Universidad Autónoma de Madrid
Escuela Politécnica Superior
Departamento de Ingeniería Informática

Sparse Linear Models and Proximal Optimization

Master's thesis presented to apply for the
Master in Computer Science and Telecommunication Engineering

By
Alberto Torres Barrán

under the direction of
José R. Dorronsoro Ibero

Madrid, September 24, 2013

Contents

Abstract

Sparse models are growing in popularity as the amount of data available is becoming bigger. Feature selection methods have been around for many years, but the problems to solve evolve quite rapidly. If ten years ago a dataset with hundreds of variables was considered quite large, today datasets with ten thousands of variables are a common problem and easy to find. Hence, sparse models are not evaluated only by the error of the fit but also by its interpretability. This means that sometimes we might prefer a model with "worse" error if we can learn some structure of the problem from it.

In this thesis we will provide an overview of the classic sparse models, starting with the Lasso and, from that point, we will dicuss the more modern proximal optimization theory. The main result is the FISTA algorithm, which is a general framework to optimize any combination of differentiable function and convex regularization. An implementation of FISTA in ANSI C was also developed along with this thesis, and it is publicly available on the Internet. Finally we will present some experiments where all the previous models are compared using real-world datasets.

Acknowledgements

The author is kindly supported by the FPU12/05163 Ministerio de Economía y Competitividad grant. He has been also partially supported by Cátedra IIC Modelado y Predicción and the FPI-UAM grant.

Chapter 1

Introduction

Machine learning is a branch of artificial intelligence commonly defined as the science of learning from data without the need for the computer to be explicitly programmed. Other fields that overlap with machine learning are statistics, data mining and pattern recognition, since they often use the same methods. Some example of learning problems are:

- Learn to distinguish between spam and non-spam email messages and classify new messages accordingly.
- Predict the selling price of a house based on facts such as square meters and number of bedrooms.
- Predict the price of a stock in 6 months from now, on the basis of company performance measures and economic data [1].
- Recognize handwritten digits from a digitized image, for example ZIP codes in letters.
- Identify the risk factors for prostate cancer, based on clinical and demographic variables [1].
- Identify the clients of an insurance company who are likely to upgrade their policy to premium if an offer is made to them.

In the typical scenario we have an outcome measurement we want to predict, which can be either quantitative (price of the house) or categorical (spam/non-spam), based on a set of features or variables. We have a set of data, in which we observe both the features and the outcome, and the goal is to build a prediction model which allows us to predict the outcome of new samples, not used to train the model.

Notice that it is impossible to compute the accuracy of the predictions for the new data, since the real outcome is not available. So, in order to asses the quality of the model, the original dataset is usually divided into a training set and a test set. The former is used to build the prediction model while the latter is used to estimate its performance. This is done by computing the accuracy of the predictions in the test set and, since they were not used for training, they are a good estimate of the accuracy for new data (assuming they are *similar*).

The ability to predict correctly the outcome of new unseen data is known as generalization. In practice the variability of the data will be such that the training set can comprise only a small amount of all possible examples, so generalization is a central goal in machine learning.

Machine learning systems also have to deal with the representation of the data. For instance in the case of handwritten digits recognition, it is usually convenient to represent the digitized images as numerical vectors using a greyscale approach. Data is also usually pre-processed to transform it into another space where the problem of building the model is easier to solve. Continuing with the example of digit recognition, the images are typically traslated and scaled so that each digit is contained in a fixed size box [2]. Note that new data must be processed using the same steps as the training data.

So far we have only mentioned the task where the training data contains both the features and the outcome. That kind of data is said to be *labeled*, and such applications are known as *supervised learning* problems. These problems can be further divided into *classification*, if the outcome is a finite number of discrete categories, or *regression*, if the outcome consists of one or more continuous variables. Examples of classification problems are spam detection, handwritten digits recognition and identifying prossible premium users. Examples of regression problems are house princing, prostate cancer detection and stock price prediction.

There are other machine learning problems where the input data consists only of features, and no outcome is available. In those *unsupervised learning* problems the goal is to find some kind of structure in the data: groups of similar examples (*clustering*), distribution of the data in the input space (*density estimation*) or subspaces in which the data is still "meaningful" but with less dimensions (*dimensionality reduction*).

1.1 Optimization theory

In this section we are going to conver some basic optimization theory, which will be useful in the following chapters, since models are usually built by minimizing a function. Most of the definitions and theorems are taken from [3], so please refer to it for more details on this subject.

The general form of the problem we are going to consider is that of finding the maximum or minimum of a function subject to some constraints.

Definition 1.1.1. *(Primal optimization problem) Given function f, g_i, $i = 1, \cdots, k$, y h_i, $i = 1, \cdots, m$, defined over a domain $\mathbf{w} \subseteq \mathcal{R}^n$.*

$$\begin{array}{lll} \min & f(\mathbf{w}), & \mathbf{w} \in \mathbf{w}, \\ s.t & g_i(\mathbf{w}) \leq 0, & i = 1, \cdots, k, \\ & h_i(\mathbf{w}) = 0, & i = 1, \cdots, m, \end{array}$$

where $f(\mathbf{w})$ is known as objective function, $g_i(\mathbf{w})$ are the inequality constraints and $h_i(w)$ are the equality contraints. From now on we are going to write $g(\mathbf{w}) \leq 0$ to indicate $g_i(\mathbf{w}) \leq 0$, $i = 1, \cdots, k$, and the same with $h(\mathbf{w})$.

Definition (1.1.1) is general, since maximization problems can be transformed into minimization ones simply by flipping the sign of the objective function $f(\mathbf{w})$. In the same way, every constraint can be re-written in one of the previous forms. For instance, if we have the constrain $\sum \mathbf{w} < t$, we can always write it like $\sum \mathbf{w} - t < 0$.

The domain region where the objective function is defined and all the constraints are satisfied is known as *feasible region*

$$R = \{\mathbf{w} \in \mathbf{w} : g(\mathbf{w}) \leq 0,\ h(\mathbf{w}) = 0\} \tag{1.1.1}$$

The solution of the problem is a point $\mathbf{w}^* \in \mathbb{R}$ such that there is no other point $\mathbf{w} \in \mathbb{R}$ for which $f(\mathbf{w}) < f(\mathbf{w}^*)$.

Definition 1.1.2. *An optimization problem where the objective function and all the constraints are linear is a linear problem. If the objective function is quadratic and the constraints are linear is a quadratic problem.*

For the purposes of the following chapters we can restrict ourselves to the case where the constraints are linear, the objective function is convex and quadratic. A nice property of unconstrained convex problems is that any local minimum $\mathbf{w}^*$ is also a global minimum. An exception is multilayer perceptrons (section 1.3.1), whose objective function is not convex an thus many local minima will exist. Convex problems will be discussed further in chapter 3, where we present the definitions of convex set and convex function. We are going to present next the basics of the Lagrangian theory, that will be used in subsequent chapters.

1.1.1 Lagrange theory

The Fermat theorem that characterizes the solutions of unconstrained convex problems will be discussed in detail in chapter 3. Let us consider here the more general case where the optimization problem has both equality and inequality constraints. We are going to define first the generalized Lagrangian.

Definition 1.1.3. *(Lagrangian function) Given the optimization problem* (1.1.1)*, we defined the generalized Lagrangian function as*

$$\begin{aligned} L(\mathbf{w}, \boldsymbol{\alpha}, \boldsymbol{\beta}) &= f(\mathbf{w}) + \sum_{i=1}^{k} \alpha_i g_i(\mathbf{w}) + \sum_{i=1}^{m} \beta_i h_i(\mathbf{w}) \\ &= f(\mathbf{w}) + \boldsymbol{\alpha}^t \mathbf{w} + \boldsymbol{\beta}^t \mathbf{w}. \end{aligned}$$

We can now define the Lagrangian dual problem

Definition 1.1.4. *(Dual optimization problem)*
The Lagrangian dual problem of the primal problem of definition 1.1.1 is the following problem:

$$\begin{array}{ll} \max & \Theta(\boldsymbol{\alpha}, \boldsymbol{\beta}) \\ s.t & \boldsymbol{\alpha} \geq 0 \end{array}$$

where $\Theta(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \inf_w L(\mathbf{w}, \boldsymbol{\alpha}, \boldsymbol{\beta})$

One of the fundamental relationships between the primal and the dual problem is given by the weak duality theorem, presented next.

Theorem 1.1.1. *(Weak duality theorem) Let* $\mathbf{w} \in \Omega$ *be a feasible solution of the primal problem 1.1.1 and* $(\boldsymbol{\alpha}, \boldsymbol{\beta})$ *a feasible solution of the dual problem 1.1.4. Then* $f(\mathbf{w}) \geq \Theta(\boldsymbol{\alpha}, \boldsymbol{\beta})$.

Proof. From the definition of $\Theta(\boldsymbol{\alpha}, \boldsymbol{\beta})$ for $\mathbf{w} \in \Omega$ we have

$$\inf_u L(\mathbf{u}, \boldsymbol{\alpha}, \boldsymbol{\beta}) \leq L(\mathbf{w}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = f(\mathbf{w}) + \boldsymbol{\alpha}^t g(\mathbf{w}) + \boldsymbol{\beta}^t h(\mathbf{w}) \leq f(\mathbf{w}). \tag{1.1.2}$$

The last inequality holds since the feasibility of $\mathbf{w}$ implies that

$$g(\mathbf{w}) \leq 0 \text{ and } h(\mathbf{w}) = 0 \tag{1.1.3}$$

while the feasibility of $\boldsymbol{\alpha}$ implies

$$\boldsymbol{\alpha} \geq 0. \tag{1.1.4}$$

Therefore from (1.1.3)-(1.1.4)

$$\boldsymbol{\beta}^t h(\mathbf{w}) = 0 \text{ and } \boldsymbol{\alpha}^t g(\mathbf{w}) \leq 0.$$

□

The difference between the values of the primal and dual problems is known as *duality gap*. Now we are going to see the strong duality theorem, which guarantees that the duality gap is 0 in the optimization problems we are discussing later. This means that the dual and the primal problems have the same value. The proof of the strong duality theorem can be found in [4].

Theorem 1.1.2. *(Strong duality theorem) Given an optimization problem like the one in definition 1.1.1, where the g_i and h_i are affine functions, that is*

$$h(\mathbf{w}) = \mathbf{A}\mathbf{w} - \mathbf{b} \tag{1.1.5}$$

for some matrix $\mathbf{A}$ *and vector* $\mathbf{b}$*, the duality gap is* 0*.*

The last theorem of this section characterizes the optimum solution of a general optimization problem [3].

Theorem 1.1.3. *(Khun-Tucker) Given an optimization problem in the convex domain* $\Omega \subseteq \mathbb{R}^n$

$$\begin{array}{lll} \min & f(\mathbf{w}), & \mathbf{w} \in \Omega, \\ s.t & g_i(\mathbf{w}) \leq 0, & i = 1, \cdots, k, \\ & h_i(\mathbf{w}) = 0, & i = 1, \cdots, m, \end{array}$$

with convex f*, affine* g_i*,* h_i*, the necessary and sufficient conditions for a normal point* $\mathbf{w}^*$ *to be an optimum are the existence of* $\boldsymbol{\alpha}^*$ *and* $\boldsymbol{\beta}^*$ *such that*

$$\begin{aligned} \frac{\partial L(\mathbf{w}^*, \alpha^*, \beta^*)}{\partial \mathbf{w}} &= 0 \\ \frac{\partial L(\mathbf{w}^*, \alpha^*, \beta^*)}{\partial \beta} &= 0 \\ \alpha_i^* g_i(\mathbf{w}^*) &= 0, \quad i = 1, \cdots, k \\ g_i(\mathbf{w}^*) &\leq 0, \quad i = 1, \cdots, k \\ \alpha_i^* &\geq 0, \quad i = 1, \cdots, k \end{aligned}$$

1.2 Linear models

1.2.1 Regression

A linear regression model has the form

$$\mathbf{y} = f(\mathbf{X}) = \mathbf{X}\mathbf{w} + w_0, \tag{1.2.1}$$

where $\mathbf{X}$ is the $n \times p$ data matrix and $\mathbf{y}$ is the $n \times 1$ response vector (n number of samples or observations, p number of variables). The linear model either assumes that the regression function $\mathbb{E}(y|\mathbf{X})$ is linear or that the linear model is a reasonable aproximation, which is usually the case. Therefore we assume that data comes from a model

$$\mathbf{y} = f(\mathbf{X}) = \mathbf{X}\mathbf{w} + w_0 + \boldsymbol{\epsilon} \tag{1.2.2}$$

where $\boldsymbol{\epsilon} \sim N(0, \sigma)$ is the *noise*, and it stresses the fact that the linear model is only an aproximation of the underlying true model, since it is very difficult in practice to have real data that comes from a perfectly linear model.

The term w_0 is known as the bias or intercept and it is usually included into the vector $\mathbf{w}$. That way, if we also add a column of 1s to the matrix $\mathbf{X}$, we can write the model in the more convenient form

$$\mathbf{y} = \mathbf{X}\mathbf{w}.$$

The components of the vector $\mathbf{w}$, w_j, are known as parameters or coefficients and the columns of the matrix $\mathbf{x}_j$ are the variables or features. These variables can come from different sources [1]:

- quantitative inputs;
- transformations of quantitative inputs, $\mathbf{x}_3 = \mathbf{x}_1^t\mathbf{x}_2$;
- basis expansions, such as $\mathbf{x}_2 = \mathbf{x}_1^2$;
- "dummy" variables coding of the levels of qualitative inputs. For instance if we have a feature with 5 possible values we might create five different variables that are all set to 0 but one.

Let $\mathbf{x}_i$ be now the ith pattern, that is, the ith row of the matrix $\mathbf{X}$. Tipically we have a *training set* $\{\mathbf{x}_i, \mathbf{y}_i\}, i = 1, \dots, n$ from which to estimate the parameters $\mathbf{w}$. The most popular estimation method is ordinary least squares (OLS), in which coefficients are obtained by minimizing the residual sum of squares, defined as

$$\text{RSS}(\mathbf{w}) = \sum_{i=1}^{n} (y_i - \mathbf{x}_i^t\mathbf{w})^2 = (\mathbf{y} - \mathbf{X}\mathbf{w})^t(\mathbf{y} - \mathbf{X}\mathbf{w}) = \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 \tag{1.2.3}$$

that is

$$\widehat{\mathbf{w}} = \text{argmin}\left\{\|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2\right\}. \tag{1.2.4}$$

It is easy to show that the optimization problem (3.3.4) has a closed-form solution. Differenciating in (1.2.3) with respect to $\mathbf{w}$ we obtain

$$\frac{\partial \text{RSS}}{\partial \mathbf{w}} = -2\mathbf{X}^t(\mathbf{y} - \mathbf{X}\mathbf{w}) \tag{1.2.5}$$
$$\frac{\partial^2 \text{RSS}}{\partial \mathbf{w}} = 2\mathbf{X}^t\mathbf{X}.$$

Assuming that $\mathbf{X}$ has full column rank and hence $\mathbf{X}^t\mathbf{X}$ is positive definite, we set the first derivative to zero

$$\mathbf{X}^t(y - \mathbf{X}\mathbf{w}) = 0, \tag{1.2.6}$$

to obtain the unique solution

$$\widehat{\mathbf{w}} = (\mathbf{X}^t\mathbf{X})^{-1}\mathbf{X}^t\mathbf{y}. \tag{1.2.7}$$

The fitted values at the training inputs are

$$\widehat{\mathbf{y}} = \mathbf{X}\widehat{\mathbf{w}} = \mathbf{X}(\mathbf{X}^t\mathbf{X})^{-}1\mathbf{X}^t\mathbf{y}. \tag{1.2.8}$$

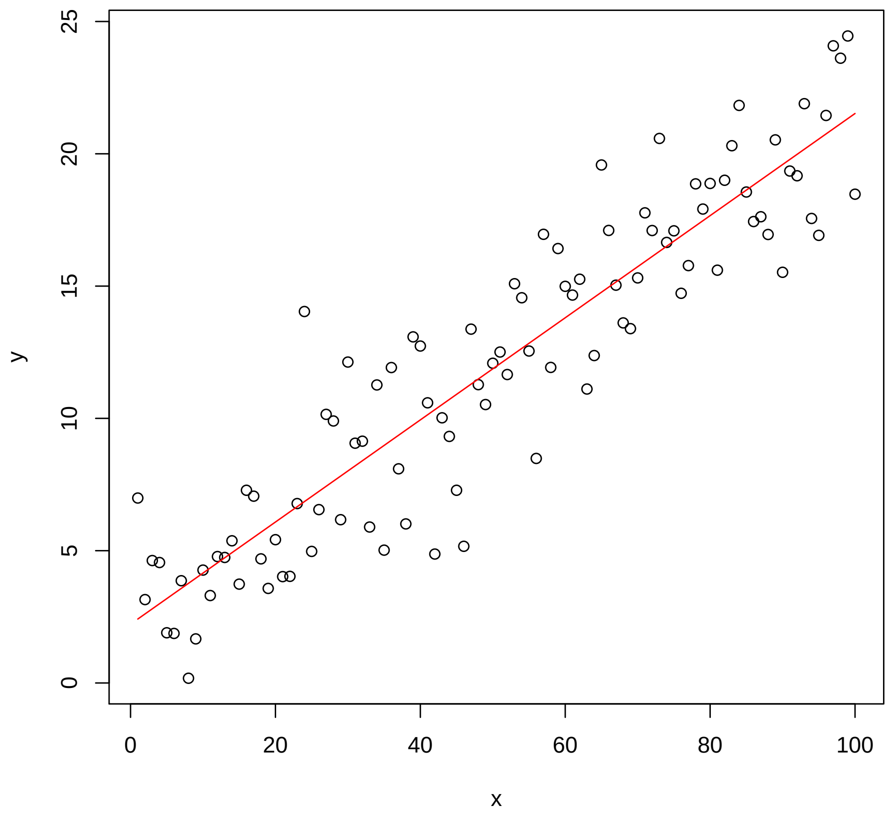


Figure 1.2.1: Linear least squares fitting with $\mathbf{X} \in \mathbb{R}$

The matrix $\mathbf{H} = \mathbf{X}(\mathbf{X}^t\mathbf{X})^{-1}\mathbf{X}^t$ is sometimes called the "hat" matrix. We can also make predictions for new data $\tilde{\mathbf{X}}$ that was not used to fit the model. The predicted values are given by $f(\tilde{\mathbf{X}}) = \tilde{\mathbf{X}}\widehat{\mathbf{w}}$.

From a geometrical point of view, the least squares solution is the $p+1$ dimensional hyperplane that minimizes the sum of squared residuals. The coefficients are chosen so that the residual vector $\mathbf{y} - \hat{\mathbf{y}}$ is orthogonal to the subspace spaned by the columns of the input matrix $\mathbf{X}$. This orthogonality is expresed in (1.2.5), and the resulting estimate $\hat{\mathbf{y}}$ is the orthogonal projection of $\mathbf{y}$ into this subspace. Hence the matrix $\mathbf{H}$ is also known as the projection matrix. Figure 1.2.1 shows an example of the regression hyperplane in a two-dimensional space for randomly generated data with $w_1 = 0.2$, $w_0 = 2$ and $\epsilon \sim N(0, 2.5)$.

It may happen that the columns of $\mathbf{X}$ are not linearly independent so that $\mathbf{X}$ is not full rank. This is the case, for example, if two of the variables are perfectly correlated. Then the matrix $\mathbf{X}^t\mathbf{X}$ is singular, the inverse in equation (2.2.3) cannot be computed and the coefficients $\mathbf{w}$ are not uniquely defined. The natural way to solve this non-unique representation is to remove from the matrix $\mathbf{X}$ all the redundant variables.

Rank deficiencies also accur when the number of variables p is bigger than the number of observations n. This happens frequently in fields such as image processing and bioinformatics. Redundant variables are non-existent in this case, so the less important ones must be filtered for the problem to be solvable. Another option is to control the fit by regularization, as explained in section 1.4.

1.2.2 Classification

As explained before, in the classification problem the predictor $G(x)$ takes values in a discrete set $\mathcal{G}$. Therefore we can always divide our input space into regions labeled according to the classification. In the linear methods for classification, these *decission boundaries* will be linear.

One of the most common ones is logistic regression, where we model the posterior probabilities of the K classes via linear functions in x, while at the same time we make sure that they sum one. Let $\mathbf{x}_i$ be the ith observation (row) of the input matrix $\mathbf{X}$ and y_i the associated label. For the sake of simplicity, we are going to assume that $K = 2$ and we are going to label the two classes as $0/1$. Then, the probability of belonging to the positive class ($y_i = 1$) is defined as

$$p(y_i = 1|\mathbf{x}_i) = \pi(\mathbf{x}_i) = \frac{1}{1 + \exp(-\mathbf{w}^t\mathbf{x}_i)}, \tag{1.2.9}$$

and the probability of belonging to the negative class ($y_i = 0$) is

$$p(y_i = 0|\mathbf{x}_i) = 1 - \pi(\mathbf{x}_i) = \pi(-\mathbf{x}_i) = \frac{1}{1 + \exp(\mathbf{w}^t\mathbf{x}_i)}. \tag{1.2.10}$$

Logistic regression models are usually fit by maximum likelihood, using the conditional likelihood of G given $\mathbf{X}$. The log-likelihood for n observations is

$$\begin{aligned}\mathcal{L}(\mathbf{w}) &= \sum_{i=1}^{n} y_i \log \pi(\mathbf{x}_i) + (1 - y_i) \log \pi(-\mathbf{x}_i) \\ &= \sum_{i=1}^{n} y_i \mathbf{w}^t \mathbf{x}_i + \log \pi(-\mathbf{x}_i).\end{aligned}$$

Next we have to maximize the log-likelihood (or minimize the negative log-likelihood, which is usually more convinient). The derivatives of the log-likelihood with respect to $\mathbf{w}$ are

$$\frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}} = \sum_{i=1}^{n} \mathbf{x}_i(y_i - \pi(\mathbf{x}_i)), \tag{1.2.11}$$

that is, $p+1$ equations non-linear in $\mathbf{w}$. This optimization can be performed with different algorithms such as *iteratively reweighted least squares* or IRLS. It is worth mentioning that maximum likelihood can exhibit severe over-fitting for data sets that are linearly separable [2]. Once the weights are estimated, new observations can be classified using the rule

$$\hat{y} = G(\mathbf{x}) = \begin{cases} 0 & \text{if } \pi(\mathbf{x}) \geq 0.5 \\ 1 & \text{if } \pi(\mathbf{x}) < 0.5 \end{cases}$$

Last it is also important to mention that logistic regression can be extended to the multinomial setting ($K > 2$). See [1] and [2] for more details.

1.3 Non-linear models

1.3.1 Multilayer perceptron

A multilayer perceptron (MLP) is a feedforward artificial neural network with multiple layers of nodes in a direct graph, with each layer fully connected to the next one. The first layer is the input layer and the last layer is known as the output layer. All the middle layers are called hidden layers. In figure 1.3.1 there is an example of a multilayer perceptron with one hidden layer, which is the most common one. Sometimes the more general term "neural network" is used to described this type of multilayer perceptron, and in what follows both will be used interchangeably.

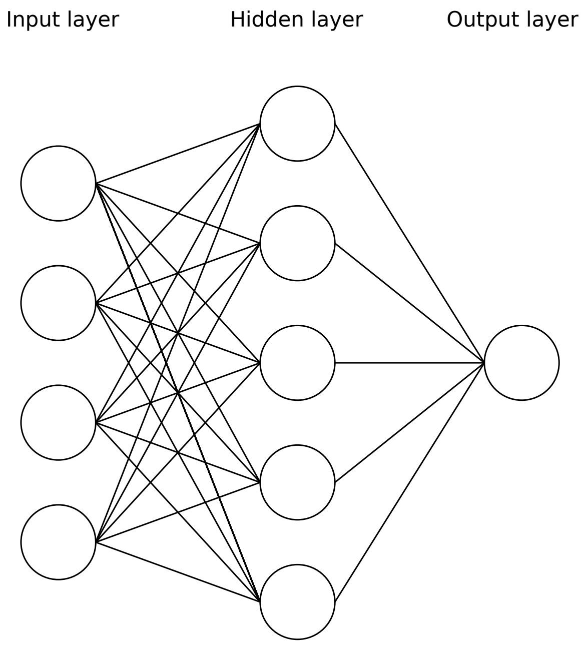


Figure 1.3.1: An example of a multilayer perceptron with one hidden layer

The neural network model can be described as series of functional transformations. First we construct H linear combination of the input variables $x_1, \dots, x_P$

$$a_j = \sum_{i=1}^{P} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \tag{1.3.1}$$

where H is the number of units in the hidden layer and the superscript indicates that the corresponding parameters or "weights" connect the input layer to the hidden layer. The parameters w_{j0} are the biases, and they can be included with the other weights if we add a new input x_0 equal to one. The quantities a_j are the *activations* of the hidden units. These values are again linearly combined after being transformed using a differentiable, nonlinear activation function Φ

$$a_k = \sum_{j=1}^{H} w_{kj}^{(2)} \Phi(a_j) + w_{k0}^{(2)} \tag{1.3.2}$$

where $k = 1, \dots, K$, and K is the number of outputs. Here the superscript indicates that these weights are the ones from the hidden layer to the ouput layer. The nonlinear functions Φ are usually chosen to be sigmoidal functions such as logistic sigmoid or the 'tanh' function. Finally the output unit activations are transformed again using another activation function to give the network outputs y_k,

$$y_k = \sigma(a_k). \tag{1.3.3}$$

This output activation function depend on the type of problem we are trying to solve. Therefore, for regression problems the output activation function is the identity, $\sigma(a) = a$

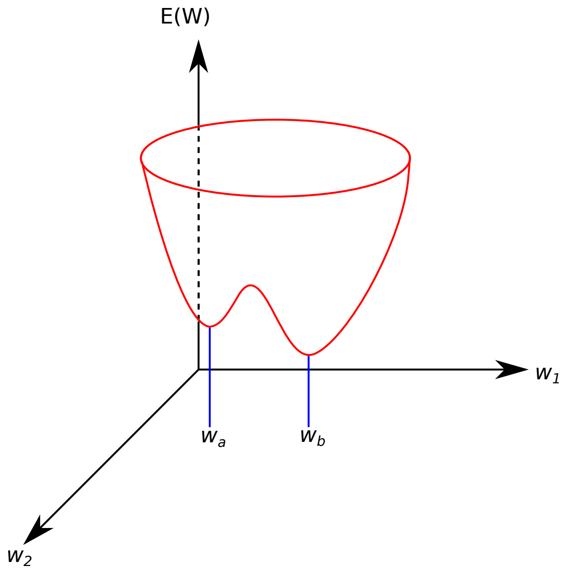


Figure 1.3.2: Geometrical view of the error function $E(\mathbf{w})$. Point w_A is a local minimum and point w_B is the global minimum [2]

while for binary classification problems the output is transformed using a logistic sigmoid function

$$\sigma(a) = \frac{1}{1+\exp(-a)}. \tag{1.3.4}$$

The previous steps can be combined to give the overall network function, that, for one output k takes the form

$$y_k(\mathbf{x},\mathbf{w}) = \sigma\left(\sum_{j=1}^{H} w_{kj}^{(2)}\Phi\left(\sum_{i=1}^{P} w_{ji}^{(1)}x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right). \tag{1.3.5}$$

1.3.1.1 Backpropagation

Given a training set of input vectors $\{\mathbf{x}_n\}$, where $n = 1,\dots,N$, together with a corresponding set of target vectors $\{\mathbf{t}_n\}$, we minimize the error function

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N} \|\mathbf{y}(\mathbf{x}_n,\mathbf{w}) - \mathbf{t}_n\|^2 \tag{1.3.6}$$

Because the the error $E(\mathbf{w})$ is a smooth continuous function of $\mathbf{w}$, its smallest value will occur at a point in weight space where the gradient is zero,

$$\nabla E(\mathbf{w}) = 0.$$

However, there will be many points where the gradient vanishes but not all of them are global minima, that is, the smallest value of the function for any weight. Any other minima corresponding to higher values of the error function are local minima. Figure 1.3.2 shows an example of error function as a surface in the weight space, with one local minimum and one global minimum.

