Bachelor Thesis Jurriaan Parie

# Statistical Learning: Methods for Regression

*Supervised by:*
Erwan Scornet
dr. Cristian Spitoni

Utrecht University

UPMC
SORBONNE UNIVERSITÉS

January 14, 2016

# Contents

**Preface** **2**

**Variable Types and Notation** **3**

**1 A Introduction on Supervised Learning** **4**
1.1 Statistical Decision Theory [1] . . . . . 4
1.1.1 A Statistical Model and Least Squares Estimation . . . 5
1.1.2 Supervised Learning . . . . . 7
1.2 Model Selection [1] [2] [3] [4] [10] . . . . . 7
1.2.1 Model Complexity . . . . . 8
1.2.2 Empirical Risk Minimization (ERM) . . . . . 10
1.2.3 Bias-Variance Tradeoff . . . . . 10
1.2.4 Bias-Variance Decomposition . . . . . 13
1.2.5 Model Assessment . . . . . 13
1.3 Linear Regression Models and Least Squares [1] [4] [10] . . . . . 15
1.3.1 Bias-Variance Decomposition . . . . . 16

**2 Towards Kernelised Regression Models** **18**
2.1 Linear Ridge Regression [1] [4] [5] [10] . . . . . 18
2.1.1 Geometric Interpretation . . . . . 19
2.1.2 Bias-Variance Decomposition . . . . . 20
2.1.3 Discussion . . . . . 21
2.2 Kernels [1] [3] [4] . . . . . 22
2.2.1 Reproducing Kernel Hilbert Spaces (RKHS) . . . . . 23
2.3 Kernelised Ridge Regression [1] [4] . . . . . 25
2.3.1 Examples of RKHS . . . . . 26
2.3.2 Kernel Selection . . . . . 28

**3 Support Vector Regression** **30**
3.1 Vapnik-Chervonenkis (VC) Dimension [1] [3] [6] . . . . . 30
3.1.1 Structural Risk Minimization (SRM) . . . . . 31
3.2 $\epsilon$-insensitive Loss Function [4] [9] . . . . . 32
3.3 Support Vector Regression [7] [8] [9] . . . . . 32
3.3.1 Support Vector Expansion . . . . . 34

**4 Performance of Regression Methods** **37**
4.1 Train Error, Test Error and Model Complexity [10] . . . . . 37
4.1.1 Polynomial Regression and the Degrees of Freedom . . 37
4.1.2 Ridge Regression and the Choice of $\delta$ . . . . . 39
4.2 Kernelised Regression . . . . . 41
4.2.1 Kernelised Ridge Regression and the Choice of the Kernel Function . . . . . 42
4.2.2 SVR and the Loss Function . . . . . 45
4.3 Linear Regression versus Support Vector Regression [11] . . . . 46
4.4 Kernelised Ridge Regression versus Support Vector Regression [11] . . . . . 47
4.5 Conclusions . . . . . 49

# Preface

One major goal of learning is prediction. In this report we discuss the performance of *kernelised regression* and *support vector regression* to model empirical data by *supervised learning*.

Supervised learning is the *machine learning* task of inferring a function from input data. Machine learning in this context is about learning a general rule that maps input data to their desired output values. These given input-output pairs are called the *training data*. A supervised learning model analyzes the training data and produces an inferred function, which can be used in a predictive way, to map new input data. A wide range of supervised learning models are available, each with its pros and cons. The most widely used learning algorithms are *linear regression*, *kernelised ridge regression* (KRR) and *support vector regression* (SVR). Linear regression is a commonly known but restricted technique for learning problems. The latter two techniques are non-parametric methods to find a non-linear relation between an *input* and an *output* variable and are therefore more convenient for learning problems.

This report is my attempt to bring together many of the important ideas of supervised learning and explain them in a statistical framework. Illustrative examples are included to strengthen the properties of the discussed techniques. Beside the theoretical framework, we consider two major challenges supervised learning models have to deal with. Finally, we present the performance of the chosen models to different sets of data.
We stress that there is no single learning model that works best on all the supervised learning problems. In other words, in this report we delve into the understanding of the *No Free Lunch Theorem*.

# Variable Types and Notations

For a *statistical learning* model there is a set of predefined variables that are denoted as *inputs*. These input variables have influence on one or more *output* values.

Typically, a predefined input variable will be denoted by the symbol $X$. If $X$ is a vector, elements out of $X$ will be represented by $X_i$. All vectors will be column vectors, a transposed vector will give us a row vector $X^T = (X_1 \ \dots \ X_p)$. *Observed* input values (not predefined) are written in lowercase, hence the $i$th observed value of $X$ is written as $x_i$.

Matrices are represented by bold uppercase letters. For example, a set of $N$ input-$p$-vectors $x_i$, $i = 1, \ \dots \ , \ N$, would be represented by the $N \times p$ matrix $\mathbf{X}$. In general, vectors will not be bold, except when they have $N$ components. This convention distinguishes a $p$-vector of inputs $x_i$ for the $i$th observation from the $N$ vector $\mathbf{x}_j$ containing all the observations on variable $X_j$. Since all the vectors are assumed to be column vectors, the $i$th row of $\mathbf{X}$ is $x_i^T$, the transposed vector of $x_i$.

For now, we can formulate the exercise of *supervised learning* as follows: given a *training set* of $p$ measurements $\mathcal{T} = \{(x_i, y_i)\}_{i=1}^{p}$, the goal is to find a prediction of the output $Y \in \mathcal{Y}$, denoted by $\hat{Y}$, using the input $X \in \mathcal{X}$. Where $\mathcal{X}$ is the input space and $\mathcal{Y}$ is the output space.

An *input vector* can be transformed into a *feature vector*, which contains a number of features that are descriptive of the input object. In this report, we will use these two terms interchangeably.

# 1 A Introduction on Supervised Learning

Learning is a widely used concept in statistics, in this chapter we present the theoretical framework of *supervised learning*. The learning problem consists of deriving a function that maps the input to the output in a predictive way, such that the learned function can be used to predict output from future input. In doing so, there are two major challenges to attack. Along with the theoretical framework of supervised learning, in this chapter we discuss these challenges supervised learning has to deal with to make good predictions.

## 1.1 Statistical Decision Theory [1]

In this section we present some basic statistical decision theory to expose the foundation of supervised learning models. We place ourselves in the world of random variables and probability spaces.

Let $X \in \mathbb{R}^p$ be a random input vector and $Y \in \mathbb{R}$ a real valued output variable, with joint probability distribution $\mathrm{P}(X, Y)$. In supervised learning we are looking for a function $f(X)$ to predict $Y$. This prediction requires a *loss function* $L(Y, f(X))$ for penalizing errors in the prediction. Typical choices are

$$L(Y, f(X)) = \begin{cases} (Y - f(X))^2 & \text{squared error} \\ |Y - f(X)| & \text{absolute error.} \end{cases} \tag{1.1}$$

This results in a criterion for choosing $f$; we want to minimize the *Expected Prediction Error* (EPE). This can be done using a *quadratic loss*,

$$\begin{aligned} \mathrm{EPE}(f) &= \mathbb{E}\big[(Y - f(X))^2\big] \\ &= \int \big(y - f(x)\big)^2 \, \mathrm{P}(dx, dy). \end{aligned}$$

By conditioning[1] on $X$ and by the *tower property of expectation*[2], we can write the EPE as

$$\mathrm{EPE}(f) = \mathbb{E}_X \mathbb{E}_{Y|X}\big[(Y - f(X))^2 |\ X\big].$$

[1] Conditioning means factoring the joint density $\mathrm{P}(X, Y) = \mathrm{P}(Y|X)\mathrm{P}(X)$, where $\mathrm{P}(Y|X) = P(Y, X)/\mathrm{P}(X)$, and splitting up the bivariate integral accordingly

[2] If $\mathcal{H} \subset \mathcal{F}$ then $\mathbb{E}\big[\mathbb{E}[X|\ \mathcal{F}]|\ \mathcal{H}\big] = \mathbb{E}[X|\ \mathcal{H}] = \mathbb{E}\big[\mathbb{E}[X|\ \mathcal{H}]|\ \mathcal{F}\big]$

Therefore we can minimize the EPE pointwise

$$f(x) = \underset{c}{\operatorname{argmin}}\, \mathbb{E}_{Y|X}\big[(Y-c)^2|\ X = x\big].$$

The solution to this is given by

$$f(x) = \mathbb{E}\big[Y|\ X = x\big]. \tag{1.2}$$

Thus the best prediction of $Y$ at any point $X = x$ is the conditional mean. This is also known as the *regression function*.

### 1.1.1 A Statistical Model and Least Squares Estimation

The goal of supervised learning is to find a suitable approximation $\hat{f}(x)$ for the relationship between the input and output denoted by $f(x)$.

Like we saw in the preceding section, the squared error loss leads to the regression function $f(x) = \mathbb{E}\big[Y|\ X = x\big]$. However, for most of the learning problems, the input-output pairs $(X, Y)$ will not be related directly to each other in the sense that $Y = f(X)$. There will be unmeasured variables that also contribute to $Y$. The *additive error model* assumes that we can put all known and unknown relationships together via the error $\epsilon$. We suppose that our data can be described by the following model

$$Y = f(X) + \epsilon, \tag{1.3}$$

with random noise $\epsilon$, where $\mathbb{E}[\epsilon] = 0$.

Having introduced the above model, it becomes straightforward to think about the well-known *least squares estimation* as a method for prediction. Given a vector of inputs $X^T = \begin{pmatrix} X_1 & \dots & X_p \end{pmatrix}$, we predict the value of $Y$ via the model

$$\hat{Y} = \hat{\alpha}_0 + \sum_{i=1}^{p} X_i \hat{\alpha}_i. \tag{1.4}$$

The term $\hat{\alpha}_0$ is the intercept term. It is convenient to include the coefficient $X_0 = 1$ in $X$. This allows us to write equation (1.4) as an inner product

$$\hat{Y} = \langle X, \hat{\alpha} \rangle = \begin{pmatrix} 1 & X_1 & \dots & X_p \end{pmatrix} \begin{pmatrix} \hat{\alpha}_0 \\ \hat{\alpha}_1 \\ \vdots \\ \hat{\alpha}_p \end{pmatrix} = X^T \hat{\alpha}, \tag{1.5}$$

with coefficient vector $\hat{\alpha} \in \mathbb{R}^{p+1}$. We are modeling a single output $\hat{Y} \in \mathbb{R}$. In general however, if $\hat{\alpha}$ would be a $(p+1) \times N$ coefficient matrix, $\hat{\mathbf{Y}}$ will be a $N$-vector. Then, in the $2(p+1)$-dimensional input-output space, $(X, \hat{Y})$ represents a hyperplane.

The *least squares estimation* (LSE) derives the coefficients $\alpha$ by minimizing the *Residual Sum of Squares*. Which is defined as follows,

$$\text{RSS}(\alpha) = \sum_{i=1}^{N} (y_i - x_i^T \alpha_i)^2.$$

$\text{RSS}(\alpha)$ is a quadratic function, hence a minimum exists but may not be unique. [1] The solution is easiest to represent in matrix notation. We can write

$$\hat{\alpha}_{LSE} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}, \tag{1.6}$$

where $\mathbf{X}$ is an $N \times p$ matrix with each row an input vector and $\mathbf{y}$ is an $N$-vector of the outputs of the training set. Assuming that $\mathbf{X}$ has full rank[3], this minimum will give us an estimation for the parameters $\alpha$.

Using these estimated coefficients for the linear model, for given input $x_i$, gives us the output $\hat{y}_i = \hat{y}_i(x_i) = x_i^T \hat{\alpha}$. The fitted function is defined by the $p$ coefficients of $\hat{\alpha}$.

While the least squares method is a convenient method for estimation, it is not the only criterion used. A more general approach for estimating the coefficients $\alpha$ is the *maximum likelihood estimation* (MLE).

Suppose we have an independent and identically distributed (i.i.d.) random variable $y_i$ for $i = 1, \ldots, N$ from a density $\text{P}(y_i)$. We can write the *log-likelihood* as follows

$$\ell(\alpha) = \sum_{i=1}^{N} \log \text{P}(y_i).$$

The idea of MLE is that the best values for $\alpha$ are those for which the probability of the observed $y_i$ are the largest. For the additive error model in

[3]We can state that $\mathbf{X}$ has full rank if and only if $\mathbf{X}^T \mathbf{X}$ is invertible

equation (1.3), under the assumption that $\epsilon \sim \mathcal{N}(0, \sigma^2)$, the MLE is the same as the LSE (1.6).
By introducing a linear model, we are now able to make predictions given the input data. The purpose of the next sections is to understand how we can learn successfully from predictions in general, independent of the chosen learning model.

### 1.1.2 Supervised Learning

Supervised learning attempts to learn $f$ by example through a *teacher*. The term *teacher* refers to the fact that in supervised learning the true outcome value is known such that we can compare the predicted outcome values to the true ones. The input values $x_i$ of the training data $\mathcal{T}$ for $i = 1, \ldots, p$ will be put into a learning model that produces outputs $\hat{f}(x_i)$. The learning algorithm has the property that it can modify its input/output relationship $\hat{f}$ in response to differences between $y_i - \hat{f}(x_i)$. This technique is called *learning by example*.

The models we mentioned in the preface are all used as learning models. We introduce *kernelised ridge regression* (KRR) and *support vector regression* (SVR) in Chapter 2 and Chapter 3. However, the differences between performance of these models rely on the statistical learning theory discussed in this chapter. In Chapter 4 we study the empirical results and performance of these models to real learning tasks.

## 1.2 Model Selection [1] [2] [3] [4] [10]

Recall out of section 1.1.1 the assumptions that the data is i.i.d. from an unknown underlying probability distribution $\mathrm{P}(X, Y)$. For a given learning problem, we have a variable $Y$, a random vector of inputs $X$ and a prediction function $\hat{f}(X)$ that has been estimated from a training set $\mathcal{T}$.

