UNIVERSITY OF CALIFORNIA,
IRVINE

Multivariate Hypothesis Tests for Statistical Optimization

THESIS

submitted in partial satisfaction of the requirements
for the degree of

MASTER OF SCIENCE

in Computer Science

by

Anoop Korattikara Balan

Dissertation Committee:
Professor Max Welling, Chair
Professor Alexander Ihler
Professor Deva Ramanan

2011

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

I would like to thank my thesis committee for their valuable feedback. In particular, I thank my advisor, Prof. Max Welling, for introducing me to research and for his excellent guidance and support throughout this work. I would like to thank students in Prof. Welling's research group, especially Levi Boyles, for many insightful comments and useful discussions. And finally, thanks to my parents, for everything else.

This material is based upon work supported by the National Science Foundation under Grant No's. 0447903, 0914783. The work in Chapter 6 was done in collaboration with Prof. Haesun Park and Jingu Kim from Georgia Institute of Technology.

ABSTRACT OF THE DISSERTATION

Multivariate Hypothesis Tests for Statistical Optimization

By

Anoop Korattikara Balan

Master of Science in Computer Science

University of California, Irvine, 2011

Professor Max Welling, Chair

Many machine learning problems involve minimizing an *expected* loss function defined with respect to some true underlying distribution of the data. However, in practice, one works with an *empirical* loss function, defined over a finite dataset, that is only an approximation of the expected loss. After the empirical loss function has been defined, many practitioners often ignore the expected loss and tend to consider learning as a mere mathematical optimization procedure. However, concentrating on minimizing the empirical loss function to high accuracy is wasteful, and often leads to over-fitting. Motivated by such considerations and by treating learning as a statistical procedure, we propose a stochastic optimization method that can be used to speed-up a variety of iterative parameter estimation algorithms. In each iteration, our method computes using only just enough data to reliably take a single step. Thus in the early iterations, when our parameter estimates are far from their true values and we just need a general direction to move in parameter space, we compute parameter updates using only a very small subset of the data. Then as learning proceeds, we use a frequentist hypothesis testing framework to adaptively increase the batch size. Other than the obvious computational advantages, our method also helps guard against over-fitting by stopping when the the hypothesis tests fail after using all available data. In addition, our method has the advantage that there is only a single interpretable parameter to tune. We apply these ideas to three learning problems: parameter estimation in Generalized Linear Models, Least Absolute Deviation regression and Non-negative Matrix Factorization. As our experiments on many real world datasets show, the proposed method significantly improves the performance of these algorithms, especially on large datasets.

Chapter 1

Introduction

Many problems that involve learning from data are commonly framed as optimization problems. Given a dataset $\mathcal{D}_N$ consisting of N observations, learning often involves defining a loss function f over the dataset and a set of model parameters $\boldsymbol{\beta}$, and solving for the set of parameters $\boldsymbol{\beta}^*$, in a given domain U_β, that minimize this loss function. That is, we solve the following optimization problem:

$$\boldsymbol{\beta}^* = \underset{\boldsymbol{\beta} \in U_\beta}{\arg\min}\, f(\mathcal{D}_N, \boldsymbol{\beta}) \tag{1.1}$$

For many learning problems, there is no closed form solution for $\boldsymbol{\beta}^*$ and we have to resort to iterative algorithms to perform the optimization. These iterative algorithms usually start with an initial estimate, $\boldsymbol{\beta}_0^*$, for $\boldsymbol{\beta}^*$, and in each iteration t, produces a better estimate $\boldsymbol{\beta}_t^*$ using the estimate from the previous iteration, $\boldsymbol{\beta}_{t-1}^*$, and the dataset $\mathcal{D}_N$. This is shown in Algorithm 1.

Algorithm 1 Iterative Parameter Estimation

Require: Dataset $\mathcal{D}_N$
Ensure: Parameter estimate $\hat{\boldsymbol{\beta}}^*$
1: Initialize $\boldsymbol{\beta}_0^*$
2: $t \leftarrow 1$
3: **repeat**
4: $\quad \hat{\boldsymbol{\beta}}^* \leftarrow \boldsymbol{\beta}_t^* \leftarrow u(\boldsymbol{\beta}_{t-1}^*, \mathcal{D}_N, t)$
5: $\quad t \leftarrow t + 1$
6: **until** convergence
7: **return** $\hat{\boldsymbol{\beta}}^*$

Here $u(.)$ denotes an update function that depends on the specific optimization problem. It is not necessary that all parameter estimates $\boldsymbol{\beta}_t^*$ are updated at once as in Algorithm 1, but instead, it is also possible to cycle through the parameters and update only a single parameter or a subset of parameters in each sub step.

Each update step $\boldsymbol{\beta}_t^* \leftarrow u(\boldsymbol{\beta}_{t-1}^*, \mathcal{D}_N, t)$ can be viewed as moving our current estimate $\boldsymbol{\beta}_{t-1}^*$ closer, through parameter space, to the true solution $\boldsymbol{\beta}^*$. Very commonly in machine learning, the number of observations N is very large, and evaluating the update function on a huge dataset in each iteration is computationally very expensive, or even impossible if $N \to \infty$. Often these computations are wasteful: when our estimate is far from $\boldsymbol{\beta}^*$, the updates need not be very precise as we just need a general direction to move in parameter space. However, we need more precision in the updates as we move closer to $\boldsymbol{\beta}^*$.

To apply this idea to speed up learning, we first acknowledge the stochastic nature of our data. We consider the N observations in $\mathcal{D}_N$ as samples from a stochastic data generating process. If we had access to this generating process, we can construct infinitely many datasets of size N that have similar statistical properties, and $\mathcal{D}_N$, the dataset available to us, will be just one sample from this infinite pool of datasets. Our update function, u, is a quantity that depends on the particular dataset that we have, and in certain cases, we can model the variability of u under different samplings of the dataset. As N increases, this variability decreases since more information is likely to be shared between different draws of the dataset.

Hence, when our estimate $\boldsymbol{\beta}_{t-1}^*$ is far from $\boldsymbol{\beta}^*$, and we do not need the updates to be very precise, we can evaluate the update function using a subset $\mathcal{D}_{N_t}$ of $\mathcal{D}_N$ with $N_t < N$ samples, to compute the new estimate as $\boldsymbol{\beta}_t^* = u(\boldsymbol{\beta}_{t-1}^*, \mathcal{D}_{N_t}, t)$. Usually in early iterations our estimate is very far from $\boldsymbol{\beta}^*$, and therefore the batch size N_t can be very small. As learning proceeds and we move closer to $\boldsymbol{\beta}^*$, we can increase N_t and query more samples to increase the precision of our updates. Thus, we propose an 'as needed' approach that uses only just enough data and computation to make a step.

In every iteration t, we need a way of determining if the update computed using the current mini-batch $\mathcal{D}_{N_t}$ is reliable enough, so that we can increase N_t when needed. If we can model the distribution of the updates under different draws of the mini-batch $\mathcal{D}_{N_t}$, we can devise frequentist hypothesis tests to determine if at least the proposed update *direction* is correct with high probability. More precisely, we fail the hypothesis test if ρ_t, the probability that the proposed update direction is not within 90 degrees of the true update direction $u(\boldsymbol{\beta}_{t-1}^*, \mathcal{D}_N, t) - \boldsymbol{\beta}_{t-1}^*$, is higher than a threshold ρ^*. To calculate this probability, we first have to determine the pdf p_{u_t} of the proposed update $u(\boldsymbol{\beta}_{t-1}^*, \mathcal{D}_{N_t}, t)$ and compute:

$$\rho_t = \int_{\Omega} p_{u_t}(\mathbf{u}) d\mathbf{u} \quad \text{where} \quad \Omega = \{\mathbf{u} : \left\langle \mathbf{u} - \boldsymbol{\beta}_{t-1}^*, u(\boldsymbol{\beta}_{t-1}^*, \mathcal{D}_N, t) - \boldsymbol{\beta}_{t-1}^* \right\rangle < 0\} \quad (1.2)$$

Here, Ω represents proposed updates whose directions are significantly different from the true update direction. Note, that we do not actually compute the true update $u(\boldsymbol{\beta}_{t-1}^*, \mathcal{D}_N, t)$ in Eqn.(1.2), since it is the very computation that we are trying to avoid by using our sub-sampling scheme. Thus, in practice, we use $u(\boldsymbol{\beta}_{t-1}^*, \mathcal{D}_{N_t}, t)$ as an estimate of $u(\boldsymbol{\beta}_{t-1}^*, \mathcal{D}_N, t)$. If $\rho_t > \rho^*$ and we fail the test, i.e. if we are not even

sure about the direction to move in parameter space, we have to increase our batch size N_t by querying more samples and reduce the variability of the proposed updates.

This hypothesis testing framework also provides a natural stopping criterion for the algorithm. If the hypothesis tests fail when $N_t = N$, we cannot try to decrease the variability of our updates by querying more data points. Also, the fact that the hypothesis tests failed indicate that if we were to draw another dataset of size N from the generating process, there is substantial probability that the update direction will be very different. At this point, if we were to continue learning, we would just be over-fitting to the particular dataset available to us. Therefore, all hypothesis tests failing when $N_t = N$ represents an automated and principled criterion to terminate learning. The general structure of our algorithm is shown in Algorithm 2.

Algorithm 2 Iterative Parameter Estimation with Sub-Sampling

Require: Dataset $\mathcal{D}_N$, Threshold ρ^*
Ensure: Parameter Estimate $\hat{\boldsymbol{\beta}}^*$
Initialize $\boldsymbol{\beta}_0^*$
Initialize $N_1 \ll N$
$\mathcal{D}_{N_1} \leftarrow$ Sample N_1 observations from $\mathcal{D}_N$
$t \leftarrow 1$
done $\leftarrow$ **false**
repeat
  $u_t \leftarrow u(\boldsymbol{\beta}_{t-1}^*, \mathcal{D}_{N_t}, t)$
  Calculate p_{u_t}
  Calculate ρ_t as in Eqn. 1.2
  if $\rho_t \leq \rho^*$ **then**
    $\hat{\boldsymbol{\beta}}^* \leftarrow \boldsymbol{\beta}_t^* \leftarrow u_t$
    $N_{t+1} \leftarrow N_t$
    $\mathcal{D}_{N_{t+1}} \leftarrow \mathcal{D}_{N_t}$
  else
    if $N_t = N$ **then**
      done $\leftarrow$ **true**
    else
      $\boldsymbol{\beta}_t^* \leftarrow \boldsymbol{\beta}_{t-1}^*$
      $N_{t+1} \leftarrow \min(N, 2N_t)$
      $\mathcal{D}_{N_{t+1}} \leftarrow$ Sample N_{t+1} observations from $\mathcal{D}_N$
    end if
  end if
  $t \leftarrow t + 1$
until *done*
return $\hat{\boldsymbol{\beta}}^*$

The main challenge in applying this method is to determine p_{u_t}, the pdf of the update function. Fortunately, for many machine learning algorithms, the update function involves sums or averages over data points. Thus, with sufficiently large batch sizes,

the Central Limit Theorem dictates that the update function will be normally distributed, and we can fully specify p_{u_t} by estimating its mean and covariance using the current batch.

In the rest of the paper, we illustrate how this method can be used to significantly optimize three learning algorithms:

- A Fisher Scoring algorithm for parameter estimation in Generalized Linear Models (GLMs)
- An Expectation-Maximization (EM) algorithm for Least Absolution Deviation(LAD) Regression.
- The Block Principal Pivoting (BPP) algorithm, an active set type method which is state-of-the-art for Non-Negative Matrix Factorization (NMF).

All of these methods are iterative algorithms and their update functions have a form similar to the solution of normal equations for least squares problems, i.e.

$$u(\boldsymbol{\beta}^*_{t-1}, \mathcal{D}_N, t) = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b} \tag{1.3}$$

It will be specified in later chapters what $\mathbf{A}$ and $\mathbf{b}$ represent for each of the above algorithms. The rest of this thesis is structured as follows. In chapter 2, we review related work. Then, in chapter 3, we develop a hypothesis testing framework for testing the reliability of updates having the form of Eqn. 1.3. We apply this hypothesis testing framework to optimize different learning algorithms in chapters 4, 5 and 6. Finally, we conclude in chapter 7.

Chapter 2

Related Work

Our method is related to Stochastic Approximation (SA) [17, 12, 18], a class of iterative algorithms that can be used to minimize a loss function using only noisy gradients. In SA, the updates at each step are multiplied with a gain value, that decreases over time to ensure convergence. Thus, a key difference is that we systematically reduce the *uncertainty in the updates* by using larger and larger batches, whereas in SA, the *effect of uncertainty* is reduced by using smaller and smaller gain values. In SA, although it is easy to choose a gain sequence that ensures convergence, in general it requires a lot of tuning to find the one that gives the best performance. In contrast, our method has a single interpretable parameter (the significance level for hypothesis tests) and has a statistically principled stopping criterion.

Our ideas also have a close connection to the observations in [4] and [20]. Usually in learning problems, one is interested in minimizing an *expected* loss function with respect to some true underlying distribution of the data. However, in practice, one works with an *empirical* loss function defined with respect to a finite dataset, that is only an approximation of the expected loss. Therefore, it is wasteful to expend too much computational time in minimizing the empirical loss to very high accuracy. Additionally, as discussed in [4], when working with large datasets, methods that approximate updates at each step are often more effective. However, we are not aware of previous work where hypothesis tests are used to increase the batch size or as a stopping criterion.

Chapter 3

Hypothesis Testing

Let us consider applying our sampling strategy to iterative algorithms where the update function is of the form $(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b}$. We assume $\mathbf{A} \in \mathbb{R}^{N\times P}$ and $\mathbf{b} \in \mathbb{R}^{N\times 1}$, and that $\mathbf{A}$ and $\mathbf{b}$ are functions of the dataset $\mathcal{D}_N$ and the parameter estimate from the previous iteration $\boldsymbol{\beta}^*_{t-1}$. If we denote the i^{th} row of $\mathbf{A}$ by $\mathbf{a}_i^T$ and the i^{th} component of $\mathbf{b}$ by b_i, we can write the P dimensional update function as:

$$\mathbf{u}_N = u(\boldsymbol{\beta}^*_{t-1}, \mathcal{D}_N, t) = \left(\frac{1}{N}\sum_{i=1}^{N}\mathbf{a}_i\mathbf{a}_i^T\right)^{-1}\left(\frac{1}{N}\sum_{i=1}^{N}\mathbf{a}_i b_i\right) \tag{3.1}$$

If N is very large, evaluating this update function in every iteration becomes very costly. However, we can think of $\mathbf{a}_i$'s and b_i's as i.i.d instances of some random variables $\mathbf{a}$ and b, and introduce the following stochastic generative process $b = \mathbf{a}^T\mathbf{u} + \epsilon$, where ϵ is an error term and $\mathbf{u}$ are the parameters of the model. Thus, the update function in (3.1), $\mathbf{u}_N$, becomes the maximum likelihood estimate of the model parameters, calculated from N observations, assuming that the errors ϵ are i.i.d normally distributed random variables with zero mean and constant variance. In each iteration t, we can consider approximating the update function in (3.1) using a subset, $\mathcal{D}_{N_t}$, of the full dataset, $\mathcal{D}_N$, with $N_t < N$ samples of $\mathbf{a}$ and b:

$$\mathbf{u}_{N_t} = u(\boldsymbol{\beta}^*_{t-1}, \mathcal{D}_{N_t}, t) = \left(\frac{1}{N_t}\sum_{i=1}^{N_t}\mathbf{a}_i\mathbf{a}_i^T\right)^{-1}\left(\frac{1}{N_t}\sum_{i=1}^{N_t}\mathbf{a}_i b_i\right) \tag{3.2}$$

If N_t is sufficiently large for the Central Limit Theorem to take effect, and the following assumptions hold:

1. No multicollinearity: $\mathbf{Q_{aa}} = \mathbb{E}[\mathbf{a}\mathbf{a}^T]$ is positive definite
2. Exogeneity: $\mathbb{E}[\epsilon|\mathbf{a}] = 0$
3. Homoscedasticity: $Var[\epsilon|\mathbf{a}] = \sigma^2$

then the maximum likelihood estimator for $\mathbf{u}$ is known to be asymptotically normal [19]:

$$\mathbf{u}_{N_t} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \text{ where } \boldsymbol{\Sigma} = \frac{\mathbf{Q}_{\mathbf{aa}}^{-1}\boldsymbol{\sigma}^2}{N_t} \tag{3.3}$$