This function can be minimized using iterative procedures such as, *gradient descent*, also known as *steepest descent*. This technique starts with an initial value or “guess” for the weights, $\mathbf{w}^{(0)}$ and update the weight vector taking a small step in the direction of the negative gradient, so that,

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \nabla E(\mathbf{w}^{(t)}), \tag{1.3.7}$$

where the parameter $\eta > 0$ is the learning rate. It is worth mentioning that the error function is defined with respect to a training set, so each step requires that the entire set is processed. This is called *batch learning*, as oppose to *online learning*, where only a few observations are present at each iteration (or even only one observation). There are faster algorithms for batch training such as conjugate gradient and quasi-Newtown methods, although not as simple as gradient descent.

As mentioned earlier, the error function has many local minima, so it may be necessary to run the procedure multiple times starting from different initial points, randomly selected, in order to find a good minimum. However, most of the times in practice we will not know if it is a global minimum or not.

Finally, we need an efficient procedure to compute the gradient $\nabla E(\mathbf{w})$, since this operation is performed at each iteration. This can be achieved using an algorithm called *backpropagation*. We present next a slightly modified derivation of the algorithm from [2], for a multilayer perceptron with one hidden layer and arbitrary differentiable nonlinear activation functions. First we have to compute the activations of all the hidden and output units for all the training patterns, using equations (1.3.1)-(1.3.3). Next, consider the error for a particular training pattern n,

$$E = \frac{1}{2} \sum_{k=1}^{K} (y_k - t_k)^2. \tag{1.3.8}$$

In order to compute the partial of the error function with respect to a weight w_{ji} we have to apply the chain rule of partial derivatives

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{a_j} \frac{\partial a_j}{\partial w_{ji}} \tag{1.3.9}$$

We now introduce the notation

$$\delta_j \equiv \frac{\partial E}{\partial a_j}. \tag{1.3.10}$$

Then, using equation (1.3.1) we get

$$\frac{\partial a_j}{\partial w_{ji}} = \Phi(a_i). \tag{1.3.11}$$

Substituting (1.3.11) and (1.3.10) in (1.3.9) we obtain

$$\frac{\partial E}{\partial w_{ji}} = \delta_j \Phi(a_i). \tag{1.3.12}$$

Thus, in order to evaluate the derivatives, we only need to calculate the value of δ_j for each output unit and hidden unit and then apply (1.3.12). For the output units with linear activation functions, $y_k = a_k$ and then

$$\delta_k = y_k - t_k. \tag{1.3.13}$$

Let j be a hidden unit, then using again the chain rule for partial derivatives we get

$$\delta_j \equiv \frac{\partial E}{\partial a_j} = \sum_k \frac{\partial E}{a_k} \frac{\partial a_k}{\partial a_j}, \tag{1.3.14}$$

summing over all the output units k. Making use of (1.3.2) we can compute the derivative

$$\frac{\partial a_k}{\partial a_j} = w_{kj}\Phi'(a_j). \tag{1.3.15}$$

Substituting (1.3.10) and (1.3.15) into (1.3.14) we get the following backpropagation rule

$$\delta_j = \Phi'(a_j)\sum_k w_{kj}\delta_k \tag{1.3.16}$$

The backpropagation procedure can be summarized as follows [2]:

1. Forward propagate an input vector $\mathbf{x}_i$ through the network using equation (1.3.5), storing all the intermediate activations.
2. Evaluate the δ_k for all the output units using (1.3.13).
3. Backpropagate the δ's using (1.3.16) to obtain δ_j for each hidden unit in the network.
4. Use (1.3.12) to evaluate the required derivatives.

Finally, the gradient descent step for the multilayer perceptron can be written as

$$\mathbf{w}_{ji}^{(t+1)} = \mathbf{w}_{ji}^{(t)} - \eta\delta_j\Phi(a_i). \tag{1.3.17}$$

It is worth mentioning that for batch methods we can obtain the total error simply by summing over all patterns.

1.3.2 Support vector machines

Support Vector Machines (SVM) are a classification method capable of dealing with high dimensional data. SVMs belong to the class of algorithms known as *kernel methods*, since they depend on data only through dot-products. Therefore, they can be replaced by kernel functions, that compute these products in another feature space, different from the input space, and possibly with higher dimension. This presents two main advantages:

1. The hability to generate non-linear decision functions using methods designed for linear classifiers.
2. The clasifier can be applied to data that do not have a representation in a fixed dimension space.

1.3.2.1 The linearly separable case

Given a training set $\{\mathbf{x}_i, y_i\}$, $i = 1, \cdots, l$, $y_i \in \{-1, 1\}$, $\mathbf{x}_i \in \mathbf{R}^d$, let's assume that there is an hyperplane that separates positive samples from the negative ones. The points $\mathbf{x}$ that are on the hyperplane satisfy $\mathbf{w}^t\mathbf{x} + b = 0$, where $\mathbf{w}$ is normal to the hyperplane, $|b|/\|\mathbf{w}\|$ is the distance perpendicular from the hyperplane to the origin and $\|\mathbf{w}\|$ is the Euclidean norm of $\mathbf{w}$. Let d_+ (d_-) the distance from the hyperplane to the closest positive (negative) point. Then, we define the margin of the hyperplane as $d_+ + d_-$. For the linearly separable case, the SVM simply tries to find the maximum-margin separating hyperplane. This can be formulated more formally as follows: assume that the points satisfy the restrictions

$$\mathbf{x}_i^t\mathbf{w} + b \geq +1 \quad \text{for} \quad y_i = +1 \tag{1.3.18}$$
$$\mathbf{x}_i^t\mathbf{w} + b \leq -1 \quad \text{for} \quad y_i = -1 \tag{1.3.19}$$

These can be combined in

$$y_i(\mathbf{x}_i^t\mathbf{w} + b) - 1 \geq 0 \quad \forall i. \tag{1.3.20}$$

The points that satisfy the equality constraint in equation (1.3.18) are on the hyperplane $\mathbf{x}_i^t\mathbf{w} + b = 1$ and the points that satisfy the equality constraint in equation (1.3.19) are on the hyperplane $\mathbf{x}_i^t\mathbf{w} + b = -1$.

If the data is linearly separable, these hyperplanes separate the data and there are no points between them. The distance from this hyperplanes to $\mathbf{x}_i^t\mathbf{w} + b = 0$ is $d_+ = d_- = 1/\|\mathbf{w}\|$ and the margin defined above is simply $2/\|\mathbf{w}\|$. The maximum-margin hyperplane can be found minimizing $\|\mathbf{w}\|^2$ subject to the constraints (1.3.20).

$$\begin{array}{cc} \min\limits_{\mathbf{w},b} & \frac{1}{2}||\mathbf{w}||^2 \\ \text{s.t} & y_i(\mathbf{x}^t\mathbf{w} + b) - 1 \geq 0 \quad \forall i \end{array} \tag{1.3.21}$$

Next lets transform the previous problem into the dual equivalent. The objective function (1.3.21) is clearly convex and the restrictions are affine, therefore the Lagrangian of the problem is

$$L_P = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{l} \alpha_i y_i(\mathbf{x}_i^t\mathbf{w} + b) + \sum_{i=1}^{l} \alpha_i \tag{1.3.22}$$

Now we have to minimize L_P with respect to $\mathbf{w}$ and b, requiring also that the derivatives of L_P with respect to α_i are 0 and everything under the constraints $\alpha_i \geq 0$. The derivatives of L_P with respect to $\mathbf{w}$ and b are

$$\partial_w L_P = w - \sum_{i=1}^{l} \alpha_i y_i x_i \tag{1.3.23}$$

$$\partial_b L_P = -\sum_{i=1}^{l} \alpha_i y_i. \tag{1.3.24}$$

Thus, the optimum $(\mathbf{w}^*, b^*)$ is obtained if the following constraints are met (KKT *conditions*):

$$w - \sum_i \alpha_i y_i x_i = 0 \tag{1.3.25}$$

$$-\sum_i \alpha_i y_i = 0 \tag{1.3.26}$$

$$y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 \geq 0 \quad i = 1, \cdots, l \tag{1.3.27}$$

$$\alpha_i \geq 0 \quad \forall i \tag{1.3.28}$$

$$\alpha_i(y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1) = 0 \quad \forall i \tag{1.3.29}$$

Substituting (1.3.23) and (1.3.24) in (1.3.22) we get

$$L_D = \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i\alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j. \tag{1.3.30}$$

Therefore the new dual problem is defined as

$$\begin{array}{cc} \max\limits_{\alpha} & \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i\alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \\ \text{s.t} & \alpha_i \geq 0 \quad \forall i \\ & \sum_i \alpha_i y_i = 0 \end{array} \tag{1.3.31}$$

The dual problem is much easier to solve since the constraints are simpler. However, most real world datasets are not linearly separable. In the following section we are going to introduce the most common case in practice, that is, when data is not linearly separable.

1.3.2.2 Non-linearly separable case

In order to allow classification errors when data is not linearly separable, we replace the constraints (1.3.20) by

$$y_i(\mathbf{x}_i^t\mathbf{w} + b) - 1 + \xi_i \geq 0 \quad \forall i \tag{1.3.32}$$

where $\xi_i \geq 0$ are slack variables that make possible for a point to be inside the margin $0 \leq \xi_i \leq 1$ or to be misclassified ($\xi > 1$). Since a data point is wrongly classified if the value of its slack variable is larger than 1, $\sum_i \xi_i$ is an upper bound on the number of classification errors. The new goal is to maximize the margin i.e. minimize $||\mathbf{w}||^2$ but penalizing classification errors with the term $C\sum_i \xi_i$. The parameter $C > 0$ controls the tradeoff between margin maximization and error minimization. The optimization problem is now

$$\begin{array}{cc} \min\limits_{w,b} & \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{l}\xi_i \\ \text{s.t} & y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 + \xi_i \geq 0 \quad \forall i \\ & \xi_i \geq 0 \quad \forall i \end{array} \tag{1.3.33}$$

The dual formulation of this problem is obtained in the same way as the linearly separable case

$$\begin{array}{cc} \max\limits_{\alpha} & \sum_i \alpha_i - \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \\ \text{s.t} & 0 \leq \alpha_i \leq C \quad \forall i \\ & \sum_i \alpha_i y_i = 0 \end{array} \tag{1.3.34}$$

1.3.2.3 Kernel SVM

Despite extending SVMs to deal with non-separable data, the decision function is still linear, that is, the surface that separates both classes is an hyperplane. Most problems in practice are in fact non-linear and thus it is useful to extend the SVM formulation for this type of problems. The non-linear SVM is based on the kernel trick explained below.

The kernel trick comes from the idea of transforming the input variables into another set of features in a higher dimensional space, where the data points are linearly separable. More formally, let $\phi(\mathbf{x}_i)$ be a function that takes a point in a d-dimensional space and returns another point in a D-dimensional space, $D >> d$. If ϕ is choosen correctly then we obtain another problem that is linearly separable in this new feature space.

However, if we try to compute explicitly the value of $\phi(\mathbf{x}_i)$, we run into two main problems, one of them practical and another one theoretical:

1. The feature space can have a very large dimension, even infinite.
2. It can be very computationally expensive to compute the mapping values every time they are needed, or even storing them in memory.

The kernel trick comes from the observation that the dual function (1.3.34) only depends on the $\mathbf{x}$ through dot products, that is, they always appear in pairs. If we define the kernel function as

$$k(\mathbf{x}_i\mathbf{x}_j) = \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j) \tag{1.3.35}$$

the problem can be rewritten

$$\begin{array}{cc} \max\limits_{\alpha} & \sum_i \alpha_i - \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j k(\mathbf{x}_i\mathbf{x}_j) \\ \text{s.t} & 0 \leq \alpha_i \leq C \quad \forall i \\ & \sum_i \alpha_i y_i = 0 \end{array} \tag{1.3.36}$$

Therefore it is not neccessary to know the mapping function ϕ but only the kernel function k. The only condition is that the kernel function must be decomposed into a dot product of two functions. Mercer's theorem states which types of kernels can be used, although in practice there are a few known kernels that met Mercer's condition and they are the most used. An example is the RBF kernel

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{||\mathbf{x}_i - \mathbf{x}_j||^2}{2\sigma^2}\right) \tag{1.3.37}$$

The value of $\mathbf{w}$ in the transformed space is

$$\mathbf{w} = \sum_i \alpha_i y_i \phi(\mathbf{x}_i)$$

and, usually, it cannot be computed explicitly. However, a new point $\mathbf{x}$ can be classified using again the kernel trick

$$\mathbf{w}^t\phi(\mathbf{x}) = \sum_i \alpha_i y_i k(\mathbf{x}_i, \mathbf{x}) \tag{1.3.38}$$

Note also that only patterns with $\alpha_i > 0$ (support vectors) are needed for the classification of a new point, which is a nice property.

1.3.2.4 SVMs for regression

The full kernel SVM problem (1.3.36) is a binary classfication problem, since the possible values for the targets are either -1 or $+1$. However, the SVM formulation can be extended in order to solve regression problems, and we are going to do so in this section.

SVMs applied to a regression problem perform a linear regression in the feature space using the ϵ-insensitive loss as cost function and, at the same time, regularizing the weights so large values are penalized (more on regularization in section 1.4). It is important to notice that the feature space is different from the input space, and thus depending on the kernel the problem could be non-linear on the input variables. The ϵ-insensitive loss is defined as

$$L_\epsilon(y, f(\mathbf{x}, \mathbf{w})) = \begin{cases} 0 & \text{if } |y - f(\mathbf{x}, \mathbf{w})| \leq \epsilon \\ |y - f(\mathbf{x}, \mathbf{w})| - \epsilon & \text{otherwise} \end{cases} \tag{1.3.39}$$

and the SVM for regression solves the following optimization problem

$$\begin{array}{cc} \min\limits_{w,b} & \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{l}(\xi_i + \xi_i^*) \\ \text{s.t} & y_i - f(\mathbf{x}_i, \mathbf{w}) - b \leq \epsilon + \xi_i^* \\ & f(\mathbf{x}_i, \mathbf{w}) + b - y_i \leq \epsilon + \xi_i \\ & \xi_i, \xi_i^* \geq 0. \end{array} \tag{1.3.40}$$

where the parameter C determines the tradeoff between model complexity and the amount of deviations larger than ϵ allowed, and the parameter ϵ controls the width of the ϵ-*insensitive* tube. It can be proven that a good value for ϵ has to be proportional to the noise of the input value, i.e. $\epsilon \propto \sigma$ [5]. However, ϵ also depends on the number of samples in the training set.

The selection of the hyperparameters is a big problem, since the SVM performance depends greatly on their value. One possible option is to use cross-validation over a 3D grid of (C, σ, ϵ) values, selecting the tuple with the lowest error.

1.4 Regularization

1.4.1 Ill-posed problems

In the previous sections, given some data $\mathcal{D}$, we minimize some kind of error function $E_{\mathcal{D}}(x)$. However, as we mentioned before, minimizing the error term alone is often an ill-posed problem: solutions are not unique and sensitive to data variations. Therefore, a regularization term is usually added to enforce some desirable properties on the solution, such us smoothness, sparsity, low-rank, and so on, changing the criterion function to

$$J(x) = \underbrace{E_{\mathcal{D}}(x)}_{\text{error}} + \underbrace{\omega(x)}_{reg.}. \tag{1.4.1}$$

Another common problem solved by regularization is known as *overfitting*, and it occurs when a statistical model is too complex and it exhibits poor generalization. This means that the model is memorizing the particular structure of the training data, but it is unable to learn the underlying relationship. As a result, such model will perform poorly on new unseen data, also known as test data. An example can be seen in figure 1.4.1.

1.4.2 Bias-variance tradeoff

Let's assume that data comes from a model

$$Y = f(X) + \epsilon \tag{1.4.2}$$

with $\mathbb{E}[\epsilon] = 0$ and $\text{Var}(\epsilon) = \sigma^2$. For simplicity we also assume that the values of $\mathbf{x}$ are fixed in advance. Then, the expected prediction error of an estimator $\hat{f}(X)$ at a point x, also know as simply prediction, test or generalization error is defined as

$$\text{PE} = \mathbb{E}[(Y - \hat{f}(x))^2] \tag{1.4.3}$$

and it can be decomposed into

$$\begin{aligned}\text{PE} &= \left(\mathbb{E}[\hat{f}(x)] - f(x)\right)^2 + E\left[\hat{f}(x) - \mathbb{E}[\hat{f}(x)]\right]^2 + \text{Var}(\epsilon) \\ &= \left(\mathbb{E}[\hat{f}(x)] - f(x)\right)^2 + E\left[\hat{f}(x) - \mathbb{E}[\hat{f}(x)]\right]^2 + \sigma^2 \\ &= \text{Bias}^2 + \text{Variance} + \text{Noise}\end{aligned}$$

The third term is the irreducible error, the inherent uncertainty from noise in the true relationship and cannot be reduced by any model. The first and second terms are under our control and make up the *mean squared error* of $\hat{f}(x)$ in estimating $f(x)$.

In practice, for every model there is a *bias-variance tradeoff* [6], when varying its "complexity" parameters. For instance in the model from figure 1.4.1 we could consider the complexity parameter to be the degree of the polynomial d used to fit the points. If $d \geq 11$ the model is very complex and it fits the training points perfectly, i.e, the bias is very close to 0. However, the variance is very large, since for any other point not in the training set the model will perform poorly. On the other hand, if $d = 2$, the model is much

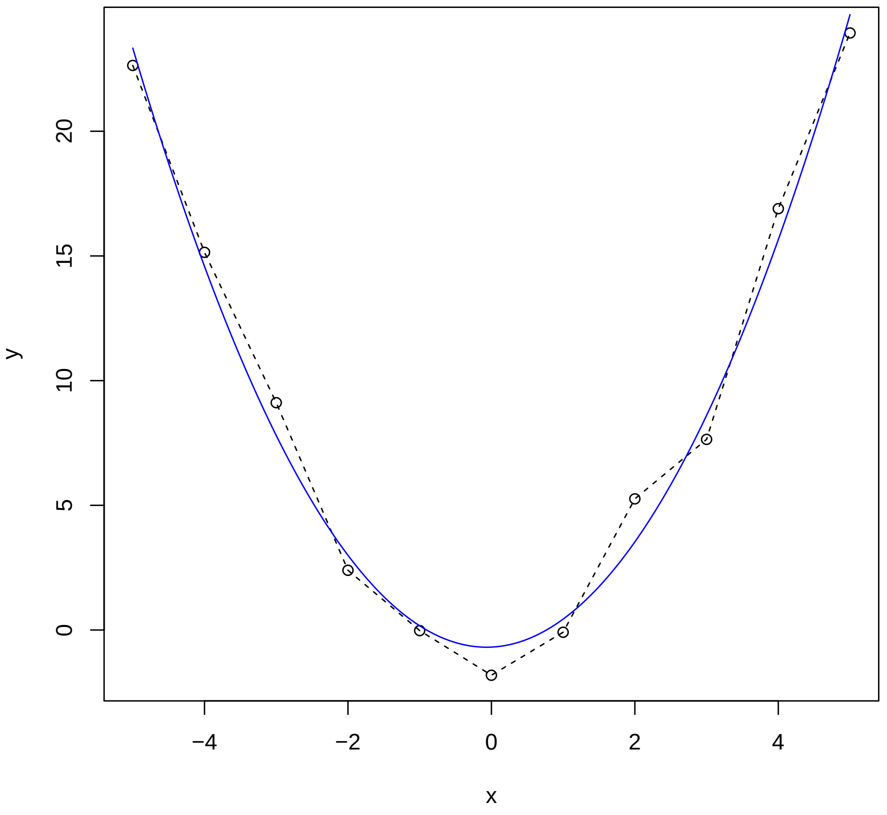


Figure 1.4.1: Points are generated from a random model $y = x^2 + \mathcal{N}(0, 2)$ and they are fitted to both quadratic (blue solid line) and polynomial functions (black dashed line). The high-order polynomial fits perfectly the training data, but it will exhibit poor generalization, since it is also learning the random error or noise of the data.

simpler. Now the bias is clearly not zero but the variance is reduced significantly, so we may consider this to be a better model.

Typically we would like to choose our model complexity to trade bias off with variance in such a way as to minimize the test error, which is the error of new observations (not used to fit the model). Figure 1.4.2 shows the typical behavior of train and test error as model complexity is varied. The training error tends to decrease whenever we increase the model complexity, overfitting the data. However, with too much fitting the model adapts too closely and will not generalize well. In contrast, if the model is not complex enough, it will underfit and may have large bias.

As mentioned earlier, a common approach to control the complexity of the problem is through regularization. We present next some common regularization techniques involving the l_2 penalty. More recent methods involving the l_1 penalty and mixed penalties will be described further in chapter 2.

1.4.3 Ridge regression

Consider the linear model with noise $\epsilon \sim N(0, \sigma)$

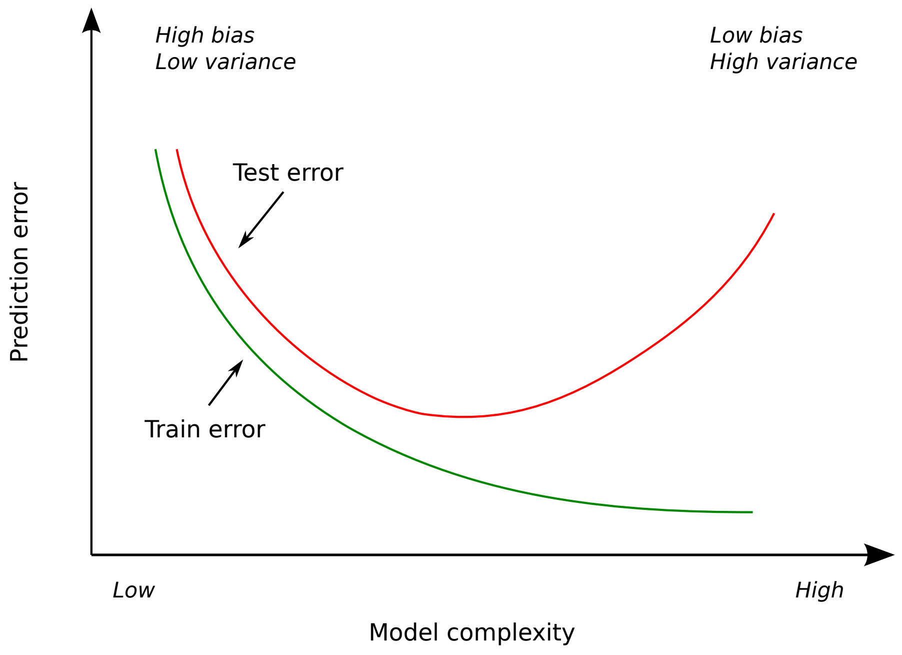


Figure 1.4.2: Test and training error as a function of the model complexity [1]

$$\mathbf{y} = \mathbf{X}\mathbf{w} + \boldsymbol{\epsilon}, \tag{1.4.4}$$

we are going to prove that the least squares estimates are unbiased. Substitute (1.4.4) into (2.2.3)

$$\begin{aligned}\widehat{\mathbf{w}} &= (\mathbf{X}^t\mathbf{X})^{-1}\mathbf{X}^t\mathbf{y}\\ &= (\mathbf{X}^t\mathbf{X})^{-1}\mathbf{X}^t(\mathbf{X}\mathbf{w} + \boldsymbol{\epsilon})\\ &= \mathbf{w} + (\mathbf{X}^t\mathbf{X})^{-1}\mathbf{X}^t\boldsymbol{\epsilon}\end{aligned}$$

Now fix the data matrix $\mathbf{X}$ and take expectations on both sides,

$$\mathbb{E}[\widehat{\mathbf{w}}] = \mathbf{w} + \mathbb{E}[(\mathbf{X}^t\mathbf{X})^{-1}\mathbf{X}^t\boldsymbol{\epsilon}] = \mathbf{w} + (\mathbf{X}^t\mathbf{X})^{-1}\mathbf{X}^t\mathbb{E}[\boldsymbol{\epsilon}] = \mathbf{w}, \tag{1.4.5}$$

since $\mathbb{E}[\boldsymbol{\epsilon}] = 0$ and the $\mathbf{X}$ are fixed. In addition of being unbiased estimates, the Gauss-Markov theorem [7] also states that in a linear regression model they are the best ones, where "best" means giving the lowest variance. Let $\mathrm{Var}(y) = \sigma^2$, then the variance of the OLS estimates is

$$\begin{aligned}\mathrm{Var}(\widehat{\mathbf{w}}) &= \mathrm{Var}\left((\mathbf{X}^t\mathbf{X})^{-1}\mathbf{X}^t\mathbf{y}\right)\\ &= ((\mathbf{X}^t\mathbf{X})^{-1}\mathbf{X}^t)\,\mathrm{Var}(\mathbf{y})\left((\mathbf{X}^t\mathbf{X})^{-1}\mathbf{X}^t\right)^t\\ &= \mathrm{Var}(\mathbf{y})((\mathbf{X}^t\mathbf{X})^{-1}\mathbf{X}^t)(\mathbf{X}(\mathbf{X}^t\mathbf{X})^{-1})\\ &= \sigma^2(\mathbf{X}^t\mathbf{X})^{-1}\end{aligned}$$

Now we are going to prove the Gauss-Markov theorem. Let $\tilde{\mathbf{w}}$ be another linear estimator,

$$\tilde{\mathbf{w}} = \left((\mathbf{X}^t\mathbf{X})^{-1}\mathbf{X}^t + \mathbf{D}\right)\mathbf{y}, \tag{1.4.6}$$

where $\mathbf{D}$ is a $p \times n$ non-zero matrix. The expectation of $\tilde{\mathbf{w}}$ is

$$\begin{aligned}
\mathbb{E}[\tilde{\mathbf{w}}] &= \mathbb{E}[((\mathbf{X}^t\mathbf{X})^{-1}\mathbf{X}^t + \mathbf{D})\mathbf{y}] \\
&= \mathbb{E}[((\mathbf{X}^t\mathbf{X})^{-1}\mathbf{X}^t + \mathbf{D})(\mathbf{X}\mathbf{w} + \boldsymbol{\epsilon})] \\
&= \mathbb{E}[((\mathbf{X}^t\mathbf{X})^{-1}\mathbf{X}^t + \mathbf{D})(\mathbf{X}\mathbf{w})] + \mathbb{E}[(\mathbf{X}^t\mathbf{X})^{-1}\mathbf{X}^t + \mathbf{D})\boldsymbol{\epsilon}] \\
&= ((\mathbf{X}^t\mathbf{X})^{-1}\mathbf{X}^t + \mathbf{D})(\mathbf{X}\mathbf{w}) + ((\mathbf{X}^t\mathbf{X})^{-1}\mathbf{X}^t + \mathbf{D})\mathbb{E}[\boldsymbol{\epsilon}] \\
&= (\mathbf{X}^t\mathbf{X})^{-1}\mathbf{X}^t\mathbf{X}\mathbf{w} + \mathbf{D}\mathbf{X}\mathbf{w} \\
&= (\mathbf{I} + \mathbf{D}\mathbf{X})\mathbf{w}.
\end{aligned}$$

From this we conclude that $\mathbf{DX} = 0$ for this estimator to be unbiased. The variance of $\tilde{\mathbf{w}}$ is

$$\begin{aligned}
\mathrm{Var}(\tilde{\mathbf{w}}) &= \mathrm{Var}\left(\left((\mathbf{X}^t\mathbf{X})^{-1}\mathbf{X}^t + \mathbf{D}\right)\mathbf{y}\right) \\
&= \mathrm{Var}(\mathbf{y})((\mathbf{X}^t\mathbf{X})^{-1}\mathbf{X}^t + \mathbf{D})((\mathbf{X}^t\mathbf{X})^{-1}\mathbf{X}^t + \mathbf{D})^t \\
&= \sigma^2((\mathbf{X}^t\mathbf{X})^{-1}\mathbf{X}^t + \mathbf{D})(\mathbf{X}(\mathbf{X}^t\mathbf{X})^{-1} + \mathbf{D}^t) \\
&= \sigma^2((\mathbf{X}^t\mathbf{X})^{-1}\mathbf{X}^t\mathbf{X}(\mathbf{X}^t\mathbf{X})^{-1} + (\mathbf{X}^t\mathbf{X})^{-1}\mathbf{X}^t\mathbf{D}^t + \mathbf{D}\mathbf{X}(\mathbf{X}^t\mathbf{X})^{-1} + \mathbf{D}\mathbf{D}^t) \\
&= \sigma^2((\mathbf{X}^t\mathbf{X})^{-1} + (\mathbf{X}^t\mathbf{X})^{-1}(\mathbf{D}\mathbf{X})^t + \mathbf{D}\mathbf{D}^t) \\
&= \sigma^2((\mathbf{X}^t\mathbf{X})^{-1} + \mathbf{D}\mathbf{D}^t) \\
&= \sigma^2((\mathbf{X}^t\mathbf{X})^{-1} + \sigma^2\mathbf{D}\mathbf{D}^t \\
&= \mathrm{Var}(\widehat{\mathbf{w}}) + \sigma^2\mathbf{D}\mathbf{D}^t \\
&\geq \mathrm{Var}(\widehat{\mathbf{w}}),
\end{aligned}$$

since $\mathbf{DD}^t$ is a positive definite matrix, $\mathbf{DD}^t \geq 0$. Therefore the least squares estimators $\widehat{\mathbf{w}}$ are the unbiased linear estimators with the lowest variance, or BLUE (best linear unbiased estimator).

