Master's Thesis in Statistics
Department of Statistics

Examensarbete i statistik för masterexamen,
Statistiska institutionen

Stochastic Gradient Descent for Efficient Logistic Regression

Alexander Thorleifsson

Examensarbete 30 högskolepoäng, vt 2016
Handledare (supervisor): Prof. Hans Nyquist
Abstract

The optimization algorithm Stochastic Gradient Descent (SGD) for efficient parameter estimation has proved to perform well in statistical challenges were data sets are very big. There have been numerous variants and improvements proposed since the original SGD algorithm was first proposed by Sakrison (1965) but Toulis’ et al. (2015) recent contribution Averaged Implicit SGD, or AISGD for short, performs especially well and can be implemented for Generalized Linear Models (GLM) with logistic regression as a special case. The logistic regression model is the standard workhorse in many industries and the major objective of this thesis is to investigate the efficiency of Stochastic Gradient Descent (SGD) when implemented for estimation of logistic regression models. Experiments showed that SGD algorithms can be considerably faster than the more traditional estimation method Iteratively Reweighted Least Squares (IRLS) when working with large data sets. Evaluations of convergence behaviour revealed the effects and sensitivity of learning rate specifications and potential pitfalls of automatic convergence thresholds. Complementing previous research on the stability of SGD algorithms, this thesis provides evidence that SGD is a serious contender to more commonly used estimation methods for logistic regression, especially when training time is the main concern.
Contents

Abstract .................................................. 2

1 Introduction ................................................. 4

2 Theory and literature review ................................. 6
  2.1 Statistical Learning .................................. 6
    2.1.1 Online Learning ................................ 6
  2.2 Logistic Regression .................................. 7
  2.3 Gradient Descent .................................... 7
  2.4 Stochastic Gradient Descent ......................... 8
  2.5 Recent improvements ................................. 10
  2.6 Learning Rates .................................... 11
  2.7 Iteratively Reweighted Least Squares ................. 12
  2.8 Cross Validation .................................. 13

3 Method ..................................................... 14
  3.1 Implementation ................................... 15
  3.2 Data ............................................... 15

4 Results ..................................................... 16
  4.1 Runtime ........................................... 16
  4.2 Explicit Stochastic Gradient Descent (ESGD) ........ 18
  4.3 Averaged Implicit Stochastic Gradient Descent (AISGD) 22

5 Discussion .................................................. 25

6 Conclusions ................................................ 27

A Example Code ............................................ 28
Chapter 1

Introduction

Statistical challenges in situations were data sets are very big, as well as in the online setting were new observations are added sequentially in real time, are becoming increasingly common in many modern industries. In these situations it becomes challenging to satisfy the ideal properties of computational efficiency, statistical optimality, and numerical stability with only one estimation method (Tran et al., 2015). Also, because data sizes have grown faster than computer processing speed in the last decade, the bottleneck is often computing time and not sample size. The optimization algorithm Stochastic Gradient Descent (SGD) for computationally efficient parameter estimation performs very well in this context (Bottou, 2012).

Widely used parameter estimation methods of Generalized Linear Models (GLMs) such as Iteratively Reweighted Least Squares (IRLS) must iterate over all observations in the data set to do one single update of the parameters. Also, IRLS is hard to implement in the online setting while SGD is well fit for this and also more computationally efficient since it only uses one observation in each iteration to update the parameters. Specifically, SGD algorithms only require $O(n)$ memory which is the minimum required for storing the $i^{th}$ iterate $\hat{\theta}_i$, where $O$ is a notation for memory requirement, see more details on page 11. IRLS on the other hand, when implemented with the standard function \texttt{glm} in R, requires roughly $O(mn^2)$ of memory and when implemented with \texttt{biglm}, designed to work especially well with big data sets, require $O(n^2)$. Theoretically, SGD algorithms are more efficient since they replace the inversion of $n \times n$ matrices, where $n$ is the number of parameters, in IRLS with a scalar sequence $\alpha_i$ and a matrix $C_i$ that are faster to manipulate, by design (Tran et. al, 2015).

Although the theoretical foundations of stochastic gradient descent dates back to the 1960s, the development in recent years has been led by a limited set of authors, such as Leon Bottou, Panos Toulis, and Edoardo Airoldi. These authors among others have proven the applicability of SGD to a number of different models. The original SGD algorithm was first proposed by Sakrison (1965) and is typically referred to as explicit SGD, which I will refer to as ESGD in this thesis. There have been numerous variants and
improvements proposed since then but Toulis et. al (2015) recent contribution of what they call implicit SGD with averaging, or AISGD for short, performs especially well. Both ESGD and AISGD can be applied to logistic regression and GLMs in general. However, a more in-depth focus on the efficiency of these algorithms applied to binary logistic regression has been lacking. The logistic regression model is the standard workhorse in many industries, applied in many different contexts such as credit risk modeling and strategic marketing. In the cases where SGD has been studied and applied in the context of GLMs the focus has either been on the standard normal linear model or narrowly focused on machine learning problems, such as the classification of multinomial text or digital image data.

The major objective of this thesis was therefore to investigate the efficiency of Stochastic Gradient Descent (SGD) when implemented for estimation of logistic regression models. The efficiency was measured by the convergence runtime and further complemented by plotting the convergence behaviour. Specifically, inspired by the methodology used in leading works such as Tran et al. (2015) and Bottou (2010), the computational efficiency of ESGD and AISGD was compared to the more traditional estimation method IRLS, implemented both through the standard function \texttt{glm} and the popular alternative \texttt{biglm}. The convergence behaviour of SGD and AISGD was later examined individually. In particular, a validation set approach of cross validation was used to study the convergence of the algorithms both in terms of runtime and number of iterations needed for convergence. Different specifications of learning rates were also compared. The experiments were carried out on simulated, normally distributed, data of different sizes (see section 3.2).

The structure of this thesis is as follows. In chapter 2 I present a short background of statistical learning, the theoretical basis of Gradient Descent (GD) and SGD applied to logistic regression, learning rates, IRLS, and cross validation. In chapter 3 the methodology and the practical implementation is explained. The results of the experiments are presented in chapter 4. Comments, interpretations, and explanations of the results are presented in chapter 5 and conclusions in chapter 6.
Chapter 2

Theory and literature review

2.1 Statistical Learning

Statistical learning is a recently emerged subfield of statistics and is tightly connected and influenced by machine learning, a subfield of computer science. It refers to a set of computationally heavy tools for understanding and modeling complex data sets. The field include a wide range of methods, including support vector machines, classification and regression trees, but also popular algorithms for fitting such models like stochastic gradient descent (SGD). Although the term statistical learning is quite new, many methods that are fundamental to the field were developed a long time ago.