Here $\boldsymbol{\mu}$ denotes the true value of the model parameters $\mathbf{u}$. Note that the variance is inversely proportional to the number of samples N_t, i.e. the update is more precise if computed from a large number of samples. We can approximate this distribution as $\mathbf{u}_{N_t} \sim \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$ by estimating its parameters from the subset $\mathcal{D}_{N_t}$:

$$\boldsymbol{\mu}_t = \left(\frac{1}{N_t}\sum_{i=1}^{N_t} \mathbf{a}_i \mathbf{a}_i^T\right)^{-1} \left(\frac{1}{N_t}\sum_{i=1}^{N_t} \mathbf{a}_i b_i\right) \tag{3.4a}$$

$$\mathbf{Q}_{\mathbf{aa},t} = \frac{1}{N_t - 1}\sum_{i=1}^{N_t} \mathbf{a}_i {\mathbf{a}_i}^T \tag{3.4b}$$

$$\sigma_t^2 = \frac{1}{N_t - 1}\sum_{i=1}^{N_t} \epsilon_i^2 \tag{3.4c}$$

$$\boldsymbol{\Sigma}_t = \frac{Q_{\mathbf{aa},t}^{-1}\sigma_t^2}{N_t} \tag{3.4d}$$

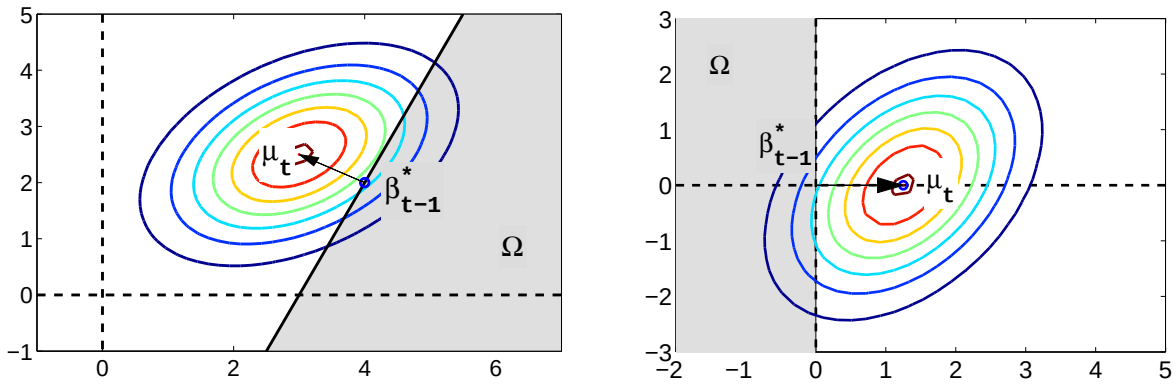
Estimating this distribution of the proposed approximate update, $\mathbf{u}_{N_t}$, allows us to devise frequentist hypothesis tests to determine the reliability of this proposition. As in (1.2), we compute ρ_t, the probability that the proposed updated direction is not within 90 degrees of the true update direction $\boldsymbol{\mu} - \boldsymbol{\beta}_{t-1}^* \approx \boldsymbol{\mu}_t - \boldsymbol{\beta}_{t-1}^*$. Denoting the pdf of the distribution $\mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$ of the proposed update by p_{u_t}, we compute ρ_t as:

$$\rho_t = \int_\Omega p_{u_t}(\mathbf{u}) d\mathbf{u} \quad \text{where} \quad \Omega = \{\mathbf{u} : \left\langle \mathbf{u} - \boldsymbol{\beta}_{t-1}^*, \boldsymbol{\mu}_t - \boldsymbol{\beta}_{t-1}^* \right\rangle < 0\} \tag{3.5}$$

To compute ρ_t efficiently, consider a transformation of the co-ordinate system so that $\boldsymbol{\beta}_{t-1}^*$ is at the origin and the true update direction, $\boldsymbol{\mu}_t - \boldsymbol{\beta}_{t-1}^*$, is along the first co-ordinate axis. Under this transformation, the distribution of the proposed update changes as:

$$\mathbf{u}_{N_t} \sim \mathcal{N}(\boldsymbol{\mu}_t', \boldsymbol{\Sigma}_t') \text{ where } \boldsymbol{\mu}_t' = [\|\boldsymbol{\mu}_t - \boldsymbol{\beta}_{t-1}^*\|; \mathbf{0}_{P-1}] \text{ and } \boldsymbol{\Sigma}_t' = \mathbf{R}^{-1}\boldsymbol{\Sigma}_t\mathbf{R} \tag{3.6}$$

Here, $\mathbf{0}_{P-1}$ represents a vector of $P-1$ zeros and $\mathbf{R}$ is the rotation matrix that aligns the true update direction $\boldsymbol{\mu}_t - \boldsymbol{\beta}_{t-1}^*$ with the first co-ordinate axis. Now, consider a hyperplane passing through the origin and orthogonal to the first co-ordinate axis. This divides $\mathbb{R}^P$ into two half spaces, $\mathbb{R}_+^P$ and $\mathbb{R}_-^P$, containing the positive and negative parts of the first co-ordinate axis respectively. Since the true update direction is directed from the origin to the positive side of the first co-ordinate axis, the region Ω that represents proposed update directions which differ by more than 90 degrees from the true direction, is just $\mathbb{R}_-^P$. Now, ρ_t can be easily computed



(a) (b)

Figure 3.1: Hypothesis testing: In 3.1a, the old estimate $\boldsymbol{\beta}^*_{t-1}$ and the distribution of the proposed update $\mathbf{u}_{N_t}$ are shown. The shaded region, Ω, represents proposed update direction which are not within 90 degrees of the true direction. 3.1b shows how transforming to an alternate coordinate system can help compute ρ_t efficiently.

from the marginal distribution of $\mathbf{u}_{N_t}[1]$, the first component of $\mathbf{u}_{N_t}$. This distribution is just $\mathcal{N}(\boldsymbol{\mu}'_t[1], \boldsymbol{\Sigma}'_t[1,1])$ and $\rho_t = \Phi(0)$ where $\Phi(.)$ is the CDF of the $\mathbf{u}_{N_t}[1]$. This is shown in Figures 3.1a and 3.1b.

If ρ_t is greater than a threshold ρ^*, there is substantial probability that the proposed update direction is significantly different from the true direction, and we fail the hypothesis test. At this point, we have to increase the batch size to reduce the variance in the updates so that they pass the hypothesis tests. If this is not possible, either due to unavailability of more data or because of computational limitations, we terminate learning so as to avoid over-fitting.

We will now write this hypothesis test as a procedure so that it can be easily referenced from later sections.

Procedure 3 HYPOTHESIS-TEST

Require: $\mathbf{A} \in \mathbb{R}^{N_t \times P}$, $\mathbf{b} \in \mathbb{R}^{N_t \times 1}$, $\boldsymbol{\beta}^*_{t-1} \in \mathbb{R}^{P \times 1}$, ρ^*

Ensure: *pass*

$\boldsymbol{\mu}_t \leftarrow (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b}$

$\mathbf{Q}_{\mathbf{aa},t} \leftarrow \dfrac{1}{N_t - 1}\mathbf{A}^T\mathbf{A}$

$\sigma_t^2 \leftarrow \dfrac{1}{N_t - 1}\|\mathbf{A}\boldsymbol{\beta}^*_{t-1} - \mathbf{b}\|_2^2$

$\Sigma_t \leftarrow \frac{\mathbf{Q}^{-1}_{\mathbf{aa},t}\sigma_t^2}{N_t}$

$\boldsymbol{\mu}'_t \leftarrow [\|\boldsymbol{\mu}_t - \boldsymbol{\beta}^*_{t-1}\|; \mathbf{0}_{P-1}]$

$\mathbf{R} \leftarrow$ Rotation matrix s.t. $\mathbf{R}(\boldsymbol{\mu}_t - \boldsymbol{\beta}^*_{t-1}) = \boldsymbol{\mu}'_t$

$\boldsymbol{\Sigma}'_t \leftarrow \mathbf{R}^{-1}\boldsymbol{\Sigma}_t\mathbf{R}$

$\rho_t \leftarrow \Phi(0)$ where $\Phi(.)$ is cdf of $\mathcal{N}(\boldsymbol{\mu}'_t[1], \boldsymbol{\Sigma}'_t[1,1])$

if $\rho_t > \rho^*$ **then**

    $pass \leftarrow$ **false**

else

    $pass \leftarrow$ **true**

end if

return *pass*

Chapter 4

Generalized Linear Models

In this Chapter, we will show how the sub-sampling algorithm and hypothesis testing framework of Chapters 1 and 3 can be used to speed-up parameter estimation in Generalized Linear Models (GLMs). In section 4.1, we give a brief introduction to GLMs. Section 4.2 reviews the Exponential Family of distributions, as they form an important part of the theory of GLMs. Section 4.3 describes a Fisher scoring method for parameter estimation in GLMs. In Section 4.4, we show how our sub-sampling idea can be used to speed-up this Fisher scoring algorithm. Finally, in section 4.5, we present experimental results showing how our method optimizes Fisher scoring for parameter estimation in GLMs on 2 real world datasets.

4.1 Introduction

Generalized Linear Model (GLM)[15] is a flexible extension of the classical linear model for regression. Hence, we will first examine the classical linear model. In this model, we are given a dataset $\mathcal{D}_N$ consisting of N statistical units, with the i^{th} unit consisting of a target variable y_i and a set of P covariates denoted by the vector $\mathbf{x}_i \in \mathbb{R}^{P\times 1}$. The classical linear model has the following 3 part specification:

1. *Random Component*: The targets y_i are considered as realizations of random variables Y_i, which are assumed to have independent normal distributions with constant variance.

2. *Systematic Component*: The covariates are used to produce a linear predictor η_i given by:

$$\eta_i = \langle \mathbf{x}_i, \boldsymbol{\beta} \rangle \tag{4.1}$$

Here $\boldsymbol{\beta} \in \mathbb{R}^{P\times 1}$ represents the unknown parameters of the model and the process of calculating them is called *parameter estimation*.

3. *Link*: The link between the random component and the systematic component is specified as:

$$\eta_i = g(\mathbb{E}[Y_i]) \tag{4.2}$$

where $g(.)$ is the identity function.

GLMs allow the following 2 generalizations of the classical linear model:

1. In the random component, the distribution of Y_i's need not be normal, but can be any distribution from the exponential family (see Section 4.2).
2. The link $g(.)$ is not restricted to be the identity function, but can be any monotonic differentiable function.

We will briefly review the exponential family of distributions in the next section.

4.2 The Exponential Family

In GLMs, the random variables Y_i's are allowed to have any distribution from the exponential family, which has the following general form:

$$p_Y(y;\theta,\phi) = \exp\left\{\frac{y\theta - r(\theta)}{q(\phi)} + s(y,\phi)\right\} \tag{4.3}$$

for some specific functions $q(.)$, $r(.)$ and $s(.)$. Here, θ is usually the parameter of interest and is called the canonical parameter of the distribution. ϕ is known as the dispersion parameter and in GLMs, it is the same for all observations. The expectation and variance of Y can be shown to be:

$$\mathbb{E}[Y] = \mu = r'(\theta) \tag{4.4a}$$
$$\text{Var}[Y] = r''(\theta)q(\phi) \tag{4.4b}$$

Note that the expression for variance contains the term $r''(\theta)$ which is dependent on the mean. This term when written as a function of the mean is called the *variance function* and is denoted by $V(\mu) = r''(\theta)$. Also, it is worth mentioning that a link function $g(.)$ for which $\eta = g(\mu) = \theta$ is called the canonical link function of the distribution.

We will now consider two illustrative examples of distributions from the exponential family: the Normal distribution and the Bernoulli distribution.

4.2.1 Example 1: Normal distribution as an Exponential Family distribution

The pdf of a normally distributed random variable y is:

$$\begin{aligned} p(y;\alpha,\omega^2) &= \frac{1}{\sqrt{2\pi\omega^2}}\exp\left\{-\frac{(y-\alpha)^2}{2\omega^2}\right\} \\ &= \exp\left\{\frac{y\alpha-(\alpha^2/2)}{\omega^2}-\frac{1}{2}\log 2\pi\omega^2-\frac{y^2}{2\omega^2}\right\} \end{aligned} \tag{4.5}$$

Comparing the coefficient of y in Eqns. 4.5 & 4.3, we see that $\theta=\alpha$ and we choose $\phi=\omega$. Now, we can see that the normal distribution belongs to the exponential family with $r(.)$, $q(.)$ and $s(.)$ defined as follows:

$$r(\theta) = \frac{\theta^2}{2} \tag{4.6a}$$

$$q(\phi) = \phi^2 \tag{4.6b}$$

$$s(y,\phi) = -\frac{1}{2}\log 2\pi\phi^2 - \frac{y^2}{2\phi^2} \tag{4.6c}$$

We can now calculate the expectation and variance of y as:

$$\mathbb{E}[Y] = \mu = r'(\theta) = \theta = \alpha \tag{4.7a}$$

$$Var[Y] = r''(\theta)q(\phi) = \phi^2 = \omega^2 \tag{4.7b}$$

The variance function is given by,

$$V(\mu) = r''(\theta) = 1 \tag{4.8}$$

4.2.2 Example 2: Bernoulli distribution as an Exponential Family distribution

The pdf of a Bernoulli distributed random variable y is:

$$\begin{aligned} p(y;\alpha) &= \alpha^y(1-\alpha)^{1-y} \\ &= \exp\left\{y\log\left(\frac{\alpha}{1-\alpha}\right)+\log(1-\alpha)\right\} \end{aligned} \tag{4.9}$$

Comparing Eqns. 4.9 and 4.3, we have:

$$\theta = \log\left(\frac{\alpha}{1-\alpha}\right) \text{ or, equivalently: } \alpha = \frac{1}{1+e^{-\theta}} \tag{4.10}$$

We can see that the Bernoulli distribution also belongs to the Exponential Family with $q(.)$, $r(.)$ and $s(.)$ defined as:

$$r(\theta) = -\log(1-\alpha) = \log(1+e^{\theta}) \tag{4.11a}$$
$$q(\phi) = s(y,\phi) = 1 \tag{4.11b}$$

The expectation and variance of y are:

$$\mathbb{E}[Y] = \mu = r'(\theta) = \frac{1}{1+e^{-\theta}} = \alpha \tag{4.12a}$$
$$Var[Y] = r''(\theta)q(\phi) = \frac{e^{-\theta}}{(1+e^{-\theta})^2} = \alpha(1-\alpha) \tag{4.12b}$$

The variance function is given by,

$$V(\mu) = r''(\theta) = \mu(1-\mu) \tag{4.13}$$

4.3 Parameter Estimation using Fisher scoring

Once a particular generalized linear model has been selected by choosing a link function and a distribution from the exponential family for the Y_i's, we have to calculate the parameter vector $\boldsymbol{\beta}^* \in \mathbb{R}^{P\times 1}$ that makes the model best agree with the given data. This is typically done by Maximum Likelihood Estimation, where we try to find the $\boldsymbol{\beta}^*$ which maximizes the following log likelihood:

$$\ell(\boldsymbol{\beta}) = \sum_{i=1}^{N} \left\{ \frac{y_i\theta_i - r(\theta_i)}{q(\phi)} + s(y_i,\phi) \right\} \tag{4.14}$$

Note that ℓ depends on $\boldsymbol{\beta}$ through θ, since $\langle \mathbf{x}_i, \boldsymbol{\beta} \rangle = \eta_i = g(\mu_i) = g(r'(\theta_i))$. This maximization can be performed using the Fisher scoring method, an iterative optimization algorithm (as in Algorithm 1) having the following update rule:

$$\boldsymbol{\beta}_t^* = u(\boldsymbol{\beta}_{t-1}^*, \mathcal{D}_N, t) = \boldsymbol{\beta}_{t-1}^* + \mathcal{I}^{-1}(\boldsymbol{\beta}_{t-1}^*)\mathcal{S}(\boldsymbol{\beta}_{t-1}^*) = \mathcal{I}^{-1}(\boldsymbol{\beta}_{t-1}^*)\left[\mathcal{I}(\boldsymbol{\beta}_{t-1}^*)\boldsymbol{\beta}_{t-1}^* + \mathcal{S}(\boldsymbol{\beta}_{t-1}^*)\right] \tag{4.15}$$