However, in practice there are many reason why the least square estimates are not a satisfactory solution to the regression problem. Recall that one of these reasons was the non-uniqueness of the OLSsolution when the number of variables is larger than the number of observations.

Ridge regression its a regularized version of linear regression, that adds a l_2 penalty term to the standard least squares objective function

$$\widehat{\mathbf{w}} = \operatorname{argmin}\left\{\sum_{i=1}^{n}\left(y_i - \sum_j w_j x_{ij}\right)^2 + \lambda\sum_j w_j^2\right\} = \operatorname{argmin}\left\{\|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 + \lambda\|\mathbf{w}\|_2^2\right\}. \tag{1.4.7}$$

Via the Lagrangian form, it can be seen [1] that the previous optimization problem is equivalent to

$$\min \quad \sum_{i=1}^{N}\left(y_i - \sum_j w_j x_{ij}\right)^2 \qquad \text{s.t} \quad \sum_j w_j^2 \leq t. \tag{1.4.8}$$

Using similar arguments to the ordinary least squares case, it is easy to show that ridge regression has also a closed form solution

$$\widehat{\mathbf{w}} = (\mathbf{X}^t\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^t\mathbf{y} \tag{1.4.9}$$

Note that, in contrast to ordinary least squares, now the matrix $(\mathbf{X}^t\mathbf{X} + \lambda\mathbf{I})$ is always positive definite and therefore invertible, resulting in a unique solution for the ridge estimates.

1.4.4 Weight decay

Weight decay adds a l_2 penalty term to the multilayer perceptron error function (1.3.8)

$$E = \frac{1}{2}\sum_{k=1}^{K}(y_k - t_k)^2 + \frac{\lambda}{2}\sum w_{ji}^2. \tag{1.4.10}$$

In a linear model, this form of weight decay is equivalent to ridge regression. The new derivative of the error function with respect to the weights is

$$\frac{\partial E}{\partial w_{ji}} = \delta_j \Phi(a_i) + \lambda w_{ji}, \tag{1.4.11}$$

and the gradient descent step is changed as follows

$$\mathbf{w}_{ji}^{(t+1)} = \mathbf{w}_{ji}^{(t)} - \eta\delta_j\Phi(a_i) - \eta\lambda w_{ji}^{(t)}. \tag{1.4.12}$$

Therefore, in addition to each weight update by backpropagation, the weight is also decreased by a part $\eta\lambda$ of its old value. An additional problem with this approach is that we have to select the value of the parameter λ, since the generalization hability of the network can depend crucially on this constant. One possible approach is to train several networks with different values for λ and select the one with the lowest generalization error. This error can be estimated either by using cross-validation or a whole new different validation set.

Another practical consideration for getting good results with weight decay is to standarize both inputs and targets so they are in the same range. It is also a good idea to omit the biases from the penalization [8].

We should also mention that there is two more meta-parameters whose values should be selected: the network architecture (number of hidden neurons) and the η constant. This may result in cross-validation not being computationally feasible, since we have to try with every triplet of parameters in a 3D grid. However, most modern algorithms such as conjugate gradient automatically select the optimal value for η in each step using a linear search. This means that we only have a 2D grid of values to optimize. In chapter 4 we will also discuss some other methods to select the number of hidden units.

1.4.5 SVM as a Penalization Method

Consider the optimization problem with $f(x) = \phi(x)^t\mathbf{w} + b$

$$\min_{\mathbf{w}} \sum_{i=1}^{n}[1 - y_i f(x_i)]_+ + \frac{\lambda}{2}\|\mathbf{w}\|^2 \tag{1.4.13}$$

where the subscript "+" indicates the positive part. This has the form *loss* + *penalty* (equation (1.4.1)) and it can be shown that the solution to (1.4.13), with $\lambda = \frac{1}{C}$, is the same as that for (1.3.33) [1].

The loss function $L(y, f(x)) = [1 - yf(x)]_+$ is known as the "hinge" loss and it is a particular case of the ϵ-insensitive loss (1.3.39) used for SVR with $\epsilon = 0$. Figure 1.4.3 shows that it is reasonable for two-class classification, when compared with the logistic regression loss function.

The conclusion of this section is that SVM can be considered as a regularization problem, where the coefficients $\mathbf{w}$ are shrunk towards 0 (excluding the bias). Similarly, the SVR problem can be seen as a regularized version of the ϵ-insensitive loss [1].

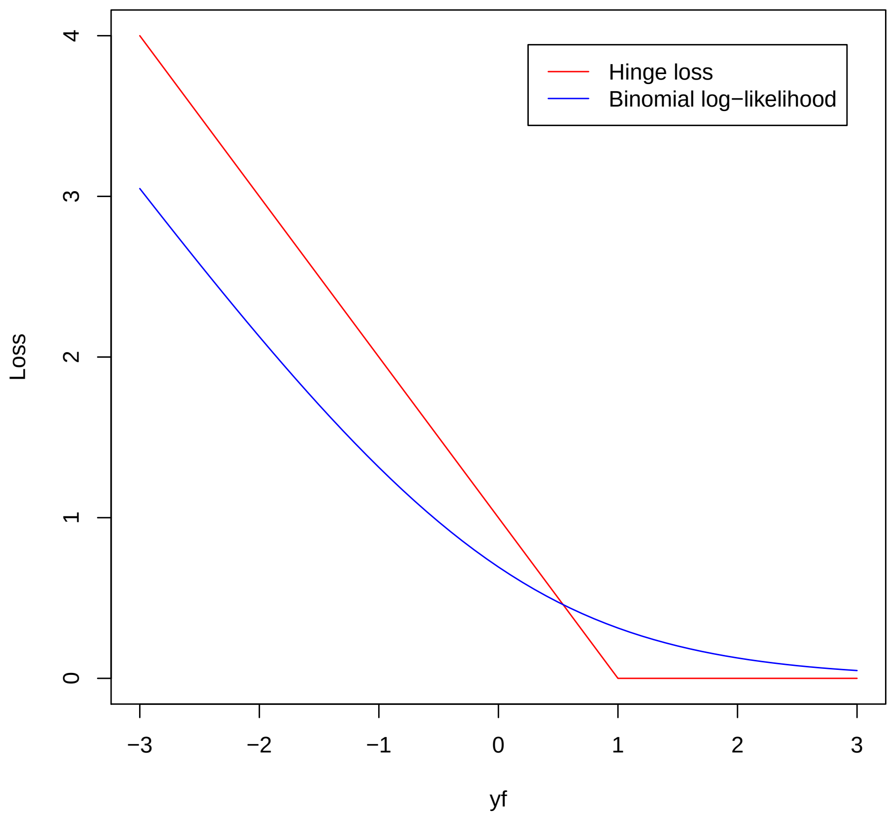


Figure 1.4.3: Support Vector Machines loss function (hinge loss) compared to the logistic regression loss (negative log-likelihood loss). They are shown as a function of yf rather than f because of the symmetry between the $y = +1$ and $y = -1$ case [1].

Chapter 2

Classic sparse models

2.1 Introduction

Let's consider the usual regression problem

$$\mathbf{y} = \mathbf{X}\mathbf{w} + \epsilon \tag{2.1.1}$$

where $\mathbf{X}$ is the $n \times p$ input matrix (x_{ij} denotes the j–th coordinate of the pattern i), $\mathbf{y}$ is the $n \times 1$ output vector, $\epsilon \sim N(0, \sigma)$ some white noise, n is the number of patterns and p is the number of features. Recall from chapter 1 that ordinary least squares (OLS) estimates are obtained as

$$\widehat{\mathbf{w}} = \operatorname{argmin} \left\{ \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 \right\}. \tag{2.1.2}$$

There are at least two main reasons why these estimates may not be very useful in most cases:

1. *Accuracy of the predictions*: OLS estimates usually have low bias but high variance (in the common case where $n >> p$). In some cases accuracy increases by reducing or even setting to 0 some coefficients. This increases the bias a little bit but reduces variance to a great extent.
2. *Interpretability*: When the number of predictors is big, sometimes it is interesting to find the subset that affects outputs the most. These estimates do not provide this information.

The two standard techniques to improve OLS estimates, *subset selection* [1] and *ridge regression* (chapter 1), have disadvantages. Subset selection obtains interpretable models but it can be very variable since it is a discrete process: predictors are either kept in the model or dropped from it. On the other hand, ridge regression is a continuous process, in the sense that all coefficients are shrunk (but not dropped) and hence more robust. However, it is very rare that coefficients become strictly 0 and therefore models are not interpretable.

2.2 Lasso

Lasso (Least Absolute Shrinkage and Selection Operator) is a technique that reduces (shrinks) some coefficients and sets others to 0; therefore it tries to maintain the advantages of both subset selection and ridge regression. Let's assume, without loss of generality, that the x_{ij} are normalized to have zero mean and unit variance, that is, $\sum_i x_{ij}/n = 0$,

$\sum_i x_{ij}^2/n = 1$ and the output variables y_i have zero mean. Let $\widehat{\mathbf{w}} = (\widehat{w}_1, \widehat{w}_2, \ldots, \widehat{w}_p)^t$ be the lasso estimate, defined by

$$\widehat{\mathbf{w}} = \operatorname{argmin} \left\{ \sum_{i=1}^{n} \left(y_i - \sum_j w_j x_{ij} \right)^2 \right\} \qquad \text{s.t} \quad \sum_j |w_j| \leq t \tag{2.2.1}$$

where t is a tuning parameter. This parameter controls the amount of shrinking that is applied to the estimates. Let $\widehat{w}_j^0$ be the full OLS estimates and $t_0 = \sum_j |\widehat{w}_j^0|$. Values of $t < t_0$, will cause shrinking of the coefficients towards 0, and some coefficients may be strictly 0. For instance if $t = t_0/2$, the effect will be similar to finding the best subset of size $p/2$ [9]. The motivation for the lasso came from and interesting proposal of Breiman, the non-negative garotte [10]

$$\min \sum_{i=1}^{n} \left(y_i - \sum_j c_j \widehat{w}_j^0 x_{ij} \right)^2 \qquad \text{s.t} \quad c_j \geq 0, \quad \sum c_j \leq t. \tag{2.2.2}$$

The garotte starts with the full OLS estimates ($\widehat{w}_j^0$) and shrinks them using non-negative factors (c_j) whose sum is bounded. The $\widehat{w}_j(t) = c_j \widehat{w}_j^0$ are the new predictor coefficients. As t is decreased, more of the $\{c_j\}$ become 0, and the remaining non-zero $\widehat{w}_j(t)$ are shrunk.

Orthonormal case The orthonormal design case is a highly simplified situation (it is never the case in real-world datasets) where lasso, subset selection, non-negative garrote and ridge regression solutions can be computed exactly. This can give interesting insight into the comparative behaviour of those methods.

Recall that $\mathbf{X}$ is the $n \times p$ matrix with ijth entry x_{ij} and suppose that $\mathbf{X}^t\mathbf{X} = \mathbf{I}$. Let $\widehat{\mathbf{w}}^0$ be the ordinary least squares solution (chapter 1)

$$\widehat{\mathbf{w}}^0 = (\mathbf{X}^t\mathbf{X})^{-1}\mathbf{X}^t\mathbf{y} \tag{2.2.3}$$

In the case of an orthogonal design, equation (2.2.3) simplifies to

$$\widehat{\mathbf{w}}^0 = \mathbf{X}^t\mathbf{y}.$$

Using Lagrangian theory, it can be seen [1] that the lasso problem (2.2.1) is equivalent to

$$\widehat{\mathbf{w}} = \operatorname{argmin} \left\{ \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 + \lambda \|\mathbf{w}\|_1 \right\}, \tag{2.2.4}$$

with $\lambda \geq 0$. Note that the lasso penalty is convex, but not strictly convex (refer to chapter 3 for the definitions). Expanding the first term of the previous equation and using the fact that $\mathbf{X}$ is orthonormal, we get $\mathbf{y}^t\mathbf{y} - 2\mathbf{y}^t\mathbf{X}\mathbf{w} + \mathbf{w}^t\mathbf{w}$. Since $\mathbf{y}^t\mathbf{y}$ does not contain any of the variables to minimize and noting that $\widehat{\mathbf{w}}^0 = \mathbf{X}^t\mathbf{y}$, we can rewrite the problem (2.2.4) as

$$\widehat{\mathbf{w}} = \operatorname{argmin} \left\{ 2\widehat{\mathbf{w}}^0\mathbf{w} + \|\mathbf{w}\|_2^2 + \lambda\|\mathbf{w}\|_1 \right\} = \operatorname{argmin} \left\{ \sum_j 2w_j^0 w_j + w_j^2 + \lambda |w_j| \right\} \tag{2.2.5}$$

The objective function is now a sum of objectives, each corresponding to a separate variable w_j, so they may be solved individually. Fixing a certain j, then we want to minimize

$$2w_j^0 w_j + w_j^2 + \lambda |w_j|.$$

Note that if $w_j^0 > 0$ then $w_j \geq 0$ since otherwise we could flip the sign and get a lower value for the objective function. Similarly, if $w_j^0 < 0$ then $w_j \leq 0$. For the first case, differentiating with respect to w_j and setting equal to 0 we get

$$w_j = w_j^0 - \frac{\lambda}{2}.$$

But this is only positive if the right-hand side is non-negative, that is, we have to take

$$\widehat{w}_j = \left(\widehat{w}_j^0 - \frac{\lambda}{2}\right)^+ = \operatorname{sign}(\widehat{w}_j^0)\left(|\widehat{w}_j^0| - \frac{\lambda}{2}\right)^+ \tag{2.2.6}$$

Using a similar argument the same solution is obtained for the $w_j^0 < 0$ case. Finally, letting $\gamma = \frac{\lambda}{2}$ we get that the lasso solutions for the orthonormal case are

$$\widehat{w}_j = \operatorname{sign}(\widehat{w}_j^0)(|\widehat{w}_j^0| - \gamma)^+ \tag{2.2.7}$$

where γ is determined by the condition $\sum \widehat{w}_j = t$. In fact, for every $t \geq 0$ it is possible to find a $\gamma \geq 0$ for which the solution of the problem is the same [11].

In this scenario, the best subset selection of size k reduces to choosing the k largest coefficients in absolute value and setting the rest to 0. This is equivalent to

$$\widehat{w}_j = \begin{cases} \widehat{w}_j^0 & \text{if } |\widehat{w}_j^0| > \lambda \\ 0 & \text{otherwise} \end{cases}$$

for some choice of λ [9]. Recall from chapter 1 that ridge regression solutions are

$$\widehat{\mathbf{w}} = (\mathbf{X}^t\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^t\mathbf{y}.$$

If $\mathbf{X}^t\mathbf{X} = \mathbf{I}$ and noting again that $\widehat{\mathbf{w}}^0 = \mathbf{X}^t\mathbf{y}$, the solutions for the orthonormal case can be rewritten as

$$\widehat{w}_j = \frac{1}{1+\lambda}\widehat{w}_j^0.$$

On the other hand, the solutions for the garotte are [10]

$$\widehat{w}_j = \left(1 - \frac{\lambda^2}{(\widehat{w}_j^0)^2}\right)^+ \widehat{w}_j^0.$$

Figure 2.2.1 shows the comparison between lasso, ridge regression, garotte and best subset selection solutions as a function of the OLS estimates. As it can be seen the lasso often obtains some coefficients equal to 0, while shrinking the rest towards 0. Best subset selection also obtains coefficients equal to 0, but the rest remain unchanged. The garotte is similar to the lasso, with less shrinkage for larger coefficients. According to [9], the differences can be large if the design is not orthonormal. Finally, rigde regression shrinks all coefficients, but none of them is strictly 0.

Geometry Now let's look at the lasso penalty from a geometrical point of view. Figure 2.2.2 shows the elliptical contours of the objective function, centered at the OLS estimates. The lasso solution is the first place where these estimates intersect with the unit square. Sometimes this will happen at a corner and thus that coefficient will be 0. In the ridge regression case, the constrain region is a circle and hence zero solutions will rarely result. As it can be deduced from the figure, ridge regression contour plots intersect with the circle in a point $(0, y)$ only if the corresponding OLS estimate is already in the y-axis, i.e. the coefficient is already 0, which is a much more unlikely event.

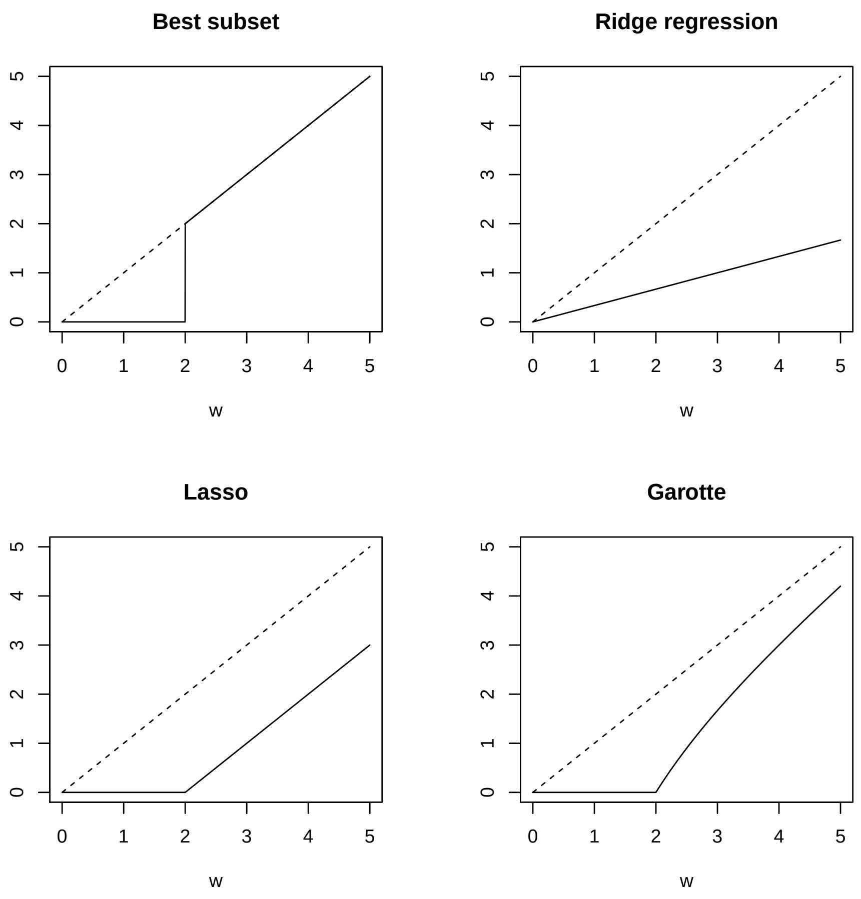


Figure 2.2.1: Subset regression, ridge regression, lasso and garotte shrinkage comparison in the case of an orthonormal design.

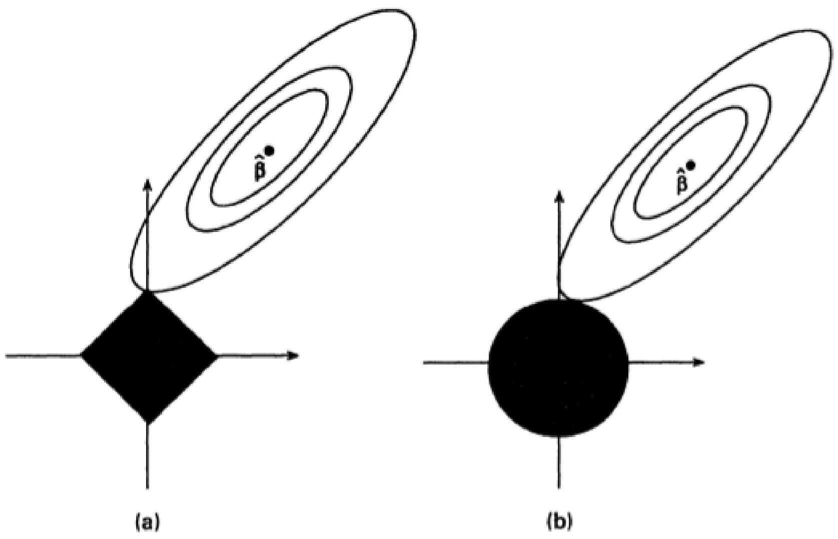


Figure 2.2.2: Lasso (a) and ridge regression (b) estimates [9]

2.2.1 Prediction error and estimation of t

In this section we describe the three methods proposed in [9] in order to estimate the lasso parameter t: cross-validation, generalized cross-validation and an analytical unbiased estimation of risk.

Cross-validation Let's assume a linear model

$$\mathbf{y} = \eta(\mathbf{X}) + \boldsymbol{\epsilon}$$

where $\mathbb{E}(\boldsymbol{\epsilon}) = 0$ and $var(\boldsymbol{\epsilon}) = \sigma^2$. The mean squared error of an estimator $\widehat{\eta}(\mathbf{X})$ is defined by

$$\text{MSE} = \mathbb{E}\{(\widehat{\eta}(\mathbf{X}) - \eta(\mathbf{X}))^2\}. \tag{2.2.8}$$

The prediction error is a similar measure, already defined in chapter 1, that may be written in terms of the MSE

$$\begin{aligned}
\text{PE} &= \mathbb{E}\{(\mathbf{y} - \widehat{\eta}(\mathbf{X}))^2\} \\
&= \mathbb{E}\{(\eta(\mathbf{X}) + \boldsymbol{\epsilon} - \widehat{\eta}(\mathbf{X}))^2\} \\
&= \mathbb{E}\{(\eta(\mathbf{X}) - \widehat{\eta}(\mathbf{X}))^2\} + 2\mathbb{E}\{(\eta(\mathbf{X}) - \widehat{\eta}(\mathbf{X}))\boldsymbol{\epsilon}\} + E\{\boldsymbol{\epsilon}^2\} \\
&= \mathbb{E}\{(\widehat{\eta}(\mathbf{X}) - \eta(\mathbf{X}))^2\} + \sigma^2 \\
&= MSE + \sigma^2
\end{aligned} \tag{2.2.9}$$

The empirical prediction error for the lasso is estimated using 5-fold cross-validation. The parameter used is a normalized version of t, $s = t/\sum \widehat{w}_j^0$, and the prediction error is computed for values of s from 0 to 1, both included. Finally, the value $\widehat{s}$ with the lowest PEis selected.

The advantages of using s instead of t is that it lies in the interval $[0, 1]$, since t is an upper bound on the weights, $\sum |w_j| \leq t$. As we have seen before, values of $t < \sum \widehat{w}_j^0$ will shrink the OLS coefficients towards 0. The first value for which there is no shrinkage is $t = \sum \widehat{w}_j^0$ and it corresponds to $s = 1$. For smaller values of s the amount of shrinkage increases until we reach the extreme trivial case $s = 0$, where all the weights are also 0.

The main disadvantage of using the normalized parameter s is that first we have to perform an ordinary least squares regression on the data in order to obtain the weights $\widehat{w}_j^0$. Although there is a closed form solution for these estimates, it involves computing an inverse, which may not exist (see equation (2.2.3)). Moreover, even if we can compute the inverse theoretically, it may be very expensive or even not possible at all in practice, due to memory limitations (especially if the data has very high dimension).

Generalized cross-validation This method is derived from a linear approximation of the lasso estimate. The constraint $\sum |w_j| \leq t$ can be rewritten as $\sum w_j^2/|w_j| \leq t$. As we have seen before, using Lagrangian theory, this is equivalent to adding the penalty $\lambda \sum w_j^2/|w_j|$ to the residual sum of squares, with λ depending on t. The solution for this problem can be written as the ridge regression estimator [9]

$$\tilde{\mathbf{w}} = (\mathbf{X}^t\mathbf{X} + \lambda\mathbf{W}^+)^{-1}\mathbf{X}^t\mathbf{y} \tag{2.2.10}$$

where $\mathbf{W} = \text{diag}(|\tilde{w}_j|)$ and $\mathbf{W}^+$ denotes the generalized inverse.

Therefore the number of effective parameters in $\tilde{\mathbf{w}}$ can be approximated by

$$p(t) = \text{tr}\{\mathbf{X}(\mathbf{X}^t\mathbf{X} + \lambda\mathbf{W}^+)^{-1}\mathbf{X}^t\}.$$

Letting $\mathrm{rss}(t)$ be the residual sum of squares, the following statistic is defined in [9]

$$\mathrm{GCV}(t) = \frac{1}{n}\frac{\mathrm{rss}(t)}{\{1 - p(t)/n\}^2} \tag{2.2.11}$$

Finally, the parameter t with the lowest value of the statistic $\mathrm{GCV}(t)$ is selected.

Given a discrete grid with, for example, 15 different values of the parameter, it is important to notice that this methods only needs 15 applications of the optimization procedure, while the previous method needs $15 \times 5 = 75$, assuming we use 5−fold cross validation.

Unbiased estimator of risk Let $\widehat{\tau} = \widehat{\sigma}/\sqrt{n}$ be the estimated standard error of the OLS model defined by $\widehat{w}_j^0$, where $\widehat{\sigma}^2 = \sum (y_i - \widehat{y}_i)^2/(n-p)$. Then, according to [9], $\widehat{w}_j^0/\widehat{\tau}$ are approximately independent standard normal variables (conditioned to $\mathbf{X}$). Using Stein's unbiased estimate of risk [12], the following formula can be derived

$$R\{\widehat{\mathbf{w}}(\gamma)\} \approx \widehat{\tau}^2 \left\{ p - 2\#(j; \widehat{w}_j^0/\widehat{\tau} < \gamma) + \sum_{j=1}^{p} \max(\widehat{w}_j^0/\widehat{\tau}, \gamma)^2 \right\}$$

as an unbiased estimator of the risk or mean-squared error $E[(\widehat{\mathbf{w}}(\gamma) - \mathbf{w})^2]$, where $\widehat{w}_j(\gamma) = \mathrm{sign}(\widehat{w}_j^0)(|\widehat{w}_j^0/\widehat{\tau}| - \gamma)^+$. Therefore an estimator for γ can be obtained as the minimizer of $R\{\widehat{\mathbf{w}}(\gamma)\}$:

$$\widehat{\gamma} = \underset{\gamma \geq 0}{\mathrm{argmin}}[R\{\widehat{\mathbf{w}}(\gamma)\}].$$

Assuming an orthonormal design, we can obtain an estimator of the lasso parameter t

$$\widehat{t} = \sum (|\widehat{w}_j^0| - \widehat{\gamma})^+.$$

Although the previous derivation for $\widehat{t}$ assumes an orthogonal design, we can still try to use it in a non-orthogonal setting. In [9] we can find simulated examples where this method gives useful estimates of t, but they only provide heuristic arguments in favor of it.

