Published in final edited form as:
Comput Stat Data Anal. 2011 January 1; 55(1): 26–33. doi:10.1016/j.csda.2010.04.019.

A Fast Procedure for Calculating Importance Weights in Bootstrap Sampling

Hua Zhou[a] and **Kenneth Lange**[b]

[a]Department of Human Genetics University of California Los Angeles, CA 90095-1766 huazhou@ucla.edu

[b]Departments of Biomathematics, Human Genetics, and Statistics University of California Los Angeles, CA 90095-1766 klange@ucla.edu

Abstract

Importance sampling is an efficient strategy for reducing the variance of certain bootstrap estimates. It has found wide applications in bootstrap quantile estimation, proportional hazards regression, bootstrap confidence interval estimation, and other problems. Although estimation of the optimal sampling weights is a special case of convex programming, generic optimization methods are frustratingly slow on problems with large numbers of observations. For instance, interior point and adaptive barrier methods must cope with forming, storing, and inverting the Hessian of the objective function. In this paper, we present an efficient procedure for calculating the optimal importance weights and compare its performance to standard optimization methods on a representative data set. The procedure combines several potent ideas for large scale optimization.

Keywords

importance resampling; bootstrap; majorization; quasi-Newton acceleration

1. Introduction

Because it involves Monte Carlo estimation, the nonparametric bootstrap is an obvious candidate for importance sampling. To our knowledge, Johns [10] and Davison [1] first recognized the possibilities in the context of quantile estimation. The general idea is to sample cases with nonuniform weights. If the weights are carefully tuned to a given statistic, then importance sampling can dramatically reduce the variance of the bootstrap sample average estimating the mean of the statistic [6]. Bootstrap importance sampling has expanded beyond quantile estimation to include proportional hazards regression, bootstrap confidence interval estimation, and many other applications [2,4,10,7].

Although estimation of the optimal sampling weights is a constrained optimization problem that yields to standard methods of convex programming, there is still room for improvement, particularly in problems with large numbers of observations. Interior point and adaptive barrier methods incur heavy costs in forming, storing, and inverting the Hessian of the objective function. In the current paper, we present an efficient procedure for calculating the optimal importance weights and compare its performance to standard optimization methods on a representative data set. The procedure combines several potent ideas for large scale optimization. Briefly these include: (a) approximating the objective function by a quadratic, (b) majorizing the quadratic by a simple quadratic surrogate with parameters separated, (c) reparameterizing the surrogate so its minimization reduces to finding the closest point to a simplex, (d) mapping the simplex solution back to the original parameters, and (e) accelerating the entire scheme by a quasi-Newton improvement for finding the fixed point of a smooth algorithm map. The procedure sounds complicated, but each step is fast and straightforward to implement. On a test example with 1,664 observations, our accelerated algorithm surpasses the performance of current standard methods for optimization.

Section 2 introduces the convex optimization problem defining the importance weights and derives our optimization procedure. Section 3 reviews and generalizes the clever simplex projection algorithm of Michelot. Section 4 summarizes our quasi-Newton acceleration; this scheme is specifically tailored to high-dimensional problems. Section 5 compares our new procedure, both unaccelerated and accelerated, to the standard methods of convex optimization on our sample problem. Finally, our discussion points readers to other applications of the design principles met here for high-dimensional optimization.

2. Optimization in Importance Resampling

In standard bootstrap resampling with n observations, each observation x_i is resampled uniformly with probability n^{-1}. As just argued, it is often helpful to implement importance sampling by assigning different resampling probabilities p_i to the different observations [1,3, 5]. For instance, with univariate observations $(x_1, \ldots, x_n)$, we may want to emphasize one of the tails of the empirical distribution. If we elect to resample nonuniformly according to the multinomial distribution with proportions $p = (p_1, \ldots, p_n)^t$, then the change of measure equality

$$\begin{aligned}\mathbf{E}\left[T\left(\mathbf{x}^*\right)\right] &= \mathbf{E}_p\left[T\left(\mathbf{x}^*\right)\frac{\binom{n}{m_1^* \cdots m_n^*}\left(\frac{1}{n}\right)^n}{\binom{n}{m_1^* \cdots m_n^*}\prod_{i=1}^n p_i^{m_i^*}}\right]\\ &= \mathbf{E}_p\left[T\left(\mathbf{x}^*\right)\prod_{i=1}^n \left(np_i\right)^{-m_i^*}\right]\end{aligned}$$

