Bachelorarbeit

Kai Thomas Brusch

Generalized Additive Models for very large datasets with Apache Spark

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Kai Thomas Brusch

Generalized Additive Models for very large datasets with Apache Spark

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Köhler-Bußmeier
Zweitgutachter: Dipl-Math. Markus Schmaus

Eingereicht am: 26. September 2016

Kai Thomas Brusch

Thema der Arbeit
Generalized Additive Models for very large datasets with Apache Spark

Stichworte
Statistik, Statistical Learning, Regression, Generalisiertes Additive Modell, Splines, B-Splines, Apache Spark, Generalisiertes Lineare Modell, Big Data, Data Science, Machine Learning

Kurzzusammenfassung
Das Generalisierte Additive Modell erweitert das Generalisierte Lineare Modell mit der FÃ¤higkeit, eine oder mehrere unabhängige Variablen as Unbekannte glätende Funktionen darzustellen. Das Generalisierte Additive Model erfreut sich Beliebtheit in Feldern, die sehr grosse Datenmengen erfordern. Obwohl dies der Fall ist, wurde das Generalisierte Additive Modell noch in keinem modernen Cluster Computing Framework implementiert. Apache Spark ist ein modernes Cluster Computing Framework, welches eine Implementation des Generalisierten Linearen Modell beinhaltet. Diese Arbeit nutzt die, in Spark verfügbare, FunktionalitÃ¤t, um eine Teilmenge des Generalisierten Additive Modell in ein modernes Cluster Computing Framework zu implementierten. Ein Generalisiertes Additive Model kann als Generalisiertes Lineare Modell mit B-Splines realisiert werden. Dieses implementiert ein Generalisiertes Additive Modell als Generalisierte Lineare Modell mit B-Splines in der Cox-de-Boor Form als Erweiterung in Spark.

Kai Thomas Brusch

Title of the paper
Generalized Additive Models for large datasets with Apache Spark

Keywords
Statistics, Statistical learning, Regression, Generalized Additive Model, Splines, B-Splines, Apache Spark, Generalized Linear Model, Big Data, Data Science, Machine Learning

Abstract
The generalized additive model extends of the generalized linear model with the ability to describe predictor variables as the sum of unknown smoothing functions. The generalized additive model has been used in domains that require very large datasets. Although the generalized additive model is helpful in those scenarios, no public attempt has been made to implement it in a modern cluster computing environment. Apache Spark, a modern cluster

computer framework, has provided an implementation for the generalized linear model to be estimated on very large datasets. The unknown smoothing functions can be expressed as B-Splines and estimated as part of a generalized linear model. This thesis implements the generalized additive models as an extension of the generalized linear model in Apache Spark by representing the unknown smooth functions as B-Splines in the Cox-de-Boor form.

Contents

List of Figures

1 Introduction

Karl Pearson, the father of applied statistics, famously described statistics as the "grammar of science". Leveraging data as a method of inquiry has become the maxim of modern scientific thought. Hypothesis testing, statistical modeling and measurements are areas in which statistics are essential to quantitative science. Eric Schmidt, Executive Chairman of Alphabet, claims that consumer electronics and the modern Internet generates and processes more data every two days than in the prior 80.000 years. This ever growing ability to process and store data enables the use of scientific inquiry in areas previously dominated by subjective reasoning. The use of applied statistics has recently transcended from science into the commercial sphere, as statistical models are now employed to further the understanding of businesses or provide essential functionality to them. Statistical models often even constitute the core business product itself. This development has broadly been coined as 'Data Science' and 'Big Data'. The exponential growth of computing power has placed the computer at the heart of statistics. Modern statistics is leveraging parallel computing to build models that discover and explain patterns in very large data sets. The aim of this thesis is to enable a specific statistical model to harness the power of modern cluster computing.

The generalized linear model ('GLM') is the most widely used statistical model. This thesis focuses on an extension of the GLM called generalized additive model ('GAM'). The GAM offers the interpretability of the traditional GLM but allows for smooth and flexible estimation of some variables (Hastie, 1990). GLMs and GAMs and much of modern statistical research is published in the programming language R (Wood, 2006). R was written by statisticians and has become the lingua franca for statistical research, surpassing Stata as the most popular statistical programming language. Though ideal for statistics and widely used R lacks the ability to harness parallelization and cluster computing. The Mixed GAM Computation Vehicle ('mgcv') is a popular R library that uses R's limited parallelization mechanics to estimate GAMs (Wood u. a., 2015). Apache Spark ('Spark') is emerging as a prominent general purpose cluster computing frame work. Spark was designed to perform distributed, in-memory computation while preserving a high level of abstraction (Zaharia u. a., 2010). This allows Spark to handle terabytes of data and has proven to be an ideal environment to handle very large datasets (Xin,

2014). Spark offers SparkR, a high level language binding that allows R to interact with Spark's machine learning library ('Mllib'). Recently Mllib released a GLM implementation that can be accessed with R. Though Spark offers a GLM implementation no public efforts have yet been made to bring GAMs to the Spark ecosystem. This thesis introduces an implementation that established some GAM functionality in Spark by leveraging the existing GLM implementation.

The GAM can be treated as a special case of the GLM where smooth functions are represented as B-Splines (Hastie u. a., 2001). B-Splines can express an unknown smooth function as piecewise polynomial basis functions. The Cox-de-Boor form is a recursive, numerical stable and fast method to construct the polynomial basis functions (De Boor, 2001) given a value range, the basis order and knots. The formed polynomial basis functions are appended to the GLM's model matrix are estimated using the GLM's iterative reweighted least squares ('IRLS') method (Fahrmeir u. a., 2009). Estimating GAMs with Spark thus amounts to expressing particular predictor variables as a B-spline and utilizing the existing estimation methods present in Spark. By using this the approach, the spline estimation of GAMs can be performed in a modern cluster computing environment.

This thesis comprises seven sections. Following the introduction, a chapter on statistical learning sets out the conceptual framework for this thesis. The following section introduces the process of maximum likelihood estimation. Section 4 and 5 describes the generalized additive model as an extension of the linear model and the generalized linear model. Estimating spline with the linear model is the key aspect of those two sections. Sections 6 introduces Apache Spark and describes a working approach on how to leverage existing elements in Spark to implement GAMs. The thesis ends with a conclusion and an overview of future work.

2 Statistical Learning

The aim of this chapter is to introduce the framework of statistical learning. After an example of the use of statistical learning, the concepts of inference and prediction are described. The last section of this chapter will introduce the main concept of statistical learning relevant for this thesis: estimating an unknown function.

2.1 Introduction to Statistical Learning

Statistical learning is a set of tools for modeling and understanding complex data sets. Recent developments in statistics and computer science have created an interesting field that empowers modern statistical methods with computational methods. Statistical learning provides a framework for asking and answering questions by using data.

The benefits of statistical learning can be best explaining by the use of the following example James u. a. (2014). Assuming a statistical consultant is in charge of advising a client on how to improve the sales of a given product. The client provides a data set containing the sales of a given product across several markets and the associated marketing budget for three different media outlets: TV, radio and newspaper. The client is unable to impact the amount of sales directly, but he is able to change the amount spent on each media outlet. Understanding the number of items sold as a function of media spending enables the consultant to suggest a modified spending plan. This spending plan aims to increase sales by modifying media spending. In **??** the sales are plotted on the Y-axis and the budget for a media outlet on the X-axis. This model interprets the media budgets as the input variables and sales as the output variables. The input variables are commonly denoted with X_i, in this example X_1 is the TV budget, X_2 the radio budget and X_3 the newspaper budget. The name for input variables has become inconsistent. The terms predictor, independent variables and features are treated as synonyms. The output variable, in this example the sales, is also called the response or dependant variable and denoted with Y.

The applied method can be generalized to answer the following question: Given a data set with observations of a quantitative response Y and several predictors $X_1, X_2, ...X_i$, what is

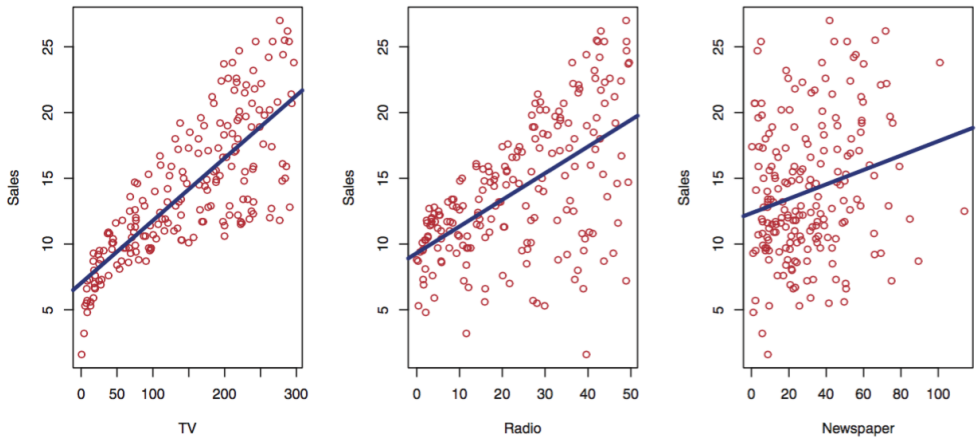

Figure 2.1: James u. a. (2014) describes the units sold as a function of spending on each of the advertisement channels. The blue line is a linear model describing the underlying relationship between units sold and media outlet budgets.

the underlying unknown relationship $f()$? $f()$ explains Y in terms of X. This can be formally stated as

$$Y = f(X) + \epsilon \tag{2.1}$$

The function $f()$ is a fixed but unknown function of the input variables $X_1, ...X_p$. ϵ is a random, identically and independently distributed, error term. Another way of framing this is to see the function $f()$ as the systematic information in the observations. All statistical learning methods are concerned with finding an optimal $f()$. This thesis will discuss three related methods of finding $f()$. Each method has a different approach to finding $f()$ and a different interpretation of the optimal function f.

2.2 Inference and Prediction

Exploring the relationship between dependent and independent variables by estimating a function that satisfies $Y = f(X) + \epsilon$ is beneficial for inference and prediction. James u. a. (2014) argues that inference is concerned with understanding a system while prediction uses the estimated function to predict Y on different values for X.

Once a function $f()$ has been estimated on a particular data set, this function can also be used to predict the value of Y for different $X_1, ..., X_p$. Predicting the value of Y new X is

simple when the error is constant. In this case the value of Y can be computed with the following equation:

$$\widehat{Y} = \widehat{f}(X) \tag{2.2}$$

The wide hat notation stands for an approximated value. $\widehat{f}$ is the estimated function for the true unknown relationship $f()$ and $\widehat{Y}$ is an estimation of Y. The accuracy of our predicted $\widehat{Y}$ depends on two quantities: the reducible error and the irreducible error. The reducible error originates from the approximating nature of $\widehat{f()}$. The difference between $\widehat{f}()$ and $f()$ directly causes the reducible error. A different approximation can influence the reducible error. However, some of the error will always remain unexplained and is inherent in the measurements taken. This implicit error in our measurements can not be reduced and is thus called the irreducible error. These two errors can formally be described as:

$$E(Y - \widehat{Y}) = E[f(X) + \epsilon - \widehat{f}(X)]^2 = [f(X) - \widehat{f}(X)] + Var(\epsilon)^{1} \tag{2.3}$$

$Var(\epsilon)$ is the irreducible error and $[f(X) - \widehat{f}(X)]$ the reducible error. Estimating f to reduce the reducible error lies at the core of every method discussed in this thesis. However, the irreducible error can not be neglected from discussion. The irreducible error is only assumed to be zero mean sum for the easiest of models.

While prediction is forward-looking, inference, the analytical perspective of statistical learning, is focused on looking backwards. Inference focuses on analyzing the estimated function $\widehat{f}$. Examining the respective contribution of each $X_1, ..., X_p$ to Y can give qualitative insight into finding the best predictors. Returning to the previous example of budget spending on media outlets: In 2.1 sales respond differently to additional spending on each media category. An analysis of each function of X shows that TV has the steepest slope. Thus, additional spending on TV advertisement leads to the largest increase in sales. This analysis illustrates the use of inference to understand the underlying mechanism. Understanding X's influence on Y is a key aspect of inference and therefore the question is: Does Y in- or decrease with a change in X and, if so, by how much? Inference also examines the fit of the function. **??** assumes a linear relationship between each X and Y. While this trend aptly explains the mechanism of the TV sales, it does not explain the Newspaper category. A linear relationship between sales and Newspaper budget spending seems unreasonable.

[1] (James u. a., 2014)

2.3 Estimating an unknown function

There are many approaches to estimating an unknown function for a given data set. James u. a. (2014) states that all approaches share certain characteristics. First, all methods require a set of training data. This training set contains tuples of dependant and independent variables. This set of tuples is used as input for an algorithm that approximates $\widehat{f}()$. Methods for approximating $\widehat{f}()$ can broadly be categorized as parametric and non-parametric.

Parametric models, such as the linear and generalized linear model, assume a specific shape of the true underlying function $f()$. Parametric models are always estimated in two steps. First, an assumption about the functional form of f is made. Then the function is estimated. The simplest assumption is the linear model, which is subject of the following section. A linear assumption enforces the following form:

$$f(X) = \beta_0 + \beta_1 X_1 + ... + \beta_p X_p \tag{2.4}$$

The function f is linear in the parameters β. The linear assumption poses tight restrictions on the shape of f but also on search space of potential functions. A non-parametric function requires searching in an infinite-dimensional function space. However, the linear assumption limits the search space to p+1 coefficients.

After deciding on a shape for f, the function needs to be estimated. The process of actually finding f is also referred to as training or fitting a function. In this example, fitting amounts to finding the coefficients $\beta_0, \beta_1, ..., \beta_p$. Therefore the full problem of fitting a linear model is:

$$Y \approx \beta_0 + \beta_1 X_1 + ... + \beta_p X_p \tag{2.5}$$

The shape decided in step one does not dictate a specific method of finding the coefficients, however there are certain methods associated with particular functional shape and error distributions. The linear model assumes a Gaussian distributed error term and can be estimated by the least squares method. Least squares is the simplest estimation method and is introduced in the next chapter. The choice of parametric form solely dictates the position and number of the parameters involved, thus the name: parametric.