2.2.2 Algorithm to find lasso solutions

If we set $t \geq 0$, the problem (2.2.1) can be expressed as a least squared problem with 2^p linear inequality constraints, corresponding to the 2^p possible signs of the coefficients w_j.

More precisely, let $g(\mathbf{w}) = \sum_{i=1}^{n} (y_i - \sum_j w_j x_{ij})^2$ and $\boldsymbol{\delta}_i$, $i = 1, 2, \ldots, 2^p$ be p−tuples of the form $(\pm 1, \pm 1, \ldots, \pm 1)$. Then the condition $\sum |w_j| \leq t$ is equivalent to $\boldsymbol{\delta}_i \mathbf{w} \leq t$ for all i.

There are procedures such as Lawson and Hansen (1974) that solve the least-squares problem subject to a general inequality restriction $G\mathbf{w} \leq \mathbf{h}$, where G is an $m \times p$ matrix corresponding to the m inequality restrictions of the p-vector $\mathbf{w}$. For this problem $m = 2^p$ is a very large value and this procedure cannot be applied directly. However, this problem can be approached introducing the inequality restrictions sequentially.

In more detail, for a given $\mathbf{w}$ and $\boldsymbol{\delta}_i$ as before, let us define $E = \{i : \boldsymbol{\delta}_i \mathbf{w} = t\}$, that is, the set of restrictions that are fulfilled exactly. The set $S = \{i : \boldsymbol{\delta}_i \mathbf{w} < t\}$ corresponds to the rest of the restrictions. Let G_E be the matrix whose rows are $\boldsymbol{\delta}_i$ for $i \in E$. Finally, let $\mathbf{1}$ be a vector of 1 whose size is equal to the number of rows in G_E. The pseudo-code is given in algorithm 1.

The algorithm converges in a finite number of steps, since in each one a new element is added to the set E, and there is a total of 2^p elements. The final result of the algorithm is

Algorithm 1: Lawson and Hansen (1974).

```
**Start** with $E = \{i_0\}$ where $\boldsymbol{\delta}_{i_0} = \mathrm{sign}(\widehat{\mathbf{w}}^0)$, with $\widehat{\mathbf{w}}^0$ the full OLS estimates.
**Find** $\widehat{\mathbf{w}}$ that minimizes $g(\mathbf{w})$, subject to $G_E\mathbf{w} \leq t\mathbf{1}$
**While** $\sum |\widehat{w}_j| > t$
    **Add** $i$ to the set $E$, where $\boldsymbol{\delta}_i = \mathrm{sign}(\widehat{\mathbf{w}})$
    **Find** $\widehat{\mathbf{w}}$ that minimizes $g(\mathbf{w})$ subject to $G_E\mathbf{w} \leq t\mathbf{1}$
**End**
```

a solution to the original problem since it can be shown that the Kuhn-Tucker conditions are fulfilled for the sets E and S [9].

Although the algorithm is guaranteed to terminate in at most 2^p iterations, if p is large this rate of convergence is a limitation. However, in practice the average number of iterations needed by the algorithm lies in the range $(0.5p, 0.75p)$.

2.3 Fused lasso

2.3.1 Introduction

Fused lasso is a modification of the lasso where feature ordering is taken into account. If we denote with $\mathbf{x}_j$ the jth column of the design matrix $\mathbf{X}$, we now assume that the x_{ij} are realizations of features $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_p$ that can be ordered in a meaningful manner. The goal is to predict y given $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_p$, so the coefficients w are also grouped in blocks of consecutive values. In particular, we are interested in problems where $p >> n$. We start with a standard linear model

$$y_i = \sum_j x_{ij} w_j + \epsilon_i \tag{2.3.1}$$

where the errors ϵ_i have zero mean and σ^2 variance. We also assume that the predictors are standardized to have zero mean and unit variance, and the outputs have zero mean.

It is important to notice that p is much larger than n in the applications we are interested in, so in these problems the standard linear regression (OLS) is ill–conditioned and prone to over-fitting. One possible solution for this is the lasso [9]. Recall that the lasso finds the solution to the problem (2.2.1), where the bound t is a tunning parameter. For sufficiently large t we obtain the OLS solution, or one of the many if $p > n$. For small values of t, the solutions are sparse, that is, some coefficients are exactly 0. In contrast, ridge regression does not produce sparse models. Subset selection does produce sparse models but it is not a convex operation, since it can be understood as l_0 constrained regression. Moreover, best subset selection is essentially a combinatorial problem and therefore it is not computationally feasible for, say, $p > 30$.

The lasso can be applied even if $p > n$, and it has an unique solution assuming that two predictors are not co-linear [13]. Another interesting property of the solution is that the number of coefficients different from 0 is at most $\min(p, n)$ [14], appendix A. A disadvantage of the lasso is that it ignores feature ordering. In order to solve this problem we define the fussed lasso as

$$\widehat{\mathbf{w}} = \mathrm{argmin} \left\{ \sum_{i=1}^{n} \left(y_i - \sum_j w_j x_{ij} \right)^2 \right\} \quad \text{s.t} \sum_j |w_j| \leq s_1 \text{ and } \sum_{j=2}^{p} |w_j - w_{j-1}| \leq s_2 \tag{2.3.2}$$

The first constraint is the standard Lasso one that encourages sparsity in the coefficients, while the second encourages dispersity in their differences; s_1 y s_2 are tuning parameters that have to be estimated.

2.3.2 Computational approach

First we are going to show how the fused lasso can be reduced to a Quadratic Program. Assuming fixed s_1 and s_2, the lasso objective function is a quadratic programming problem. This type of problems are difficult to solve if p is large. In order to solve it efficiently one option is the algorithm SQOPT from Gill *et al.* (1997). This algorithm is designed for quadratic problems with sparse linear constraints and we write next the fused lasso in this way.

Let's consider the decomposition $w_j = w_j^+ - w_j^-$, where $w_j^+, w_j^- \geq 0$. Notice that, if $w_j > 0$ then $w_j^+ = |w_j|$, $w_j^- = 0$ and if $w_j < 0$ then $w_j^+ = 0$, $w_j^- = |w_j|$. We define

$$\boldsymbol{\theta} = \begin{cases} w_1 & \text{if } j = 1 \\ w_j - w_{j-1} & \text{if } j > 1 \end{cases}$$

Consider also $\theta_j = \theta_j^+ - \theta_j^-$ with $\theta_j^+, \theta_j^- \geq 0$ and let $\mathbf{L}$ be the matrix

$$\mathbf{L} = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 & 0 \\ -1 & 1 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & -1 & 1 \end{pmatrix}$$

so $\boldsymbol{\theta} = \mathbf{Lw}$ (notice that $\mathbf{L}$ is invertible). If $\mathbf{X}$ is the $n \times p$ data matrix, $\mathbf{w}$ the p-vector of coefficients and $\mathbf{y}$ the n-vector of outputs, the problem (2.3.2) can be rewritten as

$$\widehat{\mathbf{w}} = \operatorname{argmin}\{(\mathbf{y} - \mathbf{Xw})^t(\mathbf{y} - \mathbf{Xw})\} \tag{2.3.3}$$

subject to

$$\begin{pmatrix} -\mathbf{a}_0 \\ \mathbf{0}^t \\ 0 \\ 0 \end{pmatrix} \leq \begin{pmatrix} \mathbf{L} & \mathbf{0} & \mathbf{0} & -\mathbf{I} & \mathbf{I} \\ \mathbf{I} & -\mathbf{I} & \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{e}^t & \mathbf{e}^t & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{e}_0^t & \mathbf{e}_0^t \end{pmatrix} \begin{pmatrix} \mathbf{w} \\ \mathbf{w}^+ \\ \mathbf{w}^- \\ \boldsymbol{\theta}^+ \\ \boldsymbol{\theta}^- \end{pmatrix} \leq \begin{pmatrix} \mathbf{a}_0 \\ \mathbf{0}^t \\ s_1 \\ s_2 \end{pmatrix}, \tag{2.3.4}$$

in addition to the non-negativity constraints $\mathbf{w}^+, \mathbf{w}^-, \boldsymbol{\theta}^+, \boldsymbol{\theta}^- \geq 0$, where e is a column vector with p 1s and I is the $p \times p$ identity matrix. Here $\mathbf{a}_0 = (\infty, 0, 0, \dots, 0)$. Since $w_1 = \theta_1$, setting its bounds to $\pm\infty$ avoids a double penalty for $|w_1|$. Similarly $e_0 = (0, e)$, i.e., we add a first component equal to 0 to e. Notice also that the larger matrix has dimension $(2p + 2) \times 5p$ and the $\mathbf{0}$s are in fact row vectors of dimension p with only 0s.

Now we are going to check that the previous problem is in fact equivalent to the fused lasso problem. Starting with the objective function, by simple algebra

$$\sum_{i=1}^{n} \left(y_i - \sum_j w_j x_{ij} \right)^2 = \|\mathbf{y} - \mathbf{Xw}\|_2^2 = (\mathbf{y} - \mathbf{Xw})^t(\mathbf{y} - \mathbf{Xw}).$$

Next, in order to see why the constraints are equivalent, note the following

$$\sum_j |w_j| = \sum_j |w_j^+ - w_j^-| = \sum_j w_j^+ + w_j^- = \sum_j w_j^+ + \sum_j w_j^-.$$

Similarly,

$$\sum_{j=2}^{p} |w_j - w_{j-1}| = \sum_{j=2}^{p} |\theta_j| = \sum_{j=2}^{p} \theta_j^+ + \sum_{j=2}^{p} \theta_j^- .$$

Finally, multiplying the matrix in equation (2.3.4), we get as the last two rows

$$\begin{pmatrix} 0 \\ 0 \end{pmatrix} \leq \begin{pmatrix} \mathbf{e}^t\mathbf{w}^+ + \mathbf{e}^t\mathbf{w}^- \\ \mathbf{e}_0^t\boldsymbol{\theta}^+ + \mathbf{e}_0^t\boldsymbol{\theta}^- \end{pmatrix} \leq \begin{pmatrix} s_1 \\ s_2 \end{pmatrix}, \tag{2.3.5}$$

which are the same as the fused lasso constraints.

Next we discuss the search strategy to find s_1 and s_2. For small problems ($p \simeq 1000$ y $N \simeq 100$), the previous procedure is fast enough to use it over a bi-dimensional grid of s_1 and s_2 values, with a moderate s resolution. For larger problems, it is necessary to use a more restrictive search. First we exploit the fact that the complete path of lasso ($s_2 = \infty$) and fusion solutions ($s_1 = \infty$) can be solved efficiently using the Least Angle Regression (LARS) algorithm [15].

More precisely, the lasso problem is solved by a simple modification of LARS described in [15]. In order to solve the fusion problem, first we transform it into a lasso-type problem. Let $\mathbf{L}$ be the same matrix as before and create then a new data matrix $\mathbf{Z} = \mathbf{X}\mathbf{L}^{-1}$. The optimization problem

$$\widehat{\boldsymbol{\theta}} = \operatorname{argmin}\{\|\mathbf{y} - \mathbf{Z}\boldsymbol{\theta}\|_2^2 + s_2|\boldsymbol{\theta}|\}$$

is a lasso-type problem with weights $\boldsymbol{\theta} = \mathbf{L}\mathbf{w}$, and it can be solved using again the LARS procedure. The solution to the original fusion problem is obtained by transforming back the weights, that is, $\widehat{\mathbf{w}} = \mathbf{L}^{-1}\widehat{\boldsymbol{\theta}}$.

Note that for a given problem only some values of the bounds (s_1, s_2) are possible. It is also important to mention that the degrees of freedom of the lasso fit g can be estimated as the number of non-zero coefficients [16], and they change as the parameter s_1 is varied. Therefore we write $s_1(g)$ to refer to a parameter s_1 for which the lasso has g degress of freedom, that is, g non-zero coefficients.

Since solutions for the constraints $(s_1(g), \infty)$ can be found efficiently using the least angle regression (LARS) procedure, first we use cross-validation in order to estimate an optimal value for the degrees of freedom $\widehat{g}$. As we will see later, LARS is an iterative procedure that starts with all the coefficients equal to 0 and they join the model in sucesive steps. Therefore, to obtain a solution with g non-zero coefficients we just simply stop the algorithm exactly when g coefficients have joined the model (we refer to section 2.5 for detailed information on the LARS algorithm). Now let

$$s_{2\max}\{s_1(\widehat{g})\} = \sum_j |\widehat{w}_j(s_1(\widehat{g})) - \widehat{w}_{j-1}(s_1(\widehat{g}))|.$$

Once s_1 is fixed, this is the largest value of the bound s_2 which affects the solution. Finally, the following points are defined in a bi-dimensional grid

$$\begin{aligned} c_1 &= (s_1(\widehat{g}/2), s_{2\max}\{s_1(\widehat{g}/2)\}), \\ c_2 &= (s_1(\widehat{g}), s_{2\max}\{s_1(\widehat{g})\}), \\ c_3 &= (s_1(\{\widehat{g} + \min(n,p)\}/2), s_{2\max}\{s_1(\{\widehat{g} + \min(n,p)\}/2)\}). \end{aligned}$$

The search strategy consists in starting on those points and solve the fused lasso problem for each pair of parameters, moving in the direction $(1, -2)$. This search direction is proposed in [17] where they mention it was found after extensive empirical experiments.

2.3.3 Fused lasso degrees of freedom and solution sparsity

It is useful to consider how many degrees of freedom has the fused lasso as s_1 and s_2 vary. The following definition of degrees of freedom is considered in [15]

$$\mathrm{df}(\widehat{y}) = \frac{1}{\sigma^2}\sum_{i=1}^{N}\mathrm{cov}(y_i, \widehat{y}_i) \tag{2.3.6}$$

where σ^2 is the variance of y_i. For standard linear regression with $p < n$, $\mathrm{df}(\widehat{y}) = \mathrm{tr}(\widehat{y}) = \mathrm{tr}(\mathbf{X}\widehat{\mathbf{w}}) = p$, assuming that the matrix $\mathbf{X}$ has full rank, i.e., $\mathrm{rank}(\mathbf{X}) = p$. In an orthonormal design, the lasso estimates are simply the soft-thresholding estimates (2.2.7) and [15] shows that the degrees of freedom are equal to the number of coefficients different from 0. For the fused lasso, [17] propose the following estimate of the degrees of freedom

$$\mathrm{df}(\widehat{y}) = \#\{\text{non-zero coefficient blocks in } \widehat{\mathbf{w}}\}.$$

This is equivalent to count as 1 degree of freedom every consecutive non-zero sequence of one or more equal w_j-values.

Regarding the sparsity of the fused lasso solutions, we have mentioned before that if $p > n$, lasso solutions will have at most n non-zero coefficients. Fused lasso has a similar property, which applies not to the number of non-zero coefficients but rather to the the number of sequences of identical non-zero coefficients. Let $w_0 = 0$ and $n_{\mathrm{seq}}(\mathbf{w}) = \sum_{j=1}^{p}\mathbf{1}\{w_j \neq w_{j-1}\}$. Then, under certain no-redundancy conditions on the matrix $\mathbf{X}$ the fused lasso problem (2.3.2) has a unique solution $\widehat{\mathbf{w}}$ with $n_{\mathrm{seq}}(\widehat{\mathbf{w}}) \leq n$ [17].

2.4 Elastic Net

We consider the usual regression model (2.1.1) with p predictors. As shown before there are several options to find the weight vector $\widehat{\mathbf{w}}$, such as OLS, ridge regression or the lasso. It is well known that OLS estimates often do poorly in both prediction and interpretation. As a continuous shrinkage method, ridge regression achieves better prediction performance, although it does not produce sparse and, hence, interpretable models.

The lasso fixes both problems, since it does simultaneously continuous shrinkage and automatic variable selection. Some works ([9] and [18]) compare the prediction performance of the lasso and ridge regression and found that none uniformly dominates over the other. However, as data is rapidly increasing in both number and dimension, lasso is more appealing due to its sparse representation. Although lasso has shown success in many situations, it has some limitations:

1. In the $p > n$ case, the lasso selects at most n variables, because of the nature of the convex optimization problem [15]. From a variable selection method point of view, this seems like a limiting feature. However, from a regression method point of view this is not a bad property since the problem is ill-posed.

2. If there is a group of variables for which pairwise correlations are very high, then the lasso tends to select somewhat randomly only one variable from the group CITE.

3. In the $n > p$ case, if there are high correlations between predictors, it has been empirically observed that the prediction of ridge regression is better than that of the lasso.

Moreover, for some problems, such as the gene selection problem in microarray data, the lasso may not be a good variable selection method [19]. A typical microarray data has many thousands of variables (genes) and often fewer than 100 samples. There are also some genes with high correlations between them, forming a group. The ideal gene selection method should be able to eliminate the trivial genes and automatically include whole groups into the model. For this kind of $p >> n$ and grouped variables situation, the lasso is not the ideal method, because it can only select at most n variables out of p candidates and it lacks the ability to reveal grouping information.

The elastic net is a method that tries to improve on both lasso and ridge regression as it simultaneously does automatic selection and continuous shrinkage like the lasso, but it can also select groups of correlated variables. It also lacks the limitation of selecting at most n features out of p when $p > n$.

2.4.1 Naïve elastic net

Definition Suppose that the data set has n observations and p predictors. Let $\mathbf{y} \in \mathbb{R}^n$ be the response vector and $\mathbf{X} \in \mathbb{R}^{n \times p}$ the data matrix. As in the lasso, we assume that the response has zero mean and the predictors are standarized. The naïve elastic net criterion is defined as

$$L(\mathbf{w}, \lambda_1, \lambda_2) = ||\mathbf{y} - \mathbf{X}\mathbf{w}||_2^2 + \lambda_2||\mathbf{w}||_2^2 + \lambda_1||\mathbf{w}||_1 \tag{2.4.1}$$

where we recall that

$$\|\mathbf{w}\|_2^2 = \sum_{j=1}^{p} w_j^2 \tag{2.4.2}$$

$$\|\mathbf{w}\|_1 = \sum_{j=1}^{p} |w_j|. \tag{2.4.3}$$

The naïve elastic net estimator $\widehat{\mathbf{w}}$ is the minimizer of equation (2.4.1):

$$\widehat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmin}}\{L(\mathbf{w}, \lambda_1, \lambda_2)\}. \tag{2.4.4}$$

Let $\alpha = \lambda_2/(\lambda_1 + \lambda_2)$, then equation (2.4.4) is equivalent to [19]

$$\widehat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmin}} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2, \qquad \text{s.t. } (1-\alpha)\|\mathbf{w}\|_1 + \alpha\|\mathbf{w}\|_2^2 \le t. \tag{2.4.5}$$

The function $(1-\alpha)\|\mathbf{w}\|_1 + \alpha\|\mathbf{w}\|_2^2$ is the elastic net penalty, which is a convex combination of the lasso penalty (l_1-norm) and the ridge regression penalty (l_2-norm). When $\alpha = 1$, the naïve elastic net becomes simply ridge regression. For all $\alpha \in (0, 1)$ the elastic net penalty is strictly convex. Recall that the lasso penalty ($\alpha = 0$) is convex but not strictly convex and non-differenciable. See figure 2.4.1 for a graphical representation of the elastic-net penalty.

We discuss next how to solve the elastic net problem. It turns out that minimizing equation (2.4.1) is equivalent to a lasso-type optimization problem.

Lemma 2.4.1. *Given a dataset* $(\mathbf{X}, \mathbf{y})$ *and* (λ_1, λ_2), *define an artificial dataset* $(\widetilde{\mathbf{X}}, \widetilde{\mathbf{y}})$ *by*

$$\widetilde{\mathbf{X}}_{(n+p)\times p} = (1+\lambda_2)^{-1/2} \begin{pmatrix} \mathbf{X} \\ \sqrt{\lambda_2}\mathbf{I}_p \end{pmatrix}, \qquad \widetilde{\mathbf{y}}_{(n+p)} = \begin{pmatrix} \mathbf{y} \\ 0 \end{pmatrix}. \tag{2.4.6}$$

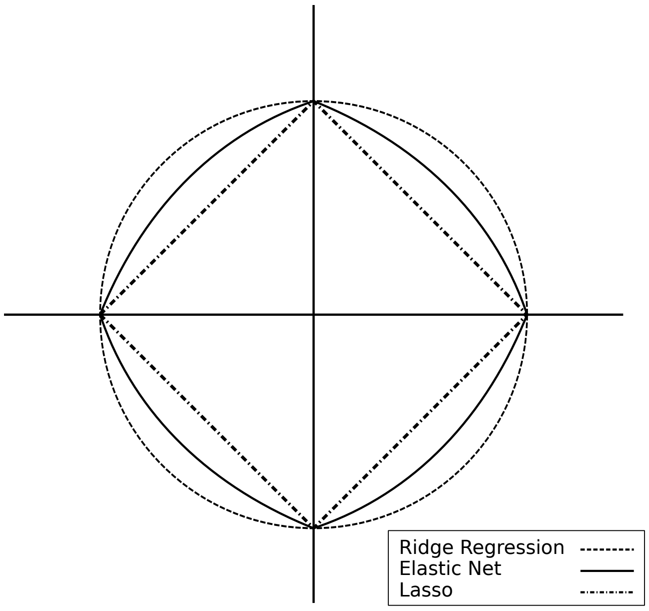


Figure 2.4.1: Lasso, ridge regression and elastic net penalties for the two variable case

Let $\gamma = \lambda_1/\sqrt{(1+\lambda_2)}$ and $\widetilde{\mathbf{w}} = \sqrt{(1+\lambda_2)}\mathbf{w}$. Then the naïve elastic net criterion can be written as

$$L(\mathbf{w},\gamma) = L(\widetilde{\mathbf{w}},\gamma) = \|\widetilde{\mathbf{y}} - \widetilde{\mathbf{X}}\widetilde{\mathbf{w}}\|_2^2 + \gamma\|\widetilde{\mathbf{w}}\|_1. \tag{2.4.7}$$

Moreover, let

$$\widetilde{\mathbf{w}}^* = \underset{\widetilde{\mathbf{w}}}{\operatorname{argmin}}\{L(\widetilde{\mathbf{w}},\gamma)\};$$

then the elastic net optimum $\widehat{\mathbf{w}}$ is given by

$$\widehat{\mathbf{w}} = \frac{1}{\sqrt{(1+\lambda_2)}}\widetilde{\mathbf{w}}^*.$$

Proof. We are going to show how to recover the criterion function (2.4.1) from the function (2.4.7). First, substituting the values of γ and $\widetilde{\mathbf{w}}$ in equation (2.4.7) we get

$$\begin{aligned} L(\widetilde{\mathbf{w}},\gamma) &= \left\|\widetilde{\mathbf{y}} - \widetilde{\mathbf{X}}\sqrt{(1+\lambda_2)}\mathbf{w}\right\|_2^2 + \frac{\lambda_1}{\sqrt{(1+\lambda_2)}}\left\|\sqrt{(1+\lambda_2)}\mathbf{w}\right\|_1 \\ &= \left\|\widetilde{\mathbf{y}} - \widetilde{\mathbf{X}}\sqrt{(1+\lambda_2)}\mathbf{w}\right\|_2^2 + \lambda_1\left\|\mathbf{w}\right\|_1. \end{aligned}$$

Now substituting the values of $\widetilde{\mathbf{X}}$ and $\widetilde{\mathbf{y}}$ into the previous equation we get

$$\begin{aligned} L(\mathbf{w},\lambda_1,\lambda_2) &= \left\|\begin{pmatrix}\mathbf{y}\\0\end{pmatrix} - (1+\lambda_2)^{-1/2}\begin{pmatrix}\mathbf{X}\\\sqrt{\lambda_2}\mathbf{I}_p\end{pmatrix}\sqrt{(1+\lambda_2)}\mathbf{w}\right\|_2^2 + \lambda_1\left\|\mathbf{w}\right\|_1 \\ &= \left\|\begin{pmatrix}\mathbf{y}\\0\end{pmatrix} - \begin{pmatrix}\mathbf{X}\mathbf{w}\\\sqrt{\lambda_2}\mathbf{w}\end{pmatrix}\right\|_2^2 + \lambda_1\left\|\mathbf{w}\right\|_1 \\ &= \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 + \| - \lambda_2^{1/2}\mathbf{w}\|_2^2 + \lambda_1\left\|\mathbf{w}\right\|_1 \\ &= \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 + \lambda_2\|\mathbf{w}\|_2^2 + \lambda_1\|\mathbf{w}\|_1, \end{aligned}$$

which is exactly the same as equation (2.4.1) □

Lemma 2.4.1 shows that we can transform the naïve elastic net problem into a equivalent lasso problem on augmented data. Note that the sample size in the augmented problem is $n+p$ and $\widehat{\mathbf{X}}$ has rank p, which means that the naïve elastic net can potentially select all p predictors in all situations. This property overcomes the limitation of the lasso that were described in section 2.2. Lemma 2.4.1 also shows that naïve elastic net can perform an automatic variable selection in a fashion similar to the lasso.

Let's briefly discuss now the grouping effect in the elastic net model. In the $p \gg n$ case, the grouped variables situation is particularly important, for the reasons mentioned in section 2.4. Qualitatively speaking, a regression method exhibits the grouping effect if the regression coefficients of a group of highly correlated variables tend to be equal in absolute value. In particular, in the extreme situation where some variables are exactly identical, the regression method should assign identical coefficients to the identical variables. In order to quantify the grouping effect, in [19] they consider the following generic penalization problem

$$\widehat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmin}}\{\|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 + \lambda J(\mathbf{w})\} \tag{2.4.8}$$

where $J(\cdot)$ is positive valued for $\mathbf{w} \neq 0$. The following lemma characterizes the grouping effect depending on the penalty term $J(\cdot)$.

Lemma 2.4.2. *Assume that $\mathbf{x}_i = \mathbf{x}_j$, $i \neq j \in \{1, \dots, p\}$.*

1. *If $J(\cdot)$ strictly convex, then $\widehat{w}_i = \widehat{w}_j$, $\forall \lambda > 0$.*
2. *If $J(\mathbf{w}) = \|\mathbf{w}\|_1$, then $\widehat{w}_i\widehat{w}_j \geq 0$ and for any $s \in [0,1]$, the weights $\widehat{\mathbf{w}}^*(s)$*

$$\widehat{w}_k^*(s) = \begin{cases} \widehat{w}_k & \text{if } k \neq i \text{ and } k \neq j, \\ (\widehat{w}_i + \widehat{w}_j)\ s & \text{if } k = i, \\ (\widehat{w}_i + \widehat{w}_j)(1-s) & \text{if } k = j, \end{cases}$$

are also minimizers of equation (2.4.8).

Lemma 2.4.2 shows a clear distinction between strictly convex penalty functions and the lasso penalty. Strict convexity guarantees the grouping effect, i.e., equal coefficients in the extreme situation with identical predictors. In contrast the lasso does not even have an unique solution. The elastic net penalty is strictly convex for $\lambda_2 > 0$, thus enjoying lemma 2.4.2 property 1.

Theorem 2.4.1. *Given data $(\mathbf{X}, \mathbf{y})$ and parameters (λ_1, λ_2), assume the $\mathbf{y}$ has zero mean and the predictors $\mathbf{X}$ are standarized. Let $\widehat{\mathbf{w}}(\lambda_1, \lambda_2)$ be the naïve elastic net estimate. Suppose that $\widehat{w}_i(\lambda_1,\lambda_2)\widehat{w}_j(\lambda_1,\lambda_2) > 0$ and define*

$$D_{\lambda_1,\lambda_2}(i,j) = \frac{1}{\|\mathbf{y}\|_1}|\widehat{w}_i(\lambda_1,\lambda_2) - \widehat{w}_j(\lambda_1,\lambda_2)|;$$

then

$$D_{\lambda_1,\lambda_2}(i,j) \leq \frac{1}{\lambda_2}\sqrt{2(1-\rho_{ij})},$$

where $\rho_{ij} = \mathbf{x}_i^t\mathbf{x}_j$ is the $\mathbf{x}_i, \mathbf{x}_j$ sample correlation.

