DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Integration of Prior Knowledge for Regression and Classification with Sparse Grids

Lukas Krenz

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

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:	Prof. Dr. Bungartz
Advisor:	Valeriy Khakhutskyy
Submission Date:	July 5, 2016

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

Munich, July 5, 2016 Lukas Krenz

Contents

1 Introduction

The goal of this thesis is to introduce and evaluate different ways to integrate prior information into our data mining procedure.

We make the following contributions:

- In chapter X we evaluate different regularization methods, which help us to solve ill-conditioned problems.
- In chapter Y we evaluate different grid construction methods.
- In

2 Learning with Sparse Grids

2.1 Sparse Grids

Sparse grids is a discretization technique that originates from numerical partial differential equations. They have many applications for various problems. We concentrate here on regression.

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

Cite Sparse Grids basics.

$$\varphi(x) = \max(1 - |x|, 0). \tag{2.1}$$

We now define a one-dimensional basis function for a level l and an index i by

$$\varphi_{l,i} = \varphi(2^l x - i). \tag{2.2}$$

The basis functions are centered on a grid point, and have support of .... The d-dimensional basis functions are given by the tensor product

$$\varphi_j(x_1, \ldots, x_d) = \prod_{k=1}^{d} \varphi_{jk}(x_k). \tag{2.3}$$

Let $\varphi_i(\mathbf{x})$ be a family of basis functions. We now define

$$\boldsymbol{\phi}(\mathbf{x}) = \begin{pmatrix} \varphi_1(\mathbf{x}) \\ \vdots \\ \varphi_m(\mathbf{x}) \end{pmatrix}, \quad \boldsymbol{\Phi}(\mathbf{x}) = \begin{bmatrix} \boldsymbol{\phi}(x_1)^T \\ \vdots \\ \boldsymbol{\phi}(x_n)^T \end{bmatrix}, \tag{2.4}$$

which we use to express our regression formula

$$\hat{y} = \sum_{j=1}^{m} \alpha_j \varphi_j(\mathbf{x}) = \boldsymbol{\alpha}^T \boldsymbol{\phi}(\mathbf{x}) \tag{2.5}$$

as a weighted sum of the basis functions.

We split out d-dimensional basis functions as follows using a tensor product over several one dimensional functions

$$\varphi_j(x_1, \ldots, x_n) = \prod_{k=1}^{d} \varphi_{jk}(x_k). \tag{2.6}$$

We can now formulate our goal as an optimisation problem of the form

$$\min_{\alpha} \|\Phi\alpha - y\|_2^2 + \lambda S(\alpha), \tag{2.7}$$

with $S(\alpha)$ as a regularisation operator and λ as a constant. The standard regularisation method is the L2 regularisation

$$S(\alpha) = \|\alpha\|_2^2. \tag{2.8}$$

From a Bayesian perspective it resembles a Gaussian prior on the weights with mean 0 and a constant variance and is therefore distributed as follows

$$\alpha \sim \mathcal{N}(0, I\sigma^2). \tag{2.9}$$

2.2 Supervised Learning: Regression & Classification

Consider ...

Add concrete example for Regression.

Let the set

$$T = \{(x, y) \subset \mathbb{R}^D \times y\}$$

Insert definition for training set.

be a set of labeled examples with x as predictors and y as the target variable. This set is called training set. We assume that the examples are independent and drawn from the same probability distribution. Our goal is not interpolation, 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, that is a function that allows us to make accurate predictions for yet unseen, unlabeled data. In other words we want to learn the structure of the data set.

We differentiate between

Regression if we want to predict a continuous y and

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

Our procedure also works, if the examples are dependent. The assumption makes the analysis easier.

3 Regularization Methods

During the discussion of Equation 2.7 we have neglected the regularization operator. Regularization allows all to solve illconditioned problems by adding a smoothness term. This term allows us to limit our solution space to certain solutions, depending on our choice of the regularization functional. We can impose priors on the grid point weights, that resemble prior information we have about the function we want to approximate. Even missing prior information doesn't hinder us, we then have to evaluate different methods.