The method of least squares was first formalized in the early nineteenth century by Adrien-Marie Legendre and Carl Friedrich Gauss and first successfully applied in the field of astronomy. Since linear regression is only used to predict quantitative values an alternative method for predicting categorical values was developed by Fisher (1936) called Linear Discriminant Analysis (LDA). This classification technique was further refined when various authors presented logistic regression in the 1940s. A flexible generalization of linear regression was presented in the 1970s by John Nelder and Robert Wedderburn, the generalized linear model (GLM), that included logistic regression as a special case (Nelder and Wedderburn, 1972).

Since then statistical learning has emerged as a subfield of statistics thanks in part to the increasing availability and use of powerful statistical software, such as the R programming language. This development has in recent years made an increasing impact on a much broader community, outside of the original confines of statistics and computer science (Hastie et al., 2013).

2.1.1 Online Learning

Statistical learning algorithms that receive their input in a sequential order, and are used to update parameter estimates at each step, are said to operate in an online setting. This can be contrasted with the more traditional approach of fitting the model on the entire data set at once. Online
learning techniques are applicable in many real world problems that involve streaming data and in situations where it is computationally inefficient to update a model on the entire data set. These types of data sources are dynamic and generated in real time, at high speed. Such situations can be found in sensor applications, traffic management, log records, email, news feeds, call-detail records, etc (Read et al., 2012).

2.2 Logistic Regression

In a logistic regression model, the response variable $y_i$ is categorical. The focus in this thesis will be on binary responses, even though it can be generalized to more than two categories. Therefore, $y_i \in \{0, 1\}$, where 0 is usually denoted as the negative class (the absence of something) and 1 the positive class (the presence of something). The logistic function $h(z) = 1/(1 + e^{-z})$ can, when $z = \theta^T x$, be expressed as

$$h(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}} \quad (2.1)$$

Where $\theta$ is the parameter vector and $h(z)$ is always bounded between 0 and 1. As $z \to \infty$ we get that $h(z) \to 1$ and as $z \to -\infty$ we get that $h(z) \to 0$. In this context $h(\theta^T x)$ can also be called the hypothesis function and the output of this hypothesis is the estimated probability that $y = 1$ given input $x$, i.e. $h(\theta^T x) = P(y = 1|x; \theta)$ (Ng, 2012).

2.3 Gradient Descent

Given a data set of $i = 1, 2, ..., m$ observations and $j = 1, 2, ..., n$ variables $D = \{x_{ij}, y_i\}$ where $x_{i,0} = 1$ and $y_i \in \{0, 1\}$ we can define the cost function in the following way:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \text{Cost}(h(\theta^T x_i), y_i) \quad (2.2)$$

The function $\text{Cost}(h(\theta^T x_i), y_i)$ is the cost we want the algorithm to pay when the outcome is $h(\theta^T x_i)$, given that the actual outcome is $y_i$. The cost function for the $i^{th}$ observation in logistic regression is

$$\text{Cost}(h(\theta^T x_i), y_i) = \begin{cases} -\log(h(\theta^T x_i)) & \text{if } y = 1 \\ -\log(1 - h(\theta^T x_i)) & \text{if } y = 0 \end{cases} \quad (2.3)$$

which can be compressed to a more compact equation:

$$\text{Cost}(h(\theta^T x_i), y_i) = -y_i \log(h(\theta^T x_i)) - (1 - y_i) \log(1 - h(\theta^T x_i)) \quad (2.4)$$
since when \( y_i = 1 \) we are left with \(-\log(h(\theta^T x_i))\) and when \( y_i = 0 \) we are left with \(-\log(1 - h(\theta^T x_i))\).

In summary, we can write the logistic regression cost function as:

\[
J(\theta) = -\frac{1}{m} \left[ \sum_{i=1}^{m} y_i \log h(\theta^T x_i) + (1 - y_i) \log(1 - h(\theta^T x_i)) \right]
\]

which can be minimized, i.e. \( \min_{\theta} J(\theta) \), to estimate the true parameter value \( \theta \). This can be achieved using Gradient Descent that repeatedly updates the parameters with the help of a learning rate \( \alpha \) and an initial guess \( \theta_0 \):

Repeat \{ 
\[
\hat{\theta} := \hat{\theta} - \alpha \sum_{i=1}^{m} (h(\theta^T x_i) - y_i) x_i
\]
\}

The notation "\( a := b \)" is used to denote an operation in a computer program in which the value of a variable \( a \) is set to be equal to the value of \( b \). Therefore, this operation overwrites \( a \) with the value of \( b \), i.e. it is not "\( a = b \)" where \( a \) is equal to \( b \) (Ng, 2012). The learning rate \( \alpha \) is a number that controls how big steps will be taken by the algorithm in each step. The bigger the learning rate the more "aggressive" is the algorithm (Ng, 2016a).

### 2.4 Stochastic Gradient Descent

Gradient descent, as well as more traditional optimization methods for parameter estimation like iteratively reweighted least squares are theoretically less efficient the bigger the data sets. Of course, in many cases a random sample will suffice and therefore not require faster estimation methods. But when sampling is not feasible or less competitive, faster methods like Stochastic Gradient Descent (SGD) can be powerful since it only requires a single observation to be stored in memory. For example, imagine a data set of 1 million observations. The gradient descent algorithm will have to sum over all 1 million observations in each step. In SGD however, parameter estimates are incrementally updated with only one observation at a time and is suitable for estimation in the online learning setting, in which new observations are added continuously, as well as in big data sets where traditional estimation techniques are computationally impractical. SGD is a class of optimization algorithms that was first proposed by Sakrison (1965) and is also referred to as stochastic approximation algorithms (Kushner & Yin, 1997).

