Linear regression

1. Basics: setting up the problem

Linear regression is a form of supervised learning, which we have already introduced in the previous chapter. Here, we give a slightly more formal description of the process that supervised learning entails. We start with a brief summary of the typical workflow starting with some data.

The first step, as we said, is EDA (see Chapter 0). Then we need to decide on the task as well as the variables to use in the analysis as predictors and outcomes. Formally, this will lead to a declaration of the *predictor space* X and the *outcome space* Y. Both can be either continuous (i.e. quantitative) or discrete (e.g., categorical). Then we define the samples that we use for the *training* task, i.e. $x^{(i)} \in X$, as well as $y^{(i)} \in Y$ (the *training set*). At this point, we will also have to keep some samples aside for *validation*, but this will be explained in depth at a later point.

For a given task and training set, the goal is to find a function $y = f(x)$ relating *predictor* variables x to *outcome* variable y by defining a *loss function* $L\big(f(x), y\big)$. The loss function can sometimes be thought of as a function that computes the error between the model's estimates and the ground truth: f and L are thus intimately coupled. The data of *training set* is a set of N input-output pairs, or data points:

$$x^{(i)}, y^{(i)}, \quad i = 1, \ldots, N$$

Note that neither $x^{(i)}$ nor $y^{(i)}$ have to be uni-variate, as they can both have multiple features. The goal is to be able to predict y^{in} from a new observation x^{in}. Somehow we should be able to take uncertainty into account.

Schematically, the basic process for supervised learning is as following:

(1) Start with data $(x^{(i)}, y^{(i)}), \quad i = 1, \ldots, N$.

(2) Decide: Classification or regression?

(3) Define a *loss function* $L(y, f(x))$

(4) Minimise the *mean sample loss*: $E(L) = \frac{1}{N} \sum_{i=1}^{N} L(y^{(i)}, f(x^{(i)}))$

(5) Try to also achieve a reasonable *expected test loss*:

$$E\big(L(y^{\text{in}}, f(x^{\text{in}}))\big)$$

In regression, the mean sample loss is typically the mean squared error:

$$L_{MSE}\big(y, f(x)\big) = \frac{1}{N}\sum_{i=1}^{N}\big(y^{(i)} - f(x^{(i)})\big)^2 \tag{1.1}$$

In general, the solution for f will be the result of an *optimisation*, minimising the in-sample loss (error) L. That is, we are aiming to minimise the following:

$$\mathbb{E}\Big[L\big(f(\{x^{(i)}\}), \{y^{(i)}\}\big)\Big] \text{ for } \{(x^{(i)}, y^{(i)}\}_{i\in\text{training}}$$

However, we also want to avoid *over-fitting*, i.e. obtaining a model that fits the training data perfectly, but will not perform well on unseen data. This means that we need to ensure that the expected out-of-sample loss

$$\mathbb{E}\Big[L\big(f(\{x^{(k)}\}), \{y^{(k)}\}\big)\Big] \text{ for } \{(x^{(k)}, y^{(k)})\}_{k\in\text{test}}$$

is also small. This a very important property is also known as *generalisability* - we will discuss several strategies to avoid overfitting and improve generalisability in this course.

When the model has good generalisation power, we have that $f(x_{\text{unknown}})$ is be a good predictor for *totally unseen* y_{unknown} for which we have no ground truth.

2. The Solution of Linear Regression: the Least Squares method

In the case of linear regression, we assume that our observations follow a **linear** relationship with respect to the input variables. This assumption could be based on some prior knowledge about the data (e.g. based on a known model with a basis in physics, biology, economics, etc); it could follow from our exploratory data analysis; or it could be dictated by convenience.

For simplicity of notation, we will restrict ourselves to the case where the observable variable is univariate and real: $y^{(i)} \in \mathbb{R}$.

The data will look like:

$$\big\{x_1^{(i)}, x_2^{(i)}, \ldots, x_p^{(i)}, \quad y^{(i)}\big\}_{i=1}^{N} \tag{1.2}$$

We then write the following linear model for our observations:

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p =: f_{\text{LR}}(\mathbf{x}, \boldsymbol{\beta}), \tag{1.3}$$

where the vector $\boldsymbol{\beta}$

$$\boldsymbol{\beta} = \begin{pmatrix}\beta_0\\ \vdots \\ \beta_p\end{pmatrix}_{(p+1)\times 1} \in \mathbb{R}^{(p+1)}$$

contains the $(p+1)$ *parameters* of the model, which need to be estimated (*learnt*) from the training data.

For every data point, we will then have a predicted value

$$\hat{y}^{(i)} = f_{\text{LR}}(\mathbf{x}^{(i)}, \boldsymbol{\beta}) = \beta_0 + \beta_1 x_1^{(i)} + \cdots + \beta_p x_p^{(i)}, \qquad i = 1, \ldots, N.$$

Ideally, given the data (1.2), we would want to find the values of the parameters $\boldsymbol{\beta}$ such that the prediction $\hat{y}^{(i)}$ is equal to the observation $y^{(i)}$ for every single data point:

$$\left\{\begin{array}{l} \hat{y}^{(1)} = \beta_0 + \beta_1 x_1^{(1)} + \cdots + \beta_p x_p^{(1)} \stackrel{?}{=} y^{(1)} \\ \hat{y}^{(2)} = \beta_0 + \beta_1 x_1^{(2)} + \cdots + \beta_p x_p^{(2)} \stackrel{?}{=} y^{(2)} \\ \quad \vdots \\ \hat{y}^{(N)} = \beta_0 + \beta_1 x_1^{(N)} + \cdots + \beta_p x_p^{(N)} \stackrel{?}{=} y^{(N)} \end{array}\right\} \tag{1.4}$$

But the system of linear equations (1.4) in the parameters is usually over-determined ($N \gg (p+1)$, more data points, hence equations, than unknowns β_i), which means that unless we have a very special case where all the data points are collinear, there will *not* be one solution for $\beta_0, \ldots, \beta_p$ that satisfies each equation in the system with the $\stackrel{?}{=}$ in (1.4).

What to do? The solution comes from an associated problem, the *Least Squares* (LS) problem, which has many nice and interesting properties. Essentially, it gives a set of parameters such that our predictions are *close* to the observations in a precise sense.

To see how this problem is solved in generality, we rewrite (1.4) in matrix-vector form. Let us organise our given data into a matrix $\mathbf{X}$ containing the descriptor variables and a vector $\mathbf{y}$ containing the observed variables:

$$\mathbf{X} = \begin{bmatrix} 1 & x_1^{(1)} & \cdots & x_p^{(1)} \\ 1 & x_1^{(2)} & \cdots & x_p^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(N)} & \cdots & x_p^{(N)} \end{bmatrix}_{N\times(p+1)} \qquad \mathbf{y} = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(N)} \end{bmatrix}_{N\times 1}$$

It is then clear that the $N \times 1$ vector of predicted values $\hat{\mathbf{y}}$ is given by:

$$\hat{\mathbf{y}} = \mathbf{X}\boldsymbol{\beta}.$$

Since we want the parameter values such that prediction and observations are close, we need to quantify the *error* of our model:

$$\mathbf{e} = \mathbf{y} - \hat{\mathbf{y}} = \mathbf{y} - \mathbf{X}\boldsymbol{\beta}.$$

The meaning of the error is clear in Figure 1.1. The vector $\mathbf{e}_{N\times 1}$ contains all the deviations between the predicted values and the observed outcomes.

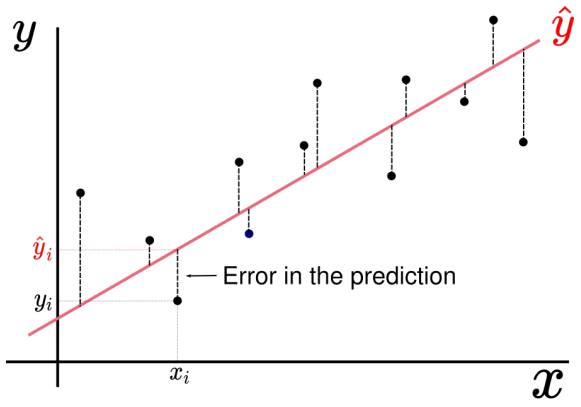

Figure 1.1. Simple linear regression task with one predictor and one outcome variable.