The *hypothesis space* $\mathcal{H} \subset \mathcal{F}$ is a space of functions $\hat{f}_{\mathcal{H}} : \mathcal{X} \to \mathcal{Y}$ of some underlying target space $\mathcal{F}$, which contains all possible functions from $\mathcal{X}$ to $\mathcal{Y}$. The hypothesis space is the space of functions the learning model will search through.

The loss function in equation (1.1) measures the error of a function on some

individual data point. The *risk* of a function $\hat{f}$ is the average loss over data points according to the underlying distribution P, defined as

$$R(\hat{f}) := \mathbb{E}\big[L(Y, \hat{f}(X))|\ \mathcal{T}\big]. \tag{1.7}$$

The risk of a function $\hat{f}$, as in equation (1.7), is the expected loss of the fit at all the data points $X \in \mathcal{X}$.

The learning model searches the best fit $\hat{f} \in \mathcal{H}$, based on the given training points. In theory we know exactly what the best fit is in $\mathcal{H}$, namely the one with the smallest risk. For simplicity we assume that it is unique and denote it as

$$\hat{f}_{\mathcal{H}} = \underset{\hat{f} \in \mathcal{H}}{\operatorname{argmin}}\ R(\hat{f}).$$

However, at this point, it is impossible to compute the risk of a fit $\hat{f}$ because we have no knowledge about the underlying density function P. This is where statistical learning theory provides a framework to analyze the situation and comes up with solutions.

### 1.2.1 Model Complexity

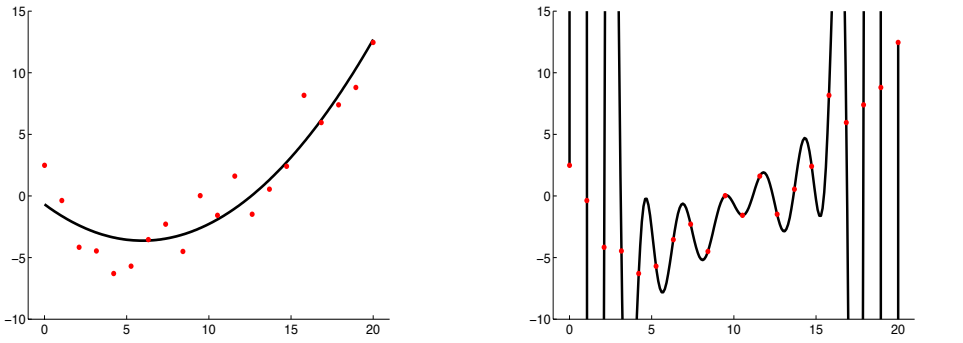
The performance of a learning model when making predictions for new test data is called *generalization*. [1] Assessment of this performance is important because it tells us how to choose our learning method.

Given the training data $\mathcal{T}$, let $\hat{f}$ be the fit our learning model came up with. We cannot compute the underlying risk $R(\hat{f})$, but we can count the number of mistakes the fit makes on the training points. The result is called the *empirical risk* or *training error*. Mathematically speaking,

$$R_{\text{emp}}(\hat{f}) := \frac{1}{N} \sum_{i=1}^{N} L(Y_i, \hat{f}(X_i)). \tag{1.8}$$

We say that a fit $\hat{f}$ *generalizes* well, if the difference $|R(\hat{f}) - R_{\text{emp}}(\hat{f})|$ is small. Note that with this definition, it just means that the empirical error $R_{\text{emp}}(\hat{f})$ is a good approximation of the true risk $R(\hat{f})$. It does not necessarily mean that $\hat{f}$ has a small overall empirical risk $R_{\text{emp}}(\hat{f})$.

Consider the following regression example. Given empirical data $\{(x_i, y_i)\}_{i=1}^{p}$, for simplicity we take $\mathcal{X} = \mathcal{Y} = \mathbb{R}$. Figure 1 shows a plot of such a dataset, along with two possible fits. Fit $\hat{f}_{(\mathrm{b})}$ represents a fairly complex model, but fits the training data perfectly with 0 training error.




(a) Degree 2 polynomial fit $\hat{f}_{(\mathrm{a})}$ (b) Degree 21 polynomial overfit $\hat{f}_{(\mathrm{b})}$

Figure 1: Polynomial fit to $N = 20$ points of data

On the other hand, we have $\hat{f}_{(\mathrm{a})}$, a quadratic line that does not perfectly fit the training data. There are some residual errors, resulting in a small positive training error, for example measured by a quadratic loss function.

What about the true risk $R(\hat{f}_{(\mathrm{a})})$ and $R(\hat{f}_{(\mathrm{b})})$? We know we cannot compute this risk from the training data. We can however say that $\hat{f}_{(\mathrm{a})}$ and $\hat{f}_{(\mathrm{b})}$ perform differently. For example, if $\hat{f}_{(\mathrm{a})}$ is the true underlying function, then $\hat{f}_{(\mathrm{b})}$ would have a huge true risk since the distance between the true and the fitted function is large, we call this *overfitting*. [1]

This example shows the crucial decision we have to make. Do we fit the training data to a complex function, resulting in a small or zero training error but bad generalization performance? Or do we fit the training data to a simple function at the cost of a slightly higher training error? In statistics this is known as the *bias-variance dilemma*.

### 1.2.2 Empirical Risk Minimization (ERM)

Recall that we assumed the data to be i.i.d. from an unknown underlying distribution $\mathrm{P}(X, Y)$. As we have seen in the previous section, the learning problem involves minimizing the risk introduced in equation (1.7).

The fact that we do not know the underlying distribution $\mathrm{P}(X, Y)$, makes it impossible to minimize this risk. But we do know the training data generated by the distribution. So to proceed, we can approximate the true risk by computing the *empirical risk* on the data and we will try to minimize the empirical risk instead of minimizing the true risk. We will derive a fit $\hat{f}$ as the function

$$\hat{f} := \operatorname*{argmin}_{f \in \mathcal{H}} R(f).$$

The ultimate goal of a learning model is to find this best fit in $\mathcal{H}$, for which the risk is minimal. This technique is known as the *empirical risk minimization principle* (ERM).

Note that the empirical risk (1.8) is defined as the average of the loss function $L(Y_i, \hat{f}(X_i))$ on specific training points and that the true risk (1.7) is the mean of this loss function over the whole distribution. Hence, from the law of large numbers we can conclude that for a fixed function $\hat{f}$, the empirical risk converges to the true risk as the number of training points goes to infinity,

$$R_{\text{emp}}(\hat{f}) = \sum_{i=1}^{N} L(Y_i, \hat{f}(X_i)) \to \mathbb{E}\big[L(Y, \hat{f}(X)\big] \qquad \text{for } N \to \infty.$$

For a given, finite sample this means that we can approximate the true risk (the one we are interested in) very well by the empirical risk (the one we can compute on the sample).

### 1.2.3 Bias-Variance Tradeoff

In section 1.2.1 we already pointed out the problem of model complexity: when is a model "simpler" than another one? We stated that the task of regression is to find a approximation $\hat{f}$ with an acceptable risk. To realize this, can we choose $\mathcal{H}$ equal to the space $\mathcal{F}$ of all functions and define

$\hat{f} := \underset{f \in \mathcal{F}}{\operatorname{argmin}} \, R_{\text{emp}}(f)$?
Unfortunately, the answer is no. In this section we will see that if we optimize over a too large function class $\mathcal{H}$, in particular if we make $\mathcal{H}$ so large that it contains all $\hat{f}$ for all different probability distributions P, this will not be able to learn successfully.

Identical to section 1.2.1, the *generalization error* or *test error* is the prediction error for new test data. It arises from two events:

- *Approximation Errors*, also called the *bias*. These errors are due to the fact that the hypothesis space is smaller than the target space. Hence the underlying function may lie outside the hypothesis space. A poor choice of the hypothesis space will result in high approximation errors and we be called a *model mismatch*.
- *Estimation Errors*, also called *variance*. These errors are, given a certain hypothesis space, the differences between the error of the value in the training data and the error of the best predictor. If the estimation error is large, then the hypothesis class is probably too large for the given amount of data.

Recall that $f$ is the true function, $\hat{f}$ is our approximation of $f$ and $\hat{f}_{\mathcal{H}}$ are the possible approximation functions which are in $\mathcal{H}$. In figure 2 we illustrate that the generalization error can be decomposed in the following way

$$R(\hat{f}) - R(f) = \underbrace{\left( R(\hat{f}) - R(\hat{f}_{\mathcal{H}}) \right)}_{\text{estimation error}} + \underbrace{\left( R(\hat{f}_{\mathcal{H}}) - R(f) \right)}_{\text{approximation error}}.$$

At this point, the size of the hypothesis space $\mathcal{H}$ is the mechanism to balance the trade-off between the estimation and approximation error, see figure 3. When we choose a very large hypothesis space $\mathcal{H}$, the approximation error will become small. However, the estimation error will be rather large because the hypothesis space $\mathcal{H}$ will contain complex functions which will lead to overfitting. When the hypothesis space $\mathcal{H}$ is small however, the opposite will happen, underfitting.

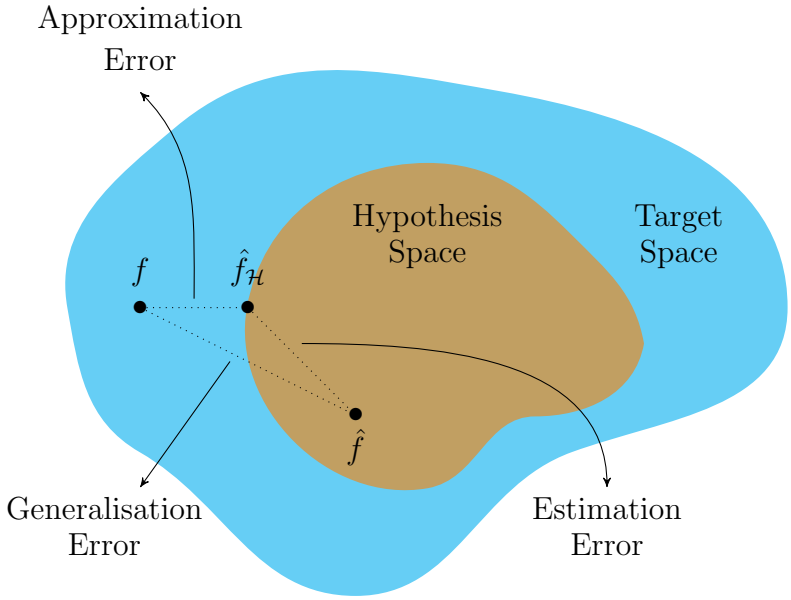



Figure 2: Illustration of errors in modelling

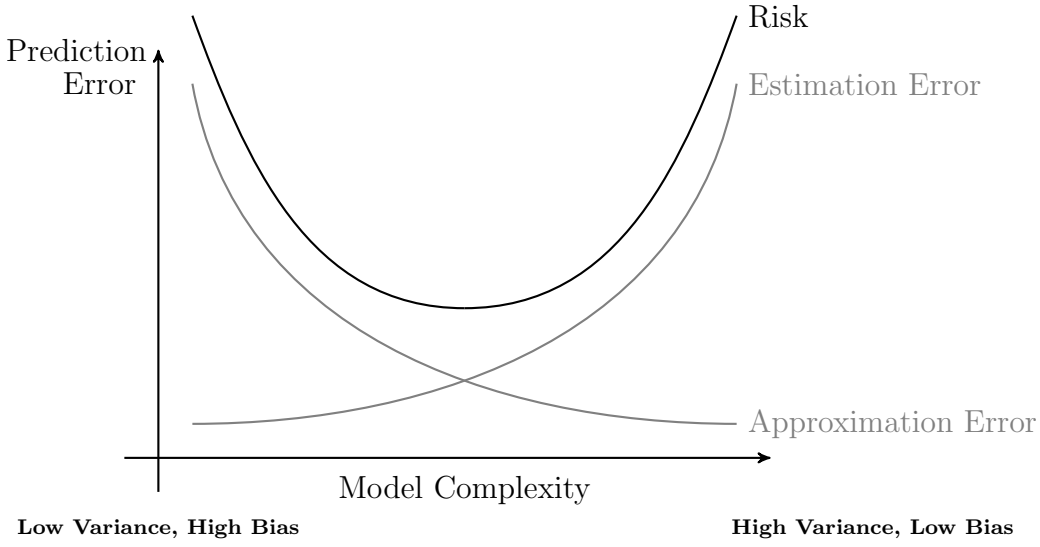



Figure 3: The trade-off between the estimation and the approximation error

### 1.2.4 Bias-Variance Decomposition

In section 1.1.1 we assumed that $Y = f(X) + \epsilon$ where $\epsilon \sim \mathcal{N}(0, \sigma^2)$. In the context of the bias-variance tradeoff, we can derive an expression for the expected prediction error (EPE) from section 1.1.1 for a regression fit $\hat{f}(X)$ for an input vector $X$, using the squared error loss

$$\begin{aligned}
\mathrm{EPE}(\hat{f}) &= \mathbb{E}[(Y - \hat{f}(X))^2 |\ X] \\
&= \sigma^2 + \big(\mathbb{E}[\hat{f}(X)] - f(X)\big)^2 + \mathbb{E}\big[\hat{f}(X) - \mathbb{E}[\hat{f}(X)]\big]^2 \\
&= \sigma^2 + \mathrm{Bias}^2(\hat{f}(X)) + \mathrm{Var}(\hat{f}(X)).
\end{aligned}$$

The first term, $\sigma^2$, is the *irreducible error* and cannot be avoided since it depends on the chosen model and not on the procedure.
However, we do have control over the second and the third term. The second term is the squared bias, the amount by which the average of our estimate differs from the true mean. The last term is the variance, the expected squared deviation of $\hat{f}(X)$ around its mean. In the last sections, we described the origin of the bias and variance of a model. KRR and SVR are both models that control these errors in a special way. We will elaborate the techniques these models use in the next chapters.