Choosing a functional shape for $f()$ drastically simplifies the fitting process by predetermining the amount of coefficients to be estimated. However, the chosen shape may enforce conditions that are very different to the actual function f. A poor choice of parametric form leads to a poor estimate. This problem can be addressed with more flexible methods that allow for more parameters. A higher number of parameters, however, bears the associated risk of overfitting the data. Overfitting estimates a function too close to the original data points,

possibly omitting important systemic information. A major benefit of parametric methods is the ability to interpret the results. A functional shape is usually determined by factors outside of the data set. The choice of parametric shape allows for representation of the estimated coefficients in meaningful way. Thus the choice of shape enforces a certain model in which the coefficients are to be interpreted.

The non-parametric approach is the complement of the parametric one James u. a. (2014). Non-parametric methods, such as regression splines, do not assume the functional form of f before estimating it. This brings a major advantage over parametric methods since they allows f to follow the data as closely as possible without the influence of a predetermined shape. Allowing f to assume any arbitrary form offers much more flexibility than parametric methods. While non-parametric methods avoid the pitfall of a bad functional shape, they have to search the whole possible space of f. Searching an arbitrary functional space is very time- and space-intensive and suffers from overfitting. Non-parametric methods provide much more flexibility than parametric methods and are useful for certain predictions. Non-parametric methods excel at interpolating, predicting new cases within the training sets' range. However, their results' arbitrary shape makes the interpretation of the result harder to understand and they struggle with predictions outside the training sets' range.

The GAM constitutes an interesting collation of parametric as well as non-parametric elements: it leverages non-parametric estimation of some variables of a parametric method. It uses spline estimation to estimate arbitrary functions for some variables of a generalized linear model.

3 Maximum likelihood estimation

3.1 Introduction to maximum likelihood estimation

Maximum likelihood estimation ('MLE') provides a single framework for estimating linear models, generalized linear models and generalized additive models. Unlike probability, likelihood is used after data is available to describe a function of a parameter vector for a given outcome. Likelihood also does not always sum to one and may not even be integrable with respect to beta. Dobson und Barnett (2008) describes the principle of maximum likelihood estimation as searching for probability distribution parameters that makes the observed data most likely. This results in a search for the parameter vector that maximizes the likelihood function $L(\beta|y)$. The parameter vector is found by searching the multi-dimensional parameter space. Many statistical objects can be expressed in terms of a random variable. A random variable is defined in terms of one or several parameters. Usually the distribution parameters are the observed variance and mean. The Poisson distribution, for example, assumes mean $=$ variance and thus only requires one parameter $P(\lambda)$. The gamma distribution is defined in terms of shape k and scale θ. Maximum likelihood estimation provides a single framework to allow parameter estimation for any of the exponential family distributions Dobson und Barnett (2008).

From a statistical analysis point of view the vector y of observed data is a random sample from an unknown population. The goal of maximum likelihood estimation is to find the parameters of a given distribution that most likely produced this sample. This process is described with a probability density function ('PDF') $f()$ of observed data y given a parameter β: $f(y|\beta)$. If individual observations, y_i, are statistically independent of one another, the PDF for the data y, given the parameter vector β, can be expressed as a multiplication of PDFs for individual observations.

$$f(y|\beta) = f((y_1, y_2, ..., y_n)|(\beta_1, \beta_2, ..., \beta_n)) = \prod_{i=1}^{n} f_i(y_i|\beta_i)) \tag{3.1}$$

Given a set of parameter values, the corresponding PDF will show that some data are more probable than other data. However, the data is already given and the search is for the parameters of the distribution that most likely produced the data. Thus the conditional probability is reversed. From $f(y|\beta)$ to $L(\beta|y)$ produces the likelihood of y given the parameters β. For computational convenience, the MLE is obtained by maximizing the log-likelihood function, $ln(L(\beta|y))$. This is because the two functions, $ln(L(\beta|y))$ and $L(\beta|y)$, are monotonically related to each other. In consequence, the same MLE estimate is obtained by maximizing either one. However, the log-likelihood is preferred because it transforms the product in 3.1 to a sum. Assuming that the log-likelihood function $ln(L(\beta|y))$ is differentiable, if β exists, it must satisfy the following partial differential equation known as the likelihood equation:

$$\frac{\partial lnL(\beta|y)}{\partial \beta_i} = 0 \tag{3.2}$$

These properties are given because the definition of the maximum or minimum of a continuous differentiable function implies that its first derivatives vanish at such points. The likelihood equation represents a necessary condition for the existence of an MLE estimate. An additional condition must also be satisfied to ensure that $ln(L(\beta|y))$ is a maximum and not a minimum, since the first derivative cannot reveal this. The log-likelihood must be convex near β. This is verified by calculating the second derivatives of the log-likelihoods and checking if they are negative.

$$\frac{\partial^2 lnL(\beta|y)}{\partial \beta_i^2} < 0 \tag{3.3}$$

This form is only of theoretical value as practical models often involve many parameters and have highly non-linear PDFs. Thus the estimate must be sought numerically using non-linear optimization methods. The basic idea for these methods is to find optimal parameters that maximize the log-likelihood focusing on smaller sub-sets of the multi-dimensional parameter space. This is the preferred approach because exhaustively searching the whole parameter space becomes intractable with an increasing amount of parameters. The practical method for MLE searches by trial and error over the course of a series of iterative steps. Each iteration changes the parameters vector from the previous iteration by a small value. The choice of the value is tailored to improve the log-likelihood. Though the choice of parameters can be challenged by other means of estimation, it be can shown that distribution parameter estimation via MLE will result in the best unbiased estimator J. A. Nelder (1972).

4 Linear Models

This chapter introduces the linear model and the generalized linear model. The linear model explains the matrix vector form and estimation via the least square method. The generalized linear model establishes much of the theory behind the generalized additive model and introduces MLE via the iterative reweighted least squares model. The proposed GAM implementation will leverage the GLM's iterative reweighted least squares estimation to realized spline approximation of predictor variables.

4.1 Linear model

The linear model is the elementary form of statistical models. It enforces a strict parametric form on the estimated function and consequently belongs to the parametric methods. Even though linear models have limited real world applications, they introduce many relevant ideas and concepts for the following methods. The generalized linear model is the generalized version of the linear model and eases some restrictions set by the linear model. Both models are essential for understanding the GAM, which is the main subject of this thesis. The GAM is an extension of the generalized linear model. In the following, the theory behind linear and generalized linear models is explained before these theories are used in the estimation process.

Simple linear model The simple linear model (or linear regression) explains the response variable as the linear combination of a predictor variable and the estimated coefficients. It is the simplest linear model and explains a normal distributed response variable in terms of one explanatory variable. Given n observations of x_i and y_i where y_i is the observations of a random variable Y_i with expectation $\mu_i \equiv E(Y_i)$, a linear model has the following parametric form [1]:

$$Y_i = x_i\beta + \epsilon_i \; where \; \mu_i = x_i\beta \tag{4.1}$$

The key assumption for the linear model is that ϵ_i is a Gaussian distributed error term and independent variance of x and y. A linear model assumes that ϵ_i are mutually independent

[1] (Wood, 2006)

zero mean random variables with a constant variance σ^2. The assumption of constant variance for all x_i is also called homoscedasticity. The linear model explains the response variable Y in terms of a predictor variable x multiplied by an estimated coefficient β plus a normal distributed random error term ϵ. y_i and x_i are known but β is unknown and thus needs to be estimated.

4.1.1 Maximum likelihood and least squares

The simple linear model described in 4.1 can be estimated with several similar techniques. This section describes the estimation process via maximum likelihood. MLE for simple linear models assumes five conditions. (1) The x values can be described in terms of an arbitrary random variable X. (2) If $X = x$, then $Y = \beta_0 + \beta_1 x + \epsilon$ for some coefficients β_0 and β_1 and some random noise variable ϵ. (3) The error term ϵ is approximately $N(0, \sigma^2)$ and independent of X with a constant variance. (4) ϵ is independent across observations. (5) The response variable Y is independent across observations and conditional on X. The simple linear model can be estimated without the third and fourth assumption but they are required for estimation with MLE [2]. Because of the Gaussian error term this approach is also called the Gaussian-noise simple linear regression model. The Gaussian-noise simple linear regression model allows for stating the conditional PDF of Y for each x as:

$$p(y|X = x; \beta_0, \beta_1, \sigma^2) \tag{4.2}$$

This notation explicitly removes the parameters from the random variables. Treating the random variable parameters individually highlights the estimation through distribution parameter estimation. Given the Normal distribution PDF and dataset of observations in tuple form (x_n, y_n), a general form of the MLE PDF can expressed:

$$\prod_{i=1}^{n} p(y|X = x_i; \beta_0, \beta_1, \sigma^2) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{(y_i - (\beta_0 + \beta_1 x_1))^2}{2\sigma^2}} \tag{4.3}$$

The independence assumption of Y_i allows to write the PDF as a product of conditional probabilities. However, the relevant question goes in the opposite direction: What are the parameters β_0, β_1 and σ^2 that are likeliest to produce the probability density function that generated the observed data? Thus the PDF has be to rearranged to express the desired

[2] (Myung, 2003)

parameters. Given a choice of a parameter vector (b_0, b_1, s^2) the likelihood as a function of the those parameters can be given as:

$$\prod_{i=1}^{n} p(y|X = x_i; b_0, b_1, s^2) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{(y_i-(b_0+b_1x_1))^2}{2s^2}} \tag{4.4}$$

As described in the MLE section, the likelihood l is best examined in terms of the log-likelihood L. Leveraging the log-likelihood, the likelihood can be expressed in terms of a sum:

$$L(b_0, b_1, s^2) = log \prod_{i=1}^{n} p(y_i|X = x_i; b_0, b_1, s^2) \tag{4.5}$$

$$= \sum_{i=1}^{n} log\ p(y_i|X = x_i; b_0, b_1, s^2) \tag{4.6}$$

$$-\frac{n}{2}\ log\ 2\pi - n\ log\ s - \frac{1}{2s^2} \sum_{i=1}^{n} (y_i - (b_0 + b_1 x_i))) \tag{4.7}$$

With this formulation the log-likelihood for a given data set and a choice of distribution parameters can be computed. A key feature of the log-likelihood is that the distribution parameters can be approximated in the following form Cramer (1974):

$$\beta_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} = \frac{c_{XY}}{s_X^2} \tag{4.8}$$

$$\beta_0 = \bar{y} - \beta_1 \bar{x} \tag{4.9}$$

$$\sigma^2 = \frac{1}{n} \sum_{n}^{1} (y_i - (\widehat{\beta_0} + \widehat{\beta_1} x_i))^2 \tag{4.10}$$

A special property of the Gaussian-noise model is that the MLE coefficients in log-space are equal to the least squares solution [3]. The estimated coefficients β_1 and β_2 are also the slope and intercept of a linear function that passes through the data tuples (x_i, y_i). The variance σ^2 is also the average error of the linear function that passes through the data. This connection only works under the assumption of a Gaussian error term and a constant variance. The constant variance has been reflected in **??** by giving equal weight for each data point. The least squares approach gives an alternative approach to estimating β from the given data x_i,y_i. In

[3] (Myung, 2003)

this approach the model seeks to find the β that minimizes the squared difference between y_i and x_i. Formalizing this notion leads to the definition of the least-squares measure S:

$$\mathbf{S} = \sum_{i=1}^{n}(y_\mathrm{i} - (\beta_1 + x_\mathrm{i}\beta_1))^2 = \sum_{i=1}^{n}(y_\mathrm{i} - \mu_i)^2 \tag{4.11}$$

Per this definition, a good choice of β minimizes the difference between y_i and μ_i. As S converges to 0, the better of an estimate β becomes. The problem of least square estimation becomes minimizing S with respect to β_0 and β_1. To minimize S, differentiate with respect to β_0 and β_1. For algebraical reasons $-S$ is maximized.

$$\frac{\partial S}{\partial \beta_0} = 0 \tag{4.12}$$

$$\frac{\partial S}{\partial \beta_1} = 0 \tag{4.13}$$

These two equations are called the normal equations. To find the parameters both have to be maximized. In the normal equations two point estimators b_0 and b_1 estimate β_0 and β_1 respectively.

$$\sum_{n}^{1} y_i = nb_0 + b_1 \sum_{n}^{1} x_i \tag{4.14}$$

$$\sum_{n}^{1} x_i y_i = b_0 \sum_{n}^{1} x_i + b_i \sum_{n}^{1} x_i^2 \tag{4.15}$$

The normal equations in this case are two equations with two unknowns. This system of equations can be solved Cramer (1974).

$$\beta_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} = \frac{c_{XY}}{s_X^2} \tag{4.16}$$

$$\beta_0 = \bar{y} - \beta_1 \bar{x} \tag{4.17}$$

This shows that under the given conditions the point estimators b_0 and b_1 are equivalent to the MLE estimates b_0 and b_1. Thus the estimation under Gaussian-noise with MLE is the same as estimating the linear model with least squares. This link provides a strong framework for

numerically stable and efficient fitting of the general linear model in terms of matrices and vectors [4].

4.1.2 Linear Model

Wood (2006) introduces the linear model as the generalization of the simple linear model, allowing for the response variable to be explained with multiple predictor variables. Though the linear model generalizes the simple linear model, it still assumes a normal distributed Y and equal variance.

4.1.3 Linear Model in the vector-matrix form

Several predictor variables require rewriting the simple linear model in terms of vectors and matrices. A major benefit of using the matrix-vector forms is that the problem of finding β becomes the problem of solving an overdetermined system of equations. Again, given n observations of y_i and x_i plus some additive constant. Explicitly writing each $\mu_i = x_i\beta_i$ illustrates the shape of the system of equations.

$$\begin{aligned} \mu_1 &= \beta_0 + x_1\beta_1 \\ \mu_2 &= \beta_0 + x_2\beta_1 \\ &\dots \\ \mu_n &= \beta_0 + x_n\beta_1 \end{aligned} \tag{4.18}$$

For the linear model this system of equations can be rewritten in a matrix-vector form. This form allows to phrase the estimation process as finding a solution to an over-determined system of equations. Finding this solution is mathematically well established and highly optimized. The matrix-vector form for the simple linear model takes the following shape:

$$\begin{bmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \\ \dots \\ \mu_n \end{bmatrix} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ 1 & x_3 \\ \dots & \\ 1 & x_n \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} \tag{4.19}$$

[4] (Myung, 2003)

Adding predictor variables and generalizing the simple linear model amounts to appending a predictor variable vector to the matrix and introducing a new coefficient to the coefficient vector:

$$\begin{bmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \\ \dots \\ \mu_n \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_2 \\ 1 & x_2 & x_2 \\ 1 & x_3 & x_3 \\ \dots & & \\ 1 & x_n & x_n \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{bmatrix} \tag{4.20}$$

The matrix-vector form yields the general form of the linear model $\mu = X\beta$. This means that the value vector μ is given by the model matrix X multiplied by the unknown parameter vector β. The model matrix can account for an arbitrary amount of predictor variables by adding a column for each predictor. This formulation assumes numerical predictor variables. Numerical predictor variables can be either continuous or discrete. Continuous predictor variables are usually measurements without clear boundaries between values. A classic example is the measurement of length. Since there are many arbitrary values between 1 meter and 2 meters, i.e. 1.1, 1.01, 1.001, they are called continuous. Discrete values have a clean border between two values. The count of occurrences is an example of a discrete variable. Categorical variables require special encoding in the model matrix. Dummy and factor encoding are the two approaches for factor variables. For a detailed explanation on factor encoding in linear models see Fahrmeir u. a. (2009). The model matrix is also known as design matrix and the terms are used interchangeably. $\mu = X\beta$ is the canonical form of a linear model. Algebraically the method for estimating β for one predictor and estimating the unknown vector β are very similar. However, a stable and efficient numerical solution for the matrix-vector form requires adjustments.