The new cost function which represents the cost of \( \theta \) for a specific observation \((x_i, y_i)\) in the SGD setting becomes
\[ J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \text{Cost}(\theta, (x_i, y_i)) \]  

(2.7)

We get the SGD algorithm by modifying (2.6) to only take into account one single observation at a time. However, instead of minimizing the cost function \( J(\theta) \) we can maximize the log likelihood \( \ell(\theta) \) for the same result. The derivative of the logistic function is

\[ h'(z) = h(z)(1 - h(z)) \]  

(2.8)

Remember that

\[
P(y = 1|x; \theta) = h(\theta^T x)
\]

(2.9)

\[
P(y = 0|x; \theta) = 1 - h(\theta^T x)
\]

Which can be written more compactly as

\[
P(y|x; \theta) = (h(\theta^T x))^y(1 - h(\theta^T x))^{1-y}
\]  

(2.10)

Assuming that \( m \) observations were generated i.i.d, we can write the log-likelihood function as

\[
\ell(\theta) = \log L(\theta) = \sum_{i=1}^{m} y_i \log h(\theta^T x_i) + (1 - y_i) \log(1 - h(\theta^T x_i))
\]  

(2.11)

If we focus on only one observation \((x, y)\), we can derive the stochastic gradient descent rule:

\[
\frac{\partial}{\partial \theta_j} \ell(\theta) = \left( y \frac{1}{h(\theta^T x)} - (1 - y) \frac{1}{1 - h(\theta^T x)} \right) \frac{\partial}{\partial \theta_j} h(\theta^T x)
\]

\[
= \left( y \frac{1}{h(\theta^T x)} - (1 - y) \frac{1}{1 - h(\theta^T x)} \right) h(\theta^T x)(1 - h(\theta^T x)) \frac{\partial}{\partial \theta_j} \theta^T x
\]

\[
= (y(1 - h(\theta^T x)) - (1 - y)h(\theta^T x)) x_j
\]

\[
= (y - h(\theta^T x)) x_j
\]  

(2.12)

where the fact that \( h'(z) = h(z)(1 - h(z)) \) was used and subscript \( i \) was eliminated for ease of notation. The SGD algorithm therefore becomes

1. Randomly shuffle data set
2. Repeat

\[
\{ \text{for } \; i := 1, 2, \ldots, m \{ \hat{\theta} := \hat{\theta} + \alpha(y_i - h(\theta^T x_i)) x_i \} \}
\]  

(2.13)
in conformity with (Ng, 2016b). Since (2.13) aims to maximize the log-likelihood it is actually an ascent algorithm, sometimes referred to Stochastic Gradient Ascent. However, I will refer to it as SGD since leading works do the same in order to keep in line with the relevant optimization literature, see for example Toulis & Airoldi (2015, p. 782).

2.5 Recent improvements

Toulis & Airoldi (2015) develop and improve on existing stochastic approximation methods. They present the limitations of standard stochastic approximation procedures, of which numerical instability and statistical inefficiency are the primary issues. They develop theory and implementations for generalized linear models. Their results suggest that ”stochastic gradient methods are poised to become benchmark principled estimation procedures for large data sets, especially those in the family of stable proximal methods, such as implicit stochastic gradient descent.” (Toulis & Airoldi 2015, p. 1). In particular, they introduce what they call implicit SGD with averaging, or AISGD, which is an improvement to the more traditional Explicit SGD, or ESGD for short. ESGD can then be written as

$$\hat{\theta}_i = \hat{\theta}_{i-1} + \alpha_i C_i (y_i - h(x_i^T \hat{\theta}_{i-1}))x_i$$ (2.14)

while AISGD is defined by the following procedure

$$\hat{\theta}_i = \hat{\theta}_{i-1} + \alpha_i C_i (y_i - h(x_i^T \hat{\theta}_i))x_i$$

$$\bar{\theta}_i = \frac{1}{i} \sum_{k=1}^{i} \hat{\theta}_k$$ (2.15)

where $\alpha_i$ is the learning rate sequence and can be adaptive, $C_i$ is used to better condition the iteration and is a sequence of positive-definite matrices and is equal to the identity matrix in the simplest case, i.e. $C_i = I$, for more detail see 2.6. The averaging of the iterates, $\bar{\theta}_i$, aims to achieve statistical optimality of the algorithm, in that it is an optimal unbiased estimator of the true parameter value. For more details of the properties of ESGD and AISGD and algorithmic representation see Toulis, Tran, and Airoldi (2015).

Since the true parameter vector is a constant, $E(\nabla_\theta \ell(\theta)) = 0$ where $\nabla_\theta \ell(\theta)$ is the vector of partial derivatives of $\ell(\theta)$ with respect to the true parameter vector $\theta$. SGD is furthermore justified since $\hat{\theta}_i$ converges to a point $\theta_\infty$ such that $E(\nabla_\theta \ell(\theta_\infty)) = 0$ (online setting) and $\hat{\theta}_i$ approximates the maximum likelihood estimator $\theta_{mle}$ (finite setting, i.e. when $m$ is finite), proved by the theory of stochastic approximations (Robbins and Monro, 1951; Tran et. al 2015, p. 2). Therefore, $\theta_\infty = \theta$, i.e. SGD converges to the true parameter value.

ESGD and AISGD are computationally optimal in the sense that they only
require $O(n)$ memory, which is the minimum required for storing the $i$th iterate $\hat{\theta}_i$ (Tran et. al, 2015). The notation $O$, or "big O", is frequently used in computer science and complexity theory to classify the response of algorithms to changes in input size. It denotes how much space is required (i.e. memory, but can also denote time) to solve a problem as the input grows (Mohr, 2014). For example, an algorithm that uses $O(1)$ memory will always execute in the same space and the amount of input is inconsequential. The memory usage is constant. $O(n)$ on the other hand describes an algorithm whose memory usage grows linearly with the size of the input $n$. Therefore, the SGD algorithms with memory requirements of $O(n)$ are examples of this linearity where the memory usage grows linearly with the input, in this case the number of parameters $n$. Correspondingly, an algorithm that uses $O(n^2)$ memory will execute proportional to the square of the input size. This notation will become relevant in the next chapter when we consider iteratively reweighted least squares.

### 2.6 Learning Rates

The learning rate, also known as step size, is important for the algorithm to converge. For sufficiently small learning rate $\alpha$, the cost function $J(\theta)$ should decrease on every iteration. The disadvantage of setting a learning rate that is too small is that the algorithm can be slow to converge. If on the other hand the learning rate is too large, the algorithm may instead diverge (Ng, 2016c).