Proofs of lemma 2.4.2 and theorem 2.4.1 can be found in [19], appendix A. The quantity $D_{\lambda_1,\lambda_2}(i,j)$ describes the difference between the coefficients paths of predictors i and j. If $\mathbf{x}_i$ and $\mathbf{x}_j$ are highly correlated, theorem 2.4.1 says that the difference between the coefficient paths of predictor i and predictor j is almost 0.

2.4.2 Elastic net

As an automatic variable selection model, the naïve elastic net overcomes the limitations of the lasso in scenarios (1) and (2). However, empirical evidence (see [19] sections 4 and 5) shows that elastic net may not perform well unless it is very close to either ridge regression or the lasso.

As we have mentioned before in section 1, an accurate penalization method achieves good prediction performance through the bias-variance trade-off. The naïve elastic net estimator is a two stage procedure: for each fixed λ_2 first the ridge regression coefficients are found and then a lasso-type shrinkage is done along the lasso coefficient solution paths. This suggests that a double amount of shrinkage is being performed in the coefficients. This double shrinkage does not help to reduce the variance much and introduces unnecessary extra bias.

The elastic net estimate In order to improve the prediction performance of the naïve elastic net estimate this double shrinkage has to be corrected. Given data $(\mathbf{X}, \mathbf{y})$, penalty parameters (λ_1, λ_2), and augmented data $(\widetilde{\mathbf{X}}, \widetilde{\mathbf{y}})$, the naïve elastic net solves the lasso-type problem

$$\widetilde{\mathbf{w}}^* = \operatorname*{argmin}_{\widetilde{\mathbf{w}}} \left\{ \|\widetilde{\mathbf{y}} - \widetilde{\mathbf{X}}\widetilde{\mathbf{w}}\|_2^2 + \frac{\lambda_1}{\sqrt{(1+\lambda_2}} \|\widetilde{\mathbf{w}}\|_1 \right\} \tag{2.4.9}$$

The corrected elastic net estimates are defined in [19] by

$$\widehat{\mathbf{w}}(\text{elastic net}) = \sqrt{(1+\lambda_2)}\widetilde{\mathbf{w}}^* \tag{2.4.10}$$

Recalling that we had $\widehat{\mathbf{w}}(\text{naïve elastic net}) = (1/\sqrt{(1+\lambda_2)})\widetilde{\mathbf{w}}^*$, it follows that

$$\widehat{\mathbf{w}}(\text{elastic net}) = (1+\lambda_2)\widehat{\mathbf{w}}(\text{naïve elastic net}) \tag{2.4.11}$$

Hence the elastic net coefficients are a rescaled version of the naïve elastic net coefficients. Such an scaling transformation preserves the variable selection property of the naïve elastic net and it is the simplest way to undo the double shrinkage. Empirically it was also found in [19] that the elastic net performs very well when compared with the lasso and ridge regression. We also see that behavior in the experimental results of chapter 4.

2.5 LARS

In the next chapter we will see how Lasso, Elastic Net and similar methods can be considered from the point of view of convex optimization. In particular, we will discuss advanced modern methods that yield general algorithms to solve all these problems. But before doing so, we will discuss in this section Least Angle Regression (LARS) [15], the first algorithmic procedure to solve lasso and related problems. LARS is a model selection algorithm related to others such as the Lasso, Forward Stagewise regression and Forward Stepwise regression. In fact, simple modifications of the LARS procedure implement algorithms to solve both problems using less computer time.

One of the first model selection method was Forward Stepwise regression [20]. Given a collection of possible predictors, we select the one having largest absolute correlation with the response y, say x_1, and perform a linear regression of y on x_1. This leaves a residual vector orthogonal to x_1 that is now considered to be the response. We project the other predictors orthogonally to x_1 and repeat the selection process. After k steps we have a set of predictors $x_1, x_2, \dots x_k$ that can be used to construct a linear model with k parameters.

This procedure may be overly greedy since useful predictors can be discarded too early if they are very correlated to previously selected x_i predictors.

Next we describe Forward Stagewise regression [15], which is a much more cautious version of Forward Stepwise regression, since it makes a lot of tiny steps (more than k) as it moves towards a final model. Forward Stagewise starts with $\widehat{\mathbf{w}} = 0$ and builds up the regression function in successive small steps. If $\widehat{\mathbf{w}}$ is the current estimated weight vector and $\widehat{\mathbf{y}} = \mathbf{X}\widehat{w}$, let $c(\widehat{\mathbf{y}})$ be the vector of current correlations, i.e.,

$$\widehat{c} = c(\widehat{\mathbf{y}}) = \mathbf{X}^t(\mathbf{y} - \widehat{\mathbf{y}}) \tag{2.5.1}$$

so that its j–th component $\widehat{c}_j$ is proportional to the correlation between covariate x_j and the current residual vector. The next step of the algorithm is taken in the direction of the largest current correlation j, i.e., we first find j^* as

$$j^* = \text{argmax}\,|\widehat{c}_j|$$

and then we update the estimate

$$\widehat{\mathbf{y}} = \widehat{\mathbf{y}} + \boldsymbol{\epsilon} \cdot \text{sign}(\widehat{y}_{j^*})\mathbf{x}_{j^*} \tag{2.5.2}$$

with ϵ some small constant. Notice that if $\epsilon = |\widehat{c}_j|$ this reduces to the Forward Stepwise algorithm and the number of iterations is equal to k, the number of parameters in the final model. As ϵ approaches 0, the algorithm is going to increase the number of iterations (taking more computer time), but possibly also its accuracy, since it will not discard early useful predictors very correlated with the response.

LARS is an intermediate approach [15], since a simple formula allows to implement Forward Stagewise with fairly large steps, although not as large as those in Forward Stepwise, reducing the computational burden. LARS builds up the estimates $\widehat{\mathbf{y}} = X\widehat{w}$ in k steps, adding one covariate to the model in each step, so at the end only k of the $\widehat{w}_j$ are non-zero. First we start with $\widehat{w} = 0$ and find the predictor most correlated with the response using (2.5.1)-(2.5.2). We take the largest step possible in that direction until some other predictor has as much correlation with the current residual. At this point, LARS proceeds in a direction equiangular to the current predictors until a third variable earns its way into the most correlated set, and so on.

More formally, we begin at $\widehat{\mathbf{w}} = 0$, that is, $\widehat{\mathbf{y}}_0 = 0$, and suppose that $\widehat{\mathbf{y}}_{\mathcal{A}}$ is the current LARS output estimate. Then, the vector of current residual correlations is (2.5.1)

$$\widehat{c} = X^t(\mathbf{y} - \widehat{\mathbf{y}}_{\mathcal{A}}).$$

The *active set* $\mathcal{A}$ is the set of indices corresponding to covariates with the greatest absolute current correlations,

$$\widehat{C} = \max_j\{|\widehat{c}_j|\} \qquad \text{and} \qquad \mathcal{A} = \{j : |\widehat{c}_j| = \widehat{C}\}. \tag{2.5.3}$$

Assuming that the feature vectors $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_p$ are linearly independent, we define the $n \times |\mathcal{A}|$-matrix

$$X_{\mathcal{A}} = (\dots \text{sign}(\widehat{c}_j)\mathbf{x}_j \dots)_{j \in \mathcal{A}}, \tag{2.5.4}$$

with sign being the ± 1-valued sign function. Let

$$G_{\mathcal{A}} = X^t_{\mathcal{A}} X_{\mathcal{A}} \qquad \text{and} \qquad A_{\mathcal{A}} = (\mathbf{1}^t_{\mathcal{A}} G^{-1}_{\mathcal{A}} \mathbf{1}_{\mathcal{A}})^{-1/2} \tag{2.5.5}$$

with $\mathbf{1}_{\mathcal{A}}$ a vector of 1's of lenght $|\mathcal{A}|$. Then

$$\omega_{\mathcal{A}} = A_{\mathcal{A}} G^{-1}_{\mathcal{A}} \mathbf{1}_{\mathcal{A}}$$

is the unit-norm vector making equal angles, less than 90°, to the columns of $X_{\mathcal{A}}$, that is,

$$X_{\mathcal{A}}^t \mathbf{u}_{\mathcal{A}} = A_{\mathcal{A}} \mathbf{1}_{\mathcal{A}} \qquad \text{and} \qquad ||\mathbf{u}_{\mathcal{A}}||^2 = 1. \tag{2.5.6}$$

We also consider the equiangular vector

$$\mathbf{u}_{\mathcal{A}} = X_{\mathcal{A}} \omega_{\mathcal{A}}. \tag{2.5.7}$$

and compute the vector of lenght $|\mathcal{A}|$

$$\mathbf{a} = X^t \mathbf{u}_{\mathcal{A}}. \tag{2.5.8}$$

Then the next step of the LARS algorithm selects first the new covariate $\widehat{j}$ to be added as

$$\widehat{j} = \underset{j \in \mathcal{A}^c}{\text{argmin}^+} \left\{ \frac{\widehat{C} - \widehat{c}_j}{A_{\mathcal{A}} - a_j}, \frac{\widehat{C} + \widehat{c}_j}{A_{\mathcal{A}} + a_j} \right\} = \underset{j \in \mathcal{A}^c}{\text{argmin}^+} \{\gamma_j\}$$

where by argmin$^+$ we indicate that the minimum is taken only over positive components. Then $\widehat{\mathbf{y}}_{\mathcal{A}}$ is updated as

$$\widehat{\mathbf{y}}_{\mathcal{A}^+} = \widehat{\mathbf{y}}_{\mathcal{A}} + \widehat{\gamma} \mathbf{u}_{\mathcal{A}} \tag{2.5.9}$$

where $\widehat{\gamma} = \gamma_{\widehat{j}}$, i.e.,

$$\widehat{\gamma} = \underset{j \in \mathcal{A}^c}{\min^+} \left\{ \frac{\widehat{C} - \widehat{c}_j}{A_{\mathcal{A}} - a_j}, \frac{\widehat{C} + \widehat{c}_j}{A_{\mathcal{A}} + a_j} \right\}. \tag{2.5.10}$$

Here min$^+$ indicates again that the minimum is taken only over positive components for each choice of j.

We discuss next the rationale for the above choices (see [15] for more details). Define

$$\mathbf{y}(\gamma) = \widehat{\mathbf{y}}_{\mathcal{A}} + \gamma \mathbf{u}_{\mathcal{A}} \tag{2.5.11}$$

for $\gamma > 0$, so that the new residual correlation is

$$\begin{aligned} c_j(\gamma) &= \mathbf{x}_j^t(\mathbf{y} - \mathbf{y}(\gamma)) \\ &= \mathbf{x}_j^t(\mathbf{y} - \widehat{\mathbf{y}}_{\mathcal{A}} - \gamma \mathbf{u}_{\mathcal{A}}) \\ &= \mathbf{x}_j^t(\mathbf{y} - \widehat{\mathbf{y}}_{\mathcal{A}}) - \gamma \mathbf{x}_j^t \mathbf{u}_{\mathcal{A}} \\ &= \widehat{c}_j - \gamma a_j \end{aligned} \tag{2.5.12}$$

For $j \in \mathcal{A}$, equations (2.5.1), (2.5.6) and (2.5.3) yield

$$|c_j(\gamma)| = \widehat{C} - \gamma A_{\mathcal{A}} \tag{2.5.13}$$

showing that all of the maximal absolute correlations of the features already selected will decline equally for any value of γ. For $j \in \mathcal{A}^c$, γ is selected so that the next covariate to join the active set has a new correlation exactly equal to (2.5.13). Equating (2.5.12) with (2.5.13) shows that $c_j(\gamma)$ equals the maximal value at $\gamma = (\widehat{C} - \widehat{c}_j)/(A_{\mathcal{A}} - a_j)$. Likewise $-c_j(\gamma)$ achieves its maximum at $\gamma = (\widehat{C} + \widehat{c}_j)/(A_{\mathcal{A}} + a_j)$. Therefore we want to choose $\widehat{\gamma}$ in equation (2.5.10) as the smallest possible such value of γ. In other words, we select the next covariate j to join the active set, i.e., $\mathcal{A}_+ = \mathcal{A} \cup \{\hat{j}\}$, as

$$\widehat{j} = \underset{j \in \mathcal{A}^c}{\text{argmin}^+} \left\{ \frac{\widehat{C} - \widehat{c}_j}{A_{\mathcal{A}} - a_j}, \frac{\widehat{C} + \widehat{c}_j}{A_{\mathcal{A}} + a_j} \right\}.$$

The new maximum absolute correlation $\widehat{C}_+$ verifies then $\widehat{C}_+ = \widehat{C} - \widehat{\gamma} A_{\mathcal{A}}$.

2.6 Group variants

2.6.1 Group lasso

Let's consider a modification of the regression problem (2.1.1), where the features or variables are grouped into factors

$$y = \sum_{j=1}^{J} X_j w_j + \boldsymbol{\epsilon} \tag{2.6.1}$$

where y is an $n \times 1$ vector, $\boldsymbol{\epsilon} \sim N_n(0, \sigma^2 I)$, X_j is an $n \times p_j$ matrix corresponding to the jth factor and w_j is a coefficient vector of size $p_j, j = 1, \ldots, J$. J is the number of factors and p_j is the number of subvariables in factor j

An special case of equation (2.6.1) is when $p_j = \cdots = p_J = 1$. Then, the number of "factors" is equal to p, and this problem is equivalent to equation (2.1.1). This is the most studied model selection problem, and we have already discussed some methods to solve it such as the non-negative garrotte or the Lasso. Some of this methods are described in section 2.2.

Group lasso [21] is an extension of the lasso for selecting groups of variables or factors. For a vector $\eta \in \mathbb{R}^b$, $d \geq 1$, and a symmetric $d \times d$ positive definite matrix K, we define

$$||\eta||_K = (\eta^t K \eta)^{1/2}.$$

Given positive definite matrices $K_1, \ldots, K_J$, the group lasso estimate is defined as the solution to

$$\frac{1}{2}\left\|Y - \sum_{j=1}^{J} X_j w_j\right\|^2 + \lambda \sum_{j=1}^{J} ||w_j||_{K_j} \tag{2.6.2}$$

where $\lambda \geq 0$ is a tuning parameter. It is clear that expression (2.6.2) reduces to the lasso if $p_j = \cdots = p_J = 1$. The penalty function used in expression (2.6.2) is intermediate between the l_1-penalty that is used in the lasso and the l_2-penalty that is used in ridge regression. The l_1-penalty encourages sparsity in individual coefficients and the l_2-penalty does not. The group lasso wants to encourage sparsity at the factor level.

There are many reasonable choices for the kernel matrices K_js. An obvious choice would be $K_j = I_{p_j}$, $j = 1, \ldots, J$. For this case the penalty term in (2.6.2) can also be written as the mixed norm

$$||\mathbf{w}||_{2,1} = \sum_{j=1}^{J} ||w_j||_2.$$

Another choice, used in the implementation of [21], is $K_j = p_j I_{p_j}$.

The group lasso problem can be solved along the lines mentioned above [21]. However, we will see in the next chapter how to solve it along the same lines of convex optimization that we will discuss for Lasso and Elasic Net.

Chapter 3

Proximal optimization

3.1 Convex optimization theory

First we are going to introduce some basic concepts in convex analysis, such as convex sets and functions. We will limit ourselves to define those concepts in the Euclidean space, although with some more careful work they can be extended to Hilbert spaces. $\mathbb{R}^n$ and $\mathbb{E}$ are used interchangeably throughout this chapter to denote the Euclidean space.

Definition 3.1.1. *An Euclidean space $\mathbb{E}$ is a finite-dimensional real vector space with an inner product and a norm*

Definition 3.1.2 (Convex set)**.** *A set $C \subseteq \mathbb{E}$ is a convex set if for all $t \in (0,1)$ the following holds*

$$tx + (1-t)y \in C, \quad \forall x, y \in C$$

Intuitively, the previous definition means that all the points of the segment joining x and y must be inside the set C, and this has to be true for any two points in C. For instance an hyperplane $H = \{x \in \mathbb{R}^n : w^t x - \beta = 0\}$ or a ball $B = \{x \in \mathbb{R}^n : |x - x_0| \leq \beta\}$ are examples of convex sets. However, the sphere $S = \{x \in \mathbb{R}^n : |x - x_0| = \beta\}$ provides an example of a set that is not convex ($\beta > 0$). Figure 3.1.1 (left) shows another example of an arbitrary convex set. On the other hand, figure 3.1.1 (right) is an example of a non-convex set. It is easy to see that any intersection of two convex sets is also convex. Given i convex sets S_i, the finite sum is a new set S formed by taking all the terms $\sum s_i$, where $s_i \in S_i$. Finite sums of convex sets are also convex. We now introduce the concept of effective domain and convex function.

Definition 3.1.3 (Effective domain)**.** *The effective domain of f is the set*

$$\mathrm{dom}(f) = \{x \in \mathbb{E} : f(x) < +\infty\}.$$

If $\mathrm{dom}(f)$ is not empty, the function f is called proper.

Definition 3.1.4 ((Strictly) convex function)**.** *An function $f : \mathbb{E} \to \mathbb{R} \cup \{+\infty\}$ is a convex function if $\mathrm{dom}(f)$ is a convex set and, for every $x, y \in \mathbb{E}$ and any $t \in [0,1]$,*

$$f(tx + (1-t)y) \leq tf(x) + (1-t)f(y).$$

The same function is said to be strictly convex if for every $x, y \in \mathbb{E}$, $x \neq y$ and any $t \in (0,1)$

$$f(tx + (1-t)y) < tf(x) + (1-t)f(y).$$

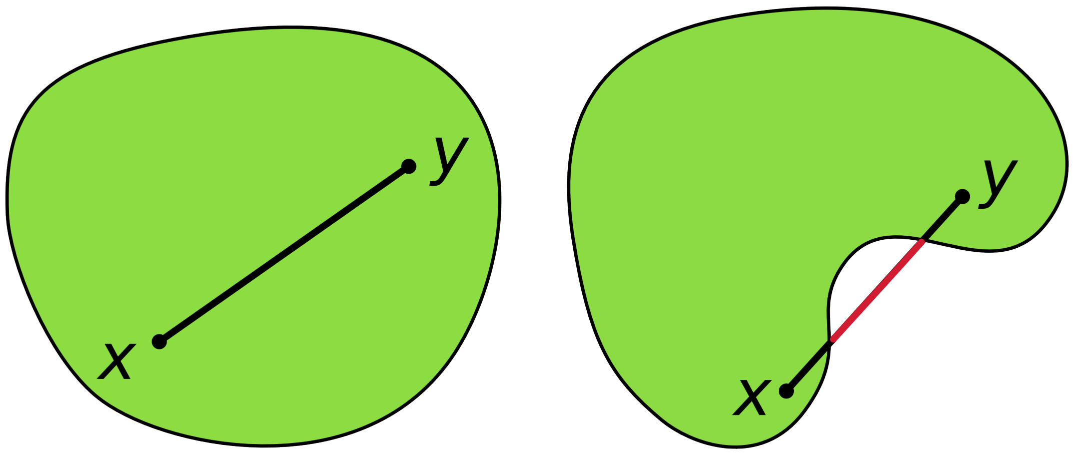


Figure 3.1.1: Example of convex set (left) and non-covex set (right) [Wikipedia, "Convex set"]

The previous definition does not take into account functions that take the value $-\infty$, although it can be expanded to do so [22]. Some methods to create new convex functions from known convex functions $f_1, \dots, f_m : \mathbb{R}^n \to (-\infty, +\infty]$ are:

1. Pointwise sum: for any $\alpha_1, \dots, \alpha_m \in \mathbb{R}_+$, $f(x) = \sum_{i=1}^m \alpha_i f_i(x)$
2. Pointwise maximum: $f(x) = \max_{1 \le i \le m} f_i(x)$

In the following sections we will want to minimize some functions that are not differentiable (or at least have some component that it is not). The subdifferential is a fundamental tool in the analysis of nondifferentiable convex functions.

Definition 3.1.5 (Subdifferential)**.** *The subdifferential of a proper convex function is the set-valued map $\partial f : \mathbb{E} \to 2^{\mathbb{E}}$, defined as*

$$\partial f(x) = \{\xi \in \mathbb{E} : f(x) + \xi^t (y - x) \le f(y), \quad \forall y \in \mathbb{E}\}$$

where $2^{\mathbb{E}}$ is the power set, that is, the set of all subsets of $\mathbb{E}$.

Let $x \in \mathbb{E}$. Then f is subdifferentiable at x if $\partial f(x) \neq \emptyset$. The elements of $\partial f(x)$ are the subgradients of f at x. Observe that this definition is only non-trivial if the function is proper, that is, $x \in \operatorname{dom} f$. Otherwise $f(x) = +\infty$ and $\partial f(x) = \emptyset$. Some properties of the subdifferential are:

(i) For any $x \in \mathbb{E}$, $\partial f(x)$ is either empty or a closed convex set (see [23] proposition 16.3).

(ii) For any $x \in \operatorname{int}(\operatorname{dom}(f))$, $\partial f(x)$ is not empty and bounded (see [22] lemma 2.16).

(iii) $\partial f(x) = \{\nabla f(x)\}$ if and only if f is differentiable at $x \in \mathbb{E}$ (see [22] proposition 2.6).

(iv) A point $x \in \mathbb{E}$ is a (global) minimizer of f if and only if

$$0 \in \partial f(x).$$

This is known as the Fermat's rule in convex optimization, and it will be proven later in this chapter.

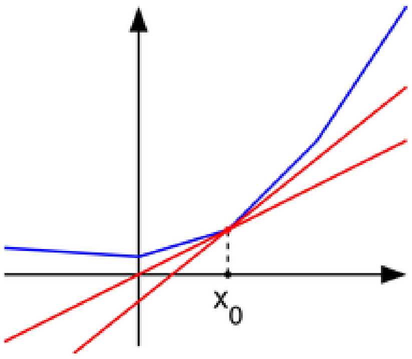


Figure 3.1.2: A convex function (blue) and "subtangent" lines at x_0 (red) [Wikipedia "Subdifferential"]

Theorem 3.1.1 (Moreau-Rockafellar). *Let f and g be proper convex functions. Then, for every $x_0 \in \mathbb{R}^n$*

$$\partial f(x_0) + \partial g(x_0) \subset \partial(f+g)(x_0).$$

Moreover, suppose that $\operatorname{int}(\operatorname{dom} f \cap \operatorname{dom} g) \neq \emptyset$*. Then for every $x_0 \in \mathbb{R}^n$ we also have*

$$\partial(f+g)(x_0) \subset \partial f(x_0) + \partial g(x_0).$$

Proof. For the first part, let $\xi_1 \in \partial f(x_0)$ and $\xi_2 \in \partial g(x_0)$. Then, for all $x \in \mathbb{R}^n$

$$f(x) \geq f(x_0) + \xi_1^t(x - x_0)$$
$$g(x) \geq g(x_0) + \xi_2^t(x - x_0)$$

Adding the previous inequalities gives

$$f(x) + g(x) \geq f(x_0) + g(x_0) + (\xi_1 + \xi_2)^t(x - x_0)$$

and hence $(\xi_1 + \xi_2) \in \partial(f+g)(x_0)$. For the second part, check [22]. □

3.1.1 Minimizing convex functions

We are going to start this section by reviewing briefly some theory on monotone operators, since they play a central role in nondifferentiable convex optimization. The most prominent example of monotone operator is the subdifferential operator, defined in the previous section. The main result will be the Fermat's rule for convex optimization, which characterizes the solutions of convex optimization problems.

Definition 3.1.6 (Monotone operator). *A set-valued operator $T : \mathbb{E} \to 2^{\mathbb{E}}$ is called monotone if*

$$(\xi_1 - \xi_2)^t(x_1 - x_2) \geq 0, \quad \forall x_1, x_2 \in \mathbb{E}, \quad \xi_1 \in T(x_1), \quad \xi_2 \in T(x_2)$$

Proposition 3.1.1 (Monotonicity of subdifferencial). *The subdifferencial is a monotone operator, that is, for all $x_1, x_2 \in \mathbb{E}$, $\xi_1 \in \partial f(x_1)$, $\xi_2 \in \partial f(x_2)$ the following holds*

$$(\xi_1 - \xi_2)^t(x_1 - x_2) \geq 0.$$

Proof. From the definition of subdifferential, for all $x_1, x_2 \in \mathbb{E}$, there are ξ_1 and ξ_2 also in $\mathbb{E}$ such as

$$f(x_1) - f(x_2) \leq \xi_1^t(x_1 - x_2)$$
$$f(x_2) - f(x_1) \leq \xi_2^t(x_2 - x_1)$$

Adding together the previous inequalities we get

$$\begin{aligned}\xi_1^t(x_1-x_2)+\xi_2^t(x_2-x_1)\geq 0 &\Leftrightarrow \xi_1^t(x_1-x_2)+(-\xi_2)^t(x_1-x_2)\rangle \geq 0\\ &\Leftrightarrow (\xi_1-\xi_2)^t(x_1-x_2)\rangle \geq 0\end{aligned}$$

□

The following proposition reveals some interesting properties of monotone operators.

Proposition 3.1.2. *Monotone operators have the following properties:*

(i) Non-negative scaling: αT *is monotone for any alpha* ≥ 0*.*

(ii) Inverse: T^{-1} *is monotone.*

(iii) Resolvent: The resolvent operator

$$R_T=(I+T)^{-1}$$

is singled-valued and firmly non-expansive:

$$\langle R_T(x_1)-R_T(x_2),x_1-x_2\rangle \geq ||R_T(x_1)-R_T(x_2)||^2, \quad \forall x_1,x_2\in\mathbb{E}.$$

Proof. (i) follows from the definition. First let's prove (ii). We consider the graph of T, $\text{graph}(T)=\{(x,\xi):\xi\in T(x)\}$. The graph of T^{-1} is $\text{graph}(T^{-1})=\{(\xi,x):(x,\xi)\in \text{graph}(T)\}$, that is $(\xi,x)\in\text{graph}(T^{-1})\Leftrightarrow \xi\in T(x)$. Since T is monotone, $(\xi_1-\xi_2)^t(x_1-x_2)\geq 0$, $\forall x_1,x_2\in\mathbb{E}$, $\xi_1\in T(x_1)$, $\xi_2\in T(x_2)$ and the monotonicity of T^{-1} is obvious from the definition.

Next we prove (iii) showing first that R_T is single valued. Assume there is one point z with two values $(z,x_1),(z,x_2)\in\text{graph}(R_T)$. Then we have $z\in x_1+T(x_1)$, $z\in x_2+T(x_2)$ and there are $\xi_1\in T(x_1)$, $\xi_2\in T(x_2)$ such that $z=x_1+\xi_1=x_2+\xi_2$, i.e., $\xi_1-\xi_2=-(x_1-x_2)$. Since T is monotone,

$$0\leq(\xi_1-\xi_2)^t(x_1-x_2)=-\|x_1-x_2\|^2$$

and $x_1=x_2$.