The LS method finds a solution for the values of $\boldsymbol{\beta}$ that minimises the *Mean Squared Error* (MSE) (1.1). In our matrix-vector notation, the MSE can be written compactly as:

$$L\big(f_{\mathrm{LR}}(\mathbf{x}), y\big) = \frac{1}{N}\mathbf{e}^T\mathbf{e} = \frac{\|\mathbf{e}\|^2}{N}.$$

Key message: The parameters $\boldsymbol{\beta}$ of the linear regression model are obtained by solving the **Least Squares optimisation problem**:

$$\min_{\boldsymbol{\beta}} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|^2$$

i.e., given $\mathbf{X}$ and $\mathbf{y}$, find $\boldsymbol{\beta}$ that minimises the MSE loss function:

$$L(\boldsymbol{\beta}) = \frac{1}{N}\|\mathbf{e}\|^2 = \frac{1}{N}\big[(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})\big]. \tag{1.5}$$

Optimisations can be (and in most interesting cases are) very complicated. In fact, you will spend this course optimising difficult loss functions. But the LS optimisation for this linear case is, unsurprisingly, easy.

We will do two things about this easy optimisation: (1) in this section, we will first solve it explicitly, so that you gain some insight into notation that is used widely in the literature (and throughout the course); (2) in a later section, we will show how it can be solved numerically in a systematic manner.

2.1. Explicit, analytical minimisation: We have to minimise the loss function L (1.5) in the parameter space of the β_i's, i.e., the loss function is a multivariable real function of $\boldsymbol{\beta}$. To this end, we find the value $\boldsymbol{\beta}^\star$ where the gradient vanishes:

$$\left.\frac{dL}{d\boldsymbol{\beta}}\right|_{\boldsymbol{\beta}^\star} = \nabla_{\boldsymbol{\beta}} L\big|_{\boldsymbol{\beta}^\star} = \mathbf{0}$$

Expand the loss function:

$$L(\boldsymbol{\beta}) = \frac{1}{N}\big[\mathbf{y}^T\mathbf{y} - \mathbf{y}^T\mathbf{X}\boldsymbol{\beta} - \boldsymbol{\beta}^T\mathbf{X}^T\mathbf{y} + \boldsymbol{\beta}^T\mathbf{X}^T\mathbf{X}\boldsymbol{\beta}\big]$$

Quick aside: Check you understand these facts and notation as they are widely used in ML algorithms.

$$\nabla_{\boldsymbol{\beta}}(\boldsymbol{\alpha}^T\boldsymbol{\beta}) = \nabla_{\boldsymbol{\beta}}(\alpha_1\beta_1 + \ldots + \alpha_p\beta_p) = \begin{pmatrix}\alpha_1\\ \vdots \\ \alpha_p\end{pmatrix} = \boldsymbol{\alpha}$$

$$\nabla_{\boldsymbol{\beta}}(\boldsymbol{\beta}^T\boldsymbol{\alpha}) = \boldsymbol{\alpha}$$

$$\nabla_{\boldsymbol{\beta}}(\boldsymbol{\beta}^T\mathbf{A}\boldsymbol{\beta}) = \mathbf{A}\boldsymbol{\beta} + \mathbf{A}^T\boldsymbol{\beta} = (\mathbf{A} + \mathbf{A}^T)\boldsymbol{\beta}$$

Back to the loss function, it is then easy to check that:

$$\begin{aligned}\nabla_{\boldsymbol{\beta}} L(\boldsymbol{\beta}) &= \frac{1}{N}\Big[-\mathbf{X}^T\mathbf{y} - \mathbf{X}^T\mathbf{y} + \big(\mathbf{X}^T\mathbf{X} + (\mathbf{X}^T\mathbf{X})^T\big)\boldsymbol{\beta}\Big]\\ &= -\frac{2}{N}\Big[\mathbf{X}^T\mathbf{y} - \mathbf{X}^T\mathbf{X}\boldsymbol{\beta}\Big]\end{aligned}$$

In order to achieve $\nabla_{\boldsymbol{\beta}} L|_{\boldsymbol{\beta}^\star} = \mathbf{0}$, we arrive at:

$$\mathbf{X}^T\mathbf{y} = (\mathbf{X}^T\mathbf{X})\boldsymbol{\beta}^\star$$

This is called the *Normal Equation*, a name that becomes obvious when rewritten as:

$$\mathbf{X}^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}^\star) = 0,$$

i.e., the error is normal to $\mathbf{X}$. Given the definition of $\mathbf{X}$, we have that $\mathbf{X}^T\mathbf{X}$ is invertible. Therefore we can write explicitly:

$$\boldsymbol{\beta}^\star = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

This is our LS solution.

To ensure that the solution $\boldsymbol{\beta}^\star$ is a minimum, we have to check that the Hessian matrix H evaluated at the optimum is positive definite, where the elements of the matrix are given by:

$$H(L)_{ij} = \frac{\partial^2 L}{\partial\beta_i\partial\beta_j}$$

Hence the Hessian matrix can be written compactly as:

$$H(L) = \nabla_{\boldsymbol{\beta}}(\nabla_{\boldsymbol{\beta}}L)$$

Remember that in the particular case of linear regression, we have:

$$L = \frac{1}{N}\left[(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})\right]$$

and:

$$\nabla_{\boldsymbol{\beta}}L = -\frac{2}{N}\Big[\mathbf{X}^T\mathbf{y} - (\mathbf{X}^T\mathbf{X})\boldsymbol{\beta}\Big]$$

Using the aside from above:

$$H = \frac{2}{N}\nabla_{\boldsymbol{\beta}}\Big((\mathbf{X}^T\mathbf{X})\boldsymbol{\beta}\Big)$$

Quick Aside: Once again, a little aside will help:

$$\nabla_{\boldsymbol{\beta}}\left(\overrightarrow{f}(\boldsymbol{\beta})\right) = \begin{pmatrix}\nabla_{\boldsymbol{\beta}}(f_1) & \dots & \nabla_{\boldsymbol{\beta}}(f_m)\end{pmatrix},$$

$$\text{where } \overrightarrow{f} = \begin{pmatrix} f_1 \\ \vdots \\ f_m \end{pmatrix}$$

$$\nabla_{\boldsymbol{\beta}}(\mathbf{A}\boldsymbol{\beta}) = \nabla_{\boldsymbol{\beta}}\begin{bmatrix}\mathbf{a}_1^T\boldsymbol{\beta} \\ \vdots \\ \mathbf{a}_m^T\boldsymbol{\beta}\end{bmatrix}$$

$$= \begin{bmatrix}\nabla_{\boldsymbol{\beta}}(\mathbf{a}_1^T\boldsymbol{\beta}) & \dots & \nabla_{\boldsymbol{\beta}}(\mathbf{a}_m^T\boldsymbol{\beta})\end{bmatrix}$$

$$\text{where } A = \begin{bmatrix}\mathbf{a}_1^T \\ \vdots \\ \mathbf{a}_m^T\end{bmatrix}$$

Remember that from a previous aside:

$$\nabla_{\boldsymbol{\beta}}(\boldsymbol{\alpha}^T\boldsymbol{\beta}) = \boldsymbol{\alpha}$$

Therefore the above equation then becomes:

$$\nabla_{\boldsymbol{\beta}}(\mathbf{A}\boldsymbol{\beta}) = \begin{bmatrix}\mathbf{a}_1 & \dots & \mathbf{a}_m\end{bmatrix} = \mathbf{A}^T$$

Now back to the Hessian:

$$H = \frac{2}{N}\nabla_{\boldsymbol{\beta}}\Big((\mathbf{X}^T\mathbf{X})\boldsymbol{\beta}\Big) = \frac{2}{N}(\mathbf{X}^T\mathbf{X})^T = \frac{2}{N}(\mathbf{X}^T\mathbf{X})$$

We have thus concluded that H does not depend on $\boldsymbol{\beta}$. Since $\mathbf{X}$ only contains data (i.e. it is constant), H also remains the same everywhere in parameter space and gives information on the local curvature of the function minimised. It is obvious that H is positive definite, which is defined to be the case iff $\mathbf{z}^T H\mathbf{z} > 0 \quad \forall \mathbf{z} \neq 0$. This works almost by inspection, but for completeness we show that $\mathbf{X}^T\mathbf{X}$ is positive definite:

$$\begin{aligned}\mathbf{z}^T\mathbf{X}^T\mathbf{X}\mathbf{z} &= (\mathbf{X}\mathbf{z})^T(\mathbf{X}\mathbf{z}) \\ &= \|\mathbf{X}\mathbf{z}\|^2 > 0 \text{ if } \mathbf{z} \neq 0.\end{aligned}$$

This shows that MSE is a convex quadratic function in the space of parameters. (Basically, the MSE is a multidimensional parabola.)