Here $\mathcal{S}(\boldsymbol{\beta})$ is the score function, the gradient of the log likelihood with respect to $\boldsymbol{\beta}$, and $\mathcal{I}(\boldsymbol{\beta})$ is the Fisher information matrix, the variance of the score. It can be

shown[15] that:

$$\mathcal{S}(\boldsymbol{\beta}) \triangleq \nabla \ell = \frac{1}{a(\phi)} \sum_{i=1}^{N} W(\mu_i)(y_i - \mu_i) g'(\mu_i) \mathbf{x}_i \tag{4.16}$$

$$\text{where} \quad \frac{1}{W(\mu_i)} = (g'(\mu_i))^2 V(\mu_i) \tag{4.17}$$

and

$$\mathcal{I}(\boldsymbol{\beta}) \triangleq -\mathbb{E}\left[\mathcal{H}(\ell)\right] = \frac{1}{a(\phi)} \sum_{i=1}^{N} W(\mu_i) \mathbf{x}_i \mathbf{x}_i^T \tag{4.18}$$

In Eqn. (4.16), the ∇ symbol represents the gradient operator and in Eqn. (4.18), $\mathcal{H}(\ell)$ represents the Hessian matrix of ℓ. Also μ is a function of $\boldsymbol{\beta}$ although we have not indicated it explicitly, in order to avoid clutter. From Eqn. (4.18), we can calculate $\mathcal{I}(\boldsymbol{\beta})\boldsymbol{\beta}$ as:

$$\mathcal{I}(\boldsymbol{\beta})\boldsymbol{\beta} = \left(\frac{1}{a(\phi)} \sum_{i=1}^{N} W(\mu_i) \mathbf{x}_i \mathbf{x}_i^T \right) \boldsymbol{\beta} = \frac{1}{a(\phi)} \sum_{i=1}^{N} W(\mu_i) \eta_i \mathbf{x}_i = \frac{1}{a(\phi)} \sum_{i=1}^{N} W(\mu_i) g(\mu_i) \mathbf{x}_i \tag{4.19}$$

Using (4.16) and (4.19), we have:

$$\mathcal{I}(\boldsymbol{\beta})\boldsymbol{\beta} + \mathcal{S}(\boldsymbol{\beta}) = \frac{1}{a(\phi)} \sum_{i=1}^{N} \{ g(\mu_i) + (y_i - \mu_i) g'(\mu_i) \} W(\mu_i) \mathbf{x}_i \tag{4.20}$$

It is convenient to introduce the following definitions: the design matrix $\mathbf{X} \in \mathbb{R}^{N \times P}$ whose i^{th} row is $\mathbf{x}_i^T$, the diagonal weighting matrix $\mathbf{W}(\boldsymbol{\beta}) \in \mathbb{R}^{N \times N}$ where $\mathbf{W}_{ii}(\boldsymbol{\beta}) = W(\mu_i)$ and the vectors $\mathbf{y}, \boldsymbol{\mu} \in \mathbb{R}^{N \times 1}$ whose i^{th} components are y_i and μ_i respectively. Additionally, we define the adjusted response vector, $\mathbf{z}(\boldsymbol{\beta}) \in \mathbb{R}^{N \times 1}$ as:

$$\mathbf{z}(\boldsymbol{\beta}) = g(\boldsymbol{\mu}) + (\mathbf{y} - \boldsymbol{\mu}) g'(\boldsymbol{\mu}) \tag{4.21}$$

This allows us to write:

$$\mathcal{I}(\boldsymbol{\beta}) = \frac{1}{a(\phi)} \mathbf{X}^T \mathbf{W}(\boldsymbol{\beta}) \mathbf{X} \quad \text{and} \quad \mathcal{I}(\boldsymbol{\beta})\boldsymbol{\beta} + \mathcal{S}(\boldsymbol{\beta}) = \frac{1}{a(\phi)} \mathbf{X}^T \mathbf{W}(\boldsymbol{\beta}) \mathbf{z}(\boldsymbol{\beta}) \tag{4.22}$$

Using (4.22) and defining $\mathbf{W}_t = \mathbf{W}(\boldsymbol{\beta}_t^*)$ and $\mathbf{z}_t = \mathbf{z}(\boldsymbol{\beta}_t^*)$, we can write the Fisher scoring update rule (4.15) as:

$$\boldsymbol{\beta}_t^* = (\mathbf{X}^T \mathbf{W}_{t-1} \mathbf{X})^{-1} \mathbf{X}^T \mathbf{W}_{t-1} \mathbf{z}_{t-1} \tag{4.23}$$

Since the updates to the $\boldsymbol{\beta}^*$ estimate are similar to the solution to a weighted least squares problem, and the weights $\mathbf{W}_t$ are recomputed in every iteration, this algorithm

is also known as the Iterative Reweighted Least Squares (IRLS) algorithm. We now introduce the additional definitions $\tilde{\mathbf{X}}_t = \mathbf{W}_t^{1/2}\mathbf{X}$ and $\tilde{\mathbf{y}}_t = \mathbf{W}_t^{1/2}\mathbf{z}_t$, which enables us to write the update rule for $\boldsymbol{\beta}_t^*$ as:

$$\boldsymbol{\beta}_t^* = u(\boldsymbol{\beta}_{t-1}^*, \mathcal{D}_N, t) = (\tilde{\mathbf{X}}_{t-1}^T\tilde{\mathbf{X}}_{t-1})^{-1}\tilde{\mathbf{X}}_{t-1}^T\tilde{\mathbf{y}}_{t-1} \tag{4.24}$$

We outline the general structure of the Fisher scoring method in Algorithm 4. The algorithm calls FS-UPDATE, (Procedure 5) which represents the update function $u(\boldsymbol{\beta}_{t-1}^*, \mathcal{D}_N, t)$. Note that Algorithm 4 (along with Procedure 5) describes the Fisher scoring algorithm for any Generalized Linear Model. Fully specifying the algorithm for a particular model involves working out the model specific details of the FS-UPDATE procedure. We will now illustrate this using the Logistic Regression model as an example.

Algorithm 4 Fisher Scoring for GLM fitting

Require: Dataset $\mathcal{D}_N = \{\mathbf{y} \in \mathbb{R}^{N\times 1}, \mathbf{X} \in \mathbb{R}^{N\times P}\}$
Ensure: Parameter estimate $\hat{\boldsymbol{\beta}}^* \in \mathbb{R}^{P\times 1}$
1: Initialize $\boldsymbol{\beta}_0^* \in \mathbb{R}^{P\times 1}$
2: $t \leftarrow 1$
3: **repeat**
4: $\quad \hat{\boldsymbol{\beta}}^* \leftarrow \boldsymbol{\beta}_t^* \leftarrow$ FS-UPDATE$(\mathcal{D}_N, \boldsymbol{\beta}_{t-1}^*)$
5: $\quad t \leftarrow t + 1$
6: **until** convergence
7: **return** $\hat{\boldsymbol{\beta}}^*$

Procedure 5 FS-UPDATE

Require: Dataset: $\mathcal{D}_N = \{\mathbf{y} \in \mathbb{R}^{N\times 1}, \mathbf{X} \in \mathbb{R}^{N\times P}\}$, Old Parameter Estimate: $\boldsymbol{\beta}_{\text{old}}^* \in \mathbb{R}^{P\times 1}$
Ensure: New Parameter estimate: $\boldsymbol{\beta}_{\text{new}}^* \in \mathbb{R}^{P\times 1}$
1: $\boldsymbol{\mu} \leftarrow g^{-1}(\mathbf{X}\boldsymbol{\beta}_{\text{old}}^*)$
2: $\mathbf{w} \leftarrow \dfrac{1}{(g'(\boldsymbol{\mu}))^2 V(\boldsymbol{\mu})}$
3: $\mathbf{W} \leftarrow \text{diag}(\mathbf{w})$
4: $\mathbf{z} \leftarrow g(\boldsymbol{\mu}) + (\mathbf{y} - \boldsymbol{\mu})g'(\boldsymbol{\mu})$
5: $\tilde{\mathbf{X}} \leftarrow \mathbf{W}^{1/2}\mathbf{X}$
6: $\tilde{\mathbf{y}} \leftarrow \mathbf{W}^{1/2}\mathbf{z}$
7: $\boldsymbol{\beta}_{\text{new}}^* \leftarrow (\tilde{\mathbf{X}}^T\tilde{\mathbf{X}})^{-1}\tilde{\mathbf{X}}^T\tilde{\mathbf{y}}$
8: **return** $\boldsymbol{\beta}_{\text{new}}^*$

4.3.1 Example: Fisher Scoring for Logistic Regression

Consider a GLM where the Y_i's have independent Bernoulli distributions and $g(.)$ is the canonical link for the Bernoulli distribution i.e. $g(.)$ is a function such that $g(\mu) = \theta$. From Eqns. (4.10) and (4.12a), we have:

$$g(\alpha) = \log\left(\frac{\alpha}{1-\alpha}\right) \quad \text{and} \quad g^{-1}(\theta) = \frac{1}{1+e^{-\theta}} \tag{4.25}$$

The random variable, Y_i, representing the target, and the covariate vector, $\mathbf{x}_i$, are related as:

$$\mathbb{E}[Y_i] = \mu_i = g^{-1}(\eta_i) = \frac{1}{1+e^{-\langle \mathbf{x}_i, \boldsymbol{\beta} \rangle}} \tag{4.26}$$

This is the familiar logistic regression model. We will now derive expressions for the weighting matrix, $\mathbf{W}$, and the adjusted response vector, $\mathbf{z}$, to fully specify the Fisher scoring algorithm for this model. First, from (4.25), we see that:

$$g'(\alpha) = \frac{1}{\alpha(1-\alpha)} \tag{4.27}$$

$\mathbf{W}$ is a diagonal matrix, with diagonal entries given by:

$$\mathbf{W}_{ii} = W(\mu_i) \triangleq \frac{1}{(g'(\mu_i))^2 V(\mu_i)} = \mu_i(1-\mu_i) \tag{4.28}$$

Here, we have plugged in the definitions for $g'(\mu_i)$ and $V(\mu_i)$ from Eqns. (4.27) and (4.8) respectively. The adjusted response vector $\mathbf{z}$ can be calculated as:

$$\begin{aligned} \mathbf{z} &\triangleq g(\boldsymbol{\mu}) + (\mathbf{y} - \boldsymbol{\mu}) g'(\boldsymbol{\mu}) \\ &= \mathbf{X}\boldsymbol{\beta} + \mathbf{W}^{-1}(\mathbf{y} - \boldsymbol{\mu}) \end{aligned} \tag{4.29}$$

We have used the fact that $g(\boldsymbol{\mu}) = g(g^{-1}(\boldsymbol{\eta})) = \boldsymbol{\eta} = \mathbf{X}\boldsymbol{\beta}$ and $g'(\mu_i) = \frac{1}{\mu_i(1-\mu_i)} = \frac{1}{W(\mu_i)}$. Using the expressions for $\mathbf{W}$ and $\mathbf{z}$, we describe the model specific update function for logistic regression in Procedure 6 (FS-UPDATE-LOGISTIC).

Replacing the call to FS-UPDATE by FS-UPDATE-LOGISTIC in Algorithm 4, we obtain the Fisher scoring algorithm for logistic regression.

Procedure 6 FS-UPDATE-LOGISTIC

Require: Dataset: $\mathcal{D}_N = \{\mathbf{y} \in \mathbb{R}^{N\times 1}, \mathbf{X} \in \mathbb{R}^{N\times P}\}$, Old Parameter Estimate: $\boldsymbol{\beta}^*_{\text{old}} \in \mathbb{R}^{P\times 1}$

Ensure: New Parameter estimate: $\boldsymbol{\beta}^*_{\text{new}} \in \mathbb{R}^{P\times 1}$

1: $\boldsymbol{\mu} \leftarrow \dfrac{1}{1+\exp\{-\mathbf{X}\boldsymbol{\beta}^*_{\text{old}}\}}$
2: $\mathbf{w} \leftarrow \boldsymbol{\mu}(\mathbf{1}-\boldsymbol{\mu})$
3: $\mathbf{W} \leftarrow \text{diag}(\mathbf{w})$
4: $\mathbf{z} \leftarrow \mathbf{X}\boldsymbol{\beta}^*_{\text{old}} + \mathbf{W}^{-1}(\mathbf{y}-\boldsymbol{\mu})$
5: $\tilde{\mathbf{X}} \leftarrow \mathbf{W}^{1/2}\mathbf{X}$
6: $\tilde{\mathbf{y}} \leftarrow \mathbf{W}^{1/2}\mathbf{z}$
7: $\boldsymbol{\beta}^*_{\text{new}} \leftarrow (\tilde{\mathbf{X}}^T\tilde{\mathbf{X}})^{-1}\tilde{\mathbf{X}}^T\tilde{\mathbf{y}}$
8: **return** $\boldsymbol{\beta}^*_{\text{new}}$

4.4 Statistical Optimization of Fisher scoring

In this section, we will apply the ideas described in Chapters 1 and 3 to speed-up the Fisher scoring algorithm. First, let us consider Procedure 5 (FS-UPDATE), the update function at each step of the Fisher scoring algorithm. It is easy to see that the running time of this procedure is proportional to N, the number of statistical units in the dataset. Thus, when faced with large datasets, computing the true update function in every step is very expensive.

Therefore, in the early iterations where our estimate is far from the true solution and we just need a rough direction to move in parameter space, we can approximate the update function using a mini-batch of statistical units. Since the updates are of the form $(A^TA)^{-1}A^Tb$, we can use the hypothesis testing framework (Procedure 3) developed in Chapter 3, to measure the reliability of these updates. If the hypothesis tests fail and we are not even sure about the direction to move in parameter space, we grow our mini-batch by adding more statistical units, so as to increase the precision of our updates and pass the hypothesis tests. When the hypothesis tests fail after using all the data available to us, we stop learning so as to avoid over-fitting.

It is fairly straightforward to use the general skeleton of our sub-sampling method (Algorithm 2) along with Procedure 5 (FS-UPDATE) and Procedure 3 (HYPOTHESIS-TEST) to develop an optimized version of Fisher scoring. This is shown in Algorithm 7.

Algorithm 7 Fisher Scoring for GLM fitting with Sub-Sampling

Require: Dataset $\mathcal{D}_N = \{\mathbf{y} \in \mathbb{R}^{N\times 1}, \mathbf{X} \in \mathbb{R}^{N\times P}\}$, Threshold ρ^*
Ensure: Parameter Estimate $\hat{\boldsymbol{\beta}}^*$
Initialize $\boldsymbol{\beta}_0^*$
Initialize $N_1 \ll N$
$\{\mathbf{y}_1, \mathbf{X}_1\} \leftarrow$ Sample N_1 statistical units from $\{\mathbf{y}, \mathbf{X}\}$
$t \leftarrow 1$
done $\leftarrow$ **false**
repeat
  $\mathbf{u}_t \leftarrow$ FS-UPDATE$(\{\mathbf{y}_t, \mathbf{X}_t\}, \boldsymbol{\beta}_{t-1}^*)$
  $htPassed \leftarrow$ HYPOTHESIS-TEST$(\mathbf{X}_t, \mathbf{y}_t, \boldsymbol{\beta}_{t-1}^*, \rho^*)$
  if $htPassed$ **then**
    $\hat{\boldsymbol{\beta}}^* \leftarrow \boldsymbol{\beta}_t^* \leftarrow \mathbf{u}_t$
    $N_{t+1} \leftarrow N_t$
    $\{\mathbf{y}_{t+1}, \mathbf{X}_{t+1}\} \leftarrow \{\mathbf{y}_t, \mathbf{X}_t\}$
  else
    if $N_t = N$ **then**
      done $\leftarrow$ **true**
    else
      $\boldsymbol{\beta}_t^* \leftarrow \boldsymbol{\beta}_{t-1}^*$
      $N_{t+1} \leftarrow \min(N, 2N_t)$
      $\{\mathbf{y}_{t+1}, \mathbf{X}_{t+1}\} \leftarrow$ Sample N_{t+1} statistical units from $\{\mathbf{y}, \mathbf{X}\}$
    end if
  end if
  $t \leftarrow t + 1$
until *done*
return $\hat{\boldsymbol{\beta}}^*$

4.5 Experiments

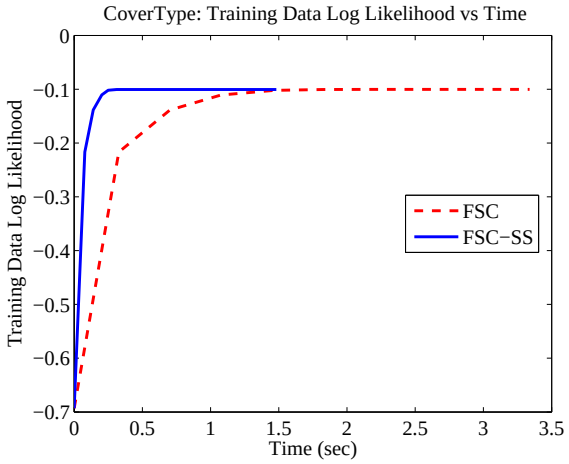
We compared the performance of the Fisher Scoring algorithm for Logistic regression (FSC) versus its sub-sampled version (FSC-SS) on 2 real world datasets from the UCI Machine Learning Repository [7]: the Poker Hands dataset and the Cover Type dataset. Both algorithms were implemented in MATLAB and all experiments were run on a 2.2 GHz Core2 Duo, 3GB RAM machine running Windows 7.

