Honours Computer Science Thesis: Implementation of Variance Reduction Techniques to BigSurvSGD

Andrew Cheng

December 2020

The **coxph** algorithm found in the Survival package of R is the standard computational survival analysis model. However, its performance scales poorly with big data. This is due to two main issues: (i) floating-point inaccuracies accumulated from computing the Hessian (or full gradient if using gradient descent) and (ii) R's poor performance memory management resulting in substandard estimations of the true coefficients and low concordance scores [1]. This renders the algorithm useless for the big data setting. As the use of electronic medical datasets increase, the need for survival algorithms that perform well under large scale problems become increasingly desired. Work has been done to speed up **coxph()**'s run-time (from $\mathcal{O}(n^2)$ to $\mathcal{O}(n)$) [2] but remain prone to numerical instability. This algorithm presents a survival computational method, BigSurvSGD, that is robust to big data as well as three modern variance reduction techniques: SAGA, SVRG, and k-SVRG. The first part of the paper introduces the aforementioned algorithms, their shortcomings, and their practicality. The second part provides simulation studies illustrating that incorporating SAGA can improve performance of BigSurvSGD on large scale data while incorporating SVRG fails to do so. The final part analyzes these results and discusses future work to be done.

1 Introduction

Survival analysis is a branch of statistics that extracts information from data to predict an individual's risk of experiencing an event of interest. It is widely used in the financial, medical, engineering, and scientific fields. For example, insurance firms aim to predict the death of an insured individual and doctors aim to predict an individual's risk of illness. In practice, we often deal with incomplete survival data known as "censored data" in which the individuals' true time-to-event is unknown due to a variety of reasons. For example, the individuals may drop out of an ongoing study or a study may conclude before individuals experience the event of interest. Survival analysis models aim to leverage the structure of censored data in order to gain as much information as possible without naively discarding these censored data. One such method is the Cox proportional hazard model.

The standard survival analysis model is the Cox proportional hazard (CoxPH) (Cox, 1972). It is a regression method that assumes a semi-parametric relationship between the covariates of an individual and the risk of experiencing the event of interest at any given time. To achieve the optimal parameters for CoxPH, it is standard to maximize a log-convex function known as the "partial likelihood." The gold-standard **survival** package in **R** utilizes Newton-Raphson, a second-order descent method to solve this convex problem

[3]. Although much work has been done in speeding up the computation [4], computational studies have shown that CoxPH still suffers from numerical instability when dealing with large datasets and high dimensional data. As a result, this model has been restricted to small to moderate datasets with few features [4]. This would motivate Tarkhan and Simon to propose BigSurvSGD which uses a stochastic first order method in place of Newton-Raphson resulting in efficient theoretical and computational outcomes [1].

However, for data sets of moderate and relatively large sizes, BigSurvSGD's runtime is much slower than **coxph()**. Our goal is to further improve the BigSurvSGD algorithm by applying the following variance reduction techniques: SAGA and SVRG. The next section introduces the background of variance reduction techniques and their shortcomings as well as the details of the Cox proportional hazard model and BigSurvSGD.

2 Variance Reduction Techniques

In machine learning, it is common to minimize an optimization problem with the following finite sum structure:

$$f(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x) \tag{1}$$

where $x \in \mathbb{R}^d$, each f_i L-smooth. Often, we are interested in incorporating a convex regularization function $h : \mathbb{R}^d \to \mathbb{R}$ that may not be differentiable resulting in minimizing the following structure:

$$F(x) = f(x) + h(x), \tag{2}$$

where the proximal operation of h is computationally feasible. In practice, h often assumes a sparsity inducing penalty to prevent overfitting in high-dimensional data. We assume h is non-differentiable and convex (e.g., lasso penalty).

One standard first-order method in minimizing (2) is proximal gradient descent (PGD), which can be described by the following update rule for iterations $t = 1, 2, \ldots$

$$x^{(t)} = \underset{\text{h,t}}{\text{prox}}\left(x^{(t-1)} - \frac{\gamma_t}{n}\sum_{i=1}^{n} \nabla f_i(x^{(t-1)})\right), \tag{3}$$

where $\underset{\text{h,t}}{\text{prox}}(x) = \underset{z\in\mathbb{R}^d}{\arg\min}\frac{1}{2t}||z - x||_2^2 + h(z)$. If we set $h(x) = 0$, then the update reduces to vanilla gradient descent. In the big data setting, (proximal) gradient descent becomes computationally infeasible as each iteration requires the computation of n derivatives (and the proximal operator). To offset the computational costs, one may use the stochastic proximal gradient descent (SPGD) which is the stochastic variant of PGD. In each iteration $t = 1, 2, \ldots$ we pick (uniformly) at random $i_k \in \{1, 2 \ldots, n\}$, and take the following update:

$$x^{(t)} = \underset{\text{h,t}}{\text{prox}}\left(x^{(t-1)} - \gamma_t \nabla f_{i_k}(x^{(t-1)})\right) \tag{4}$$

Each iteration of PSGD requires the computation of one single gradient and the proximal operator. The caveat is that we introduce too much variance by randomly selecting which gradient to compute. However, this method does not converge due to the stochasticity introduced. The introduced variance forces us to select a diminishing learning rate $\gamma_t = O(\frac{1}{t})$ to ensure convergence, albeit a slower convergence rate.

We may opt for batch PGD which interpolates between the computational cost of PGSD and the convergence rate of PGD. However, recent work in variance reduction techniques enable even faster convergence rates than PGD while maintaining computational feasibility of PGSD which is exactly what we need to deal with big data.

2.1 Intuition Behind the Reduction Techniques

Consider an unbiased estimator X for a parameter θ. Now consider the modified estimator $Z := X - Y$, where $E(Y) \approx 0$. Then clearly $E(Z) \approx \theta$. The variance of Z is:

$$Var(Z) = Var(X) + Var(Y) - 2Cov(X, Y),$$

and we observe that if the correlation between X and Y is high then $Var(Z)$ is much smaller than $Var(X)$. In otherwords, we constructed a modified unbiased estimator Z of θ which has lower variance than our original estimator X. SAGA, SVRG, and k-SVRG all follow this line of thought.

