Bachelor's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

Jonas Nordstrøm

# Non-parametric Regression for Machine Learning: A Comparison of the Probabilistic and Algorithmic Approach

Bachelor's thesis in Statistics
Supervisor: Geir-Arne Fuglstad
May 2022

NTNU
Kunnskap for en bedre verden

Jonas Nordstrøm

# Non-parametric Regression for Machine Learning: A Comparison of the Probabilistic and Algorithmic Approach

Bachelor's thesis in Statistics
Supervisor: Geir-Arne Fuglstad
May 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

NTNU
Kunnskap for en bedre verden

**Abstract**

This report was submitted as a bachelor thesis for Bachelor in Mathematics at the Norwegian University of Science and Technology. The term non-parametric regression encompasses a great number of machine learning technique that predict the outcome of natural processes using functions that are more flexible than their parametric counterparts in order to they gain an edge in predictive capabilities. One may further subdivide the category into algorithmic and probabilistic models. The algorithmic approach focus on building models that has high predictive performance, while the probabilistic perspective builds models with a statistical approach that aims to mimic the process. For the purpose of comparing these two approaches, two algorithmic models (local regression and relaxation) and one probabilistic model (Gaussian processes) are to be thoroughly covered. In a practical comparison between Gaussian processes and local regression, Gaussian processes where found to have a predictive perform at a similar level to local regression, while yielding much better variance estimation, but had a significantly longer computation time. By exploring the rich theory behind kernel functions found in Mercers theorem and reproducing kernel Hiblert spaces it was shown that all tree techniques can be expressed as estimating a function by linearly combining observation weighed by a kernel function.

# 1 Introduction

The problem of predicting uncertain natural processes is both a important and a difficult task. The usefulness of answering questions such as: "Who is going to win the next election? What is going to be the price of bitcoin next month?", "How do your genes contribute to your chance of getting hearth disease?", is not to be understated. Still, anyone who works within the respective fields of political science, finance and medicine knows that answering such questions is not an easy task. Whenever there is a need to predict the outcome of random processes and assess the uncertainty of predictors, it is wise to consult the field of statistics. In particular, the field of statistical modeling and machine learning has made some great achievements in the last decades.

Inspired by the article *Statistical Modeling: The Two Cultures* by Leo Brierman, we may divided the field of statistical modeling into two cultures: the Data Modeling Culture and the Algorithmic Modeling Culture. Models from these cultures will be referred to as *probabilistic* and *algorithmic* models, respectively. Both cultures aim to predict the outcomes in nature. One can think of nature as a kind of black box. We may observe inputs and outputs into and out of the black box, but not the precise mechanisms of the process itself. The Data Modeling Culture takes a statistical approach. They build a statistical model which they assume to describe the natural process. Then they use the model to predict future outcomes, hopefully mirroring the mechanisms of the natural process. Predictions of the model can easily be questioned by criticizing the assumptions the model is taking. The Algorithmic approach ignores the natural process all together, instead focusing solely on making predictions. The methods used make no attempt to mirror the natural mechanisms. Therefore, few assumptions are needed. Instead, their ability to make accurate predictions is the only measure of their effectiveness [Breiman, 2001].

Section 2.1 consists of an introduction to central concepts of this text, such as Non-parametric regression and methods of measuring performance of models. The focus of Section 3 is on the algorithmic models of local regression and regularization, which can be generalized to a class of functions called linear smoothers. Section 4 introduces the essence of the data modeling culture in building probabilistic models. The focus of this section is on Bayesian inference, and more specifically Gaussian processes. Section 5 consists of a practical comparison of local regression and Gaussian processes by implementing the two methods and comparing the results when used on two different data sets. Lastly, Section 6 will take a deeper dive into the theory of kernels

with the goal of describing the mathematical similarities between Regularization and Gaussian Processes and discussing their differences.

# 2 Problem Specification

## 2.1 Non-parametric Regression

The problems the two cultures aims to answer is either of the from of a regression problem or a classification problem. Both of these concepts are closely related. They define a set of quantitative properties we observe as inputs to a natural process and aim to predict the outcome of the natural process given the properties. The difference lie in the type of output natural process produces. If the output is discreet, that is, the output belongs to a class, we describe it as a classification problem. A classical example is to predict the actual digit from the picture of a hand written one. In this case, the input is a set of pixels, those that make up the picture of the digit, while the output the digit the human wanted to write. One may assign a desecrate value, 0-9, that corresponds to witch class the digit belongs to. On the other hand, a regression problem has a continuous output. For example, predicting the amount of drag that acts upon a rocket relative to its speed. Sometimes, the relationships are simple, like the linear relationship between how many items are sold and how much money is made. But often, the interaction is complex. Acceleration of the head in a motorcycle accident is a good example of this. As we can see in Figure 1, the amount of g-force that acts on the head is a constant value of zero right after the accident, then suddenly spikes to above negative 100 g, before increasing to around 50 g and then reverting back to around zero.

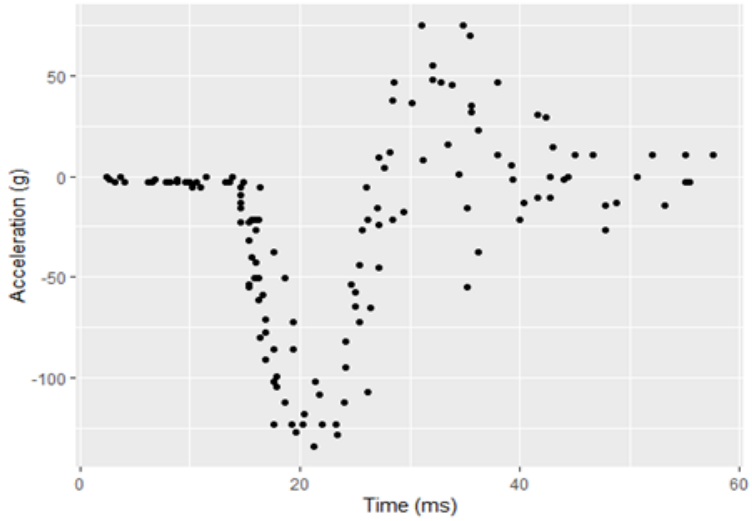



Figure 1: Acceleration of the head in a simulated motorcycle accident. Time measured in milliseconds after impact and acceleration measured in g.

This behavior is hard to simulate with parametric functions. For example, no polynomial starts of as a constant, before sharply decreasing. For this reason, it is useful to use non-parametric functions, which has unlimited complexity. This is because, in parametric regression we are limited to a fixed number of parameters, while in non-parametric regression, the number of pa-

rameters grows with the size of the training data [Murphi, 2012]. So, contrary to the name, a non-parametric model fits a function using a potentially infinite number of parameters.

This complexity comes at a cost however. Non-parametric regressions are viewed as black box methods in that their main focus is on *prediction* of future output of a process, rather than *inference*. The goal of inference is to quantify the relationship between the predictors and the response. It is often the case that non-parametric models give excellent prediction results while we have little insight into why the model works. Still, there are mayor differences within the field of statistical modeling. Probabilistic models, such as Gaussian processes, tend to be more aligned with the goal of performing inference than algorithmic models. Still, by the merit of being non-parametric, parametric models such as linear regression are significantly more easy to interpret.

A general formulation of any regression problem is that we have a set of data $\mathcal{D} = \{(\mathbf{x}_1, y_1), .., (\mathbf{x}_n, y_n)\}$, where $\mathbf{x}_i$ is a p-dimensional vector. In this text, I will focus on 1-d regression, where our goal is to map a one dimensional input, $x$, to a 1-d output $y$. The reasoning for this choice is that it is both easier to understand and visualize, while still sufficient to explain the core ideas of non-parametric regression. However, in some examples it is more suitable to use multidimensional input vector. The motivation for this is to demonstrate that it is often not difficult to convert a one dimensional problem to a multidimensional one. Whenever we have a multidimensional input it will be denoted as a vector, $\mathbf{x}$, instead of $x$.

We want to use future $x$ values to predict $y$. To do this, we need to quantify a relationship between $x$ and $y$. We assume

$$y = f(x) + \epsilon, \tag{2.1}$$

where $f : \mathbb{R} \to \mathbb{R}$. $y$ is our observations, like the points in Figure 1. These observations are the sum of two components: the true value of the process, like the actual acceleration of the head, and some noise that makes our observations diverge from the true value. This noise might be due to random differences in the process each time we repeat it or errors in the measurement of it. We may assume that these errors are evenly spread around the true value of the process and are more likely to be closer to the true value rather than further away. Mathematically, we express these assumptions by assuming that the error, $\epsilon$, are normally distributed, that is $\epsilon \sim \mathcal{N}(0, \sigma_n^2)$. To predict the process we need to find an accurate representation of $f$. Form the data modeling perspective, this process consists of making assumptions about the nature of $f$ and building a model from these assumptions. From the Algorithmic perspective, we simply want to find a good approximation of $f$. In both cases, we want to find a function $\hat{f}$ that estimates $f$. To determine the performance of $f$, or to test if our assumptions are justified, we need a measure of the accuracy and precision of $\hat{f}$ relative to $f$.

## 2.2 Measuring Predictive Performance

### 2.2.1 Loss Function

We would like to measure to what degree the estimated function, $\hat{f}$ differ from the true function, $f$. We may call a function that quantifies this general difference an *error function*. In practice, since we don't have access to the true value of $f(x)$, we use $y$ instead. In general we use a *loss function*, $L$, to quantify the difference between $\hat{f}$ and $y$ at a specific point. For example, in a classification problem with two classes a natural choice is to use the indicator function $I(\hat{f} \neq y)$, which is zero when $\hat{f}$ is the same class as $f$ and one otherwise. In the continuous case of a regression problem, a more natural choice is for $L$ to be a distance measure. The most common choice is the square error $L(\hat{f}, y) = (\hat{f}(x) - y)^2$ ([James, Witten, Hasti, Tibshirani, 2021] pp.

29). One way of measure the total error is to quantify the degree to which we may expect $\hat{f}$ to differ from $y$. Mathematically, this can be formulated as the expected value of the loss function, $\mathrm{E}[L(\hat{f}, f)]$, where $\hat{f}(x)$ is a random variable, since it implicitly depends on the observed data ([Wasserman, 2006] pp. 51). Additionally $\epsilon$ in $y = f(x) + \epsilon$ is by assumption a random variables, while $f(x)$ is not. We can estimate the expected value with the mean. Therefor, we may take the total error of our estimated function, given our data set $\mathcal{D}$, to be the mean square error (MSE) ([James, Witten, Hasti, Tibshirani, 2021] pp. 29)

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{f}(x_i))^2. \tag{2.2}$$

It is important to know that square error is not always the right choice. The main contender is the absolute error $L(\hat{f}, f) = |\hat{f}(x) - f(x)|$. Square error is preferred because, unlike absolute error, it is a continuous differentiable function. Since it is also convex, it is ideal for optimization, the importance of which is going to become obvious later. The weakness of MSE is that it punishes large outliers heavily, making the regression less *robust*. Robustness is a measure of the sensitive a prediction has to the introduction of new data points. A non-robust function changes significantly with the introduction of one single outlier into to data set. With absolute error, distance grows linearly instead of quadratically, so the effect of outliers is less severe, making it the preferred choice when outliers are known to be frequent. In this report, it is assumed that the effect of outliers is negligible, making MSE i viable choice.

The common way of building an algorithmic model is by specifying an error function, such as MSE, and finding the optimal solution to the problem by minimizing it. Probabilistic models differs in this regard. They build there models by specify beliefs about the behavior of the true function. Instead of being the starting point, the validation function is a product of the model building process. This can be viewed as the one of the main differences between algorithmic and probabilistic models.

When building a non-parametric algorithmic model, the MSE is unsuitable as a error function of two main reasons. Firstly, there exist a infinite number of functions that mininizes (2.2) – any function that interpolates every data point is a valid choice ([Hasti Tibshirani, 2009] pp. 30). Each of the algorithmic models covered in this text uses a modified version of the MSE which has a unique solution. Local regression uses a weighted sum, while regularization adds a penalty term. Both these approaches also solves the second problem of MSE; accounting for the bias-variance trade-off.

### 2.2.2 Bias-Variance trade-off

When trying measure MSE on a actual data set, we quickly encounter a problem. To reduce the MSE on the current data set, we could simply use a function that tries to interpolate the data as closely as possible, like in Figure 2

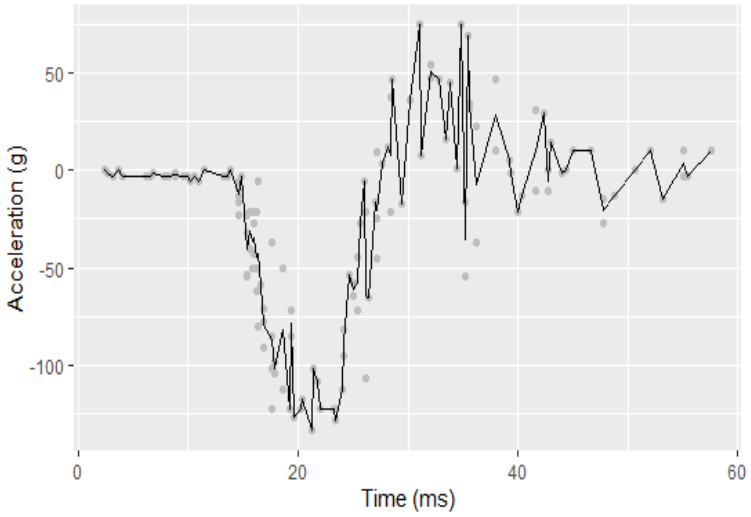



Figure 2: An example of a function that overfits our data.

The problem with this is that we are very likely to fit random noise, what we call *overfiting*. Because of this, our estimated function is unlikely to be a good approximation of the true function, $f$, and therefor not a good predictor of future data. Intuitively, we need that our function is close the data points, so that it is likely to approximate $f$, but not to close, so that we end up overfiting. We may express this duality mathematically. Since we assume $y$ to be on the form (2.1) can we decompose (2.2) into tree components

$$E[L(y, \hat{f}(x))] = E[(y - \hat{f}(x))^2] = E[\hat{f}(x)^2] - 2E[\hat{f}(x)y] + E[y^2]$$

$$= E[\hat{f}(x)^2] - 2(E[f\hat{(x)}]f(x) + E[\epsilon]E[\hat{f}(x)]) + f(x)^2 + 2E[\epsilon]f(x) + E[\epsilon^2].$$

Since we assume $\epsilon \sim \mathcal{N}(0, \sigma^2)$ we have

$$E[\epsilon] = 0 \text{ and } Var[\epsilon] = E[\epsilon^2] - E[\epsilon]^2 = E[\epsilon^2].$$

Using this and by adding and subtracting $E[\hat{f}(x)]^2$ we get

$$= (E[\hat{f}(x)]^2 - 2E[f\hat{(x)}]f(x) + f(x)^2) + (E[\hat{f}(x)^2] - E[\hat{f}(x)]^2) + Var[\epsilon]$$

$$= (E[\hat{f}(x)] - f(x))^2 + Var[\hat{f}(x)] + \sigma^2 = \text{BIAS}^2 + \text{VAR} + \sigma^2. \tag{2.3}$$

As denoted in the above equation, the square bias is defined as the square error between the expected value of $\hat{f}$ and the true function $f$. The square bias decreases when the complexity of $\hat{f}$ increases. The more complex $\hat{f}$ is, the closer it may approximate each data point $y_i$. However, the reduction of bias comes at a cost. To approximate every data point, $\hat{f}$ must fluctuate to a high degree and have low smoothness. Such a function has high variance, and therefor increases the second term in (2.3), and the total error with it. Because of this, variance increases with function complexity, and so, there is a trade-off between variance and bias. If we take the function to be to simple, we *underfit* the data. However, if we make it to complex we overfit instead. The sweet spot lies somewhere in between. One of the main challenges in any regression is to fine tune the model so that we may obtain this minimum. Even if we find it, $\hat{f}$ may never perfectly

predict new data points. This is because of the last term in (2.3). The error due to the variance of $\epsilon$, $\sigma_n^2$, is irreducible. Because of this, it is a lower bound of any error estimation, which can be seen in Figure 3 ([James, Witten, Hasti, Tibshirani, 2021] pp. 33-36).

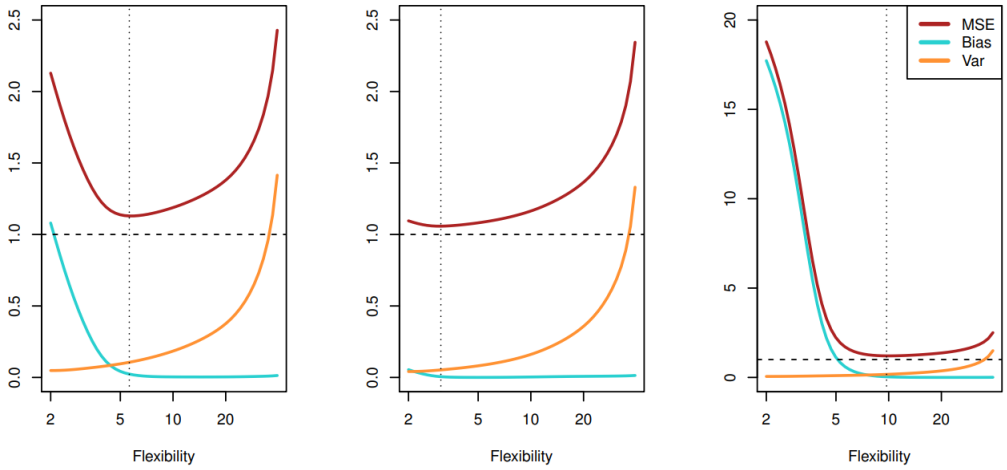



Figure 3: The Bias-Variance Trade-off for tree different data sets. The square bias (blue curve), variance (orange curve), MSE (red curve) and Var($\epsilon$) (dashed horizontal line). The vertical doted line denotes the minimum of MSE value. Source: [James, Witten, Hasti, Tibshirani, 2021]

### 2.2.3 Cross-Validation

Instead of using an error function, which only attempts to reduce bias, we must use a *validation function* which incorporates the bias-variance trade-off in it's error estimate. In the algorithmic culture, two techniques in particular are popular for accomplishing this task: *cross-validation* and *bootstrapping*. Both techniques attempt to simulate the process of observing new data and measuring the performance of our regression in predicting that data, but they accomplish this in very different ways. Bootstrapping uses our current data to estimate the distribution that our data is produced from. Then it uses the estimated distribution to produce new data points which we measure our regressions performance on. The detail of how this is done will be omitted. Instead, the focus is directed on cross-validation for the remainder of the text.

Cross-validation is similar to a more naive technique in which one sets aside a part of the data set that is used for validation, then fit the regression on the remaining data – the training set. The validation set is independent of the training set, so our error estimate is punished for fitting noise. The problem with this technique is that we are unable to use a significant portion of our data for training our regression function; worsening it's performance. Cross-validation has a clever solution to this problem. In p-fold cross validation, the data set is divided into p number of equally sized disjoint sets. Like in the naive method, we use on of the p sets for validation, the rest for training. But unlike the naive method, we repeat this process p times, once for each unique validation set, and then use the average performance ([James, Witten, Hasti, Tibshirani, 2021] pp. 203-204). For example, if we have 100 data points and perform 5 fold cross validation, we would divide the data into 5 sets of 20 data points and repeat the training validation process 5 times. Figure 4 shows an illustration of 5 fold cross validation.


Figure 4: Illustration of 5 fold cross validation. (Source: [Patro]

An alternative to dividing the data into p distinct data sets is to use leave p out cross validation. In this version, we leave p data points, train on the remaining, and use the p left out data points to for testing. At each test, we take the average of the p errors. We repeat this $\binom{n}{p}$ times, once for each possible combination of p, then taking the average. The most common choice of p is 1, and is called leave one out cross validation (LOOCV). LOOCV is computationally efficient when the estimated function is a linear combination of responses $y$ ([Hasti Tibshirani, 2009] p. 161). It turn out that many non-parametric functions is of this form. They can be generalized to a class of functions called linear smoothers.

# 3 Linear Smoothers

In this section we are going to take a close look at two algorithmic models, local regression and relaxation. Both these methods has a direct approach to estimating $f$. In local regression we stitch together a function from a series of points that each are a local weighted average. In relaxation we instead need to make some assumptions about the form of $f$. In both approaches, we make assumptions about the smoothness, or the variance, of $f$. In local regression, these assumptions is specified by the weights, while in relaxation we use a penalty function that punishes non-smooth functions. In both cases, these assumptions are encoded with a bandwidth or variance variable. They also share the general form of a linear smoother function.

## 3.1 Linear Regression as a Linear Smoother

A linear smoother is any function that is a linear combination of the response variables $\mathbf{y} = [y_1, \ldots, y_n]$. This is a concept that should be familiar to everyone with a basic knowledge of regression. Indeed, the most common regression type, linear regression, is of this form. In multiple linear regression, we expect $f$ to be a liner function, so $f(\mathbf{x}) = \mathbf{x}^T\mathbf{w} = \sum_{i=1}^{p} w_i x_i$, where $\mathbf{w}$ is a vector of weights and $\mathbf{x}$ a p-dimensional input. We find the best fit by minimizing the residual sum-of-squares (RSS), where $RSS = n * MSE$. Because we limit the freedom of $f$

by assuming it is a linear function there exist a unique minimizer of RSS. Taking $\mathrm{X} = [\mathbf{x}_1|\dots|\mathbf{x}_n]$ to be a matrix of data points, we may write RSS as

$$\mathrm{RSS}(\mathbf{w}) = (\mathbf{y} - \mathrm{X}\mathbf{w})^T(\mathbf{y} - \mathrm{X}\mathbf{w}).$$

The function is convex with respect to $\mathbf{w}$, so it is minimal when it's derivative is equal to zero. This is when $\hat{\mathbf{w}} = (\mathrm{X}^T\mathrm{X})^{-1}\mathrm{X}^T\mathbf{y}$. We get the estimate $\hat{\mathbf{y}} = \mathrm{X}\hat{\mathbf{w}}$ ([Hasti Tibshirani, 2009] pp. 64). The following definition is a generalization of this

**Definition 3.1.** *Let $\hat{f}$ be a linear smoother of $f : \mathbb{R}^p \to \mathbb{R}$ if there exist a vector $s(\mathbf{x}) = [s_1(\mathbf{x}), ..., s_n(\mathbf{x})]$, with $s_i : \mathbb{R}^p \to \mathbb{R}$ for each $\mathbf{x}$ such that*

$$\hat{f}(\mathbf{x}) = s(\mathbf{x})^T\mathbf{y} = \sum_{i=1}^{n} s_i(\mathbf{x})y_i \tag{3.1}$$

([Wasserman, 2006] p. 71, def 5.39). Using the smoothing function $s(\mathrm{x}) = \mathbf{x}(\mathrm{X}^T\mathrm{X})^{-1}X^T$, we see that linear regression can indeed be written on this form ([Wasserman, 2006] pp. 64).

The procedure of finding an estimate of $\hat{f}$ on the form of a linear smoother by defining a form of $f$ and optimizing an error function is going to be common case throughout this text. The purpose of introducing liner regression is to show how non-parametric regression techniques is far from alien, but is instead closely related to well know regression methods. One such non-parametric regression technique is local regression.

## 3.2 Local Regression

We continue to assume that $y$ to be on the form of (2.1), but we limit our self to the case where $x$ is one dimensional. The essence of local regression is build on the assumption that at each point $x$, a good estimate of $f$ is a weighed average of local data points. The flexibility in local regression arises from our freedom in choosing what we define local to be. For example, one simple approach is to use the K-nearest neighbours of $x$. Assume $\mathcal{B}_x$ represents the K-nearest points to $x$, then we may estimate $\hat{f}(x)$ by ([James, Witten, Hasti, Tibshirani, 2021] p. 105)

$$\hat{f}(x) = \frac{1}{K}\sum_{x_i \in \mathcal{B}_x} y_i.$$

Observe that if we define $s_i$ to be

$$s_i(x) = \begin{cases} \frac{1}{K}, & \text{if } x_i \in \mathcal{B}_x \\ 0, & \text{if } x_i \notin \mathcal{B}_x \end{cases}$$

then K-nearest regression is on the form a linear smoother.

There is a few problems with K-nearest regression. Firstly, we can't guarantee that all data points are uniformly distributed. Some regions might be sparse in its data intensity, while others much denser. Therefor, the general rule of always using the K nearest point might vary in accuracy depending on the sparsity. Secondly, we may assume that points closest to x are also closer to the true value of $f(x)$ than those further away. One solution to both these problems would be to assign weights to each $y_i$ depending on the distance from their corresponding x-coordinate, $x_i$, to $x$. At each point $x$ we would like to find a smoothing vector $s(x)$ that minimizes the distance to each $y_i$ and use a distance measure to favor those $y_i$ who's corresponding $x_i$

coordinate is the closest to $x$. Mathematical, we would like to minimize the weighted sum of squares (ref. [Wasserman, 2006] pp. 75)

$$\sum_{i=1}^{n} w_i(x)(y_i - \hat{f}(x))^2 \tag{3.2}$$

where $w(x) = [w_1(x), ..., w_n(x)]$ is a weight function. Minimizing with respect to $\hat{f}(x)$ we find

$$\hat{f}(x) = \frac{\sum_{i=1}^{n} w_i(x) y_i}{\sum_{i=1}^{n} w_i(x)} \tag{3.3}$$

since (3.2) is a convex function, (3.3) must be a global minimizer. A common choice of weigh functions is called kernel functions, so we take $w_i(x) = k(x, x_i)$ [Wasserman, 2006]. Using kernels in (3.3), we get what is referred to as the Nadaraya-Watson kernel estimator for the smoothing vector $s$ ([Wasserman, 2006] pp. 71 def. 5.39).

$$s_i(x) = \frac{k(x, x_i)}{\sum_{i=1}^{n} k(x, x_i)}. \tag{3.4}$$

This expression can be viewed as a normalized kernel, since $\sum_{i=1}^{n} s_i(x) = 1$. Using (3.4), we may write (3.3) on the general form of a smoothing function (3.1). From this, the non-parametric nature of local regression becomes obvious. Each new data point contributes to estimation of $s$. So when the size of the data grows, we may utilize more information in our estimation of the true function.

It is important to note that N-W regression is not the only form of local regression. Indeed, it is merely a special case of local polynomial regression. From Taylor's theorem we know that the polynomial

$$f(x) + f'(x)(u - x) + f''(x)\frac{(u - x)^2}{2!} + ... + f^{(p)}\frac{(u - x)^p}{p!}$$

is a good approximation of $f$ for all $u$ in the neighborhood of $x$. Motivated by this, we estimate a vector $a(x) = [a_0(x), ..., a_p(x)]$ such that

$$P_x(u; a) = a_0(x) + a_1(x)(u - x) + a_2(x)\frac{(u - x)^2}{2!} + ... + a_p(x)\frac{(u - x)^p}{p!}.$$

Similar to N-W regression, we want the vector $a(x)$ to minimize ([Wasserman, 2006] pp. 75)

$$\sum_{i=1}^{n} w_i(x)(y_i - P_x(x_i; a))^2,$$

the solution of which is omitted in this text. Notice that if we take the degree of the polynomial to be $p = 0$ – the case where we estimate a constant – we have regular kernel regression, which has the solution of the N-W kernel estimator. In Figure 5 we find an illustration of the Nadaraya-Watson (N-W) kernel estimator using the Epanechnikov kernel as weight function.

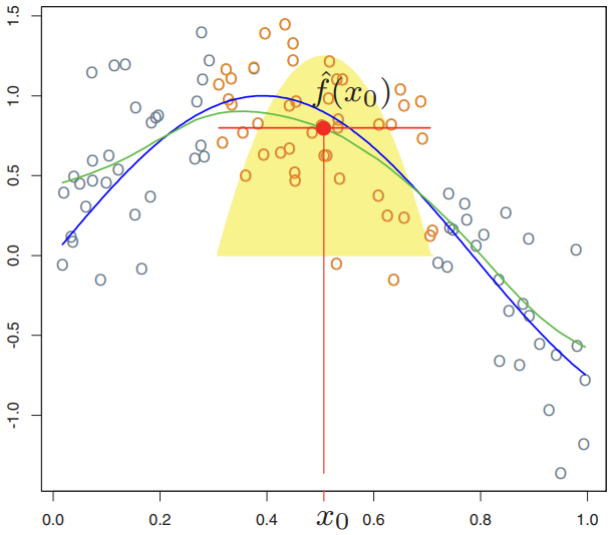



Figure 5: Nadaraya-Watson kernel estimator using the Epanechnikov kernel as weight function (green) and the true function from which the data points has been generated from (blue). The point $\hat{f}(x_0)$ is a weighed average of each of the red points in the neighborhood, where their weight corresponds to the height of Epanechnikov curve, shown in yellow. Source: [Hasti Tibshirani, 2009]

For a comparison, Figure 6 shows the result if we where to use either linear or quadratic regression.