The Cover Type dataset has 581012 observations, each with 54 cartographic attributes. Each observation represents a 30×30 meter cell of land and the goal is to predict its forest cover type. Out of the 54 attributes, we removed all binary attributes to avoid sparsity, since sparsity in the design matrix causes the normality assumptions of the updates to fail and make the performance of our method unpredictable. We were left with a design matrix having 581012 observations with 10

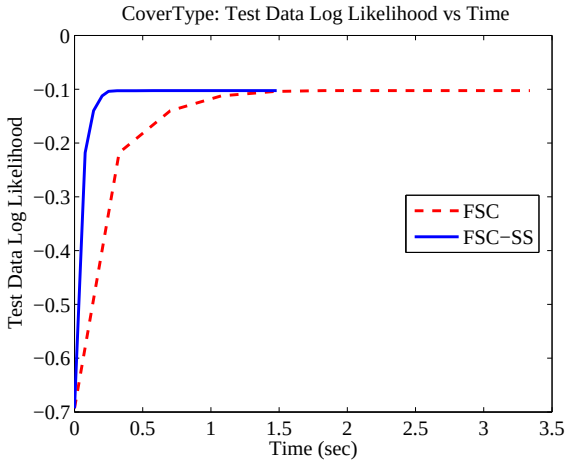
attributes each, to which we added the usual 'constant feature' of ones. Although the original task was to predict 1 out of 7 cover types, the task in our experiment was to build a one-versus-all classifier for one of these types. Thus, we chose to build a logistic regression classifier to determine whether the cover type for a given patch of land is 'Ponderosa Pine' or not. This gave us a dataset of 35754 positive instances and 545258 negative instances which we randomly split into a training set, consisting of 90% of the total data, and a test set, consisting of the remaining 10%.

The performance of the FSC and FSC-SS algorithms on the cover type dataset are shown in Figure 4.1. For FSC-SS, we set the value of ρ_t^*, the threshold for failing a hypothesis test, to be 0.01, and the initial sample size to be around 60000. We also added an L_2 regularization term with a weight of 0.01 for both the algorithms. In Figures 4.1a and 4.1b, we show how quickly FSC-SS is able to increase the log likelihood on the training and testing sets, as compared to the original FSC algorithm. Figures 4.1c and 4.1d show how the log-likelihood increases as a function of iteration. Note that the log-likelihood curves for both the algorithms are almost identical, supporting our intuition that we do not need to use all the available data in the earlier iterations to compute very accurate updates. In Figure 4.1e, we show sample size for FSC-SS as a function of iteration, and in Figure 4.1f, we compare the amount of time taken in each iteration for both the algorithms. Figures 4.1e and 4.1f give a good picture of how the FSC-SS algorithm is able to save on computation as compared to the original FSC algorithm.

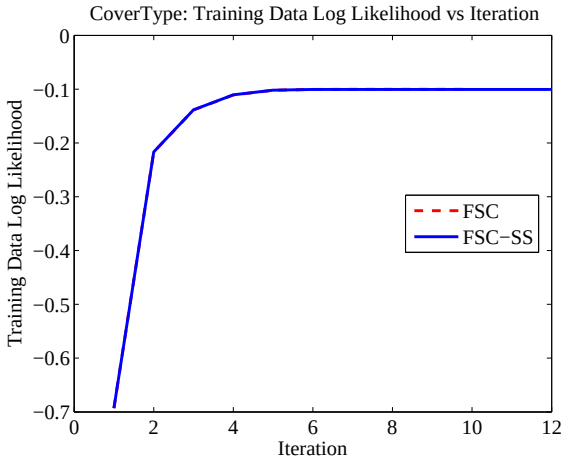
Our next set of experiments was on the Poker Hands dataset. Each observation in this dataset corresponds to a 5 card poker hand and the original task was to predict the type of a given hand as one of 13 hand types. We used the original testing set consisting of 1,000,000 examples, which we randomly split into a training set containing 90 % of the observations and a testing set containing the rest. There were 10 features for each observation, representing the suit and rank of each of the 5 cards, to which we added a constant feature of ones. As with the Cover Type dataset, the task in this experiment was to build a one-versus-all classifier for deciding whether a particular hand is a 'Four of a Kind' or not. This gave us a dataset consisting of 230 positive examples and 999770 negative examples. We used the same settings as for the experiments on the CoverType dataset. The performance of both the algorithms is summarized in Figure 4.2, and is similar to that on the Cover Type dataset.



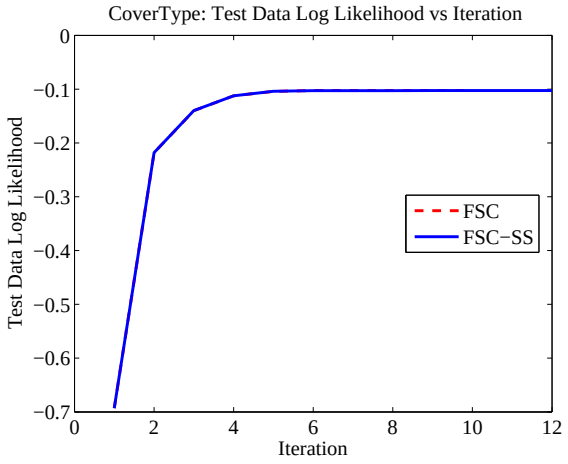
(a) Training Data Log Likelihood vs Time



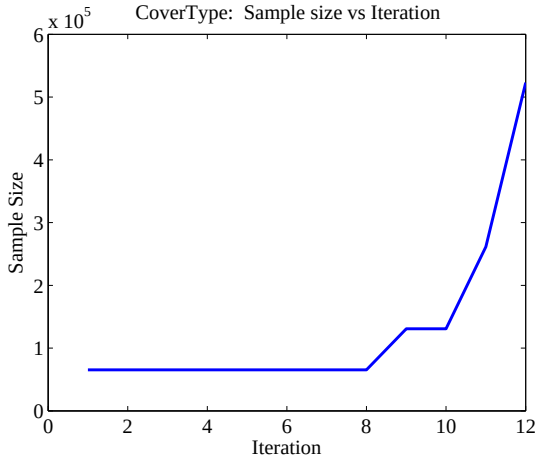
(b) Testing Data Log Likelihood vs Time



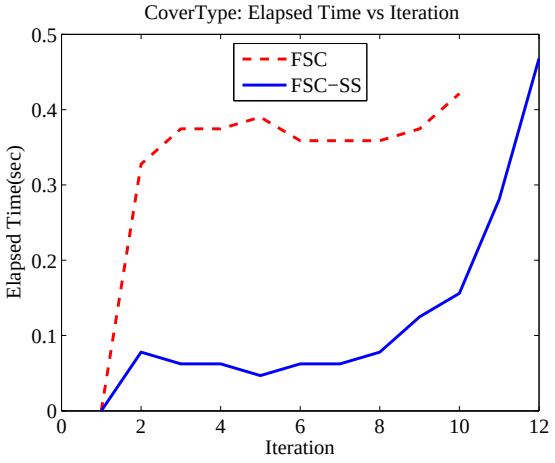
(c) Training Data Log Likelihood vs Iteration



(d) Testing Data Log Likelihood vs Iteration

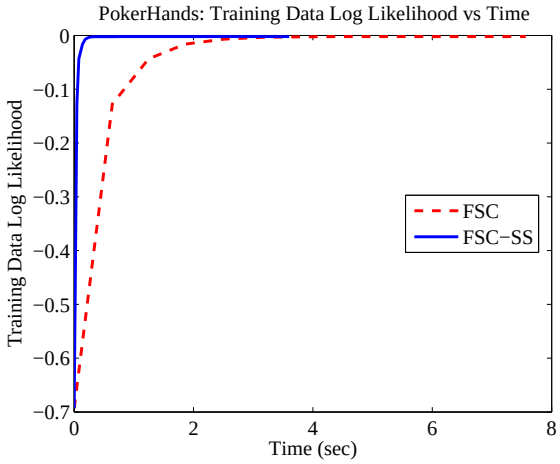


(e) Sample Size vs Iteration

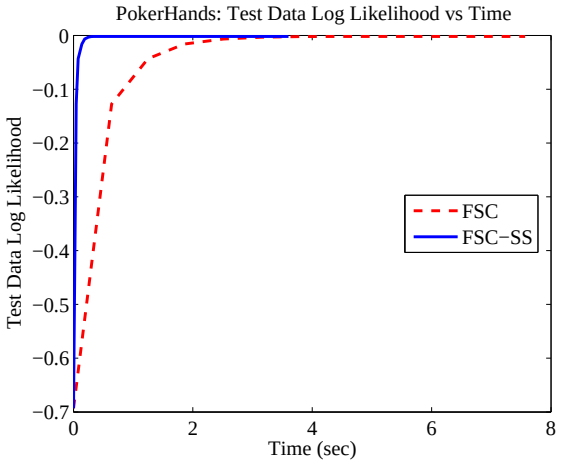


(f) Elapsed Time vs Iteration

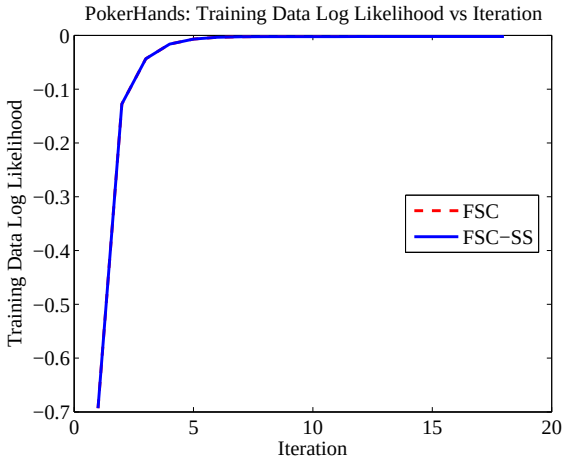
Figure 4.1: Performance of Fisher Scoring(FSC) versus its sub-sampled version(FSC-SS) for Logistic Regression on the Cover Type dataset



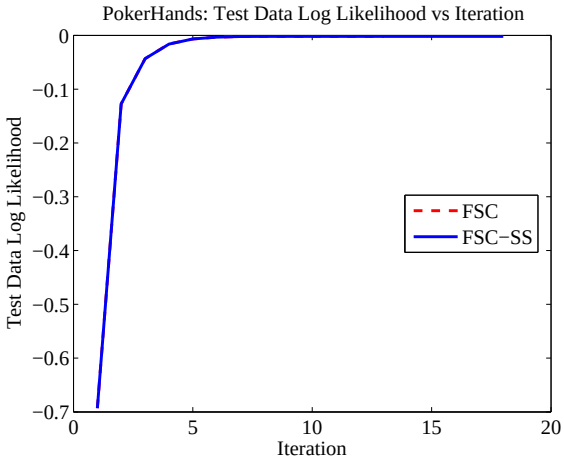
(a) Training Data Log Likelihood vs Time



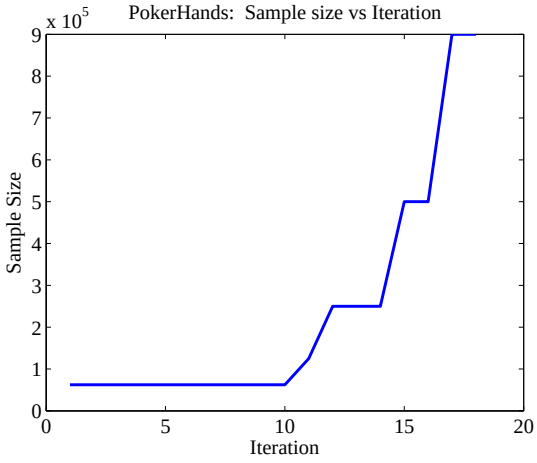
(b) Testing Data Log Likelihood vs Time



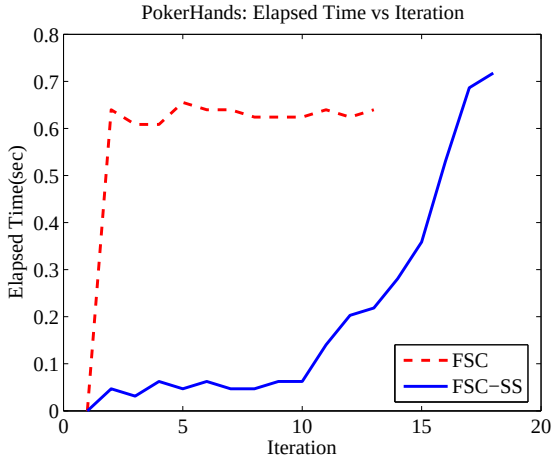
(c) Training Data Log Likelihood vs Iteration



(d) Testing Data Log Likelihood vs Iteration



(e) Sample Size vs Iteration



(f) Elapsed Time vs Iteration

Figure 4.2: Performance of Fisher Scoring(FSC) versus its sub-sampled version(FSC-SS) for Logistic Regression on the Poker Hands dataset

Chapter 5

Least Absolute Deviation Regression

5.1 Introduction

Consider the classical linear regression model, introduced in Chapter 4, which has the following generative model for the target, y_i, given the covariate vector, $\mathbf{x}_i \in \mathbb{R}^{P\times 1}$, and conditioned on the model parameters, $\boldsymbol{\beta} \in \mathbb{R}^{P\times 1}$:

$$y_i = \mathbf{x}_i^T \boldsymbol{\beta} + \epsilon_i \quad \text{where} \quad \epsilon_i \sim \mathcal{N}(0, \sigma^2) \tag{5.1}$$

Given a dataset $\mathcal{D}_N$ consisting of N statistical units $\{y_i, \mathbf{x}_i\}$, the goal in regression is to estimate the parameters $\boldsymbol{\beta}^*$ which makes the model best agree with the given data. This is commonly done by Maximum Likelihood(ML) estimation where we try to find the parameters which maximize the likelihood of the data. A crucial assumption of the classical linear regression model is that the errors ϵ_i are i.i.d. normally distributed random variables with zero mean and constant variance. However, the assumption of constant variance of the error across observations is not often valid in real world datasets and makes the resulting ML solution very sensitive to outliers.