In this chapter we are going to compare multiple choices and evaluate their effectiveness.

3.1 Tikhonov Regularization

We start with Tikhonov regularization, which is also called ridge regression in statistics. Instead of solving an unconstrained minimization problem, we solve the following problem:

$$\begin{aligned} &\text{minimize} && \|\mathbf{\Phi}\boldsymbol{\alpha} - \mathbf{y}\|_2^2 \\ &\text{subject to} && \|\mathbf{\Gamma}\boldsymbol{\alpha}\|_2 \leqslant \mathfrak{l}, \end{aligned} \tag{3.1}$$

for a certain constant $\mathfrak{l}$ and a linear operator $\mathbf{\Gamma}$. By doing this we force our (scaled) weights to lie inside a (d-dimensional) sphere with a diameter of length $\mathfrak{l}$. To get a more convenient representation of Equation 3.1 we use Lagrange multipliers to cast it into the equivalent Langrangian form

$$\mathcal{S} = \lambda\|\Gamma\boldsymbol{\alpha}\|_2^2. \tag{3.2}$$

There is an one-to-one correspondence between $\mathfrak{l}$ in Equation 3.1 and λ in Equation 3.2. Both variables determine our constraints.

Explain what these constants do exactly.

A common choice for Γ is the identity matrix as proposed in [**SpatAdaptGrid**]. From a Bayesian perspective this operator resembles a Gaussian prior on the weights with mean 0 and a constant variance. The weights are therefore distributed as follows

Laplace?

$$\alpha \sim \mathcal{N}(0, \mathbf{I}\sigma^2). \tag{3.3}$$

We assume that all weights are distributed with the same variance.

We construct an alternative matrix as follows:

$$k = |l|_1 - d + 1 \tag{3.4}$$

$$\Gamma_{i,i} = |\tfrac{1}{4}|^{k-1}. \tag{3.5}$$

This construction originates from [**bungartzSparse**].

3.2 Lasso & Elastic net

Another possible regularization functional is the so called Lasso regularization. We can represent this procedure in a form similiar to Equation 3.1:

$$\text{minimize} \quad \|\Phi\alpha - y\|_2^2 \tag{3.6}$$

$$\text{subject to} \quad \|\alpha\|_1 \leqslant c. \tag{3.7}$$

Again, we minimise the Lagrangian of Equation 3.6 given by:

$$\mathcal{S} = \lambda\|\alpha\|_1,$$

which uses the Manhattan norm defined by $\|x\|_1 = \sum_i |x_i|$.

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. Another useful perspective is the geometric one. Lasso restricts our possible solutions to a hypercube with lenghts l. Both viewpoints lead to the same intutive result — solutions where one variable is excactly zero are more likely to occur for the Lasso functional. This leads to sparsity, i.d. weight vectors with many entries that are zero.

Because the absolute value function is not differentiable at 0, we cannot use optimization methods which rely on the existence of the gradient such as conjugated gradients. We must use a proximal gradient method, for example FISTA.

3.2.1 FISTA

A possible solver is FISTA, first introduced in [**fista**]. The discussion of FISTA follows [**fista**], the description of the proximal operators follows [**proxsurvey**]. FISTA is able to minimize functions that can be expressed as a sum of a convex, smooth function f and another convex, possibly non-smooth function g:

$$F(\alpha) = f(\alpha) + g(\alpha). \tag{3.8}$$

It is required that the function $f(\alpha)$ has a Lipschitz-continuous gradient. This means that the following condition has to be valid for all possible vectors x and y for some constants L:

$$\|\nabla f(x) - \nabla f(y)\| \leqslant L\|x - y\|. \tag{3.9}$$

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

In our case we set f to

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

The gradient of f is then given by

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

The Lipschitz constant for the gradient of f is

$$L_{\nabla f} = [\sigma_{\max}(\boldsymbol{\Phi})]^2, \tag{3.10}$$

where $\sigma_{\max}$ corresponds to the maximum singular value. Additionally we set g to

$$g(\boldsymbol{\alpha}, \lambda) = \mathcal{S}(\boldsymbol{\alpha}, \lambda) .$$