Key message: The solution to the Least-Squares (LS) problem is:

$$\boldsymbol{\beta}^\star = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} = \mathbf{X}^+\mathbf{y},$$

where $(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T =: \mathbf{X}^+$ is the *Moore-Penrose pseudo-inverse* of $\mathbf{X}$.

Our original problem had the following form:

$$\mathbf{X}\boldsymbol{\beta} = \mathbf{y}$$

where $\mathbf{X}$ is not invertible (over-determined system). Hence we **cannot** 'solve' the problem by inverting the $\mathbf{X}$ on the left:

$$\text{“}\cancel{\boldsymbol{\beta} = \mathbf{X}^{-1}\mathbf{y}}\text{”}.$$

The LS solution applies a pseudo-inversion:

$$\boldsymbol{\beta}^\star = \mathbf{X}^+\mathbf{y}.$$

There exist many computational tools that can compute Moore-Penrose pseudo-inverse $\mathbf{X}^+$. The name pseudo-inverse is not accidental, since $\mathbf{X}^+$ has many of the properties of the inverse, if the inverse does not exist. Here are some properties of the pseudo-inverse with the equivalent properties of some invertible $\mathbf{A}$:

$$\mathbf{X}^+\mathbf{X}\mathbf{X}^+ = \mathbf{X}^+ \longleftrightarrow \mathbf{A}^{-1}\mathbf{A}\mathbf{A}^{-1} = \mathbf{A}^{-1}$$
$$\mathbf{X}\mathbf{X}^+\mathbf{X} = \mathbf{X} \longleftrightarrow \mathbf{A}\mathbf{A}^{-1}\mathbf{A} = \mathbf{A}.$$

To conclude: in the case of a linear regression, we are able to find an exact, analytical solution by computing the Normal Equation. This is usually not possible with other models. In most cases, we need to use numerical optimisation instead to arrive at a solution, by looking (numerically) for a minimum of the loss function. This only works absolutely reliably if the function is convex. We will come back to this point in Section 4 when we cover briefly the computational aspects of solving the LS optimisation problem.

3. Statistical interpretation of the Least-Squares solution

The optimisation perspective that we introduced above also has an intuitive statistical interpretation. As a rule of thumb, **optimisation of convex quadratics is usually related to (multivariate) Gaussian distributions.** This is the case here. As we will show briefly in a few lines below, the above optimisation is equivalent to maximising the likelihood of a Gaussian linear model.

Without loss of generality, let us take $p = 1$ for simplicity. Then, given some data

$$(x^{(i)}, y^{(i)}),\ i = 1, \dots, N$$

we expect that our outcome will be a random variable that can be expressed as a linear combination of the input and some Gaussian *i.i.d.* noise:

$$y^{(i)} = \beta_0 + \beta_1 x_1^{(i)} + \varepsilon^{(i)}$$

This noise will be a particular instance $\varepsilon^{(i)}$ drawn i.i.d. (*independent and identically distributed*) from a Gaussian distribution with zero mean and variance σ^2:

$$\varepsilon \sim \mathcal{N}(0, \sigma^2)$$

and the different $\varepsilon^{(i)}$ are all independent.

Going back to your knowledge of Statistics, for this we can write down a *likelihood function* which is given by:

$$\forall i, \quad \mathrm{Lik}(y^{(i)} \mid \boldsymbol{\beta}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp \frac{-\big(y^{(i)} - (\beta_0 + \beta_1 x_1^{(i)})\big)^2}{2\sigma^2}$$

Then, using independence, we can compute the total likelihood for the whole data set:

$$\underset{\text{tot}}{\mathrm{Lik}}(\mathbf{y} \mid \boldsymbol{\beta}) = \prod_{i=1}^{N} \mathrm{Lik}(y^{(i)} \mid \boldsymbol{\beta})$$

From this statistical perspective, we are interested in finding the $\boldsymbol{\beta}$ that maximises the total likelihood of the model. We make our life a bit easier by maximising the logarithm of the likelihood (which is equivalent to the original problem, since the logarithm is monotonically increasing, and gives greater numerical stability). Therefore consider the *log-likelihood*:

$$\begin{aligned}
\mathcal{L}_{\text{tot}} = \log\Big(\underset{\text{tot}}{\mathrm{Lik}}\Big) &= \sum_{i=1}^{N} \log\big(\mathrm{Lik}(y^{(i)} \mid \boldsymbol{\beta})\big) \\
&= C - \frac{1}{2\sigma^2} \sum_{i=1}^{N} \big(\underbrace{y^{(i)} - (\beta_0 + \beta_1 x_1^{(i)})}_{e^{(i)}}\big)^2 \\
&= C - \frac{1}{2\sigma^2} \mathbf{e}^T \mathbf{e} \\
&= C - \frac{N}{2\sigma^2} L_{\text{MSE}},
\end{aligned}$$

where C is constant.

Key message: In statistical terms, minimising the loss function (as in the previous section) means maximising the likelihood:

$$\underbrace{-\frac{d\mathcal{L}_{\text{tot}}}{d\boldsymbol{\beta}}}_{\text{maximum likelihood}} \longleftrightarrow \underbrace{\frac{dL_{\text{MSE}}}{d\boldsymbol{\beta}}}_{\text{minimal loss (MSE)}}$$

having assumed that the variability across data points is Gaussian distributed.

4. Numerical Optimisation with Gradient Descent

In the previous sections, we optimised the loss (or the likelihood) of the linear regression problem by minimising

$$L(\boldsymbol{\beta}) = \frac{1}{N}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})$$

over the space of $\boldsymbol{\beta}$. We already showed that this function (a quadratic) is convex and has a unique minimum, given by the Normal Equation, and obtained using the pseudo-inverse of $\mathbf{X}$.

$$\boldsymbol{\beta}^\star = \mathbf{X}^+\mathbf{y}.$$

This formal solution is straightforward and easily done on paper. But in some cases, $\mathbf{X}$ may become huge and inverting big matrices may become computationally infeasible.

More importantly, there are many other problems (i.e., most other models of any interest beyond linear regression), for which the minimum of the loss function **cannot** be found analytically. That is to say, one would not be able to solve $\nabla_{\boldsymbol{\beta}} L|_{\boldsymbol{\beta}^\star} = 0$ explicitly. In other words, we will not be able to solve for the equivalent of the normal equations. In such cases, the optimisation needs to be done numerically.

We will use the least-squares problem to illustrate these ideas (although here we do have the analytical solution available and we would not need to follow the purely numerical route). We have shown that finding the optimal linear model, i.e. minimising the sum of squared errors, is in reality the minimisation of a function that is quadratic in the parameters $\boldsymbol{\beta}$. How is this done numerically without solving the normal equations?

The general type of methods through which this is done is called *gradient methods*. (We covered this in Calculus in Year 1 and you will have seen some of those in Numerical Analysis, think also of Newton-Raphson from high school.)