connects the uniform expectation and the importance expectation on the bootstrap resampling space. Here m_i^* represents the number of times sample point x_i appears in x*. Thus we can approximate the mean **E**[T (x*)] by taking a bootstrap average

$$\frac{1}{B}\sum_{b=1}^{B} T\left(\mathbf{x}_b^*\right)\prod_{i=n}^{n}\left(np_i\right)^{-m_{b_i}^*}$$

with multinomial sampling relative to p. This Monte Carlo approximation has variance

$$\frac{1}{B}\left\{\mathbf{E}_p\left[T(\mathbf{x}^*)^2\prod_{i=1}^{n}\left(np_i\right)^{-2m_{b_i}^*}\right] - \mathbf{E}\left[T\left(\mathbf{x}^*\right)\right]^2\right\},$$

which achieves its minimum with respect to *p* when the theoretical second moment $\mathbf{E}_p\left[T(\mathbf{x}^*)^2\prod_{i=1}^n (np_i)^{-2m_{b_i}^*}\right]$ is minimized.

Hall [4] suggests approximately minimizing the second moment by taking a preliminary uniform bootstrap sample of size B_1. Based on the preliminary resample, we approximate $\mathbf{E}_p\left[T(\mathbf{x}^*)^2\prod_{i=1}^n (np_i)^{-2m_{b_i}^*}\right]$ by the Monte Carlo average

$$\begin{aligned} s(p) &= \frac{1}{B_1}\sum_{b=1}^{B_1} T\left(\mathbf{x}_b^*\right)^2 \prod_{i=1}^n (np_i)^{-2m_{b_i}^*} \prod_{i=1}^n (np_i)^{m_{b_i}^*} \\ &= \frac{1}{B_1}\sum_{b=1}^{B_1} T\left(\mathbf{x}_b^*\right)^2 \prod_{i=1}^n (np_i)^{-m_{b_i}^*}. \end{aligned}$$

The function *s*(*p*) serves as a surrogate for $\mathbf{E}_p\left[T(\mathbf{x}^*)^2\prod_{i=1}^n (np_i)^{-2m_{b_i}^*}\right]$. It is possible to minimize *s*(*p*) on the open unit simplex by standard methods. Unfortunately, Newton's method is hampered when the *n* is large by the necessity of evaluating, storing, and inverting the Hessian matrix at each iteration. This dilemma prompted our quest for a more efficient algorithm for minimizing *s*(*p*).

Consider the optimization problem

$$\min_p s(p) \qquad \text{subject to} \sum_{i=1}^n p_i = 1, \qquad p_i \geq \epsilon, \qquad 1 \leq i \leq n.$$

Here the lower bound ε > 0 is imposed so that sampling does not entirely neglect some observations. In practice we take $\varepsilon = n^{-2}$ or n^{-3}. The gradient and second differential (Hessian) of *s*(*p*) are

$$\nabla s(p) = -\frac{1}{B_1}\sum_{b=1}^{B_1} T\left(x_b^*\right)^2 \prod_{j=1}^n \left(np_j\right)^{-m_{b_j}^*} \begin{pmatrix} \frac{m_{b_1}^*}{p_1} \\ \vdots \\ \frac{m_{b_n}^*}{p_n} \end{pmatrix}$$

and

$$d^2 s(p) = \frac{1}{B_1}\sum_{b=1}^{B_1} T\left(x_b^*\right)^2 \prod_{j=1}^n \left(np_j\right)^{-m_{b_j}^*} \times \left[\begin{pmatrix} \frac{m_{b_1}^*}{p_1} \\ \vdots \\ \frac{m_{b_n}^*}{p_n} \end{pmatrix} \begin{pmatrix} \frac{m_{b_1}^*}{p_1} & \cdots & \frac{m_{b_n}^*}{p_n} \end{pmatrix} + \begin{pmatrix} \frac{m_{b_1}^*}{p_1^2} & & \\ & \ddots & \\ & & \frac{m_{b_n}^*}{p_n^2} \end{pmatrix}\right].$$

Because $d^2s(p)$ is positive definite, *s*(*p*) is strictly convex. Evaluation of the gradient and Hessian requires $O(nB_1)$ and $O(n^2B_1)$ operations, respectively.