To prove the firmly non-expansiveness, let $x_i=R_T(z_i)$, $i=1,2$. Then $z_i\in x_i+T(x_i)$ and $z_i=x_i+\xi_i$ with $\xi_i\in\partial T(x_i)$. Now

$$\begin{aligned}(x_1-x_2)^t(z_1-z_2)&=(x_1-x_2)^t(x_1-x_2)+(x_1-x_2)^t(\xi_1-\xi_2)\\ &=\|x_1-x_2\|^2+(x_1-x_2)^t(\xi_1-\xi_2),\end{aligned}$$

and since T is monotone, $(x_1-x_2)^t(\xi_1-\xi_2)\geq 0$ and firmly non-expansiveness of R_T follows. □

Proposition 3.1.3 (Zeros of monotone operator)**.** *The zeros of a monotone operator T coincide with the fixed points of the resolvents $R_{\alpha T}$, that is,*

$$0\in T(x) \quad\Leftrightarrow\quad x=R_{\alpha T}(x).$$

Proof.

$$\begin{aligned}0\in T(x)\ \Leftrightarrow\ 0\in\alpha T(x)\ \Leftrightarrow\ x\in x+\alpha T(x)\ \Leftrightarrow\ x\in(I+\alpha T)(x)\\ \Leftrightarrow\ x=(I+\alpha T)^{-1}(x)\ \Leftrightarrow\ x=R_{\alpha T}(x)\end{aligned}$$

□

Some examples of monotone operators taken from [24] are:

- Any linear positive semidefinite operator:

$$\langle T(x), x\rangle \geq 0, \quad \forall x \in \mathbb{E}.$$

- The projection operator over a non-empty closed convex set C, defined as

$$P_C(x) = \operatorname*{argmin}_{y \in C} ||x - y||.$$

We show next two alternative definitions of the proximal operator. Proposition 3.1.4 demonstrates the equivalence between both definitions.

Definition 3.1.7 (Proximal mapping)**.** *The resolvent of the subdifferential is called the proximal mapping, i.e.*

$$\operatorname{prox}_f = (I + \partial f)^{-1}.$$

Notice that $x = \operatorname{prox}_f(z)$ iff $z - x \in \partial f(x)$, i.e., iff $0 \in x - z + \partial f(x)$. Moreover, given z, we have seen that x is unique.

Definition 3.1.8 (Proximal mapping (alternative))**.** *Let f be a lower semicontinous, proper convex function. For every $z \in \mathbb{R}^N$, the proximal mapping of f is defined as*

$$\operatorname{prox}_f(z) = \operatorname*{argmin}_{y \in \mathbb{R}} \left(f(y) + \frac{1}{2}||z - y||^2 \right).$$

Proposition 3.1.4. *Definitions 3.1.7 and 3.1.8 are equivalent.*

Proof. First we are going to show that the first definition can be obtained from the second one. Let $\phi(y) = f(y) + \frac{1}{2}||z - y||^2$ and $x = \operatorname{argmin} \phi(y)$. The subdifferential of ϕ is $\partial\phi(y) = y - z + \partial f(y)$ and since x is a minimizer $0 \in x - z + \partial f(x)$. As we have seen before, this is equivalent to $x = (I + \gamma\partial f)^{-1}(z)$, which is the first definition.

Now we are going to show that the second definition can be obtained from the first one. Assume $x = (I + \gamma\partial f)^{-1}(z)$; then $z - x \in \partial f(x)$, i.e, $z - x$ is a subgradient of f at x. Using the definition of subdifferential we have that for all y,

$$f(y) \geq f(x) + (z - x)^t(y - x).$$

Adding $\frac{1}{2}\|y - z\|^2$ to both sides gives

$$f(y) + \frac{1}{2}\|y - z\|^2 \geq f(x) + (z - x)^t(y - x) + \frac{1}{2}\|y - z\|^2. \tag{3.1.1}$$

Next let $\psi(y) = (z - x)^t(y - x) + \frac{1}{2}\|y - z\|^2$. Then $\nabla\psi(y) = z - x + y - z = y - x$ and ψ attains a minimum at x, for which $\psi(x) = \frac{1}{2}\|x - z\|^2$. Substituting in (3.1.1) we get $\forall y$

$$\begin{aligned} f(y) + \frac{1}{2}\|y - z\|^2 &\geq f(x) + \psi(y) \\ &\geq f(x) + \frac{1}{2}\|x - z\|^2, \end{aligned}$$

and, therefore, $x = \operatorname{argmin}_y\{f(y) + \frac{1}{2}||z - y||^2\}$, which is the second definition. □

Some examples of proximal operators are:

- For $f = 0$, $\text{prox}_f(x) = x$.
- For $f = I_C$, $\text{prox}_f(x) = P_C(x)$. Here I_C denotes the indicator function (definition 3.1.10). Given the second definition of proximal mapping and the definition of the indicator function, it is easy to see why this is true.
- Let $f : \mathbb{R}^n \to \mathbb{R}, \quad f(x) = ||x||_1 = \sum_{i=1}^n |x_i|$. Then,

$$[\text{prox}_f(x)]_i = \text{sign}(x_i) \max(0, |x_i| - 1).$$

The optimization problem we are interested in solving is the following

Definition 3.1.9 (Constrained optimization problem)**.**

$$\min_{x \in \mathbb{E}} f(x), \quad \textit{subject to } x \in S$$

where $S \subseteq \mathbb{E}$

This constrained optimization problem can be rewritten as a unconstrained optimization problem, since (3.1.9) is equivalent to

$$\min_{x \in \mathbb{E}} (f(x) + I_S(x))$$

where the indicator function I_S is defined next.

Definition 3.1.10 (Indicator function)**.** *The indicator function of the set* $S \subseteq \mathbb{E}$ *is*

$$I_S(x) = \begin{cases} 0, & x \in S \\ +\infty, & \textit{else} \end{cases}$$

Convex optimization is a subfield of optimization where both the function f and the set S are convex. Global minimizers of convex functions are characterized by Fermat's rule, a simple but powerful principle.

Theorem 3.1.2 (Fermat's rule)**.** *Let* f *be a proper convex function. Then,*

$$\text{argmin}\, f = \text{zer}\, \partial f = \{x \in \mathbb{E} \mid 0 \in \partial f(x)\}.$$

Moreover, $x = \text{prox}_f(x)$.

Proof. Let $x \in \mathbb{E}$. Then $x \in \text{argmin}\, f$ if it does not exist any other point $y \in \mathbb{E}$ such as $f(y) < f(x)$. This is equivalent to

$$(y - x)^t 0 + f(x) \leq f(y)$$

$\forall y \in \mathbb{E}$, since $(y - x)^t 0 = 0$. Looking at the definition of subdifferential, that is also equivalent to $0 \in \partial f$.

For the second fact, notice that $\partial f(x)$ is a monotone operator and its resolvent is the proximal mapping (definition 3.1.7). Therefore using proposition 3.1.3, $0 \in \partial f(x) \iff x \in x + \partial f(x) = (I + \partial f)(x) \iff x = \text{prox}_f(x)$ and the proof is complete. Note that we can add a constant term $\gamma > 0$ multiplying ∂f and this result still holds. □

The conclusion of this section is that we can obtain solutions for $0 \in \partial f(x)$ as the fixed points of the proximal operator. This suggests the proximal point method in section 3.3.

3.2 Minimizing the sum of two functions

In many problems we further decompose problem (3.1.9) into two functions: the loss function and the regularizer, as we have seen in section 1.4. The minimization problem then becomes

$$\min_{x\in\mathbb{E}} \; f(x)+g(x). \tag{3.2.1}$$

Using the Fermat's rule and the fact that $\partial(f+g)\in\partial f+\partial g$ (theorem 3.1.1), the optimality condition for this problem is

$$0\in\partial f(x)+\partial g(x).$$

The previous result does not assume that either f or g are differentiable functions. Most of the times in practice we find that one of them it is indeed differentiable, while the other is not. This can be further exploited to simplify the optimality condition. If the function f is differentiable and Lipschitz continuous, the new optimality condition is

$$0\in\nabla f(x)+\partial g(x),$$

where we have used the third property of the subdifferential. The next proposition summarizes the previous results.

Proposition 3.2.1. *Let f, g be proper convex functions and f differentiable. Then, the following are equivalent:*

(i) *x is a solution of the problem 3.2.1;*

(ii) $0\in\nabla f(x)+\partial g(x)$*;*

(iii) $x=\operatorname{prox}_{\gamma g}(x-\gamma\nabla f(x))$*.*

Proof. If x is a solution of the problem 3.2.1 then, using Fermat's theorem (theorem 3.1.2),

$$\begin{aligned}
0\in\partial(f+g)(x) &\Leftrightarrow 0\in\partial f(x)+\partial g(x) && \text{(theorem 3.1.1)}\\
&\Leftrightarrow 0\in\{\nabla f(x)\}+\partial g(x) && \text{(property (iii) of the subdifferential)}\\
&\Leftrightarrow -\nabla f(x)\in\partial g(x)\\
&\Leftrightarrow x-\gamma\nabla f(x))\in x+\gamma\partial g(x)\\
&\Leftrightarrow x=\operatorname{prox}_{\gamma g}(x-\gamma\nabla f(x)).
\end{aligned}$$

□

3.3 Algorithms

3.3.1 Proximal gradient method (forward-backward splitting)

Equation (iii) of proposition 3.2.1 gives the idea of iterating

$$x_{k+1}=\operatorname{prox}_{\gamma g}(x_k-\gamma\nabla f(x_k)), \tag{3.3.1}$$

starting from an initial point x_0. It is important to note that, on one hand, when $g=0$, (3.3.1) reduces to the *gradient descent method* (see the previous examples of proximal mappings)

$$x_{k+1}=x_k-\gamma\nabla f(x_k) \tag{3.3.2}$$

for minimizing a Lipschitz-differentiable function [25]. On the other hand, when $f = 0$, (3.3.1) reduces to the *proximal point algorithm*

$$x_{k+1} = \operatorname{prox}_{\gamma g}(x_k), \tag{3.3.3}$$

for minimizing a non-differentiable function.

Theorem 3.3.1 (Convergence of the proximal gradient method)**.** *If f has Lipschitz continuous gradient with constant L, $(f+g)$ admits minimizers and*

$$\gamma \in \left(0, \frac{2}{L}\right),$$

then the proximal-gradient iterations x_k converge to a minimizer.

The proof of this theorem can be found in [23] (corollary 27.9).

3.3.2 ISTA

A classical approach to solve the regression problem is the least squares approach, in which the estimator is chosen to minimize the error

$$\hat{\mathbf{w}}_{OLS} = \operatorname*{argmin}_{\mathbf{w}}\{\|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2\}. \tag{3.3.4}$$

As we mentioned in chapter 2, the previous problem is usually replaced by a well-conditioned problem whose solution approximates the required solution. Recall from chapter 1 that one popular regularization technique is Tikhonov regularization, in which a quadratic penalty is added to the objective function:

$$\hat{\mathbf{w}}_{RLS} = \operatorname*{argmin}_{\mathbf{w}}\{\|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2 + \lambda\|\mathbf{w}\|_2^2\} \tag{3.3.5}$$

Another regularization method is l_1 regularization, where you try to find the solutions of

$$\min_{\mathbf{w}}\{||\mathbf{X}\mathbf{w} - \mathbf{y}||^2 + \lambda\|\mathbf{w}\|_1\}. \tag{3.3.6}$$

This is the Lasso problem (see chapter 2) and it fits the general framework of convex optimization problems (3.1.9), where $f(\mathbf{w}) = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2$ and $g(\mathbf{w}) = \lambda\|\mathbf{w}\|_1$. In addition, since f is a differentiable function and g it is not, we can apply the proximal gradient method of the previous section. Specifically, the general step of ISTA is

$$\mathbf{w}_{k+1} = \mathcal{T}_{\lambda t}(\mathbf{w}_k - 2t\mathbf{w}^t(\mathbf{X}\mathbf{w}_k - \mathbf{y})), \tag{3.3.7}$$

where $\mathcal{T}_\alpha$ is the proximal mapping of the l_1-norm, that is, the soft-thresholding operator defined by

$$[\mathcal{T}_\alpha(\mathbf{x})]_i = (|x_i| - \alpha)_+ \operatorname{sign}(x_i). \tag{3.3.8}$$

Now let's see an alternative, more general, derivation of the algorithm from [26]. Consider the general formulation:

$$\min\{F(\mathbf{w}) \equiv f(\mathbf{w}) + g(\mathbf{w}) : \mathbf{w} \in \mathbb{R}^n\}. \tag{3.3.9}$$

The following assumptions are made:

- $g : \mathbb{R}^n \to \mathbb{R}$ is a continous convex function which is possibly *nonsmooth.*

- $f : \mathbb{R}^n \to \mathbb{R}$ is a smooth convex function of the type $C^{1,1}$, i.e, continuosly differentiable with Lipschitz continous gradient $L(f)$:

$$||\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})|| \leq L(f)||\mathbf{x} - \mathbf{y}||,$$

  where $|| \cdot ||$ denotes the standard Euclidean norm and $L(f) > 0$ is the Lipschitz constant of ∇f.

- The problem is solvable, i.e. $\operatorname{argmin} F \neq \emptyset$.

Now we are going to consider the following quadratic approximation of $F(\mathbf{w}) := f(\mathbf{w}) + g(\mathbf{w})$ at a given point $\mathbf{y}$:

$$Q_L(\mathbf{w}, \mathbf{y}) := f(\mathbf{y}) + \langle \mathbf{w} - \mathbf{y}, \nabla f(\mathbf{y}) \rangle + \frac{L}{2}||\mathbf{w} - \mathbf{y}||^2 + g(\mathbf{w}) \tag{3.3.10}$$

which admits an unique minimizer

$$p_L(\mathbf{y}) := \operatorname{argmin}\{Q_L(\mathbf{w}, \mathbf{y}) : \mathbf{w} \in \mathbb{R}^n\} \tag{3.3.11}$$

Simple algebra shows that (ignoring constant terms in $\mathbf{y}$)

$$p_L(\mathbf{y}) = \underset{\mathbf{w}}{\operatorname{argmin}} \left\{ g(\mathbf{w}) + \frac{L}{2} \left\| \mathbf{w} - \left(\mathbf{y} - \frac{1}{L} \nabla f(\mathbf{y}) \right) \right\|^2 \right\} \tag{3.3.12}$$

Note that the operator p_L is just the proximal operator of g (3.1.8) evaluated at the point $\mathbf{y} - \frac{1}{L}\nabla f(\mathbf{y})$,

$$\begin{aligned} \operatorname{prox}_{\frac{1}{L}g} \left(\mathbf{y} - \frac{1}{L}\nabla f(\mathbf{y}) \right) &= \underset{\mathbf{w}}{\operatorname{argmin}} \left\{ \frac{1}{L} g(\mathbf{w}) + \frac{1}{2} \left\| \left(\mathbf{y} - \frac{1}{L}\nabla f(\mathbf{y}) \right) - \mathbf{w} \right\|^2 \right\} \\ &= \underset{\mathbf{w}}{\operatorname{argmin}} \left\{ g(\mathbf{w}) + \frac{L}{2} \left\| \mathbf{w} - \left(\mathbf{y} - \frac{1}{L}\nabla f(\mathbf{y}) \right) \right\|^2 \right\} = p_L(\mathbf{y}) \end{aligned}$$

that is, after taking a small step in the direction of the negative gradient. Finally, the basic step of the algorithm is

$$\mathbf{w}_k = p_L(\mathbf{w}_{k-1}) \tag{3.3.13}$$

This is a more general version of ISTA [26] since $g(\mathbf{w})$ could be any nonsmooth regularizer and $f(\mathbf{w})$ any smooth convex function. However, in [26] they still refer to this more general method as ISTA, which may be confusing.

Algorithm 2: ISTA with constant stepsize

```
Input: $L := L(f)$
Step 0. Take $w_0 \in \mathbb{R}^n$.
Step k. ($k \geq 1$) Compute
            $\mathbf{w}_k = p_L(\mathbf{w}_{k-1})$;
```

If $f(\mathbf{w}) = ||\mathbf{X}\mathbf{w} - \mathbf{y}||^2$ and $g(\mathbf{w}) = \lambda||\mathbf{w}||_1$ ($\lambda > 0$) then equation (3.3.13) reduces to (3.3.7) with $t = \frac{1}{L(f)}$. A possible drawback of this basic scheme is that the Lipschitz constant $L(f)$ is not always known or computable. For instance, the Lipschitz constant in the l_1-regularization problem depends on the maximum eigenvalue of $\mathbf{X}^t\mathbf{X}$. For large-scale problems, this quantity is not always easily computable. The trivial algorithm to compute eigenvalues needs $O(n^3)$ operations, where n is the size of the matrix, although there are faster approaches if we only need a few of them. We therefore also analyze ISTA with a backtracking stepsize rule.

```
Algorithm 3: ISTA with backtracking
  Step 0. Take $L_0 > 0$, some $\eta > 1$, and $\mathbf{w}_0 \in \mathbb{R}^n$.
  Step k. ($k \geq 1$) Find the smallest nonnegative integers $i_k$ such that with
  $\bar{L} = \eta^{i_k} L_{k-1}$
      $F(p_{\bar{L}}(\mathbf{w}_{k-1})) \leq Q_{\bar{L}}(p_{\bar{L}}(\mathbf{w}_{k-1}), \mathbf{w}_{k-1}).$
  Set $L_k = \eta^{i_k} L_{k-1}$ and compute
      $\mathbf{w}_k = p_{L_k}(\mathbf{w}_{k-1})$
```

Theorem 3.3.2. *Let $\{\mathbf{w}_k\}$ be the sequence generated by algorithms 2 or 3. Then, for any $k > 1$*

$$F(\mathbf{w}_k) - F(\mathbf{w}^*) \leq \frac{\alpha L(f) \|\mathbf{w}_0 - \mathbf{w}^*\|^2}{2k} \; \forall \mathbf{w}^* \in \mathbf{W}_* \tag{3.3.14}$$

where $\alpha = 1$ for the constant stepsize setting and $\alpha = \eta$ for the backtracking stepsize setting.

The previous result can be interpreted as follows. The number of iterations required to obtain a solution $\tilde{\mathbf{w}}$ such that $F(\tilde{\mathbf{w}}) - F(\mathbf{w}^*) \leq \epsilon$, is at most $\lceil C/\epsilon \rceil$, where $C = \frac{\alpha L(f)\|\mathbf{w}_0 - \mathbf{w}^*\|^2}{2}$. The proof of theorem 3.3.2 can be found in [26]. In the next section we will discuss FISTA, a different method as simple as ISTA but with a faster convergence rate, $O(1/k^2)$.

3.3.3 FISTA

In the previous section we showed that ISTA has a worst-case complexity of $O(1/k)$. In this section we will introduce FISTA (fast iterative shrinkage-thresholding algorithm), an improved version of ISTA with a worst-case complexity of $O(1/k^2)$. We recall that ISTA is just a specific version of the more general proximal gradient method (3.3.1), which reduces to the gradient method when $g(\mathbf{x}) = 0$. Nesterov showed in [27] that exists a gradient method with complexity $O(1/k^2)$ which is an "optimal" first order method for smooth problems. Beck and Teboulle [26] extended the previous method to composite functions, where one of them is (possibly) non-smooth. The pseudocode for FISTA is showed in algorithm 4.

FISTA can be also modified in the same way as ISTA in order to get rid of the Lipschitz constant $L(f)$. The pseudocode for FISTA with a backtracking stepsize rule can be seen in algorithm 5.

Note that the only important difference between algorithms 2-3 and 4-5 is that the operator p_L is not applied to the previous point $\mathbf{w}_{k-1}$ but to a smartly chosen linear combination of the previous two, $\mathbf{w}_{k-1}$ and $\mathbf{w}_{k-2}$. Since the computational burden is in the p_L operator and both algorithms require the same number of p_L evaluations, the cost per iteration is almost identical. Clearly, the extra computation performed by FISTA is marginal in comparison to the p_L evaluation. In addition, they are almost equally difficult to implement, but FISTA has an improved convergence rate of $O(1/k^2)$.

Algorithm 4: FISTA with constant stepsize

```
**Input**: $L := L(f)$
**Step 0.** *Take* $\mathbf{y}_1 = \mathbf{w}_0 \in \mathbb{R}^n$, $t_1 = 1$.
**Step k.** ($k \geq 1$) *Compute*
```

$$\begin{aligned}\mathbf{w}_k &= p_{L_k}(\mathbf{y}_k)\\ t_{k+1} &= \frac{1+\sqrt{1+4t_k^2}}{2}\\ \mathbf{y}_{k+1} &= \mathbf{w}_k + \left(\frac{t_k - 1}{t_{k+1}}\right)(\mathbf{w}_k - \mathbf{w}_{k-1})\end{aligned}$$

Algorithm 5: FISTA with backtracking

```
**Input**: $L := L(f)$
**Step 0.** *Take* $L_0 > 0$, *some* $\eta > 1$, *and* $\mathbf{w}_0 \in \mathbb{R}^n$. *Set* $\mathbf{y}_1 = \mathbf{x}_0$, $t_1 = 1$.
**Step k.** ($k \geq 1$) *Find the smallest nonnegative integers* $i_k$ *such that with*
$\bar{L} = \eta^{i_k} L_{k-1}$
```

$$F(p_{\bar{L}}(\mathbf{y}_k)) \leq Q_{\bar{L}}(p_{\bar{L}}(\mathbf{y}_k), \mathbf{y}_k).$$

Set $L_k = \eta^{i_k} L_{k-1}$ *and compute*

$$\begin{aligned}\mathbf{w}_k &= p_{L_k}(\mathbf{y}_k)\\ t_{k+1} &= \frac{1+\sqrt{1+4t_k^2}}{2}\\ \mathbf{y}_{k+1} &= \mathbf{w}_k + \left(\frac{t_k - 1}{t_{k+1}}\right)(\mathbf{w}_k - \mathbf{w}_{k-1})\end{aligned}$$

Theorem 3.3.3. *Let* $\{\mathbf{w}_k\}$, $\{\mathbf{y}_k\}$ *be generated by FISTA. Then, for any* $k > 1$

$$F(\mathbf{w}_k) - F(\mathbf{w}^*) \leq \frac{2\alpha L(f)\|\mathbf{w}_0 - \mathbf{w}^*\|^2}{(k+1)^2} \; \forall \mathbf{w}^* \in \mathbf{W}_*, \tag{3.3.15}$$

where $\alpha = 1$ *for the constant stepsize setting and* $\alpha = \eta$ *for the backtracking stepsize setting.*

Theorem 3.3.3 shows that the number of iterations required by FISTA to obtain a solution $\tilde{\mathbf{w}}$ such that $F(\tilde{\mathbf{w}}) - F(\tilde{\mathbf{w}^*}) \leq \epsilon$, is at most, $\lceil C/\sqrt{\epsilon} - 1 \rceil$, where $C = \sqrt{2\alpha L(f)\|\mathbf{w}_0 - \mathbf{w}^*\|^2}$, which clearly improves ISTA. The proof of the theorem can be found in [26]. It is worth mentioning that the value of the function in FISTA does not decrease in every iteration. In [28] they propose a modification of the algorithm, known as monotone FISTA or MFISTA that guarantees descent at each iteration, that is

$$f(\mathbf{w}_k) < f(\mathbf{w}_{k-1}).$$

The theoretical complexity of this new algorithm is the same as FISTA, and it can be found also in [28].

Chapter 4

Experiments

4.1 Wind-power prediction

In this section we will apply the previous algorithms to the problem of predicting the energy production of a wind farm. Sotavento[1] is an experimental wind farm whose objectives are, apart from its commercial exploitation, the following:

- Being the "showcase" of the different wind technologies present in Galicia;
- Being a framework for the realization of I+D activities;
- Formation and debate center;
- Center of renewable energies spreading.

Sotavento's real production data is publicly available on the web, as well as other useful technical information. This makes working with Sotavento very convenient, since it is very difficult to find real production data for wind farms. Some of this information taken from Sotavento's website is depicted below:

- Number of wind turbines: 24
- Technologies present: 5
- Different machines: 9
- Nominal power of the farm: 17.56 MW
- Predicted annual production: 38.500 MWh

The features will be numerical weather predictions (NWP) for a rectangular grid around Sotavento, which is located at coordinates 43.34190°N, 7.86169°W. Five meteorological variables that have been proven effective in the past [29] for wind energy prediction are used:

- V, norm of the wind speed
- V_x, x component of wind speed
- V_y, y component of wind speed

[1]http://www.sotaventogalicia.com

	Geopotential height (m)					
Pressure level (hPa)	Min	1st Quartile	Median	Mean	3rd Quartile	Max
1000	-53.67	20380	20510	20540	20790	21020
975	-44.09	16090	16200	16240	16460	16690
950	-26.67	13550	13700	13720	13940	14230
925	-14.41	11750	11930	11920	12120	12430
900	3.6	10350	10530	10510	10690	10960
875	1.87	9182	9342	9322	9481	9703
850	6743	7230	7362	7340	7457	7631
800	5218	5623	5730	5707	5801	5935
700	2727	3053	3110	3099	3160	3245
500	1683	1986	2031	2020	2067	2153
400	1198	1492	1533	1523	1567	1655
300	964	1252	1293	1284	1328	1417
250	734.9	1018	1059	1050	1095	1188
200	510.6	789.2	830.1	822.7	867.7	967.5
150	291.1	565.8	606.5	600.3	646.0	751.2
100	76.32	347.1	387.8	383.1	429.8	539.4
50	-133.9	133.4	174.1	170.7	217.6	332.5

Table 4.1.1: Geopotential height statistics for the 17 pressure levels provided by AEMET at 43.25°N 7.75°W (1 year of data)

- T, temperature
- G, geopotential height

These variables are normalized to have zero mean and unit variance. AEMET[2] forecasts for those variables are used over a $6 \times 6 \times 17$ grid with horizontal resolution 0.25°. The grid contains a total of 612 points with longitudes in the interval [8.5°W, 7.25°W], latitudes in the interval [42.75°N, 44°N] and the following pressure levels: 1000, 975, 950, 925, 900, 875, 850, 800, 700, 500, 400, 300, 250, 200, 150, 100 and 50 hPa. The dimension of the input space is then $612 \times 5 = 3060$. Target values are wind energy productions normalized to the interval $[0, 1]$ as a percentage of the total nominal power of the farm.

Although meteorological data is available in 17 different pressure levels, where level 1 is the highest and 17 the lowest, it is obvious that not all of them have an effect on the energy production of the wind farm. Table 4.1.1 shows some statistics of the geopotential height for 1 year of data and each one of the different pressure levels. This statistics are computed at the closest point to Sotavento in the grid (43.25°N, 7.75°W). A first selection of pressure levels can been done using the elevation of this particular wind farm, which is between $600 - 700$ m above sea level. According to the table 4.1.1, the first 10 levels are located consistently much higher than 700 m and they are immediately discarded. Note also that the first 6 levels contain some errors in the data, since the minimum should not be that low.

This can be further confirmed by computing correlation plots between wind speed and production of the wind farm for all the pressure levels. Figure 4.1.1 shows that correlation is below 0.5 for the first 10 levels and reaches its maximum value at levels 13, 14, 15, 16 and

[2]Spanish State Meteorological Agency, `http://www.aemet.es`

17. The mean elevation of these four levels is 1050, 822.7, 600.3 and 347.1 m respectively, which makes sense taking into account that the elevation of Sotavento is $600 - 700$ m and wind turbines are usually $80 - 100$ m tall.