> **Key message:** Gradient methods rely on the simple idea: the gradient of a multivariate function gives the direction of maximum variation of the function. Hence, **following the gradient along an optimisation trajectory leads to a maximum of the function** (minimising is easily done by changing signs.)

The above idea would seem to be the solution to any problem but, of course, things are not that simple. The above approach will always take us to a *local* maximum (or minimum) but once we are at that point, the gradient is zero and we stop moving. However, we will not know if the maximum we found is the *global* maximum of the function unless we have extra conditions. **Only for convex functions we have guarantees that there is one global maximum that can be reached with gradient methods.** The LS problem is one such case since we have a quadratic function with positive definite Hessian everywhere, as shown above and exploited below.

> *Quick Aside.* Most problems you will see in the course, and those that you will find in real applications, tend to be *non-convex*, hence 'hard' to optimise. To complicate matters (but to make things interesting), convexity is elusive and some problems can be convexified by changes of coordinates or relaxations, suddenly making them 'easy'. You can read more about these issues in the book *Convex Optimisation* by Stephen Boyd (see 'Additional Reading' folder).

Numerical optimisation of the Least Squares problem: The loss function of the least squares problem can be rewritten in the following form:

$$L(\boldsymbol{\beta}) = L(\boldsymbol{\beta}^\star) + \frac{1}{2}(\boldsymbol{\beta} - \boldsymbol{\beta}^\star)^T \underbrace{H}_{\frac{2}{N}(\mathbf{X}^T\mathbf{X})} (\boldsymbol{\beta} - \boldsymbol{\beta}^\star) \tag{1.6}$$

This follows immediately from the definition of the loss function by 'completing the square':

$$\begin{aligned} L(\boldsymbol{\beta}) &= \frac{1}{N}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) \\ &= \frac{1}{N}\left((\mathbf{y} - \mathbf{X}\boldsymbol{\beta}^\star) - \mathbf{X}(\boldsymbol{\beta} - \boldsymbol{\beta}^\star)\right)^T\left((\mathbf{y} - \mathbf{X}\boldsymbol{\beta}^\star) - \mathbf{X}(\boldsymbol{\beta} - \boldsymbol{\beta}^\star)\right) \\ &= L(\boldsymbol{\beta}^\star) - \frac{2}{N}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}^\star)^T\mathbf{X}(\boldsymbol{\beta} - \boldsymbol{\beta}^\star) + \frac{1}{N}(\boldsymbol{\beta} - \boldsymbol{\beta}^\star)^T\mathbf{X}^T\mathbf{X}(\boldsymbol{\beta} - \boldsymbol{\beta}^\star) \\ &= L(\boldsymbol{\beta}^\star) + \frac{1}{2}(\boldsymbol{\beta} - \boldsymbol{\beta}^\star)^T\left(\frac{2}{N}\mathbf{X}^T\mathbf{X}\right)(\boldsymbol{\beta} - \boldsymbol{\beta}^\star), \end{aligned}$$

where we have used the normal equation condition:

$$\mathbf{X}^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}^\star) = 0$$

Clearly, this function has positive curvature at all points:

$$\nabla_{\boldsymbol{\beta}}(\nabla_{\boldsymbol{\beta}}L) = \frac{2}{N}\mathbf{X}^T\mathbf{X} = H$$

since the Hessian is positive definite. Hence the loss function $L(\boldsymbol{\beta})$ is convex in the space of parameters. Making use of the fact that if a function is convex over the whole space, the local minimum is equivalent to the global minimum, any algorithm that converges to a minimum will lead to the global minimum for this loss function.

To get some intuition, see Figure 1.2, where we show a sketch of this loss function, with lines indicating the level sets

$$s_k = \{\boldsymbol{\beta} \mid L(\boldsymbol{\beta}) = k\},$$

i.e., the sets of points where the loss function has a constant value. In the particular case of the convex loss function defined above, these ellipses are concentric and as we decrease k, we move closer to the optimal point $\boldsymbol{\beta}^\star$.

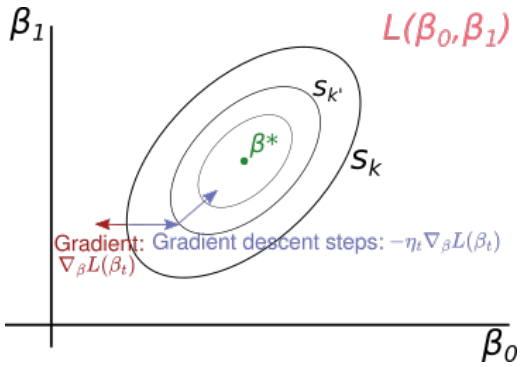

Figure 1.2. Finding $\boldsymbol{\beta}^\star$ is a convex optimisation problem in the case of linear regression. This sketch shows the optimal value of $\boldsymbol{\beta}^\star$ in green, as well as the level sets where the loss function is constant. The gradient of the loss at each point gives the direction of maximal positive variation, hence gradient descent, to find the minimum, follows the opposite of the gradient direction and makes a step size controlled by η_t at a certain iteration t.

Such a loss function can be globally minimised with gradient methods, which use the fact that $\nabla_{\boldsymbol{\beta}}L$ marks the direction of maximum change of L.

Key message: The algorithm for gradient-based optimisation of the LS problem is: iteratively follow the direction of $-\nabla_{\boldsymbol{\beta}} L$. The iteration $t+1$ will be of the form:

$$\boldsymbol{\beta}_{t+1} = \boldsymbol{\beta}_t - \eta_t \nabla_{\boldsymbol{\beta}} L(\boldsymbol{\beta}_t)$$

where η_t is the step size, which can be adjusted at each step of the algorithm.

There are many algorithms that use the gradient, amongst which are: line search, back tracking or conjugate gradient. Each of these methods has different strategies (e.g., how to choose η_t) to speed up convergence to the minimum in a shorter number of steps, see again Stephen Boyd's book on *Convex optimization* if you are interested.

As an illustration of the importance of the non-convexity of the loss function, consider Figure 1.3. From the picture, it is clear that, if we use gradient methods naively, choosing an initial guess at random for the parameter set $\boldsymbol{\beta}$ will likely result in convergence to the green point $\boldsymbol{\beta}^\star$. However, in this case there is another (deeper) minimum (the global optimum) of the function that is more difficult to reach by using gradient-based trajectories in parameter space. Hence this problem becomes difficult for gradient-based methods to crack. In fact, most loss functions in more complicated models (beyond linear regression) are usually highly non-convex, with multiple local minima. We will therefore see other optimisation methods that attempt to alleviate this issue later in the course, but be warned that no perfect solution exist for this problem.

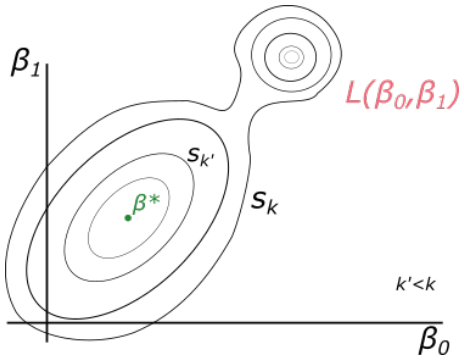

Figure 1.3. Cartoon of a non-convex loss function, with a second (deeper) well that contains the globally optimal solution that is hard to find using gradient descent methods.

5. Bias vs. variance

Here, we will describe what is sometimes known in Machine Learning as the *no-free-lunch* theorem, that is, the fact that an unbiased model will generally have high variance and, conversely, reducing the variance will increase the bias. Therefore, both aspects need to be balanced to achieve models whose predictions are as close as possible to the true value ('accurate') with as reduced variability as possible.

Quick Aside. Whilst this course will not go further into the statistical details of this problem, some pointers for interested students are as follows. For more in-depth derivations, see *Hastie, Tibshirani, Friedman, The Elements of statistical learning, chapter 3, sections 3.2, 3.3*; for the *no-free-lunch theorem*, see *Goodfellow, Bengio, Courville, Deep Learning*, chap. 5, section 5.2.1.