Our first step in minimizing the objective function *s*(*p*) is to approximate it by a quadratic around the current iterate p^k. According to Taylor's theorem, we have

$$\begin{aligned} s(p) &\approx s\left(p^k\right)+\nabla s\left(p^k\right)^t\left(p-p^k\right)+\tfrac{1}{2}\left(p-p^k\right)^t d^2 s\left(p^k\right)\left(p-p^k\right) \\ &=s\left(p^k\right)-\tfrac{1}{B_1}\sum_b c_b v_b^t\left(p-p^k\right)^k+\tfrac{1}{2B_1}\left(p-p^k\right)^t\sum_b c_b\left(v_b v_b^t+D_b\right)\left(p-p^k\right) \\ &=r\left(p \mid p^k\right), \end{aligned}$$

where $c_b=T\left(x_b^*\right)^2\prod_{j=1}^n\left(np_j\right)^{-m_{b_j}^*}$ and

$$v_b=\left(m_{b_1}^* p_1^{-1},\ldots,m_{b_n}^* p_n^{-1}\right)^t, \qquad D_b=\mathrm{diag}\left(m_{b_1}^* p_1^{-2},\ldots,m_{b_n}^* p_n^{-2}\right).$$

Our second step is to majorize the quadratic $r(p \mid p^k)$ by a quadratic with parameters separated. If we set $u = \sum_b c_b v_b$ and $D=\sum_b c_b\left(\|v_b\|^2 I_n+D_b\right)$, then application of the Cauchy-Schwartz inequality $\|v_b\|^2\|w\|^2 \geq \left(v_b^t w\right)^2$ yields the inequality

$$r\left(p \mid p^k\right) \leq s\left(p^k\right)-\frac{1}{B_1}\left[u^t+\left(p^k\right)^t D\right]p+\frac{1}{2B_1}p^t Dp+c^k=q\left(p \mid p^k\right),$$

where c^k is an irrelevant constant that does not depend on p. Because equality holds in this last inequality whenever $p = p^k$, the function $q(p \mid p^k)$ is said to majorize $r(p \mid p^k)$. The guiding principle of the MM algorithm [8,12] is that minimizing $q(p \mid p^k)$ drives $r(p \mid p^k)$, and presumably $s(p)$, downhill. Thus, we achieve a steady decrease in $s(p)$.

Our third step is to transform minimization of $q(p \mid p^k)$ into a problem of finding the closest point to a truncated simplex. This step is effected by the reparameterization $p^* = D^{1/2}p$, where $D^{1/2}$ is the matrix square root of D. Minimization of $q(p \mid p^k)$ reduces to minimizing the squared distance

$$\frac{1}{2}\|p^*-\left(D^{-1/2}u+D^{1/2}p^k\right)\|^2$$

subject to the constraints $\mathbf{1}^t D^{-1/2}p^* = 1$ and $p^* \geq \varepsilon D^{1/2}\mathbf{1}$. Before we discuss how to project $(D^{-1/2}u + D^{1/2}p^k)$ onto this truncated simplex, let us summarize our overall algorithm in pseudo-code.

Several remarks are pertinent. (a) Projection onto the truncated simplex can be solved by a slight generalization of an efficient algorithm of Michelot [13]. The details spelled out in the next section show that projection requires at most $O(n^2)$ operations and usually much fewer in practice. (b) Evaluation of u and D requires $O(nB_1)$ operations. These represent potentially huge gains over Newton's method if convergence occurs fast enough. Recall that Newton's method needs $O(nB_1)$ operations for evaluating the gradient, $O(n^2B_1)$ operations for evaluating the Hessian matrix, and $O(n^3)$ for inverting the Hessian matrix. (c) The boundary conditions and linear constraint are incorporated in the algorithm gracefully. (d) A side effect of majorization is the loss of the superlinear convergence enjoyed by Newton's method. We therefore accelerate convergence by applying a general quasi-Newton scheme for fixed point problems. As discussed in Section 4, this scheme requires little extra computation per iteration and only $O(n)$ storage. It is particularly attractive for high-dimensional problems. By contrast Newton's method requires $O(n^2)$ storage for manipulating the Hessian matrix.