2.2 SAGA

First, create a table to store the gradients g_i of $f_i, i = 1, \dots, n$. Then, initialize the vector $x^0 \in \mathbb{R}^d$ and let $g_i^{(0)} = x^{(0)}, i = 1, \dots, n$. At iterations $t = 1, 2, \dots,$ we randomly sample (uniformly) $i_t \in \{1, \dots, n\}$ and modify $g_{i_t}^{(t)} = \nabla f_{i_t}(x^{(t-1)})$. Then update

$$x^{(t)} = x^{(t-1)} - \gamma_t \left(\underbrace{g_{i_k}^{(t)}}_{X} \underbrace{- g_{i_k}^{(t-1)} + \frac{1}{n} \sum_{i=1}^{n} g_i^{(t-1)}}_{Y} \right) \tag{5}$$

By uniformly sampling $i_t \in \{1, \dots, n\}$, we have $E(X) = \nabla f(x^{(t)})$. Furthermore, $E(Y) = 0$, so $X - Y$ is an unbiased estimator of the full gradient $\nabla f(x^{(t)})$. Y appears to be correlated with X hence should follow the aforementioned reduction technique. In particular we have $X - Y \xrightarrow{t \to \infty} 0$ since $x^{(t-1)}$ and $x^{(t)}$ converge to the optimum x^* which renders the difference between the first two sums in the big bracket to be zero. The third term converges to zero since it converges to the gradient (at the optimum). It follows that the overall estimator $||X - Y||_2 \to 0$ hence variance decays to 0.

2.3 SVRG

Recall the form of the function we wish to optimize (2). Stochastic variance reduced gradient (SVRG) aims to reduce the number of computations. This is done by creating an unbiased estimator for the full gradient used in (3) but with a lower variance than the estimator used in (4). The process is as follows. First initialize two vectors $\tilde{x}$, x^0 and set $x^0 = \tilde{x}$. Then, after every m iterations (starting at iteration 1) calculate and store the full gradient $\tilde{g} = \nabla f(\tilde{x})$. Following this, over the next $k = 1, \dots, m$ iterations, we pick a uniformly random $i_k \in \{1, \dots, n\}$, and use it to compute an estimate for the full gradient (denoted by

g_k), and our next estimate (denoted $x^{(k)}$)

$$
\begin{aligned}
g_k &= (\nabla f_{i_k}(x^{k-1}) - \nabla f_{i_k}(\tilde{x})) + \tilde{g} \\
x^{(k)} &= x^{(k-1)} - \gamma_t g_k
\end{aligned}
$$

It is important to see that during these m iterations, we are only computing the gradient for one of the functional components of $f(x)$. After repeating this process m times, we set $\tilde{x} = \frac{1}{m}\sum_{i=1}^{m} x^i$, re-evaluate the full gradient $\tilde{g}$, and repeat the iterative steps until we reach convergence. At every step, g_k is an unbiased estimator for $\nabla f(x)$, and as shown in the previous section, has reduced variance.

2.4 Shortcomings of SVRG and SAGA

In this section, we present the practical shortcomings of SVRG and SAGA when dealing with large datasets in explicit detail. There are three issues with SVRG:

1. **Uneconomical:** On big datasets, the computation of the full gradient may take hours and during this time, no progress towards the optimal solution is made.
2. **Discontinuity:** The theory requires SVRG to restart at every snapshot resulting in discontinuous behaviour, i.e., non-smooth convergence.
3. **Non-rigorous:** Theoretical justifications require snapshots to be updated every $\Omega(k)$, where $k = L/\mu$. This poses a problem when k is large since this implies that SVRG will rely on obsolete deterministic information. In practice, the update interval is recommended to be set to $\mathcal{O}(n)$, without theoretical justification.

SAGA sacrifices an additional $\mathcal{O}(dn)$ memory costs by maintaining a table of gradients in order to treat every iterate as a "partial snapshot" point. This effectively reduces the obsoleteness of SVRG's snapshots thus ameliorates the "stopping phase" problem of SVRG. Of course, SAGA's high memory cost renders it useless when dealing with large scale problems.

2.5 *k*-SVRG

k-SVRG is an interpolation between SAGA and SVRG: a table of size much smaller than SAGA of snapshots are maintained and the inner-loop in which the iterates are updated is much shorter than SVRG in order for more frequent updates of the snapshots. As we will see, the implementation of *k*-SVRG does not require a full-pass of the data at the end of the outer loop as seen in SVRG. In short, *k*-SVRG aims to incorporate the limited memory use of SVRG and the frequent update of snapshots of SAGA.

2.6 The Algorithm

First, we present the notation used and explicitly discuss how the algorithm deviates from SAGA and SVRG. Then we present the algorithm and its results.

2.6.1 Notation:

There are two updates in the algorithm: updates that occur in the *outer* for-loop, and updates that occur in the *inner* for-loop. We define the iterates of the algorithm by x_t^m, where t denotes the iteration number of the inner for-loop, and m as the iteration number of the *outer* for-loop. We maintain a set of snapshots $\Theta \subseteq \mathbb{R}^d$ of cardinality $\tilde{\mathcal{O}}(k \log k)$. We denote $\theta_i^m \in \Theta$ for the snapshot corresponding to the component f_i in the m^{th} out for-loop. Explicitly, $\Theta^m := \{\theta_i^m | i \in \{1, \ldots, n\}\}$.

Remark: In SVRG the cardinality of Θ is 1, and in SAGA, the cardinality of Θ is n.