### 1.2.5 Model Assessment

When we have a diversity of models, (potentially) with different complexity, to choose from then how do we select the best fit for our data?

We do know that for the assessment of a chosen model, the most important is the generalization error. We have seen that this error can be approximated by computing this risk on a large *test set*, data that is not used for training the model. By assumption, when we train the model, we do not have access to the test set.

We can create a test set by partitioning the available data set into two disjoint parts: a *training set* and a *test* or *validation set*. The training set is used for training the model and the validation set is used in order to select the complexity of the model. We fit the different models to the training set and evaluate its performance on the validation set. Consequently, we choose

the model with the best performance.

A guideline could be to use 80% of the data to train the model and 20% is used for validating the model. If there is not enough data, we face difficulties to train our model and to make a trustworthy estimate of prediction performance. A criterion could be: reduce the number of variables such that there are less points of data than variables that has to be estimated. Since generally, we can not choose the number of data points.
An elegant solution could be *cross validation*. This method splits the training data into $K$-*folds*, then for each fold $k \in \{1, \ldots, K\}$ we train our model on all the folds except the $k$th, illustrated in figure 4.

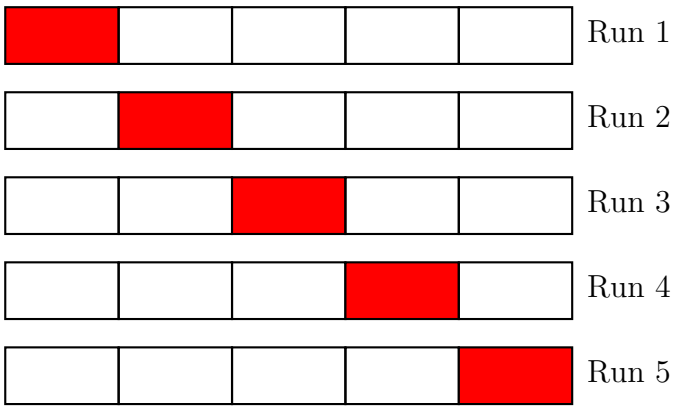



Figure 4: Scheme of 5-fold cross validation

Then we average the $(K-1)$ runs to estimate the generalization error, the average training error when we apply the function $\hat{f}(X)$ to an independent test sample from the joint distribution of $X$ and $Y$.
To conclude, estimation of test error for a particular training set is not easy in general, given just the data from that same training set. Instead, cross validation is a generally applicable way to predict the performance of a model on a validation set using computation instead of mathematical analysis.

## 1.3 Linear Regression Models and Least Squares [1] [4] [10]

We made the first steps towards a learning model in section 1.1.1 using a linear model to make predictions. A linear regression model assumes that the regression function from equation (1.2) is linear in the inputs $X_1, \ldots, X_p$. However, it is extremely unlikely that the true function $f(X)$ is really linear in $X$. Therefore, to obtain more advanced models, we replace the input vector $X$ with transformations of $X$ and then use linearity in a new space. This technique is known as *linear basis expansion*. Denote $\phi_i(X) : \mathbb{R}^p \to \mathbb{R}$ the $i$th transformation of $X$ for $i = 1, \ldots, p$. We then model

$$f(X) = \sum_{i=1}^{p} \phi_i(X)\alpha_i + \epsilon.$$

A $p^{\text{th}}$-polynomial can be fit in this way by choosing $\phi_i(x) = x^i$ for $i = 1, \ldots, p$ or for trigonometric functions we can choose $\phi_1(x) = \cos(\omega_1 x)$, $\phi_2(x) = \sin(\omega_1 x)$, $\phi_3(x) = \cos(\omega_2 x)$ and $\phi_4(x) = \sin(\omega_2 x)$. Figure 5 shows the illustration of a least squares fit to a 2-dimensional input vector.

Until now, we have made minimal assumptions about the distribution of the data. To study the characteristics of $\alpha$ we will assume that the $y_i$ are independent and have constant variance $\sigma^2$ for $i = 1, \ldots, N$. The covariance matrix of the LSE could be easily deduced from equation (1.6) and is given by

$$\text{Cov}(\hat{\alpha}|\ X) = (\mathbf{X}^T\mathbf{X})^{-1}\sigma^2.$$

To make deductions from these coefficients, extra assumptions are needed. We assume that the deviations of $Y$ around its expectation are additive and Gaussian. For that reason,

$$\begin{aligned} Y &= \mathbb{E}[Y|\ X_1, \ldots, X_p] + \epsilon \\ &= \alpha_0 + \sum_{i=1}^{p} X_i\alpha_i + \epsilon, \end{aligned} \tag{1.9}$$

where $\epsilon \sim \mathcal{N}(0, \sigma^2)$. From equation (1.9) we can easily derive that

$$\hat{\alpha} \sim \mathcal{N}\big(\alpha, (\mathbf{X}^T\mathbf{X})^{-1}\sigma^2\big),$$

which is a multivariate normal distribution. We can use these distributional properties to form confidence intervals for the parameters $\hat{\alpha}_i$.

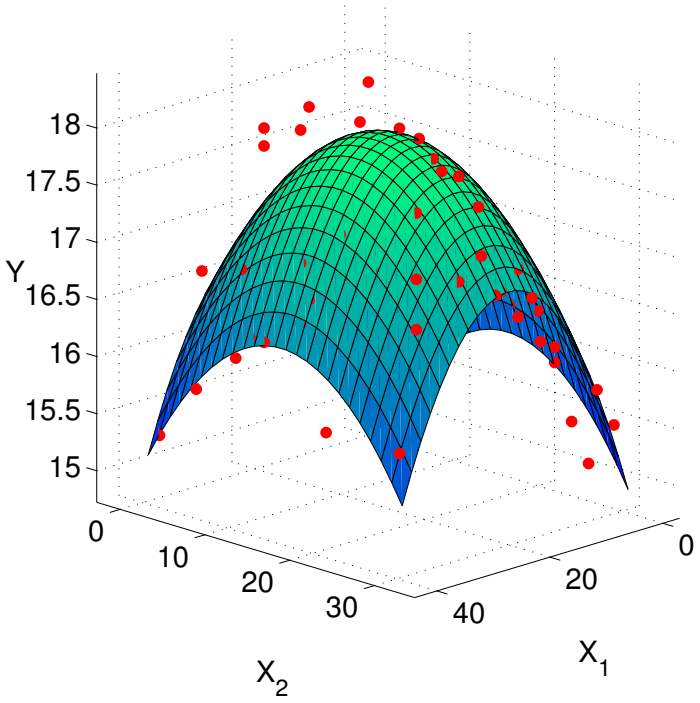



Figure 5: Least Square Estimation fitted with a quadratic

### 1.3.1 Bias-Variance Decomposition

To understand better the appearing errors using a linear model, we can do, similar to the bias-variance decomposition in section 1.2.4, such a decomposition for the expected prediction error (EPE) of a linear model $\hat{f}(X) = X^T\hat{\alpha}$

$$\begin{aligned}\mathrm{EPE}(\hat{f}) &= \mathbb{E}[(Y - \hat{f}(X))^2|\ X] \\ &= \sigma^2 + \big(f(X) - \mathbb{E}[\hat{f}(X)]\big)^2 + \|\boldsymbol{\phi}(X)^2\|\sigma^2. \qquad (1.10)\end{aligned}$$

Here, $\boldsymbol{\phi}(X) = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}X$ are the linear weights that produce the fit

$$\hat{f}(X) = X(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^TY.$$

So, $\mathrm{Var}(\hat{f}(X)) = \|\boldsymbol{\phi}(X)\|^2\sigma^2$ is dependent of $X$.

Once we found the bias-variance decomposition we can look to the generalization error of this model. The principle problem with cross validation is that it is slow, since we have to fit the model multiple times. This motivates to compute analytical bounds to the EPE. Since we have $N$ points of training data, we can take the average which results in $\frac{p}{N}\sigma^2$. In doing so,

$$\frac{1}{N}\sum_{i=1}^{N}\mathrm{EPE}(\hat{f}) = \sigma^2 + \frac{1}{N}\sum_{i=1}^{N}\big(f(x_i) - \mathbb{E}[\hat{f}(x_i)]\big)^2 + \frac{p}{N}\sigma^2.$$

Here, model complexity is directly related to the number of coefficients $p$. This sample error is an alternative for cross validation to achieve model comparison. We will discuss analytical bounds of the EPE more extensive in Chapter 2 and 3.

In this chapter we have been considering all the core statistical learning theory related to supervised learning. From now on we will work towards the introduction of the main learning models in this report, *kernelised ridge regression* and *support vector regression*.

# 2 Towards Kernelised Regression Models

In this section we will move beyond linearity and study more sophisticated models for regression and their performances for learning tasks.

A reason why we could not be convinced by the LSE is the accuracy of the prediction. Regularly, a linear regression model contains many correlated variables, i.e. coefficients become poorly determined and produce models with high variance and low bias. To alleviate this, we could rise the predicting accuracy by shrinking or setting some coefficients of $\alpha$ to zero. In doing this, we give up some amount of bias to reduce the variance of the prediction and may improve the overall predicting accuracy.

In this chapter we discuss different methods to control the bias and variance. In linear models this is known as *regularization*. Later on we introduce non-parametric models to find a non-linear relationship between the input and output data.

## 2.1 Linear Ridge Regression [1] [4] [5] [10]

Ridge regression is a simple example of a *regularization method*. Ridge regression shrinks the coefficients of $\alpha$ by introducing a penalty on the absolute value of each element of $\alpha$. The ridge coefficients minimize a penalized residual sum of squares

$$\hat{\alpha}_{ridge} = \underset{\alpha}{\operatorname{argmin}} \left\{ \sum_{i=1}^{N} (y_i - \alpha_0 - \sum_{j=1}^{p} x_{ij}\alpha_j)^2 + \delta \sum_{j=1}^{p} \alpha_j^2 \right\}, \tag{2.1}$$

where $x_{ij}$ denotes an element of the matrix $\mathbf{X}$. In matrix notation,

$$\mathrm{RSS}(\delta) = (\mathbf{y} - \mathbf{X}\alpha)^T(\mathbf{y} - \mathbf{X}\alpha) + \delta\alpha^T\alpha. \tag{2.2}$$

Here $\delta > 0$ is a parameter that stands for complexity. In this way we can control the amount of shrinkage, if $\delta$ is large, the amount of shrinkage is large.

Notice that the intercept $\alpha_0$ has been left out of the penalty term. It can be shown that adding a constant $c$ to each $y_i$ would not results (as expected) in a shift of the predictions by the same amount $c$. Therefore we estimate $\alpha_0$ by $\bar{y} = \frac{1}{N}\sum_{i=1}^{N} y_i$. The other coefficients will be estimated by ridge regression

without intercept. [1]

It can be shown that the solution for equation (2.1) and (2.2) is of the form

$$\hat{\alpha}_{ridge} = \left(\mathbf{X}^T\mathbf{X} + \delta\mathbf{I}\right)^{-1}\mathbf{X}^T\mathbf{y}.$$

We have to pay attention to the choice of the quadratic penalty $\alpha^T\alpha$. As a consequence of this choice, the ridge regression solution is again a linear function of $\mathbf{y}$. The solution adds positive elements to the diagonal of matrix $\mathbf{X}^T\mathbf{X}$ before inversion. This makes the problem nonsingular, even if the system of equations is poorly conditioned, i.e. when $\mathbf{X}^T\mathbf{X}$ does not have full rank, or when the eigenvectors are small.

### 2.1.1 Geometric Interpretation

An illustration will help us to better understand the technique of ridge regression. Figure 6 depicts the ridge regression method when there are only two coefficients.

$$\mathrm{RSS}(\delta) = \underbrace{(\mathbf{y} - \mathbf{X}\alpha)^T(\mathbf{y} - \mathbf{X}\alpha)}_{①} + \underbrace{\delta\alpha^T\alpha}_{②}$$

Component ① has elliptical contours, centered at the LSE/MLE. For different fixed heights $c$ component ② will represent the circle $\alpha_1^2 + \alpha_2^2 = c_1$, where $c_1 = \frac{c}{\delta}$. For all values of $\delta$ the range of solutions for minimizing $\mathrm{RSS}(\delta)$ will be an intersection of the level curve of the circle and a level curve of the ellipse.

We can do some reasoning for this, when we are moving out of a black node, an intersection point, along a blue level curve our likelihood will not change because we are at the same height but we have moved out of the red circle, so we are doing worse. And vice versa for following the red curve and moving away of the blue one.
Bottom line, the points of intersection between the level curves will be our solutions.