3. Michelot Algorithm

Michelot [13] derived an efficient algorithm for projecting a point onto the unit simplex in $\mathbb{R}^n$. This algorithm converges in at most n iterations and often much sooner. We consider a trivial generalization that maps a point $x \in \mathbb{R}^n$ to the closest point $P_K(x)$ in the dilated and truncated simplex

$$K=\left\{y \in \mathbb{R}^n: \sum_{i=1}^{n} \alpha_i y_i = c, \quad y_i \geq \epsilon_i, \quad 1 \leq i \leq n\right\}, \tag{1}$$

where the α_i and ε_i are strictly positive and together satisfy $\sum_i \alpha_i \epsilon_i \leq c$. The unit simplex is realized by taking $c = 1$ and $\alpha_i = 1$ and $\varepsilon_i = 0$ for all i. The revised algorithm cycles through the following steps.

The Michelot algorithm stops after a finite number of iterations because every iteration reduces the dimension n by at least 1. The first two steps of the algorithm are motivated by the following propositions, whose proofs are straightforward generalizations of those of Michelot [13]. Full validation of the revised algorithm follows from his further arguments.

Proposition 3.1

Suppose C is a closed convex set wholly contained within an affine subspace V. Then the projection $P_C(x)$ *onto C and the projection* $P_V(x)$ *onto V satisfy* $P_C(x) = P_C \circ P_V(x)$.

Proof—See Michelot's paper [13].

Proposition 3.2

Suppose $x \in \mathbb{R}^n$ satisfies $\sum_i \alpha_i x_i = c$, where $\alpha_i > 0$ for all i. If $x' \in \mathbb{R}^n$ has coordinates $x_i' = \max\{x_i, \epsilon_i\}$, then $P_K(x) = P_K(x')$ *for the truncated simplex (1).*

Proof—Consider minimizing the objective function $y \mapsto \frac{1}{2}\|y-x\|^2$ subject to the linear constraint $\sum_{i=1}^n \alpha_i y_i = c$ and boundary conditions $y_i \geq \varepsilon_i$ for every i. The Lagrangian function is

$$L(y, \lambda, \mu_i) = \frac{1}{2}\|y-x\|^2 + \lambda\left(\sum_i \alpha_i y_i - c\right) - \sum_i \mu_i (y_i - \epsilon_i).$$

Because this is a convex programming problem, the Karush-Kuhn-Tucker (KKT) optimality conditions are both necessary and sufficient. These conditions can be stated as

$$y_i - x_i + \lambda\alpha_i - \mu_i = 0 \tag{2}$$

$$(y_i - \epsilon_i)\mu_i = 0 \tag{3}$$

$$\mu_i \geq 0 \tag{4}$$

for multipliers λ and μ_i. Multiplying both sides of equality (2) by α_i and summing over i determines λ as the ratio

$$\lambda = \frac{\sum_i \mu_i \alpha_i}{\sum_i \alpha_i^2} \geq 0.$$

If $x_i < \varepsilon_i$, then condition (2) implies $\mu_i = y_i - x_i + \lambda\alpha_i > 0$. But condition (3) now compels the equality $y_i = \varepsilon_i$. Therefore we can replace x_i by ε_i and μ_i by $\lambda\alpha_i$ and still maintain the KTT conditions at the point y. In other words, $P_K(x) = P_K(x')$.

4. A Quasi-Newton Acceleration Scheme

In this section, we review a general quasi-Newton acceleration [17] for fixed point problems. If $F(x)$ is an algorithm map, then the idea of the scheme is to approximate Newton's method for finding a root of the equation $0 = x - F(x)$. Let $G(x)$ now denote the difference $G(x) = x - F(x)$. Because $G(x)$ has the differential $dG(x) = I - dF(x)$, Newton's method iterates according to

$$x^{k+1} = x^n - dG\left(x^k\right)^{-1} G\left(x^k\right) = x^k - \left[I - dF\left(x^k\right)\right]^{-1} G\left(x^k\right). \tag{5}$$

If we can approximate $dF(x^k)$ by a low-rank matrix M, then we can replace $I - dF(x^k)$ by $I - M$ and explicitly form the inverse $(I - M)^{-1}$.