In Least Absolute Deviation (LAD) regression, we relax the classical linear regression model by allowing the error ϵ_i to have different variances across observations. The generative process for the LAD regression model is:

$$y_i = \mathbf{x}_i^T \boldsymbol{\beta} + \frac{\sigma}{\sqrt{2}v_i}\epsilon_i \quad \text{where} \quad \epsilon_i \sim \mathcal{N}(0, 1) \tag{5.2}$$

Here, σ and $\boldsymbol{\beta}$ are the parameters of the model that has to be estimated from the data. We consider ϵ_i to be standard normally distributed and the error $\frac{\sigma}{\sqrt{2}v_i}\epsilon_i$ is allowed to have different variances across observations because of the dependence on v_i. We assume that the v_i's are positive and independent of the ϵ_i's. Also, we assume

that $\frac{1}{2v_i^2}$ has a standard exponential distribution so that the pdf of v_i is given by $p_v(v_i) = \frac{1}{v_i^3}\exp\{-\frac{1}{2v_i^2}\}$. From Eqn. (5.2) we see that the conditional distribution of y_i given $\mathbf{x}_i$ and v_i is:

$$p_{y|v,\mathbf{x}}(y_i|v_i, \mathbf{x}_i; \boldsymbol{\beta}, \sigma) = \frac{v_i}{\sqrt{\pi}\sigma}\exp\left\{-\frac{(y_i - \mathbf{x}_i^T\boldsymbol{\beta})v_i^2}{\sigma^2}\right\} \tag{5.3}$$

Also, we can write down the joint distribution of y_i and v_i given $\mathbf{x}_i$ as:

$$p_{y,v|\mathbf{x}}(y_i, v_i|\mathbf{x}_i; \boldsymbol{\beta}, \sigma) = p_{y|v,\mathbf{x}}(y_i|v_i, \mathbf{x}_i; \boldsymbol{\beta}, \sigma)p_v(v_i) = \frac{1}{\sqrt{\pi}\sigma v_i^2}\exp\left\{-\frac{1}{2v_i^2} - \frac{(y_i - \mathbf{x}_i^T\boldsymbol{\beta})^2 v_i^2}{\sigma^2}\right\} \tag{5.4}$$

Note however that we do not observe the v_i's directly. Thus, to perform maximum likelihood estimation of the parameters, we have to compute the conditional distribution of y_i given $\mathbf{x}_i$ by marginalizing out v_i from equation Eqn. 5.4. This is a scale mixture of normal distributions and can be shown to be the Laplacian (double exponential) distribution[1], i.e.:

$$\begin{aligned} p_{y|\mathbf{x}}(y_i|\mathbf{x}_i; \boldsymbol{\beta}, \sigma) &= \int_0^\infty p_{y,v|\mathbf{x}}(y_i, v|\mathbf{x}_i; \boldsymbol{\beta}, \sigma)dv = \int_0^\infty p_{y|v,\mathbf{x}}(y_i|v, \mathbf{x}_i; \boldsymbol{\beta}, \sigma)p_v(v)dv \\ &= \frac{1}{\sqrt{2}\sigma}\exp\left\{-\sqrt{2}\frac{|y_i - \mathbf{x}_i^T\boldsymbol{\beta}|}{\sigma}\right\} \end{aligned} \tag{5.5}$$

Now we can write down the log likelihood of the dataset $\mathcal{D}_N$ as:

$$\ell(\boldsymbol{\beta}, \sigma; \mathcal{D}_N) = \sum_{i=1}^N \log p_{y|\mathbf{x}}(y_i|\mathbf{x}_i; \boldsymbol{\beta}, \sigma) = -N\log(\sqrt{2}\sigma) - \frac{\sqrt{2}}{\sigma}\sum_{i=1}^N |y_i - \mathbf{x}_i^T\boldsymbol{\beta}| \tag{5.6}$$

Maximizing this log likelihood with respect to $\boldsymbol{\beta}$ is equivalent to minimizing $\sum_{i=1}^N |y_i - \mathbf{x}_i^T\boldsymbol{\beta}|$,i.e. the absolute value of prediction errors, and hence this is known as the Least Absolute Deviation regression model. However, maximizing the log likelihood in Eqn. (5.6) is hard and is typically performed using the Expectation-Maximization (EM) algorithm[16, 6].

We will now review the EM algorithm in Section 5.2. In Section 5.3, we describe the EM algorithm for parameter estimation in Least Absolute Deviation regression and show that it is essentially an IRLS algorithm. Then, in Section 5.4, we will describe how our sub sampling method can be used to optimize this algorithm. Finally we present our experimental results on two real world datasets in Section 5.5

5.2 The Expectation-Maximization (EM) Algorithm

The Expectation-Maximization(EM) algorithm[6] is a method that can be used to find maximum likelihood (ML) estimates of model parameters, $\mathbf{\Theta}$, given incomplete data, $\mathbf{D}$. By incomplete data, we mean that the model depends on certain latent variables, $\mathbf{Z}$, which are not directly observable. ML estimation involves finding the parameters $\mathbf{\Theta}$ which maximize the following likelihood:

$$L(\mathbf{\Theta}; \mathbf{D}) = p(\mathbf{D}; \mathbf{\Theta}) = \sum_{\mathbf{Z}} p(\mathbf{D}, \mathbf{Z}; \mathbf{\Theta}) \tag{5.7}$$

The function $L(\mathbf{\Theta}; \mathbf{D})$ is called the *incomplete* data likelihood, since the latent variables are not known. The EM algorithm is useful when the incomplete data likelihood is intractable, but the *complete* data likelihood, $L(\mathbf{\Theta}; \mathbf{D}, \mathbf{Z}) = p(\mathbf{D}, \mathbf{Z}; \mathbf{\Theta})$, is easier to work with. The EM algorithm finds a local maxima of the incomplete data likelihood by alternating between the following two steps till convergence: First, in the 'E Step', we compute the expectation of the complete log likelihood with respect to the conditional distribution of the latent variables given the data and conditioned on the current estimate of the parameters. Then in the 'M step', we find the parameters which maximize this expected complete log likelihood. This is shown in Algorithm 8. A more detailed treatment of the EM algorithm can be found, for instance, in [6, 3].

Algorithm 8 Expectation-Maximization (EM) Algorithm

Require: Dataset $\mathbf{D}$
Ensure: Parameter estimate $\hat{\mathbf{\Theta}}^*$
1: Initialize $\mathbf{\Theta}_0^*$
2: $t \leftarrow 1$
3: **repeat**
4: E Step:

$$Q(\mathbf{\Theta}|\mathbf{\Theta}_{t-1}^*) \leftarrow \mathbb{E}_{\mathbf{Z}|\mathbf{X};\mathbf{\Theta}_{t-1}^*} \log L(\mathbf{\Theta}; \mathbf{D}, \mathbf{Z}) \tag{5.8}$$

5: M Step:

$$\hat{\mathbf{\Theta}}^* \leftarrow \mathbf{\Theta}_t^* \leftarrow \arg\max_{\mathbf{\Theta}} Q(\mathbf{\Theta}|\mathbf{\Theta}_{t-1}^*) \tag{5.9}$$

6: $t \leftarrow t + 1$
7: **until** convergence
8: **return** $\hat{\mathbf{\Theta}}^*$

5.3 EM Algorithm for LAD regression

We will now describe the EM algorithm for parameter estimation in LAD regression [16]. Here, the incomplete data $\mathbf{D}$ is the dataset $\mathcal{D}_N$ consisting of N statistical units $\{y_i, \mathbf{x}_i\}$, the parameters are $\boldsymbol{\Theta} = \{\boldsymbol{\beta}, \sigma\}$ and we consider the v_i's to be the hidden variables $\mathbf{Z}$. If we have knowledge of the latent variables $\mathcal{V}_N = \{v_i\}_{i=1}^N$ in addition to the dataset $\mathcal{D}_N$, we can write down the complete data log likelihood (using Eqn. 5.4) as:

$$\ell(\boldsymbol{\beta}, \sigma; \mathcal{D}_N, \mathcal{V}_N) = \sum_{i=1}^{N} \log p_{y,v|x}(y_i, v_i|x_i; \boldsymbol{\beta}, \sigma) = -N \log \sigma - \frac{1}{\sigma^2} \sum_{i=1}^{N} (y_i - \mathbf{x}_i^T \boldsymbol{\beta})^2 v_i^2 + C \tag{5.10}$$

Here C represents the terms that does not depend on the parameters $\boldsymbol{\beta}$ and σ. Now, in the E step of the t^{th} iteration, we compute the expectation of this log likelihood under the conditional distribution of the latent variables $\mathcal{V}_N$ given the data $\mathcal{D}_N$ using the estimate of the parameters $\boldsymbol{\beta}_{t-1}^*$ and σ_{t-1}^*. This is given by:

$$\begin{aligned} Q(\boldsymbol{\beta}, \sigma|\boldsymbol{\beta}_{t-1}^*, \sigma_{t-1}^*) &\leftarrow \sum_{i=1}^{N} \mathbb{E}_{v_i|y_i, \mathbf{x}_i; \boldsymbol{\beta}_{t-1}^*, \sigma_{t-1}^*} \left[\log p_{y,v|x}(y_i, v_i|x_i; \boldsymbol{\beta}, \sigma)\right] \\ &= -N \log \sigma - \frac{1}{\sigma^2} \sum_{i=1}^{N} (y_i - \mathbf{x}_i^T \boldsymbol{\beta})^2 \mathbb{E}[v_i^2|y_i, \mathbf{x}_i; \boldsymbol{\beta}_{t-1}^*, \sigma_{t-1}^*] + C \end{aligned} \tag{5.11}$$

It can be shown[16] that the conditional expectation of v_i^2 is:

$$\mathbb{E}[v_i^2|y_i, \mathbf{x}_i; \boldsymbol{\beta}_{t-1}^*, \sigma_{t-1}^*] = \frac{\sigma_{t-1}^*}{\sqrt{2}|y_i - \mathbf{x}_i^T \boldsymbol{\beta}_{t-1}^*|} \tag{5.12}$$

Now, in the M step, we find the parameters $\{\boldsymbol{\beta}_t^*, \sigma_t^*\}$ which maximize $Q(\boldsymbol{\beta}, \sigma|\boldsymbol{\beta}_{t-1}^*, \sigma_{t-1}^*)$, the expected complete log likelihood calculated in the E step. Maximizing this with respect to $\boldsymbol{\beta}$ we have:

$$\begin{aligned} \boldsymbol{\beta}_t^* &= u(\boldsymbol{\beta}_{t-1}^*, \mathcal{D}_N, t) = \arg\min_{\boldsymbol{\beta}} \sum_{i=1}^{N} w_i (y_i - \mathbf{x}_i^T \boldsymbol{\beta})^2 \\ \text{where } w_i &= \mathbb{E}[v_i^2|y_i, \mathbf{x}_i; \boldsymbol{\beta}_{t-1}^*, \sigma_{t-1}^*] = \frac{\sigma_{t-1}^*}{\sqrt{2}|y_i - \mathbf{x}_i^T \boldsymbol{\beta}_{t-1}^*|} \end{aligned} \tag{5.13}$$

The update for $\boldsymbol{\beta}_t^*$ is a solution to a weighted least squares problem and can be alternately written as an IRLS update $\boldsymbol{\beta}_t^* \leftarrow (\mathbf{X}^T \mathbf{W}_{t-1} \mathbf{X})^{-1} \mathbf{X}^T \mathbf{W}_{t-1} \mathbf{y}$. Here $\mathbf{X} \in \mathbb{R}^{N \times P}$ represents the design matrix whose i^{th} row is $\mathbf{x}_i^T$, $\mathbf{y} \in \mathbb{R}^{N \times 1}$ is a vector of targets whose i^{th} component is y_i and $\mathbf{W}_{t-1}$ is a diagonal matrix of weights with diagonal elements given by $\mathbf{W}_{t-1}[i, i] = w_i$. Note that the factor $\sigma_{t-1}^*/\sqrt{2}$ in $\mathbf{W}_{t-1}$ cancels out

in the update formula for $\boldsymbol{\beta}_t^*$ and hence we can equivalently write the weights as:

$$\mathbf{W}_{t-1}[i,i] = \frac{1}{|y_i - \mathbf{x}_i^T \boldsymbol{\beta}_{t-1}^*|} \tag{5.14}$$

Now, since the updates for $\boldsymbol{\beta}_t^*$ are independent of the estimates for σ^*, we need not compute a σ_t^* in every iteration. Instead, we can directly compute a final estimate $\hat{\sigma}^*$ after the estimates for $\boldsymbol{\beta}_t^*$ has converged. Maximizing $Q(\boldsymbol{\beta}, \sigma | \boldsymbol{\beta}_{t-1}^*, \sigma_{t-1}^*)$ with respect to σ, we see that:

$$(\hat{\sigma}^*)^2 = \frac{2}{N} \sum_{i=1}^{N} (y_i - \mathbf{x}_i^T \boldsymbol{\beta})^2 \mathbb{E}[v_i^2 | y_i, \mathbf{x}_i; \boldsymbol{\beta}_{t-1}^*, \sigma_{t-1}^*] \tag{5.15}$$

Substituting the formula for $\mathbb{E}[v_i^2 | y_i, \mathbf{x}_i; \boldsymbol{\beta}_{t-1}^*, \sigma_{t-1}^*]$ from , we get:

$$\hat{\sigma}^* = \frac{\sqrt{2}}{N} \sum_{i=1}^{N} |y_i - \mathbf{x}_i^T \hat{\boldsymbol{\beta}}^*| \tag{5.16}$$

where $\hat{\boldsymbol{\beta}}^*$ is the final estimate of $\boldsymbol{\beta}$ after convergence. The complete algorithm for parameter estimation in the LAD regression model is shown in Algorithm 9 which uses Procedure 10 as the update function.

Algorithm 9 EM Algorithm for LAD regression

Require: Dataset $\mathcal{D}_N = \{\mathbf{y} \in \mathbb{R}^{N \times 1}, \mathbf{X} \in \mathbb{R}^{N \times P}\}$
Ensure: Parameter estimate $\hat{\boldsymbol{\beta}}^* \in \mathbb{R}^{P \times 1}$, $\hat{\sigma^*}$
1: Initialize $\boldsymbol{\beta}_0^* \in \mathbb{R}^{P \times 1}$
2: $t \leftarrow 1$
3: **repeat**
4: $\quad \hat{\boldsymbol{\beta}}^* \leftarrow \boldsymbol{\beta}_t^* \leftarrow$ EM-LAD-UPDATE$(\mathcal{D}_N, \boldsymbol{\beta}_{t-1}^*)$
5: $\quad t \leftarrow t + 1$
6: **until** convergence
7: $\hat{\sigma}^* \leftarrow \frac{\sqrt{2}}{N} \sum_{i=1}^{N} |y_i - \mathbf{x}_i^T \hat{\boldsymbol{\beta}}^*|$
8: **return** $\hat{\boldsymbol{\beta}}^*$, $\hat{\sigma}^*$

5.4 Statistical Optimization of EM for LAD regression

Since the EM algorithm for LAD regression reduces to an IRLS algorithm, its optimized version is almost identical to the optimized Fisher Scoring algorithm (Algo-

Procedure 10 EM-LAD-UPDATE

Require: Dataset: $\mathcal{D}_N = \{\mathbf{y} \in \mathbb{R}^{N\times 1}, \mathbf{X} \in \mathbb{R}^{N\times P}\}$, Old Parameter Estimate: $\boldsymbol{\beta}^*_{\text{old}} \in \mathbb{R}^{P\times 1}$

Ensure: New Parameter estimate: $\boldsymbol{\beta}^*_{\text{new}} \in \mathbb{R}^{P\times 1}$

$\mathbf{w} \leftarrow \dfrac{1}{|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}^*_{\text{old}}|}$
$\mathbf{W} \leftarrow \text{diag}(\mathbf{w})$
$\tilde{\mathbf{X}} \leftarrow \mathbf{W}^{1/2}\mathbf{X}$
$\tilde{\mathbf{y}} \leftarrow \mathbf{W}^{1/2}\mathbf{y}$
$\boldsymbol{\beta}^*_{\text{new}} \leftarrow (\tilde{\mathbf{X}}^T\tilde{\mathbf{X}})^{-1}\tilde{\mathbf{X}}^T\tilde{\mathbf{y}}$
return $\boldsymbol{\beta}^*_{\text{new}}$

rithm 7) that we developed for parameter estimation in Generalized Linear Models. However, for the sake of completeness of this chapter, we show this in Algorithm 11.