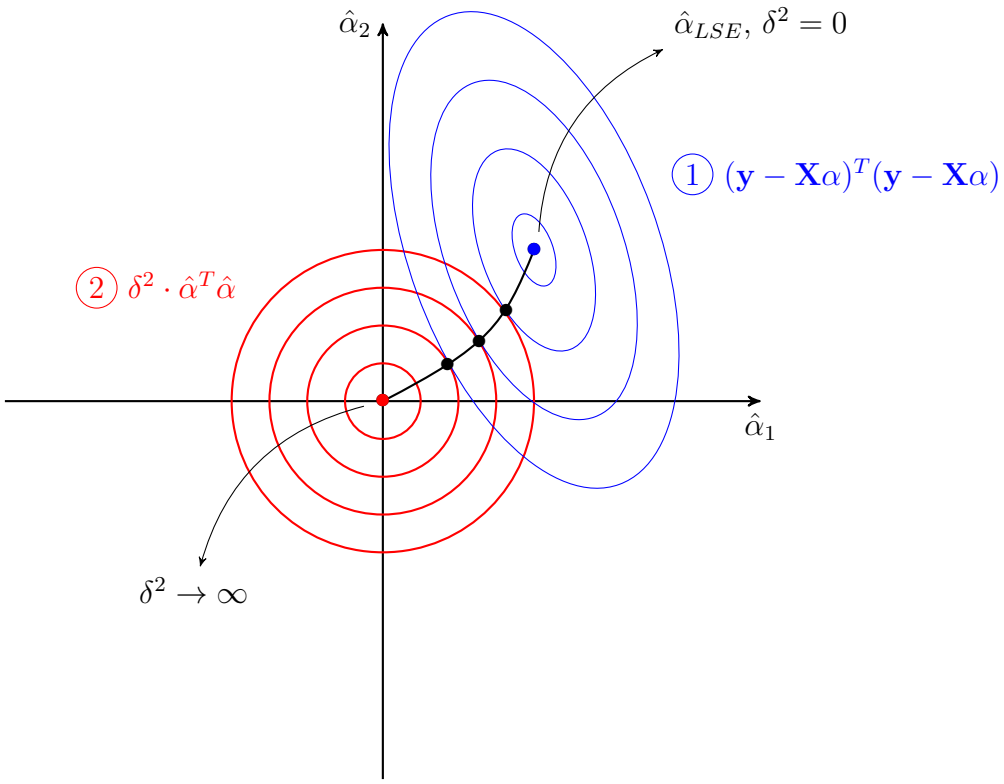



Figure 6: Geometric interpretation of ridge regression

### 2.1.2 Bias-Variance Decomposition

We can apply *bias-variance decomposition* to ridge regression like we did in section 1.3.1 to linear regression. The expected prediction error (EPE) for a ridge regression fit $\hat{f}(X) = X^T\hat{\alpha}$ is similar to equation 1.10, only the linear weights differs. For an input vector $X$,

$$\boldsymbol{\phi}(X) = \mathbf{X}(\mathbf{X}^T\mathbf{X} + \delta^2\mathbf{I})^{-1}X.$$

For ridge regression we can formulate the bias term more elegantly. Let $\alpha_*$ denote the coefficients for the optimal estimation

$$\alpha_* = \underset{\beta}{\operatorname{argmin}}\ \mathbb{E}\big[(f(X) - X^T\alpha)^2\big].$$

We can write the average squared bias for $X$ as

$$\mathbb{E}\big[f(X) - \mathbb{E}[\hat{f}(X)]\big]^2 = \mathbb{E}\big[f(X) - X^T\alpha_*\big]^2 + \mathbb{E}\big[X^T\alpha_* - \mathbb{E}[X^T\hat{\alpha}]\big]. \quad (2.3)$$

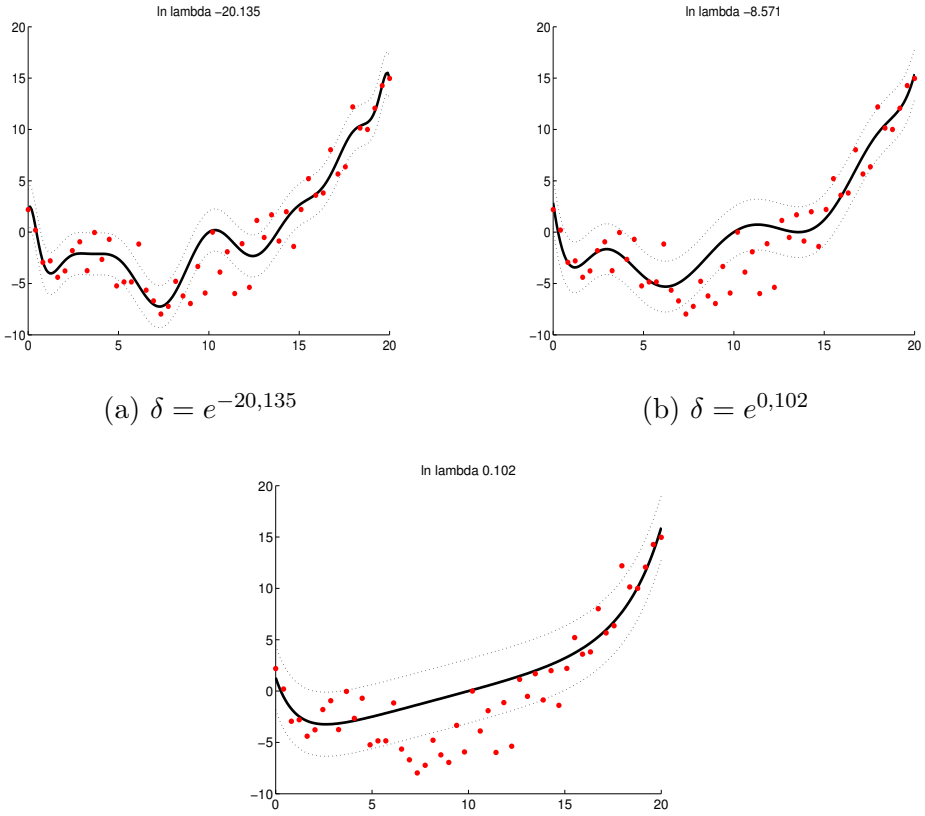
The first term at the right hand side is the average squared *model bias*, the difference between the best fit and the true function. The second term at the right hand side is the average squared *estimation bias*, the difference between the average estimate $\mathbb{E}[X^T\hat{\alpha}]$ and the best fitting approximation. So we can write equation (2.3) as

$$= \text{Average}\big(\text{Model Bias}\big)^2 + \text{Average}\big(\text{Estimation Bias}\big)^2.$$

For linear models fit by ordinary least squares, the estimation bias is zero. For restricted fits, such as ridge regression, it is positive, and we trade it off with the benefits of a reduced variance. The model bias can only be reduced by enlarging the class of linear models to a richer collection of models, by including interactions and transformations of the variables in the model.

### 2.1.3 Discussion

In figure 8 we see a polynomial fit of degree 14 using ridge regression. The data, with added noise, is originally distributed as a polynomial of degree 2. We see in figure 8(a) that when $\delta$ is too small the polynomial will still wiggle a lot and results in overfitting. When we increase $\delta$, the coefficients become smaller and henceforth, this results in a smoother fit, see figure 8(b). For a good choice of $\delta$ the polynomial wiggles less and is not that smooth, see figure 8(c). We present more analytical results of experiments with ridge regression in Chapter 4.
Other shrinkage models could be derived by replacing the so-called $\ell_2$ ridge penalty term $\sum_{i=1}^{p} \alpha_i^2$ in equation (2.1) by other penalty terms, for example the $\ell_1$ lasso penalty $\sum_{i=1}^{p} |\alpha_i|$. However, the lasso method makes the solutions nonlinear in the $y_i$'s and there is no closed form expression as in ridge regression. Each of these techniques corresponds to a different form of *regularization*, used to prevent overfitting.




(a) $\delta = e^{-20,135}$

(b) $\delta = e^{0,102}$

(c) $\delta = e^{-8,751}$

Figure 8: The error bars (dotted), representing $\sigma^2 = 2$, get wider as the fit gets smoother

## 2.2 Kernels [1] [3] [4]

In this section we place regression into the larger context of regularization methods. We can formulate the *regularization problem* introduced in section 2.1 in general as

$$\min_{f \in \mathcal{H}} \left[ \sum_{i=1}^{N} L(y_i, \hat{f}(x_i)) + \delta J(\hat{f}) \right], \tag{2.4}$$

where $L(y_i, \hat{f}(x_i))$ is a *loss function* and $J(\hat{f})$ is a *penalty function*.

The remarkable feature of the solution for this regularization problem is that while equation (2.4) is defined over an infinite dimensional space, the solution is finite dimensional. We elaborate this in the next section.

### 2.2.1 Reproducing Kernel Hilbert Spaces (RKHS)

An important subclass of problems of the form in equation (2.4) is generated by a *positive definite kernel* $\kappa(x, y)$ and the corresponding space of functions $\mathcal{H}_\kappa$ is called a *reproducing kernel Hilbert space* (RKHS). The penalty function $J$ could also be defined in terms of the kernel. Now, we will give a short introduction to this class of models.

Let $x, y \in \mathbb{R}^p$. We consider the space of functions generated by the linear span of $\{\kappa(\cdot, y),\ y \in \mathbb{R}^p\}$, i.e. arbitrary linear combinations of the form $f(x) = \sum_i \alpha_i \kappa(x, y_i)$, where each kernel term is viewed as a function of the first argument and indexed by the second.

In the scope of this report we focus on so-called *Mercer kernels* or positive definite kernels. The importance of Mercer kernels is *Mercer's theorem*. Heretofore, we have to introduce the *Gram matrix*, defined by

$$\mathbf{K} = \begin{bmatrix} \kappa(x_1, y_1) & \dots & \kappa(x_1, y_N) \\ & \vdots & \\ \kappa(x_N, y_1) & \dots & \kappa(x_N, y_N) \end{bmatrix}.$$

**Mercer's Theorem.** *If the Gram matrix is positive definite (i.e.* $x^T \mathbf{K} x \geq 0$*), we can compute an eigenvector decomposition of the matrix* $\mathbf{K}$ *such that there exists a function* $\phi$*, mapping* $x \in \mathbb{R}^p$ *to* $\mathcal{F}$ *such that*

$$\kappa(x, y) = \sum_{i=1}^{\infty} \gamma_i \phi_i(x) \phi_i(y), \tag{2.5}$$

*where* $\phi$ *depends on the eigen functions of* $\kappa$, with $\gamma_i > 0$ and $\sum_{i=1}^{\infty} \gamma_i < \infty$.

Elements of $\mathcal{H}_\kappa$ can be written in terms of these eigenfunctions

$$f(x) = \sum_{i=1}^{\infty} c_i \phi_i(x),$$

under the restriction that

$$\|f\|_{\mathcal{H}_\kappa}^2 \stackrel{\text{def}}{=} \sum_{i=1}^{\infty} \frac{c_i^2}{\gamma_i} < \infty,$$

where $\|f\|_{\mathcal{H}_\kappa}$ is the norm induced by $\kappa$. We state that $J(f) = \|f\|_{\mathcal{H}_\kappa}$ and can be clarified as a generalized ridge penalty, where functions with large eigenvalues in equation (2.5) get penalized less and vice versa.

Rewriting equation (2.4) gives

$$\min_{f \in \mathcal{H}_\kappa} \left[ \sum_{i=1}^{N} L(y_i, f(x_i)) + \delta \|f\|_{\mathcal{H}_\kappa} \right]. \tag{2.6}$$

It can be shown that the solution to equation (2.6) is finite dimensional and has the form,

$$f(x) = \sum_{i=1}^{N} \alpha_i \kappa(x, y_i).$$

For $f \in \mathcal{H}_\kappa$ holds that $\langle \kappa(\cdot, x_i), f \rangle_{\mathcal{H}_\kappa} = f(x_i)$. Likewise, $\langle \kappa(\cdot, x_i), \kappa(\cdot, x_j) \rangle_{\mathcal{H}_\kappa} = \kappa(x_i, x_j)$ (the *reproducing* property of $\mathcal{H}_\kappa$). For that reason, we can formulate

$$J(f) = \sum_{i=1}^{N} \sum_{j=1}^{N} \kappa(x_i, x_j) \alpha_i \alpha_j,$$

for $f(x) = \sum_{i=1}^{N} \alpha_i \kappa(x, x_i)$. Utilizing the above results, we can reduce equation (2.6) to the finite dimensional expression

$$\min_{\boldsymbol{\alpha}} \left( L(\mathbf{y}, \mathbf{K}\boldsymbol{\alpha}) + \delta \boldsymbol{\alpha}^T \mathbf{K} \boldsymbol{\alpha} \right). \tag{2.7}$$

The characteristic that the infinite dimensional problem in equation (2.6) can be reduced to a finite dimensional problem is called the *kernel property* or the *kernel trick*. So the idea of the kernel function is to enable operations to be performed in the input space rather than the potentially high dimensional feature space. Hence the inner product does not need to be evaluated in the feature space. This provides a way to deal with the curse of dimensionality.

## 2.3 Kernelised Ridge Regression [1] [4]

Using the kernel property for regression models, regularization problems now depend on the choice of the kernel $\kappa$ and the loss function $L$. If we consider to use the squared-error loss, the solution to the regularization problem of equation (2.6) can be written as an infinite dimensional problem,

$$\min_{\{c_j\}_1^\infty} \left[ \sum_{i=1}^{N} \left( y_i - \sum_{j=1}^{N} c_j \phi_j(x_j) \right)^2 + \delta \sum_{j=1}^{\infty} \frac{c_j^2}{\gamma_j} \right]. \tag{2.8}$$

To be consistent, the above minimization problem can be written in matrix notation as

$$\min_{\boldsymbol{\alpha}} \left( (\mathbf{y} - \mathbf{K}\boldsymbol{\alpha})^T (\mathbf{y} - \mathbf{K}\boldsymbol{\alpha}) + \delta \boldsymbol{\alpha}^T \mathbf{K} \boldsymbol{\alpha} \right). \tag{2.9}$$

It can be shown that the solution for $\boldsymbol{\alpha}$ is given by

$$\hat{\boldsymbol{\alpha}} = (\mathbf{K} + \delta \mathbf{I})^{-1} \mathbf{y}$$

and that

$$\hat{f}(x) = \sum_{i=1}^{N} \hat{\alpha}_i \kappa(x, x_i).$$

Therefore, the vector of $N$ fitted values is given by

$$\begin{aligned} \hat{\mathbf{f}} &= \mathbf{K}\hat{\boldsymbol{\alpha}} \\ &= \mathbf{K}(\mathbf{K} + \delta \mathbf{I})^{-1} \mathbf{y}. \end{aligned} \tag{2.10}$$

Now, we have kernelised the ridge regression problem. That a minimizer $\hat{f}$ of a regularized empirical risk function defined over a RKHS can be represented as a finite linear combination of kernel products evaluated on the input points is known as the *representer theorem*.

However, the computations still depend on the amount of training data $N$. So to provide a good fit for a high dimensional problem will generally require a large training set and potentially high computing time. We will discuss this problem more extendedly in the next chapter.

### 2.3.1 Examples of RKHS

With the introduction of kernels in the previous section, the obvious question that arises, which kernel function do we choose? We present two popular examples of kernel functions that satisfy Mercer's conditions down below.