Here, we mention a few aspects related to this issue. In statistics, one calls an 'estimator' a rule to estimate a given quantity based on observed data. Let $\boldsymbol{\beta}$ be the true parameters

of the problem to be estimated and $\boldsymbol{\beta}^\star$ the LS estimate. First, we define two measures, the bias and the error covariance matrix of the estimator $\boldsymbol{\beta}^\star$:

$$\text{Bias: } \|\mathbb{E}[\boldsymbol{\beta}^\star] - \boldsymbol{\beta}\|$$
$$\text{Error Covariance Matrix: } \mathbb{E}\big[(\boldsymbol{\beta} - \boldsymbol{\beta}^\star)(\boldsymbol{\beta} - \boldsymbol{\beta}^\star)^T\big]$$

Note that here $\mathbb{E}$ is meant over an ensemble of different data sets, each of them giving a different value of the loss and hence of $\boldsymbol{\beta}^\star$. These two measures, based on expectation values, allow one to assess the performance of a learning algorithm across many possible data sets.

Remember the model formulation for the linear regression:

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon} \quad \boldsymbol{\varepsilon} \sim \mathcal{N}(0, \sigma^2)$$

We can then compute the expectation of our estimated parameters:

$$\begin{aligned}\mathbb{E}[\boldsymbol{\beta}^\star] &= \mathbb{E}[(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}] \\ &= \boldsymbol{\beta} + \mathbb{E}[(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\boldsymbol{\varepsilon}] = \boldsymbol{\beta}\end{aligned}$$

This particular estimator ($\boldsymbol{\beta}^\star$) is *unbiased*, i.e. $\|\mathbb{E}[\boldsymbol{\beta}^\star] - \boldsymbol{\beta}\| = 0$. That is if we obtain an estimate for a new set of samples drawn from the same data source, then we will always expect to get the right $\boldsymbol{\beta}$ on average.

On average, we will get the correct estimate, but what about the expected error of our estimator? This can be estimated from the error covariance matrix (this is the matrix from which we can obtain the MSE of the estimator by taking its trace). It can be obtained easily from our expressions above as follows:

$$\mathbb{E}\big[(\boldsymbol{\beta} - \boldsymbol{\beta}^\star)(\boldsymbol{\beta} - \boldsymbol{\beta}^\star)^T\big] = \mathbb{E}\big[(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\underbrace{\boldsymbol{\varepsilon}\boldsymbol{\varepsilon}^T}_{\mathbb{E}[\boldsymbol{\varepsilon}\boldsymbol{\varepsilon}^T]=\sigma^2\mathbf{I}}\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\big] = (\mathbf{X}^T\mathbf{X})^{-1}\sigma^2 \tag{1.7}$$

where $\mathbf{I}$ is the identity matrix. Note that σ^2 is the variance of the noise in the observations, which we can estimate from the data. In statistical terms, the estimator for σ^2 is:

$$\begin{aligned}\widehat{\sigma}^2 &= \frac{1}{N - (p+1)}\sum_{i=1}^{N}(y^{(i)} - \widehat{y}^{(i)})^2 \\ \mathbb{E}[\widehat{\sigma}^2] &= \sigma^2.\end{aligned}$$

where we use the symbol $\widehat{\cdot}$ to denote quantities estimated from data. The expected squared error of our estimated parameters depends on the empirical variance of the data *and* on the properties of $(\mathbf{X}^T\mathbf{X})^{-1}$. However, $(\mathbf{X}^T\mathbf{X})$ might be badly conditioned, hence non-invertible (it is close to losing full rank and has small singular values). **As a result, although LS is unbiased (good!), it can have high MSE in the estimated parameters (depending on properties of the data matrix X)**.

In general, for an estimator $\theta^\star$ of the true value θ, we always have that:

$$\begin{aligned}\mathbb{E}\big[(\theta - \theta^\star)^2\big] &= \mathbb{E}[\theta^2] + \mathbb{E}[\theta^{\star 2}] - 2\mathbb{E}[\theta\theta^\star] \\ &= \theta^2 + \mathrm{var}(\theta^\star) + \mathbb{E}(\theta^\star)^2 - 2\theta\mathbb{E}(\theta^\star) \\ &= \underbrace{\mathrm{var}(\theta^\star)}_{\substack{\text{variance of} \\ \text{the estimator}}} + \underbrace{\big[\theta - \mathbb{E}(\theta^\star)\big]^2}_{\text{Bias}^2}\end{aligned}$$

where we have used the fact that θ is the true value to be estimated, i.e. not a random variable.

Let $\boldsymbol{\beta}^\star_{\text{LS}}$ define the LS estimator such that $\hat{f}_{\text{LS}}(\mathbf{x}_0) = \mathbf{x}_0^T \boldsymbol{\beta}^\star_{\text{LS}}$, which is unbiased:

$$\mathbb{E}\big[\hat{f}(\mathbf{x}_0)\big] = \mathbf{x}_0^T \boldsymbol{\beta}$$

where $\boldsymbol{\beta}$ is the true value. Then we have:

Key message:

Expected squared error $= \mathbb{E}[(\hat{f} - y)^2] = \text{var}(\hat{f}) + \big(y - \mathbb{E}[\hat{f}]\big)^2 = \text{Variance} + \text{Bias}^2$.

The main issue is that by minimising the expected squared error we could face a compromise between variance and bias (bias-variance trade-off).

It is probably good to end with a visual representation of this discussion. Effectively, how good an estimator is depends on a combination of low bias and low variance. You can see an illustration of this idea in Figure 1.4, and how in some cases we might be better off with an estimator that has some bias (i.e., it is a bit *inaccurate*) but has reduced variability (i.e., it is more *reliable*).

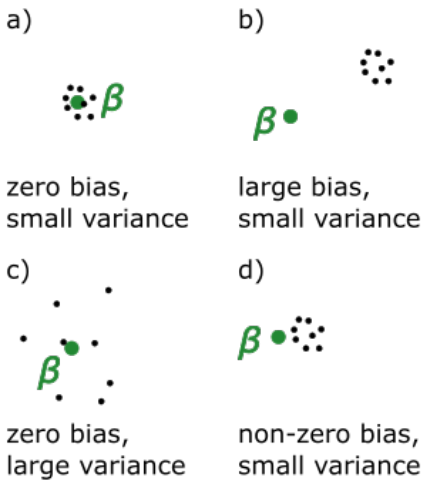

Figure 1.4. Trade-off between bias and variance. Each black dot represents an estimate $\boldsymbol{\beta}^\star$ of the true $\boldsymbol{\beta}$ (in green). In case a), we have an almost ideal case, with zero bias and small variance, which is rarely achievable in reality. Case b) shows the worse case scenario: having large bias and low variance will lead to a badly fit model. Case c) will be a somewhat good model, but with a large variance. Note that the least squares method has this issue. Finally, case d) shows the other side of the trade-off, where a small variance was achieved by introducing some small bias. In reality, one must often choose between c) and d).

Quick Aside. One could think that perhaps we could do better than LS. However, there is a general result (that follows from the Gauss-Markov theorem) that tells us that if we restrict ourselves to a *linear unbiased* estimator, we cannot do better than LS. We only sketch briefly the arguments here (see *Hastie, Tibshirani, Friedman, The Elements of Statistical Learning*, Chap. 3, for more details).

Let $\hat{\hat{f}}$ be another unbiased linear estimator different to LS. Recall that, according to the Gauss-Markov theorem, the least squares estimator is the best unbiased linear estimator (see *Hastie, Tibshirani, Friedman, The Elements of Statistical Learning*,

Chap. 3, section 3.2.2 for more details). Hence we know that:

$$\text{var}\left(\hat{f}_{\text{LS}}(\mathbf{x}_0)\right) \leq \text{var}\left(\hat{\tilde{f}}(\mathbf{x}_0)\right)$$

Since we have

$$\text{Expected squared error} = \mathbb{E}[(\hat{f} - y)^2] = \text{var}(\hat{f}) + \left(y - \mathbb{E}[\hat{f}]\right)^2$$

then it follows directly that LS has the lowest error of all linear unbiased estimators. Thus, if we want to stick with a linear model and we want it to be unbiased, we cannot do better than least-squares. On the other hand, as we saw above in (1.7), LS can have large variance.