Estimating Linear Models with least squares

Wood (2006) describes the estimation process for unknown parameter vectors as finding the least squares solution to β. The computationally stablest and fastest method involves QR-decomposing the model matrix to express $\widehat{\beta}$ in terms of the upper triangular matrix. This method is widely used in statistical packages. This approach starts with the linear model in the full matrix-vector form:

$$\mu = X\beta, y \approx N(\mu, I_n, \sigma^2) \tag{4.21}$$

The model matrix X is a matrix with n rows and p columns. With n being the number of observations and p being a product of predictors and their encoding. Estimating β requires minimizing the Euclidean length of a vector. The Euclidean length of a vector is the squared sum of its elements. For a vector v in n dimensional space the Euclidean length is defined as:

$$\|v\|^2 \equiv v^T v \equiv \sum_{n}^{i=1} v^2 \tag{4.22}$$

An essential fact about the Euclidean length is that the rotation of a matrix' coordinates does not change its length. This property can be used to estimate β from an orthogonal matrix. This also applies to the rotation of $y - X\beta$. Leveraging the Euclidean length to define measure S in the matrix-vector framework yields:

$$S = \|y - \mu\|^2 = \|y - X\beta\|^2 \tag{4.23}$$

The QR-decomposition of a matrix is a computational stable approach for estimating β Wood (2006). Any real matrix X can be decomposed into an orthogonal matrix and a triangular matrix.

$$X = Q \begin{bmatrix} R \\ 0 \end{bmatrix} = Q_f R \tag{4.24}$$

R is the upper triangular matrix with p rows and p columns and the matrix R has the rank of the original matrix. Q is an orthogonal matrix with n rows and n columns of which the first p columns form Q_f. By definition multiplying a vector with an orthogonal matrix does not change its length. The QR-decomposition divides a matrix into two district matrices that have different properties but maintain their lengths. Estimating coefficients of a over-determined system of equations can be stated in terms of a QR-decomposed matrix X. This amounts to applying the QR decomposition of the model matrix to 4.23 and can be stated as:

$$\|y - X\beta\|^2 = \left\| y - Q \begin{bmatrix} R \\ 0 \end{bmatrix} \beta \right\|^2 = \|Q^T y - \begin{bmatrix} R \\ 0 \end{bmatrix} \beta\|^2 \tag{4.25}$$

Only the orthogonal matrix of the QR-decomposed model matrix is multiplied with the response variable vector and the original model matrix. Multiplying Q^T with the response

vector can be stated as $Q^T y = \begin{bmatrix} f \\ r \end{bmatrix}$. Where f is a vector of p dimensions and hence r is a vector of $n - p$ dimensions.

$$\|y - X\beta\|^2 = \left\| \begin{bmatrix} f \\ r \end{bmatrix} - \begin{bmatrix} R \\ 0 \end{bmatrix} \beta \right\|^2 = \|f - R\beta\|^2 + \|r\|^2 \tag{4.26}$$

This form exposes the residual error r as independent of β. $\|f - R\beta\|^2$ can be reduced to zero by choosing β so that $R\beta$ equals f. The estimator $\widehat{\beta}$ can be stated as:

$$\widehat{\beta} = R^{-1} f \tag{4.27}$$

The reducible error, also called residual error, is the difference between the model matrix multiplied by the estimated β minus the value for y. $\|r\|^2 = \|y - X\widehat{\beta}\|^2$. This section shows that the β vector can be expressed in terms of the decomposed model matrix. All practical applications for linear models rely on the QR-decomposition.

4.2 Generalized Linear Model

The generalized linear model ("GLM") is an extension of the general linear model. GLMs allow the response variable to be of any exponential family distribution and the error term to be heteroscedastic (Dobson und Barnett, 2008). The previously introduced general linear model assumes a normally distributed response variables with a constant variance. The GLM is less restrictive on the response variable by allowing it to be distributed according to any of the exponential family distributions and by allowing the error to be non-constant. Exponential family distributions contain many practical distributions including Poisson, binomial and gamma. The binomial distribution is commonly used to model a binary outcome and the Poisson distribution is used to model count data. The GLM models the link function $g()$ of the expected value μ_i as the linear combination of the model matrix X and the estimated coefficients β and can be formally described as:

$$g(\mu_i) = X_i \beta_i \tag{4.28}$$

X_i is the i^th row of a model matrix X and β is a vector of unknown parameters. The GLM models the expected value of the random variable Y with $\mu_i \equiv E(Y_i)$. Y_i is now assumed to be distributed according to some exponential family. Every exponential family distribution has a link function which transforms the expected value vector μ_i into the space of the estimated

coefficients. The estimation is performed on a transformed scale but the expected variance is calculated on the original scale of the predictor variables. While the least squares approach was sufficient for estimating linear models with a normally distributed response and a zero mean error term, it fails to estimate other exponential distributions. The framework of least squares estimation thus needs to be extended to account for any exponential family distribution and a non-constant variance. Iterative reweighted least squares ('IRLS') provides a method for maximum likelihood estimation for any exponential family distribution and accounts for heteroscedasticity (Dobson und Barnett, 2008). The following sections explain the link function, how to generalize the MLE method from Normal to any exponential distribution and how to account for non-constant variance by weighting each observation. IRLS provides a single framework for estimation of all exponential family distributions and can be used to estimate a special case of the GAM.

4.2.1 Likelihood for any exponential family distribution

Generalized linear model estimation is based on MLE. Unlike the MLE for linear models the estimation process for GLM must account for two aspects: (1) Y can be of any exponential family distribution and (2) heteroscedastic error terms. [5] Formally, the GLM models an n-vector of independent response variables Y where $g()$ is the link function and μ is the expected value of Y:

$$g(\mu_i) = X_i\beta \tag{4.29}$$

With $\mu \equiv E(Y)$ and the link transformation of Y_i as $Y_i \approx f_{\theta i}(y_i)$ where $f_{\theta i}$ stands for the canonical link transformation of an exponential family distribution. The link function $g()$ connects to the expected value of y to the linear predictor of predictor variables. The influence of β is linear but the PDF can be non-linear. The link function connects the two by transforming the response variable in the space of the estimated coefficients. The canonical link function g_c is a special case of the link function where $g_c = X_i\beta$. For example, the canonical link function for the Poisson distribution is the log function. The canonical link function ensures that μ stays within the range of the response variable. To estimate any exponential family distribution, the likelihood criterion must be stated in a general form. The PDF of any exponential family distribution can be stated as:

$$f_\theta = exp[\{y\theta - b(\theta)\}/a(\phi) + c(y, \phi)] \tag{4.30}$$

[5] The process of extending MLE to account for any exponential family distribution is described in Wood (2006)

a, b and c are arbitrary functions, ϕ is an arbitrary scale parameter and θ is the canonical link parameters. The interesting property of this form is that the general expression for mean and variance of exponential family distributions can be expressed in terms of a, b and ϕ. The mean and the variance are the required parameters to fit any exponential family distribution and are computed via deriving the log-likelihood. Since the natural logarithm is the inverse of the exponential function, the log-likelihood can be written as:

$$l(\theta) = [y\theta - b(\theta)]/a(\phi) + c(y, \phi) \tag{4.31}$$

$E(Y)$ can now compute by differentiating l with respect to θ and by treating l as a random variable. This leads to a replacement of the particular value y with the random variable Y.

$$\frac{\partial l}{\partial \theta} = [y - b'(\theta))]/a(\phi)) \tag{4.32}$$

$$E(\frac{\partial l}{\partial \theta}) = [E(Y) - b'(\theta))]/a(\phi)) \tag{4.33}$$

Given that $E(\partial l/\partial \theta) = 0$, the expected value of the random variable Y can be stated as:

$$E(Y) = b'(\theta) \tag{4.34}$$

This results in a method to compute the mean of any exponential family random variable with the first derivative of b with respect to θ. This the direct link between the β and the model parameter. Allowing the parameter β to determine by the mean of the response variable and hence the canonical parameter, is an established method for finding these values glm. The variance of that random variable can be computed from the second derivative of the log-likelihood:

$$\frac{\partial^2 l}{\partial^2 \theta} = -b''(\theta)/a(\phi) \tag{4.35}$$

This formula can be re-arranged to express the variance of Y [6].

$$var(Y) = b''(\theta)a(\phi) \tag{4.36}$$

To account for the mentioned heteroscedasticity, a weight w is introduced to weigh each observation proportional to the reciprocal of the error variance for that observation. Thus, data points with low variance are given higher weights and points with higher variance are

[6] Wood (2006)

given lower weights. With this approach, the generalized linear model can be estimated on data with a non-constant error term Dobson und Barnett (2008).

$$var(Y) = b''(\theta)a(\phi)/w \tag{4.37}$$

For reasons of simplicity the function $V(\mu)$ is defined as $V(\mu) = b''(\theta)$ such that $var(Y) = V\mu)$ Wood (2006)

Given vector y of an observation random variable Y, the maximum likelihood estimation of β is possible due to the independence of all Y_i. The formal notation for the MLE of the likelihood of β, $L(\beta)$ becomes

$$L(\beta) \prod_{i=1}^{n} f_{\theta i}(y_i) \tag{4.38}$$

Given the previously stated probability mass function for exponential family members, the log likelihood $l()$ of β can be written as:

$$l(\beta) = \sum_{i=1}^{n} log[f_{\theta i}(y_i))] = \sum_{i=1}^{n} (y_i\theta_i - b_i(\theta_i))/a_i(\phi) + c_i(y_i, \phi) \tag{4.39}$$

The log function transforms the product to a sum. ϕ is assumed to be constant for all i. The only relevant choices for ϕ are ones that can be stated as $a_i(\phi) = \phi/w_i$ with w_i being a constant. The weights are assumed to be known. This assumption allows us the rewrite the previous formula as:

$$l(\beta) = \sum_{i=1}^{n} w_i(y_i\theta_i - b_i(\theta_i)/a_i(\phi) + c_i(y_i, \phi) \tag{4.40}$$

The process of finding the parameter vector β amounts to maximizing the log-likelihood by partially differentiating l with respect to each element of β, setting the resulting expressions to zero and solving for β.

$$\frac{\partial l}{\partial \beta_j} = \sum_{i=1}^{n} w_i \left(y_i \frac{\partial \theta_i}{\partial \beta_j} - b'(\theta_i) \frac{\partial \theta_i}{\partial \beta_j} \right) \tag{4.41}$$

Wood (2006) details that by applying the chain rule, $E(Y) = b'(\theta)$ and term substitution the following form can be produced:

$$\frac{\partial l}{\partial \beta_j} = \frac{1}{\phi} \sum_{i=1}^{n} \frac{[y_i - b_i{}'(\theta_i)]}{b_i''(\theta_i)/w_i} \frac{\partial \mu_i}{\partial \beta_j} \tag{4.42}$$

By applying the framework developed above to determine variance and expected value the full form for the estimation of β can be stated as:

$$\sum_{i=1}^{n} \frac{(y_i - \mu_i)^2}{V(\mu_i)} \frac{\partial \mu_i}{\partial \beta_j} = 0 \; \forall \; j. \tag{4.43}$$

This matches exactly the same equation that would have to be solved to find β by non-linear weighted least squares if the weights for $(V(\mu_i)$ were known in advance and were independent of β. In that scenario the least squares objective would be

$$S = \sum_{i=1}^{n} \frac{(y_i - \mu_i)^2}{V(\mu_i)} \tag{4.44}$$

In the formulation, μ_i depends non-linearly on β but the weights $V(\mu)$ are treated as fixed. To find the least squares estimated, $\partial S / \partial \beta_j$ must equal to zero for all js. Formulating the search for β as finding the optimal choice of distribution parameters of Y invites an iterative approach that chooses some parameter values and successively adjusts these.

4.2.2 Iterative Reweighted Least Squares

The iterative reweighted least squares ("IRLS") is a method for estimating the maximum likelihood estimates for the GLM. IRLS realizes this by minimizing the distance between the weighted current estimated response and the actual response variable Dobson und Barnett (2008). The process can be state as: Let $\widehat{\beta}^{[k]}$ be the estimated parameter vector at the k^{th} iteration. $\eta^{[k]}$ is a vector with the elements $\eta^{[k]} = X_i \widehat{\beta}^{[k]}$ and $\mu_i^{[k]}$ is defined as the inverse of the link function. $\mu_i^{[k]} = g^{-1}(\eta_i^{[k]})$. Given these definitions, the IRLS algorithm can be stated as: [7]:

1. Compute the weighted variance $V(\mu_i^{[k]})$ terms implied by the current estimate for $\widehat{\beta}^{[k]}$
2. Use these estimates and apply the method described to minimize the least squares objective with respect to β to obtain $\widehat{\beta}^{[k]}$
3. Set k to k+1

To come to a computational formulation, the least square objective for IRLS can be written as:

$$S = \left\| \sqrt{W}^{[k]} \left(z^{[k]} - X\beta \right) \right\|^2 \tag{4.45}$$

[7] The IRLS definitions are taken from Wood (2006)

Where $z^{[k]}$ is the so-called "pseudo data" and $W^{[k]]}$ is a diagonal matrix with a weight for each observation, each defined as:

$$z_i^{[k]} = g'(\mu^{[k]})(y_i - \mu_i^{[k]} + \eta_i^{[k]} \tag{4.46}$$

$$W_{ii}^{[k]} = \frac{1}{V(\mu_i^{[k]})\mu_i^{[k]}} \tag{4.47}$$

With these definitions we can finally write the full form of the practically used IRLS.