#### Polynomial Kernel

The kernel $\kappa(x, y) = \big(\langle x, y\rangle + 1\big)^d$ for $x, y \in \mathbb{R}^p$ has $M = \binom{p+d}{d}$ eigen-functions that span the space of polynomials in $\mathbb{R}^p$ of total degree $d$. For example, with $p = 2$ and $d = 2$, $M = 6$ and we find that

$$\kappa(x, y) = 1 + 2x_1y_1 + 2x_2y_2 + 2x_1x_2y_1y_2 + x_1^2{y_1}^2 + x_2^2{y_2}^2.$$

As a consequence of Mercer's Theorem, the kernel function can be written as $\phi(x)^T\phi(y)$ where $\phi : \mathbb{R}^2 \to \mathbb{R}^6$

$$\phi(x)^T = \begin{pmatrix} 1 & \sqrt{2}x_1 & \sqrt{2}x_2 & x_1^2 & x_2^2 & \sqrt{2}x_1x_2 \end{pmatrix}. \tag{2.11}$$

So using this kernel is equivalent to working in a 6 dimensional feature space.

Note that while this approach seems reasonable in the particular example above, it can easily become computationally infeasible for polynomials of higher order, caused by the combinatorial blow up of $\binom{p+d}{d}$.

#### Radial Basis Functions (RBF) Kernel

The polynomial kernel is an excellent example of computing a high dimensional inner products in a low dimensional input space. However, the simplicity of this kernel method is not without any implications. Each of the polynomials $\phi_i$ in (2.11) contains a scaling factor depending on the form of $\kappa$, which influences the imposed penalty term.

The Gaussian kernel $\kappa(x, y) = \exp(-\frac{1}{2\sigma^2}\|x-y\|^2)$ leads to a regression model that is the expansion of *Gaussian radial basis functions* (RBF) kernel,

$$k_i(x) = \kappa(x, x_i) = \exp\Big(-\frac{1}{2\sigma^2}\|x - x_i\|^2\Big), \tag{2.12}$$

each function centered at one of the training feature vectors $x_i$ for $i = 1, \ldots, N$ and given $\sigma^2$.

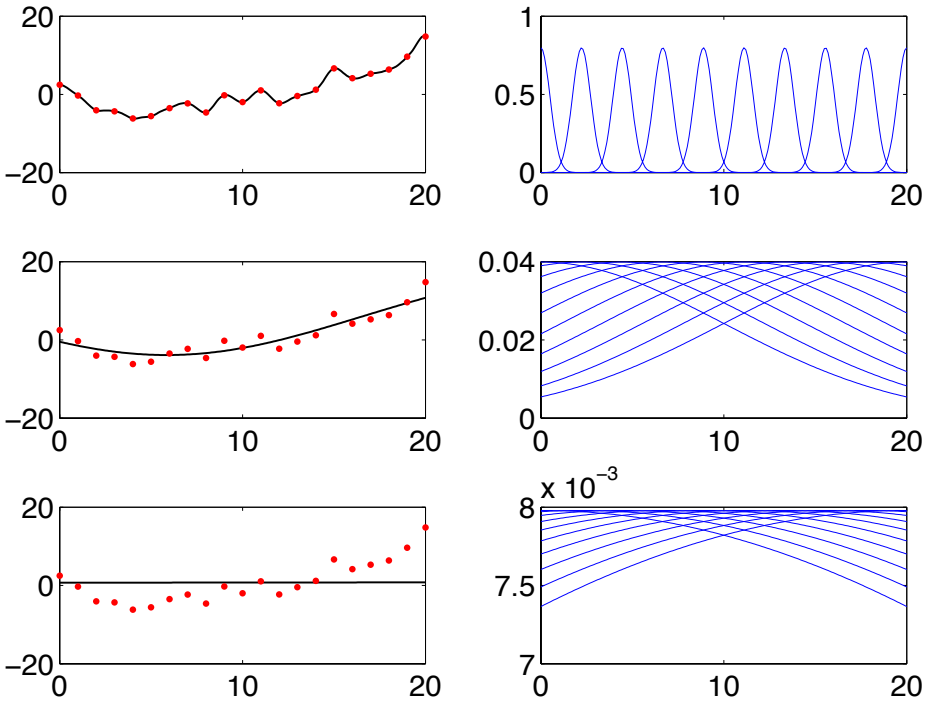



Figure 9: Left column: fitted function. Right column: uniformly spaced RBF basis functions for $\sigma^2 = 0.1$, $\sigma^2 = 0.5$ and $\sigma^2 = 50$.

For example, we can use this kernel functions $k_i$ in a linear regression mode, $\hat{f}(x) = \hat{\alpha}_0 + \sum_{i=1}^{N} k_i(x)\hat{\alpha}_i$, to fit a function to data. Where we use equation (2.9) to estimate the coefficients $\hat{\alpha}$.

Figure 9 shows a RBF kernel for different values of $\sigma^2$. Comparable with the performance of ridge regression in section 2.1.3, if we choose $\sigma^2$ too small, this will result in a very wiggly function. If we choose $\sigma^2$ too big, the fitted function is just a straight line since each datapoint is equally close to every basis function.

In contrary to the polynomial kernel, the Gaussian kernel is an inner product in some infinitely-dimensional space. This becomes clear when we look at

the Taylor-expansion of equation (2.12)

$$\exp\Big(-\frac{\|x-x_i\|^2}{2\sigma^2}\Big) = \exp\Big(-\frac{\|x\|^2}{2\sigma^2}\Big)\cdot\exp\Big(-\frac{\|x_i\|^2}{2\sigma^2}\Big)\cdot\sum_{j=0}^{\infty}\frac{(x^Tx_i)^j}{j!}$$

$$= \sum_{j=0}^{\infty}\Big\langle\sqrt[j]{\frac{\exp\big(-\frac{\|x\|^2}{2\sigma^2}\big)}{j!}}x,\ \sqrt[j]{\frac{\exp\big(-\frac{\|x_i\|^2}{2\sigma^2}\big)}{j!}}x_i\Big\rangle^j.$$

Since the inner product has the dimensionality of the range of the summation, this is an inner product in some infinitely-dimensional space.

### 2.3.2 Kernel Selection

With the introduction of different kernels, the obvious question is, which is the best kernel for a particular problem? Kernel methods achieve flexibility by fitting models to a target point $x_i$ and by varying the width of the kernel, e.g. $\sigma^2$ for a RBF kernel. To study this question we will work towards the theoretical framework of *support vector machine* (SVM) that will be introduced in the next chapter.

Recall the *bias-variance tradeoff* out of section (1.2.3). Among most of the models, it turns out that the training error $R_{\text{emp}}$ will be effectively estimated less than the true risk $R$. A fitting method typically accustoms to the training data and subsequently the training error will be an optimistic estimate of the generalization error.

The origin of this *optimism* in $R_{\text{emp}}$ is easiest to understand when we focus on the so-called *in-sample* error. Let $Y^0$ be $N$ new response values at each of the training points $x_i$, $i = 1, \ldots, N$. We define the *in-sample* error as

$$R_{\text{in}}(\hat{f}) = \frac{1}{N}\sum_{i=1}^{N}\mathbb{E}\big[L\big(Y_i^0, \hat{f}(x_i)\big)\big|\ \mathcal{T}\big]. \tag{2.13}$$

We define the optimism as the difference between $R_{\text{in}}$ and the training error $R_{\text{emp}}$

$$\text{op} \equiv R_{\text{in}} - R_{\text{emp}}.$$

Note that this is typically positive since $R_{\text{emp}}$ is a too optimistic estimate. The optimism increases linearly with the complexity (i.e. number $d$ of basis functions) we use for $\hat{f}$, but decreases as the training sample size increases. Now, a convenient way to estimate the prediction error is to estimate the optimism and then add it to the training error $R_{\text{emp}}$.

The in-sample error is not of direct interest since it is unlikely that test and training input vectors coincide. But for comparison between models, the in-sample error is effective and often leads to a convenient model selection.

# 3 Support Vector Regression

Consider the $\ell_2$ regularized empirical risk function as we did in section 2.2

$$\min_{f \in \mathcal{H}} \left[ \sum_{i=1}^{N} L(y_i, f(x_i)) + \delta \|\alpha\|^2 \right]. \tag{3.1}$$

If $L$ is a quadratic loss, equation 3.1 represents ridge regression. We saw in section 2.3 how we can obtain kernelised ridge regression.

Kernelised ridge regression relies on the kernel function $\kappa(x_n, x_m)$ where every pair of training data $x_n$ and $x_m$ must be evaluated. This could take excessive computation time when we apply kernelised ridge regression to a large training set. A solution for this problem is using *sparse kernel machines*.

To make a prediction by a sparse kernel machine, only a subset of the training data points will be evaluated by the kernel function, known as *support vectors*. To ensure that the solution will be *sparse* we will introduce in this chapter a new loss function.

The combination of the kernel trick with the modified loss function is known as *Support Vector Machines* (SVM). First, we have to prepare the framework of statistical learning theory for SVMs.

## 3.1 Vapnik-Chervonenkis (VC) Dimension [1] [3] [6]

A difficulty in estimating the in-sample error from equation (2.13) is the need to specify the complexity of the fit. In general, it is difficult to pin down the effective number of parameters for a fitted model. The *Vapnik-Chervonenkis* dimension provides such a general measure of complexity.

Suppose we have a set of functions $\{f(x, \alpha)\}$ with coefficient vector $\alpha$ and $x \in \mathbb{R}^p$. The VC dimension is a way of measuring the complexity of a class of functions by determining how wiggly the contained function can be.

**Definiton** *The VC dimension of the set of functions $\{f(x, \alpha)\}$ is $p$ if and only if there exists a set of points $\{x_i\}_{i=1}^{p}$ such that these points can be separated in all $2^p$ possible configurations, and that no set $\{x_i\}_{i=1}^{q}$ exists, where*

*$q > p$, is satisfying this property.*

The left panel of figure 10 shows that the class of linear lines in the plane can shatter three points. The right panel shows that the class of linear lines cannot shatter four points. Hence the VC dimension of the class of linear lines in a plane is three. The class of nonlinear curves could shatter four points, so this class has VC dimension greater than four. E.g. the function $a \cdot \sin(bx)$ has infinite VC dimension.

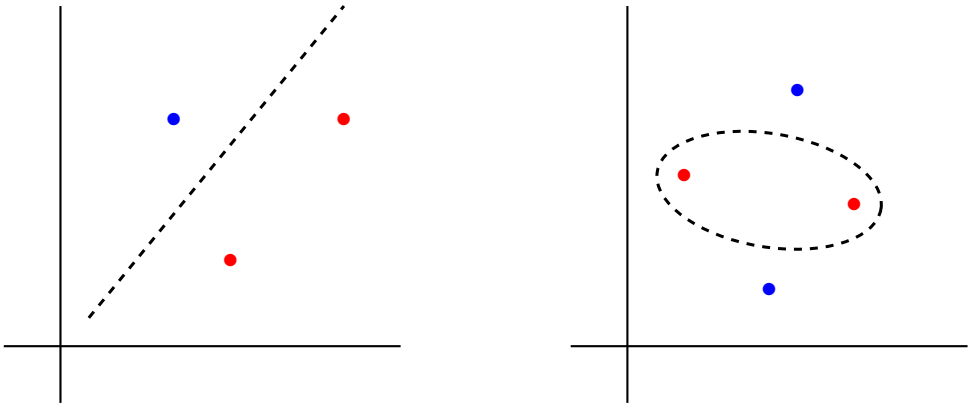


Figure 10: Illustration of VC dimension

### 3.1.1 Structural Risk Minimization (SRM)

One can use the VC dimension in constructing an estimate for the risk of a fit. If we fit $N$ training points using a class of functions $\{f(x, \alpha)\}$ having VC dimension $h$, then the following bound holds with probability $1 - \eta$

$$R(\hat{f}) \leq \frac{R_{\text{emp}}(\hat{f})}{(1 - \sqrt{\lambda})}, \qquad \text{where } \lambda = \frac{h \, \log(\frac{N}{h}) + h - \log(\frac{\eta}{4})}{N}. \tag{3.2}$$

These bounds hold simultaneously for all members of $f(x, \alpha)$ and are taken from Cherkassky and Mulier (2007, pages 116-118). Note that this expression for the risk is independent of the distribution P.

Now, *Structural Risk Minimization* (SRM) creates an increasing sequence of VC dimensions $h_1 < h_2 < \ldots$ which are related to hypothesis space $\mathcal{H}_{h_i}$

for $i = 1,\ 2,\ \ldots$ from different models. SRM consists in choosing the smallest value of the upper bound

$$\min_{\mathcal{H}_{h_i}} \left( \frac{R_{\text{emp}}(\hat{f})}{(1 - \sqrt{\lambda})} \right)$$

for $i = 1,\ 2,\ \ldots$ . An excellent example in which the SRM can be successfully carried out are *Support Vector Machines*. To discuss SVMs we have to introduce an alternative loss function called the *$\epsilon$-insensitive loss function*.

## 3.2 $\epsilon$-insensitive Loss Function [4] [9]

The problem with kernelised ridge regression is that the coefficient vector $\hat{\alpha} \in \mathbb{R}^p$ depends on all the $p$ training points, which could face difficulties with computation time and is therefore not a sparse estimate. To obtain a sparse estimate we will introduce a new loss function which includes a distance measure, the *epsilon insensitive loss function*, defined by

$$L_\epsilon\big(Y, \hat{f}(X)\big) = \begin{cases} 0 & \text{if } |Y - \hat{f}(X)| \leq \epsilon \\ |Y - \hat{f}(X)| - \epsilon & \text{otherwise.} \end{cases}$$

The above equation is saying that any point that is lying in a *$\epsilon$-tube* around the prediction $\hat{f}(x_i)$ is not penalized by the loss function. Figure 11 and 12 make this visible.