6. Methods to reduce the variance of an estimator

Remaining within unbiased linear methods, no method can do better than LS, and the only way of improving the LS solution has to do with $\mathbf{X}^T\mathbf{X}$ (see (1.7)) to reduce directly the variance. The alternative is to go for methods that are no longer unbiased but might have lower variance (see Figure 1.4).

A second consideration here is that **increasing the interpretability of the models can be achieved through *sparse models*, which reduce the number of predictors** p by judiciously eliminating redundant descriptors.

From (1.7), it is clear that one of the sources of large variance is the inverse $(\mathbf{X}^T\mathbf{X})^{-1}$. This inverse can induce large values when $\mathbf{X}$ has a high *condition number*, i.e, when it is close to losing its full rank. In that case, $\|(\mathbf{X}^T\mathbf{X})^{-1}\| \gg 1$, and the variance (1.7) is large in particular directions of the associated vector spaces. (Of course, in the extreme case where some columns are linear combinations of others, then $\mathbf{X}$ does not have full rank and $(\mathbf{X}^T\mathbf{X})$ is not invertible. Another way of saying this is that its condition number is infinite.)

If two of the columns of $\mathbf{X}$ (i.e., the predictor variables), are nearly collinear then the condition number is large. This means that those two predictor variables are linearly related, i.e., one of them is 'redundant'. One solution is to eliminate predictor variables that are redundant, i.e., reduce the number of columns by eliminating such variables.

There is several ways of performing optimal subset selection, the two main ones being: 1. brute-force complete enumeration i.e., find the subset of k best parameters out of the p parameters by complete enumeration and benchmark of all possible choices of k out of p features; 2. Greedy sequential selection, i.e. following a sequential approach of adding/reducing descriptors until a criterion is met. The algorithms for sequential subset selection are beyond the scope of our course, we refer to *Hastie, Tibshirani, Friedman, The Elements of Statistical Learning*, Chap. 3, section 3.3 if you are interested.

In any case, selection of descriptors is a 'combinatorial optimisation problem'. Such problems tend to be 'hard' and, in many cases, can only be solved by complete enumeration, which can only be applied to small problems. For an alternative approach (very different in nature), which can scale to larger systems, we turn now to shrinkage methods.

6.1. Shrinkage methods. Shrinkage methods approach this problem by relinquishing the goal of an unbiased estimator. The key is to consider *modified* linear regressions by changing the loss function using heuristics that aim to induce sparsity in the models (i.e., by reducing the number of descriptors).

Shrinkage methods come with two main alternatives: Ridge regression and LASSO. Both methods entail a change in the loss function to be optimised, based on an heuristic

that transforms the problem from a combinatorial optimisation to a continuous optimisation. In other words, since selecting predictors is a combinatorial optimisation problem ('hard'), **shrinkage methods change the loss function to try and make the *size* of coefficients (that is, the learnt parameters) small.** The idea is to keep small the coefficients that cannot be robustly determined from data, hence **enforcing a continuous version of sparsity**. In doing so, **we lose the unbiased nature of the estimators, but we gain the potential benefit of lower variance**.

6.1.1. *Ridge regression.* As mentioned above, the idea is to mimic the elimination of descriptors in a continuous fashion. In practice, one weighs them low by introducing an altered loss function that penalises large values of $\|\boldsymbol{\beta}\|$ through the inclusion of a penalty (or regularisation) term:

$$L_{\text{RIDGE}}(\boldsymbol{\beta}) = \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|^2 + \lambda\|\boldsymbol{\beta}\|^2$$

where $\lambda > 0$ is the *penalty term* and $\|\boldsymbol{\beta}\|^2 = \sum_{i=1}^{p} |\beta_i|^2$. As before, we now seek to minimise this loss:

$$\min_{\boldsymbol{\beta}} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|^2 + \lambda\|\boldsymbol{\beta}\|^2 \quad \Leftrightarrow \quad \min_{\boldsymbol{\beta}} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|^2 \text{ subject to } \|\boldsymbol{\beta}\|^2 \leq t,$$

where the two minimisations are equivalent in the sense of duality (see aside).

> *Quick aside:* The dual equivalence of the minimisations can be obtained by considering the KKT conditions, which we do not cover in detail in this course. The KKT conditions are a generalisation of Lagrange multipliers, which you learnt for the case of *equality* constraints, to the case of *inequality* constraints, such as we have here. You can think of λ as the 'Lagrange multiplier' enforcing the constraint $\|\boldsymbol{\beta}\|^2 \leq t$. The full equivalence also follows more generally from the *strong duality* of this convex problem. This is out of the remit of this course but you can read more about it in Stephen Boyd's book on *Convex optimization*. Note that λ and t are inversely related; intuitively, the smaller we want to make $\|\boldsymbol{\beta}\|^2$ the larger our 'Lagrange multiplier' λ has to be to enforce the constraint.

This problem can be solved explicitly:

$$L_{\text{RIDGE}}(\boldsymbol{\beta}) = \mathbf{y}^T\mathbf{y} - \boldsymbol{\beta}^T\mathbf{X}^T\mathbf{y} - \mathbf{y}^T\mathbf{X}\boldsymbol{\beta} + \boldsymbol{\beta}^T(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})\boldsymbol{\beta}$$

As per usual, taking the derivative:

$$\nabla_{\boldsymbol{\beta}} L_{\text{RIDGE}}(\boldsymbol{\beta}) = -2\mathbf{X}^T\mathbf{y} + 2(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})\boldsymbol{\beta}$$

Setting the above to 0 gives:

$$\mathbf{X}^T\mathbf{y} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})\boldsymbol{\beta}^\star$$

Finally, we arrive at our solution for ridge regression:

$$\boldsymbol{\beta}^\star_{\text{RIDGE}} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$$

As above, one should check that the Hessian is positive definite, which will be left as an exercise to the reader. Note that the penalty is not applied to β_0, hence, the element of the identity matrix $\mathbf{I}$ corresponding to β_0 should be set to zero.

We now compute the bias of the estimator. To this end, we first compute the following (remember that $\mathbb{E}[\boldsymbol{\varepsilon}] = 0$):

$$\begin{aligned}\mathbb{E}\big[\boldsymbol{\beta}^\star_{\text{RIDGE}}\big] &= \mathbb{E}\Big[(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}(\mathbf{X}^T\mathbf{X})\boldsymbol{\beta} + (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\boldsymbol{\varepsilon}\Big] \\ &= (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}(\mathbf{X}^T\mathbf{X})\boldsymbol{\beta}\end{aligned}$$

Quick aside: A reminder about diagonalisation and eigendecompositions. The eigendecomposition of $\mathbf{X}^T\mathbf{X}$

$$(\mathbf{X}^T\mathbf{X}) = \mathbf{V}\mathbf{D}\mathbf{V}^T$$

has the following properties:

$$\mathbf{V}\mathbf{V}^T = \mathbf{V}\mathbf{V}^{-1} = \mathbf{I}$$
$$(\mathbf{X}^T\mathbf{X})^{-1} = \mathbf{V}\mathbf{D}^{-1}\mathbf{V}^T,$$

where $\mathbf{V}$ contains the eigenvectors as columns, and $\mathbf{D}$ is a diagonal matrix with $\mathbf{D} = \text{diag}(d_i)$, where d_i are the corresponding eigenvalues. Some of the properties rely on the fact that $\mathbf{X}^T\mathbf{X}$ is symmetric.