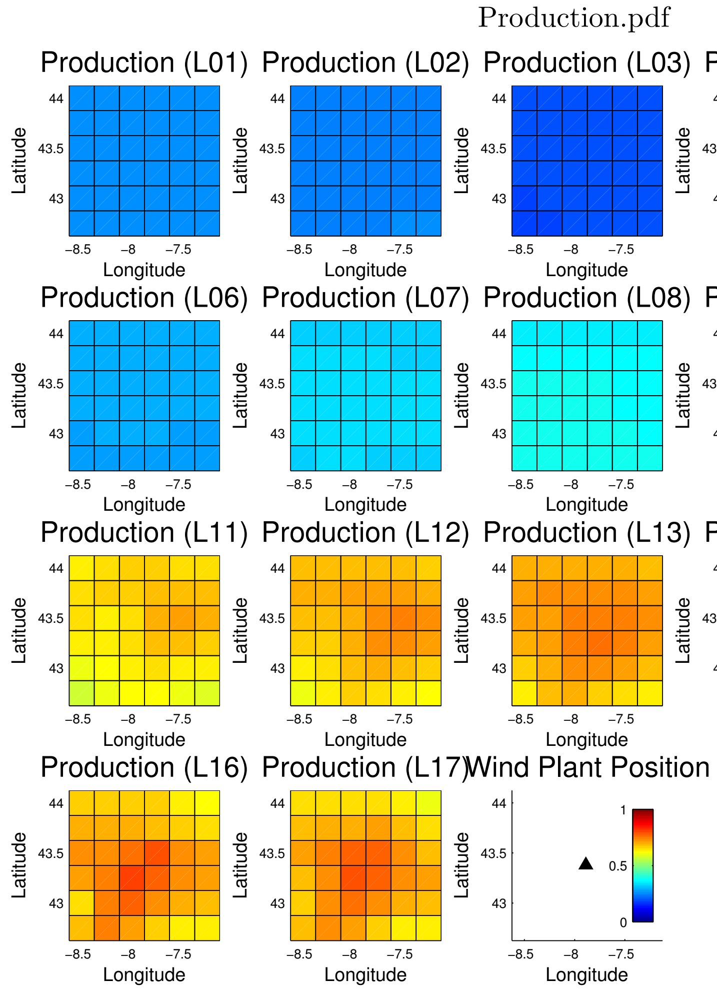


Figure 4.1.1: Correlation between wind speed (norm) and wind energy production for the 17 pressure levels and the whole grid around Sotavento

Even though the rest of the levels $(11 - 17)$ are selected for the experiments, it is not clear that all of them have the same effect (if any) on wind energy production. Sparse methods can help us to select the most useful points in order to obtain the best prediction error. This methods could have also been used in order to select the most important pressure levels out of the 17, that is, without discarding any level to begin with. However, the amount of computer power that we had available was not enough to handle an input space of dimension 3060. After the first manual level selection, the dimension of the input is reduced to $6 \times 6 \times 6 \times 5 = 1080$, which is almost 1/3 of the original dimension and much more manageable.

AEMET also provides meteorological data at surface level. The variables are now wind speed and norm at 10 m above surface, temperature at 2 m above surface and pressure at sea level. The resolution of the grid is the same as before and thus now the dimension is $6 \times 6 \times 5 = 180$. Therefore, another alternative is to train a model using only surface information. We would expect this model to be worse in terms of accuracy than the previous one, since it contains much less information, and therefore we will use it as a baseline.

The data used in our experiments corresponds to period of 1 year and 2 months. Since meteorological forecast are only available every 3 hours, there are 8 patterns per day. The first 12 months are used for training and the last 2 months are used for test. The models are evaluated using the Mean Absolute Error (MAE) and the sparsity level.

$$\text{MAE} = \frac{1}{N}\sum_{n=1}^{N} |\mathbf{x}_n^t \mathbf{w} - y_n| \ ,$$

where $\mathbf{x}_n$ denotes the nth pattern.

The models we are going to test are Ordinary Least Squares (OLS), Ridge Regression (RLS), Lasso (LA) and Elastic Net (ENet). The code used is the C implementation of the FISTA algorithm discussed in the appendix A.

An important issue for most of the algorithms is the estimation of hyper-parameters λ_1 and λ_2 that configure each model. This is done with a search over a grid that represents a discrete version of the parameter space, using a logarithmic scale, from 10^{-3} to 10^2 and steps of $10^{0.10}$. In the algorithm that involves a bidimensional grid (ENet), the step size is increased to $10^{0.20}$. At each point of the parameter grid, 5-fold cross validations is used to evaluate the given model, using the MAE as fitness. Finally, the parameters with the lowest MAE are selected. The comparison of the different models is summarized in table 4.1.2.

Algorithm	λ_1	λ_2	Sparsity (%)	MAE
ENet	0.1	4	60.463	7.489
RLS	0	16	0	7.493
LA	0.1	0	69.907	7.591
OLS	0	0	0	10.062

(a) AEMET pressure level data

Algorithm	λ_1	λ_2	Sparsity (%)	MAE
LA	0.0016	0	10.556	7.624
RLS	0	0.0016	0	7.629
ENet	0.006	0.016	23.333	7.725
OLS	0	0	0	7.759

(b) AEMET surface data

Table 4.1.2: Results for 4 different models: Lasso (LA), Elastic Net (ENet), Ridge Regression (RLS) and Ordinary Least Squares (OLS)

In terms of the MAE, the worst model is OLS, probably due to overfitting, since the number of variables (1080) is large in comparison to the number of training patterns (2640). Ridge regression performs better than the Lasso in terms of the error, but none of its coefficients are 0, so the second probably probably be a more useful model. Elastic Net has the lowest error also achieving great sparsity, almost the same as the Lasso. The main disadvantage is that it is more computationally expensive, since two parameters must be selected.

In conclusion we may say that Elastic Net is the best model, since with only 40% of the variables is capable of obtaining the lowest error, and it only sacrifices a little bit of sparsity in comparison to the Lasso. In addition, using pressure level information is useful when predicting wind power, seeing that surface results are considerably worse. Finally, if more sparsity is needed, the λ_1 parameter can always be tuned in order to get the desired % of active weights.

If we look now at the active weights (weights that are different from 0), we might find some structure in the problem. Figure 4.1.2 shows the percentage of active weights for each variable and for each pressure level.

In the first case the geopotential height (G) is almost completely discarded from the model, and hence it is not useful to predict wind power. On the other hand, the wind power norm and its x component are the most “important” variables, which makes sense according to the correlation plots 4.1.1.

In the second case, there is no clear distinction between the “best” and “worst” levels. However both models tend to select the lowest and highest levels. An explanation of this behavior could be that medium levels are quite correlated and thus Lasso selects some points from any of them somewhat randomly. On the other hand, levels 12 and 17 are usually located very far from the farm, but some days they provide some useful and “new” information, since they are more independent between them. Elastic Net is similar to the Lasso, although it seems to behave a little bit better, since it selects more points from the middle levels, closest to the farm.

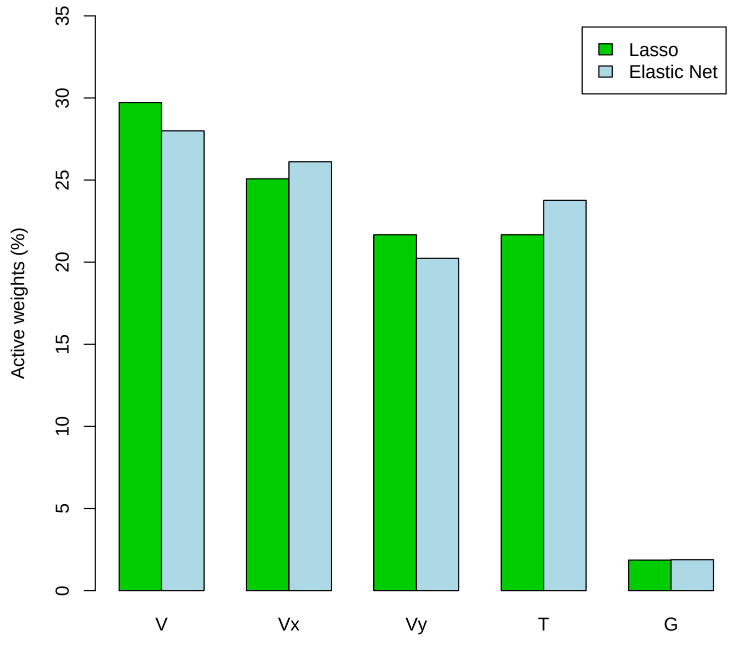
4.2 Variable selection in Rab8 detection

The problem in this section comes from the biomedical domain, and it involves classifying images of cells as having or not a certain protein (Rab8). The presence of the protein Rab8 is important mainly because it is related to the cell migration process, which causes cancer cells to move and extend the tumor to other adjacent tissue, forming colonies. These cells move by extending protrusions in the direction of propagation. Therefore the objective of this work is trying to detect the phenotype of a migratory cell, so they can be automatically classified.

All the variables for this problem were extracted from microscopic cell images. First, the images are preprocessed in order to segment individual cells and then the features are generated, all of this using the Definiens software. Some of the cells contain Rab8 combined with another fluorescent protein (GFP), so they can be distinguished from control cells. Then, cells are labeled as GFP positive, if the fluorescent marker is present, or GFP negative, if it is not. Hopefully GFP positive cells will also contain the Rab8 protein, although it is not always the case. The dataset was provided by the CNIC foundation [3] as a part of a collaboration project [30], and it contains 6497 cells with 162 variables. Example of variable types are:

- Morphometrics: measure cell shape;
- Textures;
- Spots: area proportion, mean distance to border...;
- Actin;
- Intensities;
- Ratios;

[3] http://www.cnic.es/en/index.php



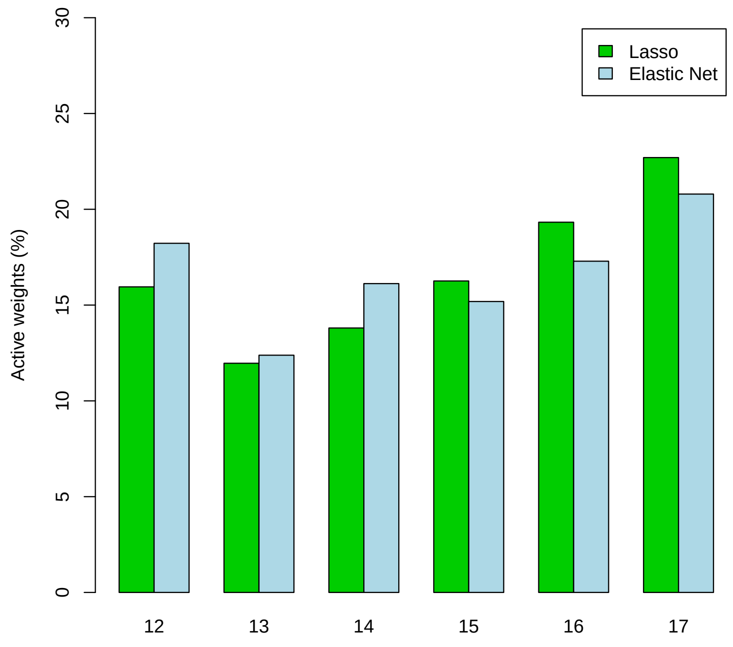


Figure 4.1.2: Percentage of active weights per variable (top) and per level (bottom)

- Binary logs of the ratios.

In this experiment we are going to use logistic regression with an l_1 penalization term in order to select the most relevant variables in a binary classification setting. The labels are GFP positive (1) and GFP negative (0). Considering all the attributes were generated from cell images, it is likely that there is a lot of redundancy in them, making variable selection a very important step before classification. It is also useful for biologists to know the most important variables selected by the model, since they have an intrinsic biological meaning and biologist can interpret the results.

As seen in section 2.2, the l_1 penalty term will shrink coefficients towards 0 and some of them will be strictly 0, acting as an embedded feature selection method. Logistic regression can be optimized using FISTA, since it takes the form $\mathcal{L}(\mathbf{w}) + \lambda\|\mathbf{w}\|_1$, where the loss function is differentiable (equation (1.2.11)) and the penalization term is convex. Besides, the penalization is the l_1 norm and thus its proximal operator is the same as Lasso, the soft thresholding operator (3.3.8). The C code developed in this thesis could be used here, but it is still not able to minimize the logistic regression loss. This is a feature that will be implemented in the future. Therefore, we will use the R package `glmnet`, which does not use FISTA but coordinate descent instead [31].

The package `glmnet` is able to compute the whole regularization path for a logistic regression model, that is, it trains the model for a sequence of λ's instead of just for one λ. This is similar to how the Lasso is solved using LARS (section 2.5). The idea behind the regularization path is the following: the algorithm starts with the smallest value of λ for which all coefficients are 0 and then, as λ is decreased, some coefficients will enter the model. For $\lambda = 0$, all coefficients are different from 0 and the model is the standard logistic regression. If we stop at any point in the middle, we have effectively performed variable selection, since features with 0 coefficients can be discarded from the model.

Computing the whole regularization path is at least equally expensive and sometimes faster than training only one model, since the weights of the previous value of λ are used as a starting point for the next value [31]. This is known as *warm starts*, and it is a remarkably efficient strategy for this kind of problems. Figure 4.2.1 shows the regularization path for our dataset, where each line represents one variable as the value of λ varies.

Once we compute the whole regularization path we have to chose one λ in order to train the final model. This "optimal" λ may be selected in, at least, two different ways. If the sparsity level is known in advance, either because we have some prior information or we want to enforce it, we can stop at the closest λ producing the desired sparsity. Recall that in LARS we know exactly how much should be decreased λ so one coefficient joins the model at every step. On the other hand, in `glmnet`, it may be the case where many variables join the model "at once", when we go from one λ to the next. This can be prevented increasing the number of λ's, but this also increases the computational burden and in practice it is usually not a problem to stop at an approximate sparsity level.

If we do not know anything or do not really care about the final sparsity level, we can still select λ to have the best possible generalization error. As usual, the generalization error is estimated using cross-validation. The leftmost dotted vertical line in figure 4.2.2 represents the value of λ with the lowest misclassification error. Since data is reused in cross-validation, this value of λ may be a little bit too optimistic (check `glmnet` help for more details). To correct for data reuse, the largest value of λ for which the error is within one standard deviation of the minimum is used instead. This is represented by the rightmost dotted vertical line in figure 4.2.2.

Finally, we measure the accuracy of the model in a different test set, not used previously. The accuracy is 81.0782% and the number of coefficients different from 0 is 56 out of the 162. Looking at the variables selected by the algorithm, we can see that the process is rather unstable. This means that if we re-run the algorithm with another cross-validation partition, it yields quite different results regarding the number of variables and which ones

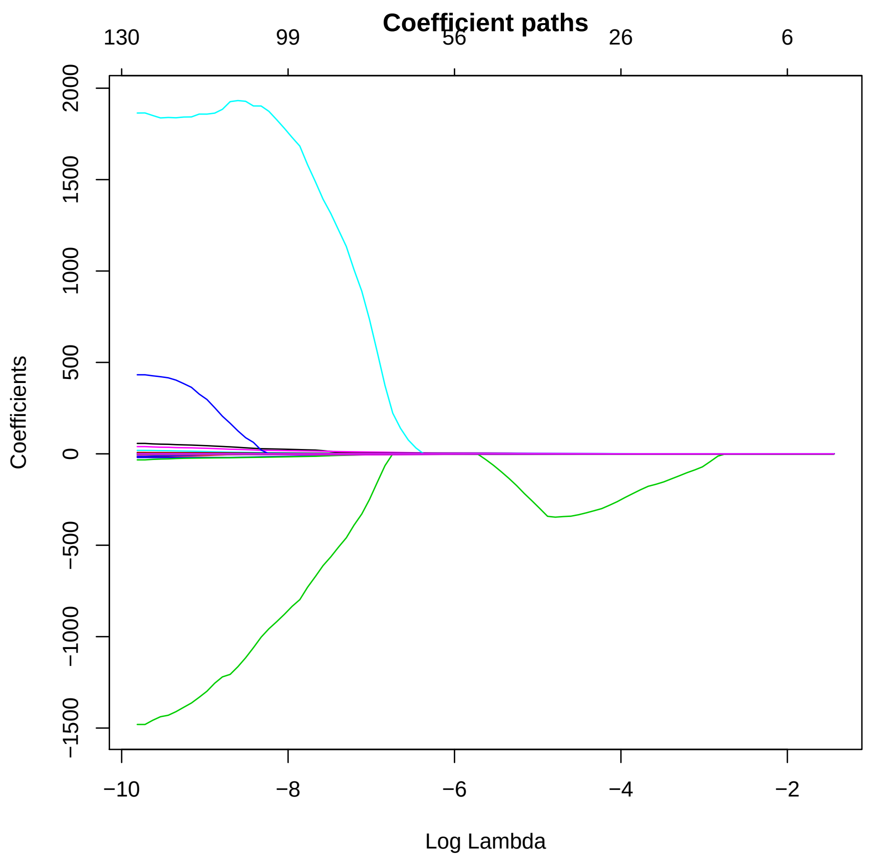


Figure 4.2.1: Coefficient paths for the logistic regression with l_1 penalty. The upper x-axis represent the number of coefficients in the model

are selected.

In order to further investigate the stability of sparse logistic regression, we can do the following experiment: repeat the previous process M times and at the end take only the coefficients that are different from 0 in all the iterations. More formally, let A_i be the set of active coefficients after the ith run of logistic regression. Then, the final set of coefficients is given by

$$B = \bigcap_{1 \leq i \leq M} A_i. \tag{4.2.1}$$

The ideal result would be that after running the previous algorithm multiple times, all the resulting sets B were the same or at least very similar. In order to compute some statistics of the algorithm, we are going to run it 100 times. Figure 4.2.3 shows the frequency of the final weights for $M = 20$. As we can see not all the resulting sets are the same, but more than 50% of the coefficients appear 100% of the time. Table 4.2.1 shows some statistics of the number of coefficients selected by sparse logistic regression after the first and the last iteration. We are interested mainly in the standard deviation, since it gives an idea of how stable is the algorithm when selecting variables. From figure 4.2.3 and table 4.2.1 we can conclude that, for this dataset, selecting variables as the intersection of different runs (equation (4.2.1)) improves the stability of the algorithm and all the “important” coefficients will always be selected.

Considering that after an iteration we are keeping only the intersection between the two

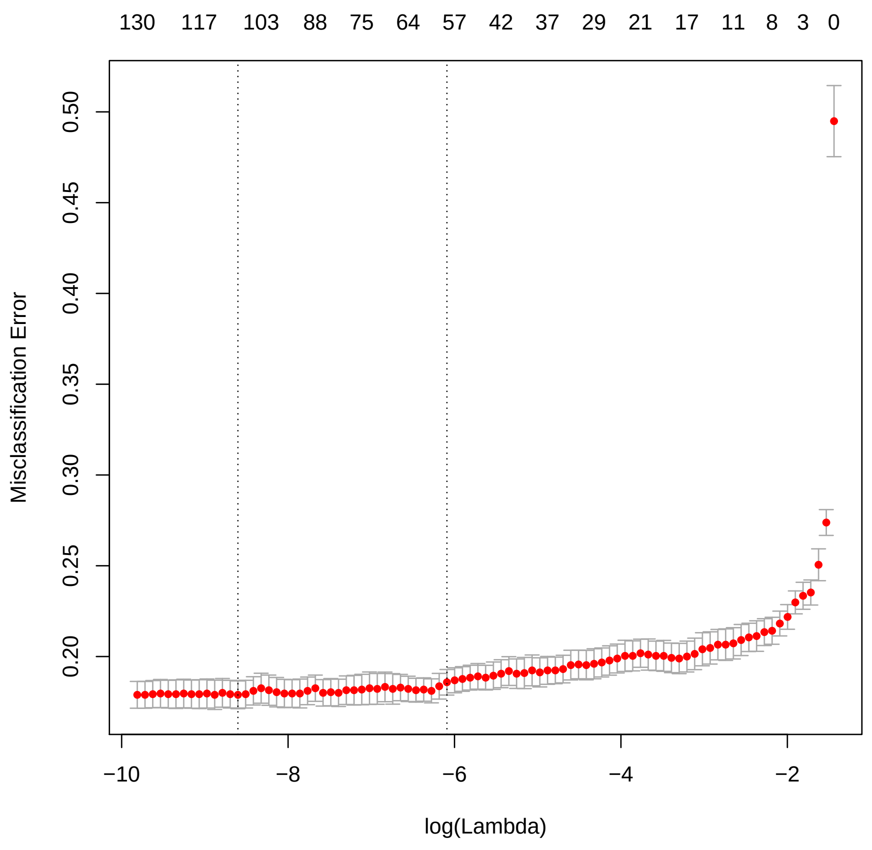


Figure 4.2.2: 10-fold cross-validation error. The upper x-axis represents the number of coefficients in the model

No. of coefficients selected	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	Sd
After first iteration	33.00	50.00	58.00	56.01	63.00	73.00	10.54
After last iteration	21.00	25.00	26.00	27.07	29.00	36.00	3.57

Table 4.2.1: Statistics of the number of coefficients selected by sparse logistic regression

sets, one could think that the number of selected variables tends to 0. However, the selected number of variables reaches a plateau, around 10 or 20 iterations in our experiments.

Clearly, the main problem of this method is of course the execution time, since for $M = 20$ it will be 20 times slower as a simple logistic regression procedure. It is also not clear how it will generalize to other datasets. For this dataset, the execution time of a full cross-validation is around 40 s, so it is quite cheap to iterate the algorithm 20 or 30 times.

Finally, in order to test if the selected variables are "useful", we can build a more powerful model (i.e. non-linear model) with only those variables and another one with all the variables. Table 4.2.2 compares these results to the previous one. As expected, the SVM with Gaussian kernel performs best in terms of accuracy. However, we can say that we are able to select the important features using this iterative version of sparse logistic regression, which is more stable.

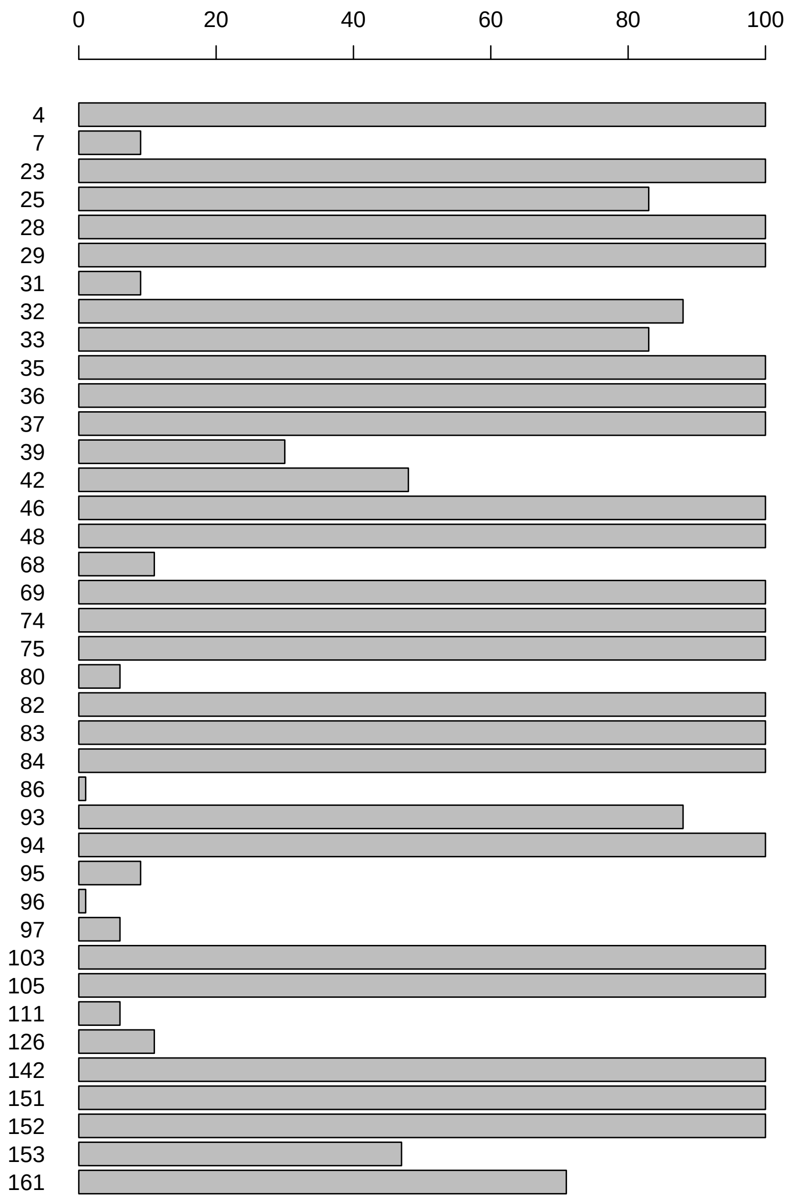


Figure 4.2.3: Frequency of the final weights after running algorithm (4.2.1) with $M = 20$ for 100 times

4.3 Sparse multi-layer perceptrons

The goal of this experiment is to perform automatic architecture selection in multi-layer perceptrons (MLP). Thus, we are going to add the l_1 penalty to the objective function somehow, inducing sparsity in the weights. In order to introduce a Lasso approach in MLP training we need to decide in which weights to apply the l_1 penalty function. Moreover, we have to take into account the different nature of the residual sum of squares (which determines the practical model error) and the l_1 term, that only affects model sparsity.

Model	Features	Accuracy (%)
SVM gaussian kernel	162/162	81.7454
Sparse logistic regression	56/162	81.0782
SVM gaussian kernel	28/162	80.9714

Table 4.2.2: Accuracies for SVM with gaussian kernel and sparse logistic regression with different sets of features

We also have to keep in mind that standard MLP training and methods such as ISTA or FISTA are quite different and it may be better to separate the two different terms of the objective function: global error and l_1 penalty.

Since our MLP model has linear output weights (that is, the activation function of the output units is linear), it is natural to apply the l_1 penalty only to them. In fact, if the task to solve is a regression problem (only one output unit), we could perform automatic architecture selection by removing all hidden units whose hidden-to-output weights are 0. More formally, let h be a hidden unit and W^O, the matrix of hidden-to-output weights. Then, after the training, if $W_h^O = 0$, the unit h can be omitted since it will not have any effect on the output y.

For the sake of simplicity let's not consider the l_2 penalty here and, as we discussed above, we are going to apply the l_1 penalty only to the output weights. This might be dangerous because the l_1 penalty could not be enough to avoid overfitting, since the input-to-hidden weights are not regularized. However, we are going to assume that this regularization is in fact sufficient. The global criterion function is

$$J(W) = J(W^H, W^O) = \mathrm{RSS}(W^H, W^O) + \frac{\alpha}{N_{W^O}}\|W^O\|_1, \tag{4.3.1}$$

where N_{W^O} denotes the number of output weights (which is the number of hidden units plus one to account for the bias). We should try to improve more or less simultaneously the RSS and the $\|W^O\|_1$ terms. Next, we are going to try using batch training as the optimization procedure.

Usually, a desirable property of optimization algorithms is monotonicity. Therefore, we are going to make sure that our training algorithm decreases J at every iteration. The simplest way to achieve that is to make a two step procedure: first we fix W^O and minimize the objective function with respect to W^H and vice versa. We show next that using this procedure the objective function J decreases monotonically after every step. Let's assume $W^H = \overline{W}^H$ is fixed, then we want to minimize the function

$$J_1(W^O) = J(\overline{W}^H, W^O) = \mathrm{RSS}(\overline{W}, W^O) + \frac{\alpha}{N_{W^O}}\|W^O\|_1. \tag{4.3.2}$$

This is just the standard linear square error plus the l_1 penalty, which can be optimized with algorithms such as FISTA. Now let's assume that $W^O = \overline{W^O}$. The new objective function to minimize is

$$J_2(W^H) = J(W^H, \overline{W}^O) = \mathrm{RSS}(W^H, \overline{W}^O) + \frac{\lambda_1}{N_{W^O}}\|\overline{W}^O\|_1 = RSS(W^H, \overline{W}^O). \tag{4.3.3}$$

This can be optimized using a simple modification of the backpropagation algorithm that does not modify the hidden-to-output weights, but takes them into account in order to compute the error.

Let (W_k^H, W_k^O) be the input and output weights at step k. Fixing W_k^H, FISTA can be applied on W^O to get a new set of weights W_{k+1}^O that represent an improvement in the objective function

$$J(W_k^H, W_{k+1}^O) = \mathrm{RSS}(W_k^H, W_{k+1}^O) + \alpha \|W_{k+1}^O\|_1 \leq J(W_k^H, W_k^O).$$

Notice that although the previous inequality holds, FISTA may increase the least squares error as a result, that is, $\mathrm{RSS}(W_k^H, W_{k+1}^O) > \mathrm{RSS}(W_k^H, W_k^O)$. The next step is aimed to correct this, and it can be achieved by applying a fast batch optimization procedure such as Conjugate Gradient (CG). This will yield a weight matrix W_{k+1}^H such that

$$J(W_{k+1}^H, W_{k+1}^O) = \mathrm{RSS}(W_{k+1}^H, W_{k+1}^O) < J(W_k^H, W_{k+1}^O) = \mathrm{RSS}(W_k^H, W_{k+1}^O).$$

Considering together the FISTA and CG steps we get

$$J(W_{k+1}^H, W_{k+1}^O) < J(W_k^H, W_{k+1}^O) < J(W_k^H, W_k^O).$$

It is important to note that the previous procedure will minimize the error no matter the order in which FISTA and CG are applied, so we can start with any of them. It is also worth mention that the FISTA algorithm itself is not monotonic, so it has to converge in order to minimize the objective function. A possible outline of the algorithm is shown next.