Quasi-Newton methods operate by secant approximations. We generate one of these by taking two iterates of the algorithm starting from the current point x^k. When we are close to the optimal point x^∞, we have the linear approximation

$$F \circ F\left(x^k\right) - F\left(x^k\right) \approx M\left[F\left(x^k\right) - x^k\right],$$

where $M = dF(x^\infty)$. If υ is the vector $F \circ F(x^k) - F(x^k)$ and u is the vector $F(x^k) - x^k$, then the secant requirement is $Mu = \upsilon$. In fact, for the best results we require several secant approximations $Mu_i = \upsilon_i$ for $i = 1, \ldots, q$. These can be generated at the current iterate x^k and the previous $q - 1$ iterates. The next proposition gives a sensible way of approximating M.

Proposition 4.1

Let $M = (m_{ij})$ be a n × n matrix and $\|M\|_F^2 = \sum_i \sum_j m_{ij}^2$ its squared Frobenius norm. Write the secant constraints $\mathrm{Mu_i} = \upsilon_i$ *in the matrix form* $MU = V$ *for* $U = (u_1, \ldots, u_q)$ *and* $V = (\upsilon_1, \ldots, \upsilon_q)$. Provided U has full column rank q, the minimum of the strictly convex function $\|M\|_F^2$ subject to the constraints is attained by the choice $\mathrm{M} = \mathrm{V}(U^tU)^{-1}U^t$.

Proof—See the reference [17].

To apply the proposition in our proposed quasi-Newton scheme, we must invert the matrix $I - V(U^tU)^{-1}U^t$. Fortunately, the explicit inverse

$$\left[I - V\left(U^tU\right)^{-1}U^t\right]^{-1} = I + V\left[U^tU - U^tV\right]^{-1}U^t$$

is a straightforward to check variant of the Sherman-Morrison formula. The $q \times q$ matrix $U^tU - U^tV$ is trivial to invert for q small even when n is large. This result suggest replacing the Newton update (5) by the quasi-Newton update

$$\begin{aligned} x^{k+1} &= x^k - \left[I - V(U^tU)^{-1}U^t\right]^{-1}\left[x^k - F\left(x^k\right)\right] \\ &= x^k - \left[I + V(U^tU - U^tV)^{-1}U^t\right]\left[x^k - F\left(x^k\right)\right] \\ &= F\left(x^k\right) - V(U^tU - U^tV)^{-1}U^t\left[x^k - F\left(x^k\right)\right]. \end{aligned}$$

The quasi-Newton method is clearly feasible for high-dimensional problems. It takes two ordinary iterates to generate a secant condition and a corresponding quasi-Newton update. If a quasi-Newton update fails to send the objective function in the right direction, then one can always revert to the second iterate $F \circ F(x^k)$. For a given q, the obvious way to proceed is to do q initial ordinary updates and form $q - 1$ secant pairs. At that point quasi-Newton updating can commence. After each accelerated update, one should replace the earliest retained secant pair by the new secant pair. The whole scheme is summarized in Algorithm 3. Note that the effort per iteration is relatively light: two ordinary iterates and some matrix times vector multiplications. Most of the entries of U^tU and U^tV can be computed once and used over multiple iterations. The scheme is also consistent with linear constraints. Thus, if the parameter space satisfies a linear constraint $w^tx = a$ for all feasible x, then the quasi-Newton iterates also satisfy $w^tx^k = a$ for all k. This claim follows from the equalities $w^tF(x) = a$ and $w^tV = 0$ in the above notation.

Earlier quasi-Newton accelerations [9,11] focus on approximating the Hessian of the objective function rather than the differential of the algorithm map. In our recent paper [17], we demonstrate that the current quasi-Newton acceleration significantly boosts the convergence rate of a variety of optimization algorithms. We apply it in the next section to importance sampling.

5. Example

Our numerical example, borrowed from chapter 14 of the book [14], contains a random sample of n = 1,664 repair times for Verizon's telephone customers. It is evident from the histogram displayed in Figure 1 that the distribution of repair times has a long right tail and is far from normal. The median is 3.59 hours, but the mean is 8.41 hours, and the maximum is 191.6 hours. For purposes of illustration, we focus on the probability that the repair time of a Verizon customer exceeds 100 hours. The statistic of interest $T(\mathbf{X}) = \frac{1}{n}\sum_i 1_{\{x_i > 100\}}$ is strongly influenced by extreme repair times. To estimate optimal importance weights, we took a preliminary bootstrap sample of size B_1 = 1,000 and executed our estimation procedure in MATLAB. We also performed three forms of interior point optimization in MATLAB's Optimization Toolbox as part of the fmincon function. The three standard methods use the exact Hessian of the objective function, a BFGS quasi-Newton approximation to it, and a limited-memory version (LBFGS) of the BFGS approximation. The LBFGS algorithm depends as well on the number q of secant conditions selected. The active set and trust region methods also implemented in MATLAB are ignored here. The first is noticeably slower than the interior point methods, and the second cannot handle equality constraints and boundary conditions.