The update of our algorithm reads as:

$$x_{t+1}^m = x_t^m - \eta g_{i_t}^m(x_t^m),$$

where,

$$g_{i_t}^m(x_t^m) = \nabla f_{i_t}(x_t^m) - \nabla f_{i_t}(\theta_{i_t}^m) + \frac{1}{n}\sum_{i=1}^n \nabla f_i(\theta_i^m)$$

This is almost the exact update structure as SVRG with one exception. In SVRG, the snapshots θ_{i_t} are all identical since $\text{Card}(\Theta) = 1$. To make the algorithm more coherent, we define

$$\alpha_i^m := \nabla f_i(\theta_i^m), \qquad \bar{\alpha}^m := \frac{1}{n}\sum_{i=1}^n \alpha_i^m$$

2.6.2 Algorithm and Properties

As mentioned before, there are two for-loops in the algorithm. The structure of these loops are very similar to SVRG. There are *two additional differences* that should be made explicit. First, as in SVRG, a new snapshot point is computed as an average of the iterates x_t^m. However, for μ-strongly convex functions, we use a *weighted average* of the iterates of the form

$$\tilde{x}^{m+1} := \frac{1}{S_\ell}\sum_{t=0}^{l-1}(1-\eta\mu)^{\ell-1-t} x_t^m,$$

where S_ℓ is the normalization constant (to be defined later). For non-convex functions, we set $\mu = 0$, and the weighted average is used. $\tilde{x}^{m+1}$ will be used as a proxy to update the snapshots in the set Θ^{m+1}.

Second, in k-SVRG, we construct a set Φ^m, to keep track of the selected indices of the m^{th} outer loop. This set serves two purposes: (i) we update the snapshot points (before moving to the $(m+1)^{th}$ outerloop) as follows:

$$\theta_i^{m+1} := \begin{cases} \theta_i^m, & i \notin \Phi^m, \\ \tilde{x}^{m+1}, & \text{otherwise} \end{cases}$$

(ii) The set Φ^m enables us to store only one copy of $\tilde{x}^{m+1}$ in memory, rather than having to store $\text{Card}(\Phi^m)$ many copies.

Algorithm 1 k-SVRG-V1 / k-SVRG-V2(q)

```
1: goal minimize $f(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x)$
2: init $x_0^0$, $\ell$, $\eta$, $\mu$, $\alpha_i^0$ $\forall i \in [n]$, $\bar{\alpha}^0 \leftarrow \frac{1}{n}\sum_{i=1}^{n} \alpha_i^0$
3: $S_\ell \leftarrow \sum_{i=0}^{\ell-1}(1-\eta\mu)^i$
4: for $m = 0 \dots M-1$
5:     init $\Phi^m \leftarrow \emptyset$
6:     for $t = 0 \dots \ell - 1$
7:         pick $i_t \in [n]$ uniformly at random
8:         $\alpha_{i_t}^m \leftarrow \nabla f_{i_t}(\theta_{i_t}^m)$
9:         $x_{t+1}^m \leftarrow x_t^m - \eta\big(\nabla f_{i_t}(x_t^m) - \alpha_{i_t}^m + \bar{\alpha}^m\big)$
10:        $\Phi^m \leftarrow \Phi^m \cup \{i_t\}$
11:    end for
12:    $\tilde{x}^{m+1} \leftarrow \frac{1}{S_\ell}\sum_{t=0}^{\ell-1}(1-\eta\mu)^{\ell-1-t} x_t^m$
13:    $x_0^{m+1} \leftarrow x_\ell^m$
14:    if variant $k$-SVRG-V2($q$)
15:        $\Phi^m \leftarrow$ sample without replacement $(q, n)$
16:    end if
17:    $\theta_i^{m+1} \leftarrow \begin{cases} \tilde{x}^{m+1}, & \text{if } i \in \Phi^m \\ \theta_i^m, & \text{otherwise} \end{cases}$
18:    $\bar{\alpha}^{m+1} \leftarrow \bar{\alpha}^m + \frac{1}{n}\sum_{i\in\Phi^m} \nabla f_i(\theta_i^{m+1}) - \frac{1}{n}\sum_{i\in\Phi^m} \nabla f_i(\theta_i^m)$
19: end for
20: return $\tilde{x}_M$
```

Pseudo-code for *k*-SVRG [5]: Note that line 14 provides an alternative construction of Φ^m, where $q = \ell = \lceil n/k \rceil$ indices are sampled without replacement from $[n]$. This is known as the k-SVRG-V2(q) algorithm. We restrict our focus on k-SVRG-V1.

Memory Requirement. The expected number of uniform samples required to sample every index in $[n]$ is $\Theta(n \log n)$.[1] Every iteration of the outer loop samples ℓ indices in $[n]$. It follows that $\mathcal{O}(k \log k)$ out loops are needed to sample all indices at least once and that $\mathcal{O}(k \log k)$ snapshots are different at any time. Therefore only $\tilde{\mathcal{O}}((dk + n)\log k)$ memory suffices to run k-SVRG-V1. This is inexpensive compared to the $\mathcal{O}(dn)$ memory requirements of SAGA when n is large.

Remark: It is important to see that no full pass over the data is required in k-SVRG as opposed to SVRG. In line 12, $\tilde{x}^{m+1}$ can be computed on the fly by creating an extra variable.

2.6.3 Summary of Convergence Results and Experiments

We restrict our discussion of convergence rates to strongly convex functions, i.e., assume f to be μ-strongly convex for $\mu > 0$.

Remark 1: Both k-SVRG-V1 and k-SVRG-V2 have linear convergence rates. In fact, the rates of these algorithms do not fundamentally differ from the rates of SVRG and SAGA (the same convergence factor $(1 - \eta\mu)$ appears in all three classes of algorithms).