Algorithm 6: Batch Conjugate Gradient and FISTA MLP training

```
Input: Sample $\mathcal{S} = \{(X^p, y^p)\}$ and FISTA parameter $\alpha$
Initialize $k = 0$, $W_0 = (W_0^H, W_0^O)$
while stopping condition == FALSE do
    $W_k^H = \mathrm{CG}(W_{k-1}^H, \mathrm{RSS}(\cdot, W_k^O))$
    $W_k^O = \mathrm{FISTA}(W_{k-1}^O, \mathrm{RSS}(W_{k-1}^H, \cdot) + \alpha\|\cdot\|_1)$
end
```

The previous algorithm can be also analyzed from the standard MLP backpropagation training point-of-view. Given an input matrix $\mathbf{X}$, the MLP function can be described as

$$y = F(X; W^H, W^O) = H^t W^O$$

where $H = \phi(X^t W^H)$ is the non-linear projection of the input X into its hidden layer representation and ϕ is the activation function (possibly non-linear, otherwise the projection is linear).

Therefore an MLP can be seen as a two step procedure: first it creates new features using non-linear combinations of the original features and then performs a linear regression on the new features. In our algorithm, instead of using the standard residual sum of squares objective function, the penalty l_1 is added, performing a Lasso on the new data matrix H. More formally, let (W_k^H, W_k^0) be the weights at step k. First the following objective function is optimized using Conjugate Gradient, starting at the point W_k^H

$$J(W^H) = RSS(W^H) = \frac{1}{2N} \sum_p (y^p - F(X; W^H, W_k^O)^2.$$

Note how it only depends on W_H, since the W_k^O are fixed. As a result we obtain the weights W_{k+1}^H. Next we compute the projection of the input data using the new set of weights W^H_{k+1}, H_k and we apply FISTA to minimize the following function

$$J(W^O) = RSS(W^O) + \alpha \sum |W^O| = \frac{1}{2N} \sum_p (y^p - H^t_k W^O)^2 + \alpha \sum |W^O|.$$

The output of FISTA are the new weights W^O_{k+1}, and these two steps are iterated until some kind of stopping criterion is met.

When applying these procedures some care should be taken. For instance, if the l_1 penalty term α is relatively large, FISTA may shrink to 0 many W^O weights, making the subsequent CG iterations rather ineffective. One possible solution to this problem consists on increasing the value of α at each iteration, starting from a low value and progressively reaching the final value. Let $\alpha_{\max}$ be the final value of α and K the number of iterations. Then, the value of α for each iteration can be computed as

$$\alpha_k = \frac{k\alpha_{\max}}{K} \tag{4.3.4}$$

Several experiments using the prostate dataset[4] have been performed to see whether it was possible to train an MLP using algorithm 6. The most straightforward approach is to perform a fixed number of iterations (100 in our experiments) using also a fixed value for α. Let f be the square error and g the l_1-norm. Figure 4.3.1 (top) shows the evolution of the objective function $f + \alpha g$ with $\alpha = 1024$ and 20 hidden units. As can be seen the value of the objective function does not decrease after a few iterations of Congugate Gradient plus FISTA. This is due to the problem mentioned before, that is, FISTA shrink too many weights to 0 in the first iterations due to the aggressive α schedule. In this example, only one weight is non-zero after the third iteration and thus successive steps are useless.

This problem can be partially solved by introducing a stopping criterion on the value of the objective function. The implemented criterion is a very simple one, where the algorithm stops if the difference in absolute value of the objective function between one iteration and the previous one is below a certain threshold. As shown in figure 4.3.1 (bottom), using this condition the algorithm successfully stops when the objective function reaches a plateau.

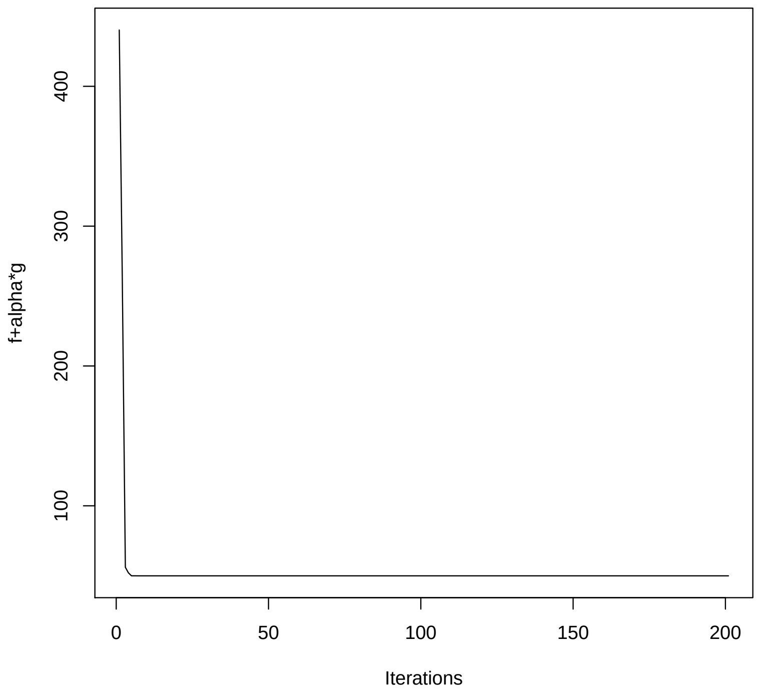
Algorithm 7: Batch Conjugate Gradient and FISTA MLP training

```
Input: Sample $\mathcal{S} = \{(X^p, y^p)\}$, number of iterations $K$, schedule parameter $t$ and
       FISTAparameter $\alpha_{\max}$
Initialize $W_0 = (W^H_0, W^O_0)$
for $k = 1$ to $K$ do
    $W^H_k = \mathrm{CG}(W^H_{k-1}, \mathrm{RSS}(\cdot, W^O_k))$
    $\alpha_k = \min\left(\alpha_{\max}, \frac{k\alpha_{\max}}{K-(t-1)}\right)$
    $W^O_k = \mathrm{FISTA}(W^O_{k-1}, \mathrm{RSS}(W^H_{k-1}, \cdot) + \alpha_k \|\cdot\|_1)$
end
```

Another possible solution, outlined in the previous section, is to use a less aggressive α schedule (equation (4.3.4)), so that FISTA will progressively shrinks the weights towards 0, and not after only a few iterations. One possible problem of equation (4.3.4) is that only one iteration of CG+FISTA will be made with the value $\alpha = \alpha_{\max}$. This may lead to the algorithm not achieving full sparsity. Thus, equation (4.3.4) can be modified as follows

$$\alpha_k = \min\left(\alpha_{\max}, \frac{k\alpha_{\max}}{K - (t-1)}\right), \tag{4.3.5}$$

[4] `http://www-stat.stanford.edu/~tibs/ElemStatLearn/datasets/prostate.data`



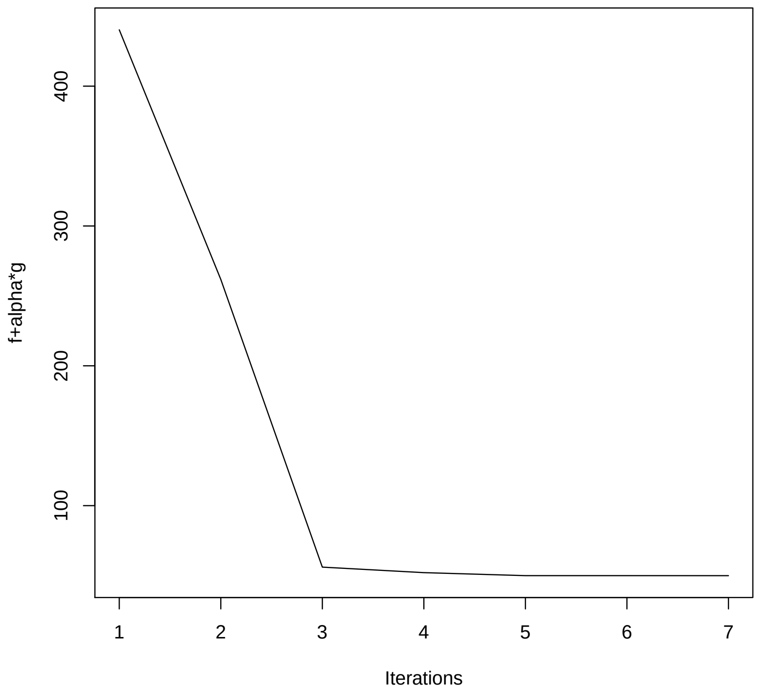


Figure 4.3.1: Evolution of the objective function with (bottom) and without (top) stopping criteria

where $t \in (0, K]$ is the number of iterations with $\alpha_{\max}$ that are going to be performed. If $t = 1$, equations (4.3.5) and (4.3.4) are equivalent. On the other hand, if $t = K$, this schedule is the same as using $\alpha = \alpha_{\max}$ in all the iterations. Algorithm 7 is a modified version of algorithm 6 that reflects all these changes. Although a stopping criterion could also be added to this algorithm, for the sake of clarity we assume that the full K iterations are always performed.

The results of the algoritm are shown in tables **??** and **??**. As we can see the amount of iterations performed at the end with $\alpha_{\max}$ increases the sparsity level and the error is practically the same. We can also see that as the number of hidden units is increased, the penalization we have to apply to achieve 100% sparsity is larger. One could expect that if we select for each number of hidden units $H_1, H_2, \ldots, H_N$ the sparsity given the lowest MAE S_i^*, then

$$\lceil H_1(S_1^*/100) \rceil = \lceil H_2(S_2^*/100) \rceil = \cdots = \lceil H_N(S_N^*/100) \rceil.$$

In that case, the optimal number of hidden units selected is clear. However, in our experiments this does not happen exactly, so we will have to further investigate reasons behind that.

nHid	alpha	MAE	Sparsity(%)
5	4	0.709434	16.6667
5	8	0.682161	16.6667
5	16	0.605985	16.6667
5	32	0.643354	16.6667
5	64	0.515909	16.6667
5	128	0.510196	16.6667
5	256	0.478706	33.3333
5	512	0.618081	100.0
5	1024	0.618081	100.0
5	2048	0.618081	100.0
5	4096	0.618081	100.0
10	4	0.557148	0.0
10	8	0.565861	0.0
10	16	0.550023	9.09091
10	32	0.549371	18.1818
10	64	0.613252	36.3636
10	128	0.511189	45.4545
10	256	0.497758	45.4545
10	512	0.526149	72.7273
10	1024	0.618081	100.0
10	2048	0.618081	100.0
10	4096	0.618081	100.0
20	4	0.628681	28.5714
20	8	0.64252	33.3333
20	16	0.611457	33.3333
20	32	0.618805	33.3333
20	64	0.53307	33.3333
20	128	0.509943	42.8571
20	256	0.435704	57.1429
20	512	0.53193	61.9048
20	1024	0.538629	90.4762
20	2048	0.618081	100.0
20	4096	0.618081	100.0
40	4	0.591329	51.2195
40	8	0.563564	56.0976
40	16	0.636757	60.9756
40	32	0.534465	63.4146
40	64	0.566172	65.8537
40	128	0.529359	68.2927
40	256	0.54396	78.0488
40	512	0.467754	82.9268
40	1024	0.485833	82.9268
40	2048	0.521624	92.6829
40	4096	0.618081	100.0
80	4	0.689637	64.1975
80	8	0.649843	72.8395
80	16	0.578509	81.4815
80	32	0.612448	77.7778
80	64	0.572195	81.4815
80	128	0.607766	83.9506
80	256	0.541409	86.4198
80	512	0.545421	83.9506
80	1024	0.590662	86.4198
80	2048	0.50664	88.8889
80	4096	0.49452	92.5926

Table 4.3.1: Results of algorithm 7 with $K = 100$ and $t = 1$

nHid	alpha	MAE	Sparsity(%)
5	4	0.707403	16.6667
5	8	0.680784	16.6667
5	16	0.601	16.6667
5	32	0.606875	16.6667
5	64	0.513783	16.6667
5	128	0.50564	16.6667
5	256	0.482147	33.3333
5	512	0.618081	100.0
5	1024	0.618081	100.0
5	2048	0.618081	100.0
5	4096	0.618081	100.0
10	4	0.598195	0.0
10	8	0.573201	0.0
10	16	0.579802	18.1818
10	32	0.560766	18.1818
10	64	0.536612	36.3636
10	128	0.528813	45.4545
10	256	0.497801	45.4545
10	512	0.526177	72.7273
10	1024	0.618081	100.0
10	2048	0.618081	100.0
10	4096	0.618081	100.0
20	4	0.629118	28.5714
20	8	0.600615	33.3333
20	16	0.587248	33.3333
20	32	0.519149	33.3333
20	64	0.570605	42.8571
20	128	0.482249	52.381
20	256	0.531595	66.6667
20	512	0.512466	71.4286
20	1024	0.537595	85.7143
20	2048	0.618081	100.0
20	4096	0.618081	100.0
40	4	0.576683	56.0976
40	8	0.550674	58.5366
40	16	0.580663	65.8537
40	32	0.545226	63.4146
40	64	0.644038	68.2927
40	128	0.541507	68.2927
40	256	0.484605	75.6098
40	512	0.509772	80.4878
40	1024	0.489849	82.9268
40	2048	0.525916	95.122
40	4096	0.618081	100.0
80	4	0.674687	60.4938
80	8	0.606669	72.8395
80	16	0.590575	77.7778
80	32	0.56183	79.0123
80	64	0.62943	81.4815
80	128	0.584787	82.716
80	256	0.588908	86.4198
80	512	0.430071	85.1852
80	1024	0.567595	90.1235
80	2048	0.475463	92.5926
80	4096	0.497544	97.5309

Table 4.3.2: Results of algorithm 7 with $K = 100$ and $t = 5$

Chapter 5

Conclusions and further work

5.1 Discussion

The main objective of this master thesis was to present the state of the art in proximal optimization. First, the classic models are discussed: Lasso, Group Lasso, Fused Lasso and Elastic Net. Then, after developing some necessary convex optimization theory, we presented the FISTA algorithm, which is a modification of Nesterov's optimal gradient method. FISTA is a general algorithm able to minimize the sum of any two functions, as long as one of them has a Lipchitz continuous gradient, the other is convex and has an easily computable proximal operator. Thus, all the previous models (Lasso, Elastic Net, ...) can be solved using FISTA.

A C implementation of the FISTA algorithm was also developed in this thesis. The main objective of this implementations is to overcome the limitations of Matlab when dealing with very large datastes. The C code was faster than two publicly available Matlab implementations of FISTA in all the benchmarks and it was also used for the experiments in sections 4.1 and 4.3.

We also performed experiments with real-world datasets, different from the well known datasets used in the literature. Lasso and Elastic Net are compared, with the second being usually the best model with a proper parameter selection. This is not surprising, since the Lasso is a particular case of Elastic Net when $\lambda_2 = 0$. The Lasso was also compared to other more straightforward algorithms such as Ordinary Least Squares and Ridge Regression. The results are as expected, that is, the Lasso and Ridge Regression are comparable in terms of the error, but the first one produces also interpretable models, since many of the weights will be zero. Ordinary Least Squares depends heavily on the dataset, and it does not work well if the number of variables is big in comparison to the number of training patterns.

The capabilities of the l_1 penalization as an embedded feature selection method were also studied in a bioinformatic classification setting. Hence, the loss function for this case is the binomial log-likelihood, instead of the usual residual sum of squares.

Finally we have investigated how to train sparse multi-layer perceptrons, that is, adding a l_1 penalization to the perceptron error function. The multi-layer perceptron training can be seen as a two step procedure, first the input variables are non-linearly combined and then a standard linear regression is performed on those new features. Therefore, we tried to replace the linear regression step by the Lasso, that adds a l_1 penalty and thus enforces sparsity in the hidden-to-ouput weights. The results are mixed. On one hand we were able to train the sparse multi-layer perceptrons effectively but on the other hand we were not able to obtain any clear conclusion regarding the relation between the λ_1 parameter of the

l_1 penalty and the number of units in the hidden layer.

5.2 Further work

As future work, the first priority is to introduce more features into the C implementation of FISTA. Right know it can only handle the residual sum of squares loss function, and the plan is to introduce others such as the binomial log-likelihood. Thus, experiments in section 4.2 could be done with this software, instead of the `glmnet` package. The C code can be further improved by using OpenMP to handle parallelization. That way the proximal operator could be computed in parallel if multiple cores are available, making the whole algorithm much faster. It is also pending work to compute benchmarks for all the public implementations listed in the appendix A.

As we mentioned throughout the thesis, there are other state-of-the-art algorithms apart from FISTA that can solve the Lasso and Elastic Net problems. Example of such algorithms are Coordinate Descent, used by R package `glmnet`, and Interior Point Methods. Therefore it would be interesting to investigate how the performance of these two general algorithms compare.

Next, continuing with the experiments presented in section 4.2, there are some other proposals like [32] and [33] to make the Lasso model more stable for feature selection. This is an interesting topic with growing attention in some fields, such as classification of microarray gene expression datasets.

Lastly, we can try to use the results in section 4.3 to automatically select the network architecture of a multi-layer perceptron, that is, the optimal number of hidden units. At the present time the results are not conclusive, but there are more options and we are working on them.

Appendix A

LIBFISTA

A.1 Introduction

One of the contributions of this thesis is the development of a library written in C which implements the FISTA algorithm (section 3.3.3). Other publicly available FISTA implementations are:

- L1 benchmark package. Matlab implementation (`http://www.eecs.berkeley.edu/~yang/software/l1benchmark/`).
- SparseLab. Matlab implementation (`http://sparselab.stanford.edu/`).
- P-FISTA. C implementation, only solves the Lasso problem (`http://www.caam.rice.edu/~optimization/disparse/p-fista.html`).
- Forward-Backward Proximal Splitting. Matlab implementation, solves many different problem (`https://www.ceremade.dauphine.fr/~peyre/numerical-tour/tours/optim_4_fb/`).
- SolveLA. Matlab implementation based on ProxTV (`http://arantxa.ii.uam.es/~gaa/software.html`).

Matlab also has its own implementation of Lasso and Elastic Net in the Statistics Toolbox, but of course it is not publicly available. The main features of LIBFISTA are:

1. Implemented in ANSI C.
2. Solves the Lasso, Elastic Net, Ordinary Least Squares and Ridge Regression models.
3. Automatically standarizes variables.
4. Two versions of FISTA are implemented, backtracking and constant step size [26].

LIBFISTA minimizes the residual sum of squares together with any combination of the l_1 and l_2 penalizations. Therefore, in order to build a Lasso model, $\lambda_1 > 0$ and $\lambda_2 = 0$. For Ridge Regression, $\lambda_1 = 0$ and $\lambda_2 > 0$ and so on. The library is coded in ANSI C, using matrix and vector functions from [34]. The FISTA algorithm with backtracking is a C implementation of the function `solveLasso` from the L1 benchmark package, with some modifications to solve also the Elastic Net model. The FISTA algorithm with constant step size is a straightforward implementation of the one in Teboulle's paper [26]. The Lipschitz constant is estimated as the largest eigenvalue of the matrix $\mathbf{X}^t\mathbf{X}$. The C code to compute eigenvalues is also taken from [34].

The help page for LIBFISTA is:

```
Usage: fista [OPTION]... TRAIN
Builds an Elastic Net model using the FISTA algorithm

Options:
-h, --help                show help and exit
-t, --test=FILE           test file
-l, --l1=LAMBDA_1         l1 norm parameter [default: 1e-6]
-r, --l2=LAMBDA_2         l2 norm parameter [default: 0]
-b, --backtracking        use backtracking in FISTA
-z, --standarize          standarize data to zero mean and unit variance
-v, --verbose             verbose
```

A.2 Benchmarks

In this section we are going to perform some benchmarks to compare the Matlab and C implementations. All the datasets we are going to use are publicly available and they are summarized in table A.2.1. Note that all of them have a really low number of variables, since it is difficult to find some regression datasets with more. We have available meteorological datasets, which usually have much more variables, so we plan to add them to this benchmark in the future.

Dataset	No. of patterns	No. of variables
abalone	4177	8
bodyfat	252	14
cpusmall	8192	12
housing	506	13
mpg	392	7
prostate	67	8
space_ga	3107	6

Table A.2.1: Datasets used in the benchmark

The benchmarks compare the MAE, number of iterations and execution time of the implementations for all datasets. The results are in tables A.2.2, A.2.3 and A.2.4. The MAE is mainly to check that the implementations are correct, since it should be really similar for all of them. As we can see, the main conclusion of this section is that C implementations clearly outperform the Matlab ones. Note that in the C implementations the execution time is measured with a precision of 0.01 s, so any execution time below that is 0.

Dataset	l1 benchmark	SolveLA	LIBFISTA constant	LIBFISTA backtracking
abalone	1103	4327	4228	1103
bodyfat	542	96	96	447
cpusmall	176	336	336	176
housing	495	699	699	479
mpg	415	889	889	611
prostate	188	141	141	166
space_ga	1291	1746	1746	1291

Table A.2.2: Comparison of the iterations

Dataset	l1 benchmark	SolveLA	LIBFISTA constant	LIBFISTA backtracking
abalone	1.584 32	1.584 33	1.584 33	1.584 32
bodyfat	0.000 926	0.000 923	0.000 924	0.000 925
cpusmall	6.163 19	6.1637	6.1637	6.163 19
housing	3.270 83	3.270 88	3.270 88	3.270 88
mpg	2.4993	2.499 31	2.499 31	2.499 31
prostate	0.498 613	0.498 613	0.498 613	0.498 604
space_ga	0.096 455	0.096 454	0.096 454	0.096 455

Table A.2.3: Comparison of the MAE

Dataset	l1 benchmark	SolveLA	LIBFISTA constant	LIBFISTA backtracking
abalone	0.347 579	0.627 829	0.13	0.16
bodyfat	0.041 969	0.040 002	0	0
cpusmall	0.078 051	0.060 374	0.08	0.06
housing	0.038 408	0.037 918	0.01	0
mpg	0.028 729	0.042 252	0	0
prostate	0.011 14	0.009 293	0	0
space_ga	0.211 725	0.142 273	0.09	0.04

Table A.2.4: Comparison of the elapsed time in seconds

Bibliography

[1] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining and Inference and and Prediction.* Springer, corrected edition, August 2003.

[2] Christopher Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics).* Springer, 1st ed. 2006. corr. 2nd printing 2011 edition, October 2007.

[3] Nello Cristianini. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods.* Cambridge University Press, 1 edition, March 2000.

[4] Aharon Ben-Tal and Arkadi Nemirovski. *Lectures on modern convex optimization: analysis and algorithms and and engineering applications.* Society for Industrial and Applied Mathematics, Philadelphia and PA and USA, 2001.

[5] Vladimir Cherkassky and Yunqian Ma. Practical selection of svm parameters and noise estimation for svm regression. *Neural Networks*, 17:113–126, 2004.

[6] Stuart Geman, Elie Bienenstock, and René Doursat. Neural Networks and the Bias/Variance Dilemma. *Neural Computation*, 4(1):1–58, January 1992.

[7] Robin L. Plackett. Some theorems in least squares. *Biometrika*, 37(1-2):149–157, 1950.

[8] Artificial neural networks FAQ, 2002. Available as `http://www.faqs.org/faqs/ai-faq/neural-nets/`.

[9] Robert Tibshirani. Regression Shrinkage and Selection Via the Lasso. *Journal of the Royal Statistical Society and Series B*, 58:267–288, 1994.

[10] Leo Breiman. Better subset regression using the nonnegative garrote. *Technometrics*, 37(4):pp. 373–384, 1995.

[11] Mario A. T. Figueiredo, Robert D. Nowak, and Stephen J. Wright. Gradient Projection for Sparse Reconstruction: Application to Compressed Sensing and Other Inverse Problems. *Selected Topics in Signal Processing and IEEE Journal of*, 1(4):586–597, 2007.

[12] Charles M. Stein. Estimation of the Mean of a Multivariate Normal Distribution. *The Annals of Statistics*, 9(6), 1981.

[13] Ryan J. Tibshirani. The Lasso Problem and Uniqueness. *Electronic Journal of Statistics*, June 2013.

[14] Saharon Rosset, Ji Zhu, and Trevor Hastie. Boosting as a Regularized Path to a Maximum Margin Classifier. *Journal of Machine Learning Research*, 5:941–973, 2004.

[15] Bradley Efron, Trevor Hastie, Iain Johnstone, and Robert Tibshirani. Least angle regression. *Annals of Statistics*, 32:407–499, 2004.

[16] Ryan J. Tibshirani and Jonathan Taylor. Degrees of freedom in lasso problems. *Annals of Statistic*, 40(2), 2012.

[17] Robert Tibshirani, Michael Saunders, Saharon Rosset, Ji Zhu, and Keith Knight. Sparsity and smoothness via the fused lasso. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(1):91–108, 2005.

[18] Wenjiang J. Fu. Penalized Regressions: The Bridge versus the Lasso. *Journal of Computational and Graphical Statistics*, 7(3):397–416, 1998.

[19] Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320, April 2005.

[20] Roush and F. W. Applied linear regression. *Mathematical Social Sciences*, 3(1):92–93, July 1982.

[21] Ming Yuan and Yi Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49–67, February 2006.

[22] Erik J. Balder. On subdifferential calculus. Available as `http://www.staff.science.uu.nl/~balde101/cao10/cursus10_1.pdf`, September 2008.

[23] Heinz H. Bauschke and Patrick L. Combettes. *Convex Analysis and Monotone Operator Theory in Hilbert Spaces.* Springer, 2011.

[24] Francesco Dinuzzo. Optimization methods for large scale regularization problems. Lecture notes, 2012.

[25] Patrick L. Combettes and Jean-Christophe Pesquet. Proximal Splitting Methods in Signal Processing. *ArXiv e-prints*, December 2009.

[26] Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM J. Img. Sci.*, 2(1):183–202, March 2009.

[27] Yurii Nesterov. A method of solving a convex programming problem with convergence rate $o(1/k^2)$. *Soviet Mathematics Doklady*, 27(2):372–376, 1983.

[28] Amir Beck and Marc Teboulle. Fast gradient-based algorithms for constrained total variation image denoising and deblurring problems. *Image Processing and IEEE Transactions on*, 18(11):2419–2434, 2009.

[29] Álvaro Barbero, Jorge López, and José R. Dorronsoro. Kernel methods for wide area wind power forecasting. In *Proceedings of the European Wind Energy Conference and Exhibition (EWEC–08)*, Brussels and Belgium, April 2008.

[30] Hind Azegrouz, Gopal Karemore, Alberto Torres, Carlos M. Alaíz, Ana M. Gonzalez, Pedro Nevado, Alvaro Salmerón, Teijo Pellinen, Miguel A. del Pozo, José R. Dorronsoro, and María C. Montoya. Cell-based fuzzy metrics enhance high-content screening (hcs) assay robustness. *Journal of Biomolecular Screening*, 2013.

[31] Jerome H. Friedman, Trevor Hastie, and Rob Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1–22, 2 2010.

[32] Roman Zakharov and Pierre Dupont. Stable lasso for high-dimensional feature selection through proximal optimization. In *Regularization and Optimization and Kernel Methods and Support Vector Machines: theory and applications*, Brussels and Belgium, July 2013.

[33] Edouard Grave, Guillaume Obozinski, and Francis R. Bach. Trace lasso: a trace norm regularization for correlated designs. *CoRR*, abs/1109.1990, 2011.

[34] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, 3 edition, September 2007.