However, the learning rate does not have to be held constant but can instead decrease over time for better performance. Many types of these improved adaptive learning rates, or learning rate sequences, have been proposed. In this thesis, the focus will be on three of the most prominent adaptive learning sequences, which are all featured in sgd. These are:

- **One-dimensional** (Xu, 2011): This is the default learning rate in the sgd package, and is of the form
  \[ \alpha_i = \alpha_0(1 + b\alpha_0 i)^{-c} \]  
  where $\alpha_0, b, c \in \mathbb{R}$ are fixed constants. As written above, $\alpha_0$ is the initial learning rate. As per default, $\alpha_0 = 1, b = 1, c = 1$ if implemented without averaging (explicit SGD) and $c = 2/3$ if implemented with averaging (AISGD).

- **AdaGrad** (Duchi et al. 2011) short for adaptive gradient algorithm, employs a diagonal conditioning matrix $C_i$ instead of a one-dimensional learning rate of the form
  \[ C_i = \eta(I_i + \epsilon I)^{-1/2} \]  

11
where \( \eta \in \mathbb{R} \) is a constant, \( \epsilon \) is a fixed value typically \( 10^{-6} \) to prevent division by zero, and \( I \) the identity matrix. Furthermore,

\[
I_i = I_{i-1} + \text{diag}(\nabla \ell(\theta_{i-1}; x_i, y_i) \nabla \ell(\theta_{i-1}; x_i, y_i)^T)
\]  

where \( \nabla \) is the gradient vector and \( \ell(\theta_{i-1}; x_i, y_i) = \sum_{i=1}^{N} \log f(y_i; x_i, \theta) \) the log-likelihood function for the whole data set \( D \).

- Rmsprop (Tieleman and Hinton 2012). Like AdaGrad, Rmsprop features the conditioning matrix

\[
C_i = \eta(I_i + \epsilon I)^{-1/2}
\]

but with a slightly modified matrix \( I_i \)

\[
I_i = \beta I_{i-1} + (1 - \beta)\text{diag}(\nabla \ell(\theta_{i-1}; x_i, y_i) \nabla \ell(\theta_{i-1}; x_i, y_i)^T)
\]

where \( \beta \in [0, 1] \) is the discount factor. Rmsprop takes a weighted average of previous and new information, and by doing this aims to "offset one problem AdaGrad often encounters in practice, where very large values occur for initial estimates of \( I_i \) (e.g., due to poor initialization), thus slowing down the AdaGrad procedure as it tries to accumulate enough curvature information to compensate for such an error" (Tran et al. 2015, p. 17).

### 2.7 Iteratively Reweighted Least Squares

To contrast SGD with the more traditional Iteratively Reweighted Least Squares (IRLS), let us first examine the Newton-Raphson method, where a single update is given by

\[
\theta_i = \theta_{i-1} - \left( \frac{\partial^2 \ell(\theta)}{\partial \theta \partial \theta^T} \right)^{-1} \frac{\partial \ell(\theta)}{\partial \theta}
\]

where the derivatives are evaluated at \( \theta_{i-1} \) and the inverted matrix is the Hessian:

\[
H = \frac{\partial^2 \ell(\theta)}{\partial \theta \partial \theta^T}
\]

If we let \( y \) denote the vector of \( y_i \) values, \( X \) the \( m \times (n+1) \) matrix of \( x_i \) values, \( p \) the vector of fitted probabilities with \( i \)th element \( h(x_i; \theta_{i-1}) \), and \( W \) an \( m \times m \) diagonal matrix of weights where the \( i \)th diagonal element is \( h(x_i; \theta_{i-1})(1 - h(x_i; \theta_{i-1})) \). Then we have

\[
\frac{\partial \ell(\theta)}{\partial \theta} = X^T (y - p)
\]

\[
\frac{\partial^2 \ell(\theta)}{\partial \theta \partial \theta^T} = -X^T W X
\]
Thus, the Newton step is
\[
\theta_i = \theta_{i-1} + (X^TWX)^{-1}X^T(y - p)
= (X^TWX)^{-1}X^T(W(X\theta_{i-1} + W^{-1}(y - p))
= (X^TWX)^{-1}X^TWz.
\] (2.25)

where
\[
z = X\theta_{i-1} + W^{-1}(y - p)
\] (2.26)

where \(z\) is called the \textit{adjusted response} and \(X\) the input matrix. This algorithm is called \textit{Iteratively Reweighted Least Squares} (IRLS) because at each iteration we solve the weighted least square problem:
\[
\theta_i := \arg \min_{\theta} (z - X\theta)^T W(z - X\theta).
\] (2.27)

More information can be found in Friedman et. al (2001, p. 120). Toulis & Airoldi (2014) have shown that IRLS require roughly \(O(mn^2)\) of memory. IRLS is therefore theoretically suboptimal to SGD from a computational efficiency standpoint. Theoretically, SGD algorithms are more efficient since it replaces the expensive inversion of \(n \times n\) matrices, as in IRLS, with a scalar sequence \(\alpha_i\) and a matrix \(C_i\). Also, the log-likelihood is evaluated at a single observation \((x_i, y_i)\) in SGD instead of the entire data set \(D\), which saves significant computation time (Tran et. al 2015, p. 2).

## 2.8 Cross Validation

Cross validation is a model validation method that measures how well a fitted model generalizes to an independent data set. By “holding out” a subset of the original data set, one can fit the model on the rest of the data and later test the predictive accuracy of that model on the held out data.

In this thesis, the validation set approach of cross validation will be used. First, we divide the available data set randomly into two parts, a training set and a test set. We fit the model on the training set and with this fitted model we can estimate how accurately it performs in practice by measuring how well it performs on the test set. The resulting misclassification rate, the number of misclassified predictions divided by the total number of predictions, gives an indication of how well the model performs and gives insight into how well the model generalizes.
Chapter 3

Method

The major objective of this thesis is to investigate the efficiency of Stochastic Gradient Descent (SGD) when implemented for estimation of logistic regression models. The efficiency is measured by the convergence runtime and further complemented by plotting the convergence behaviour.