1. Use current $\eta^{[k]}$ and $\mu^{[k]}$ to calculate pseudo data $z^{[k]}$ and iterative weights for the weights matrix $W^{[k]}$
2. Minimize the least squares objective $\left\| \sqrt{W}^{[k]} \left(z^{[k]} - X\beta \right) \right\|^2$ w.r.t. β to obtain $\widehat{\beta}^{[k+1]}$ and the resulting $\eta^{[k+1]} = X_i \widehat{\beta}^{[k+1]}$ and $\mu_i^{[k+1]}$
3. Set k to k + 1

The proposed method is an iterative method that produces a vector of pseudo data z with the current parameter estimated $\widehat{\beta}$ multiplied with the model matrix X and the expected value of Y_i. The distance between the pseudo data vector and the model matrix multiplied by the weight is then minimized to produce a new set of estimates. Dobson und Barnett (2008) have shown that the introduced method will converge on the optimal parameter vector $\widehat{\beta}$.

5 Generalized Additive Models

5.1 Generalized Additive Models

The generalized additive model ("GAM") extends the GLM with the ability to estimate variables with smoothing splines. This addition provides flexible estimation beyond the linear parametric form of the linear models. GAM has the interpretability advantages of GLMs where the contribution of each predictor variable is encoded. However, it has substantially more flexibility because the relationship between predictor and response variable is not assumed to be linear. The GAM assumes the relationship between predictor and response variable to be the sum of arbitrary functions. These functions are regularized by penalizing the second derivative. Because the acceleration of the functions are penalized, they are commonly referred to as smooth functions. Hastie (1990) introduced GAMs and described many approaches to estimating the smooth functions. Modern approaches have highlighted the use of regression splines for smoothing functions estimation Wood (2006). Woods states that the smooth function can best be represented as regression splines, which is what this thesis focuses on. This thesis focuses on representing the unknown smooth function via regression splines. The benefits of regression splines will be illustrated by progressively building a function in R that estimates a GAM. To see the similarities to the GLM, formally describe the GAM as [1]:

$$g(\mu_i) = X_i^*\theta + f_1(x_1 i) + f_2(x_2 i) + ... + f_3(x_3 i) \tag{5.1}$$

The GAM inherits the link function $g()$, parameter vector θ and the model matrix X_i^* from the GLM. The GAM follows the GLM's definition of $\mu \equiv E(Y_i)$ and Y is distributed according to some exponential family distribution. X_i^* is the i^th row of the model matrix. The GAM introduces the smooth functions f_j over the predictor variables x_k. Specifying a model in terms of a non-parametric smooth function allows x_k to have an arbitrary pattern. Allowing $f_j(x_k)$ to follow any shape can give insight into response variable behavior that the parametric form of GLMs fails to capture. The gained flexibility comes at the raises two new questions: First, how to find these arbitrary functions and second, how to smooth them.

[1] The notation and code examples for GAMs follows Wood (2006)

5.2 Regression Splines

Regression splines estimate any function by dividing the original function in non-overlaping sections and fitting an individual function for each. Each section spans between so-called "knots". Broadly speaking, a regression spline is a function that explains x_i in terms of y_i with some error ϵ by fitting smaller functions and joining them together. The error term is a random variable that is independent and identically distributed with $N(0, \sigma^2)$. This is identical to the definition of non-parametric methods from the first section of this thesis. Formally we are searching for a function $f()$ that satisfies

$$y_i = f(x_i) + \epsilon_i \tag{5.2}$$

By leveraging the methods used for fitting a simple linear model, we assume the function f to be linear in x_i. This assumption guarantees that the function can be found with linear parametric methods. Basis functions are a reasonable approach for estimating the function f Wood (2006). The basis function allows to represent the function f as a combination of a basis function and a parameter vector β. The function f can thus be written as:

$$f(x) = \sum_{i=1}^{q} b_i(x)\beta_i \tag{5.3}$$

The function f is represented as the sum of basis functions multiplied by the parameter vector. Basis splines can be represented as a sum of the basis dimension q. The basis dimension q is a design choice and tailored to model the underlying behavior. The influence of β by linear combination satisfies the linearity condition. James u. a. (2014) states that the basis over the entire range of the data amounts to a polynomial regression. Polynomial regression can be useful but suffers from instability at the edges and is insufficient for interpolation. The suitable alternative is splines estimation. Splines divide the the unknown function into sections. Each section is then fit with an individual polynomial. Each piecewise polynomial is required to be continuous at the intersection with the adjacent piecewise polynomial. Basis splines allow to find the interpolating spline through linear estimation. This approach gives enough flexibility while satisfying the linearity constraint.

5.2.1 Basis Splines and the Cox-de-Boor Form

Basis splines ("B-splines") constitute an appropriate method for the nonparametric estimation of an unknown function with the already introduced estimation methods. A B-Spline of order k is a piecewise polynomial of degree k-1, which is continuous up to the kâ2 derivative at

the transition between knots. It is possible to uniquely represent any spline function as a linear combination of B-Splines of that same degree and knots (De Boor, 2001). Expressing an unknown function of a degree k can thus be described as constructing the B-Splines of order k+1 and solving for the weights. Since the spline must be a linear combination of its B-Splines, the weights can be estimated with the introduced least squares method. With this approach a spline of a predictor variable can be fitted as part of linear or generalized linear model.

Formally, a B-Spline is defined over a range of knots which is a non-descending vector of values.

$$t_0 \leq t_1 \leq ..t_N \leq t_N + 1 \tag{5.4}$$

Every B-Spline basis of order k can be expressed recursively as a B-Spline basis of order k-1. The Cox-de-Boor provides recursive expression of a B-Spline B given an order k, a set of x values and knots t_i.

$$B_{i,1}(x) := \begin{cases} 1 \text{ if } t_i \leq x \leq t_i + 1 \\ 0 \text{ otherwise} \end{cases} \tag{5.5}$$

$$B_{i,k}(x) := \frac{x - t_i}{t_{i+k-1} - t_i} B_{i,k-1}(x) + \frac{t_{i+k} - x}{t_{i+k} - t_{i+1}} B_{i+1,k-1}(x) \tag{5.6}$$

This form defines the basis function between k adjacent knots for every knot. The following examples illustrate how the Cox-de-Boor form generates a basis for orders 1 though 3. Starting with 5.5: A basis function of order 1 is a step function that is 1 in the interval $[t, t_{i+1}]$ and 0 elsewhere. The basis is only defined between the current knot t and its neighbor t_{i+1}. Since this is a basis or order 1, the resulting polynomial is of degree 0 and constant.

Figure 5.1: B-Spline basis function $B_{i,1}(t)$ of order 1 Zhang (2006)

A B-Spline basis function of order 2 is a piecewise linear function over the interval $[t, t_{i+2}]$ and 0 elsewhere. The full recurrence for a B-Spline of order 2 is given as:

$$B_{i,2}(x) := \frac{x - t_i}{t_{i+k-1} - t_i} B_{i,1}(x) + \frac{t_{i+k} - x}{t_{i+k} - t_{i+1}} B_{i+1,1}(x) \tag{5.7}$$

A B-Spline of degree 2 produces a piecewise polynomial of degree 1 between the interval $[t, t_{i+2}]$ and 0 elsewhere. This B-Spline is defined between 3 adjacent knots.

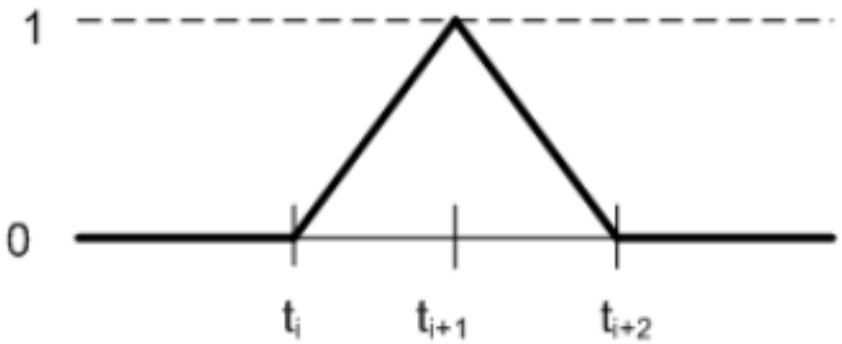

Figure 5.2: B-Spline basis function $B_{i,2}(t)$ of order 2 Zhang (2006)

A B-Spline of order 3 is piecewise quadratic function with a smooth transition at t_i, t_{i+3}. The basis are 0 outside $[t_i, t_{i+3}]$ and are defined between 4 adjacent knots. Following the recurrence of 5.6, the B-Spline of degree 3 can be given as:

$$B_{i,3}(x) := \frac{x - t_i}{t_{i+k-1} - t_i} B_{i,2}(x) + \frac{t_{i+k} - x}{t_{i+k} - t_{i+1}} B_{i+1,2}(x) \tag{5.8}$$

A main concern for curve fitting with B-Splines is the appropriate choice of knots. The location and number of knots is a matter of design and has a major influence on the resulting spline. Equidistant knots are called uniform. Uniform knots are the simplest to fit but do not provide the desired flexibility. Equidistant knots suffer from uneven distributed data points and become decreasingly effective with less data. Quantile knots are a reasonable approach to space the same number of observations between each knot. Placing the same amount of observations between each knot evens out the problem of uniform knots. The choice of knots may also reflect curvature of the underlying system. Given a time series of data with observations of a variable at a regular interval, a choice of knots can be used to model non-linear trends.

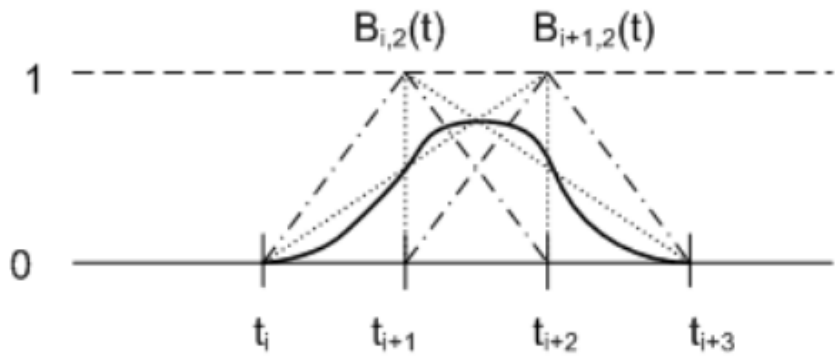

Figure 5.3: B-Spline basis function $B_{i,3}(t)$ of order 3 Zhang (2006)

This chart illustrates that for time series data the choice of uniform and equidistant knots might fail to capture the underlying system. The highlighted areas show that there is stronger curvature between some observations. A model to describe the underlying system can place a focus on certain time intervals by placing more knots in the highlighted areas and less in the other areas. The ability to place more knots in certain locations allows for very flexible modeling. Dictating the shape of the estimated spline by positioning is a powerful design tool and a major strength of B-Splines. The following section describes how a spline can be estimated with the B-Spline representation.

5.2.2 Fitting Cubic Splines

This section describes how an arbitrary function can be estimated by generating the model matrix with a cubic basis function and by solving the resulting system of equations. Cubic splines are B-Splines of order 4 with the additional condition that the first and second derivatives are equal at the knot location. This section explains the the process of fitting a cubic spline to a data set with the predictor variable x and the response y. The function rk() defines a cubic basis given a vector of values x and the relevant knots xk. The basis rk() is a special case of the Cox-de-Boor B-Spline form but follows the same theory Wood (2006). The basis is a possible cubic spline choice. With the function rk() the model matrix X for finding a cubic spline can be constructed. The function spl.X() uses the rk() function to generate a model matrix that contains the basis for each knot x-values combination. This model matrix in combination with the response vector can be solved to produce the coefficient vector β and estimate a spline.

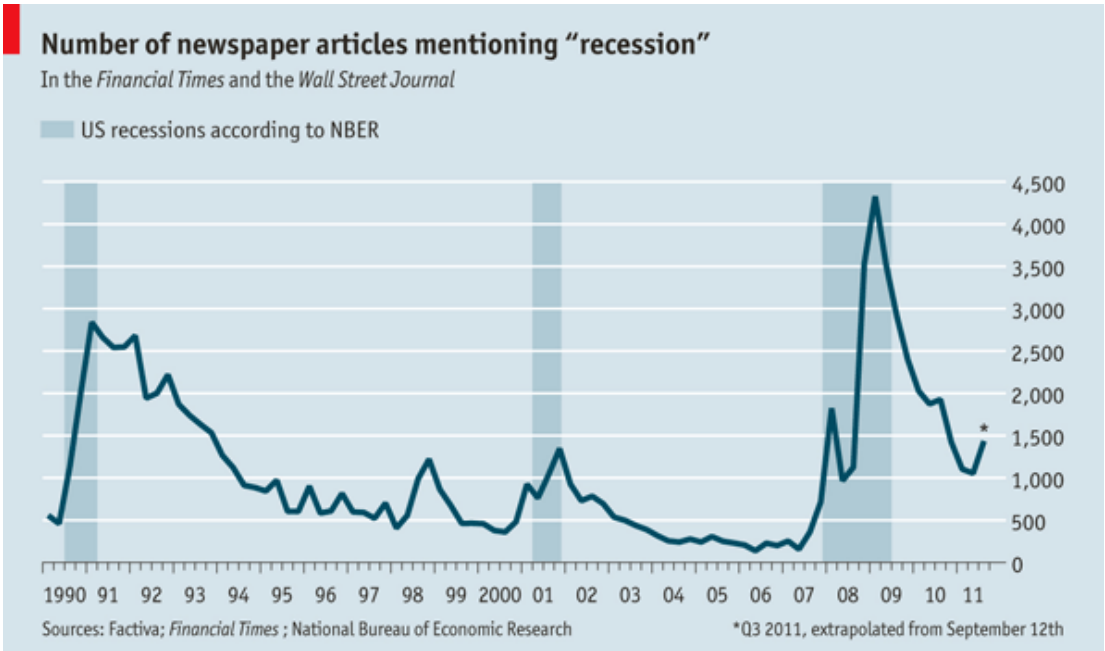

Figure 5.4: A sample times series of equidistant observations where uniform and quantile knots are inappropriate economist (2011)

```
# rk(x,z) constructs the basis for a cubic spline on [0,1]
rk<-function(x,z){
((z-0.5)^2-1/12)*((x-0.5)^2-1/12)/4-((abs(x-z)-0.5)^4-(abs(x-
z)-0.5)^2/2+7/240)/24
}
 # set up model matrix for cubic penalized regression spline
 spl.X<-function(x,xk){
    # number of parameters
    q<-length(xk)+2
    # number of data
    n<-length(x)
    # initialized model matrix
    X<-matrix(1,n,q)
    # set second column to x
    X[,2]<-x
    # and remaining to R(x,xk)
```

```
    X[,3:q]<-outer(x,xk,FUN=rk)
    X}
```