Algorithm 11 EM Algorithm for LAD regression with Sub-Sampling

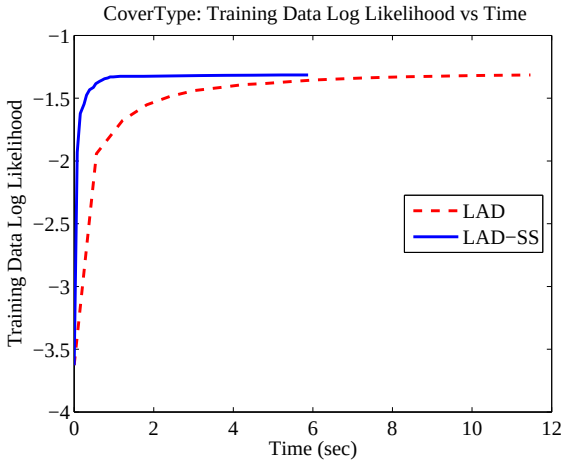
Require: Dataset $\mathcal{D}_N = \{\mathbf{y} \in \mathbb{R}^{N\times 1}, \mathbf{X} \in \mathbb{R}^{N\times P}\}$, Threshold ρ^*
Ensure: Parameter Estimate $\hat{\boldsymbol{\beta}}^*$, $\hat{\sigma}^*$

Initialize $\boldsymbol{\beta}^*_0$
Initialize $N_1 \ll N$
$\{\mathbf{y}_1, \mathbf{X}_1\} \leftarrow$ Sample N_1 statistical units from $\{\mathbf{y}, \mathbf{X}\}$
$t \leftarrow 1$
done $\leftarrow$ **false**
repeat
  $\mathbf{u}_t \leftarrow$ EM-LAD-UPDATE$(\{\mathbf{y}_t, \mathbf{X}_t\}, \boldsymbol{\beta}^*_{t-1})$
  $htPassed \leftarrow$ HYPOTHESIS-TEST$(\mathbf{X}_t, \mathbf{y}_t, \boldsymbol{\beta}^*_{t-1}, \rho^*)$
  if $htPassed$ **then**
    $\hat{\boldsymbol{\beta}}^* \leftarrow \boldsymbol{\beta}^*_t \leftarrow \mathbf{u}_t$
    $N_{t+1} \leftarrow N_t$
    $\{\mathbf{y}_{t+1}, \mathbf{X}_{t+1}\} \leftarrow \{\mathbf{y}_t, \mathbf{X}_t\}$
  else
    if $N_t = N$ **then**
      done $\leftarrow$ **true**
    else
      $\boldsymbol{\beta}^*_t \leftarrow \boldsymbol{\beta}^*_{t-1}$
      $N_{t+1} \leftarrow \min(N, 2N_t)$
      $\{\mathbf{y}_{t+1}, \mathbf{X}_{t+1}\} \leftarrow$ Sample N_{t+1} statistical units from $\{\mathbf{y}, \mathbf{X}\}$
    end if
  end if
  $t \leftarrow t + 1$
until *done*
$\hat{\sigma}^* \leftarrow \dfrac{\sqrt{2}}{N}\sum_{i=1}^{N} |y_i - \mathbf{x}_i^T\hat{\boldsymbol{\beta}}^*|$
return $\hat{\boldsymbol{\beta}}^*$, $\hat{\sigma}^*$

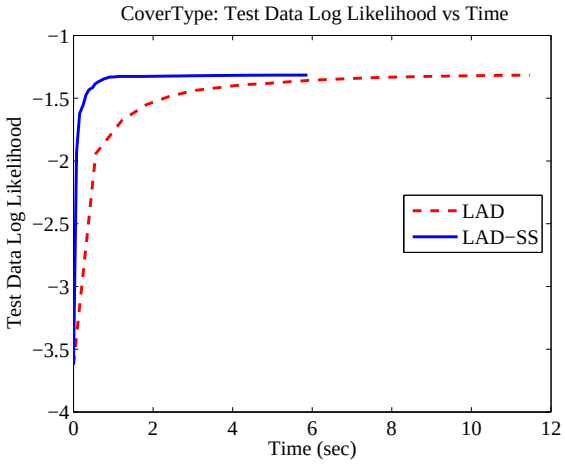
5.5 Experiments

As with the Fisher Scoring algorithm, we compared the performance of the EM algorithm for Least Absolute Deviation regression(LAD) versus its sub-sampled version (LAD-SS) on two real world datasets from the UCI Machine Learning Repository [7]: the Poker Hands dataset and the Cover Type dataset. Both algorithms were implemented in MATLAB and all experiments were run on a 2.2 GHz Core2 Duo, 3GB RAM machine running Windows 7.

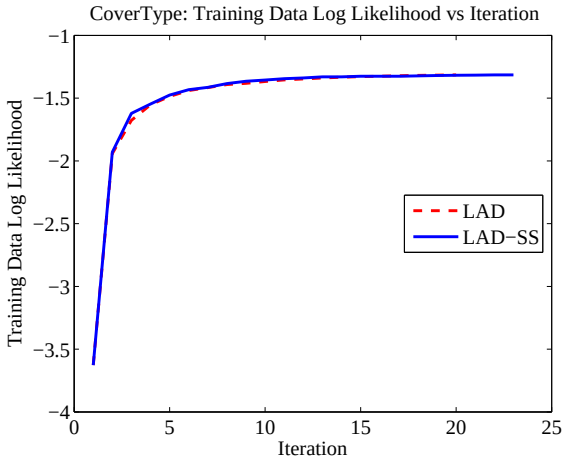
For a description of the datasets, please refer Section 4.5 of Chapter 4. In this experiment, instead of building a one-versus-all classifier as in Chapter 4, the task was to predict the the cover type as 1 of 7 types or the poker hand as 1 of 13 types, which we treated as a regression problem. For LAD-SS, we set the value of ρ_t^*, the threshold for failing a hypothesis test, to be 0.001, and the initial sample size to be around 60000. The performance of both algorithms are shown in Figures 5.1 and 5.2. The results are very similar to those for the Fisher scoring algorithm (see 4.5).



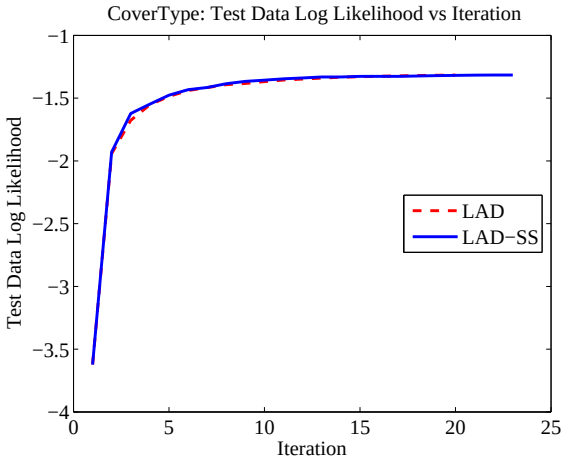
(a) Training Data Log Likelihood vs Time



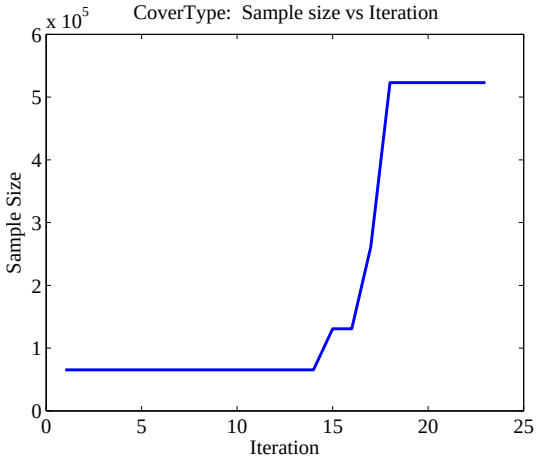
(b) Testing Data Log Likelihood vs Time



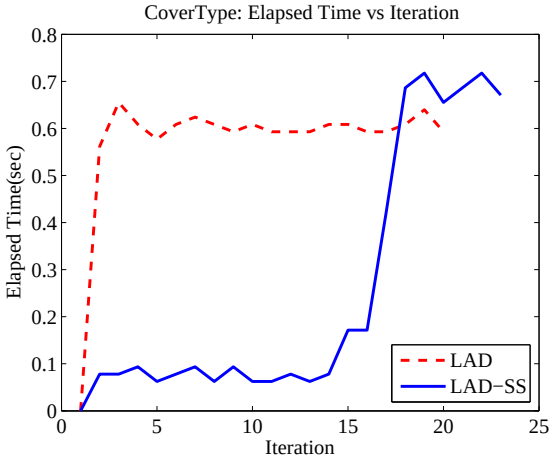
(c) Training Data Log Likelihood vs Iteration



(d) Testing Data Log Likelihood vs Iteration

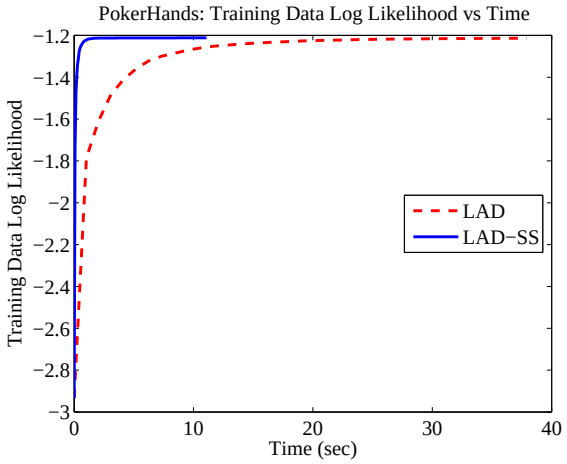


(e) Sample Size vs Iteration

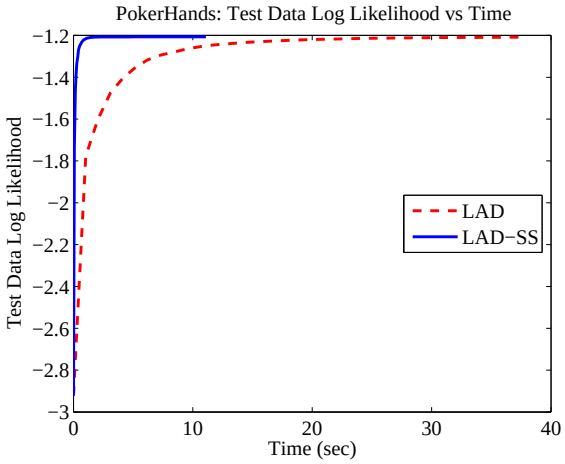


(f) Elapsed Time vs Iteration

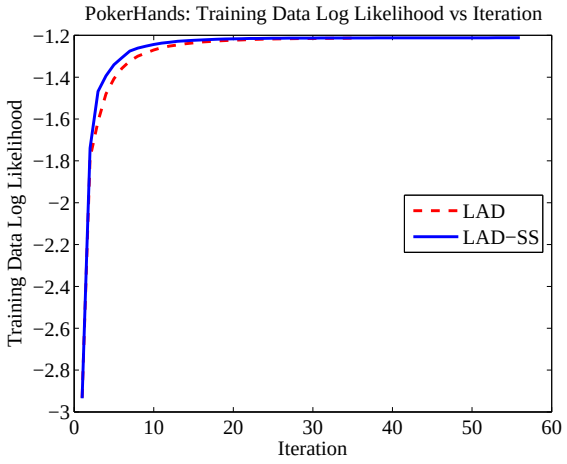
Figure 5.1: Performance of EM for LAD regression(LAD) versus its sub-sampled version(LAD-SS) on the Cover Type dataset



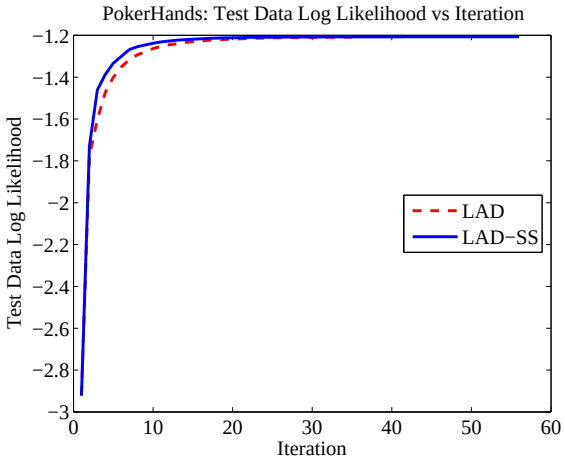
(a) Training Data Log Likelihood vs Time



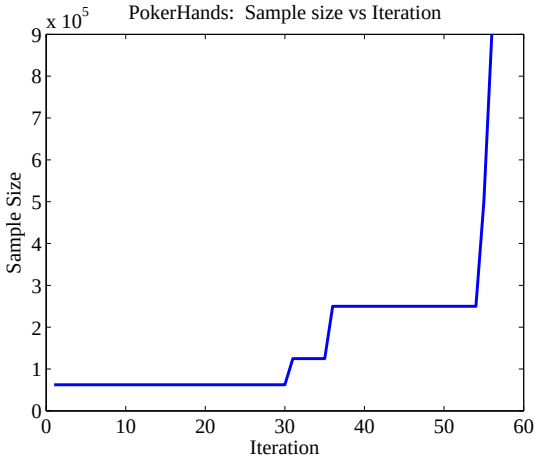
(b) Testing Data Log Likelihood vs Time



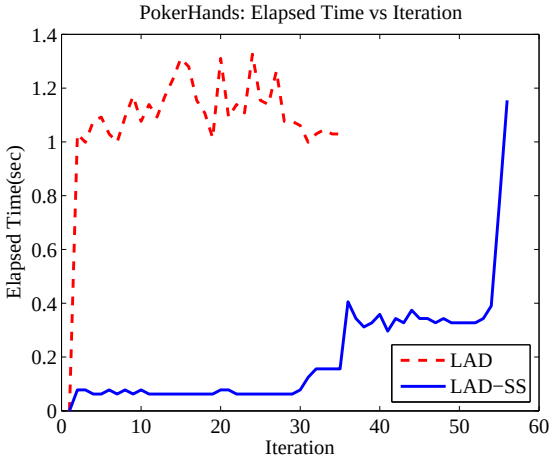
(c) Training Data Log Likelihood vs Iteration



(d) Testing Data Log Likelihood vs Iteration



(e) Sample Size vs Iteration



(f) Elapsed Time vs Iteration

Figure 5.2: Performance of EM for LAD regression(LAD) versus its sub-sampled version(LAD-SS) on the Poker Hands dataset

Chapter 6

Non-Negative Matrix Factorization

6.1 Introduction

Non-negative Matrix Factorization (NMF) is a popular dimensionality reduction technique that has many useful applications in machine learning and data mining. NMF is typically applied to high dimensional data where each element is non-negative and it provides a low rank approximation formed by factors whose elements are themselves non-negative. Unlike other dimensionality reduction techniques, the non-negativity constraints in NMF typically lead to a "sum of parts" decomposition of objects. NMF has been applied successfully to a variety of tasks in fields such as computer vision, audio analysis, text mining, spectral analysis and bioinformatics.

NMF can be formulated mathematically as follows. Given an input matrix $\mathbf{X} \in \mathbb{R}^{R \times S}$ where each element is nonnegative and an integer $K < \min\{R, S\}$, NMF aims to find two factors $\mathbf{W}^* \in \mathbb{R}^{R \times K}$ and $\mathbf{H}^* \in \mathbb{R}^{K \times S}$ with nonnegative elements such that $\mathbf{X} \approx \mathbf{W}^*\mathbf{H}^*$. The factors $\mathbf{W}^*$ and $\mathbf{H}^*$ are commonly found by solving the following non-convex optimization problem:

$$\{\mathbf{W}^*, \mathbf{H}^*\} = \underset{\mathbf{W},\mathbf{H} \geq 0}{\arg\min} f(\mathbf{W}, \mathbf{H}) = \|\mathbf{X} - \mathbf{W}\mathbf{H}\|_F^2 \tag{6.1}$$

There are no closed form solutions for $\{\mathbf{W}^*, \mathbf{H}^*\}$ and one has to resort to iterative methods to solve for these factors. Algorithms based on the Alternating Non-negative Least Squares (ANLS) framework have been found to be very efficient for this problem and has convergence properties provided by a block coordinate descent argument[2]. ANLS is a simple alternating minimizing scheme that iteratively optimizes each of

the factors keeping the other fixed:

$$\mathbf{H} \leftarrow \underset{\mathbf{H} \geq 0}{\arg\min} \|\mathbf{X} - \mathbf{W}\mathbf{H}\|_F^2 \tag{6.2a}$$

$$\mathbf{W} \leftarrow \underset{\mathbf{W} \geq 0}{\arg\min} \|\mathbf{X}^\mathbf{T} - \mathbf{H}^T\mathbf{W}^T\|_F^2 \tag{6.2b}$$

Although the original problem in Eqn. 6.1 is non-convex, the problems in Eqns. 6.2 are convex. Note that the problem in Eqn. 6.2a can be broken down into S algebraically independent sub-problems with respect to each column of $\mathbf{X}$ and $\mathbf{H}$, and similarly the problem in Eqn. 6.2b can be broken down into R independent sub-problems with respect to each column of $\mathbf{X}^T$ and $\mathbf{W}^T$. Thus we can write down the general structure of the ANLS algorithm for NMF as in Algorithm 12.

Algorithm 12 ANLS Framework for NMF

Require: Dataset $\mathcal{D}_{R,S} = \{\mathbf{X} \in \mathbb{R}^{R \times S}\}$, $K < \min\{R, S\}$
Ensure: Parameter estimates $\hat{\mathbf{W}}^* \in \mathbb{R}^{R \times K}, \hat{\mathbf{H}}^* \in \mathbb{R}^{K \times S}$
Initialize $\mathbf{W}_0^* \in \mathbb{R}^{R \times K} \geq 0$
$t \leftarrow 1$
repeat
  for $i = 1$ **to** S **do**

$$\hat{\mathbf{H}}^*[:, i] \leftarrow \mathbf{H}_t^*[:, i] \leftarrow \underset{\mathbf{h} \geq 0}{\arg\min} \|\mathbf{W}_{t-1}^*\mathbf{h} - \mathbf{X}[:, i]\|_2^2 \tag{6.3}$$

  end for
  for $j = 1$ **to** R **do**

$$\hat{\mathbf{W}}^{*T}[:, j] \leftarrow \mathbf{W}_t^{*T}[:, j] \leftarrow \underset{\mathbf{w} \geq 0}{\arg\min} \|\mathbf{H}_t^{*T}\mathbf{w} - \mathbf{X}^T[:, j]\|_2^2 \tag{6.4}$$

  end for
  $t \leftarrow t + 1$
until convergence
return $\hat{\mathbf{W}}^*, \hat{\mathbf{H}}^*$

Note that the sub-problems in Eqns. 6.3 and 6.4 are just least squares problems with an additional non-negativity constraint on the solution. These problems are called Non Negative Least Squares (NNLS) and, unlike unconstrained least squares, are much harder to solve. A number of methods have been proposed for this, starting with the classical algorithm by Lawson and Hanson [13]. Lin [14] developed a gradient descent based algorithm with a projection on to the nonnegative orthant. D. Kim et al.[8] suggested a quasi-Newton method with projections for faster convergence. H. Kim and Park[9] studied an improved active-set algorithm, and J. Kim and Park [10] proposed a more efficient active-set algorithm based on the Block Principal Pivoting

method.