Specifically, the runtime of ESGD and the newly improved AISGD is compared to the more traditional estimation method Iteratively Reweighted Least Squares (IRLS) that is used as default in the standard package for logistic regression models in R, \texttt{glm}. In addition to \texttt{glm}, the \texttt{biglm} package, that also uses IRLS but is designed to work especially well with large data sets, is included in the comparison since it is a popular alternative to \texttt{glm} when working with large data sets. The convergence behaviour of ESGD and AISGD is examined individually. In particular, a validation set approach of cross validation is used to study the convergence of the algorithms both in terms of runtime and number of iterations needed for convergence. This is presented in plots that indicate the misclassification rate on the validation set. This method is motivated by the fact that it is a common benchmark in leading works (see for instance Tran et. al (2015) and Bottou (2010)). Also, Bottou (2012) has proclaimed that it is important "to periodically evaluate the validation error during training because we can stop training when we observe that the validation error has not improved in a long time" (Bottou 2012, p. 8). In this thesis I use the term "misclassification rate" which is equivalent to validation error. The evaluation of the misclassification rate is therefore not primarily used for comparing the predictive accuracy of the algorithms, but rather for exploring the convergence behaviour since the efficiency is the main focus of this thesis. For example, if an algorithm takes 10 seconds to converge but the classification plot reveals that for the last 5 of those 10 seconds the algorithm improved very slowly (but above the threshold) one could conclude that the algorithm should stop after 5 seconds if the main concern is training time. Different specifications of learning rates, described in section 2.6, is also compared since the learning rate sequence has an effect on the resulting efficiency of the algorithm.

For the runtime comparisons, the R build-in function \texttt{system.time} is used.
This function lets users assess the amount of time a function takes to run. The function gives three values titled "user" which is the execution time of the code, "system" which gives the CPU time, and "elapsed" which is the total time from initialization of the code to the completion of the algorithm. The latter, "elapsed", is used for all timings in this thesis. Since the function \texttt{system.time} is not always precise it has been recommended (Wickham, 2014) to repeat each operation multiple times with a loop, and then divide to get the average time of each operation. In all comparisons of computational efficiency in this thesis, the operations is averaged over 10 runs.

### 3.1 Implementation

All the experiments were performed on a 2.5 GHz Intel Core i7 machine with 16 GB of memory. The R-packages that were used are the following:

- \texttt{sgd} (Tran et. al, 2015). The newly released and most robust implementation of SGD in R to date. It features both ESGD and AISGD, ready to be applied for estimation of logistic regression models among others, and is publicly available on CRAN.\footnote{Available at https://cran.r-project.org/web/packages/sgd/index.html} The \texttt{sgd$converged} is used to judge if an algorithm has converged or not.

- \texttt{glm}. The built-in function for fitting generalized linear models in R. It performs maximum-likelihood estimation through iteratively reweighted least squares.

- \texttt{biglm} (Lumley, 2013). Popular alternative to \texttt{glm} when fitting GLMs to large data sets. It processes the data in "chunks" by splitting the data in many parts and then updates the parameters using incremental QR decomposition.

### 3.2 Data

The experiments are carried out on simulated, normally distributed, data of different sizes, both in \( n \) and \( m \). The response variable \( Y \sim Bin(N,p) \) where \( p = (1/(1 + e^{-(X^T \theta)}), X \sim N(0, 1) \) i.i.d., and \( \theta = (5, ..., 5) \). The R code for this simulation is inspired by Tran (2015). A held out set of 1/8 of the total data set is used for validation and correspondingly 7/8 is used for model fitting. In chapter 4.1 on computational efficiency comparisons, nine dimensions of data sizes were chosen were \( 10,000 \leq m \leq 1,000,000 \) and \( 50 \leq n \leq 200 \). In chapters 4.2 and 4.3 the number of observations are \( m = 100,000 \) while the number of parameters \( n = \{10, 50\} \).
Chapter 4

Results

4.1 Runtime

Both packages glm and bigglm use IRLS for parameter estimation. However, bigglm is designed to work especially well with big data sets. It processes the data in one "chunk" at a time and continues until the whole dataset is processed, and requires only $O(n^2)$ memory for $n$ variables at any given time (Lumley, 2013). In contrast, the memory requirements of IRLS in glm are roughly $O(mn^2)$ (Toulis & Airoldi, 2014).

ESGD and AISGD on the other hand are computationally optimal in the sense that they only require $O(n)$ memory, which is the minimum required for storing the $i^{th}$ iterate $\hat{\theta}_i$ (Tran et. al, 2015).

All SGD algorithms in table 4.1 - 4.3 were declared to have converged if nothing else is stated. The results for the comparison with IRLS are shown in table 4.1. The SGD algorithms are faster than glm on all levels, and starts to become superior right away. Bigglm however is faster than ESGD on the smallest data set but is outcompeted for the larger ones. Interestingly, ESGD was slightly faster that AISGD on most levels.

The results for the comparison between different specifications of learning rates in the SGD algorithms can be seen in tables 4.2 and 4.3. Apparently, the influence of the different learning rates on ESGD are negligible. However, for AISGD, the one dimensional learning rate lead to the fastest convergence on all dimensions of data while AdaGrad come in second, and Rmsprop is the slowest.

Automatic convergence thresholds are used in which convergence is declared for the SGD algorithms, i.e. the algorithms stop, if they are unable to change the relative mean squared difference in the parameters by more than $10^{-5}$ in one iteration.
Table 4.1: Timing in seconds for fitting logistic regression models to different dimensions of simulated data (averaged over 10 runs). Comparison between SGD- (with the one-dimensional learning rate) and IRLS algorithms.