Remark 2: As far as we know, there is no rigorous justification of the selection of the hyperparmeter k, thus hypertuning k is suggested to select the optimal k. However, empirical studies on larger datasets in the original paper suggests that selecting larger k does not

[1]This is just the Coupon Collector problem. The proof is provided in the appendix.

seem to inhibit performance. Hence the authors of k-SVRG recommend to pick k according to the available computing resources of the user. In particular, large k should be used for systems with fast memory (RAM) and smaller k for slower memory access. The following table [5] summarizes the details of the three reduction techniques:

method	complexity	additional memory	*in situ* ∇f_i comp.	no full pass
Gradient Descent	$\mathcal{O}(n\kappa \log \frac{1}{\epsilon})$	$\mathcal{O}(d)$	$\mathcal{O}(n)$	✗
SAGA	$\mathcal{O}((n+\kappa) \log \frac{1}{\epsilon})$	$\mathcal{O}(dn)$	$\mathcal{O}(1)$	✓
SVRG	$\mathcal{O}((n+\kappa) \log \frac{1}{\epsilon})$	$\mathcal{O}(d)$	$\mathcal{O}(n)$	✗
SCSG	$\mathcal{O}((\frac{\kappa}{\epsilon} \wedge n+\kappa) \log \frac{1}{\epsilon})$	$\mathcal{O}(d)$	$< n$	✓
k-SVRG	$\mathcal{O}((n+\kappa) \log \frac{1}{\epsilon})$	$\mathcal{O}((dk+n) \log k)$	$\mathcal{O}(\frac{n}{k})$	✓

Table 1: Comparison of running times and (additional) storage requirement for different algorithms on strongly convex functions, where $\kappa = L/\mu$ denotes the condition number. Most algorithms require *in situ* computations of many $\nabla f_i(x)$ for the same x without making progress. The longest such stalling phase is indicated, sometimes amounting to a full pass over the data (also indicated).

3 Survival Analysis

3.1 Survival Data

Suppose $T_1, \ldots, T_n$ are the true time-to-events (or survivor times) and $C_1, \ldots, C_n$ are the corresponding right-censoring times for our n observations. A survival data point for the i-th observation consists of the triple (X_i, U_i, δ_i) where $X_i \in R^p$ corresponds to the subject's covariates, $U_i = \min\{T_i, C_i\}$ corresponds to the observed time-to-event, and $\delta_i = \mathbf{1}_{[\text{i is uncensored}]}$ is an indicator function taking values 1 if the i-th observation is uncensored and 0 if censoring occurs.

The response variable T is known as *time-to-event* (survival times) which corresponds to the time elapsed before a subject experiences the event of interest. In order to properly quantify this time, we must define an origin (e.g. the time of which the study begins). Call this origin $T = 0$.

3.2 Hazard Functions

The *hazard function* is defined as

$$h(t) = \lim_{h \longrightarrow 0} \frac{P(t \leq T < t+h | T \geq t)}{h} \tag{6}$$

which gives the instantaneous potential per unit time for the event to occur conditioning on the individual surviving up to time t. We can also think of the hazard function as the probability density of having an event at time t given that the subject has not had an event up to that time.

A common model in estimating the hazard function is the *proportional hazards model* which assumes the underlying form:

$$h(t|X) = \underbrace{h_0(t)}_{\text{function of t and not X}} \times \underbrace{g(X)}_{\text{function of X and not t}} \tag{7}$$

where $h_0(t)$ is the baseline hazard function (independent of covariates) and g is some function of the covariates [4]. This factorization implies

$$\frac{h(t|X = x_1)}{h(t|X = x_2)} = \frac{g(x_1)}{g(x_2)} \tag{8}$$

meaning that the hazard ratio corresponding to any two subjects is independent of time.

3.3 Cox Proportional Hazards Model

This section follows the notation and results of Tarkhan et al. [1].

Suppose for each subject we have a corresponding survival time T and vector of covariates $X = \big(X(1), \ldots, X(p)\big)$. Then CoxPH assumes the form:

$$h(t, X; \beta^*) = h_0(t) \exp(f_{\beta^*}(X)) \tag{9}$$

Where f_{β^*} is a specified function of parameters $\beta^* = (\beta_1^*, \cdots, \beta_k^*)$ that determines the role played by X in the hazard; and $h_0(t)$ is a baseline hazard function [4]. We assume that the covariates are independent and that the baseline hazard function is independent of covariates. We have that $f_\beta(X) = X^T\beta$, when $k = p$. This model assumes that the manner in which a patient's covariates modulate their risk of experiencing an event is independent of time.

The aim is to estimate β, thus we assume that we have a dataset with n independent observations drawn from the CoxPH model: $\mathcal{D}^{(n)} = \{\mathcal{D}_i = (y_i, \mathbf{x}^{(i)} | i = 1, \cdots, n)\}$. The log-partial-likelihood is used in order to conduct estimation, it is given by

$$pl^{(n)}(\beta|\mathcal{D}^{(n)}) = \sum_{i=1}^{n} \left(f_\beta(x^{(i)}) - \log(\sum_{j\in\mathcal{R}_i} \exp(f_\beta(x^{(j)})) \right) \tag{10}$$

where $\mathcal{R}_i = \{j | t_j \geq t_i\}$ is the "risk set for patient i". Aside from $\mathcal{R}_j$, the log-partial-likelihood is independent of event times.

3.4 Estimation by Minimizing the Negative Log-Partial-Likelihood

An estimate of β^* can be obtained using the log-partial-likelihood as follows

$$\hat{\beta}^{(n)} = \text{argmin}_\beta \left\{ -pl^{(n)}(\beta|\mathcal{D}^{(n)}) \right\} \tag{11}$$

when the linear function $f_\beta(X) = X^T\beta$ is used. Note that the negative log-partial likelihood, the function we're trying to minimize, is convex. Hence, in theory, gradient descent can be employed to solve (11). Taking the gradient of the negative log partial likelihood yields:

$$\nabla_\beta \left\{ -pl^{(n)}(\beta|\mathcal{D}^{(n)}) \right\} = -\sum_{i=1}^{n} \left(\dot{f}_\beta(x^{(i)}) - \frac{\sum_{j\in\mathcal{R}_i} \dot{f}_\beta(x^{(j)}) \exp(f_\beta(x^{(j)}))}{\sum_{j\in\mathcal{R}_i} \exp(f_\beta(x^{(j)}))} \right) \tag{12}$$

where $\dot{f}_\beta(X) = \nabla_\beta\{f_\beta(X)\}$.