Since the most efficient NNLS algorithm is the Block Principal Pivoting method, we use it as the backbone for our optimization procedure. We describe the BPP method in Section 6.2. Then, in Section 6.3, we describe how to use our sub-sampling strategy to optimize the BPP algorithm for NMF[1]. Finally, in section 6.4 we present experimental results on three real world datasets.

6.2 Block Principal Pivoting(BPP) algorithm

This section describes the BPP algorithm for solving NNLS problems, which have the following general form:

$$\min_{\mathbf{y}\geq 0} \|\mathbf{C}\mathbf{y} - \mathbf{d}\|_2^2 \tag{6.5}$$

where we assume $\mathbf{C} \in \mathbb{R}^{N\times P}$, $\mathbf{y} \in \mathbb{R}^{P\times 1}$ and $\mathbf{d} \in \mathbb{R}^{N\times 1}$. If $\mathbf{C}$ has full rank, the matrix $\mathbf{C}^T\mathbf{C}$ is positive definite and the problem in Eqn. (6.5) is strictly convex. Then, a solution $\mathbf{y}^*$ that satisfies the following Karush-Kuhn-Tucker (KKT) conditions is the optimal solution to Eqn. (6.5).

$$\boldsymbol{\lambda}^* = \mathbf{C}^T\mathbf{C}\mathbf{y}^* - \mathbf{C}^T\mathbf{d} \tag{6.6a}$$
$$\boldsymbol{\lambda}^* \geq 0 \tag{6.6b}$$
$$\mathbf{y}^* \geq 0 \tag{6.6c}$$
$$\boldsymbol{\lambda}_i^*\mathbf{y}_i^* = 0, i = 1, ..., P \tag{6.6d}$$

Here $\boldsymbol{\lambda} \in \mathbb{R}^{P\times 1}$ represents the Lagrange multipliers. To solve for $\mathbf{y}^*$, the index set $1, ..., P$ is first divided into two sets F and G where $F \cup G = 1, ..., P$ and $F \cap G = \phi$. Let $\mathbf{y}_F$, $\mathbf{y}_G$, $\boldsymbol{\lambda}_F$ and $\boldsymbol{\lambda}_G$ denote subsets of $\mathbf{y}$ and $\boldsymbol{\lambda}$, obtained from components with indices in F and G respectively, and similarly let $\mathbf{C}_F$ and $\mathbf{C}_G$ represent the sub-matrices of $\mathbf{C}$, obtained from columns with indices in F and G respectively. Initially, $\mathbf{y}_G$ and $\boldsymbol{\lambda}_F$ are set to $\mathbf{0}$. By construction, $\mathbf{y} = [\mathbf{y}_F; \mathbf{y}_G]$ and $\boldsymbol{\lambda} = [\boldsymbol{\lambda}_F; \boldsymbol{\lambda}_G]$ satisfy Eqn. (6.6d). We can solve for $\mathbf{y}_F$ and $\boldsymbol{\lambda}_G$ using Eqn. (6.6a):

$$\mathbf{y}_F = (\mathbf{C}_F^T\mathbf{C}_F)^{-1}\mathbf{C}_F^T\mathbf{d} \tag{6.7a}$$
$$\boldsymbol{\lambda}_G = \mathbf{C}_G^T\mathbf{C}_F\mathbf{y}_F - \mathbf{C}_G^T\mathbf{d} \tag{6.7b}$$

If $\mathbf{y}_F \geq \mathbf{0}$ and $\boldsymbol{\lambda}_G \geq \mathbf{0}$, then $\mathbf{y}^* = [\mathbf{y}_F; \mathbf{0}]$ is an optimal solution and the algorithm terminates. Otherwise, the index sets F and G are updated by exchanging variables for which Eqn. (6.6b) or Eqn. (6.6c) does not hold, and the process is repeated till a feasible solution $\mathbf{y}^*$ is found. Details of this method and its efficient extension to the

[1]By BPP algorithm for NMF, we mean an algorithm where BPP is used to solve NMF's NNLS sub-problems

case of multiple $\mathbf{d}$'s (as in Eqns. (6.2)) can be found in [10].

From now, we will write BPP-SOLVE to represent a procedure that takes in as input $\mathbf{C} \in \mathbb{R}^{N \times P}$, $\mathbf{d} \in \mathbb{R}^{N \times 1}$ and uses the BPP algorithm to produce output $\mathbf{y}^* = \arg\min_{\mathbf{y} \geq 0} \|\mathbf{C}\mathbf{y} - \mathbf{d}\|_2^2$. For instance, we can write steps 5 and 8 of Algorithm 12 as:

$$\hat{\mathbf{H}}^*[:, i] \leftarrow \mathbf{H}_t^*[:, i] \leftarrow \text{BPP-SOLVE}(\mathbf{W}_{t-1}^*, \mathbf{X}[:, i]) \tag{6.8a}$$

$$\hat{\mathbf{W}}^{*T}[:, j] \leftarrow \mathbf{W}_t^{*T}[:, j] \leftarrow \text{BPP-SOLVE}(\mathbf{H}_t^{*T}, \mathbf{X}^T[:, j]) \tag{6.8b}$$

Replacing steps 5 and 8 of Algorithm 12 with the calls to BPP-SOLVE as shown above gives us the BPP algorithm for NMF.

6.3 Statistical Optimization of the NMF BPP algorithm

Now, we will describe how our sub-sampling method can be used to optimize the NMF BPP algorithm for large input matrices [11]. Note that in every iteration of ANLS, we make a large number of calls to the BPP-SOLVE procedure to solve the $R + S$ NNLS problems. Also, each call to BPP-SOLVE computes $\mathbf{C}_F^T\mathbf{C}_F$ and $\mathbf{C}_F^T\mathbf{d}$ to solve for $\mathbf{y}^*$ and can be very expensive for large input matrices. Thus, we can significantly optimize the NMF BPP algorithm if we can reduce the number of calls to BPP-SOLVE and reduce the computational cost of BPP-SOLVE. Our sub-sampling method can achieve both of these optimizations.

Before explaining our optimization method in detail, we will first give a quick overview. In the initial iterations of ANLS, the estimates of the optimal factors $\hat{\mathbf{W}}^*$ and $\hat{\mathbf{H}}^*$ are far from their true values. Therefore, instead of using BPP-SOLVE with all the available data to estimate the updates to $\hat{\mathbf{W}}^*$ and $\hat{\mathbf{H}}^*$, we can approximate the updates using a small subset of the available data. Since these updates (see Eqn. (6.7a)) have the form $(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b}$, we can use the hypothesis testing framework of Chapter 3 to determine the reliability of these updates. As learning proceeds, and our estimates of $\hat{\mathbf{W}}^*$ and $\hat{\mathbf{H}}^*$ move closer to their true solution, more precision is needed in the updates and hence the hypothesis tests fail. When this happens, we can increase the sample size used to compute the updates, so that the updates become more precise and pass the hypothesis tests. Finally, we stop optimizing when the hypothesis tests fail while using all the data.

Now we will explain this in more detail. In each iteration t, instead of using all available data to accurately compute $\mathbf{H}_t^*[:, i]$ as in Eqn. (6.3), we will approximate $\mathbf{H}_t^*[:, i]$ using only N_t^H rows of $\mathbf{W}_{t-1}^*$ and the corresponding N_t^H components of $\mathbf{X}[:, i]$. Similarly, instead of computing $\mathbf{W}_t^{*T}[:, j]$ as in Eqn. (6.4), we will approximate it using N_t^W rows of $\mathbf{H}_t^{*T}$ and the corresponding components of $\mathbf{X}^T[:, j]$. Thus, we will replace

the calls shown in Eqns. (6.8) as follows:

$$\hat{\mathbf{H}}^*[:,i] \leftarrow \mathbf{H}_t^*[:,i] \leftarrow \text{BPP-SOLVE}(\mathbf{W}_{t-1}^*[1{:}N_t^H,:], \mathbf{X}[1{:}N_t^H,i]) \tag{6.9a}$$

$$\hat{\mathbf{W}}^{*T}[:,j] \leftarrow \mathbf{W}_t^{*T}[:,j] \leftarrow \text{BPP-SOLVE}(\mathbf{H}_t^{*T}[1{:}N_t^W,:], \mathbf{X}^T[1{:}N_t^W,j]) \tag{6.9b}$$

Since, the updates (Eqn. (6.7a)) computed using BPP-SOLVE have a form similar to the solution to a least square problem, the updates produced by the calls to BPP-SOLVE shown in Eqns. (6.9) are a valid approximation of the true updates (Eqns. (6.8)). We test the reliability of updates using Procedure 3 (HYPOTHESIS-TEST) and if the updates to $\mathbf{H}_t^*[:,i]$ or $\mathbf{W}_t^{*T}[:,j]$ fail these tests, we increase N_t^H or N_t^W respectively, for the next iteration.

In practice, we will not test updates for all columns of $\mathbf{H}^*$ (or $\mathbf{W}^{*T}$). Instead we will pick a representative sample of M columns of $\mathbf{H}^*$ (or $\mathbf{W}^{*T}$), and if the updates to these M columns pass the hypothesis tests, we will deem the sample size, N_t^H (or N_t^W), as providing necessary accuracy to compute updates to all columns of $\mathbf{H}^*$ (or $\mathbf{W}^{*T}$). We keep the value of M fixed throughout the optimization procedure, and we have chosen $M = 10$ in all our experiments. In theory, one should apply a correction for multiple hypothesis testing. For instance, the Bonferroni correction simply divides the significance level by the number of tests performed. However, these considerations are required only if a principled stopping criterion is desired. In practice, the significance level is often a tuning parameter to optimize the computational efficiency of the algorithm. Also, note that the update calculated by BPP-SOLVE is $\mathbf{y}^* = [\mathbf{y}_F; \mathbf{0}]$ where $\mathbf{y}_F$ has the form $(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b}$. We perform our hypothesis testing only on $\mathbf{y}_F$, the components of $\mathbf{y}^*$ corresponding to the index set F. Thus, we introduce a modified version of the hypothesis test in Procedure 13 (HTEST-NMF).

Procedure 13 HTEST-NMF

Require: $\mathbf{A} \in \mathbb{R}^{N_t \times P}$, $\mathbf{B} \in \mathbb{R}^{N_t \times Q}$, $\mathbf{Y}_{t-1}^* \in \mathbb{R}^{P \times Q}$, ρ^*, M
Ensure: *pass*
1: *pass* $\leftarrow$ **true**
2: Choose M indices $q_1, ..., q_M$ in $\{1...Q\}$
3: **for** $i = 1$ **to** M **do**
4: $\{\mathbf{y}_t^*, F\} \leftarrow$ BPP-SOLVE$(\mathbf{A}, \mathbf{B}[:,q_i])$
5: *pass* $\leftarrow$ HYPOTHESIS-TEST$(\mathbf{A}[:,F], \mathbf{B}[:,q_i]), \mathbf{Y}_{t-1}^*[F,q_i], \rho^*)$
6: **if not** *pass* **then**
7: **break**
8: **end if**
9: **end for**
10: **return** *pass*

In addition to reducing the size of NNLS problems and making each call to BPP-SOLVE faster, our sub-sampling method also reduces the number of NNLS problems to be solved, i.e. it also decreases the number of calls to BPP-SOLVE. To see this, consider the **for** loop in steps 4 to 6 of Algorithm 12, which makes S calls to BPP-

SOLVE to solve for each column of $\mathbf{H}_t^*$ in iteration t. However, in the next step of iteration t, using our sub-sampling method, we just need N_t^W columns of $\mathbf{H}_t^*$ to estimate the rows of $\mathbf{W}_t^*$. Thus, we need to solve for only N_t^W columns of $\mathbf{H}_t^*$ in the t^{th} iteration. Similarly, in iteration t, we just need to solve for N_{t+1}^H columns of $\mathbf{W}_t^*$ instead of solving for all R columns as in steps 7 to 9 . Since $N_t^W \ll S$ and $N_{t+1}^H \ll R$ in the initial iterations, the reduced number of calls to BPP-SOLVE yield very significant savings.

As mentioned in previous chapters, the sub-sampling method also provides us with an automatic and principled stopping criterion. In the case of NMF, we stop optimizing when the updates to at least one of the factors fail while using all the available data. The complete algorithm is shown in Algorithm 14.

6.4 Experiments

We compared the performance of the original Block Principal Pivoting (BPP) algorithm to BPP using our Sub-Sampling scheme (BPP-SS) on three real world datasets: the AT&T dataset of faces[2], the MNIST dataset of handwritten digits [3] and a dataset of HOG [5] features extracted from the INRIA pedestrian detection dataset [4]. Both algorithms were implemented in MATLAB. The experiments on the AT&T faces dataset were executed on a 2.2 GHz Core2 Duo, 3GB RAM machine running Windows 7 whereas the MNIST and HOG dataset experiments were run on a 2.93GHz 4 Intel(R) Xeon(R) X5570, 96GB RAM cluster node running Linux 2.6.

For all datasets, we started out with a sample size around 500 and chose the threshold, ρ^*, for failing a hypothesis test to be 0.4. We always performed $M = 10$ tests at every iteration of the algorithm. For BPP-SS, we stopped the algorithm when the hypothesis tests failed with the full dataset. We calculated the relative error or "residual" for BPP-SS as $\|\mathbf{X}-\hat{\mathbf{W}}^*\hat{\mathbf{H}}^*\|_F/\|\mathbf{X}\|_F$ and used this as the stopping criterion for the BPP algorithm. We have verified that our stopping criterion resulted in very similar (in fact slightly smaller) residuals as the ones reported in [10]. We ran 10 trials on each dataset using the same initial value of $\mathbf{W}_0$ for both algorithms in any particular trial. We measured the the residual, the average running time of both algorithms and its standard deviation. For each trial, we measured the speed-up factor as the ratio between the running time of BPP with respect to that of BPP-SS. Note that because the algorithms may converge to different local minima for different trials, the variation of the running times of BPP or BPP-SS between trials is high and the only good comparison is the mean *paired* speed-up and its standard deviation. Note that since we are measuring the speed-up *ratio*, the mean and standard deviation refer to the geometric mean and geometric standard deviation respectively.