The rows of the model matrix are determined by the number of elements in the data vector x. The first two columns of the model matrix are for encoding and therefore of less interest for this thesis. The remaining elements of the model matrix are computed through the basis function rk(). Given that the splines were designed to be linear in the unknown parameter β, the splines can be estimated with the lm method in R. The ability to fit an arbitrary function by defining the basis and solving the resulting matrix with a linear method is the paramount aspect of regression splines. It is of such crucial importance because this enables the estimation of splines as part of GLMs. This allows for GAM estimation in Spark. To fit the spline, the coefficients are estimated from the model matrix with a linear model. The exact fitting happens via the QR-decomposition described in the linear model section. The estimated coefficients are multiplied by the same design matrix that produced the coefficients to estimate fitted values. This process essentially fits the function to the data that was used to estimate the coefficients. Given a data set of two variables, wear and size, an interpolating cubic spline can be estimated via model matrix X and the linear estimation method $lm()$.

```
# choose some knots
xk<-1:4/5
# generate model matrix
X<-spl.X(x,xk)
# fit model with out the first column
mod.1<-lm(wear~X-1)
# x values for prediction
xp<-0:100/100
# prediction matrix
Xp<-spl.X(xp,xk)
# plot the data with a dot plot with a fitted spline
plot(x,wear,xlab="Scaled engine size",ylab="Wear index")
lines(xp,Xp%*%coef(mod.1))
```

The resulting plot illustrates the data points of wear and size on a two-dimensional plot. The estimated $f()$ interpolates the data points with piecewise cubic splines. The estimated function appears to represent the relationship between wear and engine size. The choice of basis dimensions $q = knots + 2$ was arbitrary and is addressed in the new section. This section

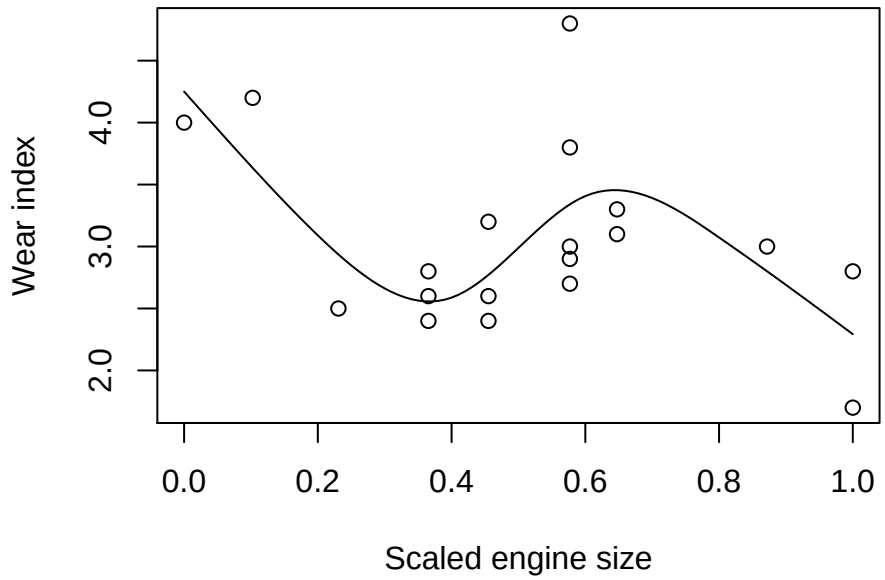

Figure 5.5: Wear engine relationship as a cubic spline

addressed how to estimate a function with the basis function and how to smooth that function is set out in the next section.

5.2.3 Penalized Cubic Splines

Penalized cubic splines seek to find the smoothest interpolating spline by penalizing the 'wiggliness' of a function. The choice of basis order does not suffice to control the smoothness of the resulting spline Wood (2006). The basis does influence the shape of the function but does not suffice to enforce the required smoothness. Penalized regression splines are a viable approach to enforce smoothness. Penalized cubic splines use a smoothing parameter that penalizes the second derivative. The second derivative or acceleration of an interpolating function is also a proxy for over- and underfitting. If the acceleration of an interpolation function is high, the function has enough curvature to account for each data point and will overfit. By penalizing the acceleration of an interpolating function, overfitting can be avoided. In the linear model section over-fitting has been stated as a weakness of non-parametric

methods. As stated, the aim of fitting a function is to find a parameter vector β that minimizes the following equation:

$$\|y - X\beta\|^2 \tag{5.9}$$

The penalty term λ weights the second derivation of the estimated function $f()$. λ penalizes a hectic function by giving less or more weight to the second derivative. As mentioned, the second derivative of a function represents the acceleration or the mentioned 'wiggliness'. With the penalty term the new subject of estimation becomes:

$$\|y - X\beta\|^2 + \lambda \int_0^1 (f''(x))^2) dx \tag{5.10}$$

The trade-off between fitting all x-values and a smooth function is controlled by the penalty term λ. Since the penalty term weights the smoothness criteria, it is also called smoothing parameter. The choice of λ is crucial to the resulting function. While a λ of 0 creates a function that will directly pass each x-data point, a high lambda value will over-penalize each acceleration of the function f and generate a straight line. Since the estimated function f is linear in β, the subject of estimation can be rewritten as:

$$\|y - X\beta\|^2 + \lambda \beta^T S \beta \tag{5.11}$$

The matrix S is a penalty matrix that is specific to the chosen basis. The penalty matrix is a diagonal matrix that contains the differences of adjacent values. For evenly spaced knots the penalty can be written as:

$$P = \sum_{i=1}^{k-1} (\beta_{i+1} - \beta_i)^2 = \beta_1^2 - 2\beta_1\beta_2 + 2\beta_2^2 - 2\beta_2\beta_3 + ... + \beta_k^2 \tag{5.12}$$

And the resulting penalty matrix can be expressed as the multiplication of estimated coefficients and a difference matrix. The diagonal matrix has the difference between adjacent coefficients and its derivatives near the diagonal.

$$P = \beta^T \begin{bmatrix} 1 & -1 & 0 & . & . \\ -1 & 2 & -1 & . & . \\ 0 & -1 & 2 & 0 & . \\ . & . & . & . & . \\ . & . & . & . & . \end{bmatrix} \beta \tag{5.13}$$

Unevenly spaced knots require a more sophisticated penalty term but this penalty matrix has a more illustrative form. The problem of estimating the penalized regression spline is to minimize the just-stated equation with respect to β and to estimate λ. It is important to state that fitting a penalized regression spline requires an estimation of β and λ. Estimating λ will be discussed in the next section. For a computationally stable fitting of regression splines the above formula can be rewritten as the following:

$$\|y - X\beta\|^2 + \lambda\beta^T S\beta = \left\| \begin{bmatrix} y \\ 0 \end{bmatrix} - \begin{bmatrix} X \\ \sqrt{\lambda}B \end{bmatrix} \beta \right\|^2 \tag{5.14}$$

The penalty matrix S for a given basis can be written as its square root B Wood (2006). Any symmetric matrix can be decomposed into the following form: $B = \sqrt{S}$. The model matrix X has been augmented with the square root of the penalty matrix S multiplied by the square root of λ. Since the X matrix is augmented, the vector of y values needs to be augmented to maintain the required dimensions. Since we still are using linear models, the number of elements in the y vector must match the number of rows in the augmented X matrix. A simple square root of a matrix can be written as:

The penalty matrix is specific to each basis function. For the author's choice of the rk basis the matrix S is created by forming an outer-knot product with the basis rk. The resulting matrix S is the specific penalty matrix for a vector of knots and a basis:

```
# set up the penalized regression spline penalty matrix,
# given knot sequence xk
spl.S<-function(xk){
    # dimension of basis
    q<-length(xk)+2
    # initialize matrix to 0
    S<-matrix(0,q,q)
    # fill in non-zero part
    S[3:q,3:q]<-outer(xk,xk,FUN=rk)
    S
}
```

The full model matrix for a penalized spline consists of two combined matrices. The spl.X function constructs the basis and spl.S produces the corresponding penalty matrix. By combining the two matrices, the full matrix of 5.21 is constructed.

```
# function to fit penalized regression spline to x,y data,
# with knots x_knots, given smoothing parameter, lambda.
prs.fit<-function(y,x,x_knots,lambda){
    # dimension of basis
    q<-length(x_knots)+2
    # number of data points
    n<-length(x)
    # create augmented model matrix
    Xa <- rbind(spl.X(x,x_knots),
        mat.sqrt(spl.S(x_knots))*sqrt(lambda))
    # augment the data vector
    y[(n+1):(n+q)]<-0
    # fit penalized regression spline with a linear model
    lm(y ~ Xa-1)
}
```

The prs.Fit function estimates a penalized regression spline for a given vector y of response variables, a vector x of predictor variables, a choice of knots x_knots and a given smoothing parameter λ. The method then forms X, B, y and $\sqrt{\lambda}$ according to 5.21. The formed system of equation is solved by the linear model. The resulting estimated model contains the estimated coefficients. The proposed function can be used to estimate a penalized regression spline for the wear data set.

```
# choose some knots
x_knots <-1:7/8
# choose a smoothing parameter lambda
lambda <- 0.0001
# fit penalized regression spline
mod.2 <- prs.fit(wear,x,x_knots, lambda)
# matrix to map params to fitted values at xp
Xp<-spl.X(xp,x_knots)
# plot data & spl. fit
plot(x,wear);lines(xp,Xp%*%coef(mod.2))
```

From the function prs.fit and 5.21 it should be be evident that the choice of λ has a significant influence on the shape of f. The figure below illustrates the influence of λ on the shape of f

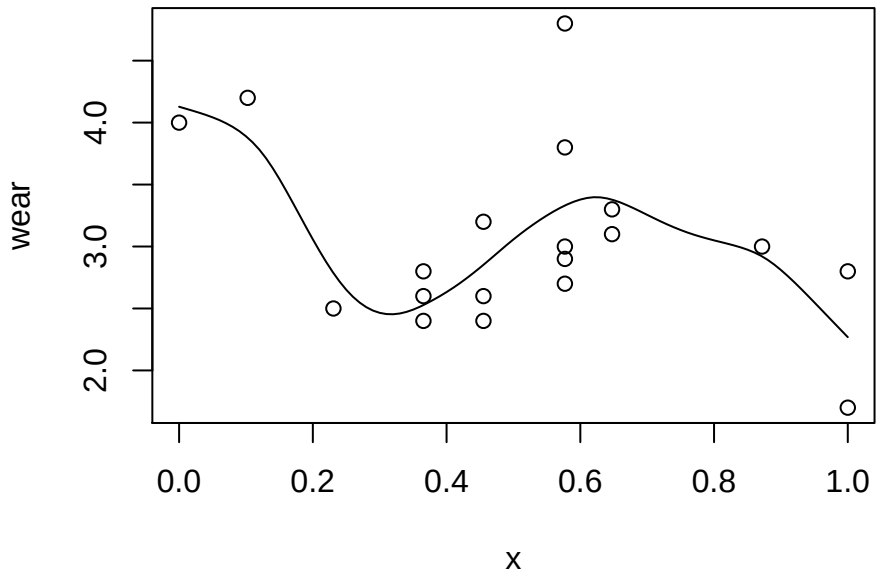

Figure 5.6: Wear engine relationship as a penalized cubic spline

Wood (2006) states that the influence of λ is significant so that it should be estimated individually. This section shows how to estimate the coefficient vector for a penalized regression spline. The estimation of λ is discussed in the next section.

5.3 Smoothing Parameter Estimation

The previous section and figure 5.7 in particular illustrated the importance of the smoothing parameter λ. A high value for λ will cause over-smoothing while a low value will under-smooth the data. Too high or too low, a bad choice for λ will result in a spline $\widehat{f}$ that is far from the original unknown function f. An ideal λ has a small distance between f and $\widehat{f}$. A measure M is defined to capture this notion.

$$M = \frac{1}{n}\sum_{i=1}^{n}(\widehat{f}_i - f_i)^2 \tag{5.15}$$

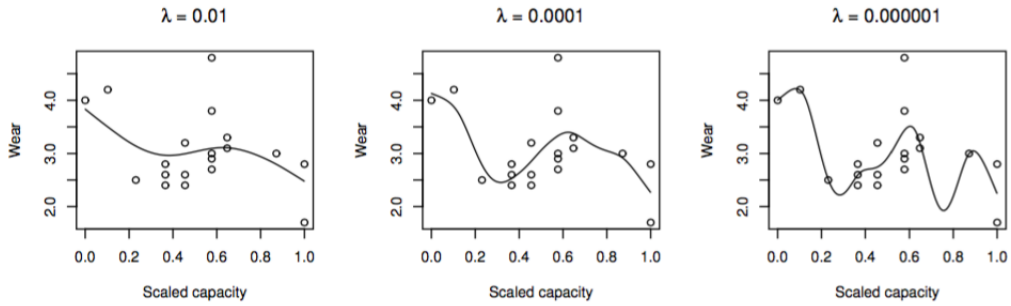

Figure 5.7: λ influence on the estimated spline function f. Wood (2006)

Wood (2006) uses the following notation for the following equations: $\widehat{f}_i \equiv \widehat{f}(x_i)$ and $f_i \equiv f(x_i)$. A suitable criterion to estimate λ is thus to minimize M. However, since the true function f is known, M cannot be estimated but it is possible to estimate the squared error $E(M) + \sigma^2$. Wood (2006) proposes to find λ through cross validation. Cross validation is widely used in statistics. Broadly speaking cross validation validates a model on data outside of the training set. Given $\widehat{f}^{-i}$ is the estimated function on all data except y_i, an ordinary cross validation can be given:

$$V_o = \frac{1}{n}\sum_{i=1}^{n}(\widehat{f}^{-i} - y_i)^2 \tag{5.16}$$

This score results from leaving out each data point in turn, fitting the model to the remaining data and calculating the squared difference between the missing data point and its predicted value. These squared differences are then averaged over all the data. It should be obvious that the process of refitting the function for each y value is $O(n^2)$. A computationally superior cross validation score can be given in terms of a projection matrix Wood (2006).