The exact Hessian method takes only 18 iterations but 1,248 seconds to converge. In practice, 99% of the execution time is spent on evaluating the Hessian matrix, which requires $O(n^2B_1)$ operations per iteration, and 1% of the time is spent on factoring the Hessian matrix, which requires $O(n^3)$ operations per iteration. This example illustrates the extreme speed of MATLAB's matrix operations. The interior point method with BFGS updates takes many more iterations but much less time per iteration because it dispenses with evaluating and factoring the Hessian matrix. Part of the slow convergence of the BFGS method may be attributed to the boundary conditions. In contrast, our algorithm takes $O(nB_1)$ operations per iteration and converges quickly under acceleration. As shown in Table 1, the accelerated algorithm with $q \geq 1$ secant conditions is a clear winner, giving massive improvements in execution time over the naïve MM algorithm and the Hessian and BFGS variants of Newton's method. Although our accelerated algorithm also beats the LBFGS algorithm for all choices of q, Table 2 shows that the later algorithm is highly competitive on large-scale problems. All comparisons listed in the two tables involve stringent stopping criteria, which are adjusted to give the same number of significant digits for the converged values of the objective function. Running times are recorded in seconds.

6. Discussion

In summary, our procedure uses: (a) quadratic approximation of the objective function, (b) majorization by a second quadratic with parameters separated, (c) the Michelot algorithm to project points onto a truncated simplex, and (d) acceleration by quasi-Newton approximation of the algorithm map. Each of these ideas has other applications.

Quadratic approximation lies at the heart of Newton's method and its many spinoffs. Our outer-product majorization balances separation of parameters against poor approximation of the Hessian. Separation of parameters is often the key to solving high-dimensional problems. The loss of quadratic convergence is largely remedied by acceleration. This combination of tactics also has potential in fitting generalized linear models (GLM). In this setting, Fisher's scoring method uses the expected information matrix

$$J(\beta)=\sum_{i=1}^{n}\frac{1}{\sigma_i^2(\beta)}q'\left(x_i^t\beta\right)^2 x_i x_i^t$$

rather than the observed information matrix. Here β is the parameter vector, $q(\cdot)$ is the inverse link function, x_i is the predictor vector for case i, y_i the response for case i, and $\sigma_i^2(\beta)=\mathbf{Var}(y_i)$. Note that $J(\beta)$ is again a sum of outer products. This fact suggests a combination of majorization and acceleration on high-dimensional GLM problems. In many such problems, it is prudent to also imposes a ridge or lasso penalty. The ridge penalty preserves separation of parameters by a quadratic surrogate. The lasso penalty also preserves separation of parameters, but not by a quadratic surrogate. Lasso penalized maximum likelihood estimation is amenable to cyclic coordinate descent because the lasso is linear on either side of 0. Our recent work on penalized ordinary and logistic regression [15,16] illustrates some of the possibilities. Finally, our paper [17] amply illustrates the virtues of acceleration by quasi-Newton approximation of an algorithm map.

References

[1]. Davison AC. Discussion of paper by D.V. Hinkley. J. Roy. Statist. Soc. Ser. B 1988;50:356–357.

[2]. Do KA, Wang X, Broom BM. Importance bootstrap resampling for proportional hazards regression. Comm. Statist. Theory Methods 2001;30(10):2173–2188.

[3]. Do KA, Hall P. On importance resampling for the bootstrap. Biometrika 1991;78(1):161–167.

[4]. Hall, P. The bootstrap and Edgeworth expansion. Springer-Verlag; New York: 1992.

[5]. Hesterberg T. Control variates and importance sampling for efficient bootstrap simulations. Statist. Comput 1996;6:147–157.

[6]. Hinkley DV, Shi S. Importance sampling and the nested bootstrap. Biometrika 1989;76(3):435–446.