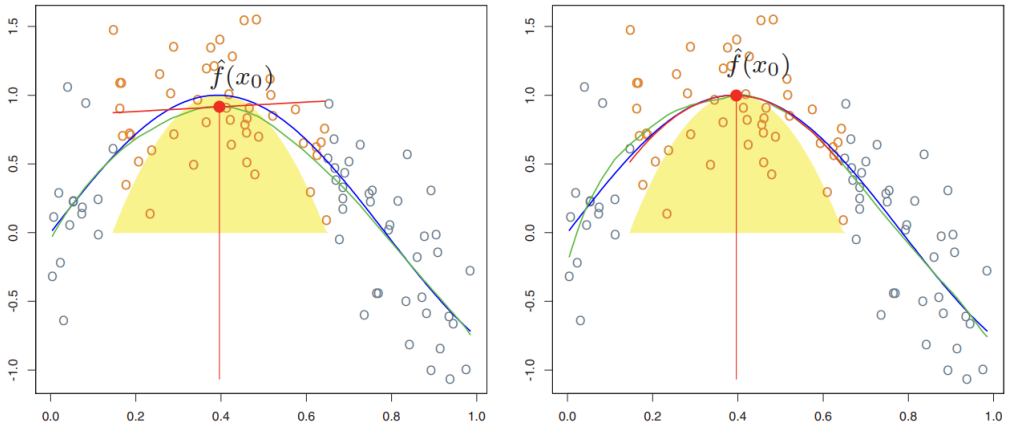



Figure 6: Linear (left) and quadratic (right) local regression. Source: [Hasti Tibshirani, 2009]

To better understand these figures, we need a basic understanding of kernel functions.

### 3.3 Kernels in Local Regression

For the purpose of this section, we may think of kernels as inverse distance measure. Kernel functions are both symmetric, $k(x, x') = k(x', x)$, positive, $k(x, x') \geq 0$, and attains it maximum

when $x = x'$ ([Wasserman, 2006] pp. 55). Since $k(x, x_i)$ is large when $x_i$ is close to $x$, the kernel grants us the property that desired for our weights. There exists a multitude of choices for kernel functions $k$. Using $k(x, x') = k(x - x')$, tree such choices are the Epanechnikov kernel defined by

$$k_e(x) = \frac{3}{4}(1 - x^2)I(x)$$

the boxcar kernel defined by

$$k_b(x) = \frac{1}{2}I(x)$$

and the Gaussian kernel

$$k_g(x) = \exp\left(-\frac{x^2}{2}\right)$$

Where

$$I(x) = \begin{cases} 1 & \text{if } |x| \leq 1 \\ 0 & \text{if } |x| > 1 \end{cases}$$

In figure 7 we see a plot of each of these functions.

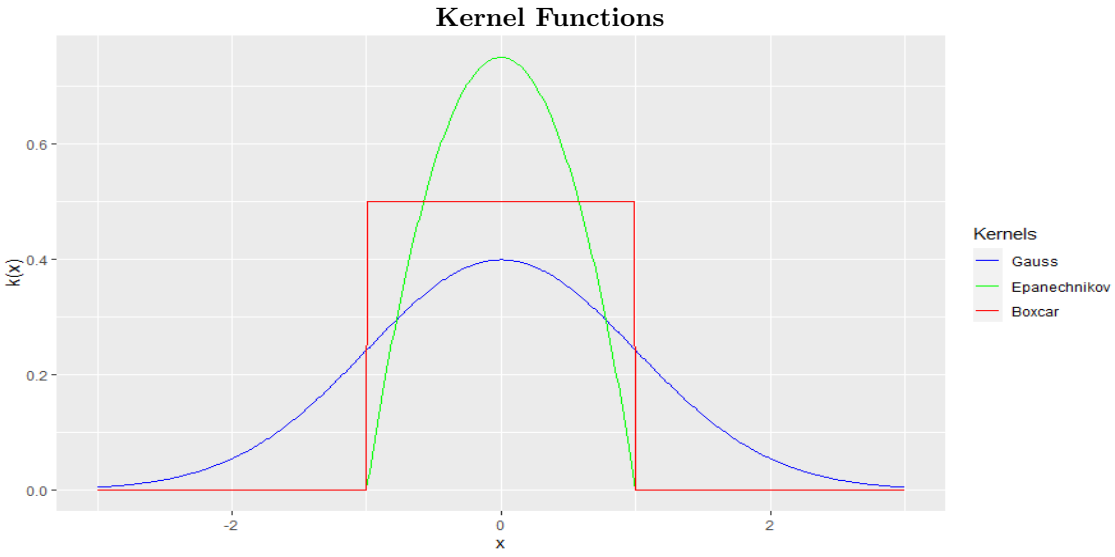



Figure 7: Value of each of the kernel functions around zero for a one dimensional input.

The choice of kernels encodes assumptions about how we think different points relate to each other. The boxcar kernel can be thought of ball $\mathcal{B}_x$ drawn around $x$ where every other point $x'$ is assigned a constant weight if they are inside the ball, or a weight of zero if they are outside. In our case, where $x$ is one dimensional, the ball is simply a interval centered around $x$. The Epanechnikov kernel differs from the boxcar kernel by favoring points closer to the center than further away, instead of assigning a constant weight. We are stating implicitly that the importance of each point $x'$ reduces gradually as they move further away from $x$ by choosing the Epanechnikov kernel instead of the Boxcar kernel.

The choice of kernel differentiate between relative distances, but say nothing about how

absolute distances should be weighted. If every data point is more than 1 unit of distance apart from each other, then the Epanechnikov and Boxcar kernel would assign every data pair $x$ and $x'$ (where $x \neq x'$) a weight of 0, which is less than desirable. To adjust absolute distance, we may introduce a constant, $h > 0$, called the *bandwidth* and define ([Hasti Tibshirani, 2009] pp. 35)

$$k_h(x) = k(x/h).$$

For both the Epanechnikov and Boxcar kernel, we see that $k(x, x')$ is non-zero when $|x - x'| \leq h$. $h$ can be thought of as the radius of $\mathcal{B}_x$. $h$ is important when considering the bias-variance trade off. If we take $h$ to be large, it will fit mostly the closes data points at each prediction point $x'$. This reduces the bias. Since fewer data points are used, the fit is more sensitive to change, so the predicted function has higher variance. When fitting a function, we have to tune $h$ to the value in which error is on the sweet spot between high bias and high variance.

Kernels is an important topic throughout this text, especially in later parts, such as Section 6. Even in methods such as regularization, which may at first seem entirely unrelated to kernels, we will eventually see that kernels hides just below the surface of the theory.

### 3.4 Regularization

Just like in local regression, we assume the true function to be close to the current data. To impose this assumption, we would like find a function that minimizes a means sum of squares. Unlike in local regression, we don't take a weighted sum of squares. Unfortunately, this makes the problem ill-posed – there exist an infinite number of solution function; any function that interpolates the data would do. In regularization, this challenges is solved by imposing smoothness conditions on the estimated function. This solves two problems: we force an unique solution and require the function to have minimal variance [Girosi, Jones, Poggio, 1995]. Mathematically, we want to minimize ([Rasmussen Williams, 2006] pp. 132)

$$J[f] = Q(f, y) + h\xi(f) \tag{3.5}$$

where $\xi$ is a penalty function that yields a low value when $f$ is smooth ([Hasti Tibshirani, 2009] pp. 165). Q is some data fit function that is small when $f$ is close to the data $y$. In Ridge regression, which is going the be the regularization technique of focus, we take $Q(f, y)$ to be the RSS ([James, Witten, Hasti, Tibshirani, 2021] pp. 237). $h$ is called a regularization parameter [Girosi, Jones, Poggio, 1995]. However, we may recognize $h$ as the bandwidth parameter that we have already encountered. When $h$ is large, we force larger smoothness requirement upon $f$, so that the variance of $f$ decreases, but the bias increases.
There are many choices for $\xi$. Remember that a linear function has zero second derivatives, while a function with large fluctuations has large second derivatives. So if $f \in C_2(\mathbb{R})$ is twice continuous differentiable, one intuitive way is to punish functions with large second derivatives. We may do this by taking the MSE of the second derivative. Since f is continuous, we use integration instead of MSE

$$\xi(f) = \int f''(t)^2 dt.$$

This is a common choice when regularizing splines and is called smoothing splines ([James, Witten, Hasti, Tibshirani, 2021], p. 301). In this text however, the focus will be on the case where $f$ is a linear combination of a set of linearly independent basis function. This can be accomplished by utilize a transformation, $\phi$, of $x$. For example, we may transform a scalar into a polynomial of degree p by using $\phi(x) = [1, x, x^2, ..., x^p]$. Then any polynomial of degree p

may be represented as

$$f(x) = \phi(x)^T \mathbf{w} \tag{3.6}$$

where $\mathbf{w}$ is vector of weights. Nothing in this formula limits us to the use of a polynomial transformation. Instead, this form is a generalization of any linear combination of basis functions $\phi(x) = [\phi_1(x), \ldots, \phi_p(x)]$. After deciding on a set of basis functions – what is often called *features* – we may adjust the behavior of $f$ by modifying the weights. However, not all functions on the form (3.6) is twice continuously differentiable. An example of this is the basis consisting of step functions, $\phi(x) = [I(x < c_0), I(c_0 x < c_1, \ldots, I(x < c_p)]$, where $I$ is the indicator function introduced in 3.3 and $c_0 < c_1 < \cdots < c_p$ a set of constants spread over a interval. Luckily, it turns out that a simple but efficient way of punishing non-smoothness of a function of the form (3.6) is simply taking the $\ell_2$-norm of the weight vector, what is called a *Ridge penalty* ([Hasti Tibshirani, 2009] pp. 68)

$$\xi(f) = \|w\|_2^2.$$

What is the intuition behind this? If the magnitude of each weight is unrestricted, then $f$ a greater variety of shapes, many of which has high variance. When the size of $\|w\|_2^2$ is restricted, the space of feasible forms of $f$ shrinks. In particular, if any feature $\phi_i(x)$ is of little importance in prediction the behavior of $f$, then the penalty is motivated to shrink it's corresponding weight, $w_i$, close to zero, making it's contribution insignificant. This effectively reduces the number of features, reducing the complexity and therefor the variance of $f$. This effect is especially prominent when using a *Lasso penalty* instead of a Ridge penalty – which uses a $\ell_1$-norm instead of a $\ell_2$-norm – since the norm shrinks many weights to exactly zero, while ridge only shrinks weights close to zero ([James, Witten, Hasti, Tibshirani, 2021] pp. 243-245).

If we take $\Phi$ to be a matrix such that $\Phi = [\phi(x_1)|\phi(x_2)|\ldots|\phi(x_n)] = \phi(\mathbf{x})$, we may rewrite (3.5) to

$$J[f] = (\mathbf{y} - \Phi\mathbf{w})^T(\mathbf{y} - \Phi\mathbf{w}) + h\mathbf{w}^T\mathbf{w}. \tag{3.7}$$

This form of regularization is called ridge regression. Minimizing (3.7), we find ([Hasti Tibshirani, 2009] pp. 64)

$$\bar{\mathbf{w}} = (\Phi^T\Phi + hI)^{-1}\Phi^T\mathbf{y} \tag{3.8}$$

We can observe several things about this result. Firstly, since $f(x) = \phi(x)^T\mathbf{w}$, we may define $s(x)^T = \phi(x)^T(\Phi^T\Phi + hI)^{-1}\Phi^T$ so that we may write $\hat{f}$ as a linear smoother, that is

$$\hat{f}(x) = s(x)^T\mathbf{y}.$$

One might find (3.8) hard to interpret. To understand what effect ridge regression has, its useful to relate the technique to principle component analysis. Firstly, remember that the singular value decomposition of $\Phi$ is

$$\Phi = \mathrm{UDV}^T,$$

where U and V is orthogonal marries and D a diagonal matrix. We find that

$$\begin{aligned} \Phi\bar{\mathbf{w}} &= \Phi(\Phi^T\Phi + hI)^{-1}\Phi^T\mathbf{y} \\ &= \mathrm{UD}(\mathrm{D}^2 + hI)^{-1}\mathrm{DU}^T\mathbf{y} \\ &= \sum_j \mathbf{u}_j \frac{d_j^2}{d_j^2 + h}\mathbf{u}_j^T\mathbf{y}. \end{aligned} \tag{3.9}$$

So ridge regression computes coordinates $\mathbf{y}$ with respect to the orthogonal basis $\{\mathbf{u_i}\}$ ([Hasti Tibshirani, 2009] p. 66). Each component in $\mathbf{y}$ along $\mathbf{u}$ is shrieked relative to $d_j^2$

([Rasmussen Williams, 2006] p. 25). But what is $d_j^2$? Form the eigendecomposition of $\Phi^T\Phi$ we find

$$\Phi^T\Phi = \mathrm{V}\mathrm{D}^2\mathrm{V}^T$$

so each $d_j^2$ is an eigenvalue corresponding to the eigenvector $v_j$ of $\Phi^T\Phi$. The eigenvectors $\{\mathbf{v_j}\}$ is called the principle component of $\Phi$ and can be thought a set of orthogonal vectors along witch most of the variance within the data can be explained. The amount of variance along this axis is relative to the size of $d_j^2$ ([Hasti Tibshirani, 2009] p. 64-66). Because of this, ridge regression shrink those coordinate that explain less of the variance compared to those that explain more. The shrinkage of all coordinates is amplified by choosing larger values of $h$.

We may assume that with more data point and better performance we may become more confident in our estimate. Yet, neither regularization nor local regression has any build in tools that quantifies uncertainty about their estimations. In Section 4.1 we will learn that variance is a natural part of Gaussian processes, which gives us a easy way to build confidence bands. In the case of the algorithmic linear smoother models of this section, we need instead to estimate the variance of the model.

## 3.5 Confidence Bands

A confidence band is a confidence interval on the form

$$\hat{f}(x) \pm z_{\alpha/2}\sqrt{\sigma_n^2 \mathrm{Var}[\hat{f}(x)]}$$

mapped onto the response for every value of x, like in figure 10. $z_{\alpha/2}$ is the $\alpha/2$ quantile value of the normal distribution. To build a confidence interval, we need to find a expression of the variance of $\hat{f}$ and an estimate of $\sigma_n^2$. Starting with $\sigma_n^2$, since

$$\mathrm{E}[(\mathbf{y} - \hat{f}(\mathbf{x}))^T(\mathbf{y} - \hat{f}(\mathbf{x}))] = \mathrm{tr}(\mathrm{Var}[\mathbf{y} - \hat{f}(\mathbf{x})]) + (\mathrm{E}[\mathbf{y} - \hat{f}(\mathbf{x})])^T(\mathrm{E}[\mathbf{y} - \hat{f}(\mathbf{x})])$$

where $\mathbf{x}$ and $\mathbf{y}$ are vectors of each data point. We use that $\mathrm{Var}[\mathbf{y}] = \mathrm{Var}[\epsilon] = \sigma_n^2$. So

$$\mathrm{Var}[\mathbf{y} - \hat{f}(\mathbf{x})] = (I - \mathrm{S})^T\sigma_n^2(I - \mathrm{S})$$

using that $\hat{f}(\mathbf{x}) = \mathrm{S}\mathbf{y}$, where S is matrix with $\mathrm{S}_{ij} = s_j(x_i)$. Let $v = \mathrm{tr}(\mathrm{S})$, $\tilde{v} = \mathrm{tr}(\mathrm{S}^T\mathrm{S})$ and $\mathbf{b} = \mathrm{E}[\mathbf{y} - \hat{f}(\mathbf{x})]$. We see then that

$$\mathrm{tr}(\mathrm{Var}[\mathbf{y} - \hat{f}(\mathbf{x})]) = \sigma_n^2\mathrm{tr}(I - 2\mathrm{S} + \mathrm{S}^T\mathrm{S}) = \sigma_n^2(n - 2v + \tilde{v})$$

So

$$\sigma_n^2 = \frac{\mathrm{E}[(\mathbf{y} - \hat{f}(\mathbf{x}))^T(\mathbf{y} - \hat{f}(\mathbf{x}))] - \mathbf{b}^T\mathbf{b}}{n - 2v + \tilde{v}}$$

Let

$$\hat{\sigma}_n^2 = \frac{(\mathbf{y} - \hat{f}(\mathbf{x}))^T(\mathbf{y} - \hat{f}(\mathbf{x}))}{n - 2v + \tilde{v}} \tag{3.10}$$

be our estimator of $\sigma_n^2$. With larger sample size, we may assume that $\mathrm{E}[\hat{f}(\mathbf{x})]$ approaches $\mathrm{E}[\mathbf{y}]$, so $\mathbf{b} \to \mathbf{0}$. So for large n we have $\mathrm{E}[\hat{\sigma}_n^2] = \sigma_n^2$. Because of this, we may assume that (3.10) is a good estimator of $\sigma_n^2$. Since $\hat{f}$ is on the form (3.1), we see that

$$\mathrm{Var}[\hat{f}(x)] = s(x)^T\mathrm{Var}[y]s(x) = \sigma_n^2\|s(x)\|^2$$

If we use (3.10) to estimate $\sigma_n^2$ we find that the confidence interval of $\hat{f}$ is of the form

$$\hat{f}(x) \pm z_{\alpha/2}\hat{\sigma}_n\|s(x)\| \tag{3.11}$$

If we would instead like a confidence band for a noisy predictions, $\hat{y}$, we may use that $\text{Var}[\hat{y}] = \text{Var}[\hat{f}(x)] + \text{Var}[\epsilon] = \sigma_n^2\|s(x)\|^2 + \sigma_n^2$. This is called a *prediction interval* ([James, Witten, Hasti, Tibshirani, 2021] pp. 82), and is given by

$$\hat{y} \pm z_{\alpha/2}\hat{\sigma}_n\sqrt{1 + \|s(x)\|^2} \tag{3.12}$$

In Bayesian inference we build confidence intervals in the same way, only that we don't need to estimate $\sigma_n^2$ since it is already build into the model and $\text{Var}[\hat{f}(x)]$ has a different form.

# 4 Bayesian Inference

A common way of data modeling is by Bayesian inference, where we first specify our prior beliefs about the nature of the function, then update our beliefs after observing data. More specifically, our prior beliefs – which we call priors – specify what kind of function we think $f$ is and what its variance and mean is. With more data, we can become more certain about the behavior of $f$, especially around each data point. We call the distribution of $f$ before observation the prior, and the distribution of $f$ after observations the posterior. Both the posterior and prior specify the probability of any one possible function being the true function. In particular, our posterior distribution gives us a space of functions that are most likely to be the true function, given the information from the observations we have. Of the infinite number of these functions, the most likely of these functions is the mean of our distribution, $\bar{f}$. If we assume that the observations are noise free, the only possible value of all function at any particular data point, $(x, y)$, is $f(x) = y$. Because of this, any possible function must intersect all data point. However, the focus of this section is going to be on the case where we have noisy observations.

To better understand the concepts of Bayesian Inference it might be useful to know that, according to neurologist Anil Seth, we perceive the world in a way closely tied to Bayesian inference [Seth, 2021]. Consider this example: When walking outside, your prior knowledge would make you assume that it is very unlikely to observing a gorilla walking down the street. So if you where to observe a furry creature in the distance, you would probably assume it to be a rather large dog instead of a Gorilla. When you come closer, you see that it is even larger than expected and has a humanoid shape. Because of this you update your posterior prediction and instead think it is a human in a gorilla costume. When you get closer still, you see that you where mistaken and that it is indeed a gorilla – now you should probably run away. If the circumstances had change, and you are in a zoo instead of walking on the street, your prior estimated probability of observing a gorilla is much higher, and therefor you would make the correct prediction of observing a gorilla with less sensory information. In this way, our predictions are both dependent on our prior beliefs and the noisy data we perceive. This applies to all sensory prediction, from predicting the path of a tennis ball, to decoding the subtle ques from of your date and guess if he/she likes you or not. This insight is the foundation on which Bayesian inference regression models are build upon.

## 4.1 Linear Bayesian Inference

We are still looking at the case where we assume that the true distribution of $y$ is in the form (2.1). Instead of merely approximating $f$, we would like to make assumptions about $f$ and build

a model. In this section, we are going to assume that $f$ takes the form of a Gaussian process. There is two main ways of interpreting Gaussian Processes (GP) ([Rasmussen Williams, 2006] pp. 7). The view that is the easiest to understand is called the weighed space view, where we assume that $f(x)$ is some weighed linear combination of some transformation of $x$. That is

$$f(x) = \phi(x)^T \mathbf{w}, \tag{4.1}$$

where $\phi(x)$ and $\mathbf{w}$ is p dimensional. In Bayesian inference, before we make any predictions, we need to make some assumptions about $\mathbf{w}$, what we call a prior. In GP's, we assume that $\mathbf{w}$ is normally distributed. Before looking at the data, we need to specify the parameters of $\mathbf{w}$. In particular, we need to define a covariance matrix, $\Sigma_p$, on the weights. We can safely take the mean we to be zero, since it does not influence the behavior of $f(x)$ inside the interval our data is located on. As we will see, our estimated function, $\hat{f}$, will naturally adjust it's mean to the most likely value at any one point after performing inference on the data. Form this, the prior becomes

$$\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \Sigma_p),$$

where $\mathbf{0}$ denotes the zero vector. The most important feature of $f$ is it's covariance matrix, which describes the behaviour of $f$. From (4.1) and the distribution of $\mathbf{w}$, we see that $\text{Cov}[f(x), f(x')] = \phi(x)^T \Sigma_p \phi(x')$. We use a covariance function, $k(x, x')$, to calculate the covariance between $f(x)$ and $f(x')$. Covariance functions can be though of as quantifying the similarity between two data points, $x$ and $x'$ ([Rasmussen Williams, 2006] pp. 79). Under the assumption that points that a closer to each other are also more similar to each other. In this perspective, $k(x, x')$ can be though of an inverse distance measure and has the same role as the weight function in local regression from 3.2. Indeed, the names covariance function and kernel is often used interchangeably, which is the reason that they are both denoted with $k$ ([Rasmussen Williams, 2006] pp. 12). In the weight space view, because of the form of the covariance function, it is natural to take $k$ to be a inner product in a higher dimensional feature space by $k(x, x') = \phi(x)^T \Sigma_p \phi(x')$. For example, consider the case when $\phi(x) = \begin{bmatrix} 1 & x \end{bmatrix}$. Then $f(x) = w_1 + w_2 x$, so $f$ can be any linear function. However, since we draw $\mathbf{w}$ from a normal distribution, $w_1$ and $w_2$ is unlikely to be far away from 0. That is, the density of $f$ is larger around zero than further away. Figure 8 shows 20 such realizations.

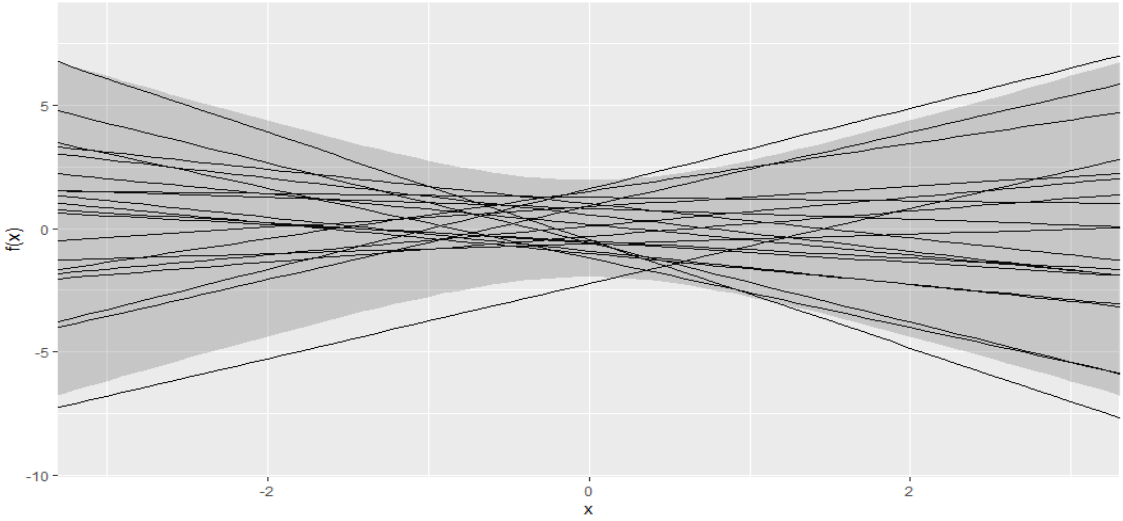



Figure 8: 20 realizations of a linear function with weights drawn from a Gaussian distribution with mean 0 and variance 1. The gray area is a 95% confidence band.

In this picture, the weights is assumed to be zero mean and have a variance of one, that is $\mathbf{w} \sim \mathcal{N}_2(\mathbf{0}, I_2)$. Because of this, the confidence bands is calculated with $\hat{f}(x) \pm z_{\alpha/2}\sqrt{\text{Var}[\hat{f}(x)]}$, where $\text{Var}[\hat{f}(x)] = k(x,x) = [1,x] \cdot [1,x] = 1 + x^2$.

It is important to note that this function is a parametric function, since we have predefined to number of feature we want to use. In this case, we use the feature $\phi_1(x) = 1$ and $\phi_2(x) = x$. In the non-parametric approach, which is the focus of this text, the number of features is tied to the number of data points. This requires the use of an infinite dimensional feature space. In this case, calculating $k(x,x') = \phi(x)^T \Sigma_p \phi(x')$ is obviously infeasible, since $\phi(x)$ is of infinite dimensions. Instead, we more generally require $k$ to be a symmetric and positive semi-definite function. By symmetric we need $k(x,x') = k(x',x)$. A function is said to be positive semi-definite if the Gram matrix, K, defined by $K_{ij} = k(x_i, x_j)$ is positive semi-definite ([Bartlett, 2008] pp. 2). If $k$ is a covariance function, we call K a covariance matrix ([Rasmussen Williams, 2006] pp. 80). Since $k$ is symmetric, the matrix K must also be symmetric. These properties are obviously satisfied when $k$ is on the form of a inner product. An example of a covariance function defined on a infinite dimensional feature space is the Gaussian kernel, also called the radial basis function ([Rasmussen Williams, 2006] pp. 14)

$$k(x,x') = \sigma_f e^{-\frac{(x-x')^2}{2h^2}}. \tag{4.2}$$

We use $\Sigma = \sigma_f I$, an infinite dimensional matrix. $\sigma_f$ quantifies the variance of $f$ and must not be confused with $\sigma_n$, which is the variance of the error $\epsilon$. $h$ can be recognized as the bandwidth that was discussed in 3.2. In the probabilistic view on machine learning, $h$ can be thought of as specifying the distance at which each point is correlated. If we take $h$ to be large, then a higher number of pairs $x_i$ and $x$ has a non-zero covariance.

It is not at all obvious how (4.2) defines a inner product on an infinite dimensional feature space. Indeed, the Gaussian kernel does not seem related to inner products at all. However, this

could not be further from the truth. In Section 6.1, we will explore the relationship between the Gaussian kernel and inner products, and how this relationship is not unique, but is instead general to a large class of symmetric and positive definite functions.

Returning to parametric case when $f$ is on the form of (4.1). Inference in the Bayesian linear model is based on the posterior distribution over the weights given by the conditional likelihood of $\mathbf{w}$ given our data. This is computed using Bayes rule ([Rasmussen Williams, 2006] pp. 9)

$$p(\mathbf{w}|\mathbf{y},\mathbf{x}) = \frac{p(\mathbf{y}|\mathbf{x},\mathbf{w})p(\mathbf{w})}{p(\mathbf{y}|\mathbf{x})}. \tag{4.3}$$

By estimating each of the terms on the right hand side in (4.3), we may obtain the posterior, $p(\mathbf{w}|\mathbf{y},\mathbf{x})$. For $p(\mathbf{y}|\mathbf{x},\mathbf{w})$, we use the likelihood to get

$$\begin{aligned} p(\mathbf{y}|\mathbf{x},\mathbf{w}) &= \prod_{i=1}^{n} p(y_i|x_i,\mathbf{w}) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_n} \exp\left(-\frac{(y_i - \phi(x_i)^T\mathbf{w})^2}{2\sigma_n}\right) \\ &= \frac{1}{(2\pi\sigma_n^2)^{n/2}} \exp\left(-\frac{1}{2\sigma_n^2}|\mathbf{y} - \phi(\mathbf{x})^T\mathbf{w}|^2\right) \sim \mathcal{N}(\phi(\mathbf{x})^T\mathbf{w}, \sigma_n^2 I). \end{aligned} \tag{4.4}$$

$p(\mathbf{y}|\mathbf{x})$ is the marginal likelihood and is given by

$$p(\mathbf{y}|\mathbf{x}) = \int p(\mathbf{y}|\mathbf{x},\mathbf{w})p(\mathbf{w})d\mathbf{w}. \tag{4.5}$$

As the term $p(\mathbf{y}|\mathbf{x})$ suggests, this is the probability the true model is $\mathbf{y}$, given the data, $\mathbf{x}$, we have observed. This is often called the evidence, since it can be used to evaluate the performance of our regression model. If $p(\mathbf{y}|\mathbf{x})$ is low, it's unlikely that we have found a good predictive model. For this reason, it plays a similar role to that of cross-validation. Combining (4.4) and (4.5) in (4.3) and using that $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \Sigma_p)$ we get [Rasmussen Williams, 2006]

$$p(\mathbf{w}|\mathbf{y},\mathbf{x}) \propto \exp\left(-\frac{1}{2\sigma_n}(\mathbf{y} - \phi(\mathbf{x})^T\mathbf{w})^T(\mathbf{y} - \phi(\mathbf{x})^T\mathbf{w})\right) \exp(-\frac{1}{2}\mathbf{w}^T\Sigma_p^{-1}\mathbf{w}). \tag{4.6}$$

It so happens that this pdf is closely related to ridge regression. After we take the negative logarithm of $p(\mathbf{w}|\mathbf{y},\mathbf{x})$, the first term can be viewed as a least square penalty and therefor penalizing high bias. Since $\Sigma_p$ is symmetric positive definite, the other term can be viewed as a weighted norm of the coefficients $\mathbf{w}$. From the theory of relaxation, we know that this punishes complexity, and therefor penalizes high variance. This relationship going to be further explored in Section 6.