$$V_o = \frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{f}_i)^2/(1 - A_{ii})^2 \tag{5.17}$$

The author thus proposes the generalized cross validation score ("gcv") to avoid the computational overhead. The function's value at the current index can be expressed in terms of the hat matrix. For the purpose of this thesis, the hat matrix can be any projection in the column space. Computing the cross validation score through the hat matrix allows for validation without forming each possible function for each y value. The computationally most stable version of

the gcv score replaces the weights $1 - A_{ii}$ by the mean weight $tr(I - A)/n$. For a detailed discussion of the gcv see Wahba (1990).

$$V_g = \frac{n \sum_{i=1}^{n} (y_i - \widehat{f}_i)^2}{tr(I - A)^2} \tag{5.18}$$

This section will use this definition to compute the gcv score for a given function and the given hat matrix. Thereby a simple loop can be established in order to find the best possible λ. This is a very simple approach to finding the smoothing parameter that minimize the gcv score. The model producing the smallest gcv score generates the smoothest possible cubic spline for the given data set.

```
lambda<-1e-8
n<-length(wear)
V<-0
for (i in 1:60) # loop through smoothing parameters
    {
        # fit model, given lambda
        mod<-prs.fit(wear,x,xk,lambda)
        # find tr(A)
        trA<-sum(influence(mod)$hat[1:n])
        # residual sum of squares
        rss<-sum((wear-fitted(mod)[1:n])^2)
        # obtain GCV score
        V[i]<-n*rss/(n-trA)^2
        # increase lambda
        lambda<-lambda*1.5
    }
plot(1:60,V,type="l",main="GCV score",xlab="i")
```

```
# extract index of min(V)
i<-(1:60)[V==min(V)]
# fit optimal model
mod.3<-prs.fit(wear,x,xk,1.5^(i-1)*1e-8)
Xp<-spl.X(xp,xk)
```

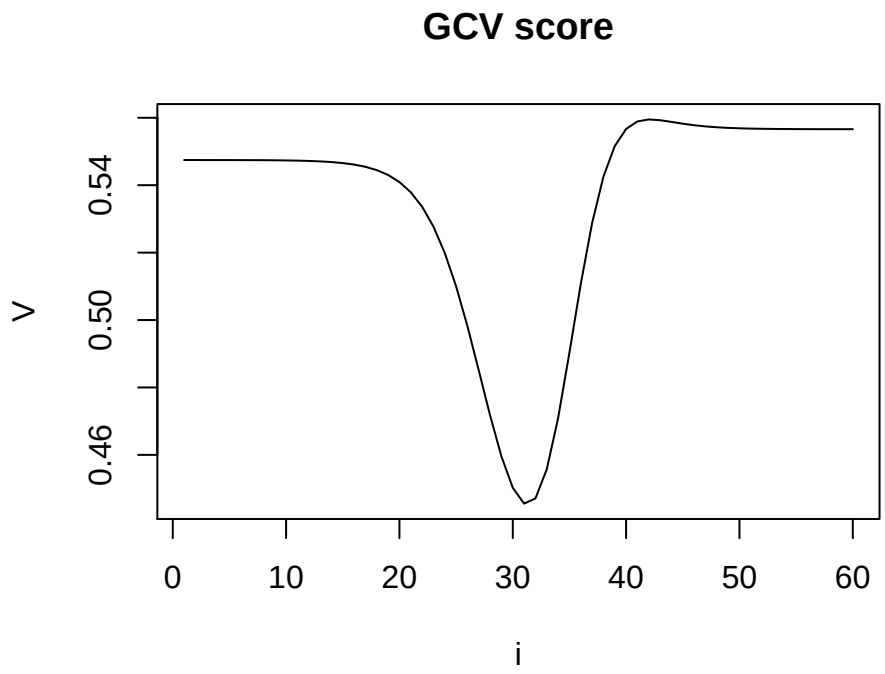

Figure 5.8: A varying penalty term has a significant influence on the resulting GCV score

```
plot(x,wear)
lines(xp,Xp%*%coef(mod.3))
```

This section introduced the gcv score as a measure to evaluate the fit of a penalized regression spline. The gcv score is the central metric in evaluating choices for λ and will be used in the following sections to search for the ideal generalized additive models.

5.4 Additive Model

The discussed methods fit a regression spline to a single variable. Almost all relevant models have more than one variable. The additive model models the response as the sum of several predictors. Wood (2006) formally describes the additive model as:

$$y_i = f_1(x_i) + f_2(z_i)...f_n(w_i) + \epsilon_i \tag{5.19}$$

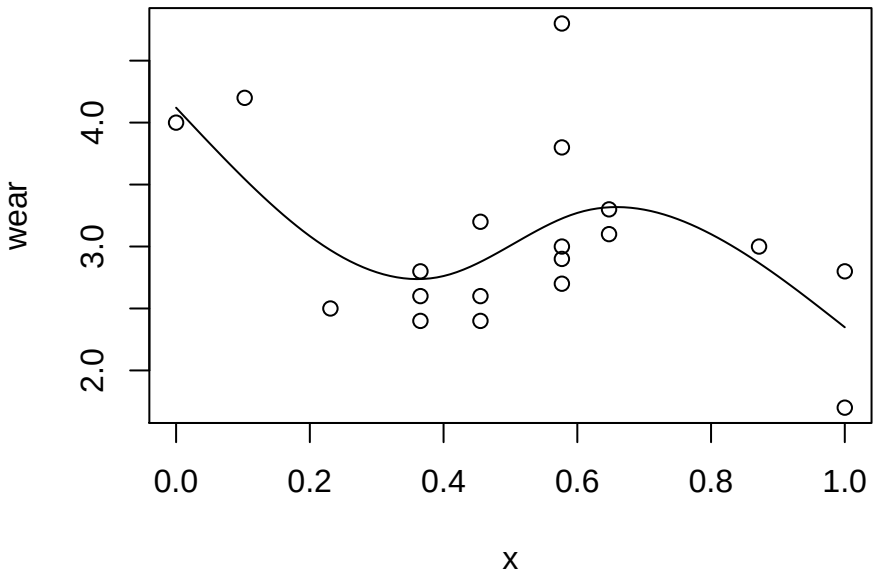

Figure 5.9: The smoothest interpolating spline for the given basis and set of knots

The functions f_j are smooth functions estimated with the methods introduced in the previous sections. The error term ϵ is distributed according to an independent identically distributed $N(0, \sigma^2)$. Modelling y_i as the sum of individual smoothing functions rather than a single function of all terms imposes a very strong condition. $f_1(x) + f_2(z)$ is a special case of the of the general smooth function of both predictors $f(x, z)$. The benefit of modelling each function individually is that each predictor maintains the interpretability of the linear model. Estimating $f(x, z)$ would provide superior flexibility but drastically decreased interpretability. The individual smooth function is a major benefit of the additive model. Fitting additive models with by the methods used for a single regression spline amounts to rewriting the penalty term. Instead of penalizing a single spline both splines need to be penalized. The unknown parameter vector β for the additive model can be estimated by the minimization of the penalized least squares objective.

$$\|y - X\beta\|^2 + \lambda_1 \beta^T S_1 \beta + \lambda_2 \beta^T S_2 \beta \tag{5.20}$$

Each smoothing function is estimated by an individual penalty matrix S_i and a smoothing parameter λ_i. The additive nature of the model enables to write the combined penalty as $S \equiv \lambda_1 S_1 + \lambda_2 S_3$. The ability to rewrite the penalty matrix as the addition of the individual penalty matrices and smoothing parameter enables to rewrite the least squares objective for computation.

$$\|y - X\beta\|^2 + \beta^T S\beta = \left\| \begin{bmatrix} y \\ 0 \end{bmatrix} - \begin{bmatrix} X \\ B \end{bmatrix} \beta \right\|^2 \tag{5.21}$$

Similar to the single smoothing function, the B is any square root matrix such that $B^T B = S$. Summing the different λ and S terms into the square root matrix enables this model to be estimated with the standard linear model. GAM estimation can be performed in two steps: (1) Set up the model and the penalty matrices. (2) Estimate the models with smoothing parameters Wood (2006).

```
  # Get X, S_1 and S_2 for a simple 2 term AM
  am.setup<-function(x,z,q=10){
# generate equidistant knots for simplicity
xk <- quantile(unique(x),1:(q-2)/(q-1))
zk <- quantile(unique(z),1:(q-2)/(q-1))
# Generate two individual, non-overlaping penalty matricies
S <- list()
S[[1]] <- S[[2]] <- matrix(0,2*q-1,2*q-1)
S[[1]][2:q,2:q] <- spl.S(xk)[-1,-1]
S[[2]][(q+1):(2*q-1),(q+1):(2*q-1)] <- spl.S(zk)[-1,-1]
# Set up the model matrix X
    n<-length(x)
    X<-matrix(1,n,2*q-1)
    # 1st smooth
    X[,2:q]<-spl.X(x,xk)[,-1]
    # 2nd smooth
    X[,(q+1):(2*q-1)]<-spl.X(z,zk)[,-1]
    list(X=X,S=S)
}
```

The function am.setup generates the model matrix and the penalty matrices for two predictor variables and a fixed set of equidistant knots Wood (2006). The knots are an arbitrary choice

and only for illustrative purposes. A key aspect is that the function sets up two penalty matrices. To maintain the additive property each penalty matrix gets placed in a scaled penalty matrix that is zeros outside of the individual penalty matrix. The scaled penalty matrix has the size of n rows and the number of parameters multiplied by the penalty matrices. For example, the first penalty matrix covers all rows of for the first number of parameter columns and is zero otherwise. The second penalty matrix is 0 where the previous matrix was defined. Setting the penalty matrix up this way allows for multiplying the smoothing parameter individually to each matrix and maintaining its additive property. The constructed model matrix X and the list of individual penalty matrices are now sufficient to estimate an additive model.

```
# function to fit simple 2 term additive model
fit.am<-function(y,X,S,sp){
    # generate the full penalty matrix and take square root
    rS <- mat.sqrt(sp[1]*S[[1]]+sp[2]*S[[2]])
    # number of params
    q <- ncol(X)
    n <- nrow(X)
    X1 <- rbind(X,rS)
    # augment data
    y1<-y;y1[(n+1):(n+q)]<-0
    # fit model
    b<-lm(y1~X1-1)
    # tr(A)
    trA<-sum(influence(b)$hat[1:n])
    # RSS
    norm<-sum((y-fitted(b)[1:n])^2)
    list(model=b,gcv=norm*n/(n-trA)^2,sp=sp)
}
```

The function fit.am takes the previously generated model and penalty matrix, a vector of y values and a vector of smoothing parameter values and produces a list of the model and the gcv score Wood (2006). Each penalty matrix receives an individual smoothing parameter. Defining the full penalty matrix as zero outside of the individual penalty matrix allows for multiplying the smoothing parameter and then adding the individual matrices to form the full penalty matrix rS. The model matrix on top of the penalty matrix forms the complete model matrix X. Since the model matrix has more rows that the data vector y, the data vectors receives zeros to

match the augmented matrix' number of rows. The resulting model is then fit via the linear model from section two. After the model is fit, the gcv score is computed and stored.

5.4.1 Additive Model Example

The process of fitting a two-term additive model can be best understood by using the setup.am and fit.am functions on a dataset. Wood (2006) uses a default R dataset called *trees*. This dataset contains three variables: Volume, Girth and Height for 31 felled cherry trees. The process of fitting an additive model via cubic splines can by illustrated by explaining the volume of a cherry as a function of girth and height. The author suggests the following model:

$$Volumne = f_1(Girth) + f_2(Height) + \epsilon_i \tag{5.22}$$

This model can be estimated with the functions defined in the previous section.

```
am0 <- am.setup(trees$Girth,trees$Height)
```

The model matrix and the penalty matrices are setup with the rescaled parameters of girth and height. The resulting list can then be used to estimate the full model but not the smoothing parameter. To estimate the smoothing parameter, the fit.am functions is called with several choices of λ. The λ that generates the lowest gcv score is estimated by iteratively trying different λ values.

```
# initialize smoothing parameter (s.p.) array
sp<-c(0,0)
# loop over s.p. grid
for (i in 1:30) for (j in 1:30){
    # s.p.s
    sp[1]<-1e-5*2^(i-1);sp[2]<-1e-5*2^(j-1)
    # fit using s.p.s.
    b<-fit.am(trees$Volume,am0$X,am0$S,sp)
    # number of data
    if (i+j==2) best<-b else
    # augmented X
    if (b$gcv<best$gcv) best<-b
}
 # lowest GCV score producing smoothing parameter
```

```
best$sp

## [1]    0.01024 5368.70912
```

The resulting smoothing parameter for girth is fairly low, presumably allowing f_1 some curvature. Height has a very high smoothing parameter, which, most likely, results in a rather straight line. The values of the smoothing functions at the predictor variable values can be obtained by zeroing all model coefficients except those corresponding to the term of interest and using R's predict function.

The resulting plot confirms the observation that the Girth smooth has more curvature than the Height smooth. The middle figure is the estimate of the smooth function of Girth at the given Girth data. The right figure is the estimate of the smooth function of Height at the given Height data.

5.5 Generalized Additive Model

Generalized additive models (GAMs) follow from additive models, as generalized linear models follow from linear models Wood (2006). Like the GLM, the GAM predicts some known smooth monotonic function of the expected value of the response. The response can follow any exponential family distribution. For the sake of illustrating the similarities to a GLM, the function fit.am can extended to account for a gamma error and log link function.

```
# function to fit simple 2 term generalized additive model
# Gamma errors and log link
fit.gamG<-function(y,X,S,sp){
    # get sqrt of combined penalty matrix
    rS <- mat.sqrt(sp[1]*S[[1]]+sp[2]*S[[2]])
    # number of parameterss
    q <- ncol(X)
    # number of data points
    n <- nrow(X)
    # augmented model matrix
    X1 <- rbind(X,rS)
    # initialize parameters
    b <- rep(0,q);b[1] <- 1
    # initialize convergence control
```

```
    norm <- 0;old.norm <- 1
    # repeat unconverged
    while (abs(norm-old.norm)>1e-4*norm) {
      # generate pseudo data by computing expected value of cur-
rent beta choice
        eta <- (X1%*%b)[1:n]
        # log link, exp of pseudo data
        mu <- exp(eta)
        # compute pseudo data
        z <- (y-mu)/mu + eta
        # augment pseudo data
        z[(n+1):(n+q)] <- 0
        # solve the linear model with pseudo data
        m <- lm(z~X1-1)
    }
    list(model=m,gcv=norm*n/(n-trA)^2,sp=sp)
}
```

The function fit.gamG takes a model matrix X and a list of penalty matrices S. Both can be generated by the am.setup function. The generalized additive model seeks the best smoothing parameter by iteratively searching for a value until convergence. Each iteration step forms a vector of pseudo data. Pseudo data is the estimated value of the current parameter estimation after transformation by the canonical link function. If the resulting gcv score of the current pseudo data does not change, the model converges. This function highlights the two step-approach to fitting GAMs. Step 1: Use a given choice of lambda parameters to generate the penalty matrix. Step 2: Merge the model and penalty matrix and estimate the coefficients.The fitting process of finding β given a choice of λ is called penalized iterative reweighted least squares and is described in the next section.

