	1.000 000

Table A.4: Box–Cox parameters for the power plant dataset.

Bibliography

[BG04] H.-J. Bungartz and M. Griebel. "Sparse grids." In: *Acta numerica* 13.1 (2004), pp. 147–269.

[BT09] A. Beck and M. Teboulle. "A fast iterative shrinkage-thresholding algorithm for linear inverse problems." In: *SIAM journal on imaging sciences* 2.1 (2009), pp. 183–202.

[PB14] N. Parikh and S. P. Boyd. "Proximal Algorithms." In: *Foundations and Trends in optimization* 1.3 (2014), pp. 127–239.