With a density of $\mathbf{w}$ given our date $\mathbf{x}$ and $\mathbf{y}$, we have information about how probable different forms of $\mathbf{w}$ is, and by extension, different forms of $f(x)$. The most probable form of $f$ is the one that uses the $\mathbf{w}$ with the highest probability. This is found by maximizing (4.6) w.r.t $\mathbf{w}$ and is called the maximum a posterior (MAP). We find that the MAP is $\bar{\mathbf{w}} = \sigma_n^{-2}(\sigma_n^{-2}\phi(\mathbf{x})\phi(\mathbf{x})^T + \Sigma_p^{-1})^{-1}\phi(\mathbf{x})\mathbf{y}$, where the bar denotes the maximum. This can be further simplified by using kernels. Defining $\mathrm{K} = \Phi^T\Sigma_p\Phi$ we may write $\bar{\mathbf{w}} = \Sigma_p\Phi(\mathrm{K} + \sigma_n^2 I)^{-1}\mathbf{y}$. After some simplification of (4.6) it is easy to see that ([Rasmussen Williams, 2006] pp. 9)

$$p(\mathbf{w}|\mathbf{y},\mathbf{x}) \sim \mathcal{N}\left(\bar{\mathbf{w}}, (\mathrm{K} + \sigma_n^2 I)^{-1}\right). \tag{4.7}$$

So the expected value of the posterior is also the MAP. We may use the posterior to predict future outcomes, $\mathbf{y}^*$, with any input $\mathbf{x}^*$. In particular, we know that the most probable value of

$f(x^*)$ is $\bar{f}(x^*) = \phi(x^*)^T \bar{\mathbf{w}}$, as given by the MAP. Because

$$\phi(x^*)^T \Sigma_p \Phi = \sum_{i=1}^{n} k(x^*, x_i)$$

We see that $\bar{f}$ has the form

$$\bar{f}(x^*) = \sum_{i=1}^{n} \alpha_i k(x^*, x_i), \tag{4.8}$$

where $\boldsymbol{\alpha} = (K + \sigma_n^2 I)^{-1}\mathbf{y}$. This form going to be important in Section 6 – where I compare the two approaches – because it is possible to show that both the MAP and solutions of regularization has this form. If we take $\alpha_i = \left(\sum_{j=1}^{n} k(x_i, x_j)\right)^{-1}$, we get the N-W estimator, so local regression has the same form.

## 4.2 Gaussian Processes

An alternative method of reaching equivalent results as with the weight space view is through the function space view. In this view, $f$ is no longer assumed to be parametric. That is, we assume that $\phi$ is of infinite dimension. Because of this, $k$, our covariance function, needs to be computed on an infinite dimensional feature space, like described with the Gaussian kernel. While assuming $f$ has a non-parametric form, we may use Gaussian processes (GP) to describe the distribution over functions. Formally ([Rasmussen Williams, 2006] pp. 13):

**Definition 4.1.** *A Gaussian process is a collection of random variables, any finite number of which have a joint Gaussian distribution.*

For a more general approach, assume that we use a d-dimensional input vector, $\mathbf{x}$, so that our data is $\mathrm{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]$. We define a mean function $m(\mathbf{x}) = \mathrm{E}[f(\mathbf{x})]$ and a covariance function $\mathrm{Cov}[f(\mathbf{x})]$. It was previously stated that the mean function does not influence the behavior of $\hat{f}$. This holds true in areas of interpolation, where data points has been observed, but is not the whole truth for extrapolation and on the boundary of our function. For any prediction point far away from our data we have little or no information about the nature of $f(x)$, so our best guess reverts back to the prior. That a problem may arise around the boundary is not obvious in the data introduced in Section 2.1, since this data is clearly centered around zero. A better example is shown in Figure 9, which depicts the logarithm of the wage of Canadian male workers in 1971 with a common education (13 years) relative to their age.

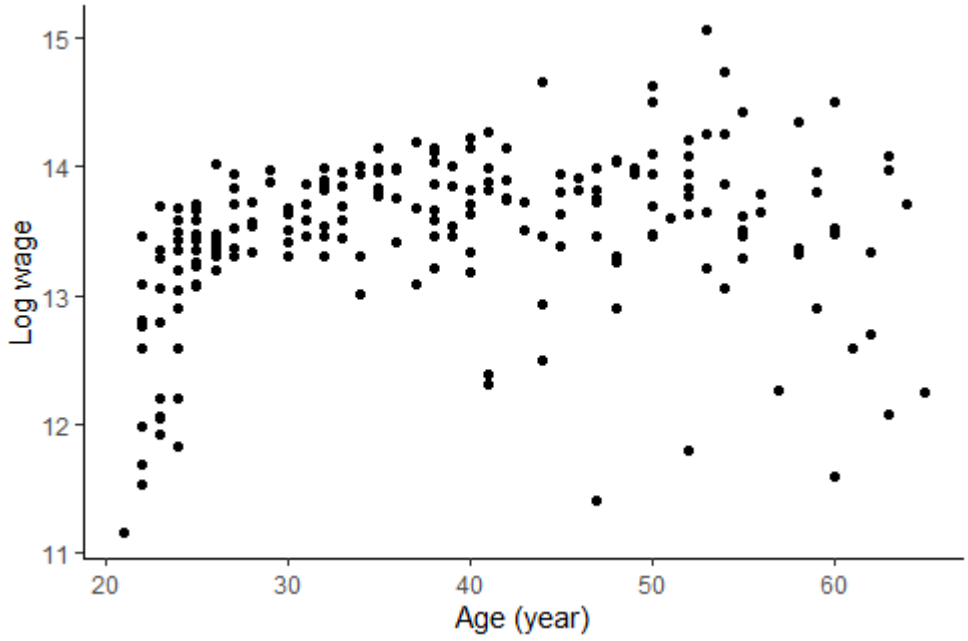



Figure 9: Logarithm of wage relative to age

We see that the data is clearly centered above zero. If we where to assume that the prior mean is zero, the predicted function would need to take a sharp turn from zero up to the predicted mean of the function on the left side of the boundary. Similarly, the function would quickly regress back to zero on the right boundary. An example of this is shown in Figure 15 in Section 5.3. To account for this, we allow $m(\mathbf{x})$ to be non-zero. From this we have that

$$f(\mathbf{x}) \sim \mathcal{N}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x})).$$

So $f$ has a multivariate Gaussian distribution. As usual, we assume that we have noisy observations, like in (2.1), so we have $\mathbf{y} \sim \mathcal{N}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}) + \sigma_n^2 I)$. We want to predict observations in new points $\mathrm{X}_* = [\mathbf{x}_1^*, \ldots, \mathbf{x}_m^*]$, with $\mathbf{f}_* = [f(\mathbf{x}_1^*) \ldots f(\mathbf{x}_m^*)]$. $f_*$ should have the same distribution as $f$, so we may assume that

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left( \begin{bmatrix} m(\mathbf{x}) \\ m(\mathbf{x}^*) \end{bmatrix}, \begin{bmatrix} k(\mathrm{X}, \mathrm{X}) + \sigma_n^2 I & k(\mathrm{X}, \mathrm{X}_*) \\ k(\mathrm{X}_*, \mathrm{X}) & k(\mathrm{X}_*, \mathrm{X}_*) \end{bmatrix} \right).$$

We may use conditional likelihood for multivariate Gaussian distribution to find [Rasmussen Williams, 2006]

$$\bar{\mathbf{f}}^* = \mathrm{E}[\mathbf{f}^* | \mathrm{X}, \mathbf{y}, \mathrm{X}_*] = k(\mathrm{X}_*, \mathrm{X})[k(\mathrm{X}, \mathrm{X}) + \sigma_n^2 I]^{-1}(\mathbf{y} - m(\mathbf{x})) + m(\mathbf{x}^*) \tag{4.9}$$

and

$$\mathrm{Cov}[\mathbf{f}_*] = k(\mathrm{X}_*, \mathrm{X}_*) - k(\mathrm{X}_*, \mathrm{X})[k(\mathrm{X}, \mathrm{X}) + \sigma_n^2 I]^{-1} k(\mathrm{X}, \mathrm{X}_*). \tag{4.10}$$

This is the exact same result as in (4.8), because $k(\mathrm{X}, \mathrm{X}) = \mathrm{K}$ and $k(\mathrm{X}_*, \mathrm{X}) = \phi(x^*)^T \Sigma_p \Phi$.

The posterior has reduced uncertainty around each observation. In particular, if we assume noise free observations, we know that $f$ must interpolate each data point, so we force $\hat{f}$ to do

the same. An example of this is shown in Figure 10, which depicts samples from both a prior and a posterior of a Gaussian distribution.

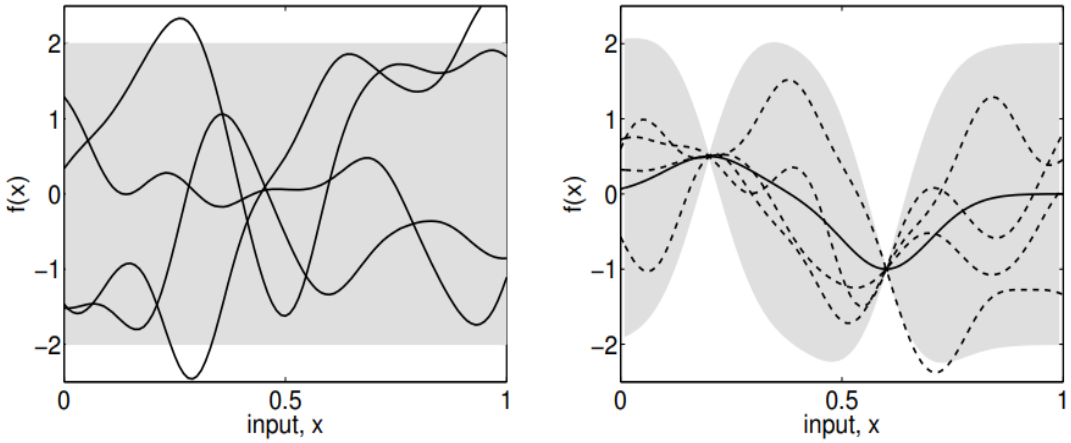



Figure 10: Panel a) shows four samples drawn from a Gaussian prior distribution. Panel b) shows 4 samples drawn from the posterior of the same distribution (dashed line) and the MAP (bold line). The gray area in both panels shows a 95% confidence band for both densities. (Source: [Rasmussen Williams, 2006] pp. 3)

Because of this effect, with more observation we limit the space of likely forms of $\hat{f}$, enabling us to be more certain about it's true form, given that our assumptions about it in the prior is correct. We may use the marginal likelihood to evaluate the validity of our assumptions.

## 4.3 Marginal Likelihood

Marginal likelihood defined by (4.5) is a tool that helps us quantify the performance of our model given our data. This is useful when comparing different models to each others and when tuning hyper-paramters, such as the bandwidth $h$ and $\sigma_n$. We would like a different form of (4.5) that is more easy to calculate. Since $p(\mathbf{f}|\mathrm{X}) \sim \mathcal{N}(\mathbf{0}, \mathrm{K})$ we have

$$\log p(\mathbf{f}|\mathrm{X}) = -\frac{1}{2}\mathbf{f}^T\mathrm{K}^{-1}\mathbf{f} - \frac{1}{2}\log|\mathrm{K}| - \frac{n}{2}\log(2\pi).$$

Combining this with this the fact that $\mathbf{y}|\mathbf{f} \sim \mathcal{N}(\mathbf{f}, \sigma_n^2 I)$ we find that ([Rasmussen Williams, 2006] pp. 19)

$$\log p(\mathbf{y}|\mathrm{X}) = -\frac{1}{2}\mathbf{y}^T(\mathrm{K} + \sigma_n I)\mathbf{y} - \frac{1}{2}\log|\mathrm{K} + \sigma_n I| - \frac{n}{2}\log 2\pi. \tag{4.11}$$

This equation has an important relationship to the bias-variance trade-off principle. In fact, each term in (4.11) can be related to the terms in the bias-variance equation (2.3). The first term, $\mathbf{y}^T(\mathrm{K} + \sigma_n I)\mathbf{y}$, is called the square error and is relates to the bias. It grows when the distance from the function to the data points is large. The second term, $\log|\mathrm{K} + \sigma_n I|$, is called model complexity and relates to variance. The determinant of K grows when the prediction function

is complex, causing more variance. A different name for this term is the Occam factor, named after William of Occam. Occam is known for Occams razor, a principle which states that one should always prefer the simplest explanation to a problem. This coincides with the role of the Occam factor, since the term punishes models which increases complexity without decreasing the square error – so simpler functions are preferred [Hennig, 2020]. Lastly, $\frac{n}{2}\log 2\pi$, is not related to K and can be viewed as a irreducible error.

Because of this relationship, marginal likelihood is maximal when the model is fine tuned, so that the bias-variance trade-off is in it's Goldilocks zone. For this reason, we would like to maximize (4.11) with respect to the hyper-parameters. Unfortunately, this optimization problem does not have a analytical solution, like we found when optimizing (4.6). Instead, we need to gradient descent methods like BFGS to search for optimal solution. To optimize the hyper-parameters is the "learning" part of machine learning. In the next section, we will see how this learning may be done in practice.

# 5 Comparison of Gaussian Processes and Local Regression with examples

We now have enough insight into local regression and Gaussian processes to use them on the first data set, which where presented in Figure 1, and the second data set, that where presented in Figure 9.

## 5.1 Implementation of Local Regression

The implementation of local regression, using has the N-W, tree important steps. Firstly, we need to optimize the bandwidth. Then we fit the model using the optimal bandwidth. Lastly, we estimate total variance and make a 95% confidence band. Since the estimated function on the form (3.1), it is linear and we can therefor use LOOCV to efficiently estimate the error with ([Hasti Tibshirani, 2009] pp. 161)

$$\text{LOOCV}(\hat{f}) = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_i - \hat{f}(x_i)}{1 - s_i(x_i)}\right)^2, \tag{5.1}$$

where $s_i(x_i)$ is a smoothing function from Section 3. $s(x)$ is with (3.4), by using the N-W kernel estimator. Tuning the bandwidth amounts using a optimization algorithm to find the $h$ that minimizes (5.1). With the smoothing vector, we calculate $\hat{f}$ with (3.3). Lastly, we estimate the variance of $\mathbf{y}$ with (3.10) and estimate the confidence interval of $\hat{f}$ with (3.11). Alternatively, we can estimate the prediction interval with (3.12).

## 5.2 Implementation of Gaussian Processes

Similar to local regression, the implementation of GP regression has 3 parts. Firstly, we need to optimize the parameters. Unlike local regression, which need to only optimize the bandwidth, GP regression require us to optimize 4 hyper-parameters. This include the bandwidth, but also the variance of $\epsilon$ in (2.1), $\sigma_n$, the variance of $f$, $\sigma_f$, from (4.2) and lastly the mean, $m(x)$. We again use a optimization algorithm. The other two steps are more closely tied than in local regression. We simply use (4.9) to find our estimated function and (4.10) to find the covariance matrix of our estimated function. The variance vector of the estimated function is the diagonal of the covariance matrix. Then we make confidence bands with $\hat{f}(x) \pm z_{\alpha/2}\sqrt{\text{Var}[f]}$.

## 5.3 Practical comparison

In all these regressions, a confidence band for $\hat{y}$ is used. When used on the first data set, we get the following the regression graph when using N-W kernel estimator.

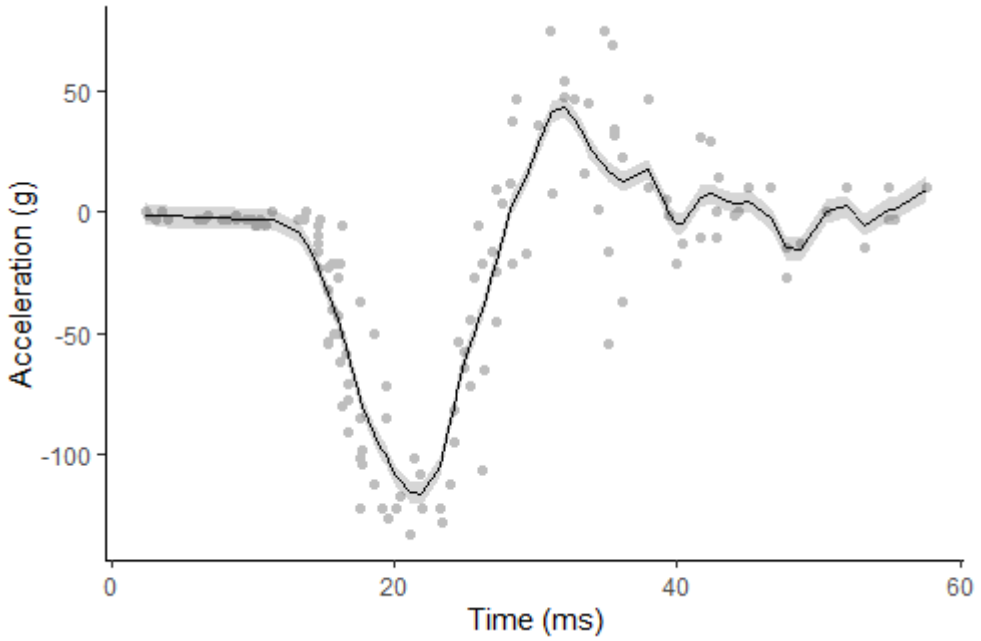



Figure 11: N-W kernel estimator used on head acceleration data. The bold line is the estimated function, while the gray area is a 95% confidence band.

The MSE of this regression is estimated to be 444.485 and the running time is 0.872 seconds. While when using GP we get the following result

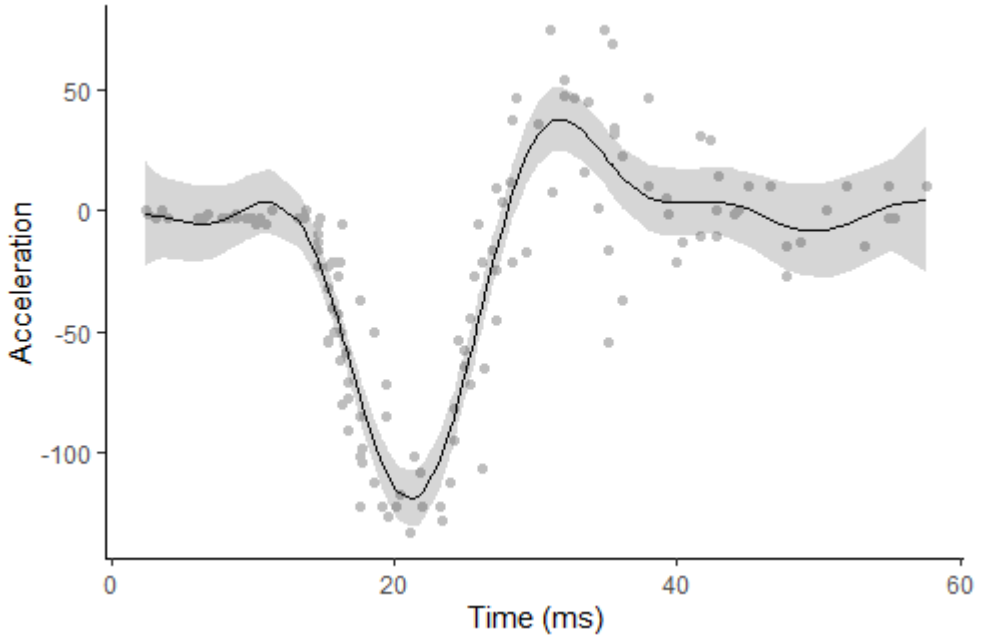



Figure 12: Gaussian Process regression used on head acceleration data. The bold line is the MAP, while the gray area is a 95% confidence band.

This time, the regression is 466.972 and running time is 5.392 seconds. Maybe the most visually obvious difference is the rough shape of the curve generated by the N-W kernel estimator, compared to the more smooth shape of Gaussian process curve. This is to be expected, since the Gaussian process is made using a series of correlated normal distributed variables. Because of this, each point along the function is unlikely to make any large jumps in value relative to the previous point. Kernel regression does not have this restriction. This higher degree of freedom is reflected in the slightly stronger performance of local regression than GP. This difference in performance is maybe magnified by the fact that the GP makes the assumption that the sample variance, $\sigma_n^2$, is constant in all time intervals. This assumption is probably not justifiable, since the sample variance seems to be quite small at the beginning and then increases firstly at the first jump around time 15, and then increases even further when the acceleration seems to revert back to zero, at around time 30. Since local regression does not make assumptions about sample variance, this problem does not affect it's result.

While the GP regression loses some performance by making the sample variance assumption, the same assumption significantly improves the quality of it's confidence bands. The main difference between the confidence bands of the two methods is their size. The confidence bands of the GP regression seems to correctly cover closer to 95% of the data, unlike local regression, which covers a very small portion of the data. This suggest that method for which the confidence bands estimated with proved to be quite ineffective. Unlike GP's, local regression is free to choose any method for estimating confidence bands, some of which might perform much better than the method chosen here.

Lastly, the local regression was significantly faster than the GP regression: 0.872 seconds compared to 5.392 seconds. This is due to the fact that GP's needs to first calculate the matrix $(\mathrm{K} + \sigma_n^2 I)$, then invert it. Calculate a matrix $n \times n$ takes $\mathcal{O}(n^2)$ time, while inverting it takes

$\mathcal{O}(n^3)$ time. Local regression need only to calculate K, not invert it. This causes local regression to have a overall quadratic time complexity, while GP's have a cubic time complexity, which is reflected in their computation times. Techniques to approximate the inverse of the covariance matrix which improves the computation time of GP's. One such technique is sparse greedy Gaussian process regression, which has a computation time of $\mathcal{O}(m^2 n)$, where n is the sample size and $m \ll n$ [Smola Bartlett, 2000].

When we perform the two regression techniques on the wage data set we get the following results

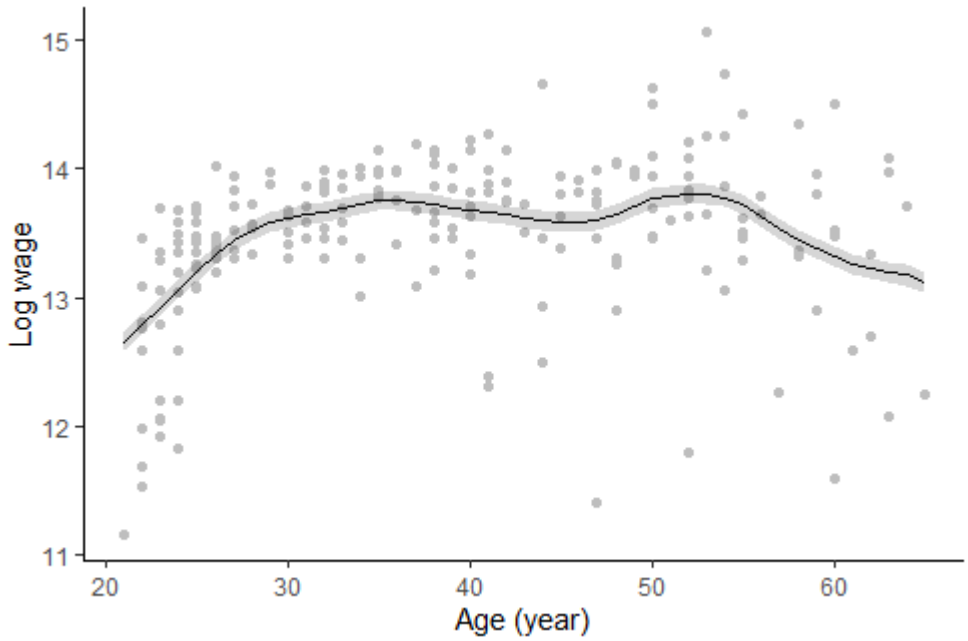



Figure 13: N-W kernel estimator regression used on logarithm of wage relative to age data set. The bold line is the estimated function, while the gray area is a 95% confidence band.

for the N-W kernel estimator, which has a MSE of 0.282 and a running time of 1.574 seconds. For the GP regression we get

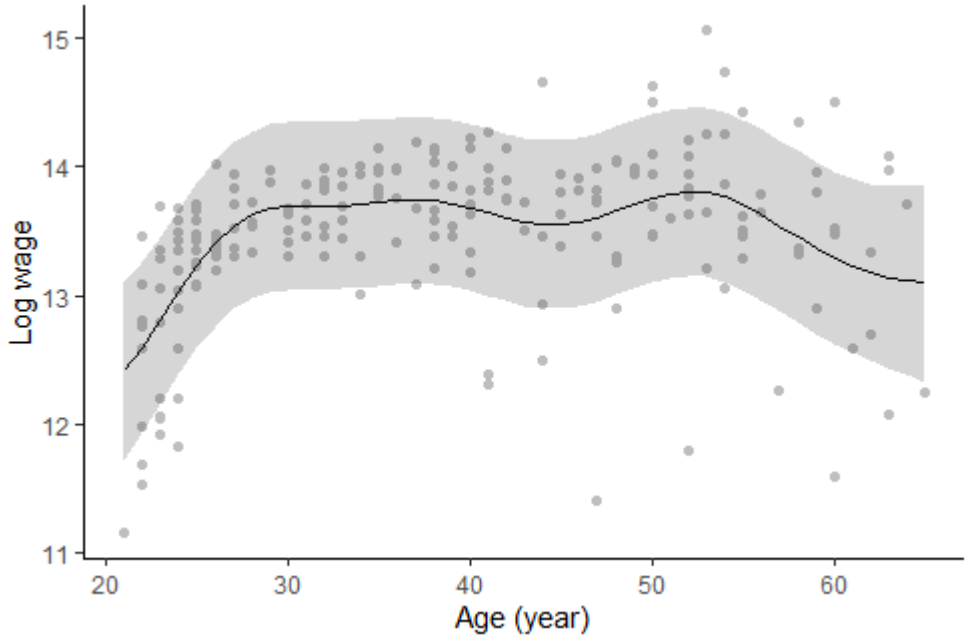



Figure 14: Gaussian process regression used on logarithm of wage relative to age data set. The bold line is the estimated function, while the gray area is a 95% confidence band.

This time, the MSE is 0.274 and the running time is 17.263 seconds. Surprisingly, the GP regression performed slightly better than the N-W kernel estimator. This might be because the assumption of a constant variance seems to be more accurate for this data set then for the head acceleration data set. The difference in computation time where significantly larger with this data set then with the head acceleration data set, due to the difference in size (133 data points compared to 205). Because of the quick growth, it would be computationally infeasible to perform this implementation of GP regression on a data set of a larger magnitude, say 1000 data points, unlike kernel regression, which would have a more acceptable computation time.

Lastly, as promised in Section 4.1, we can compare the performance of a zero mean GP regression to our current model

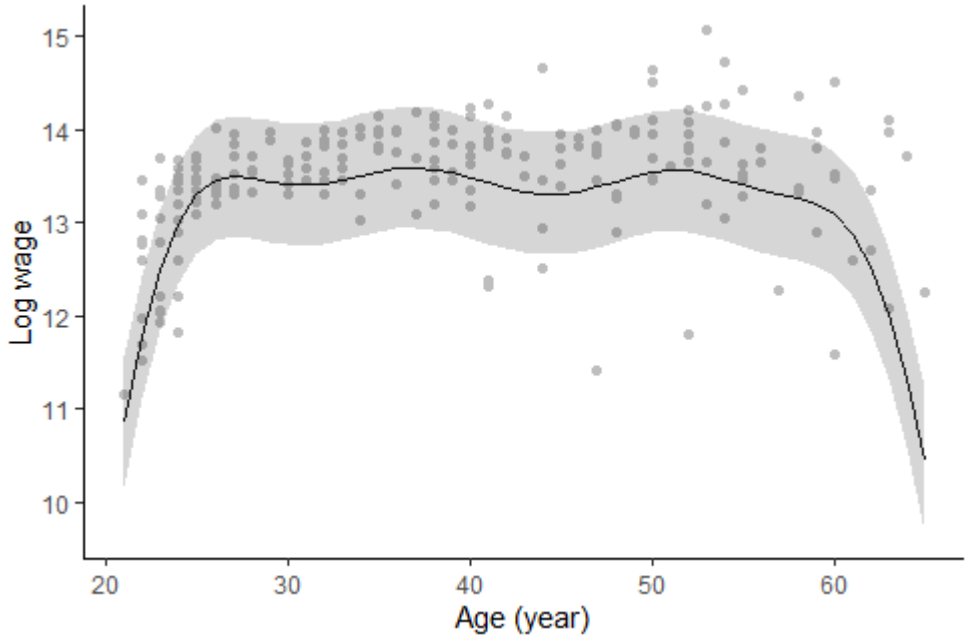



Figure 15: Same GP regression as in 14, except that the model uses a zero mean instead of an optimized mean.

As expected, the regression line takes a sharp turn up from and down to zero at the boundaries. This regression has a MSE of 0.389, so it performs significantly worse than the non-zero mean model.

The implement of regularization has been excluded from this text because of it's close relationship to GP regression. This relationship is going to be investigated more closely in the next section.

# 6 Theoretical Comparison of Regularization and GP's

This section is going to be an exploration of an important subclass of regularization problems on the form (3.5) that are generated by a positive definite kernel $k(x, x')$. The motivation for doing this is to relate solutions of our new regularization problem to that GP's, since both problems have solutions on the form of linear combinations of kernel functions. In Section 4 we saw that some kernels can be expressed as inner products. To obtain the relationship between regularization and GP's we need to further explore this property of kernels.