Using this aside, we can now use the eigendecomposition to obtain:

$$\begin{aligned}\text{bias}(\boldsymbol{\beta}^\star_{\text{RIDGE}}) &= \mathbb{E}\big[\boldsymbol{\beta}^\star_{\text{RIDGE}}\big] - \boldsymbol{\beta} \\ &= (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}(\mathbf{X}^T\mathbf{X})\boldsymbol{\beta} - \boldsymbol{\beta} \\ &= \mathbf{V}\big[(\mathbf{D} + \lambda\mathbf{I})^{-1}\mathbf{D} - \mathbf{I}\big]\mathbf{V}^T\boldsymbol{\beta}\end{aligned}$$

Using the fact that $\mathbf{D}$ is diagonal (and $\mathbf{I}$ of course, too), we can write:

$$\begin{aligned}\mathcal{D} &= (\mathbf{D} + \lambda\mathbf{I})^{-1}\mathbf{D} - \mathbf{I} \\ \mathcal{D}_{ii} &= \Big[\frac{d_i}{d_i + \lambda} - 1\Big] \\ &= -\frac{\lambda}{d_i + \lambda} \\ \Rightarrow \mathcal{D} &= -\lambda(\mathbf{D} + \lambda\mathbf{I})^{-1}\end{aligned}$$

Finally, we can write:

$$\begin{aligned}\text{bias}(\boldsymbol{\beta}^\star_{\text{RIDGE}}) &= -\lambda\mathbf{V}\big[(\mathbf{D} + \lambda\mathbf{I})^{-1}\big]\mathbf{V}^T\boldsymbol{\beta} \\ &= -\lambda(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\boldsymbol{\beta}\end{aligned}$$

As expected, as $\lambda \to 0$, we have that bias $\to 0$, because we recover the LS solution from before. Furthermore, as $\lambda \to \infty$, bias $\to -\boldsymbol{\beta}$ which corresponds to about 100% error.

Now that we have obtained an expression for the bias, we turn towards the variance:

$$\begin{aligned}\text{var}(\boldsymbol{\beta}^\star_{\text{RIDGE}}) &= \mathbb{E}\Big[\big(\boldsymbol{\beta}_{\text{RIDGE}} - \mathbb{E}[\boldsymbol{\beta}^\star_{\text{RIDGE}}]\big)^2\Big] \\ &= \mathbb{E}\Big[(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\underbrace{\boldsymbol{\varepsilon}\boldsymbol{\varepsilon}}_{\sigma^2\mathbf{I}}\mathbf{X}\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\Big] \\ &= \sigma^2(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}(\mathbf{X}^T\mathbf{X})\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\end{aligned}$$

Using the eigendecomposition of $\mathbf{X}^T\mathbf{X}$ from above:

$$= \sigma^2\mathbf{V}\Big[\underbrace{(\mathbf{D} + \lambda\mathbf{I})^{-1}\mathbf{D}(\mathbf{D} + \lambda\mathbf{I})^{-1}}_{\mathcal{P}}\Big]\mathbf{V}^T$$

By the diagonality of $\mathbf{D}$ and $\mathbf{I}$, we have:

$$\mathcal{P}_{ii} = \frac{d_i}{(d_i + \lambda)^2}$$

From this we can now draw the following conclusion: as $\lambda \to \infty$, $\mathcal{P}_{ii} \to 0$ (quadratically), thus reducing the variance.

In summary, increasing λ reduces the variance of the ridge estimator but increases its bias. Both trends can be visualised as in Figure 1.5.

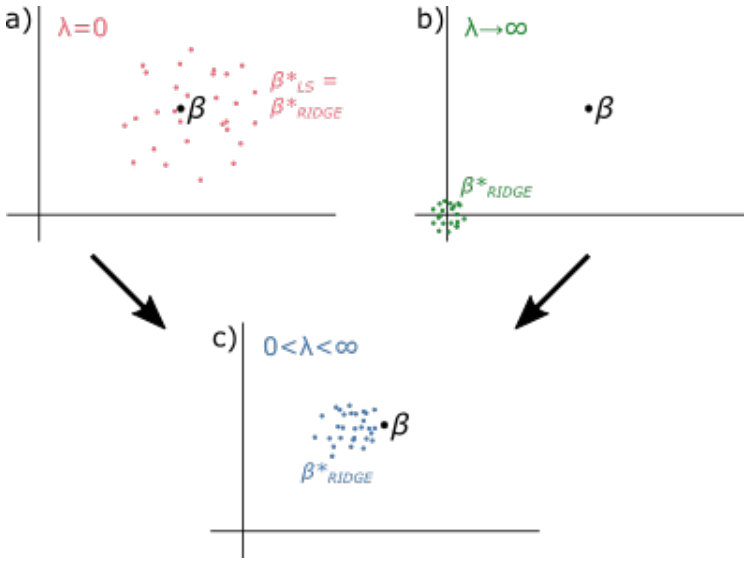

Figure 1.5. Bias and variance visualised for ridge regression. a) At $\lambda = 0$, we simply recover the least-squares estimate. b) As $\lambda \to \infty$, the bias tends towards $-\boldsymbol{\beta}$, resulting in the estimates being around 0. The variance decreases towards 0. c) As usual, the 'sweet spot' will be somewhere in-between.

6.1.2. *LASSO (Tibshirani).* In the LASSO (least absolute shrinkage and selection operator) method, as with the Ridge regression, we redefine the loss function to include a penalty term that tries to reduce the size of the parameter vector:

$$L_{\text{LASSO}}(\boldsymbol{\beta}) = \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|^2 + \lambda\|\boldsymbol{\beta}\|_1$$

Here $\|\boldsymbol{\beta}\|_1$ denotes the *Taxicab norm* (also known as the *Manhattan norm*):

$$\|\boldsymbol{\beta}\|_1 = \sum_{i=1}^{p} |\beta_i|$$

Note that i starts from one, i.e. the penalty does not apply to the intercept β_0. Compared to the 2-norm used in the penalty term of the ridge regression cost function, the 1-norm used in the LASSO penalty term is closer to 'direct selection of parameters' (which would correspond to the 0-pseudonorm) but it makes the optimisation harder (we will not be able to do it by hand); in other words, the 1-norm is a better relaxation of the 0-pseudonorm but more difficult to deal with mathematically.

Once again, we aim to minimise the corresponding loss function:

$$\min_{\boldsymbol{\beta}} L_{\text{LASSO}}(\boldsymbol{\beta}) \Leftrightarrow \min_{\boldsymbol{\beta}} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|^2 \text{ subject to } \|\boldsymbol{\beta}\|_1 \leq t.$$

The problem is convex also and strong duality applies here too. However, as opposed to ridge regression, there exists no analytical solution for LASSO. However, since the problem is convex, it is possible to apply convex optimisation techniques[1] to find the global optimum computationally, an example being gradient descent methods with the Huber regularisation to make the loss function smoother, as you will see in the coding notebook.

In Figure 1.6, we show a quick visual overview of both ridge and LASSO regressions and how the penalty functions modify the results. The important thing to notice is that LASSO tends to concentrate the solution towards the axes of the parameter space, i.e., it

[1]See again the book *Convex Optimisation* by Stephen Boyd if you want to know more.

makes many parameters have *small values*, so it induces a stronger sparsity than ridge. You can see this in Figure 1.6 by looking at where the optimal points are located in each case. From this we can get an intuition of why LASSO tends to give sparse solutions.

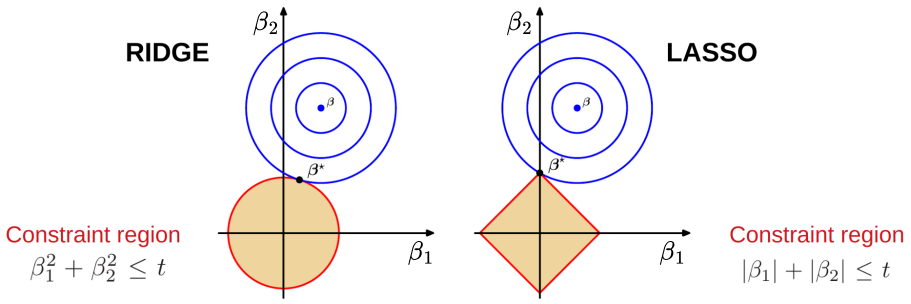

Figure 1.6. Figure showing the level curves of the unconstrained Least Squares solution (blue) and, in red, we the ridge constraint region (left) and the LASSO one (right). Figure adapted from Bishop, *Pattern Recognition and Machine Learning*, chapter 3.