<table>
<thead>
<tr>
<th>Observations</th>
<th>Parameters</th>
<th>AISGD</th>
<th>ESGD</th>
<th>glm</th>
<th>biglm</th>
</tr>
</thead>
<tbody>
<tr>
<td>10,000</td>
<td>50</td>
<td>0.122</td>
<td>0.152</td>
<td>0.574</td>
<td>0.123</td>
</tr>
<tr>
<td>10,000</td>
<td>100</td>
<td>0.203</td>
<td>0.191</td>
<td>1.688</td>
<td>0.284</td>
</tr>
<tr>
<td>10,000</td>
<td>200</td>
<td>0.350</td>
<td>0.360</td>
<td>9.739</td>
<td>0.756</td>
</tr>
<tr>
<td>100,000</td>
<td>50</td>
<td>0.372</td>
<td>0.289</td>
<td>5.580</td>
<td>1.116</td>
</tr>
<tr>
<td>100,000</td>
<td>100</td>
<td>0.698</td>
<td>0.507</td>
<td>17.089</td>
<td>2.707</td>
</tr>
<tr>
<td>100,000</td>
<td>200</td>
<td>1.276</td>
<td>0.970</td>
<td>58.21</td>
<td>7.392</td>
</tr>
<tr>
<td>1,000,000</td>
<td>50</td>
<td>3.222</td>
<td>3.053</td>
<td>70.81</td>
<td>12.01</td>
</tr>
<tr>
<td>1,000,000</td>
<td>100</td>
<td>5.459</td>
<td>5.344</td>
<td>225.89</td>
<td>27.49</td>
</tr>
<tr>
<td>1,000,000</td>
<td>200</td>
<td>10.339</td>
<td>10.313</td>
<td>765.68</td>
<td>84.97</td>
</tr>
</tbody>
</table>

Table 4.2: Timing in seconds for fitting AISGD to different dimensions of simulated data (averaged over 10 runs). Comparison between three specifications of learning rates.

<table>
<thead>
<tr>
<th>Observations</th>
<th>Parameters</th>
<th>One-Dim</th>
<th>AdaGrad</th>
<th>Rmsprop</th>
</tr>
</thead>
<tbody>
<tr>
<td>10,000</td>
<td>50</td>
<td>0.122</td>
<td>0.199</td>
<td>0.209</td>
</tr>
<tr>
<td>10,000</td>
<td>100</td>
<td>0.203</td>
<td>0.324</td>
<td>0.329</td>
</tr>
<tr>
<td>10,000</td>
<td>200</td>
<td>0.350</td>
<td>0.597</td>
<td>0.614</td>
</tr>
<tr>
<td>100,000</td>
<td>50</td>
<td>0.372</td>
<td>0.446</td>
<td>0.595</td>
</tr>
<tr>
<td>100,000</td>
<td>100</td>
<td>0.698</td>
<td>0.781</td>
<td>1.027</td>
</tr>
<tr>
<td>100,000</td>
<td>200</td>
<td>1.276</td>
<td>1.503</td>
<td>1.898</td>
</tr>
<tr>
<td>1,000,000</td>
<td>50</td>
<td>3.222</td>
<td>3.277</td>
<td>3.466</td>
</tr>
<tr>
<td>1,000,000</td>
<td>100</td>
<td>5.459</td>
<td>5.620</td>
<td>5.833</td>
</tr>
<tr>
<td>1,000,000</td>
<td>200</td>
<td>10.339</td>
<td>10.479</td>
<td>11.090</td>
</tr>
</tbody>
</table>

Table 4.3: Timing in seconds for fitting ESGD to different dimensions of simulated data (averaged over 10 runs). Comparison between three specifications of learning rates.

<table>
<thead>
<tr>
<th>Observations</th>
<th>Parameters</th>
<th>One-Dim</th>
<th>AdaGrad</th>
<th>Rmsprop</th>
</tr>
</thead>
<tbody>
<tr>
<td>10,000</td>
<td>50</td>
<td>0.152</td>
<td>0.198</td>
<td>0.209</td>
</tr>
<tr>
<td>10,000</td>
<td>100</td>
<td>0.191</td>
<td>0.337</td>
<td>0.345</td>
</tr>
<tr>
<td>10,000</td>
<td>200</td>
<td>0.360</td>
<td>0.616</td>
<td>0.631</td>
</tr>
<tr>
<td>100,000</td>
<td>50</td>
<td>0.289</td>
<td>0.297</td>
<td>0.319</td>
</tr>
<tr>
<td>100,000</td>
<td>100</td>
<td>0.507</td>
<td>0.509</td>
<td>0.525</td>
</tr>
<tr>
<td>100,000</td>
<td>200</td>
<td>0.970</td>
<td>0.970</td>
<td>0.983</td>
</tr>
<tr>
<td>1,000,000</td>
<td>50</td>
<td>3.053</td>
<td>3.096</td>
<td>3.110</td>
</tr>
<tr>
<td>1,000,000</td>
<td>100</td>
<td>5.344</td>
<td>5.325</td>
<td>5.327</td>
</tr>
<tr>
<td>1,000,000</td>
<td>200</td>
<td>10.313</td>
<td>10.637</td>
<td>10.479</td>
</tr>
</tbody>
</table>
4.2 Explicit Stochastic Gradient Descent (ESGD)

Each figure 4.1 - 4.5 illustrates the misclassification rate on the validation set. The x-axis of the figures are either the log-iteration or the runtime (in seconds). I.e., for log-iteration, each iteration from the training set is applied to the validation set. Correspondingly, for runtime, the misclassification rate on the validation set is displayed over the time in seconds it took to reach them.

As seen in figure 4.1 the AdaGrad learning rate sequence enjoys the fastest convergence in terms of iterations, almost minimizing the classification error after only 1000 iterations. However, the one-dimensional learning rate is more stable and smoother than Adagrad and especially Rmsprop. Notice also how Rmsprop oscillates greatly around the minimum, never reaching meaningful convergence.

Figure 4.2 shows ESGD with Adagrad and figure 4.3 with the one-dimensional learning rate. ESGD follows theory in that the more sparse model with \( n = 10 \) converges faster. Interestingly, in (4.3) both models do not seem to converge. The larger model \( (n = 50) \) for example keeps improving even after a runtime of 0.3 seconds. This is noteworthy since in table 4.1 the same model was judged to have converged after 0.289 seconds.
Figure 4.1: The misclassification rate on the validation set over the log-iteration of the ESGD algorithm. Comparison of different specifications of learning rates. Here, $m = 100,000$ and $n = 10$. 
Figure 4.2: The misclassification rate on the validation set over the log-iteration and runtime of the ESGD algorithm (Adagrad), respectively. Comparison of different sizes of data, \( n = \{10, 50\} \) and \( m = 100,000 \).
Figure 4.3: The misclassification rate on the validation set over the log-iteration and runtime of the ESGD algorithm (one-dim), respectively. Comparison of different sizes of data, $n = \{10, 50\}$ and $m = 100,000$. 