5.5.1 Penalized Iterative Reweighted Least Squares

While the GLM is fitted by the iterative re-weighted least squares, GAMs are fitted by the penalized reweighted least squares. The major difference is that the minimization objective contains the penalty matrix. The estimation process with the augmented model matrix is illustrated in 5.4.1 and used in the previous section. The method suggested to fit GAMs with

Apache Spark uses an unpenalized model matrix. Details for penalized reweighted least squares can be found in Wood (2006).

6 Generalized additive models for very large datasets

This chapter two describes two ecosystems for estimating the generalized additive model on very large datasets. The R language is a statistical programming language designed to handle statistical models on a single machine. Spark is a cluster computing framework designed to handle terabytes of data in distributed environments. R offers the GAM with a limited cluster computing platform, whilst Spark only offers the GLM. This section uses the aforementioned spline estimation methods to treat the GAM as an augmented GLM in Spark.

6.1 R Overview

R is a programming language designed for statistical modeling and data analysis. R was introduced as the successor to S, which was the first programming language designed for statistics. The designers of the language sharply focused on the realization of statistical concepts and their primary focus lay on executing statistical software on a single machine. Its wide use in academia and the professional world has bred a lively community and large ecosystem for libraries. R offers excellent packages for data manipulation, data analysis and data visualization. The mgcv package provides GAM estimation with a wide array of spline basis and smoothing parameter estimation. The package's gam() method is build to estimate a gam on a single machine with a single or several cores. For very large datasets the mgcv package offers a bam() method that estimate GAMs in a distributed environment Wood u. a. (2015). The bam() implementation leverages the R libraries "parallel" and "snow". Both libraries allow R processes to communicate across a network. However, the R cluster computing environment is a fragmented addition to R's core functionality. The cluster manager and the communication protocol rely on established tools but each acts independently of R. The parallel library cluster manager and the communication protocol are their own ecosystem. This fragmentation renders the execution's monitoring and manging complex. The execution of R code on a cluster thus relies on the orchestration of several independent parts and is a relic of R's initial design Schmidberger u. a. (2009). This coordination of a fragmented cluster ecosystem posed limits on

R's capability to be used on very large datasets. The following sections show how the GAM can be implemented in a homogeneous cluster computing framework called Spark.

6.2 Apache Spark Overview

Apache Spark is a general purpose cluster computing engine designed for handling iterative workloads on very large data sets Zaharia u. a. (2010). Spark is commonly seen as a successor to the class of MapReduce frameworks that emerged in the last decade. The open source projects HDFS, Hadoop and Hive brought large scale data storage, processing and analytics to many academic and professional organisations. HDFS allows for storing petabyte level data on distributed and heterogeneous infrastructure but exposes it through a homogeneous API. Hadoop and Hive are APIs that allow the performance of MapReduce operations on very large data sets stored in HDFS. Spark goes further by offering a high level general purpose API and distributed, in-memory processing. The main advantage of the general purpose API is the ability to abstract from the MapReduce pattern when solving a problem with Spark. Spark offers a flexible API in Scala, Pyhton, Java and R to perform many other operations other than MapReduce. Other than Hive or Hadoop the processing steps do not have to follow strict Map and Reduce patterns. Spark uses a scheduler pattern to execute submitted applications. Every Spark application contains a driver program with the main function and a connection to the cluster manager. This connection is called the Spark Context and connects the application logic to the cluster manager. The cluster manager coordinates the registered worker nodes and distributes the workload. Each worker node contains a set number of executors which perform the actual computation.

This architecture allows for scaling horizontally with increasing workload. Scaling horizontally means that an increase in workload, data or processing, can be accounted for by increasing the number of worker nodes. This is particularly important for handling very large data sets, as the limit of processing and storage is the number of available worker nodes. Modern cloud computing providers facilitate access to an avalanche of machines, thus its capacity is limited by the number of available worker nodes. The traditional limit for statistical models has been computing power, memory and data storage of a single machine. This model accounts for an increasing workload by vertical scaling. Vertical scaling is the process of increasing the computing resources of available machines. It is very resource intensive and a rather inflexible approach. Sparks ability to horizontally scale makes it an ideal environment to compute statistical models for very large datasets.

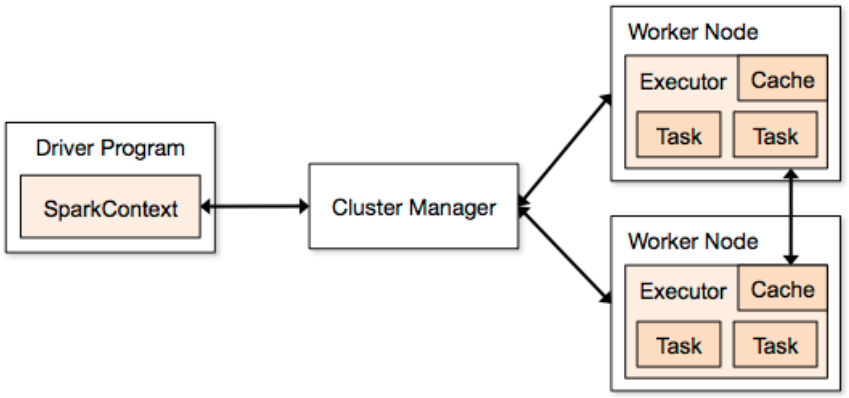

Figure 6.1: Every worker node contains a number of executors. Executors can communicate to other executors within the same application sparkdocumentation

Spark applications perform operations on an abstraction called resilient distributed data set ('RDD'). The RDD is Spark's central data structure and can be thought of as a distributed collection that facilitates four properties: memory abstraction, partitioning, resilience, and lazy execution. A major advantage of Spark is the ability to write a memory optimized application while maintaining a high level of abstraction. Historically memory optimized code often involves writing Fortran or C level code, which forces a very low level perspective on the programmer. Specifying memory allocation enables very well performing code but it forces the programmer to worry about minuscule technical details rather than the actual task. Spark, however, allows for writing memory optimized code while interacting with columns and rows. Thinking about data in terms of rows and columns is a familiar abstraction from other analytics software like Python and R. Spark combines this high level abstraction with low-level optimization that is hidden from the programmer. Spark was designed for cyclic workloads. Performing an iterative operation on the same dataset in Hive or HDFS involved loading the the data from disk several times and performing the task. This lead to a large portion of Hive and Hadoop applications to be spent on I/O, not the actually computation. Spark was designed to address this weakness and maintains the working set in memory for the entire computation of a RDD. Spark uses lazy evaluation to reduce the memory usage of a program. Spark facilitates lazy execution through a concept called lineage. A driver program translates the program into a logical plan of operations performed on a RDD. The execution plan can be thought of as a directed acyclic graph ('DAG') where each vertex is a RDD and each edge a transformation. The logical execution plan adds an edge and a vertex for each operation, thus the root of the graph is the initial RDD and the result one of the leafs. The path from the root to a vertex is called its

linage. During the execution of the driver program no computation on any RDD is performed until a result is required by a different computation or by the programmer. Spark realizes this by dividing the operations into two classes: Transformations and actions. Transformations will add a vertex and an edge to the DAG without executing the calculation. Map() and Filter() are common examples of transformations. Actions add a vertex and an edge but require particular values to be returned and thus the execution of all upstream transformations. The concept of lineage also allows for quickly recovering a failed RDD from its predecessors by simple applying the last transformation to its parent RDD. The ability to recover fast from failure adds reliability. The RDD serves as the internal data structure for Spark and the application developer usually interacts with either the DataFrame or DataSet API. Spark is written in Scala, a functional programming language for the Java virtual machine ("JVM"). The JVM has been established as an environment for parallel computing and is considered a stable platform for parallel computation Al-Jaroodi u. a. (2002).

Sparks scalability and in-memory processing capabilities provide an ideal environment for handling large datasets. The ability to perform iterative workloads in-memory on a very large scale has been proven to be a powerful framework for solving problems on a large scale Xin (2014).

6.3 Generalized Linear Models in Spark

Spark offers a GLM implementation as part of their machine learning library. Sparks high level API and in-memory processing can be used for a wide array of applications. For example, Spark offers libraries for streaming, SQL and graph computation. This thesis leverages the machine learning library ('MLlib') to implement GAMs in Spark. MLlib was designed to offer practical machine learning whilst maintaining scalability and ease of use. It offers many common learning algorithms including classification, regression and clustering. The GLM is specified in SparkR, a R language binding for some MLlib models. This approach allows for writing R code whilst executing a Spark job. The language binding is realized as an object mapper that passes the data and model description from R to Spark. The major advantage of SparkR is the ability to define models with an RFormula. Statistical models in R are traditionally expressed in terms of RFormula's domain specific language ('DSL'). The DSL contains common statistical model descriptions, such as ' ' explain, '+' column, ':' interaction, '-' deletion and '.' all. For example, the RFormula for a model explaining Y in terms of X_1 and the interaction between X_2 and X_3 looks like the following: $YX_1 + X_2 : X_3$. SparkR is implemented as an R package that provides a connection to a Spark cluster manager and the object mapper.

MLlib defines applications in terms of pipelines, estimators and transformers. A pipeline is defined as a series of stages, which can be either be an estimator or a transformer. Sparks MLlib uses the abstraction of DataFrame to interact with an RDD. A DataFrame is a SQL-like interface to the data stored in the RDD. The DataFrame API is similar to SQL regarding the querying mechanics and data manipulation through registered User Defined Function ("UDF"). All operations on a DataFrame are expressed in terms of a SQL query or an UDF. Every data selection, filter or manipulation must be stated as an SQL query or an UDF. UDFs are usually row-wise manipulations that have access to the columns of particular rows and primitive constants. A MLlib pipeline originates at a particular DataFrame and passes through several estimator and transformer stages. Every MLlib transformer implements a transform() method, this method transforms a given RDD into a different RDD. For example, a given transformer may add a new column to its parent DataFrame via an UDF. An estimator is an abstraction for the used learning model. Each estimator implements the method fit() which accepts an RDD and returns a transformer. The resulting transformer can then be used to produce predictions for a given DataFrame containing the columns of the original DataFrame.

The GLM in Spark are realized in two connected modules: SparkR and MLlib. Spark GLMs are defined in the R environment via SparkR. The SparkR library provides a spark.glm() function in R that takes the RFormula, the distribution family, along with a pointer to the the associated data frame and wraps it as a Java object that is sent to Spark. This signature of the spark.glm() is identical to the classic glm(). This object is then converted to a GLMWrapper in MLlib. The GLMWrapper has a fit() method that produces a pipeline with two stages: First, the RFormulaParser and second, the generalized linear regression. The RFormlulaParser is a transformer that encodes the input columns according to the RFormula into a model matrix. String and factor encoding happens as part of the RFormulaParser transformer. The model matrix is row wisely added to the original DataFrame as a new column called 'feature'. The Y variable is added as a column called 'label'. The DataFrame with these two added columns is passed to the next pipeline stage, the generalized linear regression. The generalized linear regression is an estimator that generates an IRLS MLlib object. The IRLS object implements the fitting of generalized linear models as described in 4.2.2. The IRLS object takes the 'feature' and 'label' columns of the DataFrame and adds initial weights to each observation with a new 'weight' column. A DataFrame only containing the 'feature', 'label' and 'weight' columns is then used to create a weighted least squared ("WLS") object. The WLS object uses basic linear algebra subprograms ("BLAS"), a highly optimized Fortran routine, to solve the system of equations using the Cholesky Decomposition. This IRLS object repeats weight adjustment and the WLS estimation until convergence.

6.4 Casting the generalized additive model as a generalized linear model in Spark

A very useful property of the GAM is the ability to cast them as a special case of the GLM Wood (2006). As outlined in previous sections, the fitting process of GLM and GAM are similar, the difference being the minimization subject. GLMs are estimated by minimizing the squared error with the IRLS method while GAMs are estimated by minimizing the penalized squared error with the P-IRLS method. However, a GAM can also be estimated with IRLS when splines are represented as basis functions and the penalty term is neglected. The GAM uses a penalty term to safeguard against overfitting; the more knots are used to estimate a spline, the likelier an estimated function too close to the training data becomes. A moderate number of knots can alleviate this problem by restraining the estimated function's ability to capture the data. The appropriate number of knots is a deliberate design choice and should aim to capture the underlying trend, not each data point. Estimating GAMs as a GLM with spline estimation is thus prone to overfitting and this must therefore be deliberately addressed by the person crafting the statistical model by an appropriate choice of knots. The question of penalty term estimation is thus deferred to the choice of knots. This approach is less safe than traditional GAM fitting with smoothing parameter estimation but allows harnessing the existing GLM and IRLS functionality offered in MLlib.

The MLlib GLM implementation can be modified to fit some predictor variables as a regression spline. To realize the spline estimation of a predictor variable, GLM pipeline must account for two changes: the RFormulaParser stage must be able to parse a spline term and the model matrix must be augmented with the basis representation of the spline. A major advantage of this approach is that the generalized linear regression stage of the GLM is able to process the augmented model matrix. To realize smoothing term parsing and model matrix augmentation, a new transformer stage is introduced to the GLM. This stage is called SplineTransformer and precedes the RFormulaParser and the GLR of the GLM fit() pipeline.