6.2. Regularisation. The above optimisation approach indicates a general procedure to introduce penalty terms that induce sparsity. This penalty terms are also known as *regularisation terms* in optimisation, as they balance two conflicting terms of a cost function. In statistical and machine learning, ***regularisation* generally refers to terms that help fix the issue of overfitting by imposing a penalty on the parameters in the loss function, keeping in this way under control the parameter values learnt**.

Key message: General variations of shrinkage methods involve regularisation terms containing the l_q-norm:

$$L_q(\boldsymbol{\beta}) = \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|^2 + \lambda\|\boldsymbol{\beta}\|_q^q$$

where:

$$\|\boldsymbol{\beta}\|_q^q = \sum_{i=1}^{p} |\beta_i|^q$$

Ridge and LASSO are particular cases of this type of regularisation with respectively $q = 2$ and $q = 1$.

Figure 1.7 shows a visual overview of this approach. All of these different regularisation variants aim to control the model's sparsity. The figure summarises the overall trend in sparsity and the ease of optimisation. In general, the sparser we want things, the more difficult to optimise.

Note that $q = 0$ is the 'l_0'-pseudonorm case. In this case, we only have solutions where some of the parameters β_i are zero, which is equivalent to the selection of optimal subsets. This leads to truly sparse models (i.e., with zeros for many parameters); however it is difficult to optimise for sparsity in this $q = 0$ (combinatorial) case. The closer we get to this $q = 0$ case, the harder it is to optimise. The most important point is when we cross the boundary of $q = 1$ and enter the realm of 'non-convex' sets. At that point, convex optimisation is not applicable and we are not able to apply any of the powerful computational methods that rely on convexity.

To note, another direction also pursued in shrinkage methods is to combine norms in the penalty term. An example of this is the *elastic net* regularisation, which produces a

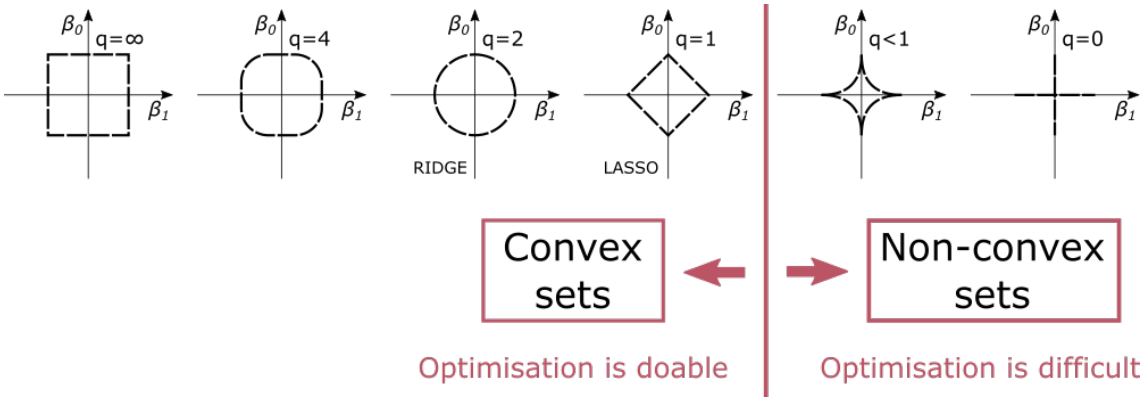

Figure 1.7. An overview of regularisation variants, i.e. $\lambda\|\boldsymbol{\beta}\|_q$ for a range of q.

convex combination of ridge and LASSO penalty terms:

$$L_{\text{EN}}(\boldsymbol{\beta}) = \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|^2 + \lambda\big[\alpha\|\boldsymbol{\beta}\|_1 + (1-\alpha)\|\boldsymbol{\beta}\|^2\big].$$

The elastic net is widely used in statistics to achieve sparsity.

6.3. Regularisation and overfitting. The regularisation parameter λ is an example of *hyperparameter*. We will see that more complex models, like neural networks, have several hyper-parameters. These parameters are called 'hyper' because typically they specify the overall structure of the model or the algorithm. These are parameters that are chosen and remain fixed during the optimisation that is done on the training set. For instance, in Linear Regression, when we optimise over β_i (the parameters of the model), we maintain fixed the value of λ (hyperparameter).

Hyperparameters are key to keeping under control over-fitting, the issue we mentioned at the beginning of this chapter. To avoid overfitting and enhance the robustness and generalisability of our models, it is customary to calibrate the influence of the hyperparameters on the results (a so-called 'hyperparameter search'), using a subset of the samples called the *validation set*. The model trained on the training set with a particular choice of hyperparameters is used to predict the outcomes on the validation set. This process is repeated with different values/choices of the hyperparameters. This scanning (or optimisation) over the hyperparameters fulfills a double function: finding the optimal combination of hyperparameters, and providing us with some reassurance that the model is robust and has the potential to generalise well. After this validation step, we will choose a model with an optimised set of hyperparameters with good performance and robustness. This model is then used on the *test data*, which has not been used *at all* in the optimisations of parameters or hyperparameters.

6.4. Practical tips. When then should we use ridge or Lasso shrinkage methods? A simple rule-of-thumb is the following:
1. If you don't have many data, always consider ridge regression to avoid overfitting. You can easily scan the model trained for different values of the penalties on a held-out validation set to check that you need ridge regularisation to control for overfitting.
2. If you want to achieve sparsity, use Lasso regression: it's an effective strategy to obtain more 'parsimonoius' models, i.e. models that explain the outcome by fewer predictors and are hence more interpretable.

Final aside: So far, all the methods discussed in this chapter have been linear. In mathematical terms, we have so far concerned ourselves with:

$$\hat{f}_{\text{lin}}(\mathbf{x}) = \mathbf{x}^T \boldsymbol{\beta}^{\star}_{\text{LS, Ridge, LASSO, EN, ...}}$$

However, introducing non-linearity is not that far from this framework. As we will see in the following chapters, many methods become nonlinear by extending this inference framework to models expressed in terms of nonlinear functions, i.e., models in the following non-linear form:

$$\hat{f}_{\text{nonlin}}(\mathbf{x}) = \mathbf{h}_m^T(\mathbf{x})\boldsymbol{\beta}^{\star}_{\text{nonlin}}$$

An example of such non-linear functions are polynomials of the inputs up to a certain degree:

$$\left\{h_{m,1}, \ldots, h_{m,T}\right\} \text{ from } \left(x_1^{d_1} \quad x_2^{d_2} \quad \ldots \quad x_p^{d_p}\right)$$

In the case of polynomials, one does have to choose how large the degrees should be, i.e. determine a D such that $\sum d_i < D$. Sparsity is desirable here since the number of polynomial terms grows combinatorially with the number of descriptors p and the largest degree D. Furthermore, polynomial models tend to overfit for high degrees. Hence sparsity is desirable here. These polynomial models are called Wiener-Volterra models in the applied math and signal processing literatures.

The collection of nonlinear functions $\left\{h_{m,1}, \ldots, h_{m,T}\right\}$ is usually called the *dictionary* in the Computer Science literature. Other examples of non-linear function dictionaries include: $\log(x_i), \sin(x_i), \cos(x_i)$, (or more generally $\sin(kx_i)$), or wavelets. The choice of dictionary is dictated by knowledge about the data based on modelling assumptions, usually. Additional properties, such as orthogonality and completeness, are also desirable. Fourier analysis and the whole of the 19th century orthogonal polynomial literature are precursors for this area in Machine Learning.