## 6.1 Kernels as Inner Products in a Possibly Infinite Dimensional Space

Kernels are central to the behavior of many non-parametric regression methods, including all techniques covered in this text. This is the reason for their name, kernel, meaning central or most important part of something. In local regression, kernels has the role of a inverse weight function. In Gaussian processes, kernels has the role of a covariance function. However, in regularization the role of kernels has not yet been made obvious. Understanding the role of kernels in a generalized form of regularization is essential in comparing Gaussian processes to

regularization. Both share the view that kernels are inner products in a higher dimensional space. Using a basis transformation into a feature space $\phi : \mathbb{R} \to \mathbb{R}^p$ and a positive definite $p \times p$ matrix $\Sigma_p$, we may use the definition of covariance functions to take $k(x, x') = \phi(x)^T \Sigma_p \phi(x')$, where $x, x' \in \mathbb{R}$ are two data points. If we allow the dimension of $\phi(x)$ to be very high, computing a kernel on this form is close to infeasible. Fortunately, many kernels allow us to simplify the computations without sacrificing complexity. This fact is often referred to as the kernel trick [Hasti Tibshirani, 2009]. For example, if we have a vector input, $\mathbf{x}$, we may consider the simplest form, where $\phi(\mathbf{x}) = \mathbf{x}$ and $\Sigma_p = I$, then $k(x, x') = \mathbf{x}^T \mathbf{x}'$. Going back to the case where $x$ is one dimensional, we may define $\phi(x) = [1, \sqrt{2}x, x^2]$. Then

$$k(x, x') = [1, \sqrt{2}x, x^2] \cdot [1, \sqrt{2}x', x'^2] = x^2 x'^2 + 2xx' + 1 = (xx' + 1)^2.$$

This can be further generalized to the case where $\mathbf{x}$ is of any dimension and the polynomial is of p degree. Then we have the polynomial kernel $k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}' + 1)^p$. The cost of computing $k$ on this form is relative to the dimensions of the input space, not the feature space, since most of the computation time comes from computing the inner product $\mathbf{x}^T \mathbf{x}'$ [Felzenszwalb, 2017]. This allows us to expand the feature space, and with it the complexity, without much increase in the computation cost. Still, the polynomial kernel has it's limitations. If we attempt to use an infinite dimensional feature space, $p \to \infty$, then we are no longer able to compute the polynomial kernel. Surprisingly, this is not always the case. More impressive yet, expanding to an infinite feature space can decrease the computation cost for computing the kernel. To give an example of this, consider the case where we want to do a regression on the interval $[c_{min}, c_{max}] \in \mathbb{R}$. We may choose a set of evenly spaced points, $\{c_1, \ldots, c_F\}$, on this interval and use the basis function $\phi(x) = \left[\exp\left(-\frac{(x-c_1)^2}{2h^2}\right), \ldots, \exp\left(-\frac{(x-c_F)^2}{2h^2}\right)\right]$, so that $F$ is the number of features. If we take $\Sigma_F = \sigma^2 \frac{c_{max} - c_{min}}{F} I$, then

$$\begin{aligned} k(x, x') &= \frac{\sigma^2 (c_{max} - c_{min})}{F} \sum_{k=1}^{F} \phi_k(x) \phi_k(x') \\ &= \exp\left(-\frac{(x - x')^2}{4h^2}\right) \frac{\sigma^2 (c_{max} - c_{min})}{F} \sum_{k=1}^{F} \exp\left(-\frac{(c_k - \frac{1}{2}(x + x'))^2}{h^2}\right). \end{aligned}$$

By expanding to a infinite number of features, $F \to \infty$, while still keeping each $c_k$ evenly spaced, we may recognize the sum as a Riemann sum, which is by definition $\int_{c_{min}}^{c_{max}} \exp\left(-\frac{(c - \frac{1}{2}(x+x'))^2}{h^2}\right) dc$. Furthermore, if we expand the interval $[c_{min}, c_{max}]$ to all of $\mathbb{R}$ with $c_{min} \to -\infty$ and $c_{max} \to \infty$, we get a Gaussian integral with the well know solution of $\sqrt{2\pi}h$. So we get that $k(x, x') = \sqrt{2\pi} h \sigma^2 \exp\left(-\frac{(x-x')^2}{4h^2}\right)$ [Hennig, 2020]. To summarize, we started out with a finite number of Gaussian features and by expanding to a infinite number of such features we increased the complexity of the kernel while simultaneously reducing the computation cost to that of computing a single Gaussian function, essentially obtaining the best of both worlds.

The Gaussian kernel is not alone in it's relationship to inner products in a infinite dimensional feature space. Shortly, we will see how Mercer's theorem allows us to write positive definite kernels as a sum of eigenfunctions, which in turn allows us to express a kernel as a inner product. We define any function $\psi$ that obeys

$$\int k(x, x') \psi(x) d\mu(x) = \lambda \psi(x)$$

to be a eigenfunction of $k$ with eigenvalue $\lambda$ with respect to a measure $\mu$. For example, if $x$ has a known probability measure, then we make take $d\mu(x) = p(x)dx$. If this is not the case,

we often use the Lebesgue measure [Rasmussen Williams, 2006]. There exists a infinite number eigenfunctions. Each of these functions are orthogonal, and may further choose to normalize them such that

$$\int \psi_i(x)\psi_j(x)d\mu(x) = \delta_{ij}, \tag{6.1}$$

where $\delta_{ij}$ is the Kronecker delta function, which is one when $i = j$ and zero otherwise. Mercer's theormen states that

**Theorem 6.1.** *(Mercer's theorem). Let $(\mathbb{X}, \mu)$ be a finite measure space and $k : \mathbb{X} \times \mathbb{X} \to \mathbb{R}$ be a continuous and positive definite kernel on $\mathbb{X}$ and $T_k : L_2(\mathbb{X}, \mu) \to L_2(\mathbb{X}, \mu)$ be a positive definite integral operator defined by*

$$(T_k f)(\cdot) \int_{\mathbb{X}} k(\cdot, x)f(x)d\mu(x)$$

*Let $\psi_i \in L_2(\mathbb{X}, \mu)$ be normalize eigenfunctions of $T_k$ associated with the eigenvalues $\lambda_i > 0$. Then:*

1. *$\{\lambda_i\}_{i=1}^{\infty}$ are absolute summable*
2. $$k(x, x') = \sum_{i=1}^{\infty} \lambda_i \psi_i(x)\psi_i(x')$$

   *holds almost $\mu^2$ everywhere, where the series converges absolute and uniformly $\mu^2$ almost everywhere*

([Rasmussen Williams, 2006] pp. 96, [Bartlett, 2008] pp. 3 and [Hennig, 2020]). Since the eigenfunctions are orthogonal, they are also linearly independent, so $\psi(x) = [\psi_1(x), \psi_2(x), \dots]$ can be viewed as a transformation into a infinite space of basis functions. Indeed, we may choose to use the transformation $\phi$ to be $\psi$. Furthermore, if we take $\Sigma$ to be a infinite diagonal matrix with diagonal elements $\Sigma_{ii} = \lambda_i$, we see that $k(x, x') = \sum_{i=1}^{\infty} \lambda_i \psi_i(x)\psi_i(x') = \psi(x)^T \Sigma \psi(x')$. So any positive definite kernel is a inner product on a infinite dimensional space.

## 6.2 Reproducing Kernel Hilbert Space

The inner product corresponding to a Mercer kernel has deep ties to a type of Hilbert space, $\mathcal{H}$, called a Reproducing Kernel Hilbert Space (RKHS). This Hilbert space is endowed with the inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ of functions on an index set $\mathbb{X}$, where ([Felzenszwalb, 2017] pp. 4)

$$k(x, y) = \phi(x)^T \Sigma \phi(x') = \langle \phi(x), \phi(x') \rangle_{\mathcal{H}}.$$

$\langle \cdot, \cdot \rangle_{\mathcal{H}}$ has the property that there exists a unique kernel so that for every $f \in \mathcal{H}$ we have

$$\langle f(\cdot), k(\cdot, x) \rangle_{\mathcal{H}} = f(x). \tag{6.2}$$

This is called the reproducing property of $k$. Also, for every $x$, $k(x, x')$ as a function of $x'$ belongs to $\mathcal{H}$. Because of this, $k$ has the reproducing property with itself [Rasmussen Williams, 2006]

$$\langle k(x, \cdot), k(x', \cdot) \rangle_{\mathcal{H}} = k(x, x'). \tag{6.3}$$

Moore-Aronszajn theorem states that for every kernel there exists a uniquely determined RKHS and vice versa ([Rasmussen Williams, 2006] p. 130). The main purpose of introducing KRHS is to use the norm $\|f\|_{\mathcal{H}}^2 = \langle f, f \rangle_{\mathcal{H}}$ as a generalization to the quadratic form $\mathbf{f}^T K^{-1} \mathbf{f}$

[Rasmussen Williams, 2006], so that we can use $\xi(f) = \|f\|_{\mathcal{H}}$ as the penalty term in regularization. This is a norm in feature space, instead of input space, like with $\mathbf{y}^T(\mathrm{K} + \sigma_n I)\mathbf{y}$.

We may define a RKHS with respect to a kernel $k$ that has a finite eigenfunction expansion, with eigenfunctions $\{\psi_1, \ldots, \psi_N\}$ and corisponding eignevalues $\{\lambda_1, \ldots, \lambda_N\}$. From Mercer's theorem we have that $k(x, x') = \sum_{i=1}^{N} \lambda_i \psi_i(x)\psi_i(x')$ and that the eigenfunctions are orthonormal. In this space, each function $f \in \mathcal{H}$ is a linear combination of the eigenfunctions of $k$. That is

$$f(x) = \sum_{i=1}^{N} c_i \psi_i(x) \tag{6.4}$$

with

$$\sum_{i=1}^{N} \frac{c_i^2}{\lambda_i} < \infty,$$

where $\{c_i\}_{i=1}^{N}$ is a sequence of coefficients. Let $g$ be another such function with coefficients $\{c_i'\}_{i=1}^{N}$. Then we may define the inner product on $\mathcal{H}$ as

$$\langle f, g \rangle_{\mathcal{H}} = \sum_{i=1}^{N} \frac{c_i c_i'}{\lambda_i}.$$

By taking each coefficient to be $\lambda_i \psi_i(x)$, we see that the function $k(x, \cdot)$ is a member of this space. It is easy to see that this space is RKHS with respect to $k$ since

$$\langle f(\cdot), k(x, \cdot) \rangle_{\mathcal{H}} = \sum_{i=1}^{N} \frac{c_i \lambda_i \psi(x)}{\lambda_i} = f(x)$$

and

$$\langle k(x, \cdot), k(x', \cdot) \rangle_{\mathcal{H}} = \sum_{i=1}^{N} \frac{\lambda_i^2 \psi_i(x)\psi_i(x')}{\lambda_i} = k(x, x').$$

So the inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ has the reproducing property that we require of a RKHS. Finally we can formulate what we introduced this space for, namely to see that the norm $\|f\|_{\mathcal{H}}^2$ takes the nice form of

$$\|f\|_{\mathcal{H}}^2 = \langle f, f \rangle_{\mathcal{H}} = \sum_{i=1}^{N} \frac{c_i^2}{\lambda_i}.$$

Since we often don't know the eigenfunctions of a kernel, it is often easier to work with the reproducing kernel map representation of functions in a RKHS [Rasmussen Williams, 2006]. That is, instead of the functions being of the form (6.4), we would rather have them to be on the form

$$f(x) = \sum_{i=1}^{N} \alpha_i k(x, x_i). \tag{6.5}$$

Since we want these functions to be in a RKHS, we need the kernel to have to reproducing property with itself, like in (6.3). Because of this, we are forced to define the inner product by

$$\begin{aligned}\langle f, g\rangle_{\mathcal{H}} &= \langle \sum_i \alpha_i k(\cdot, x_i), \sum_j \alpha'_j k(\cdot, x'_j)\rangle \\ &= \sum_{i,j} \alpha_i \alpha'_j \langle k(x_i, \cdot), k(x'_j, \cdot)\rangle \\ &= \sum_{i,j} \alpha_i \alpha'_j k(x_i, x'_j).\end{aligned}$$

In particular, we have $\|f\|^2_{\mathcal{H}} = \langle f, f\rangle_{\mathcal{H}} = \boldsymbol{\alpha}^T \mathrm{K}\boldsymbol{\alpha}$. Using this inner product, we see that the kernel has the reproducing property

$$\langle f(\cdot), k(\cdot, x)\rangle_{\mathcal{H}} = \sum_{i=1}^{N} \alpha_i k(x, x_i) = f(x).$$

With this, we finally have the groundwork theory which we can use to show the differences and similarities between GP's and the machine learning techniques regularization and kernel regression.

## 6.3 Mathematical Relationship Between GP's and Regularization

We may now further generalize the work we did in 3.5 by using $\xi(f) = \|f\|^2_{\mathcal{H}}$ as our penalty function. Similar to the standard formulation of ridge regression, where we use $\xi(f) = \|w\|_2^2$, the norm $\|f\|^2_{\mathcal{H}}$ punishes large coefficients, $c_i$, of $\psi_i(x)$. However, this penalization is less significant if the eigenvalue $\lambda_i$ of $\psi_i(x)$ is large. Because of this, large weights on eigenfunctions which has small corresponding eigenvalues is punished more severely than those with larger corresponding eigenvalues ([Hasti Tibshirani, 2009] pp. 169). This effect is very similar to the shrinkage of eigenvector with small eigenvalues which was covered in Section 3.5.

Using $\|f\|^2_{\mathcal{H}}$, we may rewrite (3.5) to

$$J[f] = Q(f, y) + \lambda \|f\|^2_{\mathcal{H}}. \tag{6.6}$$

At first, $\lambda\|f\|^2_{\mathcal{H}}$ migth seem hard to compute, since it would require us to find a eigenfunction representation of $f$. Luckily, the *representation theorem* shows that each minimizer of (6.6) has the form of a linear combination of kernels, like in (6.5) ([Rasmussen Williams, 2006] p. 132). The representation theorem has an analog to the MAP of Gaussian processes. Remember that the predictive distribution of $f(x^*) = f_*$ at a test point $x^*$ is given by the posterior distribution $p(f_*|\mathbf{y}) = \int p(f_*|\mathbf{f})p(\mathbf{f}|\mathbf{y})d\mathbf{f}$. We have that ([Rasmussen Williams, 2006] pp. 44 and 133)

$$\begin{aligned}\mathbb{E}[f_*|\mathbf{y}] &= \int E[f_*|\mathbf{f}, \mathbf{y}]p(\mathbf{f}|\mathbf{y})d\mathbf{f} \\ &= \int \mathbf{k}(x_*)^T \mathrm{K}^{-1}\mathbf{f}\, p(\mathbf{f}|\mathbf{y})d\mathbf{f} \\ &= \mathbf{k}(x^*)^T \mathrm{K}^{-1} E[\mathbf{f}|\mathbf{y}].\end{aligned}$$

Since the MAP is the expected value of the posterior distribution, we have just showed that the solution of must be on the form (6.5), with $\alpha = K^{-1}E[\mathbf{f}|\mathbf{y}]$.

To see an even closer relationship between GP's and regularization, we look at the the special case of (6.6) when $Q(f,y) = \frac{1}{2\sigma_n^2}\sum_{i=1}^n (f(x_i)-y_i)^2$ and $\lambda = 1/2$. We get

$$J[f] = \frac{1}{2}\|f\|_{\mathcal{H}} + \frac{1}{2\sigma_n^2}\sum_{i=1}^n (f(x_i)-y_i)^2.$$

Using the representation theorem, we may write $f(x) = \sum_{i=1}^N \alpha_i k(x,x_i)$ and using the reproducing property of $k$ to find that $\|f\|_{\mathcal{H}} = \boldsymbol{\alpha}^T \mathrm{K}\boldsymbol{\alpha}$. Because each $k(x,x_i)$ is known relative to our data $\mathcal{D}$, we may write $J$ as a function of $\boldsymbol{\alpha}$

$$\begin{aligned} J[\boldsymbol{\alpha}] &= \frac{1}{2}\boldsymbol{\alpha}^T \mathrm{K}\boldsymbol{\alpha} + \frac{1}{2\sigma_n^2}(\mathbf{y}-\mathrm{K}\boldsymbol{\alpha})^T(\mathbf{y}-\mathrm{K}\boldsymbol{\alpha}) \\ &= \frac{1}{2}\boldsymbol{\alpha}^T(\mathrm{K}+\frac{1}{\sigma_n^2}\mathrm{K}^2)\boldsymbol{\alpha} - \frac{1}{\sigma_n^2}\mathbf{y}^T\mathrm{K}\boldsymbol{\alpha} + \frac{1}{2\sigma_n^2}\mathbf{y}^T\mathbf{y}. \end{aligned}$$

Minimizing this w.r.t $\alpha$, we find $\bar{\boldsymbol{\alpha}} = (K+\sigma_n^2 I)^{-1}\mathbf{y}$. So we have that

$$\bar{f}(x^*) = \sum_{i=1}^n \bar{\alpha}_i k(x^*, x_i).$$

Which is the exact same as the MAP of a Gaussian process we obtain in (4.8). This should come as no surprise. As stated in Section 4.1, the posterior distribution of a GP is closely related to regularization. Minimizing (3.5) is equivalent to maximizing

$$\exp(-J[f]) = \exp(-Q(f,y)) \times \exp(-\lambda\xi(f)). \tag{6.7}$$

Working under the assumption that $f$ is of the form $f(x) = \phi(x)^T\mathbf{w}$, if we use $Q(f,y) = -\frac{1}{2\sigma_n}(\mathbf{y}-\phi(X)^T\mathbf{w})^T(\mathbf{y}-\phi(X)^T\mathbf{w})$, $\xi(f) = \mathbf{w}^T K^{-1}\mathbf{w}$ and $\lambda = \frac{1}{2}$ we see that (6.7) takes the form of the posterior distribution of a GP, as in (4.6). Therefor, by finding the MAP, we minimize the regularization penalty.

Because K is positive definite we know that it has a eigendecomposition and have only positive eigenvalues. Therefor, we may take $\mathrm{K} = \sum_{i=1}^n \lambda_i \mathbf{u}_i\mathbf{u}_i^T$, so (4.8) becomes

$$\begin{aligned} \bar{\mathbf{f}} &= \mathrm{K}(\mathrm{K}+\sigma_n^2 I)^{-1}\mathbf{y} \\ &= \sum_i \mathbf{u}_i \frac{\lambda_i}{\lambda_i+\sigma_n^2}\mathbf{u}_i^T\mathbf{y}. \end{aligned}$$

$\bar{\mathbf{f}}$ is a vector of $\bar{f}$ evaluated at each training point. Given the similarity to (3.9), we may think of GP's as shrinking the direction of which there are less variance compared to those where there is more.

Since $\bar{f}(x^*) = \mathbf{k}(x^*)^T(K+\sigma_n^2 I)^{-1}\mathbf{y}$, we may define $s(x^*) = \mathbf{k}(x^*)^T(K+\sigma_n^2 I)^{-1}$ and see that $\bar{f}$ can be written on the form of a linear smoother $\bar{f}(x^*) = s(x^*)^T\mathbf{y}$, like in (3.1). Just like in local regression, $\bar{f}$ is a linear combination of weighs found with a normalized kernel. Because of this relationship, $s_i(x)$ is often referred to as the *equivalent kernel* ([Girosi, Jones, Poggio, 1995] pp. 5).

## 6.4 Main Differences Between Regularization and GP's

As we have seen, both regularization and GP's make assumptions about the function that produces the data points $\mathbf{y}$. The GP encodes it's assumption in the prior and builds a model

according to these assumptions. Elements of the model, like the regularization function based on the covariance matrix, comes naturally from making these assumptions. Regularization's accomplices the same goal by making assumptions in a somewhat reverse order. Instead of specifying the covariance, regularization can be thought of making smoothness assumptions by specifying the inverse covariance, $(\mathrm{K} + \sigma_n^2)^{-1}$, instead ([Rasmussen Williams, 2006] pp. 136). Because of this approach, regularization doesn't have access to the covariance matrix, which is useful both for calculating the marginal likelihood and the variance of $\hat{f}$. Without these tools, regularization is forced to estimate the variance, just like local regression.

Validation functions, such as marginal likelihood, is essential for hyper-parameter tuning and model comparison ([Rasmussen Williams, 2006] pp. 136). Without access to marginal likelihood, relaxation has to make due with cross validation. What difference dose this make? Interestingly, the difference might not be so significant as one would expect. In fact, it is possible to show that exhaustive leave p-out cross-validation averaged over all values of p is formally equivalent to marginal likelihood, if you use the log posterior predictive probability [Fong Holmes, 2020]. Exhausting leave p-out cross-validation can be viewed as the most complex form that cross-validation can possibly take, since it makes every possible training and test set division. Since marginal likelihood is formally equivalent, it can be view as a method that utilizes more information in the data set compared to other cross-validation techniques, such as k-fold cross validation. However, cross validation has the advantage of been faster to compute. This hold especially true for LOOCV when $\hat{f}$ is a linear combination, which takes linear time to compute. In comparison, marginal likelihood requires access to the inverse of a matrix, which is computationally taxing to compute.

Lastly, in the case where $Q(f, y)$ is a non-convex function, $J[f]$ has multiple local solutions. In this case, the argument for the regularization solution corresponding to the Gaussian MAP is rather weak. Furthermore, since regularization does not have access to the posterior distribution, it does not handle this multimodality well, suggesting that a Bayesian approach should be used instead ([Rasmussen Williams, 2006] pp. 136).

# 7 Discussion

The beauty of math is it's ability to show that concepts which at first might see different at first are instead closely interconnected. Although the initial approach of local regression, regularization and especially GP's to the task of finding a good approximation of a natural process differ, they share the same endpoint of expressing their approximation as a linear combination of kernel functions. If one where to further explore the topic of non-parametric regression one would find that other non-parametric approaches, such as smoothing splines and support vector machines, has similar ties to kernels. Remember that kernels can be interpreted as a function that quantifies proximity between points in space. Under this perspective, we may generalize much of non-parametric regression techniques as making predictions by combine observed responses that are weighted relative to their closeness to the point at which we want to make the prediction.

Despite the advantages of GP's discussed in Section 6.4, they are not widely used in applications. Which begs the question: why? It is believed that there are tree major reasons for this ([Rasmussen Williams, 2006] pp. 197). The first and most obvious reason is their need for inverting covariance matrices, which significantly increases their computation time. While this fact made GP's almost unusable some decades ago, with the rapid improvement of computers and the introduction of inverse matrix approximation this limitation is quickly becoming less of an issue. However, the fact remains that using GP's on a very big data sets might still be

computationally infeasible. This is problematic since the performance of any regression model is closely correlated with the amount of data fitted on. For this reason, GP's are limited smaller data sets and do not have the same potential performance as for example local regression if large amounts of data are available. Because of this, the need for research on efficient inverse matrix approximation becomes increasingly important ([Rasmussen Williams, 2006] pp. 171). If such techniques are effective, GP's might have a bright future in the age of big data.

The second reason is that most of the work on GP's has historically been done using fixed covariance matrices, which limits their usefulness when faced data that has a non-uniform variance, such as in Figure 1. It is not well know that one should be able to infer the structure of the data with the choice of covariance function. If the covariance function is chosen in a arbitrary fashion in most applications, this greatly limits the potential performance of GP's ([Rasmussen Williams, 2006] pp. 197). Nevertheless, even with the this knowledge, the implementer has a greater responsibility to make sound decisions about the structure of the data, which makes fitting GP's seem like a more daunting task. Algorithmic models has the advantages of yielding good results even in the hands of the more naive implementer.

Lastly, while the use of Bayesian methods such as GP's has become quite widespread in the Algorithmic community, it sees limited use in the statistical community. This fact might be especially problematic considering that the algorithmic community tend to view their methods as "black boxes" – only suited for prediction, not inference. This is problematic, not only because it limits the usage of GP's, but also because they are nicely suited for the statistical approach of model testing. The process of discovering a covariance matrix that allows the model to fit the data well can be viewed as opportunity to perform hypothesis testing about the structure of the data ([Rasmussen Williams, 2006] pp. 197). For this reason, probabilistic models are often more suited for inference than algorithmic models. Nevertheless, since the approximation of $f$ is computed as a linear combination of responses $y$, instead of a linear combination of the covariates $x$, we can't directly relate $\hat{f}$ to the covariates. Because of this, we may not infer the contribution of each covariate to approximated function, like we can with for example linear regression. So inference in non-parametric regression is limited to deducing the general structure of the data and may not find specific correlations.

Non-parametric regression contains a multitude of different approaches with a rich theoretical background. Although these models should not for every problem, their predictive power is not to be understated and it should be the goal of any data analyst to have some of these methods in their repertoire. In particular, it is useful to know both probabilistic and algorithmic models and when to harness the strengths of each approach for a particular problem. If the subjects covered in this report sparked your interest, one should look to the future with excitement, since new developments are sure to come.

# References


[James, Witten, Hasti, Tibshirani, 2021] G. Jame, D. Witten, T. Hasti, and R. Tibshirani. (2021). An Introduction to Statistical Learning. *Springer*

[Wasserman, 2006] Larry Wasserman. (2006). All of Nonparametric Statistics. *Springer*

[Rasmussen Williams, 2006] Carl E. Rasmussen Christopher K. I. Williams. (2006). Gaussian Processes for Machine Learning. *the MIT Press*

[Hasti Tibshirani, 2009] T. Hasti, R. Tibshirani, J. Friedman. (2009). The Elements of Statistical Learning. *Springer*

[Breiman, 2001] Leo Breiman. (2001) Statistical Modeling: The Two Cultures. *Statistical Science*

[Murphi, 2012] Kevin. P. Murphy. (2012). Machine Learning: A Probabilistic Perspective. *the MIT Press*

[Girosi, Jones, Poggio, 1995] Girosi, F., Jones, M., Poggio, T. (1995). Regularization Theory and Neural Networks Architectures. *Center for Biological and Computational Learning and Artificial Intelligence Laboratory Massachusetts Institute of Technology, Cambridge*

[Fong Holmes, 2020] E Fong C C Holmes. (2020). On the marginal likelihood and cross-validation *Biometrika, Volume 107, Issue 2, June 2020, Pages 489–496*, https://doi.org/10.1093/biomet/asz077

[Seth, 2021] Anil Seth. (2021). Being You: A New Science of Consciousness. *Penguin Random House LLC*

[Hennig, 2020] Phillip Hennig. (2020). Probabilistic ML - Lecture 10 - Understanding Kernels. https://www.youtube.com/watch?v=mezshT9r_80&t=1212s

[Hennig, 2020] Phillip Hennig. (2020). Probabilistic ML - Lecture 9 - Gaussian Processes. https://www.youtube.com/watch?v=s2_L86D4kUE&t=1537s

[Hennig, 2020] Phillip Hennig. (2020). Probabilistic ML - Lecture 8 - Learning Representations. https://youtube.com/watch?v=Zb0K_S5JJU4&t=1979s

[Felzenszwalb, 2017] Pedro Felzenszwalb. (2017). Machine Learning - Lecture 12. http://cs.brown.edu/people/pfelzens/engn2520/CS1420_Lecture_12.pdf

[Bartlett, 2008] Peter Bartlett. (2008). Reproducing Kernel Hilbert Spaces. https://people.eecs.berkeley.edu/~bartlett/courses/281b-sp08/7.pdf

[Patro] Rebecca Patro. (2021). Cross-Validation: K Fold vs Monte Carlo. *Towards Data Science*. https://towardsdatascience.com/cross-validation-k-fold-vs-monte-carlo-e54df2fc179b

[Smola Bartlett, 2000] Alex J. Smola Peter Barlett. (2000). Sparse Greedy Gaussian Process Regression. *Australian National University.*

**Abstract**

This report was submitted as a bachelor thesis for Bachelor in Mathematics at the Norwegian University of Science and Technology. The term non-parametric regression encompasses a great number of machine learning technique that predict the outcome of natural processes using functions that are more flexible than their parametric counterparts in order to they gain an edge in predictive capabilities. One may further subdivide the category into algorithmic and probabilistic models. The algorithmic approach focus on building models that has high predictive performance, while the probabilistic perspective builds models with a statistical approach that aims to mimic the process. For the purpose of comparing these two approaches, two algorithmic models (local regression and relaxation) and one probabilistic model (Gaussian processes) are to be thoroughly covered. In a practical comparison between Gaussian processes and local regression, Gaussian processes where found to have a predictive perform at a similar level to local regression, while yielding much better variance estimation, but had a significantly longer computation time. By exploring the rich theory behind kernel functions found in Mercers theorem and reproducing kernel Hiblert spaces it was shown that all tree techniques can be expressed as estimating a function by linearly combining observation weighed by a kernel function.

# 1 Introduction

The problem of predicting uncertain natural processes is both a important and a difficult task. The usefulness of answering questions such as: "Who is going to win the next election? What is going to be the price of bitcoin next month?", "How do your genes contribute to your chance of getting hearth disease?", is not to be understated. Still, anyone who works within the respective fields of political science, finance and medicine knows that answering such questions is not an easy task. Whenever there is a need to predict the outcome of random processes and assess the uncertainty of predictors, it is wise to consult the field of statistics. In particular, the field of statistical modeling and machine learning has made some great achievements in the last decades.