21
4.3 Averaged Implicit Stochastic Gradient Descent (AISGD)

Figure 4.4 gives the behavior of the different learning rate sequences for AISGD. Compared to the same for ESGD, figure 4.1, AISGD is much more stable and smoother for all learning rates. This shows the power of AISGD, in that it is very robust to different specifications of learning rates and therefore needs less tuning of the initial learning rate $\alpha_0$ and the conditioning matrix $C_i$. Interesting to note is that the algorithm converges roughly around 1000 iterations for all learning rates, in contrast to the convergence of ESGD that differed wildly between the different specifications and was much more sensitive to the choice of learning rate sequences.

Notice in figure 4.5 that AISGD converges after slightly more than 1000 iterations for when $n = 10$ and after roughly 10,000 for when $n = 50$. The runtime for both are less than 0.1 seconds. This can be contrasted to ESGD in figure 4.3 that after 100,000 iterations and 0.3 seconds still had not reached convergence. This is remarkable since the runtime tests shown in table 4.1 resulted in faster convergence for the ESGD algorithm of that size.
Figure 4.4: The misclassification rate on the validation set over the log-iteration of the AISGD algorithm. Comparison of different specifications of learning rates. Here, $m = 100,000$ and $n = 10$. 
Figure 4.5: The misclassification rate on the validation set over the log-iteration and runtime of the AISGD algorithm (one-dim), respectively. Comparison of different sizes of data, $n = \{10, 50\}$ and $m = 100,000$. 
Chapter 5
Discussion

The runtime tests in table 4.1 resulted in better performance for the ESGD algorithm on almost all data sizes, albeit with a small amount. Remember, the algorithms were all judged to have converged with the help of the sgd$converged option. With this option, the algorithms stopped and convergence was declared if they were unable to change the relative mean squared difference in the parameters by more than $10^{-5}$ in one iteration. However, figure 4.3 indicates that the ESGD (one-dim) algorithm did not converge. The likely reason for this was that the automatic threshold was in fact met somewhere along the way without actually having converged to the true parameter value. The remedy for this would be to set the automatic threshold lower and try again. In fact, when trying a threshold of $10^{-7}$ the algorithm continued to improve after the initial point of declared convergence when the threshold was $10^{-5}$. These results illustrate the importance of plotting the convergence behaviour of the SGD algorithms, and especially for ESGD. This is also supported by Ng, A (2016c) who recommends plotting the convergence behaviour and not only rely on automatic thresholds. Moreover, the results showed that when considering the ESGD algorithm, choosing Adagrad over the one-dimensional learning rate is preferred.

Remember the theoretical memory requirements of the algorithms; the SGD algorithms requires $O(n)$, while the IRLS algorithms require $O(n^2)$ in the case of biglm and $O(mn^2)$ in the case of glm. Although the memory requirement of the algorithms is not directly equivalent to the speed of estimating the parameters, we saw in section 4.1 that it certainly was a good predictor. The results in table 4.1 followed theory in that SGD was the fastest, biglm came in second, and glm was the slowest. The differences were most pronounced in the largest data set of $m = 10^6$ and $n = 200$. Since SGD memory grows linearly with $n$ while IRLS grows quadratically, it is in accordance with theory that the bigger the data set the more advantageous it is to take advantage of SGD. Specifically, the difference in performance as seen in 4.1 is more dramatic for large $n$. Since the memory requirement for glm is $O(mn^2)$, i.e. quadratic in $n$ but linear in $m$, these results are in accordance with theory. For example, consider that for $m = 10^6$ the AIGSD algorithm was 21.5 times faster than glm when $n = 50$. However, when $n$
grows to 100 AISGD is 37 times faster and for $n = 200$ it is 73 times faster.
Chapter 6

Conclusions

The experiments showed that SGD algorithms can fit logistic regression models considerably faster than IRLS when working with large data sets. For example, the SGD algorithms took roughly 10 seconds to converge with a data set of 1 million observations and 200 parameters while it took 766 and 85 seconds for IRLS when implemented with the \texttt{glm} and \texttt{biglm} packages, respectively. ESGD and AISGD performed similarly well when implemented with automatic thresholds but with a slight ESGD dominance. When comparing different learning rates, AISGD performed the best with one-dimensional while the influence of the different learning rates on ESGD was negligible. However, evaluations of convergence behaviour revealed Adagrad to be superior when considering ESGD. AISGD was revealed to be less sensitive to the choice of learning rate.
Appendix A

Example Code

```r
library(ggplot2)
library(sgd)
library(Rmisc)
library(biglm)

### Runtime comparison ###

# generate data
N <- 1e5
d <- 50
set.seed(42)
X <- matrix(rnorm(N*d), ncol = d)
theta <- rep(5, d+1)
p <- 1/(1+exp(-cbind(1, X) %*% theta))
y <- rbinom(N, 1, p)
dat <- data.frame(y=y, x=X)

# defining loop number
n <- 1:10

# glm
system.time(for (i in n) glm.time <- glm(y ~ ., family = binomial, data = dat)) / length(n)

# biglm
system.time(for (i in n) bigglm.time <- bigglm(terms(y ~ ., data = dat), data = dat)) / length(n)

# aisgd
aisgd.time <- sgd(y ~ ., data = dat, model = "glm", model.control = list(family = "binomial"))
aisgd.time$converged
aisgd.time$estimates

# one-dim
system.time(for (i in n) aisgd.time <- sgd(y ~ ., data = dat, model = "glm", model.control = list(family = "binomial"))) / length(n)

# adagrad
system.time(for (i in n) aisgd.time <- sgd(y ~ ., data = dat, model = "glm", model.control = list(family = "binomial"), sgd.control = list(lr = "adagrad"))) / length(n)

# rmsprop
system.time(for (i in n) aisgd.time <- sgd(y ~ ., data = dat, model = "glm", model.control = list(family = "binomial"), sgd.control = list(lr = "rmsprop"))) / length(n)
```

28
# sgd
sgd.time <- sgd(y ~ ., data=dat, model="glm", model.control=list(family="binomial"), sgd.control=list(method="sgd"))
sgd.time$converged

# one-dim
system.time(for (i in n) sgd.time <- sgd(y ~ ., data=dat, model="glm", model.control=list(family="binomial"), sgd.control=list(method="sgd")) / length(n)

# adagrad
system.time(for (i in n) sgd.time <- sgd(y ~ ., data=dat, model="glm", model.control=list(family="binomial"), sgd.control=list(method="sgd", lr="adagrad")) / length(n)