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

$$M_g(\boldsymbol{\alpha}, \lambda) = \inf_x \left\{ g(x) + (2\lambda)^{-1} \|x - \boldsymbol{\alpha}_2^2\|_2^2 \right\} . \tag{3.11}$$

It is a regularized, smooth version of our function $g(\boldsymbol{\alpha})$ that has the same minimum as $g(\boldsymbol{\alpha})$. That means, that every point that minimizes $M_g(\boldsymbol{\alpha}, \lambda)$ also minimizes our original function g [**proxsurvey**]. Although Equation 3.11 offers a convenient alternative version of our function, the infinum may be hard to calculate. Luckily we can define an operator that returns the infinum point, which is called the proximal operator:

$$\mathbf{prox}_g(\boldsymbol{\alpha}, \lambda) = \operatorname*{argmin}_x \left\{ g(\mathbf{x}) + (1/(2\lambda)) \|\mathbf{x} - \boldsymbol{\alpha}\|_2^2 \right\} . \tag{3.12}$$

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

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

We can use Equation 3.12 to minimize M_g and thus also for optimizing g. In the most general case Equation 3.12 would imply the need to solve a convex optimization problem. Fortunately we can find closed form solutions for many functions. For example consider $g(\boldsymbol{\alpha}) = 0$. In this case the proximal operator is trivial, it is given by the identity function

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

It is obvious, that the minimum of M_g is equivalent to the minimum of $g(\boldsymbol{\alpha})$, the proximal operator is clearly identical to a gradient step on M_g.

We define an approximation of $f(\alpha) + g(\alpha)$ for a given $L > 0$ at a point y as

Fix format of ub

$$Q_L = \underbrace{f(y) + \langle \alpha - y, \nabla f(y) \rangle}_{\text{linearization of } f} + \underbrace{\frac{L}{2}\|\alpha - y\|_2^2}_{\text{regularization}} + g(\alpha), \tag{3.13}$$

where the angle brackets $\langle x, y \rangle = y^T x$ represent the inner product. The minimizer for this approximation is then given as the fixed-point equation

$$\begin{aligned} \pi_L(\alpha^*, \lambda) &= \mathbf{prox}_g(\alpha^* - L^{-1}\nabla f(\alpha^*), \lambda L^{-1}) \\ &= \mathbf{prox}_g(\alpha^* - L^{-1}\Phi^T(\Phi\alpha^* - y), \lambda L^{-1}), \end{aligned} \tag{3.14}$$

where α^* denotes the optimal solution and L is the Lipschitz constant of ∇f given by Equation 3.10. In this equation L is used to determine the optimal stepsize.