Inspired by the article *Statistical Modeling: The Two Cultures* by Leo Brierman, we may divided the field of statistical modeling into two cultures: the Data Modeling Culture and the Algorithmic Modeling Culture. Models from these cultures will be referred to as *probabilistic* and *algorithmic* models, respectively. Both cultures aim to predict the outcomes in nature. One can think of nature as a kind of black box. We may observe inputs and outputs into and out of the black box, but not the precise mechanisms of the process itself. The Data Modeling Culture takes a statistical approach. They build a statistical model which they assume to describe the natural process. Then they use the model to predict future outcomes, hopefully mirroring the mechanisms of the natural process. Predictions of the model can easily be questioned by criticizing the assumptions the model is taking. The Algorithmic approach ignores the natural process all together, instead focusing solely on making predictions. The methods used make no attempt to mirror the natural mechanisms. Therefore, few assumptions are needed. Instead, their ability to make accurate predictions is the only measure of their effectiveness [Breiman, 2001].

Section 2.1 consists of an introduction to central concepts of this text, such as Non-parametric regression and methods of measuring performance of models. The focus of Section 3 is on the algorithmic models of local regression and regularization, which can be generalized to a class of functions called linear smoothers. Section 4 introduces the essence of the data modeling culture in building probabilistic models. The focus of this section is on Bayesian inference, and more specifically Gaussian processes. Section 5 consists of a practical comparison of local regression and Gaussian processes by implementing the two methods and comparing the results when used on two different data sets. Lastly, Section 6 will take a deeper dive into the theory of kernels

with the goal of describing the mathematical similarities between Regularization and Gaussian Processes and discussing their differences.

# 2 Problem Specification

## 2.1 Non-parametric Regression

The problems the two cultures aims to answer is either of the from of a regression problem or a classification problem. Both of these concepts are closely related. They define a set of quantitative properties we observe as inputs to a natural process and aim to predict the outcome of the natural process given the properties. The difference lie in the type of output natural process produces. If the output is discreet, that is, the output belongs to a class, we describe it as a classification problem. A classical example is to predict the actual digit from the picture of a hand written one. In this case, the input is a set of pixels, those that make up the picture of the digit, while the output the digit the human wanted to write. One may assign a desecrate value, 0-9, that corresponds to witch class the digit belongs to. On the other hand, a regression problem has a continuous output. For example, predicting the amount of drag that acts upon a rocket relative to its speed. Sometimes, the relationships are simple, like the linear relationship between how many items are sold and how much money is made. But often, the interaction is complex. Acceleration of the head in a motorcycle accident is a good example of this. As we can see in Figure 1, the amount of g-force that acts on the head is a constant value of zero right after the accident, then suddenly spikes to above negative 100 g, before increasing to around 50 g and then reverting back to around zero.

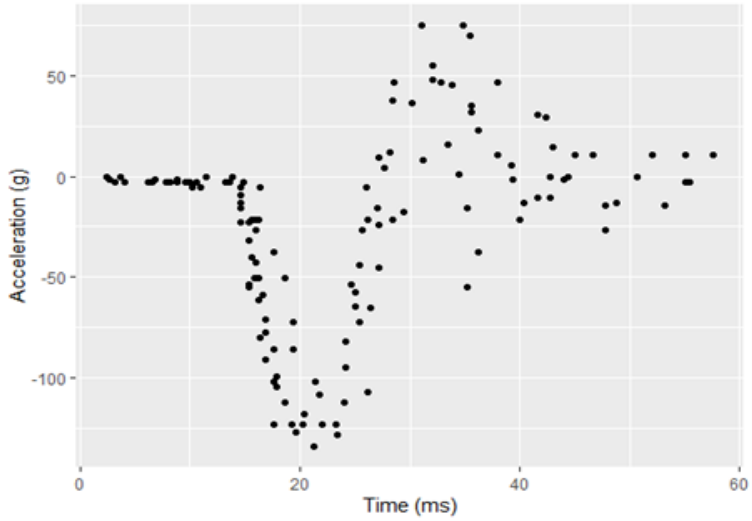



Figure 1: Acceleration of the head in a simulated motorcycle accident. Time measured in milliseconds after impact and acceleration measured in g.

This behavior is hard to simulate with parametric functions. For example, no polynomial starts of as a constant, before sharply decreasing. For this reason, it is useful to use non-parametric functions, which has unlimited complexity. This is because, in parametric regression we are limited to a fixed number of parameters, while in non-parametric regression, the number of pa-

rameters grows with the size of the training data [Murphi, 2012]. So, contrary to the name, a non-parametric model fits a function using a potentially infinite number of parameters.

This complexity comes at a cost however. Non-parametric regressions are viewed as black box methods in that their main focus is on *prediction* of future output of a process, rather than *inference*. The goal of inference is to quantify the relationship between the predictors and the response. It is often the case that non-parametric models give excellent prediction results while we have little insight into why the model works. Still, there are mayor differences within the field of statistical modeling. Probabilistic models, such as Gaussian processes, tend to be more aligned with the goal of performing inference than algorithmic models. Still, by the merit of being non-parametric, parametric models such as linear regression are significantly more easy to interpret.

A general formulation of any regression problem is that we have a set of data $\mathcal{D} = \{(\mathbf{x}_1, y_1), .., (\mathbf{x}_n, y_n)\}$, where $\mathbf{x}_i$ is a p-dimensional vector. In this text, I will focus on 1-d regression, where our goal is to map a one dimensional input, $x$, to a 1-d output $y$. The reasoning for this choice is that it is both easier to understand and visualize, while still sufficient to explain the core ideas of non-parametric regression. However, in some examples it is more suitable to use multidimensional input vector. The motivation for this is to demonstrate that it is often not difficult to convert a one dimensional problem to a multidimensional one. Whenever we have a multidimensional input it will be denoted as a vector, $\mathbf{x}$, instead of $x$.

We want to use future $x$ values to predict $y$. To do this, we need to quantify a relationship between $x$ and $y$. We assume

$$y = f(x) + \epsilon, \tag{2.1}$$

where $f : \mathbb{R} \to \mathbb{R}$. $y$ is our observations, like the points in Figure 1. These observations are the sum of two components: the true value of the process, like the actual acceleration of the head, and some noise that makes our observations diverge from the true value. This noise might be due to random differences in the process each time we repeat it or errors in the measurement of it. We may assume that these errors are evenly spread around the true value of the process and are more likely to be closer to the true value rather than further away. Mathematically, we express these assumptions by assuming that the error, $\epsilon$, are normally distributed, that is $\epsilon \sim \mathcal{N}(0, \sigma_n^2)$. To predict the process we need to find an accurate representation of $f$. Form the data modeling perspective, this process consists of making assumptions about the nature of $f$ and building a model from these assumptions. From the Algorithmic perspective, we simply want to find a good approximation of $f$. In both cases, we want to find a function $\hat{f}$ that estimates $f$. To determine the performance of $f$, or to test if our assumptions are justified, we need a measure of the accuracy and precision of $\hat{f}$ relative to $f$.

## 2.2 Measuring Predictive Performance

### 2.2.1 Loss Function

We would like to measure to what degree the estimated function, $\hat{f}$ differ from the true function, $f$. We may call a function that quantifies this general difference an *error function*. In practice, since we don't have access to the true value of $f(x)$, we use $y$ instead. In general we use a *loss function*, $L$, to quantify the difference between $\hat{f}$ and $y$ at a specific point. For example, in a classification problem with two classes a natural choice is to use the indicator function $I(\hat{f} \neq y)$, which is zero when $\hat{f}$ is the same class as $f$ and one otherwise. In the continuous case of a regression problem, a more natural choice is for $L$ to be a distance measure. The most common choice is the square error $L(\hat{f}, y) = (\hat{f}(x) - y)^2$ ([James, Witten, Hasti, Tibshirani, 2021] pp.

29). One way of measure the total error is to quantify the degree to which we may expect $\hat{f}$ to differ from $y$. Mathematically, this can be formulated as the expected value of the loss function, $\mathrm{E}[L(\hat{f}, f)]$, where $\hat{f}(x)$ is a random variable, since it implicitly depends on the observed data ([Wasserman, 2006] pp. 51). Additionally $\epsilon$ in $y = f(x) + \epsilon$ is by assumption a random variables, while $f(x)$ is not. We can estimate the expected value with the mean. Therefor, we may take the total error of our estimated function, given our data set $\mathcal{D}$, to be the mean square error (MSE) ([James, Witten, Hasti, Tibshirani, 2021] pp. 29)

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{f}(x_i))^2. \tag{2.2}$$

It is important to know that square error is not always the right choice. The main contender is the absolute error $L(\hat{f}, f) = |\hat{f}(x) - f(x)|$. Square error is preferred because, unlike absolute error, it is a continuous differentiable function. Since it is also convex, it is ideal for optimization, the importance of which is going to become obvious later. The weakness of MSE is that it punishes large outliers heavily, making the regression less *robust*. Robustness is a measure of the sensitive a prediction has to the introduction of new data points. A non-robust function changes significantly with the introduction of one single outlier into to data set. With absolute error, distance grows linearly instead of quadratically, so the effect of outliers is less severe, making it the preferred choice when outliers are known to be frequent. In this report, it is assumed that the effect of outliers is negligible, making MSE i viable choice.

The common way of building an algorithmic model is by specifying an error function, such as MSE, and finding the optimal solution to the problem by minimizing it. Probabilistic models differs in this regard. They build there models by specify beliefs about the behavior of the true function. Instead of being the starting point, the validation function is a product of the model building process. This can be viewed as the one of the main differences between algorithmic and probabilistic models.

When building a non-parametric algorithmic model, the MSE is unsuitable as a error function of two main reasons. Firstly, there exist a infinite number of functions that mininizes (2.2) – any function that interpolates every data point is a valid choice ([Hasti Tibshirani, 2009] pp. 30). Each of the algorithmic models covered in this text uses a modified version of the MSE which has a unique solution. Local regression uses a weighted sum, while regularization adds a penalty term. Both these approaches also solves the second problem of MSE; accounting for the bias-variance trade-off.

### 2.2.2 Bias-Variance trade-off

When trying measure MSE on a actual data set, we quickly encounter a problem. To reduce the MSE on the current data set, we could simply use a function that tries to interpolate the data as closely as possible, like in Figure 2

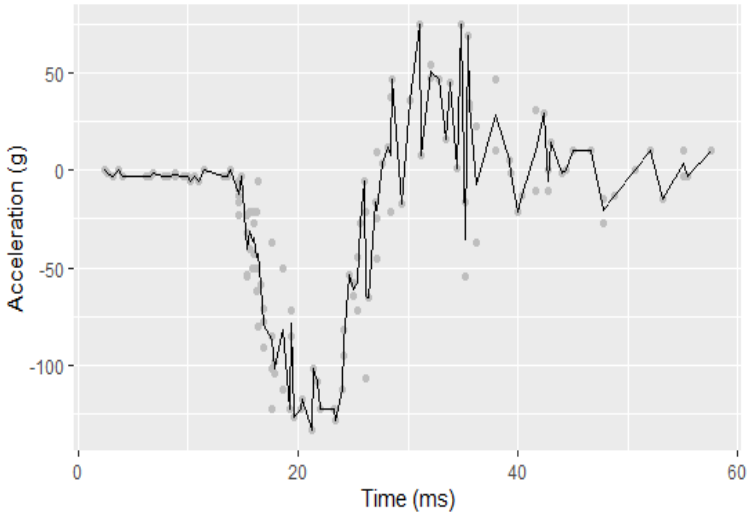



Figure 2: An example of a function that overfits our data.

The problem with this is that we are very likely to fit random noise, what we call *overfiting*. Because of this, our estimated function is unlikely to be a good approximation of the true function, $f$, and therefor not a good predictor of future data. Intuitively, we need that our function is close the data points, so that it is likely to approximate $f$, but not to close, so that we end up overfiting. We may express this duality mathematically. Since we assume $y$ to be on the form (2.1) can we decompose (2.2) into tree components

$$E[L(y, \hat{f}(x))] = E[(y - \hat{f}(x))^2] = E[\hat{f}(x)^2] - 2E[\hat{f}(x)y] + E[y^2]$$

$$= E[\hat{f}(x)^2] - 2(E[f\hat{(x)}]f(x) + E[\epsilon]E[\hat{f}(x)]) + f(x)^2 + 2E[\epsilon]f(x) + E[\epsilon^2].$$

Since we assume $\epsilon \sim \mathcal{N}(0, \sigma^2)$ we have

$$E[\epsilon] = 0 \text{ and } Var[\epsilon] = E[\epsilon^2] - E[\epsilon]^2 = E[\epsilon^2].$$

Using this and by adding and subtracting $E[\hat{f}(x)]^2$ we get

$$= (E[\hat{f}(x)]^2 - 2E[f\hat{(x)}]f(x) + f(x)^2) + (E[\hat{f}(x)^2] - E[\hat{f}(x)]^2) + Var[\epsilon]$$

$$= (E[\hat{f}(x)] - f(x))^2 + Var[\hat{f}(x)] + \sigma^2 = \text{BIAS}^2 + \text{VAR} + \sigma^2. \tag{2.3}$$

As denoted in the above equation, the square bias is defined as the square error between the expected value of $\hat{f}$ and the true function $f$. The square bias decreases when the complexity of $\hat{f}$ increases. The more complex $\hat{f}$ is, the closer it may approximate each data point $y_i$. However, the reduction of bias comes at a cost. To approximate every data point, $\hat{f}$ must fluctuate to a high degree and have low smoothness. Such a function has high variance, and therefor increases the second term in (2.3), and the total error with it. Because of this, variance increases with function complexity, and so, there is a trade-off between variance and bias. If we take the function to be to simple, we *underfit* the data. However, if we make it to complex we overfit instead. The sweet spot lies somewhere in between. One of the main challenges in any regression is to fine tune the model so that we may obtain this minimum. Even if we find it, $\hat{f}$ may never perfectly

predict new data points. This is because of the last term in (2.3). The error due to the variance of $\epsilon$, $\sigma_n^2$, is irreducible. Because of this, it is a lower bound of any error estimation, which can be seen in Figure 3 ([James, Witten, Hasti, Tibshirani, 2021] pp. 33-36).

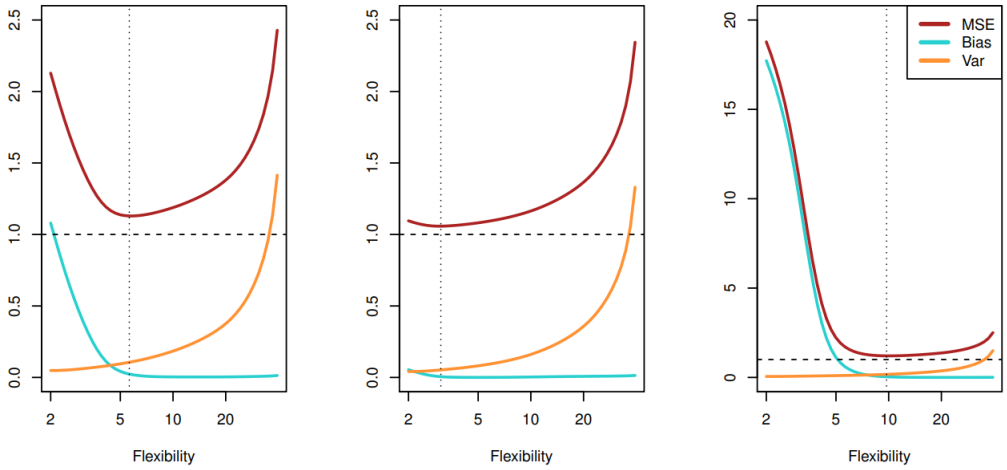



Figure 3: The Bias-Variance Trade-off for tree different data sets. The square bias (blue curve), variance (orange curve), MSE (red curve) and Var($\epsilon$) (dashed horizontal line). The vertical doted line denotes the minimum of MSE value. Source: [James, Witten, Hasti, Tibshirani, 2021]

### 2.2.3 Cross-Validation

Instead of using an error function, which only attempts to reduce bias, we must use a *validation function* which incorporates the bias-variance trade-off in it's error estimate. In the algorithmic culture, two techniques in particular are popular for accomplishing this task: *cross-validation* and *bootstrapping*. Both techniques attempt to simulate the process of observing new data and measuring the performance of our regression in predicting that data, but they accomplish this in very different ways. Bootstrapping uses our current data to estimate the distribution that our data is produced from. Then it uses the estimated distribution to produce new data points which we measure our regressions performance on. The detail of how this is done will be omitted. Instead, the focus is directed on cross-validation for the remainder of the text.

Cross-validation is similar to a more naive technique in which one sets aside a part of the data set that is used for validation, then fit the regression on the remaining data – the training set. The validation set is independent of the training set, so our error estimate is punished for fitting noise. The problem with this technique is that we are unable to use a significant portion of our data for training our regression function; worsening it's performance. Cross-validation has a clever solution to this problem. In p-fold cross validation, the data set is divided into p number of equally sized disjoint sets. Like in the naive method, we use on of the p sets for validation, the rest for training. But unlike the naive method, we repeat this process p times, once for each unique validation set, and then use the average performance ([James, Witten, Hasti, Tibshirani, 2021] pp. 203-204). For example, if we have 100 data points and perform 5 fold cross validation, we would divide the data into 5 sets of 20 data points and repeat the training validation process 5 times. Figure 4 shows an illustration of 5 fold cross validation.


Figure 4: Illustration of 5 fold cross validation. (Source: [Patro]

An alternative to dividing the data into p distinct data sets is to use leave p out cross validation. In this version, we leave p data points, train on the remaining, and use the p left out data points to for testing. At each test, we take the average of the p errors. We repeat this $\binom{n}{p}$ times, once for each possible combination of p, then taking the average. The most common choice of p is 1, and is called leave one out cross validation (LOOCV). LOOCV is computationally efficient when the estimated function is a linear combination of responses $y$ ([Hasti Tibshirani, 2009] p. 161). It turn out that many non-parametric functions is of this form. They can be generalized to a class of functions called linear smoothers.

# 3 Linear Smoothers

In this section we are going to take a close look at two algorithmic models, local regression and relaxation. Both these methods has a direct approach to estimating $f$. In local regression we stitch together a function from a series of points that each are a local weighted average. In relaxation we instead need to make some assumptions about the form of $f$. In both approaches, we make assumptions about the smoothness, or the variance, of $f$. In local regression, these assumptions is specified by the weights, while in relaxation we use a penalty function that punishes non-smooth functions. In both cases, these assumptions are encoded with a bandwidth or variance variable. They also share the general form of a linear smoother function.

## 3.1 Linear Regression as a Linear Smoother

A linear smoother is any function that is a linear combination of the response variables $\mathbf{y} = [y_1, \ldots, y_n]$. This is a concept that should be familiar to everyone with a basic knowledge of regression. Indeed, the most common regression type, linear regression, is of this form. In multiple linear regression, we expect $f$ to be a liner function, so $f(\mathbf{x}) = \mathbf{x}^T\mathbf{w} = \sum_{i=1}^{p} w_i x_i$, where $\mathbf{w}$ is a vector of weights and $\mathbf{x}$ a p-dimensional input. We find the best fit by minimizing the residual sum-of-squares (RSS), where $RSS = n * MSE$. Because we limit the freedom of $f$

by assuming it is a linear function there exist a unique minimizer of RSS. Taking $\mathrm{X} = [\mathbf{x}_1|\dots|\mathbf{x}_n]$ to be a matrix of data points, we may write RSS as

$$\mathrm{RSS}(\mathbf{w}) = (\mathbf{y} - \mathrm{X}\mathbf{w})^T(\mathbf{y} - \mathrm{X}\mathbf{w}).$$

The function is convex with respect to $\mathbf{w}$, so it is minimal when it's derivative is equal to zero. This is when $\hat{\mathbf{w}} = (\mathrm{X}^T\mathrm{X})^{-1}\mathrm{X}^T\mathbf{y}$. We get the estimate $\hat{\mathbf{y}} = \mathrm{X}\hat{\mathbf{w}}$ ([Hasti Tibshirani, 2009] pp. 64). The following definition is a generalization of this

**Definition 3.1.** *Let $\hat{f}$ be a linear smoother of $f : \mathbb{R}^p \to \mathbb{R}$ if there exist a vector $s(\mathbf{x}) = [s_1(\mathbf{x}), ..., s_n(\mathbf{x})]$, with $s_i : \mathbb{R}^p \to \mathbb{R}$ for each $\mathbf{x}$ such that*

$$\hat{f}(\mathbf{x}) = s(\mathbf{x})^T\mathbf{y} = \sum_{i=1}^{n} s_i(\mathbf{x})y_i \tag{3.1}$$

([Wasserman, 2006] p. 71, def 5.39). Using the smoothing function $s(\mathrm{x}) = \mathbf{x}(\mathrm{X}^T\mathrm{X})^{-1}X^T$, we see that linear regression can indeed be written on this form ([Wasserman, 2006] pp. 64).

The procedure of finding an estimate of $\hat{f}$ on the form of a linear smoother by defining a form of $f$ and optimizing an error function is going to be common case throughout this text. The purpose of introducing liner regression is to show how non-parametric regression techniques is far from alien, but is instead closely related to well know regression methods. One such non-parametric regression technique is local regression.

## 3.2 Local Regression

We continue to assume that $y$ to be on the form of (2.1), but we limit our self to the case where $x$ is one dimensional. The essence of local regression is build on the assumption that at each point $x$, a good estimate of $f$ is a weighed average of local data points. The flexibility in local regression arises from our freedom in choosing what we define local to be. For example, one simple approach is to use the K-nearest neighbours of $x$. Assume $\mathcal{B}_x$ represents the K-nearest points to $x$, then we may estimate $\hat{f}(x)$ by ([James, Witten, Hasti, Tibshirani, 2021] p. 105)

$$\hat{f}(x) = \frac{1}{K}\sum_{x_i \in \mathcal{B}_x} y_i.$$

Observe that if we define $s_i$ to be

$$s_i(x) = \begin{cases} \frac{1}{K}, & \text{if } x_i \in \mathcal{B}_x \\ 0, & \text{if } x_i \notin \mathcal{B}_x \end{cases}$$

then K-nearest regression is on the form a linear smoother.

There is a few problems with K-nearest regression. Firstly, we can't guarantee that all data points are uniformly distributed. Some regions might be sparse in its data intensity, while others much denser. Therefor, the general rule of always using the K nearest point might vary in accuracy depending on the sparsity. Secondly, we may assume that points closest to x are also closer to the true value of $f(x)$ than those further away. One solution to both these problems would be to assign weights to each $y_i$ depending on the distance from their corresponding x-coordinate, $x_i$, to $x$. At each point $x$ we would like to find a smoothing vector $s(x)$ that minimizes the distance to each $y_i$ and use a distance measure to favor those $y_i$ who's corresponding $x_i$

coordinate is the closest to $x$. Mathematical, we would like to minimize the weighted sum of squares (ref. [Wasserman, 2006] pp. 75)

$$\sum_{i=1}^{n} w_i(x)(y_i - \hat{f}(x))^2 \tag{3.2}$$

where $w(x) = [w_1(x), ..., w_n(x)]$ is a weight function. Minimizing with respect to $\hat{f}(x)$ we find

$$\hat{f}(x) = \frac{\sum_{i=1}^{n} w_i(x) y_i}{\sum_{i=1}^{n} w_i(x)} \tag{3.3}$$

since (3.2) is a convex function, (3.3) must be a global minimizer. A common choice of weigh functions is called kernel functions, so we take $w_i(x) = k(x, x_i)$ [Wasserman, 2006]. Using kernels in (3.3), we get what is referred to as the Nadaraya-Watson kernel estimator for the smoothing vector $s$ ([Wasserman, 2006] pp. 71 def. 5.39).

$$s_i(x) = \frac{k(x, x_i)}{\sum_{i=1}^{n} k(x, x_i)}. \tag{3.4}$$

This expression can be viewed as a normalized kernel, since $\sum_{i=1}^{n} s_i(x) = 1$. Using (3.4), we may write (3.3) on the general form of a smoothing function (3.1). From this, the non-parametric nature of local regression becomes obvious. Each new data point contributes to estimation of $s$. So when the size of the data grows, we may utilize more information in our estimation of the true function.

It is important to note that N-W regression is not the only form of local regression. Indeed, it is merely a special case of local polynomial regression. From Taylor's theorem we know that the polynomial

$$f(x) + f'(x)(u - x) + f''(x)\frac{(u - x)^2}{2!} + ... + f^{(p)}\frac{(u - x)^p}{p!}$$

is a good approximation of $f$ for all $u$ in the neighborhood of $x$. Motivated by this, we estimate a vector $a(x) = [a_0(x), ..., a_p(x)]$ such that

$$P_x(u; a) = a_0(x) + a_1(x)(u - x) + a_2(x)\frac{(u - x)^2}{2!} + ... + a_p(x)\frac{(u - x)^p}{p!}.$$

Similar to N-W regression, we want the vector $a(x)$ to minimize ([Wasserman, 2006] pp. 75)

$$\sum_{i=1}^{n} w_i(x)(y_i - P_x(x_i; a))^2,$$

the solution of which is omitted in this text. Notice that if we take the degree of the polynomial to be $p = 0$ – the case where we estimate a constant – we have regular kernel regression, which has the solution of the N-W kernel estimator. In Figure 5 we find an illustration of the Nadaraya-Watson (N-W) kernel estimator using the Epanechnikov kernel as weight function.

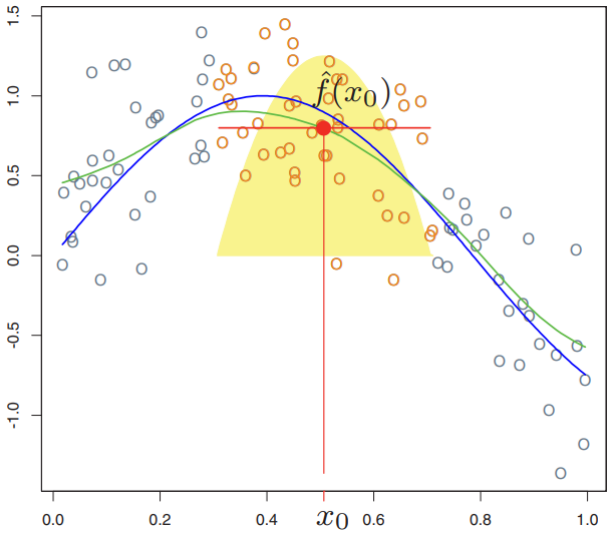



Figure 5: Nadaraya-Watson kernel estimator using the Epanechnikov kernel as weight function (green) and the true function from which the data points has been generated from (blue). The point $\hat{f}(x_0)$ is a weighed average of each of the red points in the neighborhood, where their weight corresponds to the height of Epanechnikov curve, shown in yellow. Source: [Hasti Tibshirani, 2009]

For a comparison, Figure 6 shows the result if we where to use either linear or quadratic regression.

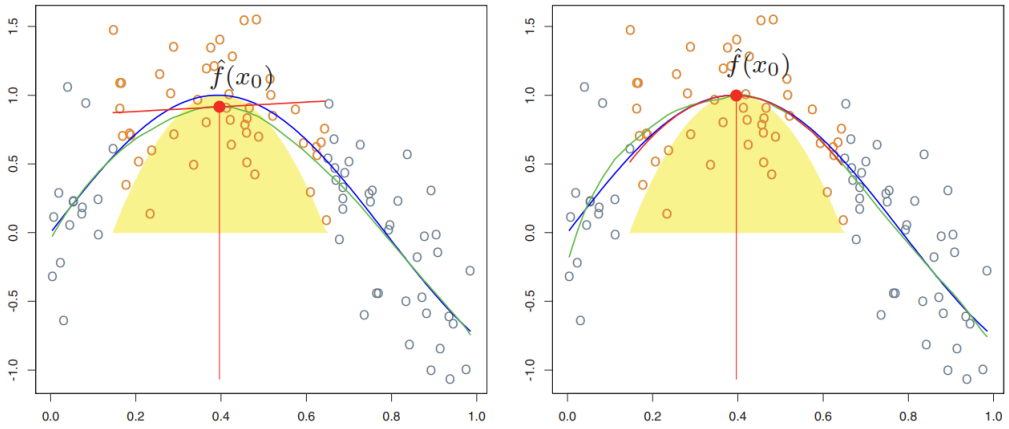



Figure 6: Linear (left) and quadratic (right) local regression. Source: [Hasti Tibshirani, 2009]

To better understand these figures, we need a basic understanding of kernel functions.

### 3.3 Kernels in Local Regression

For the purpose of this section, we may think of kernels as inverse distance measure. Kernel functions are both symmetric, $k(x, x') = k(x', x)$, positive, $k(x, x') \geq 0$, and attains it maximum

when $x = x'$ ([Wasserman, 2006] pp. 55). Since $k(x, x_i)$ is large when $x_i$ is close to $x$, the kernel grants us the property that desired for our weights. There exists a multitude of choices for kernel functions $k$. Using $k(x, x') = k(x - x')$, tree such choices are the Epanechnikov kernel defined by

$$k_e(x) = \frac{3}{4}(1 - x^2)I(x)$$

the boxcar kernel defined by

$$k_b(x) = \frac{1}{2}I(x)$$

and the Gaussian kernel

$$k_g(x) = \exp\left(-\frac{x^2}{2}\right)$$

Where

$$I(x) = \begin{cases} 1 & \text{if } |x| \leq 1 \\ 0 & \text{if } |x| > 1 \end{cases}$$

In figure 7 we see a plot of each of these functions.