[7]. Hu J, Su Z. Bootstrap quantile estimation via importance resampling. Computational Statistics & Data Analysis 2008;52(12):5136–5142.

[8]. Hunter DR, Lange KL. A tutorial on MM algorithms. Amer Statistician 2004;58:30–37.

[9]. Jamshidian M, Jennrich RI. Acceleration of the EM algorithm by using quasi-Newton methods. J. Roy. Statist. Soc. Ser. B 1997;59(3):569–587.

[10]. Johns MV. Importance sampling for bootstrap confidence intervals. J. Amer. Statist. Assoc 1988;83 (403):709–714.

[11]. Lange KL. A quasi-Newton acceleration of the EM algorithm. Statist. Sinica 1995;5(1):1–18.

[12]. Lange, KL. Optimization. Springer-Verlag; New York: 2004.

[13]. Michelot C. A finite algorithm for finding the projection of a point onto the canonical simplex of R^n. J. Optim. Theory Appl 1986;50(1):195–200.

[14]. Moore, DS.; McCabe, GP. Introduction to the Practice of Statistics. W.H. Freeman; 2005.

[15]. Wu TT, Lange KL. Coordinate descent algorithms for lasso penalized regression. Annals Appl. Stat 2008;2:224–244.

[16]. Wu TT, Chen YF, Hastie T, Sobel EM, Lange KL. Genomewide association analysis by lasso penalized logistic regression. Bioinformatics 2009;25:714–721. [PubMed: 19176549]

[17]. Zhou, H.; Alexander, D.; Lange, KL. Statistics and Computing. 2009. A quasi-Newton acceleration for high-dimensional optimization algorithms. DOI:10.1007/s11222-009-9166-3

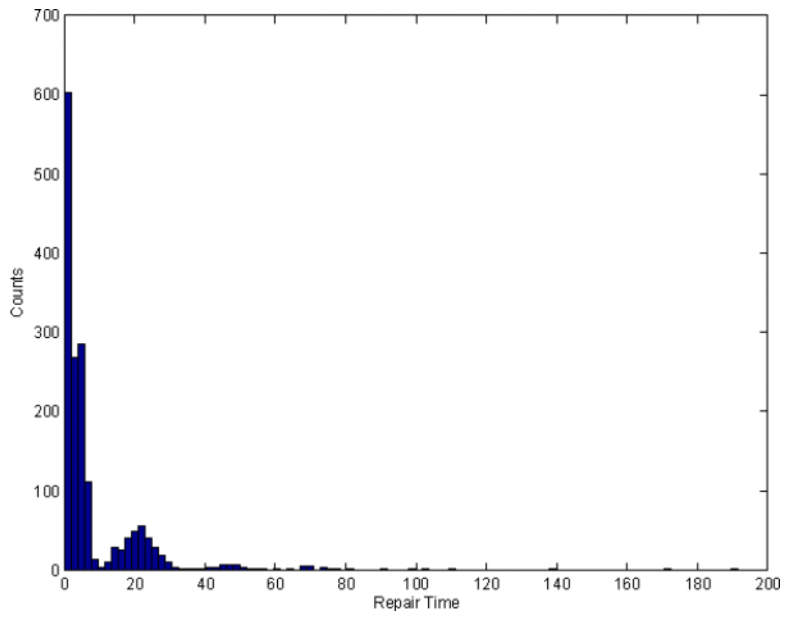

Figure 1.
Histogram of 1,664 repair times for Verizon's customers.

Table 1

Comparison of algorithms for calculating importance weights for the Verizon repair time data set. There are n = 1, 664 observations and B_1 = 1, 000 bootstrap replicates.

Algorithm	Iters	$s(p^*)$ ($\times 10^{-6}$)	Time
Naive	11586	4.666920	2023.1764
q = 1	24	4.665277	11.7226
q = 2	19	4.665277	8.9670
q = 3	16	4.665277	7.1562
q = 4	16	4.665277	6.9169
q = 5	17	4.665276	7.0566
q = 6	17	4.665277	6.7434
q = 7	18	4.665277	6.8271
q = 8	19	4.665277	7.0829
q = 9	20	4.665277	7.2667
q = 10	21	4.665277	7.4299
Int-Pt (Hessian)	18	4.665276	1247.8343
Int-Pt (BFGS)	69	4.665276	149.7225
Int-Pt (LBFGS)		Refer to Table 2	

Table 2

Comparison of our quasi-Newton acceleration and the LBFGS methods on the Verizon data set.