[2]http://www.cl.cam.ac.uk/research/dtg/attarchive/ facedatabase.html

[3]http://yann.lecun.com/exdb/mnist/

[4]http://pascal.inrialpes.fr/data/human/

Algorithm 14 Block Principal Pivoting Algorithm for NMF with Sub-Sampling

Require: Dataset $\mathcal{D}_{R,S} = \{\mathbf{X} \in \mathbb{R}^{R \times S}\}$, $K < \min\{R, S\}$, ρ^*
Ensure: Parameter estimates $\hat{\mathbf{W}}^* \in \mathbb{R}^{R \times K}, \hat{\mathbf{H}}^* \in \mathbb{R}^{K \times S}$
$M \leftarrow 10$
$t \leftarrow 1$
Initialize $N_t^H \ll R$, $N_t^W \ll S$
Initialize $\mathbf{W}_0^* \in \mathbb{R}^{R \times K} \geq \mathbf{0}$, $\mathbf{H}_0^* \in \mathbb{R}^{K \times S} \geq \mathbf{0}$
loop
  {Solve for $\mathbf{H}^*$:}

  $\mathbf{H}_t^* \leftarrow \mathbf{H}_{t-1}^*$
  $htPassed \leftarrow$ HTEST-NMF($\mathbf{W}_{t-1}^*[1{:}N_t^H, :]$, $\mathbf{X}[1{:}N_t^H, 1{:}N_t^W]$, $\mathbf{H}_{t-1}^*[:, 1{:}N_t^W]$, ρ^*, M)
  if $htPassed$ **then**
    $N_{t+1}^H \leftarrow N_t^H$
    for $i = 1$ **to** N_t^W **do**
      $\hat{\mathbf{H}}^*[:, i] \leftarrow \mathbf{H}_t^*[:, i] \leftarrow$ BPP-SOLVE($\mathbf{W}_{t-1}^*[1{:}N_t^H, :], \mathbf{X}[1{:}N_t^H, i]$)
    end for
  else
    if $N_t^H = R$ **then**
      break
    else
      $N_t^H \leftarrow \min\{2N_t^H, R\}$
    end if
  end if

  {Solve for $\mathbf{W}^*$}:

  $\mathbf{W}_t^* \leftarrow \mathbf{W}_{t-1}^*$
  $htPassed \leftarrow$ HTEST-NMF($\mathbf{H}_t^{*T}[1{:}N_t^W, :]$, $\mathbf{X}^T[1{:}N_t^W, 1{:}N_{t+1}^H]$, $\mathbf{W}_t^{*T}[:, 1{:}N_{t+1}^H]$, ρ^*, M)
  if $htPassed$ **then**
    $N_{t+1}^W \leftarrow N_t^W$
    for $j = 1$ **to** N_{t+1}^H **do**
      $\hat{\mathbf{W}}^{*T}[:, j] \leftarrow \mathbf{W}_t^{*T}[:, j] \leftarrow$ BPP-SOLVE($\mathbf{H}_t^{*T}[1{:}N_t^W, :], \mathbf{X}^T[1{:}N_t^W, j]$)
    end for
  else
    if $N_t^W = S$ **then**
      break
    else
      $N_t^W \leftarrow \min\{2N_t^W, S\}$
    end if
  end if

  $t \leftarrow t + 1$
end loop
return $\hat{\mathbf{W}}^*, \hat{\mathbf{H}}^*$

For tuning our parameters and initial testing, our first experiments were on the AT&T database of faces which consists of 400 face images of 40 different people with 10 images per person. Each face image has 92×112 pixels and we obtained a 10304×400 matrix. The average performance of both algorithms are shown in Figure 6.1a for different values of K. We show the residuals over time for K =16 and 81 in Figures 6.2a and 6.2b respectively. Note that the residual can go up for BPP-SS when we increase the sample size, since we also introduce more NNLS sub-problems to be solved. We show how the sample size N_t^H, i.e. the number of rows of $\mathbf{W}_t^*$ used to solve for $\mathbf{H}_t^*$, increases in Figures 6.2c and 6.2d. In Figures 6.2e and 6.2f, the time taken per iteration is shown. Note the spike in processing time, when the sample size is increased.

Our next set of experiments was on the MNIST database of handwritten digits. We used 60000 images of handwritten digits, each 28 $\times$ 28 pixels, giving a 60000 $\times$ 784 matrix. Results are shown in Figure 6.1b. Our final experiments were on a dataset consisting of HOG features of over 5 million windows from the INRIA pedestrian detection dataset. Each scanning window was divided into 14 $\times$ 6 cells of 8 $\times$ 8 pixels and each cell was represented by a gradient orientation histogram using 13 bins resulting in a 1092 element vector per window. We used only a 1 million subset of windows giving a $1,000,000 \times 1092$ matrix, because of memory constraints. This is an extremely large dataset which allowed the BPP-SS algorithm to achieve very significant speed-ups over the BPP algorithm (Figure 6.1c).

Our experiments on these three real world datasets clearly demonstrate that the proposed sub-sampling method significantly improves the NMF Block Principal Pivoting algorithm, especially on large datasets.

k	Residual	Running Time (sec)		Speed Up	
		BPP	BPP-SS	Mean	Std. Dev.
16	0.1888	60.53	42.85	1.90	1.21
25	0.1725	106.46	90.57	2.18	1.30
36	0.1591	179.99	143.05	1.95	1.14
49	0.1475	560.64	467.63	2.25	1.26
64	0.1372	407.29	303.99	2.26	1.16
81	0.1284	707.81	459.14	2.28	1.20

(a) Comparison on the AT&T data set

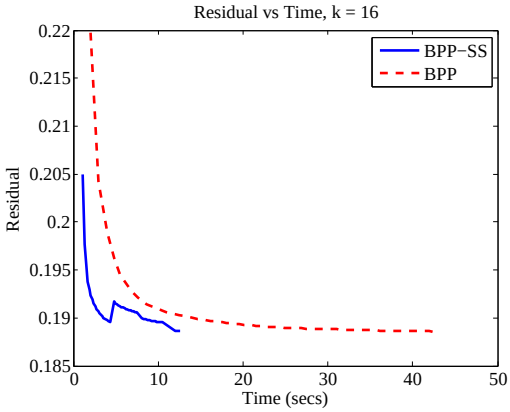
k	Residual	Running Time (sec)		Speed Up	
		BPP	BPP-SS	Mean	Std. Dev.
10	0.5936	79.2	54.41	1.21	1.43
15	0.5534	271.18	122.39	1.96	1.37
20	0.5228	303.92	150.23	1.96	1.37
25	0.4929	432.79	205.99	2.48	1.50
30	0.4673	578.28	241.86	2.79	1.32
35	0.4441	636.84	262.18	2.44	1.31
40	0.4225	742.26	285.81	2.82	1.30
45	0.4045	1021.1	430.76	2.79	1.54
50	0.3875	1409.9	620.75	2.49	1.41

(b) Comparison on the MNIST data set

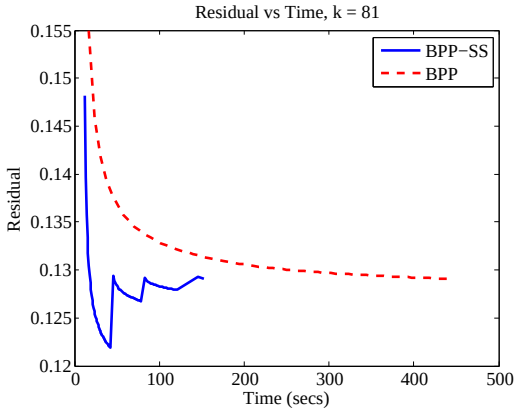
k	Residual	Running Time (sec)		Speed Up	
		BPP	BPP-SS	Mean	Std. Dev.
5	0.3818	2609.9	943.21	2.06	1.79
10	0.3632	7858.4	2084.6	4.20	1.40
15	0.3489	14763	4049.6	3.09	1.46
20	0.3394	13508	5240.6	3.67	1.73
25	0.4894	14976	3929.6	5.39	1.54
30	0.3232	18127	7347.8	5.10	1.38

(c) Comparison on the HOG features data set

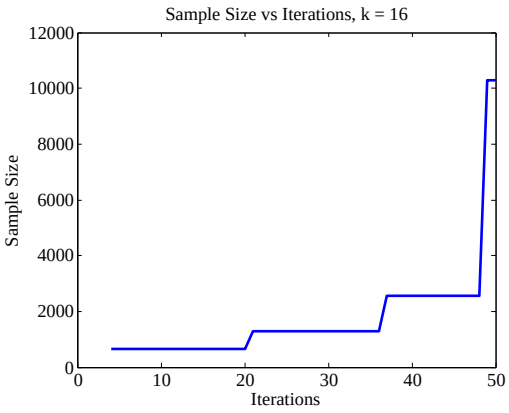
Figure 6.1: Performance Comparison of BPP versus its sub-sampled version(BPP-SS) on 3 real world data sets.



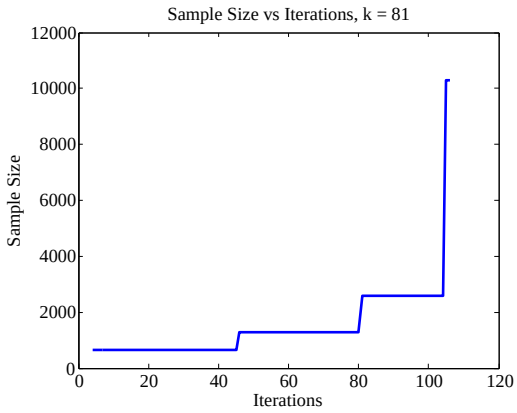
(a) Residual vs Time, k = 16



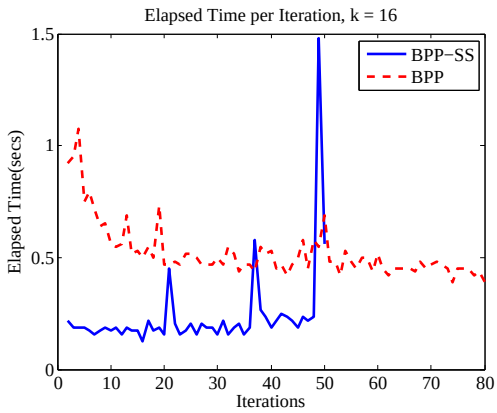
(b) Residual vs Time, k = 16



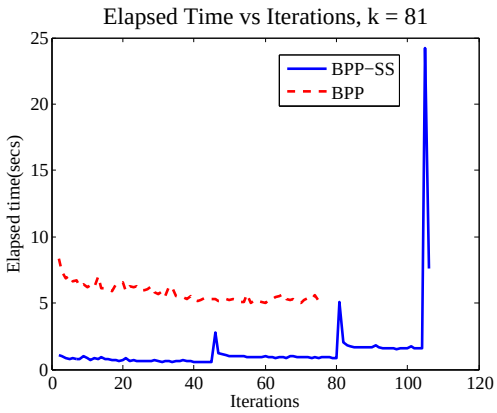
(c) Sample Size vs Iterations, k = 16



(d) Sample Size vs Iterations, k = 81



(e) Elapsed Time per Iteration, k = 16



(f) Elapsed Time per Iteration, k = 81

Figure 6.2: Performance of BPP versus its sub-sampled version(BPP-SS) on the AT&T data set.

Chapter 7

Conclusion

We argued that statistical estimation should not be considered as a mere mathematical optimization procedure once the loss function has been chosen. Instead, one should treat learning as a statistical procedure, acknowledging the fact that the loss function is a stochastic quantity which fluctuates under resampling the dataset from its generating process.

We proposed a general stochastic optimization method that can be used to speed-up a variety of iterative estimation problems. In each iteration, our method computes by using only just enough data to reliably take a single step. Thus, in early iterations, when we are far from the true solution, only a few datapoints are needed to tell us the general direction to move in parameter space. However, as we proceed towards convergence, more precision is required in the updates. The optimal batch size in each iteration is determined using frequentist hypothesis tests which check whether the proposed update direction is correct with high probability. If the hypothesis tests fail, it is an indication that more precision is needed in the updates and the batch size is increased. Thus, we proposed an "as needed" approach to computation.

Our method, besides speeding up learning by sub-sampling the data, also has the advantage of an automatic and principled stopping criterion. When all hypothesis tests fail and we cannot increase the sample size, either due to unavailability of more data or due to computational limitations, we terminate learning. This is a statistically principled stopping criterion because, at this point, another dataset drawn from the true underlying distribution of the data might propose a significantly different update direction. Another important advantage of our method is that we have only a single interpretable parameter: the significance level of the hypothesis tests. Also, our method is orthogonal to many other speed-up methods developed in the optimization literature and can often be effectively combined with them.

The method crucially depends on the central limit theorem and we have indeed observed that the method fails when such central limit tendencies are absent. This

appears to be the case when the data-matrix is very sparse. Another example where we have noticed our algorithm to fail is Non-negative Tensor Factorization. In this case, the reshaping of the tensor at every iteration induces complicated dependencies and non-Gaussian behavior (presumably due to the multiplication of random variables). However, despite the above exceptions, many interesting learning algorithms exist to which our ideas can be applied.

Experiments on real world datsets showed that our method works well in practice and can highly speed-up many learning algorithms, especially on large datasets. For example, we were able to speed-up the state of the art Non-negative Matrix Factorization algorithm by a factor of 4-5.

Bibliography

[1] D. Andrews and C. Mallows. Scale mixtures of normal distributions. *Journal of the Royal Statistical Society. Series B (Methodological)*, 36(1):99–102, 1974.

[2] D. P. Bertsekas. *Nonlinear programming*. Athena Scientific, Belmont, Mass, 1999.

[3] C. Bishop. *Pattern recognition and machine learning*, volume 4. Springer New York, 2006.

[4] L. Bottou and O. Bousquet. Learning using large datasets. *Mining Massive DataSets for Security, NATO ASI Workshop Series*, 2008.

[5] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 886–893, 2005.

[6] A. Dempster, N. Laird, D. Rubin, et al. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

[7] A. Frank and A. Asuncion. UCI Machine Learning Repository, 2010.

[8] D. Kim, S. Sra, and I. Dhillon. Fast newton-type methods for the least squares nonnegative matrix approximation problem. In *Proceedings of the 2007 SIAM International Conference on Data Mining*, 2007.

[9] H. Kim and H. Park. Nonnegative matrix factorization based on alternating nonnegativity constrained least squares and active set method. *SIAM Journal on Matrix Analysis and Applications*, 30(2):713–730, 2008.

[10] J. Kim and H. Park. Toward faster nonnegative matrix factorization: A new algorithm and comparisons. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining (ICDM)*, pages 353–362, 2008.

[11] A. Korattikara, L. Boyles, M. Welling, J. Kim, and H. Park. Statistical Optimization of Non-Negative Matrix Factorization. In *AISTATS*, 2011.

[12] H. Kushner and G. Yin. *Stochastic approximation and recursive algorithms and applications*. Springer Verlag, 2003.

[13] C. L. Lawson and R. J. Hanson. *Solving Least Squares Problems*. Society for Industrial and Applied Mathematics, 1995.

[14] C.-J. Lin. Projected gradient methods for nonnegative matrix factorization. *Neural Computation*, 19(10):2756–2779, 2007.

[15] P. McCullagh and J. Nelder. *Generalized linear models*. Chapman & Hall, CRC, 1999.

[16] R. Phillips. Least absolute deviations estimation via the EM algorithm. *Statistics and Computing*, 12(3):281–285, 2002.

[17] H. Robbins and S. Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951.

[18] J. Spall. *Introduction to stochastic search and optimization: estimation, simulation, and control*. John Wiley and Sons, 2003.

[19] M. Verbeek. *A Guide to Modern Econometrics*. John Wiley and Sons, 2000.

[20] B. Yu. Embracing statistical challenges in the information technology age. *Technometrics, American Statistical Association and the American Society for Quality*, 49:237–248, 2007.

Appendices

A Notation (and abuse thereof)

We will represent vectors using bold lowercase symbols (e.g. $\mathbf{v}$) and matrices using bold uppercase symbols (e.g. $\mathbf{M}$).We will denote the i^{th} element of a vector $\mathbf{v} \in \mathbb{R}^{N\times 1}$ by $\mathbf{v}_i$. If a vector has other subscripts, say, for instance, if $\mathbf{v}_t$ represents a vector calculated at time t, we will use $\mathbf{v}_t[i]$ to represent the i^{th} element of $\mathbf{v}_t$. Occasionally, we use a 'MATLAB like' notation for indexing vectors. If P is a list of indices, then $\mathbf{v}[P]$ represents a $|P| \times 1$ vector composed of elements of $\mathbf{v}$ with indices in P. $\mathbf{v}[a:b]$, where a and b are two integers in $[1, N]$, represents a vector formed from a subset of elements of $\mathbf{v}$ with indices ranging from a to b. If a and b are not specified, they are assumed to be 1 and N respectively, i.e. $\mathbf{v}[:]$ represents the whole vector $\mathbf{v}$. Similar conventions are also used for matrix indexing.

We will use the notation $\text{diag}(\mathbf{v})$ to represent the diagonal matrix $\mathbf{D} \in \mathbb{R}^{N\times N}$ such that $\mathbf{D}_{ii} = \mathbf{v}_i$ for $i = 1...N$. $\mathbf{1}$ or $\mathbf{0}$ represents a vector or matrix whose elements are all 1 or 0, respectively. The dimensions of $\mathbf{1}$ or $\mathbf{0}$ will be specified, if not clear from the context. If $h(.)$ is a function that takes a scalar argument and $\mathbf{v} \in \mathbb{R}^{N\times 1}$, then we abuse notation slightly and write $h(\mathbf{v}) \in \mathbb{R}^{N\times 1}$ to represent the vector obtained by applying $h(.)$ to each component of $\mathbf{v}$ separately. Similarly, ${}^{1}/_{\mathbf{v}} \in \mathbb{R}^{N\times 1}$ represents the vector obtained by taking the reciprocals of each component of $\mathbf{v}$. For two vectors $\mathbf{v}, \mathbf{w} \in \mathbb{R}^{N\times 1}$, the product $\mathbf{vw}$ denotes the vector of element-wise products.