Note that while the gradient can be written as a sum over indices $i = 1, ..., n$, the denominator for the $i = 1$ term involves all observations in the dataset. This is exactly why stochastic gradient descent can not work as there is no natural decoupling of the dataset due to the risk sets.

When the times are ordered by $t_1 < \cdots < t_n$, it follows that $\mathcal{R}_i = \mathcal{R}_{i+1} \cup \{i\}$ which allows the use of cumulative sums and differences to calculate the entire gradient in $O(n)$ computational complexity with a $n \log(n)$ complexity sort required at the beginning of the algorithm [1]. However, with large observations n and features p this algorithm is susceptible to round off issues.

4 BigSurvSGD

In this section, we introduce the Big Survival Data Analysis via Stochastic Gradient Descent (BigSurvSGD) algorithm.

We consider $\beta^{(s)}$, which is defined as the population minimizer of the expected partial likelihood of s random patients, referred to as "strata of size s". It is defined as follows:

$$\beta^{(s)} = \text{argmin}_\beta \left\{ \mathbb{E}_s[-pl^{(n)}(\beta|\mathcal{D}^{(n)})] \right\} \tag{13}$$

$D^{(s)}$ is interpreted as a draw of s random patients from the population, where the minimum value of s is 2, otherwise the log-partial-likelihood is zero. When the assumptions of the Cox model hold it follows that $\beta^{(s)} = \beta^* \ \forall s$.

In order to estimate β^*, a small fixed $s << n$ is selected and stochastic gradient descent is applied to the population optimization problem above. In practice this will amount to calculating stochastic gradients using random strata of size s. One may note that for s small, there are on the order of n^s such strata. However, results for stochastic gradient descent indicate that under strong convexity of population optimization problem above, the order of n steps should be required to obtain a rate optimal estimator (convergence at a rate of n^{-1} in MSE) [1].

4.1 BigSurvSGD Algorithm

Suppose we have n_s independent strata, $D_1^{(s)}, \cdots, D_{n_s}^{(s)}$ each with s independent patients drawn from our population (with $s \geq 2$). We let I_m denote the indices of patients in strata $D_m^{(s)}$ for each $m \leq n_s$.

When X are bounded and $f_\beta(X)$ is Lipschitz, we have that for any β:

$$\nabla_\beta \mathbb{E}_s \left[-pl^{(s)}(\beta|\mathcal{D}_m^{(s)}) \right] = \mathbb{E}_s \left[\nabla_\beta \left\{ -pl^{(s)}(\beta|\mathcal{D}_m^{(s)}) \right\} \right] \tag{14}$$

for all $m \leq n_s$.

Note that $\nabla_\beta \left\{ -pl^{(s)}(\beta|\mathcal{D}_m^{(s)}) \right\}$ is defined using $D_m^{(s)}$ as follows:

$$\nabla_\beta \left\{ -pl^{(s)}(\beta|\mathcal{D}_m^{(s)}) \right\} = -\sum_{i \in I_m} \left(\dot{f}_\beta(x^{(i)}) - \frac{\sum_{j \in \mathcal{R}_i^m} \dot{f}_\beta(x^{(j)}) \exp(f_\beta(x^{(j)}))}{\sum_{j \in \mathcal{R}_i^m} \exp(f_\beta(x^{(j)}))} \right) \tag{15}$$

where $\mathcal{R}_i^m = \{j | t_j \geq t_i \text{ and } i, j \in I_m\}$ are risk sets that include only patients in stratum m; and $\dot{f}_\beta$ is the gradient of f with respect to β.

We restrict our setting to the simplest version of stochastic gradient descent (SGD) algorithm for $\hat{\beta}^{(n)} = \text{argmin}_\beta \left\{ -pl^{(n)}(\beta|\mathcal{D}^{(n)}) \right\}$ used in BigSurv is as follows:

Choose an initial $\hat{\beta}(0)$ and at each iteration $m = 1, \cdots, n_s$ we update our estimate by

$$\hat{\beta}(m) = \hat{\beta}(m-1) + \gamma_m \times \nabla_\beta \left\{ pl^{(s)}(\beta(m-1)|\mathcal{D}_m^{(s)}) \right\}. \tag{16}$$

Where γ_m is the learning rate that is specified in advance or is determined adaptive.

The computation time to run n_s steps of SGD is $\sim sn_s = np$ (where $n = sn_s$ is the total sample size) for (17) when f_β is linear. Note that if the ordering the risk sets is not leveraged then $\sim nps$ computation is required. Additionally, using these small strata of size s, we are not prone to round-off issues when using stochastic optimization (because this sum is calculated separately for each strata).

It has been shown that SGD algorithms for strongly convex objective-functions are asymptotically more efficient if we use a running average of the iterates as the final estimate. In this case we can set $\gamma_m = \gamma$ as long as it is sufficiently small, thus the running-average estimator is

$$\tilde{\beta}(m) = \frac{1}{m} \sum_{i=1}^{m} \hat{\beta}(i). \tag{17}$$

Strong convexity of (13) depends on f_β and X. If f_β is linear and X has a non-degenerate distribution (13) will be strongly convex. If (13) is strongly convex, then $\|\tilde{\beta}(m) - \beta^{(s)}\|_2^2 = O_p(m^{-1})$ which is the optimal rate of convergence for estimating $\beta^{(s)}$ for m observations. In the simulation studies, we choose the learning rate as $\gamma_m = \frac{C}{\sqrt{m}}$, where C is a constant. This is gives us the statistically optimal rate of $\mathcal{O}_p(m^{-1})$.

4.2 Intuition to Why BigSurvSGD Works

BigSurvSGD aims to randomly partition large survival datasets into stratas in order to avoid full passes over the data to compute the full gradient. The estimates computed by the stratas $\hat{\beta}^2$ This effectively reduces numerical instability while maintaining computational efficiency. It follows that we should expect any optimization method that requires full-passes over the data to perform poorly in optimizing the partial likelihood. In fact, the next section shows that SVRG, which require full-passes of the data, is ineffective when n is large.