Algorithm 1 Iterative Shrinkage Tresholding Algorithm

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_L(α, λ)
4:     return α
```

These preliminary definitions are enough to define the simple iterative algorithm called ISTA as seen in algorithm 1. For our trivial function g this iterative scheme is identical to the standard gradient descent algorithm. This optimization method always converges to the global maximum, but only linearly. The full proof of this behavior is given in [**fista**].

So far we have only seen the proximal operator of a very simple function. There are also such operators for more complicated regularization methods. We need the following closed form solutions:

$$\text{(Lasso)} \qquad \left(\mathbf{prox}_{\|\cdot\|_1}(\alpha, \lambda)\right)_i = (\alpha_i - \lambda)_+ - (-\alpha_i - \lambda)_+, \tag{3.15}$$

$$\text{(Ridge)} \qquad \mathbf{prox}_{\|\cdot\|_2}(\alpha, \lambda) = \left[(1 - \lambda)\left(\|\alpha\|_2\right)^{-1}\right]_+, \tag{3.16}$$

where $(x)_+ = \max(x, 0)$ denotes the positive part of x. The derivations are omitted for the sake of brevity, the interested reader is referred to the survey paper [**proxsurvey**]. We observe, that by setting the regularization parameter λ equal to zero, we recover, again, the gradient minimization method.

These proximal operators can be used to define a minimizer for a non-smooth function g. For example, combining Equation 3.14 with the proximal operator for the

Lasso functional $g(\alpha) = \lambda\|\alpha\|_1$ given by Equation 3.15 results in the minimizer

$$\pi_{L\|\cdot\|_1}(\alpha^*, \lambda) = \left[\left(\alpha^* - L^{-1}\nabla f(\alpha^*)\right) - \lambda L^{-1}\right]_+ - \left[-\left(\alpha^* - L^{-1}\nabla f(\alpha^*)\right) - \lambda L^{-1}\right]_+,$$

again given as a fixed-point iteration. In this equation the function $(x)_+$ is applied element-wise on its input vector.

This optimization method always converges to the global maximum, but only linearly. The full proof of this behavior is given in [**fista**]. To overcome this problem, Beck and Teboulle combined the ISTA algorithm with the accelerated gradient descent algorithm discovered by Nesterov. This augmented algorithm archives quadratic convergence and is called FISTA. 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.

Another problem with algorithm 1 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 Equation 3.10. To avoid this expensive calculation, we use a backtracking line search to determine a suitable stepsize. In this line search procedure we use Equation 3.13 as an upper bound for Equation 3.8. It is then straightforward to derive algorithm 2.

Algorithm 2 Linesearch

Input: $L > 0, \eta > 1, \alpha$

```
1: function LINESEARCH(α, L)
2:     i ← 0
3:     do
4:         L ← η^i L
5:         prox ← π(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. This line search always finds a suitable approximation for L given correct starting values. It is possible that this procedure finds a non-optimal stepsize, i.e. a stepsize that is smaller than the optimal one. This is not a problem in practice, our optimization procedure still converges, but slower.

Using these definitions we can finally present the optimal optimization algorithm, shown by algorithm 3.

It is of course possible to use a constant stepsize like in algorithm 1. To do this, start the line search with the minimal value of L. The linesearch then terminates after the

The same line search can be used to determine the stepsize for ISTA.

first iteration and is therefore as fast as calculating Equation 3.14. A comparison of the practical speed of ISTA and FISTA with constant stepsize can be seen in Figure 3.1.

Algorithm 3 Fast Iterative Shrinkage Tresholding Algorithm

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

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

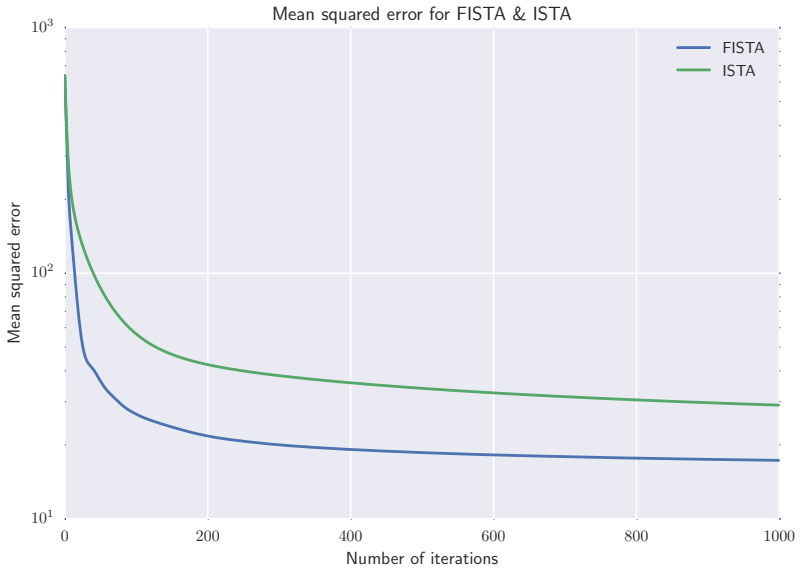

Figure 3.1: Comparison of FISTA and ISTA, both with constant stepsize. The figure shows the training mean squared error for the first 1000 iterations with $\lambda = 0.001$ for the training set of the concrete data set.

Todo list

List of Figures

List of Tables