# Fitting Generalized Additive Models for very large datasets with Apache Spark

An exposé for a bachelor thesis

Kai Thomas Brusch

31.10.2015

**Summary** This exposé describes content, goals and motivation of my bachelor thesis titled "Fitting General Additive Models for very large datasets with Apache Spark".

**Contact** kai.brusch@gmail.com

**Location** Hochschule für Angewandte Wissenschaften Hamburg

**Department** Dept. Informatik

**Examiner** Prof. Dr. Michael Köhler-Bußmeier

**Second examiner** TBA

# Contents

# 1 Introduction

## 1.1 Context

Regression analysis have become a corner stone of modern statistical analysis. Discovering the relationship between dependeant and independat variables offers great insight into the underlying system and can provide far reaching predictive capabilities. Linear models describe the relationship explain the dependant variable as the sum of the independant variables and the linear interaction with a coefficent. The general additive model describes a modelling approach which explores the relationship between dependant and independant variables as the sum of smoothing functions. This modeling approach offers more flexiblity and power than linear model and has been succesfuly established in academia and industry. However, the additional power of GAMs comes at the cost of additional computational effort, placing strict constraints on the model and data size. Modern literatur suggest two optimization to eliviate this constraint: parallelization and numerical optimization. The optimizations have been implmented and used in high-level statiscial languanges like R and SPASS. While they are already allowing to model fit on more data I feel their implementation in a truly distributed enviroment could bring more power. Apache Spark offers a general purpose cluster computing engine. It's general and distributed nature offers the ideal enviroment to fit general additive models in a distributed enviroment.

My thesis and this expose is closely tied to [**?**]. [**?**] establishes all relevant theory to GAMs and [**?**] extends this theory to the context of large data sets.

## 1.2 General Linear Models

Linear models are statistical models in which a univariate response is modelled as the sum of a 'linear predictor' and a zero mean random error term. The linear predictor depends on some predictor variables, measured with the response variable, and some unknown parameters, which must be estimated. A key feature of linear models is that the linear predictor depends linearly on these parameters. Statistical inference with such models is usually based on the assumption that the response variable has a normal distribution. Linear models are used widely in most branches of science, both in the analysis of designed experiments.

$$y_\mathrm{i} = \beta x_\mathrm{i} + \varepsilon_\mathrm{i} \tag{1}$$

The equation 1 introduces formal notation on how to describe the dependant variable y as linear combination of x and the unknown parameter βplus an error term ε. We are seeking βto fit all data as closely as possible and introduce S as the squared sum which serves as a good proxy for goodness of fit.

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

The common measure of βis the sum of squares defined in 2, Markov-Gaus have proven that the best possible unbiased estimator is βthe minimization of 2.

$$y = \mathbf{X}\beta \tag{3}$$

We can rewrite 1 in scalar form 3. The dependante variable vector $y$ is the linear combination of model matrix $X$ and the vector of unknown parameter β.

## 1.3 Generalized Linear Models

Generalized linear models (GLMs) somewhat relax the strict linearity assumption of linear models, by allowing the expected value of the response to depend on a smooth monotonic function of the linear predictor. Similarly the assumption that the response is normally distributed is relaxed by allowing it to follow any distribution from the exponential family (for example, normal, Poisson, binomial, gamma etc.).

$$g(\mu_{\mathrm{i}}) = \mathbf{X}\beta \tag{4}$$

Equation 4 introduces formal notation for GLMs and illustrates the simmilarities to the genral linear model. It is important to recognize the smooth monotonic function *g()*. The smooth monotonic function, also known as link function, is the key extension which enables to model members of the expotential family with a linear model.

## 1.4 Generalized Additive Models

A Generalized Additive Model (GAM) extends the GLM in by specifing the linear prediction in terms of a sum of smooth functions of predictor variables. The exact parametric form of these functions is unknown, as is the degree of smoothness appropriate for each of them. To use GAMs in practice GLMs must be extende with the following mechanics:

- The smooth functions must be represented somehow.
- The degree of smoothness of the functions must be made controllable, so that models with varying degrees of smoothness can be explored.
- Some means for estimating the most appropriate degree of smoothness from data is required, if the models are to be useful for more than purely exploratory work.