It is worth mentioning that this framework can facilitate the training of complex models such as neural networks with large survival datasets [1].

5 Methodology

To apply SAGA and SVRG into BigSurvSGD, one important change had to be made to the original code of BigSurvSGD. As the gradient of the partial likelihood function is not defined for less than two observations, rather than sampling one element uniformly from our dataset $\{1, \ldots, n\}$, we sample s elements from our dataset where s is the predetermined stratum size. As SAGA and SVRG are both highly dependent on the starting point, we used the converging point of SGD (shifted by some noise) as the initial point $x^{(0)}$. This is commonly referred to as the "warm up start." Implementing SAGA and SVRG was done through edits of the `bigSurvSGD` package available in **R**. This library, written with `rcpp`, can also implement the BigSurv algorithm using an ADAM or an AMSGrad optimizer. By editing the `bigSurvSGD.R` and `oneChunkC.cpp` files, the necessary tools were added to implement SAGA and SVRG. In the `do.one` function of `bigSurvSGD.R`, functionality was added to compute the full gradient and store previous values of β. From `oneChunkC.cpp`, we implemented the necessary code to compute the update steps of SAGA and SVRG.

As previously mentioned, the learning rate is chosen to be $\gamma_m = \frac{C}{\sqrt{m}}$ is chosen as it gives the convergence rate of $||\tilde{\beta}(m) = \beta^{(s)}||_2^2 = \mathcal{O}_p(m^{-1})$. The value C = 0.12 was chosen for all our results.

6 Results

Throughout this section, the value of the log-partial-likelihood is used as the metric on which each of the gradient descent methods is evaluated. SGD is the standard method used in the `BigSurvSGD` package and will serve as our baseline. Figures 1, 2 and 3 compare the performance of the non-regularized vanilla BigSurvSGD (SGD), BigSurvSGD with SVRG, and BigSurvSGD with SAGA on the synthetic datasets of the `BigSurvSGD` package.

The synthetic data generated as follows: generate the event times according to the Cox PH model. Generate the hazard baseline $h_0(t)$ using an exponential distribution with $\lambda = 1$. Then generate the censoring and event times independently. Explicitly,

$$\beta^* = \mathbf{1}_{p\times 1},$$

$$X_i \sim Uniform(-\sqrt{3}, \sqrt{3}),$$

$$y_i \sim \exp(\mu = \exp(-X_i^T \beta^*)) \text{ (time to event/censoring)},$$

$$\delta_i \sim Bernoulli(p = 1 - p_c), \quad p_c = Pr(t_i > c_i)$$

where $y_i = min(t_i, c_i)$, is the time to event or censoring which ever comes first.

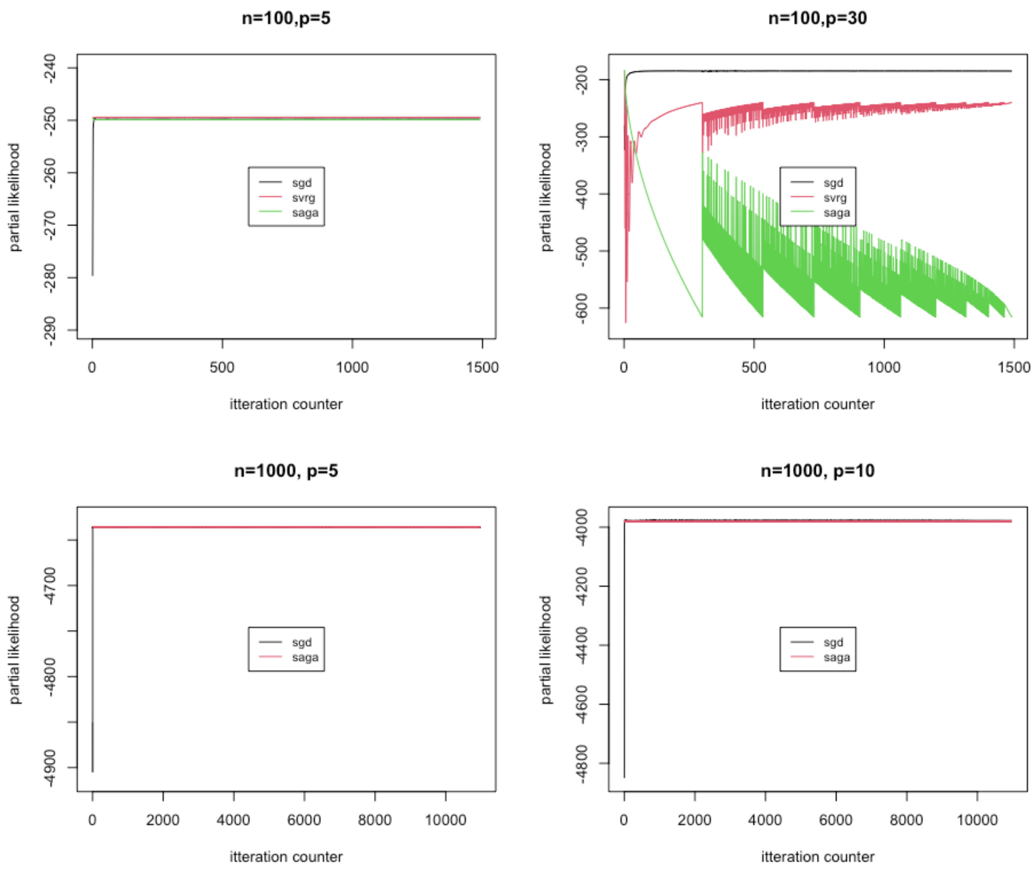

Figure 1: Partial likelihood of the **coxph()** function evaluated at each iteration. n denotes the sample size, and p denotes the feature size. For these experiments, the stratum size is fixed as 10.