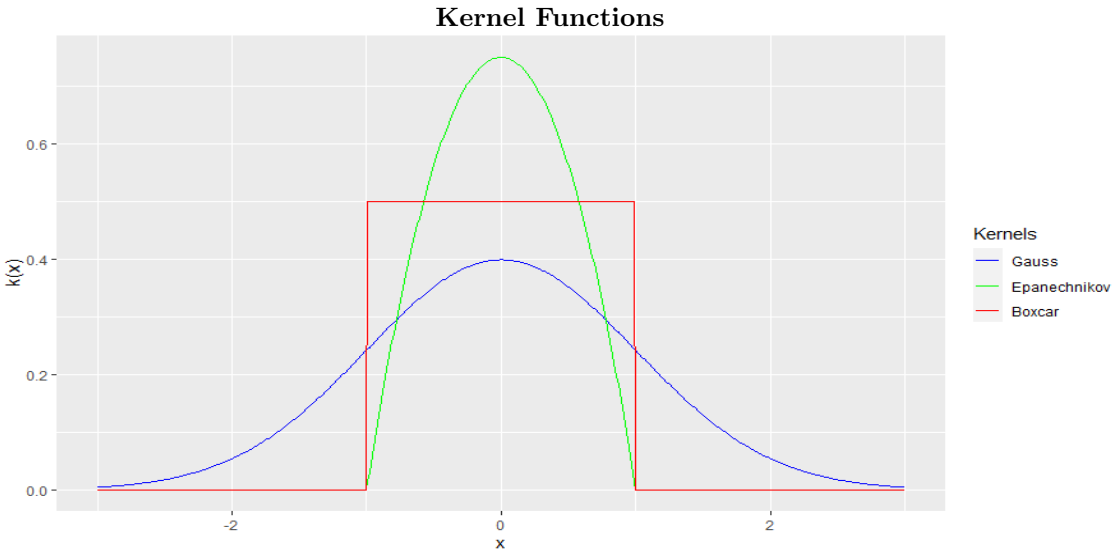



Figure 7: Value of each of the kernel functions around zero for a one dimensional input.

The choice of kernels encodes assumptions about how we think different points relate to each other. The boxcar kernel can be thought of ball $\mathcal{B}_x$ drawn around $x$ where every other point $x'$ is assigned a constant weight if they are inside the ball, or a weight of zero if they are outside. In our case, where $x$ is one dimensional, the ball is simply a interval centered around $x$. The Epanechnikov kernel differs from the boxcar kernel by favoring points closer to the center than further away, instead of assigning a constant weight. We are stating implicitly that the importance of each point $x'$ reduces gradually as they move further away from $x$ by choosing the Epanechnikov kernel instead of the Boxcar kernel.

The choice of kernel differentiate between relative distances, but say nothing about how

absolute distances should be weighted. If every data point is more than 1 unit of distance apart from each other, then the Epanechnikov and Boxcar kernel would assign every data pair $x$ and $x'$ (where $x \neq x'$) a weight of 0, which is less than desirable. To adjust absolute distance, we may introduce a constant, $h > 0$, called the *bandwidth* and define ([Hasti Tibshirani, 2009] pp. 35)

$$k_h(x) = k(x/h).$$

For both the Epanechnikov and Boxcar kernel, we see that $k(x, x')$ is non-zero when $|x - x'| \leq h$. $h$ can be thought of as the radius of $\mathcal{B}_x$. $h$ is important when considering the bias-variance trade off. If we take $h$ to be large, it will fit mostly the closes data points at each prediction point $x'$. This reduces the bias. Since fewer data points are used, the fit is more sensitive to change, so the predicted function has higher variance. When fitting a function, we have to tune $h$ to the value in which error is on the sweet spot between high bias and high variance.

Kernels is an important topic throughout this text, especially in later parts, such as Section 6. Even in methods such as regularization, which may at first seem entirely unrelated to kernels, we will eventually see that kernels hides just below the surface of the theory.

### 3.4 Regularization

Just like in local regression, we assume the true function to be close to the current data. To impose this assumption, we would like find a function that minimizes a means sum of squares. Unlike in local regression, we don't take a weighted sum of squares. Unfortunately, this makes the problem ill-posed – there exist an infinite number of solution function; any function that interpolates the data would do. In regularization, this challenges is solved by imposing smoothness conditions on the estimated function. This solves two problems: we force an unique solution and require the function to have minimal variance [Girosi, Jones, Poggio, 1995]. Mathematically, we want to minimize ([Rasmussen Williams, 2006] pp. 132)

$$J[f] = Q(f, y) + h\xi(f) \tag{3.5}$$

where $\xi$ is a penalty function that yields a low value when $f$ is smooth ([Hasti Tibshirani, 2009] pp. 165). Q is some data fit function that is small when $f$ is close to the data $y$. In Ridge regression, which is going the be the regularization technique of focus, we take $Q(f, y)$ to be the RSS ([James, Witten, Hasti, Tibshirani, 2021] pp. 237). $h$ is called a regularization parameter [Girosi, Jones, Poggio, 1995]. However, we may recognize $h$ as the bandwidth parameter that we have already encountered. When $h$ is large, we force larger smoothness requirement upon $f$, so that the variance of $f$ decreases, but the bias increases.
There are many choices for $\xi$. Remember that a linear function has zero second derivatives, while a function with large fluctuations has large second derivatives. So if $f \in C_2(\mathbb{R})$ is twice continuous differentiable, one intuitive way is to punish functions with large second derivatives. We may do this by taking the MSE of the second derivative. Since f is continuous, we use integration instead of MSE

$$\xi(f) = \int f''(t)^2 dt.$$

This is a common choice when regularizing splines and is called smoothing splines ([James, Witten, Hasti, Tibshirani, 2021], p. 301). In this text however, the focus will be on the case where $f$ is a linear combination of a set of linearly independent basis function. This can be accomplished by utilize a transformation, $\phi$, of $x$. For example, we may transform a scalar into a polynomial of degree p by using $\phi(x) = [1, x, x^2, ..., x^p]$. Then any polynomial of degree p

may be represented as

$$f(x) = \phi(x)^T \mathbf{w} \tag{3.6}$$

where $\mathbf{w}$ is vector of weights. Nothing in this formula limits us to the use of a polynomial transformation. Instead, this form is a generalization of any linear combination of basis functions $\phi(x) = [\phi_1(x), \ldots, \phi_p(x)]$. After deciding on a set of basis functions – what is often called *features* – we may adjust the behavior of $f$ by modifying the weights. However, not all functions on the form (3.6) is twice continuously differentiable. An example of this is the basis consisting of step functions, $\phi(x) = [I(x < c_0), I(c_0 x < c_1, \ldots, I(x < c_p)]$, where $I$ is the indicator function introduced in 3.3 and $c_0 < c_1 < \cdots < c_p$ a set of constants spread over a interval. Luckily, it turns out that a simple but efficient way of punishing non-smoothness of a function of the form (3.6) is simply taking the $\ell_2$-norm of the weight vector, what is called a *Ridge penalty* ([Hasti Tibshirani, 2009] pp. 68)

$$\xi(f) = \|w\|_2^2.$$

What is the intuition behind this? If the magnitude of each weight is unrestricted, then $f$ a greater variety of shapes, many of which has high variance. When the size of $\|w\|_2^2$ is restricted, the space of feasible forms of $f$ shrinks. In particular, if any feature $\phi_i(x)$ is of little importance in prediction the behavior of $f$, then the penalty is motivated to shrink it's corresponding weight, $w_i$, close to zero, making it's contribution insignificant. This effectively reduces the number of features, reducing the complexity and therefor the variance of $f$. This effect is especially prominent when using a *Lasso penalty* instead of a Ridge penalty – which uses a $\ell_1$-norm instead of a $\ell_2$-norm – since the norm shrinks many weights to exactly zero, while ridge only shrinks weights close to zero ([James, Witten, Hasti, Tibshirani, 2021] pp. 243-245).

If we take $\Phi$ to be a matrix such that $\Phi = [\phi(x_1)|\phi(x_2)|\ldots|\phi(x_n)] = \phi(\mathbf{x})$, we may rewrite (3.5) to

$$J[f] = (\mathbf{y} - \Phi\mathbf{w})^T(\mathbf{y} - \Phi\mathbf{w}) + h\mathbf{w}^T\mathbf{w}. \tag{3.7}$$

This form of regularization is called ridge regression. Minimizing (3.7), we find ([Hasti Tibshirani, 2009] pp. 64)

$$\bar{\mathbf{w}} = (\Phi^T\Phi + hI)^{-1}\Phi^T\mathbf{y} \tag{3.8}$$

We can observe several things about this result. Firstly, since $f(x) = \phi(x)^T\mathbf{w}$, we may define $s(x)^T = \phi(x)^T(\Phi^T\Phi + hI)^{-1}\Phi^T$ so that we may write $\hat{f}$ as a linear smoother, that is

$$\hat{f}(x) = s(x)^T\mathbf{y}.$$

One might find (3.8) hard to interpret. To understand what effect ridge regression has, its useful to relate the technique to principle component analysis. Firstly, remember that the singular value decomposition of $\Phi$ is

$$\Phi = \mathrm{UDV}^T,$$

where U and V is orthogonal marries and D a diagonal matrix. We find that

$$\begin{aligned} \Phi\bar{\mathbf{w}} &= \Phi(\Phi^T\Phi + hI)^{-1}\Phi^T\mathbf{y} \\ &= \mathrm{UD}(\mathrm{D}^2 + hI)^{-1}\mathrm{DU}^T\mathbf{y} \\ &= \sum_j \mathbf{u}_j \frac{d_j^2}{d_j^2 + h}\mathbf{u}_j^T\mathbf{y}. \end{aligned} \tag{3.9}$$

So ridge regression computes coordinates $\mathbf{y}$ with respect to the orthogonal basis $\{\mathbf{u_i}\}$ ([Hasti Tibshirani, 2009] p. 66). Each component in $\mathbf{y}$ along $\mathbf{u}$ is shrieked relative to $d_j^2$

([Rasmussen Williams, 2006] p. 25). But what is $d_j^2$? Form the eigendecomposition of $\Phi^T\Phi$ we find

$$\Phi^T\Phi = \mathrm{V}\mathrm{D}^2\mathrm{V}^T$$

so each $d_j^2$ is an eigenvalue corresponding to the eigenvector $v_j$ of $\Phi^T\Phi$. The eigenvectors $\{\mathbf{v_j}\}$ is called the principle component of $\Phi$ and can be thought a set of orthogonal vectors along witch most of the variance within the data can be explained. The amount of variance along this axis is relative to the size of $d_j^2$ ([Hasti Tibshirani, 2009] p. 64-66). Because of this, ridge regression shrink those coordinate that explain less of the variance compared to those that explain more. The shrinkage of all coordinates is amplified by choosing larger values of $h$.

We may assume that with more data point and better performance we may become more confident in our estimate. Yet, neither regularization nor local regression has any build in tools that quantifies uncertainty about their estimations. In Section 4.1 we will learn that variance is a natural part of Gaussian processes, which gives us a easy way to build confidence bands. In the case of the algorithmic linear smoother models of this section, we need instead to estimate the variance of the model.

## 3.5 Confidence Bands

A confidence band is a confidence interval on the form

$$\hat{f}(x) \pm z_{\alpha/2}\sqrt{\sigma_n^2 \mathrm{Var}[\hat{f}(x)]}$$

mapped onto the response for every value of x, like in figure 10. $z_{\alpha/2}$ is the $\alpha/2$ quantile value of the normal distribution. To build a confidence interval, we need to find a expression of the variance of $\hat{f}$ and an estimate of $\sigma_n^2$. Starting with $\sigma_n^2$, since

$$\mathrm{E}[(\mathbf{y} - \hat{f}(\mathbf{x}))^T(\mathbf{y} - \hat{f}(\mathbf{x}))] = \mathrm{tr}(\mathrm{Var}[\mathbf{y} - \hat{f}(\mathbf{x})]) + (\mathrm{E}[\mathbf{y} - \hat{f}(\mathbf{x})])^T(\mathrm{E}[\mathbf{y} - \hat{f}(\mathbf{x})])$$

where $\mathbf{x}$ and $\mathbf{y}$ are vectors of each data point. We use that $\mathrm{Var}[\mathbf{y}] = \mathrm{Var}[\epsilon] = \sigma_n^2$. So

$$\mathrm{Var}[\mathbf{y} - \hat{f}(\mathbf{x})] = (I - \mathrm{S})^T\sigma_n^2(I - \mathrm{S})$$

using that $\hat{f}(\mathbf{x}) = \mathrm{S}\mathbf{y}$, where S is matrix with $\mathrm{S}_{ij} = s_j(x_i)$. Let $v = \mathrm{tr}(\mathrm{S})$, $\tilde{v} = \mathrm{tr}(\mathrm{S}^T\mathrm{S})$ and $\mathbf{b} = \mathrm{E}[\mathbf{y} - \hat{f}(\mathbf{x})]$. We see then that

$$\mathrm{tr}(\mathrm{Var}[\mathbf{y} - \hat{f}(\mathbf{x})]) = \sigma_n^2\mathrm{tr}(I - 2\mathrm{S} + \mathrm{S}^T\mathrm{S}) = \sigma_n^2(n - 2v + \tilde{v})$$

So

$$\sigma_n^2 = \frac{\mathrm{E}[(\mathbf{y} - \hat{f}(\mathbf{x}))^T(\mathbf{y} - \hat{f}(\mathbf{x}))] - \mathbf{b}^T\mathbf{b}}{n - 2v + \tilde{v}}$$

Let

$$\hat{\sigma}_n^2 = \frac{(\mathbf{y} - \hat{f}(\mathbf{x}))^T(\mathbf{y} - \hat{f}(\mathbf{x}))}{n - 2v + \tilde{v}} \tag{3.10}$$

be our estimator of $\sigma_n^2$. With larger sample size, we may assume that $\mathrm{E}[\hat{f}(\mathbf{x})]$ approaches $\mathrm{E}[\mathbf{y}]$, so $\mathbf{b} \to \mathbf{0}$. So for large n we have $\mathrm{E}[\hat{\sigma}_n^2] = \sigma_n^2$. Because of this, we may assume that (3.10) is a good estimator of $\sigma_n^2$. Since $\hat{f}$ is on the form (3.1), we see that

$$\mathrm{Var}[\hat{f}(x)] = s(x)^T\mathrm{Var}[y]s(x) = \sigma_n^2\|s(x)\|^2$$

If we use (3.10) to estimate $\sigma_n^2$ we find that the confidence interval of $\hat{f}$ is of the form

$$\hat{f}(x) \pm z_{\alpha/2}\hat{\sigma}_n\|s(x)\| \tag{3.11}$$

If we would instead like a confidence band for a noisy predictions, $\hat{y}$, we may use that $\text{Var}[\hat{y}] = \text{Var}[\hat{f}(x)] + \text{Var}[\epsilon] = \sigma_n^2\|s(x)\|^2 + \sigma_n^2$. This is called a *prediction interval* ([James, Witten, Hasti, Tibshirani, 2021] pp. 82), and is given by

$$\hat{y} \pm z_{\alpha/2}\hat{\sigma}_n\sqrt{1 + \|s(x)\|^2} \tag{3.12}$$

In Bayesian inference we build confidence intervals in the same way, only that we don't need to estimate $\sigma_n^2$ since it is already build into the model and $\text{Var}[\hat{f}(x)]$ has a different form.

# 4 Bayesian Inference

A common way of data modeling is by Bayesian inference, where we first specify our prior beliefs about the nature of the function, then update our beliefs after observing data. More specifically, our prior beliefs – which we call priors – specify what kind of function we think $f$ is and what its variance and mean is. With more data, we can become more certain about the behavior of $f$, especially around each data point. We call the distribution of $f$ before observation the prior, and the distribution of $f$ after observations the posterior. Both the posterior and prior specify the probability of any one possible function being the true function. In particular, our posterior distribution gives us a space of functions that are most likely to be the true function, given the information from the observations we have. Of the infinite number of these functions, the most likely of these functions is the mean of our distribution, $\bar{f}$. If we assume that the observations are noise free, the only possible value of all function at any particular data point, $(x, y)$, is $f(x) = y$. Because of this, any possible function must intersect all data point. However, the focus of this section is going to be on the case where we have noisy observations.

To better understand the concepts of Bayesian Inference it might be useful to know that, according to neurologist Anil Seth, we perceive the world in a way closely tied to Bayesian inference [Seth, 2021]. Consider this example: When walking outside, your prior knowledge would make you assume that it is very unlikely to observing a gorilla walking down the street. So if you where to observe a furry creature in the distance, you would probably assume it to be a rather large dog instead of a Gorilla. When you come closer, you see that it is even larger than expected and has a humanoid shape. Because of this you update your posterior prediction and instead think it is a human in a gorilla costume. When you get closer still, you see that you where mistaken and that it is indeed a gorilla – now you should probably run away. If the circumstances had change, and you are in a zoo instead of walking on the street, your prior estimated probability of observing a gorilla is much higher, and therefor you would make the correct prediction of observing a gorilla with less sensory information. In this way, our predictions are both dependent on our prior beliefs and the noisy data we perceive. This applies to all sensory prediction, from predicting the path of a tennis ball, to decoding the subtle ques from of your date and guess if he/she likes you or not. This insight is the foundation on which Bayesian inference regression models are build upon.

## 4.1 Linear Bayesian Inference

We are still looking at the case where we assume that the true distribution of $y$ is in the form (2.1). Instead of merely approximating $f$, we would like to make assumptions about $f$ and build

a model. In this section, we are going to assume that $f$ takes the form of a Gaussian process. There is two main ways of interpreting Gaussian Processes (GP) ([Rasmussen Williams, 2006] pp. 7). The view that is the easiest to understand is called the weighed space view, where we assume that $f(x)$ is some weighed linear combination of some transformation of $x$. That is

$$f(x) = \phi(x)^T \mathbf{w}, \tag{4.1}$$

where $\phi(x)$ and $\mathbf{w}$ is p dimensional. In Bayesian inference, before we make any predictions, we need to make some assumptions about $\mathbf{w}$, what we call a prior. In GP's, we assume that $\mathbf{w}$ is normally distributed. Before looking at the data, we need to specify the parameters of $\mathbf{w}$. In particular, we need to define a covariance matrix, $\Sigma_p$, on the weights. We can safely take the mean we to be zero, since it does not influence the behavior of $f(x)$ inside the interval our data is located on. As we will see, our estimated function, $\hat{f}$, will naturally adjust it's mean to the most likely value at any one point after performing inference on the data. Form this, the prior becomes

$$\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \Sigma_p),$$

where $\mathbf{0}$ denotes the zero vector. The most important feature of $f$ is it's covariance matrix, which describes the behaviour of $f$. From (4.1) and the distribution of $\mathbf{w}$, we see that $\text{Cov}[f(x), f(x')] = \phi(x)^T \Sigma_p \phi(x')$. We use a covariance function, $k(x, x')$, to calculate the covariance between $f(x)$ and $f(x')$. Covariance functions can be though of as quantifying the similarity between two data points, $x$ and $x'$ ([Rasmussen Williams, 2006] pp. 79). Under the assumption that points that a closer to each other are also more similar to each other. In this perspective, $k(x, x')$ can be though of an inverse distance measure and has the same role as the weight function in local regression from 3.2. Indeed, the names covariance function and kernel is often used interchangeably, which is the reason that they are both denoted with $k$ ([Rasmussen Williams, 2006] pp. 12). In the weight space view, because of the form of the covariance function, it is natural to take $k$ to be a inner product in a higher dimensional feature space by $k(x, x') = \phi(x)^T \Sigma_p \phi(x')$. For example, consider the case when $\phi(x) = \begin{bmatrix} 1 & x \end{bmatrix}$. Then $f(x) = w_1 + w_2 x$, so $f$ can be any linear function. However, since we draw $\mathbf{w}$ from a normal distribution, $w_1$ and $w_2$ is unlikely to be far away from 0. That is, the density of $f$ is larger around zero than further away. Figure 8 shows 20 such realizations.

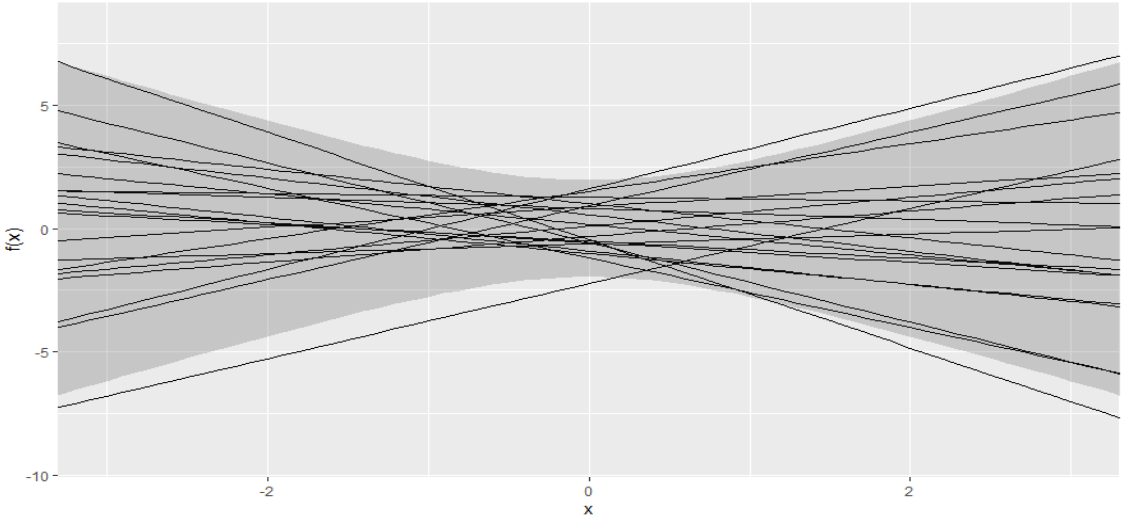



Figure 8: 20 realizations of a linear function with weights drawn from a Gaussian distribution with mean 0 and variance 1. The gray area is a 95% confidence band.

In this picture, the weights is assumed to be zero mean and have a variance of one, that is $\mathbf{w} \sim \mathcal{N}_2(\mathbf{0}, I_2)$. Because of this, the confidence bands is calculated with $\hat{f}(x) \pm z_{\alpha/2}\sqrt{\text{Var}[\hat{f}(x)]}$, where $\text{Var}[\hat{f}(x)] = k(x,x) = [1,x] \cdot [1,x] = 1 + x^2$.

It is important to note that this function is a parametric function, since we have predefined to number of feature we want to use. In this case, we use the feature $\phi_1(x) = 1$ and $\phi_2(x) = x$. In the non-parametric approach, which is the focus of this text, the number of features is tied to the number of data points. This requires the use of an infinite dimensional feature space. In this case, calculating $k(x,x') = \phi(x)^T \Sigma_p \phi(x')$ is obviously infeasible, since $\phi(x)$ is of infinite dimensions. Instead, we more generally require $k$ to be a symmetric and positive semi-definite function. By symmetric we need $k(x,x') = k(x',x)$. A function is said to be positive semi-definite if the Gram matrix, K, defined by $K_{ij} = k(x_i, x_j)$ is positive semi-definite ([Bartlett, 2008] pp. 2). If $k$ is a covariance function, we call K a covariance matrix ([Rasmussen Williams, 2006] pp. 80). Since $k$ is symmetric, the matrix K must also be symmetric. These properties are obviously satisfied when $k$ is on the form of a inner product. An example of a covariance function defined on a infinite dimensional feature space is the Gaussian kernel, also called the radial basis function ([Rasmussen Williams, 2006] pp. 14)

$$k(x,x') = \sigma_f e^{-\frac{(x-x')^2}{2h^2}}. \tag{4.2}$$

We use $\Sigma = \sigma_f I$, an infinite dimensional matrix. $\sigma_f$ quantifies the variance of $f$ and must not be confused with $\sigma_n$, which is the variance of the error $\epsilon$. $h$ can be recognized as the bandwidth that was discussed in 3.2. In the probabilistic view on machine learning, $h$ can be thought of as specifying the distance at which each point is correlated. If we take $h$ to be large, then a higher number of pairs $x_i$ and $x$ has a non-zero covariance.

It is not at all obvious how (4.2) defines a inner product on an infinite dimensional feature space. Indeed, the Gaussian kernel does not seem related to inner products at all. However, this

could not be further from the truth. In Section 6.1, we will explore the relationship between the Gaussian kernel and inner products, and how this relationship is not unique, but is instead general to a large class of symmetric and positive definite functions.

Returning to parametric case when $f$ is on the form of (4.1). Inference in the Bayesian linear model is based on the posterior distribution over the weights given by the conditional likelihood of $\mathbf{w}$ given our data. This is computed using Bayes rule ([Rasmussen Williams, 2006] pp. 9)

$$p(\mathbf{w}|\mathbf{y},\mathbf{x}) = \frac{p(\mathbf{y}|\mathbf{x},\mathbf{w})p(\mathbf{w})}{p(\mathbf{y}|\mathbf{x})}. \tag{4.3}$$

By estimating each of the terms on the right hand side in (4.3), we may obtain the posterior, $p(\mathbf{w}|\mathbf{y},\mathbf{x})$. For $p(\mathbf{y}|\mathbf{x},\mathbf{w})$, we use the likelihood to get

$$\begin{aligned} p(\mathbf{y}|\mathbf{x},\mathbf{w}) &= \prod_{i=1}^{n} p(y_i|x_i,\mathbf{w}) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_n} \exp\left(-\frac{(y_i - \phi(x_i)^T\mathbf{w})^2}{2\sigma_n}\right) \\ &= \frac{1}{(2\pi\sigma_n^2)^{n/2}} \exp\left(-\frac{1}{2\sigma_n^2}|\mathbf{y} - \phi(\mathbf{x})^T\mathbf{w}|^2\right) \sim \mathcal{N}(\phi(\mathbf{x})^T\mathbf{w}, \sigma_n^2 I). \end{aligned} \tag{4.4}$$

$p(\mathbf{y}|\mathbf{x})$ is the marginal likelihood and is given by

$$p(\mathbf{y}|\mathbf{x}) = \int p(\mathbf{y}|\mathbf{x},\mathbf{w})p(\mathbf{w})d\mathbf{w}. \tag{4.5}$$

As the term $p(\mathbf{y}|\mathbf{x})$ suggests, this is the probability the true model is $\mathbf{y}$, given the data, $\mathbf{x}$, we have observed. This is often called the evidence, since it can be used to evaluate the performance of our regression model. If $p(\mathbf{y}|\mathbf{x})$ is low, it's unlikely that we have found a good predictive model. For this reason, it plays a similar role to that of cross-validation. Combining (4.4) and (4.5) in (4.3) and using that $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \Sigma_p)$ we get [Rasmussen Williams, 2006]

$$p(\mathbf{w}|\mathbf{y},\mathbf{x}) \propto \exp\left(-\frac{1}{2\sigma_n}(\mathbf{y} - \phi(\mathbf{x})^T\mathbf{w})^T(\mathbf{y} - \phi(\mathbf{x})^T\mathbf{w})\right) \exp(-\frac{1}{2}\mathbf{w}^T\Sigma_p^{-1}\mathbf{w}). \tag{4.6}$$

It so happens that this pdf is closely related to ridge regression. After we take the negative logarithm of $p(\mathbf{w}|\mathbf{y},\mathbf{x})$, the first term can be viewed as a least square penalty and therefor penalizing high bias. Since $\Sigma_p$ is symmetric positive definite, the other term can be viewed as a weighted norm of the coefficients $\mathbf{w}$. From the theory of relaxation, we know that this punishes complexity, and therefor penalizes high variance. This relationship going to be further explored in Section 6.

With a density of $\mathbf{w}$ given our date $\mathbf{x}$ and $\mathbf{y}$, we have information about how probable different forms of $\mathbf{w}$ is, and by extension, different forms of $f(x)$. The most probable form of $f$ is the one that uses the $\mathbf{w}$ with the highest probability. This is found by maximizing (4.6) w.r.t $\mathbf{w}$ and is called the maximum a posterior (MAP). We find that the MAP is $\bar{\mathbf{w}} = \sigma_n^{-2}(\sigma_n^{-2}\phi(\mathbf{x})\phi(\mathbf{x})^T + \Sigma_p^{-1})^{-1}\phi(\mathbf{x})\mathbf{y}$, where the bar denotes the maximum. This can be further simplified by using kernels. Defining $\mathrm{K} = \Phi^T\Sigma_p\Phi$ we may write $\bar{\mathbf{w}} = \Sigma_p\Phi(\mathrm{K} + \sigma_n^2 I)^{-1}\mathbf{y}$. After some simplification of (4.6) it is easy to see that ([Rasmussen Williams, 2006] pp. 9)

$$p(\mathbf{w}|\mathbf{y},\mathbf{x}) \sim \mathcal{N}\left(\bar{\mathbf{w}}, (\mathrm{K} + \sigma_n^2 I)^{-1}\right). \tag{4.7}$$

So the expected value of the posterior is also the MAP. We may use the posterior to predict future outcomes, $\mathbf{y}^*$, with any input $\mathbf{x}^*$. In particular, we know that the most probable value of

$f(x^*)$ is $\bar{f}(x^*) = \phi(x^*)^T \bar{\mathbf{w}}$, as given by the MAP. Because

$$\phi(x^*)^T \Sigma_p \Phi = \sum_{i=1}^{n} k(x^*, x_i)$$

We see that $\bar{f}$ has the form

$$\bar{f}(x^*) = \sum_{i=1}^{n} \alpha_i k(x^*, x_i), \tag{4.8}$$

where $\boldsymbol{\alpha} = (K + \sigma_n^2 I)^{-1}\mathbf{y}$. This form going to be important in Section 6 – where I compare the two approaches – because it is possible to show that both the MAP and solutions of regularization has this form. If we take $\alpha_i = \left(\sum_{j=1}^{n} k(x_i, x_j)\right)^{-1}$, we get the N-W estimator, so local regression has the same form.