With the outlined additions to GLMs we can now introduce more notation to get a better unterstanding of GAMs.

$$g(\mu_{\mathrm{i}}) = \mathbf{X}_{\mathrm{i}}\Theta + f_1(x_{1\mathrm{i}}) + f_2(x_{2\mathrm{i}}) + f_3(x_{3\mathrm{i}}, x_{4\mathrm{i}})... \tag{5}$$

It is important to note that yi = E(Yi). Xi is a row of the model matrix with parametric component. O is the parameter vector and fJ are the smooth function of the covariates xk. Specifing Yi only interms of smooth functions allow for more flexible modelling. The smoothing function f is easiest understood in a simple univariate case illustrated bellow.

$$y_i = f(x_i) + \epsilon_i \tag{6}$$

An essetial element of GAMs is selecting a base for the smooth function. To keep the within the theory already developed for linear models we require the smoothing function to be represented by a linear basis. Since the base function is a choice we can treat the base as known. For the sake of illustration we assume 1.4 can be rewrite as the following equation if bi(x) is the ith basis function:

$$f(x) = \sum_{i=1}^{q} b_{\mathrm{i}}(x)\beta_{\mathrm{i}} \tag{7}$$

In 7 we the treat the b to be completly known and the linearity of 7 is given by β. To make this a bit more concrete we now assume our basis function to be a fourth order polynomial allowing us to rewrite 7

$$f(x) = \beta_1 + x\beta_2 + x^2\beta_3 + x^3\beta_4 + x^4\beta_5 \tag{8}$$

Which when applied to yields the full model for a univariate GAM.

$$y_i = \beta_1 + x_i\beta_2 + x_i^2\beta_3 + x_i^3\beta_4 + x_i^4\beta_5 + \epsilon_i \tag{9}$$

Equation 9 gives a formal full description of $y_i$ as the sum of a fourth degree polynomial. Fourth order polynomial basis function enforce rather strict limitations and often struggle to provide a desireable fit. Instead of trying to fit a single fourth order polynomial to all our data points we will now seek to fit a fixed degree polynomial between certain sections of our curve. This approach is called regression splines and follows this general pattern:

- Divide the curve in to a fixed number of sections, the point between two sections is called a knot.
- Find a nth degree polynomial for each section between two knot
- Each nth degree polynomial must match its neighboors in value, first and second derivative at the location of the knot.
- Each spline has zero second derivatives at the knot location to ensure a smooth transition.

Using fixed degree polynomial splines as a basis for f means that 1.4 becomes a linear model identical to 3, this is very exciting because we can now use well established linear model theory to fit the very flexible class of GAMs.

The number and location of knots as well as the degree of the polynomial are a crucial part of model with GAMs. One could use hypthesis testing or backward selection to find a good configuration of these paramters but the following methods has distinguished itelsef through applicability and simplicity.

Instead of searching for a proper base function we will keep the base function fixed and a term to controll the smoothness by adding a 'wiggliness' penalty to the least square fitting objective.

$$\|y - X\beta\|^2 \tag{10}$$

Insteat of just minimizing the squared error for parameter stimation from 3 we will have to minimize

$$\|y - X\beta\|^2 + \lambda \int_0^1 [f''(x)]^2 dx \tag{11}$$

The integrated square root of second deriviative penalizes models that have a too 'rough' function. The trade off between model fit and and model smoothness is controlled by the smoothing parameter

## 1.5 Generalized Additive Models for large data sets

[?] [?] [?]

## 1.6 Apache Spark

Distributed computing is a special interest of mine. I've had the pleasure of setting up a SparkR cluster and have been following its development closely. Apache Spark advertises itself as a fast and general cluster computing engine for large-scale data processing. He a overview of it's key features:

- fast, Apache Spark currently holds the world record in sorting 1 TB on data, beating the previus record help by Hadoop. Spark reaches this high speed by removing I/O to disc and maintaining all relevant in memory.
- general, Aparche Spark offers a high level API called Resilient Data Set. RDDs are designed to facilitate a resiliend, distributed data set as a optimal level of interaction with data. Each of the key features of RDDs are outlined below:
  - Resilient: Every RDD is evaluated lazy and constructed from it's lineage, point to different previous RDD.
  - Distributed: RDDs are partiioned accross many machines
  - Data set: RDDs are ment to think about in terms of matrixes, this should feel very natural for people familar with data.
- cluster computing, Apache Spark is build to be executed in a cluster enviroment like Mesos or Yarn. The management and design of these systems are not subject of my bachelor thesis but are centainly worthy of one.
- large scale data processing, the data managable by Apache Spark is limited by the amount of memory availble in a cluster. The advent of obiquis cluster computing and the ability to scale horizontaly

The implementation of the outlined optimization in the previously described enviroment is explained in the next section.

## 1.7 General additive models in Apache Spark

A general

- QR Update
- Parallel computation
- Getting f, r and bla without X
  - F
  - R
  - X

## 1.8 Leading question

**Can the distributed, numerically optimized, in-memory version of GAM fitting beat previous benchmark implementation in model size, data size and fitting speed?**

Subquestions arising are:

- How much faster is a parallel variant against the sequential one?
- How much faster is the Spark in-memory variant against the R one?
- What is the new limit for data and model size?