# rmsprop
system.time(for (i in n) sgd.time <- sgd(y ~ ., data=dat, model="glm", model.control=list(family="binomial"), sgd.control=list(method="sgd", lr="rmsprop")) / length(n)

### Convergence behaviour ###

# Generate data
N <- 1e5
d <- 10
set.seed(921)
X <- matrix(rnorm(N*d), ncol=d)
theta <- rep(5, d+1)
p <- 1/(1+exp(-cbind(1, X) %*% theta))
y <- rbinom(N, 1, p)
dat <- data.frame(y=y, x=X)
test.set <- sample(1:nrow(dat), size=nrow(dat)/8, replace=FALSE)
dat.test <- dat[test.set,]
dat <- dat[-test.set,]
y <- dat.test$y
dat.test <- as.matrix(dat.test[c(-1)])

# AI-SGD
# Modeling
aisgd <- sgd(y ~ ., data=dat, model="glm", model.control=list(family="binomial"))
aisgd <- sgd(y ~ ., data=dat, model="glm", model.control=list(family="binomial"), sgd.control=list(pass=T))
aisgd <- sgd(y ~ ., data=dat, model="glm", model.control=list(family="binomial"), sgd.control=list(lr="rmsprop"))
aisgd <- sgd(y ~ ., data=dat, model="glm", model.control=list(family="binomial"), sgd.control=list(lr="adagrad"))

# Classification plot aissgd learning rates
aisgd_onedim <- plot(aisgd, cbind(1, dat.test), y, label="ai-sgd", type="clf", xaxis="iteration") + labs(title = "One-dim", y = "Misclassification Rate") + theme_bw()
aisgd_rmsprop <- plot(aisgd, cbind(1, dat.test), y, label="ai-sgd", type="clf", xaxis="iteration") + labs(title = "Rmsprop", y = "Misclassification Rate") + theme_bw()
aisgd_adagrad <- plot(aisgd, cbind(1, dat.test), y, label="ai-sgd", type="clf", xaxis="iteration") + labs(title = "Adagrad", y = "Misclassification Rate") + theme_bw()
multiplot_1 <- multiplot(aisgd_onedim, aissgd_rmsprop, aissgd_adagrad)

# Classification plot aissgd runtime & iteration
plot(aisgd, cbind(1, dat.test), y, label="ai-sgd", type="clf", xaxis="iteration")
plot(aisgd, cbind(1, dat.test), y, label="ai-sgd", type="clf", xaxis="runtime")
aisgd_10params <- plot(aisgd, cbind(1, dat.test), y, label="ai-sgd", type="clf", xaxis="iteration") + labs(title = "10 Parameters", y = "Misclassification Rate") + theme_bw()
aisgd_50params <- plot(aisgd, cbind(1, dat.test), y, label="ai-sgd", type="clf", xaxis="iteration") + labs(title = "50 Parameters", y = "Misclassification Rate") + theme_bw()
aisgd_runtime1 <- plot(aisgd, cbind(1, dat.test), y, label="ai-sgd", type="clf", xaxis = "runtime") + labs(title = "10 Parameters", y = "Misclassification Rate") + theme_bw()

aisgd_runtime2 <- plot(aisgd, cbind(1, dat.test), y, label="ai-sgd", type="clf", xaxis = "runtime") + labs(title = "50 Parameters", y = "Misclassification Rate") + theme_bw()

multiplot.2 <- multiplot(aisgd_iteration1, aisgd_runtime1, aisgd_iteration2, aisgd_runtime2, cols = 2)

# SGD modeling

sgd <- sgd(y ~ ., data =dat, model ="glm", model.control=list(family ="binomial"), sgd.control=list(method="sgd"))

sgd <- sgd(y ~ ., data =dat, model ="glm", model.control=list(family ="binomial"), sgd.control=list(method="sgd", pass=T))

sgd <- sgd(y ~ ., data =dat, model ="glm", model.control=list(family ="binomial"), sgd.control=list(method="sgd", lr="rmsprop"))

sgd <- sgd(y ~ ., data =dat, model ="glm", model.control=list(family ="binomial"), sgd.control=list(method="sgd", lr="adagrad"))

# classification plot sgd learning rates n=10

sgd_onedim <- plot(sgd, cbind(1, dat.test), y, label="sgd", type="clf", xaxis = "iteration") + labs(title = "One-dim", y = "Misclassification Rate") + theme_bw()

sgd_rmsprop <- plot(sgd, cbind(1, dat.test), y, label="sgd", type="clf", xaxis = "iteration") + labs(title = "Rmsprop", y = "Misclassification Rate") + theme_bw()

sgd_adagrad <- plot(sgd, cbind(1, dat.test), y, label="sgd", type="clf", xaxis = "iteration") + labs(title = "Adagrad", y = "Misclassification Rate") + theme_bw()

multiplot.3 <- multiplot(sgd_onedim, sgd_rmsprop, sgd_adagrad)

# classification plot sgd runtime & iteration

plot(sgd, cbind(1, dat.test), y, label="sgd", type="clf", xaxis = "iteration")

sgd_iteration1 <- plot(sgd, cbind(1, dat.test), y, label="sgd", type="clf", xaxis = "iteration") + labs(title = "10 Parameters", y = "Misclassification Rate") + theme_bw()

sgd_iteration2 <- plot(sgd, cbind(1, dat.test), y, label="sgd", type="clf", xaxis = "iteration") + labs(title = "50 Parameters", y = "Misclassification Rate") + theme_bw()

sgd_runtime1 <- plot(sgd, cbind(1, dat.test), y, label="sgd", type="clf", xaxis = "runtime") + labs(title = "10 Parameters", y = "Misclassification Rate") + theme_bw()

sgd_runtime2 <- plot(sgd, cbind(1, dat.test), y, label="sgd", type="clf", xaxis = "runtime") + labs(title = "50 Parameters", y = "Misclassification Rate") + theme_bw()

multiplot.4 <- multiplot(sgd_iteration1, sgd_runtime1, sgd_iteration2, sgd_runtime2, cols = 2)
References


Wickham, H. (2014). Advanced R: "Performance". Also available online, URL: http://adv-r.had.co.nz/Performance.html. CRC Press. ISO 690