## 4.2 Gaussian Processes

An alternative method of reaching equivalent results as with the weight space view is through the function space view. In this view, $f$ is no longer assumed to be parametric. That is, we assume that $\phi$ is of infinite dimension. Because of this, $k$, our covariance function, needs to be computed on an infinite dimensional feature space, like described with the Gaussian kernel. While assuming $f$ has a non-parametric form, we may use Gaussian processes (GP) to describe the distribution over functions. Formally ([Rasmussen Williams, 2006] pp. 13):

**Definition 4.1.** *A Gaussian process is a collection of random variables, any finite number of which have a joint Gaussian distribution.*

For a more general approach, assume that we use a d-dimensional input vector, $\mathbf{x}$, so that our data is $\mathrm{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]$. We define a mean function $m(\mathbf{x}) = \mathrm{E}[f(\mathbf{x})]$ and a covariance function $\mathrm{Cov}[f(\mathbf{x})]$. It was previously stated that the mean function does not influence the behavior of $\hat{f}$. This holds true in areas of interpolation, where data points has been observed, but is not the whole truth for extrapolation and on the boundary of our function. For any prediction point far away from our data we have little or no information about the nature of $f(x)$, so our best guess reverts back to the prior. That a problem may arise around the boundary is not obvious in the data introduced in Section 2.1, since this data is clearly centered around zero. A better example is shown in Figure 9, which depicts the logarithm of the wage of Canadian male workers in 1971 with a common education (13 years) relative to their age.

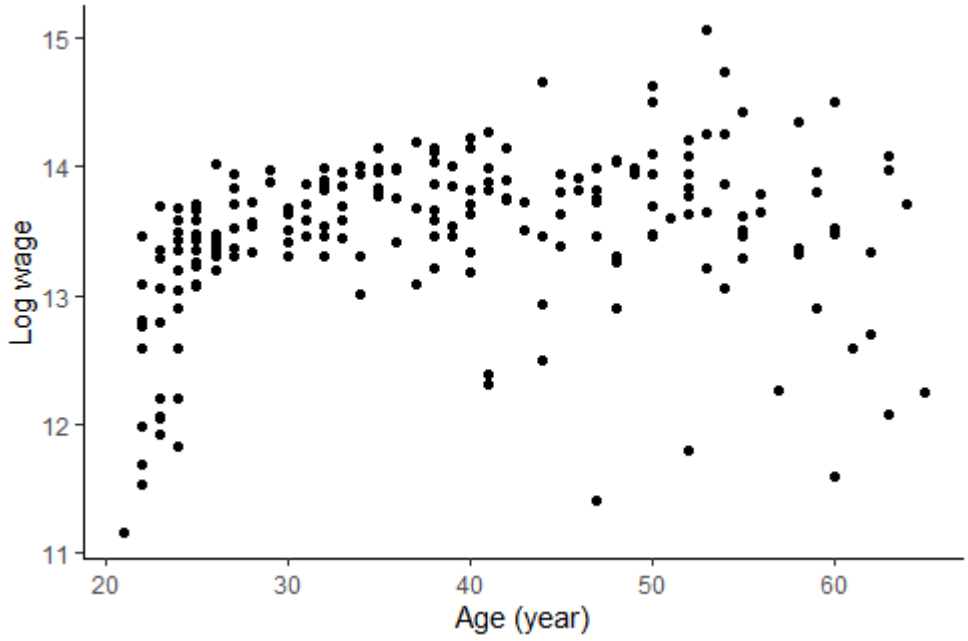



Figure 9: Logarithm of wage relative to age

We see that the data is clearly centered above zero. If we where to assume that the prior mean is zero, the predicted function would need to take a sharp turn from zero up to the predicted mean of the function on the left side of the boundary. Similarly, the function would quickly regress back to zero on the right boundary. An example of this is shown in Figure 15 in Section 5.3. To account for this, we allow $m(\mathbf{x})$ to be non-zero. From this we have that

$$f(\mathbf{x}) \sim \mathcal{N}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x})).$$

So $f$ has a multivariate Gaussian distribution. As usual, we assume that we have noisy observations, like in (2.1), so we have $\mathbf{y} \sim \mathcal{N}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}) + \sigma_n^2 I)$. We want to predict observations in new points $\mathrm{X}_* = [\mathbf{x}_1^*, \ldots, \mathbf{x}_m^*]$, with $\mathbf{f}_* = [f(\mathbf{x}_1^*) \ldots f(\mathbf{x}_m^*)]$. $f_*$ should have the same distribution as $f$, so we may assume that

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left( \begin{bmatrix} m(\mathbf{x}) \\ m(\mathbf{x}^*) \end{bmatrix}, \begin{bmatrix} k(\mathrm{X}, \mathrm{X}) + \sigma_n^2 I & k(\mathrm{X}, \mathrm{X}_*) \\ k(\mathrm{X}_*, \mathrm{X}) & k(\mathrm{X}_*, \mathrm{X}_*) \end{bmatrix} \right).$$

We may use conditional likelihood for multivariate Gaussian distribution to find [Rasmussen Williams, 2006]

$$\bar{\mathbf{f}}^* = \mathrm{E}[\mathbf{f}^* | \mathrm{X}, \mathbf{y}, \mathrm{X}_*] = k(\mathrm{X}_*, \mathrm{X})[k(\mathrm{X}, \mathrm{X}) + \sigma_n^2 I]^{-1}(\mathbf{y} - m(\mathbf{x})) + m(\mathbf{x}^*) \tag{4.9}$$

and

$$\mathrm{Cov}[\mathbf{f}_*] = k(\mathrm{X}_*, \mathrm{X}_*) - k(\mathrm{X}_*, \mathrm{X})[k(\mathrm{X}, \mathrm{X}) + \sigma_n^2 I]^{-1} k(\mathrm{X}, \mathrm{X}_*). \tag{4.10}$$

This is the exact same result as in (4.8), because $k(\mathrm{X}, \mathrm{X}) = \mathrm{K}$ and $k(\mathrm{X}_*, \mathrm{X}) = \phi(x^*)^T \Sigma_p \Phi$.

The posterior has reduced uncertainty around each observation. In particular, if we assume noise free observations, we know that $f$ must interpolate each data point, so we force $\hat{f}$ to do

the same. An example of this is shown in Figure 10, which depicts samples from both a prior and a posterior of a Gaussian distribution.

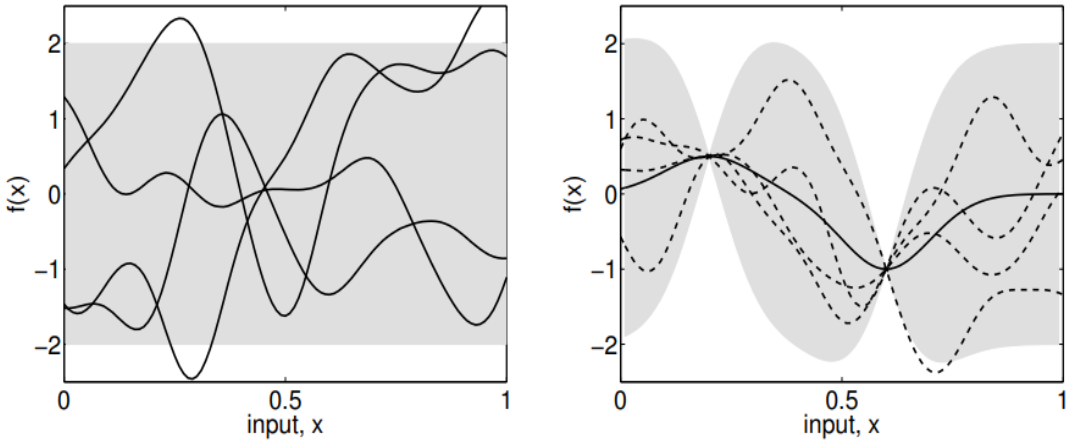



Figure 10: Panel a) shows four samples drawn from a Gaussian prior distribution. Panel b) shows 4 samples drawn from the posterior of the same distribution (dashed line) and the MAP (bold line). The gray area in both panels shows a 95% confidence band for both densities. (Source: [Rasmussen Williams, 2006] pp. 3)

Because of this effect, with more observation we limit the space of likely forms of $\hat{f}$, enabling us to be more certain about it's true form, given that our assumptions about it in the prior is correct. We may use the marginal likelihood to evaluate the validity of our assumptions.

## 4.3 Marginal Likelihood

Marginal likelihood defined by (4.5) is a tool that helps us quantify the performance of our model given our data. This is useful when comparing different models to each others and when tuning hyper-paramters, such as the bandwidth $h$ and $\sigma_n$. We would like a different form of (4.5) that is more easy to calculate. Since $p(\mathbf{f}|\mathrm{X}) \sim \mathcal{N}(\mathbf{0}, \mathrm{K})$ we have

$$\log p(\mathbf{f}|\mathrm{X}) = -\frac{1}{2}\mathbf{f}^T\mathrm{K}^{-1}\mathbf{f} - \frac{1}{2}\log|\mathrm{K}| - \frac{n}{2}\log(2\pi).$$

Combining this with this the fact that $\mathbf{y}|\mathbf{f} \sim \mathcal{N}(\mathbf{f}, \sigma_n^2 I)$ we find that ([Rasmussen Williams, 2006] pp. 19)

$$\log p(\mathbf{y}|\mathrm{X}) = -\frac{1}{2}\mathbf{y}^T(\mathrm{K} + \sigma_n I)\mathbf{y} - \frac{1}{2}\log|\mathrm{K} + \sigma_n I| - \frac{n}{2}\log 2\pi. \tag{4.11}$$

This equation has an important relationship to the bias-variance trade-off principle. In fact, each term in (4.11) can be related to the terms in the bias-variance equation (2.3). The first term, $\mathbf{y}^T(\mathrm{K} + \sigma_n I)\mathbf{y}$, is called the square error and is relates to the bias. It grows when the distance from the function to the data points is large. The second term, $\log|\mathrm{K} + \sigma_n I|$, is called model complexity and relates to variance. The determinant of K grows when the prediction function

is complex, causing more variance. A different name for this term is the Occam factor, named after William of Occam. Occam is known for Occams razor, a principle which states that one should always prefer the simplest explanation to a problem. This coincides with the role of the Occam factor, since the term punishes models which increases complexity without decreasing the square error – so simpler functions are preferred [Hennig, 2020]. Lastly, $\frac{n}{2}\log 2\pi$, is not related to K and can be viewed as a irreducible error.

Because of this relationship, marginal likelihood is maximal when the model is fine tuned, so that the bias-variance trade-off is in it's Goldilocks zone. For this reason, we would like to maximize (4.11) with respect to the hyper-parameters. Unfortunately, this optimization problem does not have a analytical solution, like we found when optimizing (4.6). Instead, we need to gradient descent methods like BFGS to search for optimal solution. To optimize the hyper-parameters is the "learning" part of machine learning. In the next section, we will see how this learning may be done in practice.

# 5 Comparison of Gaussian Processes and Local Regression with examples

We now have enough insight into local regression and Gaussian processes to use them on the first data set, which where presented in Figure 1, and the second data set, that where presented in Figure 9.

## 5.1 Implementation of Local Regression

The implementation of local regression, using has the N-W, tree important steps. Firstly, we need to optimize the bandwidth. Then we fit the model using the optimal bandwidth. Lastly, we estimate total variance and make a 95% confidence band. Since the estimated function on the form (3.1), it is linear and we can therefor use LOOCV to efficiently estimate the error with ([Hasti Tibshirani, 2009] pp. 161)

$$\text{LOOCV}(\hat{f}) = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_i - \hat{f}(x_i)}{1 - s_i(x_i)}\right)^2, \tag{5.1}$$

where $s_i(x_i)$ is a smoothing function from Section 3. $s(x)$ is with (3.4), by using the N-W kernel estimator. Tuning the bandwidth amounts using a optimization algorithm to find the $h$ that minimizes (5.1). With the smoothing vector, we calculate $\hat{f}$ with (3.3). Lastly, we estimate the variance of $\mathbf{y}$ with (3.10) and estimate the confidence interval of $\hat{f}$ with (3.11). Alternatively, we can estimate the prediction interval with (3.12).

## 5.2 Implementation of Gaussian Processes

Similar to local regression, the implementation of GP regression has 3 parts. Firstly, we need to optimize the parameters. Unlike local regression, which need to only optimize the bandwidth, GP regression require us to optimize 4 hyper-parameters. This include the bandwidth, but also the variance of $\epsilon$ in (2.1), $\sigma_n$, the variance of $f$, $\sigma_f$, from (4.2) and lastly the mean, $m(x)$. We again use a optimization algorithm. The other two steps are more closely tied than in local regression. We simply use (4.9) to find our estimated function and (4.10) to find the covariance matrix of our estimated function. The variance vector of the estimated function is the diagonal of the covariance matrix. Then we make confidence bands with $\hat{f}(x) \pm z_{\alpha/2}\sqrt{\text{Var}[f]}$.

## 5.3 Practical comparison

In all these regressions, a confidence band for $\hat{y}$ is used. When used on the first data set, we get the following the regression graph when using N-W kernel estimator.

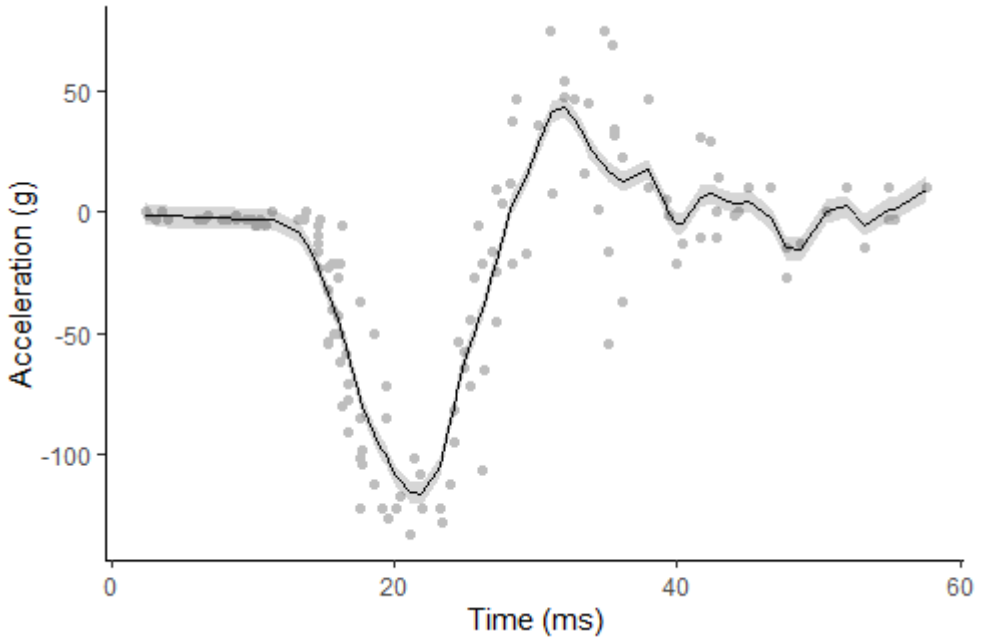



Figure 11: N-W kernel estimator used on head acceleration data. The bold line is the estimated function, while the gray area is a 95% confidence band.

The MSE of this regression is estimated to be 444.485 and the running time is 0.872 seconds. While when using GP we get the following result

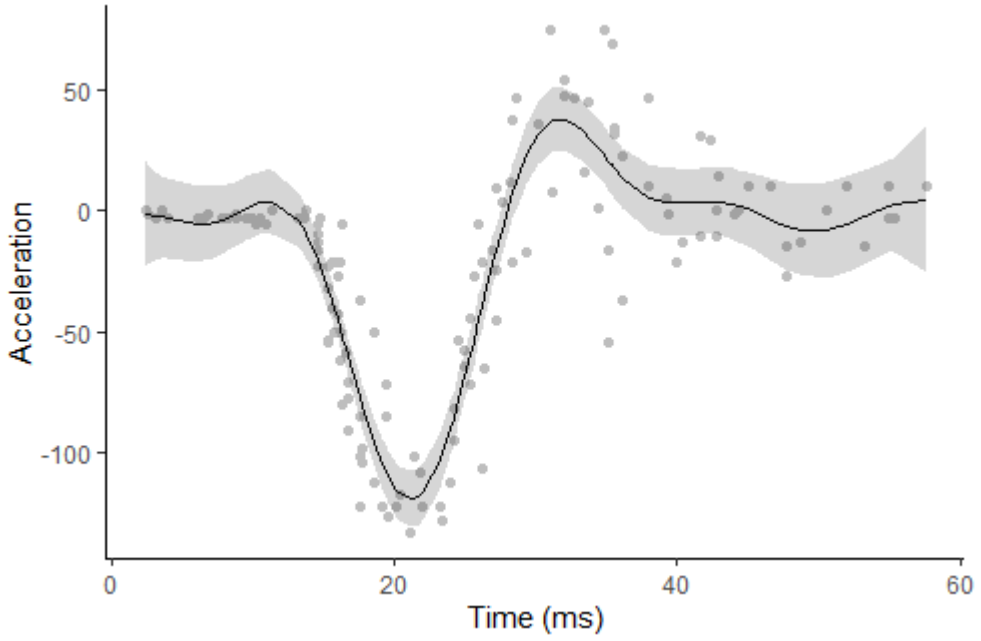



Figure 12: Gaussian Process regression used on head acceleration data. The bold line is the MAP, while the gray area is a 95% confidence band.

This time, the regression is 466.972 and running time is 5.392 seconds. Maybe the most visually obvious difference is the rough shape of the curve generated by the N-W kernel estimator, compared to the more smooth shape of Gaussian process curve. This is to be expected, since the Gaussian process is made using a series of correlated normal distributed variables. Because of this, each point along the function is unlikely to make any large jumps in value relative to the previous point. Kernel regression does not have this restriction. This higher degree of freedom is reflected in the slightly stronger performance of local regression than GP. This difference in performance is maybe magnified by the fact that the GP makes the assumption that the sample variance, $\sigma_n^2$, is constant in all time intervals. This assumption is probably not justifiable, since the sample variance seems to be quite small at the beginning and then increases firstly at the first jump around time 15, and then increases even further when the acceleration seems to revert back to zero, at around time 30. Since local regression does not make assumptions about sample variance, this problem does not affect it's result.

While the GP regression loses some performance by making the sample variance assumption, the same assumption significantly improves the quality of it's confidence bands. The main difference between the confidence bands of the two methods is their size. The confidence bands of the GP regression seems to correctly cover closer to 95% of the data, unlike local regression, which covers a very small portion of the data. This suggest that method for which the confidence bands estimated with proved to be quite ineffective. Unlike GP's, local regression is free to choose any method for estimating confidence bands, some of which might perform much better than the method chosen here.

Lastly, the local regression was significantly faster than the GP regression: 0.872 seconds compared to 5.392 seconds. This is due to the fact that GP's needs to first calculate the matrix $(\mathrm{K} + \sigma_n^2 I)$, then invert it. Calculate a matrix $n \times n$ takes $\mathcal{O}(n^2)$ time, while inverting it takes

$\mathcal{O}(n^3)$ time. Local regression need only to calculate K, not invert it. This causes local regression to have a overall quadratic time complexity, while GP's have a cubic time complexity, which is reflected in their computation times. Techniques to approximate the inverse of the covariance matrix which improves the computation time of GP's. One such technique is sparse greedy Gaussian process regression, which has a computation time of $\mathcal{O}(m^2 n)$, where n is the sample size and $m \ll n$ [Smola Bartlett, 2000].

When we perform the two regression techniques on the wage data set we get the following results

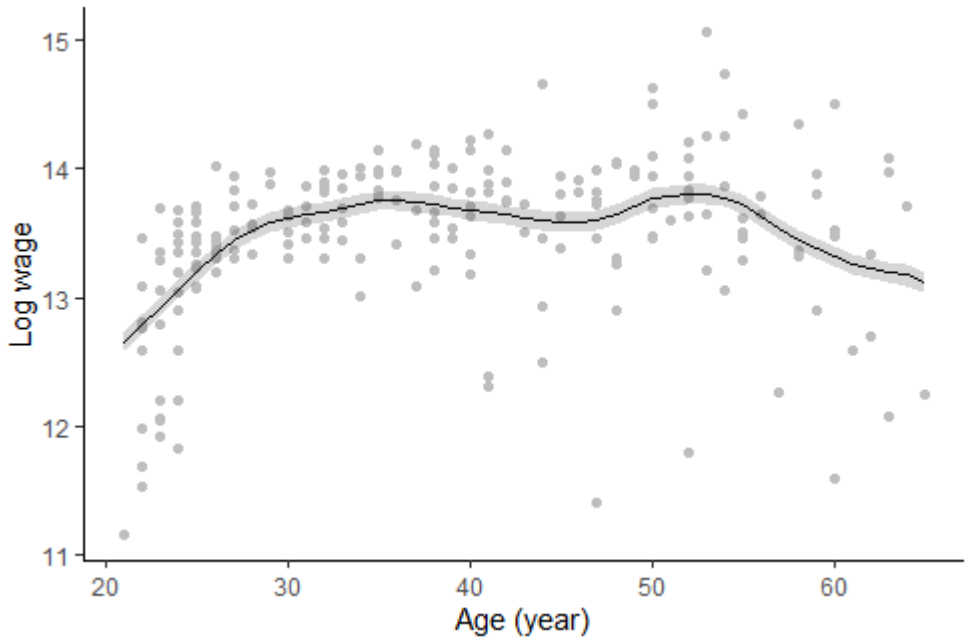



Figure 13: N-W kernel estimator regression used on logarithm of wage relative to age data set. The bold line is the estimated function, while the gray area is a 95% confidence band.

for the N-W kernel estimator, which has a MSE of 0.282 and a running time of 1.574 seconds. For the GP regression we get

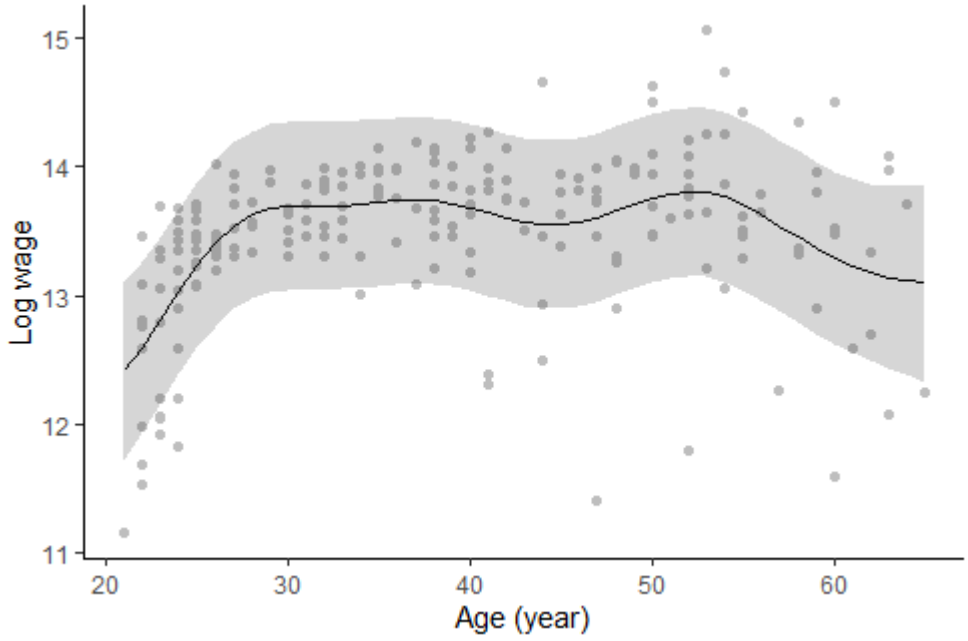



Figure 14: Gaussian process regression used on logarithm of wage relative to age data set. The bold line is the estimated function, while the gray area is a 95% confidence band.

This time, the MSE is 0.274 and the running time is 17.263 seconds. Surprisingly, the GP regression performed slightly better than the N-W kernel estimator. This might be because the assumption of a constant variance seems to be more accurate for this data set then for the head acceleration data set. The difference in computation time where significantly larger with this data set then with the head acceleration data set, due to the difference in size (133 data points compared to 205). Because of the quick growth, it would be computationally infeasible to perform this implementation of GP regression on a data set of a larger magnitude, say 1000 data points, unlike kernel regression, which would have a more acceptable computation time.

Lastly, as promised in Section 4.1, we can compare the performance of a zero mean GP regression to our current model

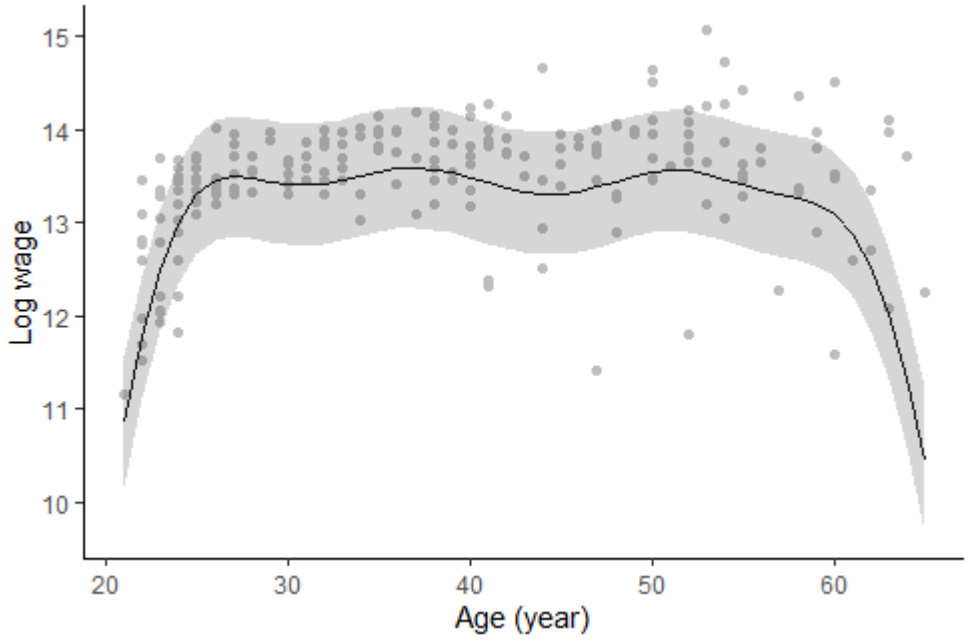



Figure 15: Same GP regression as in 14, except that the model uses a zero mean instead of an optimized mean.

As expected, the regression line takes a sharp turn up from and down to zero at the boundaries. This regression has a MSE of 0.389, so it performs significantly worse than the non-zero mean model.

The implement of regularization has been excluded from this text because of it's close relationship to GP regression. This relationship is going to be investigated more closely in the next section.

# 6 Theoretical Comparison of Regularization and GP's

This section is going to be an exploration of an important subclass of regularization problems on the form (3.5) that are generated by a positive definite kernel $k(x, x')$. The motivation for doing this is to relate solutions of our new regularization problem to that GP's, since both problems have solutions on the form of linear combinations of kernel functions. In Section 4 we saw that some kernels can be expressed as inner products. To obtain the relationship between regularization and GP's we need to further explore this property of kernels.