Parsing the spline term and the modification of the model matrix is realized as an additional stage of the GLM fit() pipeline. A SplineTransfomer transformer is introduced before RFormulaParser and the Generalized Linear Regression Estimator. The SplineTransformer facilitates the two additions required for the GAM to be cast as a GLM. First, the SplineTransformer parses the spline terms in the RFormula. Borrowing from R's mgcv package, a spline term can be specified as 's(predictor name, basis=cubic, knots=List(1,2,3...)'. Second, the parsed spline contains three information required to form the basis. Given the basis order, the column to be expressed as a spline and number of knots the basis functions are formed via the Cox-de-Boor

form. The Cox-de-Boor 5.6 allows to represent a spline through its basis. This is realized by creating a function b() that constructs the basis functions according to the Cox-de-Boor form. This function is then used on each value of the predictor variable column to produce the basis.

```
 def b(k: Int, i: Int, x: Double, knots: Vector[Double]): Double = k match {
    case 1 =>
        if (knots(i) <= x && x < knots(i + 1)) return 1 else 0
    case _ =>
        (x - knots(i)) / (knots(i + k - 1) - knots(i)) * b(k - 1, i, x, knots) +
        (knots(i + k) - x) / (knots(i + k) - knots(i + 1)) * b(k - 1, i + 1, x, knots)
}


val toSplineFunctionVector: (Double, Int, String) => Vector[Double] = {
     (x, order, knotsList) =>
     Vector.tabulate(stringListToVector(knotsList).length-order){
        index => b(order,index,x,stringListToVector(knotsList))}
}

val toSplineFunctionVectorUdf = udf(toSplineFunctionVector)
```

The UDF adds each element of the basis function vector as an individual column. Once the spline terms have been added, the RFormula is modified to account for the new columns. To leverage the existing GLM functionality, the spline terms s() are removed from the RFormula and the individual spline basis columns are added as a simple predictor '+'. This allows for treating the resulting DataFrame with the augmented RFormula as the an ordinary MLlib GLM. Thus a GAM can be fitted by representing the smooth function through the Cox-de-Boor form and treating the resulting model matrix as the minimization subject of a least squares method. This approach only accounts for spline estimation of some predictor variable but no the smoothing parameter.

6.5 GAM in Spark Example

This section shows a working example of a GAM fitted with Spark using the MLlib GLM extension described in the previous section. The transformation of the DataFrame and the RFormula becomes apparent in the log provided below.

```
# load the manipulated SparkR library
library(SparkR
        , lib.loc = c(file.path(Sys.getenv("SPARK_HOME")
        , "R"
        , "lib")))

# create a Spark context with 6 virtual worker nodes and 4g of mem-
ory
```

```
sc <- sparkR.session(master = "local[6]"
                     , sparkEnvir = list(spark.driver.memory="4g"))

# create a Spark DataFrame from the local R data table
df <- createDataFrame(sc, cars)

# describe a Mllib GLM using the RFormula in SparkR
# the spline 's(..)' term contains a column, basis and knot vector
# data is pointing to a Spark DataFrame
spark_gam <- glm(
             wt ~ disp + cyl + s(wt, bs=cs, knots=
                 c(2.01,2.3,3.2,3.6,3.9,4.1,4.5,4.9,5.2,5.4))
            , data=df
            , family="poisson")

# Spark Log Output:
#
# Passed formula string
wt ~ disp + cyl + s(wt, bs = cs, knots = c(2.01, 2.3, 3.2, 3.6,3.9,
    4.1, 4.5, 4.9, 5.2, 5.4))

# DataFrame columns
 mpg, cyl, disp, hp, drat, wt, qsec, vs, am, gear, carb

# The SplineTransformer now transforms the given RFormula
# and the DataFrame

# The augmented RFormula now describes the model
 wt ~ disp + cyl + wt_smooth_0 + wt_smooth_1 + wt_smooth_2 +
    wt_smooth_3 + wt_smooth_4 + wt_smooth_5

# The augmented DataFrame only contains columns relevant
# for the model:
disp, cyl, wt_smooth_0, wt_smooth_1, wt_smooth_2, wt_smooth_3,
    wt_smooth_4, wt_smooth_5

# The model is estimated and the coefficients printed
summary(spark_gam)
Deviance Residuals:
```

```
(Note: These are approximate quantiles with relative error <= 0.01)
Min        1Q    Median        3Q       Max
-0.50474  -0.13968   0.00221   0.12046   0.45305

Coefficients:
               Estimate   Std. Error  t value   Pr(>|t|)
   (Intercept) 0.75158    0.66655     1.1276    0.25951
   disp        0.0028066  0.0025587   1.0969    0.27268
   cyl         -0.058885  0.20252     -0.29076  0.77124
   wt_smooth_0 0.2802     0.64283     0.43589   0.66292
   wt_smooth_1 0.011658   0.50666     0.023009  0.98164
   wt_smooth_2 0.57985    1.3826      0.41938   0.67494
   wt_smooth_3 -3.7067    9.4968      -0.39031  0.6963
   wt_smooth_4 56.892     130.5       0.43596   0.66287
   wt_smooth_5 1.6079     14.822      0.10848   0.91362

   (Dispersion parameter for poisson family taken to be 1)

       Null deviance: 9.2191  on 31  degrees of freedom
   Residual deviance: 1.6766  on 23  degrees of freedom
   AIC: 114.5

   Number of Fisher Scoring iterations: 4
```

7 Conclusion and Future Work

7.1 Conclusion

This thesis has shown that the GAM can be implemented in Apache Spark by extending the GLM implementation in MLlib. Estimating splines of some predictor variables as part of the GLM by representing them as B-Splines proves to be a feasible working approach. The Cox-de-Boor form is a simple way to generate B-Splines but special cases like cyclic- or natural-splines require additional work. Apache Spark's MLlib and SparkR offer a plethora of functions but changing existing functionality requires detailed knowledge of internals. Spark's internals are under constant development, which causes some core data APIs to be inconsistent. Spark offers three APIs: RDD, DataSet and DataFrame. The GLM implementation uses all three APIs at different stages. Some API decisions are nebulous for external developers which might be only due to historic reasons. All three data APIs feel unfamiliar for R developers. DataSet and DataFrame require SQL-like interactions via an SQL query statement or an UDF. Expressing model matrix manipulations in terms of SQL statements requires additional effort. The traditional RDDs are a plain immutable collection with little functionality. This appears to be the right level of abstraction for application development, however, the matrix and vector approach in R seems a better fit to express statistical software. Sparks high level approach comes from its role as scheduler rather than computation environment. Spark itself does not perform any relevant linear algebra operations, but solely acts as a scheduler and coordinator for Fortran or Java. The solver for the IRLS algorithm in Sparks GLM is a wrapper for the Fortran package BLAS. This is very similar to R, where also many relevant operations are performed in C or Fortran. For example, the QR-decomposition in R uses the Fortran library LAPACK. Therefore a central question arises: Which environment offers a better scheduler and coordinator for highly optimized linear algebra operations, R or Spark? Spark is a framework built to schedule and operate distributed on a horizontally scalable infrastructure. The underlying JVM has been well established in distributed computing and is considered stable. R, on the other hand, was designed to perform statistical computation on a single machine. Thus, using Spark for

statistical computation might be the wrong approach but using Spark to schedule statistical software for very large data sets is very powerful.

7.2 Future Work

The GAM has two distinguishing features: spline estimation of predictor variables and smoothing parameter selection to avoid overfitting. This thesis' implementation focuses on the spline estimation with Cox-de-Boor basis functions. Spline estimation of predictor variables are a key aspect of the GAM but are prone to overfitting. A poor choice of knots, number of knots or knot location may cause a fit that mimics the underlying pattern too accurately and overfits the given data. The GAM is designed to safeguard against overfitting by penalizing functions that follows the data too closely. The next immediate step should be to implement the smoothing parameter estimation of the GAM. Smoothing parameter estimation requires searching for the smoothing parameter that produces the lowest generalized cross validation score. Smoothing parameter estimation needs to answer two questions: How is the penalty realized? How is the search performed?

The easiest way to represent the penalty term is to use P-Splines. P-Splines allow B-Splines to be extended with a penalty term. A simple implementation would realize the penalty term as a Difference Matrix appended below the matrix generated by the Cox-de-Boor form. The model matrix augmented with the penalty term can be estimated with the IRLS. This is how the penalty term is realized in 5.5. The implementation in Spark would require to add a transformer stage to the GAM fit() pipeline. The current GAM pipeline includes three stages: SplineTransfomer, RFormulaParser and Generalized Linear Regression. A P-Spline approach inserts a transformer after the RFormulaParser, which adds the Difference Matrix row-wise to the model matrix generated by the RFormulaParser. The label column would be augmented with 0 to match the new number of rows. The IRLS solver in the GLR stage is still able to handle the augmented model matrix. While this convinces by simplicity, it suffers greatly from unbalanced knots. P-Splines are a great choice for equidistant knots but not for a general implementation. A more general approach for unbalanced knots can found in Wood (2006).

Searching for the lowest gcv producing smoothing parameter can be done by a simple trial of a human defined range. This is the same approach used in the example for the GAM section. This gcv search can be realized in Spark as an estimator. This estimator would implement its own fit() function which uses a human defined range to run the GLM fit() with the augmented penalty matrix for each value in the range. This assumes the smoothing parameter to be in a specific range and depends highly on the chosen range but is the simplest

realization of smoothing parameter estimation. The estimator would return the coefficients of the augmented GLM that produced the lower gcv score. There are many more approaches to smoothing parameter estimation discussed in Wood (2006). A more efficient way of smoothing parameter estimation can be realized by computing the gcv score by forming only part of the model matrix and computing the score in parallel Wood u. a. (2015).

Bibliography

[sparkdocumentation] : *Cluster Mode Overview.* – URL https://spark.apache.org/docs/1.1.1/cluster-overview.html

[economist 2011] : *Up means down.* 2011. – URL http://www.economist.com/node/21529079

[Al-Jaroodi u. a. 2002] Al-Jaroodi, J. ; Mohamed, N. ; Jiang, Hong ; Swanson, D.: A comparative study of parallel and distributed Java projects for heterogeneous systems. In: *Parallel and Distributed Processing Symposium., Proceedings International, IPDPS 2002, Abstracts and CD-ROM*, April 2002, S. 8 pp–

[Cramer 1974] Cramer, Harald: *Mathematical methods of statistics.* 13. print. Princeton : Princeton Univ. Press, 1974 (Princeton mathematical series ; 9). – XVI, 575 S. S. – ISBN 0-691-08004-6 ; 978-0-691-08004-8

[De Boor 2001] De Boor, Carl: *A practical guide to splines; rev. ed.* Berlin : Springer, 2001 (Applied mathematical sciences). – URL https://cds.cern.ch/record/1428148

[Dobson und Barnett 2008] Dobson, Annette J. ; Barnett, Adrian G.: *An Introduction to Generalized Linear Models, Third Edition.* Boca Raton, FL : Chapman & Hall/CRC Press, 2008 (Texts in Statistical Science). – URL http://eprints.qut.edu.au/15448/. – For more information about this book please refer to the publisher's website (see link) or contact the author.

[Fahrmeir u. a. 2009] Fahrmeir, Ludwig ; Kneib, Thomas ; Lang, Stefan: *Regression: Modelle, Methoden und Anwendungen (Statistik und ihre Anwendungen) (German Edition).* 2. Springer, 9 2009. – URL http://amazon.com/o/ASIN/364201836X/. – ISBN 9783642018367

[Hastie 1990] Hastie, Trevor: *Generalized additive models.* London New York : Chapman and Hall, 1990. – ISBN 0412343908

[Hastie u. a. 2001] Hastie, Trevor ; Tibshirani, Robert ; Friedman, Jerome: *The Elements of Statistical Learning*. New York, NY, USA : Springer New York Inc., 2001 (Springer Series in Statistics)

[J. A. Nelder 1972] J. A. Nelder, R. W. M. W.: Generalized Linear Models. In: *Journal of the Royal Statistical Society. Series A (General)* 135 (1972), Nr. 3, S. 370–384. – URL http://www.jstor.org/stable/2344614. – ISSN 00359238

[James u. a. 2014] James, Gareth ; Witten, Daniela ; Hastie, Trevor ; Tibshirani, Robert: *An Introduction to Statistical Learning: With Applications in R*. Springer Publishing Company, Incorporated, 2014. – ISBN 1461471370, 9781461471370

[Myung 2003] Myung, In J.: Tutorial on Maximum Likelihood Estimation. In: *J. Math. Psychol.* 47 (2003), Februar, Nr. 1, S. 90–100. – URL http://dx.doi.org/10.1016/S0022-2496(02)00028-7. – ISSN 0022-2496

[Schmidberger u. a. 2009] Schmidberger, Markus ; Morgan, Martin ; Eddelbuettel, Dirk ; Yu, Hao ; Tierney, Luke ; Mansmann, Ulrich: State of the Art in Parallel Computing with R. In: *Journal of Statistical Software* 31 (2009), Nr. 1, S. 1–27. – URL https://www.jstatsoft.org/index.php/jss/article/view/v031i01. – ISSN 1548-7660

[Wahba 1990] Wahba, Grace: *Spline models for observational data*. Philadelphia : Society for industrial and applied mathematics, 1990 (CBMS-NSF regional conference series in applied mathematics). – URL http://opac.inria.fr/record=b1080403. – Based on a series of 10 lectures at Ohio State University at Columbus, Mar. 23-27, 1987. – ISBN 0-89871-244-0

[Wood u. a. 2015] Wood, Simon N. ; Goude, Yannig ; Shaw, Simon: Generalized additive models for large data sets. In: *Journal of the Royal Statistical Society: Series C (Applied Statistics)* 64 (2015), Nr. 1. – URL http://dx.doi.org/10.1111/rssc.12068. – ISSN 1467-9876

[Wood 2006] Wood, S.N: *Generalized Additive Models: An Introduction with R*. Chapman and Hall/CRC, 2006

[Xin 2014] Xin, Reynold: *Apache Spark officially sets a new record in large-scale sorting*. 2014. – URL https://databricks.com/blog/2014/11/05/spark-officially-sets-a-new-record-in-large-scale-sorting.html

[Zaharia u. a. 2010] Zaharia, Matei ; Chowdhury, Mosharaf ; Franklin, Michael J. ; Shenker, Scott ; Stoica, Ion: Spark: Cluster Computing with Working Sets. In: *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*. Berkeley, CA, USA : USENIX Association, 2010 (HotCloud'10), S. 10–10. – URL http://dl.acm.org/citation.cfm?id=1863103.1863113

[Zhang 2006] Zhang, Jianwei: *Maschinelles Lernen Vorlesungen*. 2006. – URL https://tams.informatik.uni-hamburg.de/lehre/2006ss/vorlesung/maschinelles_lernen/folien/06-16-bw.pdf

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 26. September 2016 Kai Thomas Brusch