From Figure 1, we observe that for small datasets with few features (top left), all 3 of the optimization methods perform equally well. When the dimension of a dataset grows however (top right) of Figure 1, both SAGA and SVRG exhibit signs of increased variance. Finally, for large datasets (bottom left and bottom right), the SVRG algorithm is subject to numerical instability as the full gradient must be computed every m steps. SAGA on the other hand performs competitively with SGD.

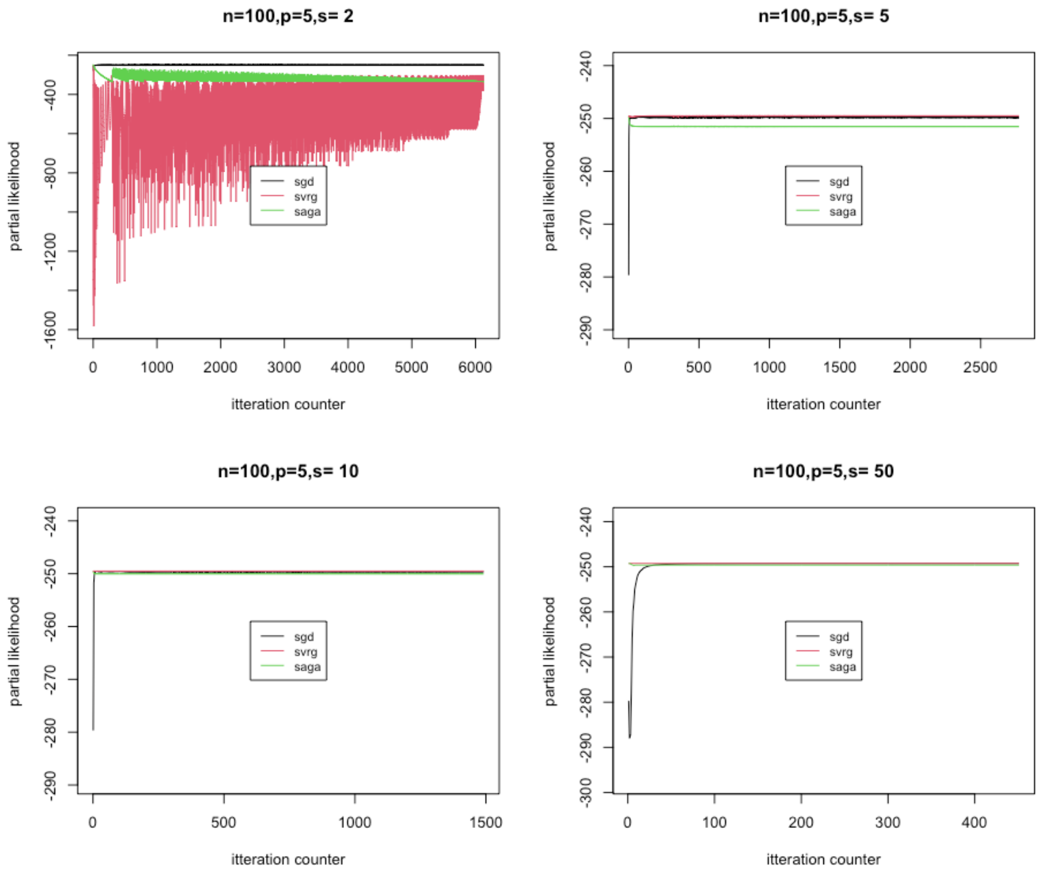

Figure 2: Partial likelihood of the **coxph()** function evaluated at each iteration. For these experiments, the sample size and feature size are fixed to 100 and 5 respectively, and the stratum size s takes on values in $\{2,5,10,50\}$.

In Figure 2, we observe the effect of strata size on small datasets. As can be seen by the top left graph, small stratum ($s = 2$) increase variance in all three of the optimization methods. SAGA and SVRG seem to be affected more drastically. With a larger choice of s, the variance of SGD, SAGA and SVRG is reduced. It is worth noting that SAGA and SVRG flattens out quicker than SGD as illustrated by the top right graph in this figure.