## 1.9 Specific Approaches in Apache Spark

The following is a brief comparision of the candidate approaches for my thesis.

## 1.10 Deciding on an approach

I have decided to implement the fitting of GAMs in Apache Spark. My choice will be explained now:

**Nested Data Parallel** Spark is fast Iterative nature is well suited for in-memory

## 1.11 What is novel in this approach?

The suggested approach has been implemented in high-level statistical languages, proving its effectiveness. The suggested methods have not been implemented in an enviroment designed for in-memory cluster computing. As I lined out earlier the iterative nature of the proposed methods is well suited for a distributed in-memory enviroment and hence should be discoved

## 1.12 When is the thesis regarded as completed?

The thesis is considered finished, once various strategies of implementing a chosen algorithm (e.g. have been implemented and compared in effectiveness/efficiency/human-workload.

## 1.13 Which tasks have to be completed for this thesis?

My work is defined by creating the progracomparing them. The focus is on showing the various transformations applied in NDP, which the programmer would have had done manually.

**Additional tasks**

- Read further papers on how NDP is implemented (especially. vectorization and fusioning)
- Read futher literature on how to make quantified comparisions of the programs (e.g. parallel complexity, benchmarks, etc...)
- Decide whether I will work with the true Haskell-Core code generated in or apply the flattening and fusioning transformations manually. The first variant is the actual source code that will later be executed - but it is hard to read. [1] The implementation of DPH is less developed than its theoretical background. The second variant is easier to work with and simpler to present, however it will almost certainly be less optimized than the actual code.
- Decide which algorithm(s) is/are to be used as examples. They should be easily visualizable and should have irregular behaviour. Connected-Components-Labeling on images is such an candidate. Combining multiple algorithms into a pipeline is also attractive, since cross-algorithm fusionsing and optimization is where Haskell can shine truely.
- Learn how make benchmarks/create runtime statistics

## 1.14 Which tasks are realistic to be accomplished within the scope of this thesis?

I am optimistic that these task can be accomplished in a timeframe of 2 months. However, I might need to keep a tight focus or choose a simple (and small) algorithm to keep the task accomplishable.

## 1.15 Time plan

My time plan is visible at table 1.

[1] See `DotP.hs` and `DotP.vect.hs` at my github repo.

Table 1: Time table

| CW | monday | thesis work |
|---|---|---|
| 17 | 20.04 | reading remaining papers, reading parallel complexity theory |
| 18 | 27.04 | deciding on an algorithm, learning benchmarking |
| 19 | 4.05 | impleme |
| 20 | 11.05 asd | |
| 21 | 18.05 | vectori |
| 22 | 25.05 asd | |
| 23 | 1.06 | *puffer* |
| 24 | 8.06 | *puffer* |
| 25 | 15.06 | Begin to write down, prepare for exams |
| 26 | 22.06 | Writing..., prepare for exams |
| 27 | 29.06 | Writing..., prepare for exams |
| 28 | 6.07 | Prepare Colloquium, Writing..., exams week 1 |
| 29 | 13.07 | Prepare Colloquium, Finalize writing, exams week 1 |
| 30 | 20.07 | Colloquium and Release |
| 31 | 27.07 | Last week for Colloquium and Release |
| 32 | 3.08 | Fin `:D` |

# 2 Draft - Contents

The following presents a draft structure of my thesis. Since the algorithm I am going to implement is not decided yet, I am using " " as its placeholder.

1. Introduction
   - 1.1. Context
   - 1.2. Goal
   - 1.3. Structure
2. Basics
   - 2.1. Parallel Computing and Complexity
   - 2.2. Haskell
   - 2.3. Nested Data Parallelism
     - i. Parallel Arrays
     - ii. Sparse Matrices - An Example
     - iii. Execution model
3. : Sequential
   - 3.1. Implementation
   - 3.2. Runtime analysis (e.g. sequential/parallel complexity)
   - 3.3. Benchmark
4. : Manually-parallel
   - 4.1. Implementation

4.2. Runtime analysis (e.g. sequential/parallel complexity)

4.3. Benchmark

5. : Nested-Data-Parallel

5.1. High-level Implementation

5.2. Transformations

i. Non-Parametric representation

ii. Lifting

iii. Vectorization

iv. Fusioning

5.3. Final low-level form

5.4. Benchmark

6. Evaluation

6.1. Sequential vs. Parallel

6.2. Manually-Parallel vs. Nested-Data-Parallel

7. Conclusion

7.1. Effectivenesss of Nested Data Parallel

7.2. Future work

i. Alternate Algorithms

ii. Best of Repa and NDP

iii. Distributed NDP

iv. NDP on GPUs

# 3 Personal notes

- Zeitplan für 2.5 Monate
- English als Ausarbeitungssprache