## 6.1 Kernels as Inner Products in a Possibly Infinite Dimensional Space

Kernels are central to the behavior of many non-parametric regression methods, including all techniques covered in this text. This is the reason for their name, kernel, meaning central or most important part of something. In local regression, kernels has the role of a inverse weight function. In Gaussian processes, kernels has the role of a covariance function. However, in regularization the role of kernels has not yet been made obvious. Understanding the role of kernels in a generalized form of regularization is essential in comparing Gaussian processes to

regularization. Both share the view that kernels are inner products in a higher dimensional space. Using a basis transformation into a feature space $\phi : \mathbb{R} \to \mathbb{R}^p$ and a positive definite $p \times p$ matrix $\Sigma_p$, we may use the definition of covariance functions to take $k(x, x') = \phi(x)^T \Sigma_p \phi(x')$, where $x, x' \in \mathbb{R}$ are two data points. If we allow the dimension of $\phi(x)$ to be very high, computing a kernel on this form is close to infeasible. Fortunately, many kernels allow us to simplify the computations without sacrificing complexity. This fact is often referred to as the kernel trick [Hasti Tibshirani, 2009]. For example, if we have a vector input, $\mathbf{x}$, we may consider the simplest form, where $\phi(\mathbf{x}) = \mathbf{x}$ and $\Sigma_p = I$, then $k(x, x') = \mathbf{x}^T \mathbf{x}'$. Going back to the case where $x$ is one dimensional, we may define $\phi(x) = [1, \sqrt{2}x, x^2]$. Then

$$k(x, x') = [1, \sqrt{2}x, x^2] \cdot [1, \sqrt{2}x', x'^2] = x^2 x'^2 + 2xx' + 1 = (xx' + 1)^2.$$

This can be further generalized to the case where $\mathbf{x}$ is of any dimension and the polynomial is of p degree. Then we have the polynomial kernel $k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}' + 1)^p$. The cost of computing $k$ on this form is relative to the dimensions of the input space, not the feature space, since most of the computation time comes from computing the inner product $\mathbf{x}^T \mathbf{x}'$ [Felzenszwalb, 2017]. This allows us to expand the feature space, and with it the complexity, without much increase in the computation cost. Still, the polynomial kernel has it's limitations. If we attempt to use an infinite dimensional feature space, $p \to \infty$, then we are no longer able to compute the polynomial kernel. Surprisingly, this is not always the case. More impressive yet, expanding to an infinite feature space can decrease the computation cost for computing the kernel. To give an example of this, consider the case where we want to do a regression on the interval $[c_{min}, c_{max}] \in \mathbb{R}$. We may choose a set of evenly spaced points, $\{c_1, \ldots, c_F\}$, on this interval and use the basis function $\phi(x) = \left[\exp\left(-\frac{(x-c_1)^2}{2h^2}\right), \ldots, \exp\left(-\frac{(x-c_F)^2}{2h^2}\right)\right]$, so that $F$ is the number of features. If we take $\Sigma_F = \sigma^2 \frac{c_{max} - c_{min}}{F} I$, then

$$\begin{aligned} k(x, x') &= \frac{\sigma^2 (c_{max} - c_{min})}{F} \sum_{k=1}^{F} \phi_k(x) \phi_k(x') \\ &= \exp\left(-\frac{(x - x')^2}{4h^2}\right) \frac{\sigma^2 (c_{max} - c_{min})}{F} \sum_{k=1}^{F} \exp\left(-\frac{(c_k - \frac{1}{2}(x + x'))^2}{h^2}\right). \end{aligned}$$

By expanding to a infinite number of features, $F \to \infty$, while still keeping each $c_k$ evenly spaced, we may recognize the sum as a Riemann sum, which is by definition $\int_{c_{min}}^{c_{max}} \exp\left(-\frac{(c - \frac{1}{2}(x+x'))^2}{h^2}\right) dc$. Furthermore, if we expand the interval $[c_{min}, c_{max}]$ to all of $\mathbb{R}$ with $c_{min} \to -\infty$ and $c_{max} \to \infty$, we get a Gaussian integral with the well know solution of $\sqrt{2\pi}h$. So we get that $k(x, x') = \sqrt{2\pi} h \sigma^2 \exp\left(-\frac{(x-x')^2}{4h^2}\right)$ [Hennig, 2020]. To summarize, we started out with a finite number of Gaussian features and by expanding to a infinite number of such features we increased the complexity of the kernel while simultaneously reducing the computation cost to that of computing a single Gaussian function, essentially obtaining the best of both worlds.

The Gaussian kernel is not alone in it's relationship to inner products in a infinite dimensional feature space. Shortly, we will see how Mercer's theorem allows us to write positive definite kernels as a sum of eigenfunctions, which in turn allows us to express a kernel as a inner product. We define any function $\psi$ that obeys

$$\int k(x, x') \psi(x) d\mu(x) = \lambda \psi(x)$$

to be a eigenfunction of $k$ with eigenvalue $\lambda$ with respect to a measure $\mu$. For example, if $x$ has a known probability measure, then we make take $d\mu(x) = p(x)dx$. If this is not the case,

we often use the Lebesgue measure [Rasmussen Williams, 2006]. There exists a infinite number eigenfunctions. Each of these functions are orthogonal, and may further choose to normalize them such that

$$\int \psi_i(x)\psi_j(x)d\mu(x) = \delta_{ij}, \tag{6.1}$$

where $\delta_{ij}$ is the Kronecker delta function, which is one when $i = j$ and zero otherwise. Mercer's theormen states that

**Theorem 6.1.** *(Mercer's theorem). Let $(\mathbb{X}, \mu)$ be a finite measure space and $k : \mathbb{X} \times \mathbb{X} \to \mathbb{R}$ be a continuous and positive definite kernel on $\mathbb{X}$ and $T_k : L_2(\mathbb{X}, \mu) \to L_2(\mathbb{X}, \mu)$ be a positive definite integral operator defined by*

$$(T_k f)(\cdot) \int_{\mathbb{X}} k(\cdot, x)f(x)d\mu(x)$$

*Let $\psi_i \in L_2(\mathbb{X}, \mu)$ be normalize eigenfunctions of $T_k$ associated with the eigenvalues $\lambda_i > 0$. Then:*

1. *$\{\lambda_i\}_{i=1}^{\infty}$ are absolute summable*
2. $$k(x, x') = \sum_{i=1}^{\infty} \lambda_i \psi_i(x)\psi_i(x')$$

   *holds almost $\mu^2$ everywhere, where the series converges absolute and uniformly $\mu^2$ almost everywhere*

([Rasmussen Williams, 2006] pp. 96, [Bartlett, 2008] pp. 3 and [Hennig, 2020]). Since the eigenfunctions are orthogonal, they are also linearly independent, so $\psi(x) = [\psi_1(x), \psi_2(x), \dots]$ can be viewed as a transformation into a infinite space of basis functions. Indeed, we may choose to use the transformation $\phi$ to be $\psi$. Furthermore, if we take $\Sigma$ to be a infinite diagonal matrix with diagonal elements $\Sigma_{ii} = \lambda_i$, we see that $k(x, x') = \sum_{i=1}^{\infty} \lambda_i \psi_i(x)\psi_i(x') = \psi(x)^T \Sigma \psi(x')$. So any positive definite kernel is a inner product on a infinite dimensional space.

## 6.2 Reproducing Kernel Hilbert Space

The inner product corresponding to a Mercer kernel has deep ties to a type of Hilbert space, $\mathcal{H}$, called a Reproducing Kernel Hilbert Space (RKHS). This Hilbert space is endowed with the inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ of functions on an index set $\mathbb{X}$, where ([Felzenszwalb, 2017] pp. 4)

$$k(x, y) = \phi(x)^T \Sigma \phi(x') = \langle \phi(x), \phi(x') \rangle_{\mathcal{H}}.$$

$\langle \cdot, \cdot \rangle_{\mathcal{H}}$ has the property that there exists a unique kernel so that for every $f \in \mathcal{H}$ we have

$$\langle f(\cdot), k(\cdot, x) \rangle_{\mathcal{H}} = f(x). \tag{6.2}$$

This is called the reproducing property of $k$. Also, for every $x$, $k(x, x')$ as a function of $x'$ belongs to $\mathcal{H}$. Because of this, $k$ has the reproducing property with itself [Rasmussen Williams, 2006]

$$\langle k(x, \cdot), k(x', \cdot) \rangle_{\mathcal{H}} = k(x, x'). \tag{6.3}$$

Moore-Aronszajn theorem states that for every kernel there exists a uniquely determined RKHS and vice versa ([Rasmussen Williams, 2006] p. 130). The main purpose of introducing KRHS is to use the norm $\|f\|_{\mathcal{H}}^2 = \langle f, f \rangle_{\mathcal{H}}$ as a generalization to the quadratic form $\mathbf{f}^T K^{-1} \mathbf{f}$

[Rasmussen Williams, 2006], so that we can use $\xi(f) = \|f\|_{\mathcal{H}}$ as the penalty term in regularization. This is a norm in feature space, instead of input space, like with $\mathbf{y}^T(\mathrm{K} + \sigma_n I)\mathbf{y}$.

We may define a RKHS with respect to a kernel $k$ that has a finite eigenfunction expansion, with eigenfunctions $\{\psi_1, \ldots, \psi_N\}$ and corisponding eignevalues $\{\lambda_1, \ldots, \lambda_N\}$. From Mercer's theorem we have that $k(x, x') = \sum_{i=1}^{N} \lambda_i \psi_i(x)\psi_i(x')$ and that the eigenfunctions are orthonormal. In this space, each function $f \in \mathcal{H}$ is a linear combination of the eigenfunctions of $k$. That is

$$f(x) = \sum_{i=1}^{N} c_i \psi_i(x) \tag{6.4}$$

with

$$\sum_{i=1}^{N} \frac{c_i^2}{\lambda_i} < \infty,$$

where $\{c_i\}_{i=1}^{N}$ is a sequence of coefficients. Let $g$ be another such function with coefficients $\{c_i'\}_{i=1}^{N}$. Then we may define the inner product on $\mathcal{H}$ as

$$\langle f, g \rangle_{\mathcal{H}} = \sum_{i=1}^{N} \frac{c_i c_i'}{\lambda_i}.$$

By taking each coefficient to be $\lambda_i \psi_i(x)$, we see that the function $k(x, \cdot)$ is a member of this space. It is easy to see that this space is RKHS with respect to $k$ since

$$\langle f(\cdot), k(x, \cdot) \rangle_{\mathcal{H}} = \sum_{i=1}^{N} \frac{c_i \lambda_i \psi(x)}{\lambda_i} = f(x)$$

and

$$\langle k(x, \cdot), k(x', \cdot) \rangle_{\mathcal{H}} = \sum_{i=1}^{N} \frac{\lambda_i^2 \psi_i(x)\psi_i(x')}{\lambda_i} = k(x, x').$$

So the inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ has the reproducing property that we require of a RKHS. Finally we can formulate what we introduced this space for, namely to see that the norm $\|f\|_{\mathcal{H}}^2$ takes the nice form of

$$\|f\|_{\mathcal{H}}^2 = \langle f, f \rangle_{\mathcal{H}} = \sum_{i=1}^{N} \frac{c_i^2}{\lambda_i}.$$

Since we often don't know the eigenfunctions of a kernel, it is often easier to work with the reproducing kernel map representation of functions in a RKHS [Rasmussen Williams, 2006]. That is, instead of the functions being of the form (6.4), we would rather have them to be on the form

$$f(x) = \sum_{i=1}^{N} \alpha_i k(x, x_i). \tag{6.5}$$

Since we want these functions to be in a RKHS, we need the kernel to have to reproducing property with itself, like in (6.3). Because of this, we are forced to define the inner product by

$$\begin{aligned}\langle f, g\rangle_{\mathcal{H}} &= \langle \sum_i \alpha_i k(\cdot, x_i), \sum_j \alpha'_j k(\cdot, x'_j)\rangle \\ &= \sum_{i,j} \alpha_i \alpha'_j \langle k(x_i, \cdot), k(x'_j, \cdot)\rangle \\ &= \sum_{i,j} \alpha_i \alpha'_j k(x_i, x'_j).\end{aligned}$$

In particular, we have $\|f\|^2_{\mathcal{H}} = \langle f, f\rangle_{\mathcal{H}} = \boldsymbol{\alpha}^T \mathrm{K}\boldsymbol{\alpha}$. Using this inner product, we see that the kernel has the reproducing property

$$\langle f(\cdot), k(\cdot, x)\rangle_{\mathcal{H}} = \sum_{i=1}^{N} \alpha_i k(x, x_i) = f(x).$$

With this, we finally have the groundwork theory which we can use to show the differences and similarities between GP's and the machine learning techniques regularization and kernel regression.

## 6.3 Mathematical Relationship Between GP's and Regularization

We may now further generalize the work we did in 3.5 by using $\xi(f) = \|f\|^2_{\mathcal{H}}$ as our penalty function. Similar to the standard formulation of ridge regression, where we use $\xi(f) = \|w\|_2^2$, the norm $\|f\|^2_{\mathcal{H}}$ punishes large coefficients, $c_i$, of $\psi_i(x)$. However, this penalization is less significant if the eigenvalue $\lambda_i$ of $\psi_i(x)$ is large. Because of this, large weights on eigenfunctions which has small corresponding eigenvalues is punished more severely than those with larger corresponding eigenvalues ([Hasti Tibshirani, 2009] pp. 169). This effect is very similar to the shrinkage of eigenvector with small eigenvalues which was covered in Section 3.5.

Using $\|f\|^2_{\mathcal{H}}$, we may rewrite (3.5) to

$$J[f] = Q(f, y) + \lambda \|f\|^2_{\mathcal{H}}. \tag{6.6}$$

At first, $\lambda\|f\|^2_{\mathcal{H}}$ migth seem hard to compute, since it would require us to find a eigenfunction representation of $f$. Luckily, the *representation theorem* shows that each minimizer of (6.6) has the form of a linear combination of kernels, like in (6.5) ([Rasmussen Williams, 2006] p. 132). The representation theorem has an analog to the MAP of Gaussian processes. Remember that the predictive distribution of $f(x^*) = f_*$ at a test point $x^*$ is given by the posterior distribution $p(f_*|\mathbf{y}) = \int p(f_*|\mathbf{f})p(\mathbf{f}|\mathbf{y})d\mathbf{f}$. We have that ([Rasmussen Williams, 2006] pp. 44 and 133)

$$\begin{aligned}\mathbb{E}[f_*|\mathbf{y}] &= \int E[f_*|\mathbf{f}, \mathbf{y}]p(\mathbf{f}|\mathbf{y})d\mathbf{f} \\ &= \int \mathbf{k}(x_*)^T \mathrm{K}^{-1}\mathbf{f}\, p(\mathbf{f}|\mathbf{y})d\mathbf{f} \\ &= \mathbf{k}(x^*)^T \mathrm{K}^{-1} E[\mathbf{f}|\mathbf{y}].\end{aligned}$$

Since the MAP is the expected value of the posterior distribution, we have just showed that the solution of must be on the form (6.5), with $\alpha = K^{-1}E[\mathbf{f}|\mathbf{y}]$.

To see an even closer relationship between GP's and regularization, we look at the the special case of (6.6) when $Q(f,y) = \frac{1}{2\sigma_n^2}\sum_{i=1}^n (f(x_i)-y_i)^2$ and $\lambda = 1/2$. We get

$$J[f] = \frac{1}{2}\|f\|_{\mathcal{H}} + \frac{1}{2\sigma_n^2}\sum_{i=1}^n (f(x_i)-y_i)^2.$$

Using the representation theorem, we may write $f(x) = \sum_{i=1}^N \alpha_i k(x,x_i)$ and using the reproducing property of $k$ to find that $\|f\|_{\mathcal{H}} = \boldsymbol{\alpha}^T \mathrm{K}\boldsymbol{\alpha}$. Because each $k(x,x_i)$ is known relative to our data $\mathcal{D}$, we may write $J$ as a function of $\boldsymbol{\alpha}$

$$\begin{aligned} J[\boldsymbol{\alpha}] &= \frac{1}{2}\boldsymbol{\alpha}^T \mathrm{K}\boldsymbol{\alpha} + \frac{1}{2\sigma_n^2}(\mathbf{y}-\mathrm{K}\boldsymbol{\alpha})^T(\mathbf{y}-\mathrm{K}\boldsymbol{\alpha}) \\ &= \frac{1}{2}\boldsymbol{\alpha}^T(\mathrm{K}+\frac{1}{\sigma_n^2}\mathrm{K}^2)\boldsymbol{\alpha} - \frac{1}{\sigma_n^2}\mathbf{y}^T\mathrm{K}\boldsymbol{\alpha} + \frac{1}{2\sigma_n^2}\mathbf{y}^T\mathbf{y}. \end{aligned}$$

Minimizing this w.r.t $\alpha$, we find $\bar{\boldsymbol{\alpha}} = (K+\sigma_n^2 I)^{-1}\mathbf{y}$. So we have that

$$\bar{f}(x^*) = \sum_{i=1}^n \bar{\alpha}_i k(x^*, x_i).$$

Which is the exact same as the MAP of a Gaussian process we obtain in (4.8). This should come as no surprise. As stated in Section 4.1, the posterior distribution of a GP is closely related to regularization. Minimizing (3.5) is equivalent to maximizing

$$\exp(-J[f]) = \exp(-Q(f,y)) \times \exp(-\lambda\xi(f)). \tag{6.7}$$

Working under the assumption that $f$ is of the form $f(x) = \phi(x)^T\mathbf{w}$, if we use $Q(f,y) = -\frac{1}{2\sigma_n}(\mathbf{y}-\phi(X)^T\mathbf{w})^T(\mathbf{y}-\phi(X)^T\mathbf{w})$, $\xi(f) = \mathbf{w}^T K^{-1}\mathbf{w}$ and $\lambda = \frac{1}{2}$ we see that (6.7) takes the form of the posterior distribution of a GP, as in (4.6). Therefor, by finding the MAP, we minimize the regularization penalty.

Because K is positive definite we know that it has a eigendecomposition and have only positive eigenvalues. Therefor, we may take $\mathrm{K} = \sum_{i=1}^n \lambda_i \mathbf{u}_i\mathbf{u}_i^T$, so (4.8) becomes

$$\begin{aligned} \bar{\mathbf{f}} &= \mathrm{K}(\mathrm{K}+\sigma_n^2 I)^{-1}\mathbf{y} \\ &= \sum_i \mathbf{u}_i \frac{\lambda_i}{\lambda_i+\sigma_n^2}\mathbf{u}_i^T\mathbf{y}. \end{aligned}$$

$\bar{\mathbf{f}}$ is a vector of $\bar{f}$ evaluated at each training point. Given the similarity to (3.9), we may think of GP's as shrinking the direction of which there are less variance compared to those where there is more.

Since $\bar{f}(x^*) = \mathbf{k}(x^*)^T(K+\sigma_n^2 I)^{-1}\mathbf{y}$, we may define $s(x^*) = \mathbf{k}(x^*)^T(K+\sigma_n^2 I)^{-1}$ and see that $\bar{f}$ can be written on the form of a linear smoother $\bar{f}(x^*) = s(x^*)^T\mathbf{y}$, like in (3.1). Just like in local regression, $\bar{f}$ is a linear combination of weighs found with a normalized kernel. Because of this relationship, $s_i(x)$ is often referred to as the *equivalent kernel* ([Girosi, Jones, Poggio, 1995] pp. 5).

## 6.4 Main Differences Between Regularization and GP's

As we have seen, both regularization and GP's make assumptions about the function that produces the data points $\mathbf{y}$. The GP encodes it's assumption in the prior and builds a model

according to these assumptions. Elements of the model, like the regularization function based on the covariance matrix, comes naturally from making these assumptions. Regularization's accomplices the same goal by making assumptions in a somewhat reverse order. Instead of specifying the covariance, regularization can be thought of making smoothness assumptions by specifying the inverse covariance, $(\mathrm{K} + \sigma_n^2)^{-1}$, instead ([Rasmussen Williams, 2006] pp. 136). Because of this approach, regularization doesn't have access to the covariance matrix, which is useful both for calculating the marginal likelihood and the variance of $\hat{f}$. Without these tools, regularization is forced to estimate the variance, just like local regression.

Validation functions, such as marginal likelihood, is essential for hyper-parameter tuning and model comparison ([Rasmussen Williams, 2006] pp. 136). Without access to marginal likelihood, relaxation has to make due with cross validation. What difference dose this make? Interestingly, the difference might not be so significant as one would expect. In fact, it is possible to show that exhaustive leave p-out cross-validation averaged over all values of p is formally equivalent to marginal likelihood, if you use the log posterior predictive probability [Fong Holmes, 2020]. Exhausting leave p-out cross-validation can be viewed as the most complex form that cross-validation can possibly take, since it makes every possible training and test set division. Since marginal likelihood is formally equivalent, it can be view as a method that utilizes more information in the data set compared to other cross-validation techniques, such as k-fold cross validation. However, cross validation has the advantage of been faster to compute. This hold especially true for LOOCV when $\hat{f}$ is a linear combination, which takes linear time to compute. In comparison, marginal likelihood requires access to the inverse of a matrix, which is computationally taxing to compute.

Lastly, in the case where $Q(f, y)$ is a non-convex function, $J[f]$ has multiple local solutions. In this case, the argument for the regularization solution corresponding to the Gaussian MAP is rather weak. Furthermore, since regularization does not have access to the posterior distribution, it does not handle this multimodality well, suggesting that a Bayesian approach should be used instead ([Rasmussen Williams, 2006] pp. 136).

# 7 Discussion

The beauty of math is it's ability to show that concepts which at first might see different at first are instead closely interconnected. Although the initial approach of local regression, regularization and especially GP's to the task of finding a good approximation of a natural process differ, they share the same endpoint of expressing their approximation as a linear combination of kernel functions. If one where to further explore the topic of non-parametric regression one would find that other non-parametric approaches, such as smoothing splines and support vector machines, has similar ties to kernels. Remember that kernels can be interpreted as a function that quantifies proximity between points in space. Under this perspective, we may generalize much of non-parametric regression techniques as making predictions by combine observed responses that are weighted relative to their closeness to the point at which we want to make the prediction.

Despite the advantages of GP's discussed in Section 6.4, they are not widely used in applications. Which begs the question: why? It is believed that there are tree major reasons for this ([Rasmussen Williams, 2006] pp. 197). The first and most obvious reason is their need for inverting covariance matrices, which significantly increases their computation time. While this fact made GP's almost unusable some decades ago, with the rapid improvement of computers and the introduction of inverse matrix approximation this limitation is quickly becoming less of an issue. However, the fact remains that using GP's on a very big data sets might still be

computationally infeasible. This is problematic since the performance of any regression model is closely correlated with the amount of data fitted on. For this reason, GP's are limited smaller data sets and do not have the same potential performance as for example local regression if large amounts of data are available. Because of this, the need for research on efficient inverse matrix approximation becomes increasingly important ([Rasmussen Williams, 2006] pp. 171). If such techniques are effective, GP's might have a bright future in the age of big data.

The second reason is that most of the work on GP's has historically been done using fixed covariance matrices, which limits their usefulness when faced data that has a non-uniform variance, such as in Figure 1. It is not well know that one should be able to infer the structure of the data with the choice of covariance function. If the covariance function is chosen in a arbitrary fashion in most applications, this greatly limits the potential performance of GP's ([Rasmussen Williams, 2006] pp. 197). Nevertheless, even with the this knowledge, the implementer has a greater responsibility to make sound decisions about the structure of the data, which makes fitting GP's seem like a more daunting task. Algorithmic models has the advantages of yielding good results even in the hands of the more naive implementer.

Lastly, while the use of Bayesian methods such as GP's has become quite widespread in the Algorithmic community, it sees limited use in the statistical community. This fact might be especially problematic considering that the algorithmic community tend to view their methods as "black boxes" – only suited for prediction, not inference. This is problematic, not only because it limits the usage of GP's, but also because they are nicely suited for the statistical approach of model testing. The process of discovering a covariance matrix that allows the model to fit the data well can be viewed as opportunity to perform hypothesis testing about the structure of the data ([Rasmussen Williams, 2006] pp. 197). For this reason, probabilistic models are often more suited for inference than algorithmic models. Nevertheless, since the approximation of $f$ is computed as a linear combination of responses $y$, instead of a linear combination of the covariates $x$, we can't directly relate $\hat{f}$ to the covariates. Because of this, we may not infer the contribution of each covariate to approximated function, like we can with for example linear regression. So inference in non-parametric regression is limited to deducing the general structure of the data and may not find specific correlations.

Non-parametric regression contains a multitude of different approaches with a rich theoretical background. Although these models should not for every problem, their predictive power is not to be understated and it should be the goal of any data analyst to have some of these methods in their repertoire. In particular, it is useful to know both probabilistic and algorithmic models and when to harness the strengths of each approach for a particular problem. If the subjects covered in this report sparked your interest, one should look to the future with excitement, since new developments are sure to come.

# References


[James, Witten, Hasti, Tibshirani, 2021] G. Jame, D. Witten, T. Hasti, and R. Tibshirani. (2021). An Introduction to Statistical Learning. *Springer*

[Wasserman, 2006] Larry Wasserman. (2006). All of Nonparametric Statistics. *Springer*

[Rasmussen Williams, 2006] Carl E. Rasmussen Christopher K. I. Williams. (2006). Gaussian Processes for Machine Learning. *the MIT Press*

[Hasti Tibshirani, 2009] T. Hasti, R. Tibshirani, J. Friedman. (2009). The Elements of Statistical Learning. *Springer*

[Breiman, 2001] Leo Breiman. (2001) Statistical Modeling: The Two Cultures. *Statistical Science*

[Murphi, 2012] Kevin. P. Murphy. (2012). Machine Learning: A Probabilistic Perspective. *the MIT Press*

[Girosi, Jones, Poggio, 1995] Girosi, F., Jones, M., Poggio, T. (1995). Regularization Theory and Neural Networks Architectures. *Center for Biological and Computational Learning and Artificial Intelligence Laboratory Massachusetts Institute of Technology, Cambridge*

[Fong Holmes, 2020] E Fong C C Holmes. (2020). On the marginal likelihood and cross-validation *Biometrika, Volume 107, Issue 2, June 2020, Pages 489–496*, https://doi.org/10.1093/biomet/asz077

[Seth, 2021] Anil Seth. (2021). Being You: A New Science of Consciousness. *Penguin Random House LLC*

[Hennig, 2020] Phillip Hennig. (2020). Probabilistic ML - Lecture 10 - Understanding Kernels. https://www.youtube.com/watch?v=mezshT9r_80&t=1212s

[Hennig, 2020] Phillip Hennig. (2020). Probabilistic ML - Lecture 9 - Gaussian Processes. https://www.youtube.com/watch?v=s2_L86D4kUE&t=1537s

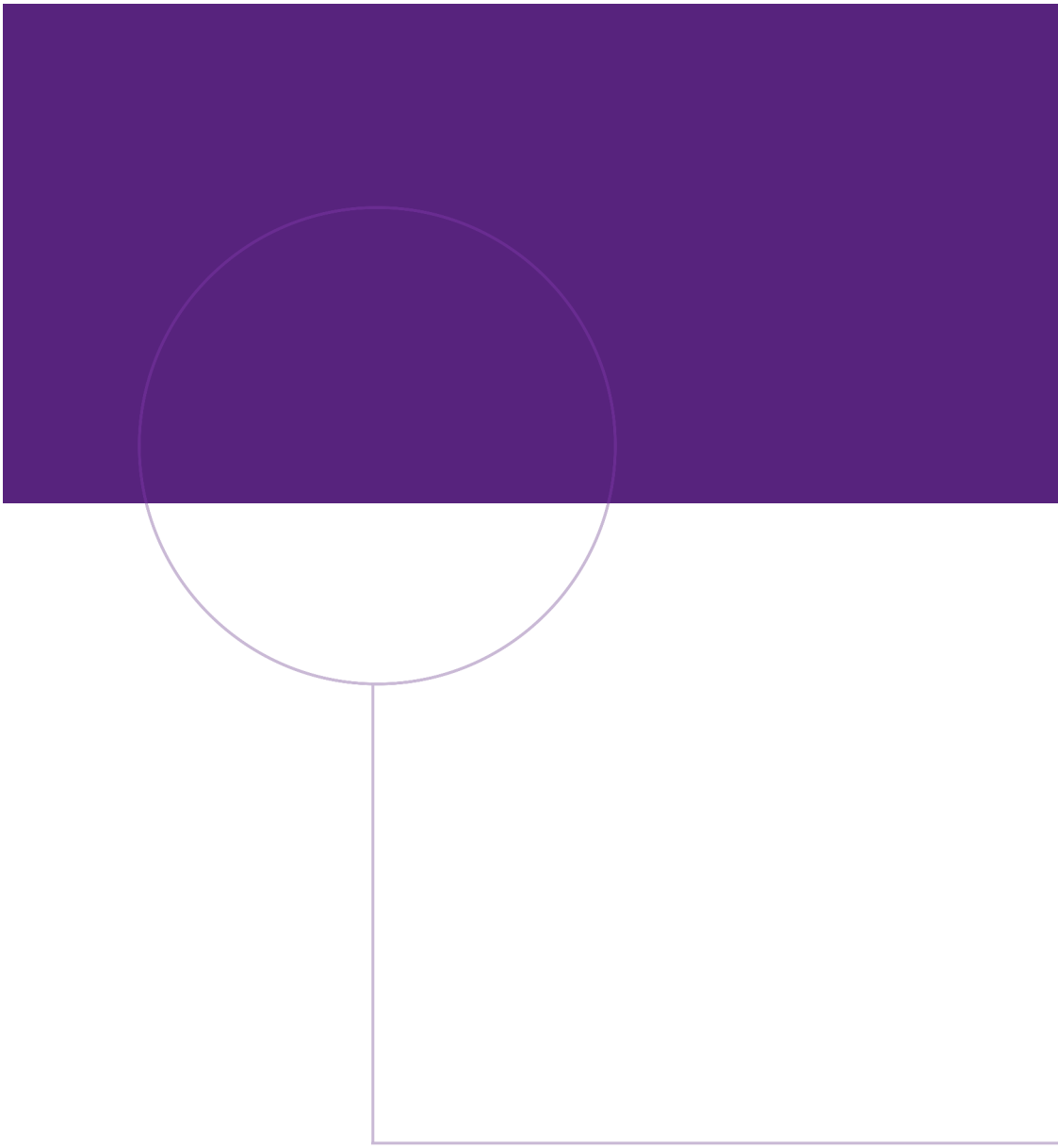
[Hennig, 2020] Phillip Hennig. (2020). Probabilistic ML - Lecture 8 - Learning Representations. https://youtube.com/watch?v=Zb0K_S5JJU4&t=1979s

[Felzenszwalb, 2017] Pedro Felzenszwalb. (2017). Machine Learning - Lecture 12. http://cs.brown.edu/people/pfelzens/engn2520/CS1420_Lecture_12.pdf

[Bartlett, 2008] Peter Bartlett. (2008). Reproducing Kernel Hilbert Spaces. https://people.eecs.berkeley.edu/~bartlett/courses/281b-sp08/7.pdf

[Patro] Rebecca Patro. (2021). Cross-Validation: K Fold vs Monte Carlo. *Towards Data Science*. https://towardsdatascience.com/cross-validation-k-fold-vs-monte-carlo-e54df2fc179b

[Smola Bartlett, 2000] Alex J. Smola Peter Barlett. (2000). Sparse Greedy Gaussian Process Regression. *Australian National University.*



NTNU

Kunnskap for en bedre verden