	Quasi-Newton			Int-Pt (LBFGS)		
q	**Iters**	**$s(p^*)(\times 10^{-6})$**	**Time**	**Iters**	**$s(p^*)(\times 10^{-6})$**	**Time**
1	21	4.66528164	11.3508	149	4.66527741	18.6839
2	17	4.66527843	8.5021	90	4.66527742	12.1505
3	15	4.66527728	7.0560	75	4.66527741	9.9297
4	15	4.66527699	6.6824	89	4.66527742	12.2690
5	16	4.66527679	7.1644	78	4.66527741	11.3548
6	16	4.66527712	7.3846	83	4.66527742	12.1189
7	17	4.66527710	6.3872	76	4.66527742	12.1928
8	18	4.66527704	6.9555	74	4.66527742	11.7458
9	19	4.66527699	6.8392	67	4.66527741	10.6362
10	20	4.66527703	7.0096	67	4.66527742	10.2277

Algorithm 1

Optimal Importance Weights

```
Initialize: $p = \left(\frac{1}{n}, \dots, \frac{1}{n}\right)$
repeat
    $c_b = T\left(x_b^*\right)^2 \Pi_{j=1}^{n}\left(n p_j\right)^{-m_{b_j}^*}, \ b = 1, \dots, B_1$
    $v_b = \left(m_{b_1}^* p_1^{-1}, \dots, m_{b_n}^* p_n^{-1}\right)^t, \ b = 1, \dots, B_1$
    $D_b = \mathrm{diag}\left(m_{b_1}^* p_1^{-2}, \dots, m_{b_n}^* p_n^{-2}\right), \ b = 1, \dots, B_1$
    $u = \sum_b c_b v_b$
    $D = \sum_b c_b (D_b + \|v_b\|^2 I_n)$
    project $x = D^{-1/2} u + D^{1/2} p$ onto
    $K = \left\{y \in \mathrm{R}^n : \sum_i d_i^{-1/2} y_i = 1, \ y_i \geq d_i^{1/2} \epsilon\right\}$
    $p = D^{-1/2} P_K(x)$
until convergence occurs
```

Algorithm 2

Michelot Algorithm: Project $x \in \mathbb{R}^n$ onto the truncated simplex $K = \left\{ y \in \mathbb{R}^n : \sum_{i=1}^{n} \alpha_i y_i = c, \quad y_i \geq \epsilon_i \quad \text{for all} \quad i \right\}$

```
repeat
    Project $x$ onto the hyperplane $H = \{x : \sum_i \alpha_i x_i = c\}$ via the map
    $P_H(x) = x - \frac{a^t x - c}{\| a \|^2} a$
    Set $x_i = \max\{x_i, \epsilon_i\}$, $i = 1, \ldots, n$
    If some $x_i < \epsilon_i$, then eliminate $x_i$ from further consideration
until $x_i \geq \epsilon_i$ for all $i$
```

Algorithm 3

Quasi-Newton Acceleration of an Algorithm Map F for Minimizing the Objective Function O

```
Initialize: $x^0$, $q$
for $i$=1 to $q + 1$ do
    $x^i = F(x^{i-1})$
end for
$u_i = x^i - x^{i-1}$, $v_i = x^{i+1} - x^i$, $i = 1,\ldots, q$
$U = [u_1 \cdots u_q]$, $V = [v_1 \cdots v_q]$
$x_n = x^{q+1}$
repeat
    $x_1 = F(x^n)$
    if $[O(x_1) - O(x^n)] \le \epsilon[|O(x^n)| + 1]$ then
        break
    end if
    $x_2 = F(x_1)$
    update the oldest $u$ in $U$ by $u = x_1 - x^n$
    update the oldest $v$ in $V$ by $v = x_2 - x_1$
    $x_{qn} = x_1 + V(U^tU - U^tV)^{-1}U^tu$
    if $x_{qn}$ falls outside the feasible region then
        project $x_{qn}$ onto feasible region
    end if
    if the objective function satisfies $O(x_{qn}) < O(x_2)$ then
        $x^{n+1} = x_{qn}$
    else
        $x^{n+1} = x_2$
    end if
until convergence occurs
```