## 3.3 Support Vector Regression [7] [8] [9]

The most important idea of applying SVMs on regression, support vector regression (SVR), is that using small subsets of the training data will give us enormous computational advantages. In SVR the basic idea is that an input vector $x$ first will be mapped onto a $m$-dimensional feature space via a non-linear fixed mapping $\phi$ and to do linear regression in this space,

$$\hat{f}(x) = \hat{\alpha}_0 + \big\langle \hat{\alpha}, \phi(x) \big\rangle \qquad \text{with} \qquad \phi : \mathbb{R}^p \to \mathcal{X}.$$

We have to keep in mind that $\hat{\alpha}_0$ is the *bias* term.

Briefly, SVR is linear regression in a high dimensional feature space which corresponds to non-linear regression in the low dimensional input space $\mathbb{R}^p$.

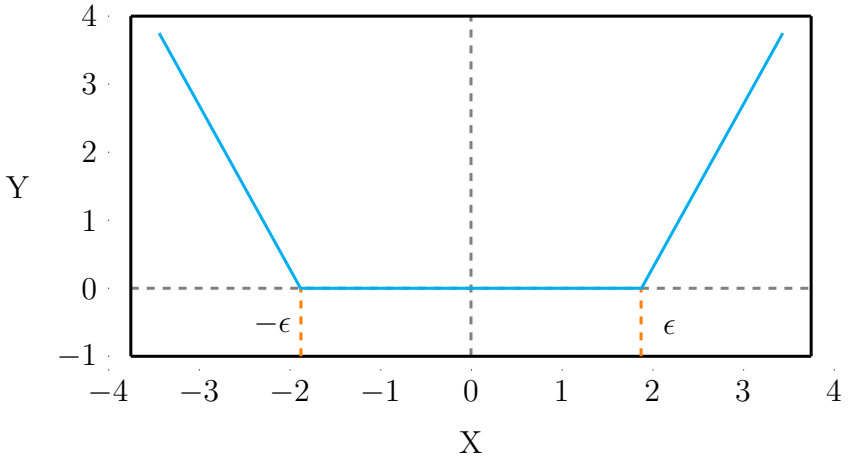



Figure 11: $\epsilon$-insensitive error function used by SVM regression

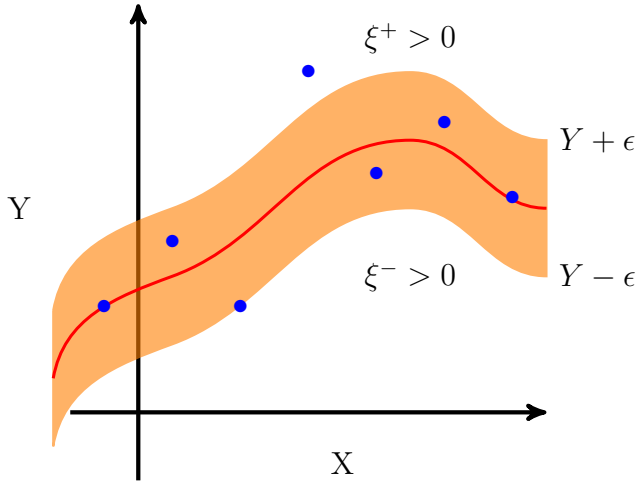



Figure 12: Illustration of the $\epsilon$-tube used in SVM regression for $\epsilon = 1$

Using the kernel trick, the dot product would not have to be computed in this high dimensional space and can be expressed in the low dimensional input space $\mathbb{R}^p$.

Since $\phi$ is fixed, we determine $\hat{\alpha}$ from the data like before: minimizing equation 3.1 but now we will use the $\epsilon$-intensive loss function. However, this

empirical risk function is not differentiable, as a result of the absolute value in the loss function. To avoid this problem we introduce the *slack variables* $\xi_i^+, \xi_i^- \geq 0$ for $i = 0, \ \ldots \ , \ p$ to express the amount of which each point lies above or below the $\epsilon$-tube. See figure 12.

Thus SVR can be formulated as minimizing the following function with $C = \frac{2}{\delta^2}$

$$\min_{f \in \mathcal{H}} \left[ C \sum_{i=0}^{p-1} (\xi_i^+ + \xi_i^-) + \frac{1}{2} \|\hat{\alpha}\|^2 \right],$$

$$\text{subject to} \quad \begin{cases} Y_i & \leq \hat{\alpha}_0 + \left\langle \hat{\alpha}, \phi(x) \right\rangle + \epsilon + \xi_i^+ \\ Y_i & \geq \hat{\alpha}_0 + \left\langle \hat{\alpha}, \phi(x) \right\rangle - \epsilon - \xi_i^- \\ \xi_i^+, \xi_i^- & \geq 0. \end{cases} \tag{3.3}$$

### 3.3.1 Support Vector Expansion

Minimizing equation 3.3 is a *quadratic programming optimization* problem. To solve this optimization problem we will use the Lagrange multipliers $\eta_i^+, \eta_i^-, \mu_i^+, \mu_i^- \geq 0$ to create the Lagrangian

$$\begin{aligned} \Lambda = \frac{1}{2} \|\hat{\alpha}\|^2 + C \sum_{i=0}^{p-1} (\xi_i^+ + \xi_i^-) - \sum_{i=0}^{p-1} (\eta_i^+ \xi_i^+ + \eta_i^- \xi_i^-) \\ - \sum_{i=0}^{p-1} \mu_i^+ \big( \hat{\alpha}_0 + \left\langle \hat{\alpha}, \phi(x) \right\rangle + \epsilon + \xi_i^+ - Y_i \big) \\ - \sum_{i=0}^{p-1} \mu_i^- \big( Y_i - \hat{\alpha}_0 - \left\langle \hat{\alpha}, \phi(x) \right\rangle + \epsilon + \xi_i^- \big). \end{aligned} \tag{3.4}$$

It follows from the existence of a saddle point that the partial derivatives of $\Lambda$ with respect to the variables $\hat{\alpha}_0, \hat{\alpha}, \xi_i^+, \xi_i^-$ have to be zero

$$\frac{\partial \Lambda}{\partial \hat{\alpha}_0} = \sum_{i=0}^{p-1}(\mu_i^+ - \mu_i^-) \qquad = 0 \tag{3.5}$$

$$\frac{\partial \Lambda}{\partial \hat{\alpha}} = \hat{\alpha} - \sum_{i=0}^{p-1}(\mu_i^+ - \mu_i^-)\phi(x_i) \qquad = 0 \tag{3.6}$$

$$\frac{\partial \Lambda}{\partial \xi_i^+} = C - \mu_i^+ - \eta_i^+ \qquad = 0 \tag{3.7}$$

$$\frac{\partial \Lambda}{\partial \xi_i^-} = C - \mu_i^- - \eta_i^- \qquad = 0 \tag{3.8}$$

Substituting equation 3.5 and 3.6 into equation 3.4 and reformulating equation 3.7 and 3.8 to $\eta_i^{\pm} = C - \mu_i^{\pm}$ will result in maximizing

$$\begin{cases} -\frac{1}{2}\sum_{i,j=0}^{p-1}(\mu_i^+ - \mu_i^-)(\mu_j^+ - \mu_j^-)\langle x_i, x_j\rangle \\ -\epsilon\sum_{i=0}^{p-1}(\mu_i^+ + \mu_i^-) + \sum_{i=0}^{p-1} Y_i(\mu_i^+ - \mu_i^-) \end{cases}$$

$$\text{subject to} \qquad \sum_{i=0}^{p-1}(\mu_i^+ - \mu_i^-) = 0 \;\text{ and }\; \mu_i^+, \mu_i^+ \in [0, C].$$

It follows that equation 3.6 could be written as

$$\hat{\alpha} = \sum_{i=0}^{p-1}(\mu_i^+ - \mu_i^-)\phi(x_i).$$

Thus $\hat{f}(x)$ can be written as

$$\hat{f}(x) = \hat{\alpha}_0 + \sum_{i=0}^{p-1}(\mu_i^+ - \mu_i^-)\big\langle \phi(x_i), x\big\rangle. \tag{3.9}$$

This is called *support vector expansion*. The coefficient estimate vector $\hat{\alpha}$ can be described as a linear combination of training data $x_i$. From another point of view, the complexity of the function $\hat{f}$ only depends on the number of *support vectors* and is independent of the dimension of the input space $\mathcal{X}$.

Finally, we can use the *kernel trick* to replace the inner product $\langle \phi(x_i), x \rangle$ with a kernel function $\kappa(x_i, x)$, making the prediction

$$\hat{f}(x) = \hat{\alpha}_0 + \sum_{i=0}^{p-1} (\mu_i^+ - \mu_i^-) \kappa(x_i, x). \tag{3.10}$$

The difference between equation 3.9 and equation 3.10 is that $\hat{\alpha}$ can no longer completely be described as a linear combination of the training data $x_i$.

# 4 Performance of Regression Methods

In this section we put the theory of the first three chapters into work. According to Chapter 1, we have been doing experiments with the amount of available *train* and *test* data to see the relationship between model complexity and the resulting *train* and *test error*.
Afterwards, we present familiar experiments using the discussed techniques in chapter 2 and 3, we give a close look at the performance of kernelised regression methods. Finally, we compare the performance of *linear*-, *kernelised ridge*- and *support vector regression* (SVR) on a single data set.

To accomplish these experiments we have been using implementations for `MatLab` and `Python` to produce and calculate error margins, execution time, plots etc. Especially, the MatLab toolbox `PMTK3` which comes together with the book *Machine Learning: a Probabilistic Perspective* written by K.P. Murphy.

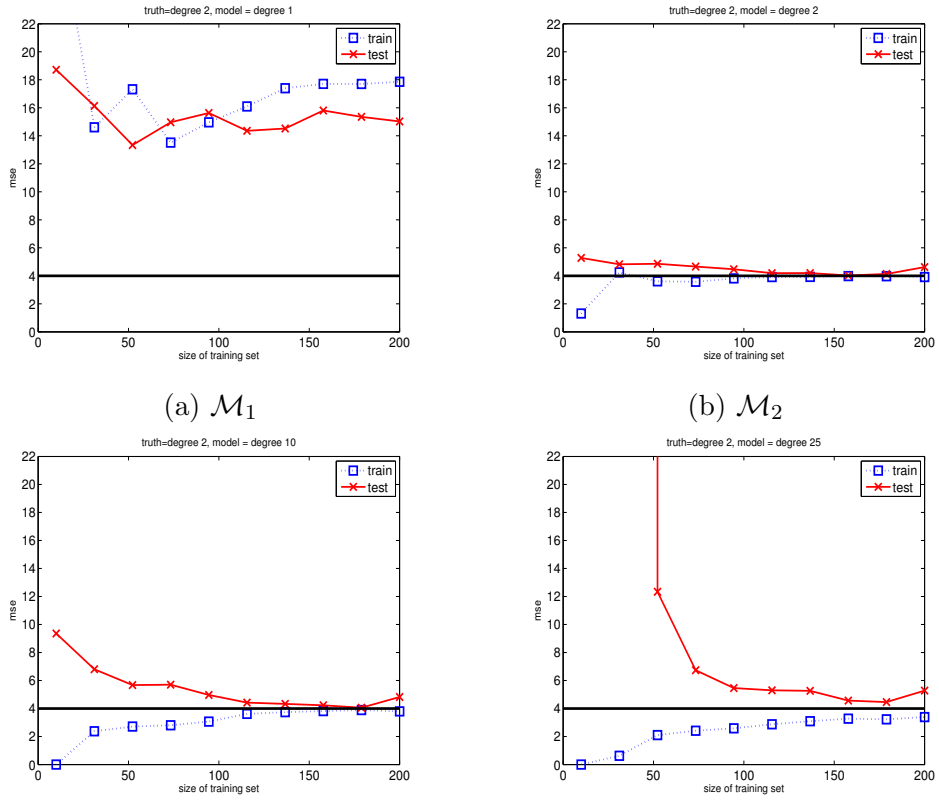
## 4.1 Train Error, Test Error and Model Complexity [10]

### 4.1.1 Polynomial Regression and the Degrees of Freedom

Intuitively, it is obvious that the more training data is available, the better we will be able to learn. We did an experiment with data distributed as a degree 2 polynomial with Gaussian noise of variance $\sigma^2 = 4$. We tried to fit polynomials of degree $1, 2, 10$ and 25 with an increasing amount of available training data. We call the four models $\mathcal{M}_1$, $\mathcal{M}_2$, $\mathcal{M}_{10}$ and $\mathcal{M}_{25}$.

The results of the mean squared training error (dotted blue) and test error (solid red) for increasing amount of available data are illustrated in figure 13. We see that for $\mathcal{M}_2$, $\mathcal{M}_{10}$ and $\mathcal{M}_{25}$ the test error decreases to some limit. This limit consists of the approximation and the estimation error, together they form the *noise floor* of the model.

We see that the *estimation error* for model $\mathcal{M}_2$, $\mathcal{M}_{10}$ and $\mathcal{M}_{25}$ is zero, since both are able to properly estimate the regression function. However, the estimation error of $\mathcal{M}_1$ is substantial, which becomes clear from the fact that the error occur high above the noise floor.




(a) $\mathcal{M}_1$ (b) $\mathcal{M}_2$

(c) $\mathcal{M}_{10}$ (d) $\mathcal{M}_{25}$

Figure 13: Train and test errors for polynomial regression with different degrees of freedom and with an increasing amount of available training data

In other words, for any model that is capable enough to estimate the true regression function, the test error will go to the noise floor as $N \to \infty$.

On top of this, we realize that there will be some discrepancy between the parameters that we estimate and the best parameters that we could estimate given a particular model. Before, we discussed this as the *approximation error* and goes to zero as $N \to \infty$, but again, it goes faster to zero for simpler models. We can see this happening in figure 13.

So far we have been discussing the test error, which is what we care most about since ideally our model has to generalize well. What about the train-

ing error? For models that are too simple, the training error is initially high, since we cannot model the truth. But for models that can estimate properly the regression function, the training error will increase to some limit as $N$ increases. The reason for this is that initially the model is powerful enough to memorize the training data. But when more data will be available, it becomes harder to fit all the data perfectly given a fixed model. Eventually, error on the training set will match the error on the test set, which also becomes visible in figure 13.
Beside this, if the error on the training set increases with $N$, it is a sure sign that we are overfitting.

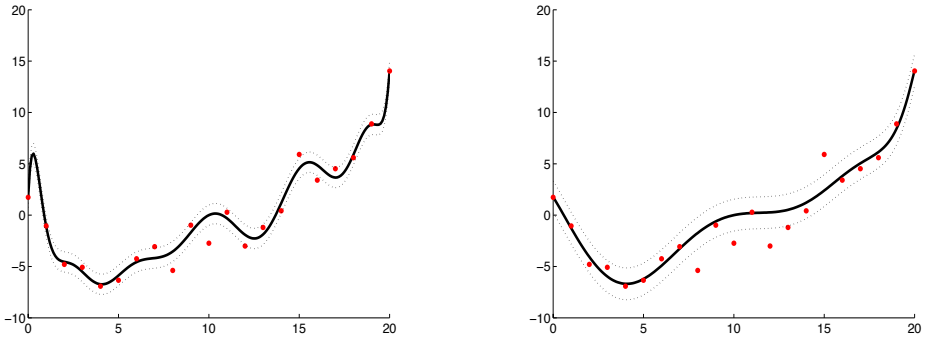
To conclude, when lots of data are available, simple models can work surprisingly well. However, there will be problems if there is too little data. This is a reason to study more sophisticated learning models. For example, even in the data-rich domain as web search, from the moment we want to start personalizing the results, the amount of data available for individual users look small again relative to the complexity of the problem.

### 4.1.2 Ridge Regression and the Choice of $\delta$

In section 2.1.3 we discussed three degree 14 polynomial fits using ridge regression. Similar to the preceding section, in this section we study the behavior of the train and test error but now in relation to the choice of $\delta$. Beside this, we take a closer look to the shrinkage technique of ridge regression. As a simple example, suppose we fit again a degree 14 polynomial to $N = 21$ points of data for $\delta = e^{-20,135}$, note that this is almost comparable with LSE. Therefore, the resulting curve in figure 14(a) is very wiggly. The corresponding coefficients of $\hat{\alpha}$ are as follows (without $\hat{\alpha}_0$).

| $\hat{\alpha}_1$ | $\hat{\alpha}_2$ | $\hat{\alpha}_3$ | $\hat{\alpha}_4$ | $\hat{\alpha}_5$ | $\hat{\alpha}_6$ | $\hat{\alpha}_7$ |
|---|---|---|---|---|---|---|
| 6.560 | -36.934 | -109.255 | 543.452 | 1022.561 | -3046.224 | -3768.013 |
| $\hat{\alpha}_8$ | $\hat{\alpha}_9$ | $\hat{\alpha}_{10}$ | $\hat{\alpha}_{11}$ | $\hat{\alpha}_{12}$ | $\hat{\alpha}_{13}$ | $\hat{\alpha}_{14}$ |
| 8524.540 | 6607.897 | -12640.058 | -5530.188 | 9479.730 | 1774.639 | -2821.562 |

Table 1: Coefficients of the degree 14 polynomial fit in figure 14(a)



(a) $\delta = e^{-20,135}$ (b) $\delta = e^{0,102}$

Figure 14: Ridge regression fit for different choices of $\delta$

We see that there are many large positive and negative coefficients. These balance out exactly to make the curve wiggly such that it almost perfectly interpolates the data. But we realize that this situation is unstable, if we change the data a little the coefficients would change a lot. By penalizing the the sum of the magnitudes of the coefficients, we ensure the function is more simple. We see that increasing $\delta$ results in a smoother function, illustrated in figure 14(b). This time using $\delta = e^{0,102}$ we find the following coefficients.

| $\hat{\alpha}_1$ | $\hat{\alpha}_2$ | $\hat{\alpha}_3$ | $\hat{\alpha}_4$ | $\hat{\alpha}_5$ | $\hat{\alpha}_6$ | $\hat{\alpha}_7$ |
|---|---|---|---|---|---|---|
| 2.128 | -36.934 | 0.807 | 16.457 | 3.704 | -24.948 | -10.472 |
| $\hat{\alpha}_8$ | $\hat{\alpha}_9$ | $\hat{\alpha}_{10}$ | $\hat{\alpha}_{11}$ | $\hat{\alpha}_{12}$ | $\hat{\alpha}_{13}$ | $\hat{\alpha}_{14}$ |
| 4.360 | 13.711 | 10.063 | 8.716 | 3.966 | -9.349 | -9.232 |

Table 2: Coefficients of the degree 14 polynomial fit in figure 14(b)

In figure 15(a) we plotted the mean squared error (MSE) on a training and a test set versus $\log(\delta)$ for the above ridge regression fits. Complex functions (small delta) begin on the left moving towards simple functions (big delta) on the right. We see that when we increase $\delta$ the error on the training set (dotted blue) increases. For the error of the test set (solid red), we first see an overfit and thereafter an underfit.

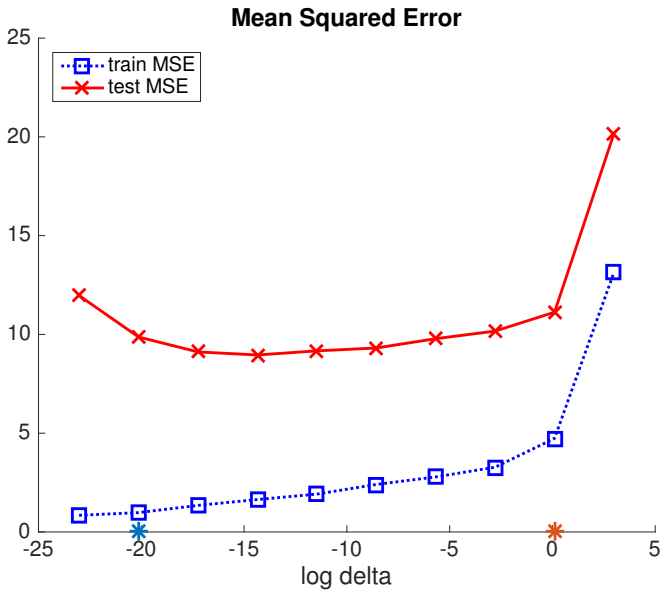



Figure 15: The stars correspond to the values of $\delta$ used for ridge regression in figure 14 .

In this section, we saw more analytical results of the ridge regression method for different values of $\delta$. A common way to pick $\delta$ is using cross validation. A 5-fold cross validation estimate of the future MSE for this set of data is shown in figure 16. As we see, CV comes up with the value $\delta = e^{0,102}$.

## 4.2 Kernelised Regression

One big disadvantage of KRR is that we do not have sparseness in the vector $\alpha$ which can lead to excessive computation time if big datasets need to be evaluated. SVR instead, is using the concept of support vectors. In the context of computation time, this is useful because when we test a new data, i.e. new test data, we only have to sum over the support vectors which is much faster than summing over the entire training set. First we take a look at the performance of KRR. Afterwards, the relationship between sparseness and support vectors becomes clear from the experiments with SVR using different loss functions.

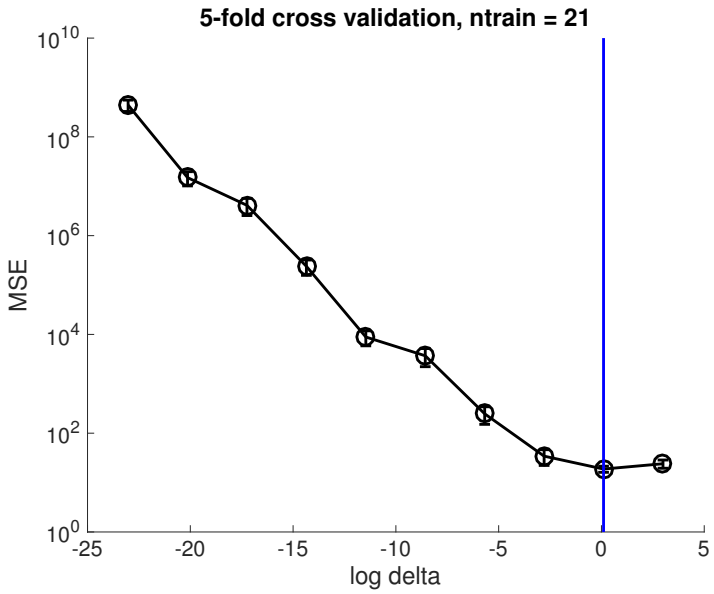
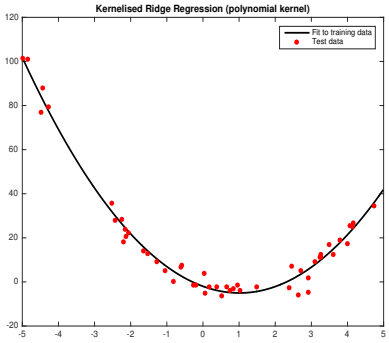



Figure 16: 5-fold cross validation, the blue line indicates the lowest MSE value

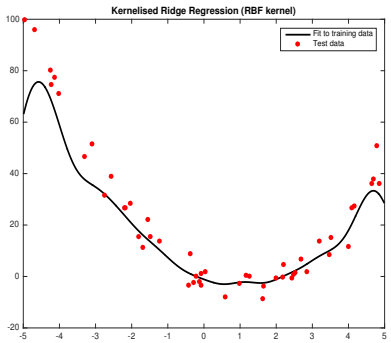
### 4.2.1 Kernelised Ridge Regression and the Choice of the Kernel Function

In this section, we apply KRR to data with added noise, the data was originally distributed as a degree 2 polynomial and the function $\frac{\sin(x)}{x}$. We are using the two different kernel functions we discussed in section 2.3.1. We separated the available data randomly into training data and test data according to the 80-20% ratio. The resulting fit relies only on the set of training data, the plotted red dots are the test data.

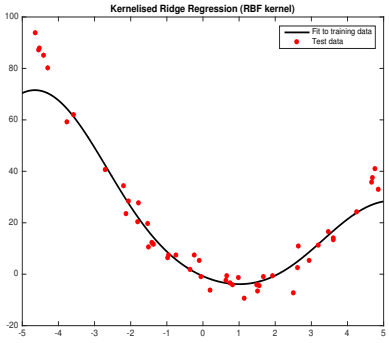
In the case of the originally degree 2 polynomial distributed data, for fixed $\delta$ and polynomial kernel function $\kappa(x, y) = \big(\langle x, y\rangle + 1\big)^2$ we obtained the fit in figure 17(a). Which is an excellent result. For fixed $\delta$ and RBF kernel function $k_i(x) = \exp(-\frac{1}{2\sigma^2}\|x - x_i\|^2)$ using different values for $\sigma^2$ we obtained the the other results in figure 17. It is self evident that the kernel functions does not fit the data so well.



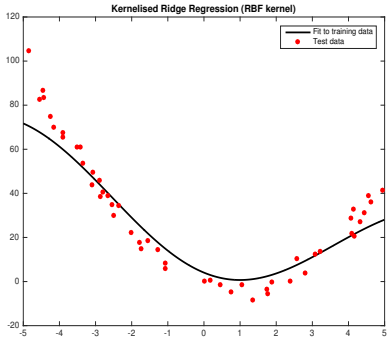

(a) KRR using the polynomial kernel function $\kappa(x, y) = \big(\langle x, y\rangle + 1\big)^2$




(b) KRR using the kernel function $k_i(x) = \exp(-\frac{1}{2\sigma^2}\|x - x_i\|^2)$, $\sigma^2 = 0.5$



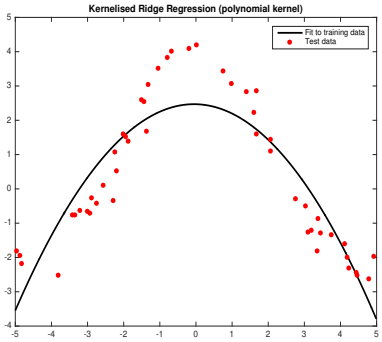

(c) KRR using the kernel function $k_i(x) = \exp(-\frac{1}{2\sigma^2}\|x - x_i\|^2)$, $\sigma^2 = 2$



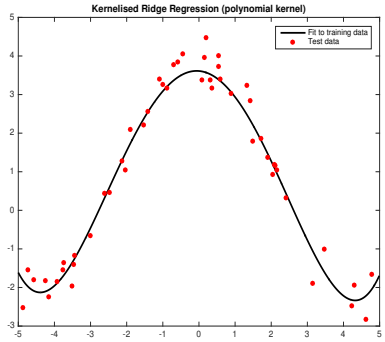

(d) KRR using the kernel function $k_i(x) = \exp(-\frac{1}{2\sigma^2}\|x - x_i\|^2)$, $\sigma^2 = 4$

Figure 17: KRR fits to originally polynomial distributed data using different kernel functions

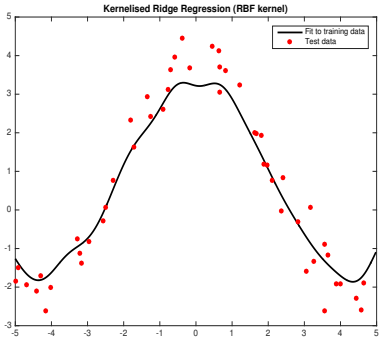
However, if we apply KRR to data originally distributed as the function $\frac{\sin(x)}{x}$, the RBF kernel performs better. This is illustrated in figure 18.



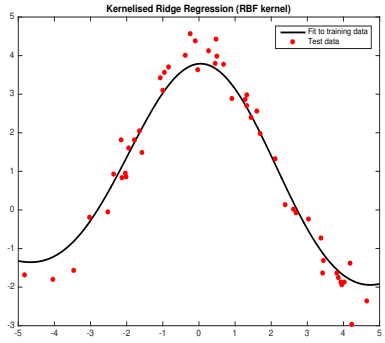

(a) KRR using the polynomial kernel function $\kappa(x, y) = \big(\langle x, y\rangle + 1\big)^2$




(b) KRR using the polynomial kernel function $\kappa(x, y) = \big(\langle x, y\rangle + 1\big)^4$




(c) KRR using the kernel function $k_i(x) = \exp(-\frac{1}{2\sigma^2}\|x - x_i\|^2)$, $\sigma^2 = 0.5$

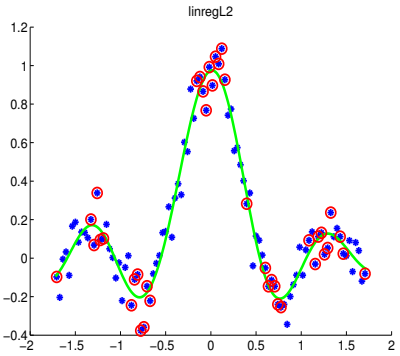



(d) KRR using the kernel function $k_i(x) = \exp(-\frac{1}{2\sigma^2}\|x - x_i\|^2)$, $\sigma^2 = 2$

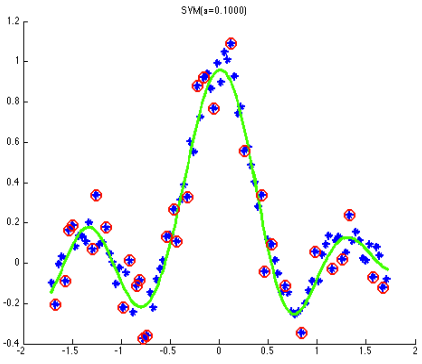
Figure 18: KRR fits to originally $\frac{\sin(x)}{x}$ distributed data using different kernel functions

### 4.2.2 SVR and the Loss Function

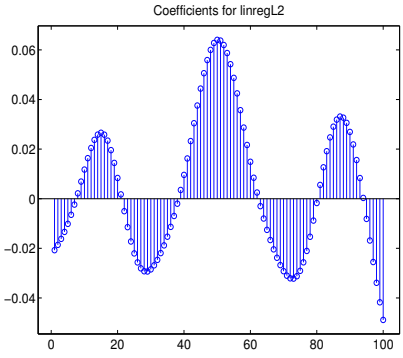
In figure 19 we compare SVR models with a RBF kernel using different loss functions. Due to the $\epsilon$-insensitive loss function SVR obtains a sparse model, leading to good generalization. However, if we take a way this characteristic loss function, the sparseness will be gone. We illustrate this using the quadratic loss function for SVR in figure 19.



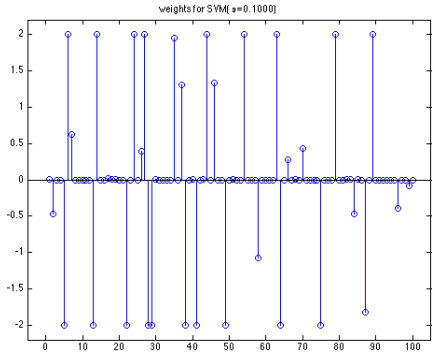

(a) SVR with the quadratic loss function




(b) SVR with the $\epsilon$-insensitive loss function, $\epsilon = 0.1$




(c) Coefficient vectors for the above regression model (non-sparse)




(d) Coefficient vectors for the above regression model (sparse)

Figure 19: Support Vector Regression using different loss functions

Red circles denote the training data. We see that the two methods give similar performance. However, SVR using the $\epsilon$-insensitive loss function is much sparser (and hence faster in time) then SVR using the quadratic loss function.

## 4.3 Linear Regression versus Support Vector Regression [11]

Linear regression assumes that the relationship between the input $X$ and the output $Y$ is approximately linear. To find the best linear fit we minimize the sum of squared errors (1.6). However, it is extremely unlikely that the true function $f(X)$ is indeed linear in $X$. So often, a fit of linear regression model is useless and will never be used. The performance of the linear fit to data is given in figure 20. The data is generated according to a sinus function where strong noise has been added to every fifth data point.

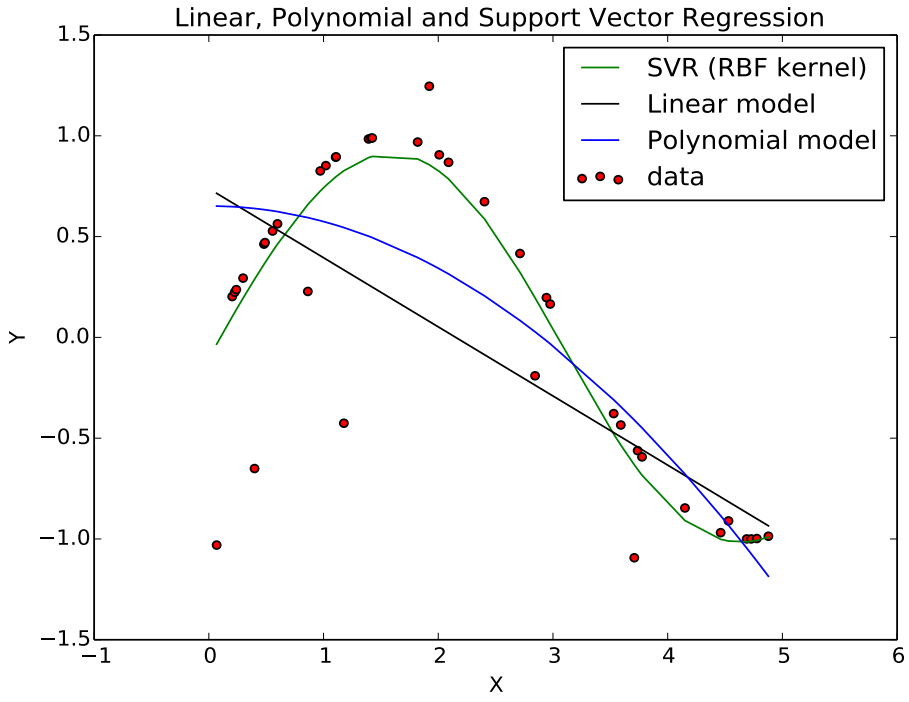



Figure 20: Performance of linear, polynomial and SVR learning models

The introduction of linear basis function expansion allowed us to use polynomial regression models. Polynomial regression assumes that the relationship between the input $X$ and the output $Y$ is modeled as an $n$-th degree polynomial. The performance of a degree 2 polynomial fit is given in figure 20.

A final alternative is to use kernelised models such as support vector regression with a polynomial or a RBF kernel. Kernelised regression, as a non-parametric regression method, does not assume any underlying function between input $X$ and output $Y$. The performance of SVR using a RBF kernel with $\sigma^2 = 2$ is illustrated in figure 20.

## 4.4 Kernelised Ridge Regression versus Support Vector Regression [11]

Kernelised Ridge Regression (KRR) and Support Vector Regression (SVR) are both kernelised regression models but differ in the penalty function (ridge versus $\epsilon$-insensitive loss function). For KRR we found a closed-form solution (2.10) instead of the quadratic programming problem SVR has to deal with, discussed in section 3.3. Because of this, KRR is typically faster for small/medium sized datasets. On the other hand, for bigger sized datasets, SVR will have lower prediction time since KRR is not sparse.

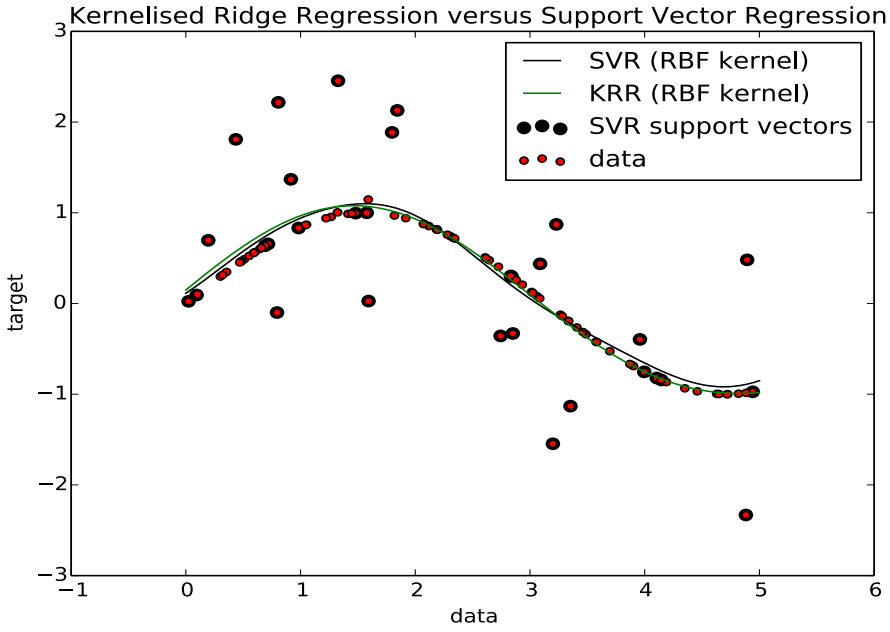
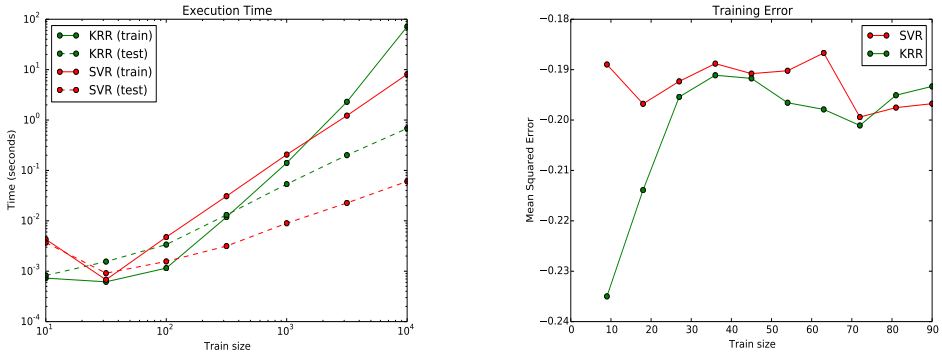



Figure 21: The fit of KRR and SVR, $\sigma^2 = 2$




(a) (b)

Figure 22: Left: Execution Time of KRR and SVR. Right: training error of KRR and SVR

Figure 21 illustrates the performance of KRR and SVR on a dataset. The data is generated according to a sinus function where strong noise has been added to every fifth data point. The two methods give similar performance. Fitting KRR is faster than SVR for small/medium training sets (for less than $10^3$ samples). However, for larger training sets SVR scales better. The execution time for the above fit is represented in figure 22(a). Note that the degree of sparsity and thus the fitting time depends on the parameters $\epsilon$ and $C$ of the SVR. Which can be determined by $k$-fold cross validation.

## 4.5 Conclusions

The theoretical framework of supervised learning, introduced in Chapter 1, helped us to understand the origin of the challenges supervised learning models have to deal with. In this report, we have been focussing on two major challenges: the tradeoff between *bias* and *variance* and the *model complexity* related to the *amount of training data*.

We decomposed bias and variance in the conceptually and mathematically way to understand better the data fitting process. Using the *empirical risk minimization* (ERM) principle, we computed the theoretical decomposition of the bias-variance tradeoff for *linear* - and *linear ridge regression*.
Then, to control better the stability of the obtained learning models, we introduced the non-parametric method *kernel regression*. Different from linear regression methods, kernel regression makes no assumptions about the probability distribution of the variables. But kernel regression is known as a non-sparse method, which could lead to excessive computation time for big datasets.

An example of a sparse kernel regression method is *support vector regression*. This learning model relies only on relevant information from the data, so called support vectors. Since less data has to be evaluated, this leads to computational advantages. However, support vector regression relies on the more complex *structural risk minimization* (SRM) principle. SRM minimizes an upper bound on the expected risk opposing ERM that minimizes the error on the training data. This difference results in a better performance of SVR in supervised learning.

Away from all the theory, we have been experimenting in Chapter 4 with the above discussed techniques to model and learn from empirical data. We became aware of the fact that when enough data is available, simple models can work well. When less data is available, we have to rely on other models, i.e. non-parametric support vector regression. In doing so, we have to consider the sparsity-accuracy tradeoff as part of the model selection process. Ultimately, we have shown that using sparse kernel regression (SVR) is more effective in terms of execution time and leads to a slightly better predictive performance.

Finally, the advantage of statistical learning theory compared to cross validation is that the bounds on the risk are quicker to compute than using cross validation. The disadvantage is that it is hard to compute the VC dimension for many interesting models.

# References


[1] T. Hastie, R. Tibshirani, J. Friedman, The Elements of Statistical Learning Second Edition; Data Mining, Inference and Prediction (2008) - **Chapter 2, 3, 5, 6, 7**

[2] U. von Luxburg, B. Schölkopf, Statistical Learning Theory: Models, Concepts and Results (2008)

[3] S. R. Gunn, Technical Report, Support Vector Machines for Classification and Regression (1998)

[4] K.P. Murphy, Machine Learning a Probabilistic Perspective, The MIT Press (2012) - **Chapter 1, 6, 7, 14**

[5] N. de Freitas, Deep Learning Lecture 4: Regularisation, Model Complexity and Data Complexity - slides.pdf (2015)

[6] www.quora.com/What-is-the-difference-between-a-sparse-kernel-machine-and-a-non-sparse-kernel-machine

[7] K.R. Müller, A.J. Smola, G.Rätsch et al., Predicting Time Series with Support Vector Machines

[8] www.kernelsvm.tripod.com

[9] A.J. Smola, B. Schölkopf, A Tutorial on Support Vector Regression (2003)

[10] K.P. Murphy, M. Dunham, PMTK3 MatLab package (2011)

[11] Pedregosa et al., Scikit-learn: Machine Learning in Python, JMLR 12, pp. 2825-2830, (2011)