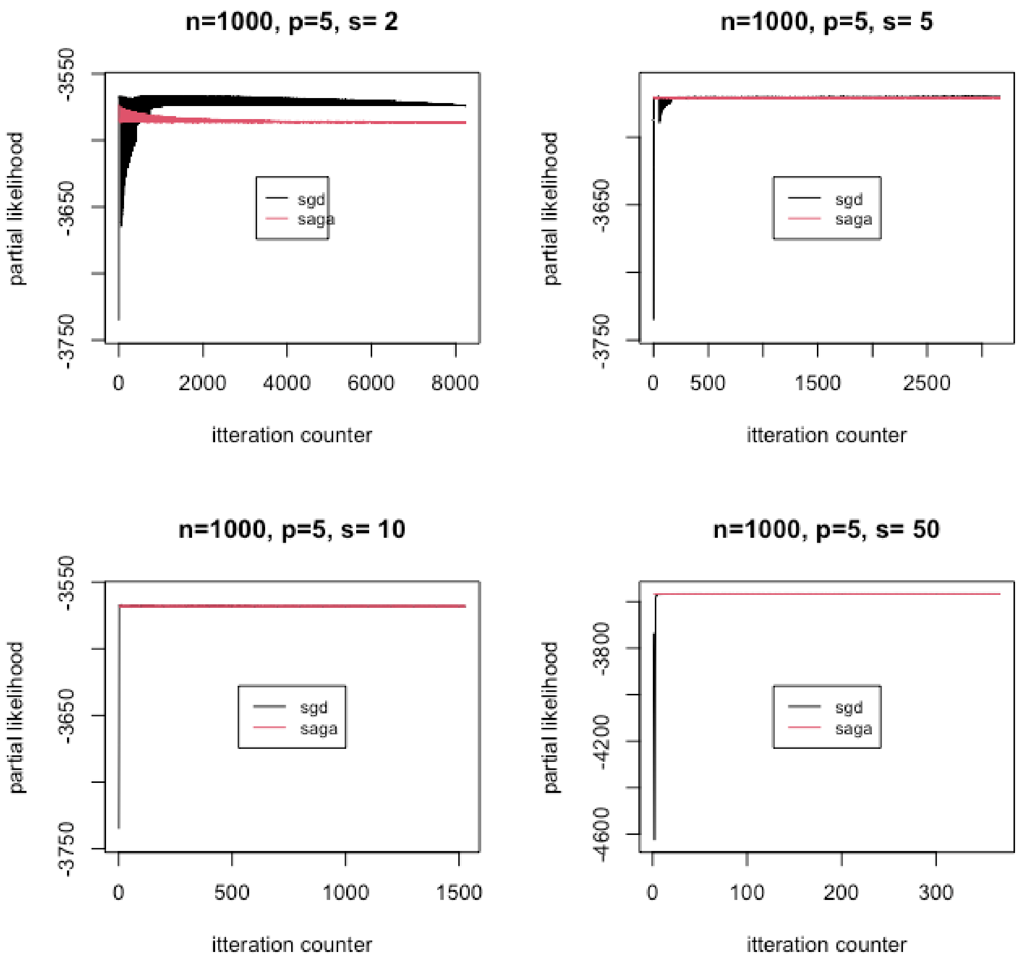

Figure 3: Partial likelihood of the **coxph()** function evaluated at each iteration. For these experiments, the samples size and feature size are fixed to 1000 and 5 respectively, and the stratum size s takes on values in $\{2,5,10,50\}$.

For large datasets, SVRG cannot readily be computed. For this reason, Figure 3 compares the effect of stratum size on larger data sets. As this figure shows, for large datasets, the effects of stratum size on variance are more evident. SGD shows evidence of much more variance at higher values of s than SAGA.

7 Discussion

General trends from the figures shows us that vanilla **BigSurvSGD** performs best on datasets with small number of observations n and that incorporating variance reduction techniques will only hinder its performance. For larger datasets $n = 1000$, we observe that incorporating SAGA improves BigSurvSGD's performance as it decreases the variance. It is not surprising that SVRG does not work for $n = 1000$, as it requires full-passes of the data, an issue that BigSurvSGD addresses. Although we made improvements by using BigSurvSGD with SAGA, it is important to remember SAGA's memory limitations. As datasets become larger, it may not be feasible to store the table of gradients and thus this approach may not work in practice. The performance of BigSurvSGD is quite dependent hyperparamater s as illustrated in figures 2 and 3.

The inefficacy of optimization methods that require full-pass over the data was evident during the simulation studies. For example, SVRG was unable to perform on datasets with $n = 1000$ as numerical instability forced `Nans` in our results. In fact, this is exactly the problem BigSurvSGD addresses: to avoid methods that require full passes on the data. We can extrapolate on this result and conclude that if an optimization technique requires full passes over the dataset, then it is likely to perform poorly when used to optimize the Cox partial likelihood on large scale data. For example, if one is using the Cox proportional hazard model on large scale data, one should avoid using gradient descent as an optimization technique.

A natural question to ask is when to use **coxph()** and when to use BigSurvSGD? The **coxph()** algorithm is considered the gold-standard computational survival model and continuously updated. In fact, the last update was in 2020. BigSurvSGD was released in 2020. Both models perform well with datasets on the magnitude of $n = 10^3$ and $d = 10^2$ [1] with **coxph()** performing slightly better on smaller datasets. In all cases, we found that **coxph()**'s runtime is significantly quicker. The caveat is that **coxph()** struggles to converge to the optimum coefficients when either d, or n, is large. As a rule of thumb, we recommend users to select **coxph()** for small datasets ($n < 10^3, p < 50$) and BigSurvSGD for moderate to large datasets (i.e., when $n \geq 10^3, p > 50$). For large datasets, we may include SAGA to improve on BigSurvSGD if necessary computing resources are available. If computing resources are limited, one may experiment with k-SVRG and set k as large as possible to see if improvements are made to the performance of BigSurvSGD.

8 Future Work

BigSurvSGD outperforms the standard cox model in **R** when dealing with large datasets simulated by Tarkan et al. However, as far as we know, datasets that do not follow the Cox proportional hazard model nor BigSurvSGD have not been tested on real-world datasets. Thus future work can be done in observing the performance of BigSurvSGD on large scale real world datasets as well as big datasets that include survival times that exhibit higher order behaviour with respect to subjects' covariates. These results could then be compared to other survival methods to test whether BigSurvSGD's superior performance on large scale data still holds.

Although not mentioned in this paper, BigSurvSGD can easily estimate the confidence interval estimated coefficient. One could explore the effects if any changes are made to these confidence intervals when variance reduction techniques are employed.

In addition, we did not analyze the behaviour of BigSurvSGD with lasso regularization and variance reduction techniques. This could be relatively easy to implement as the framework for proximal gradient descent for the lasso penalty is already implemented in the code. As mentioned in the previous section, SAGA's memory requirement limits the scope of this algorithm. One could explore the performance of BigSurvSGD with variance reduction techniques that interpolate between SVRG and SAGA, such as k-SVRG. Moreover, a plethora of additional methods could be incorporated to BigSurvSGD such as acceleration or momentum based variance reduction techniques that could improve on BigSurvSGD's performance on smaller datasets [6]. Lastly, we noticed that the current BigSurvSGD algorithm does not handle ties, rather if there are any ties present, the algorithm adds a Gaussian noise of small magnitude to the eliminate tied event times. Work could be done in implementing tie methods such as Breslow or Efron [7] to better accommodate tied survival times.

Appendix

Coupon Collector: Suppose we have n indices to sample from $\{1, \ldots, n\}$. We show that an expected $\Theta(n \log n)$ uniform samples are needed to sample each index at least once.

Proof. We invoke a Markov chain argument. Our state space $\mathcal{S} = \{0, 1, \ldots, n\}$ represents how many distinct indices we have sampled so far. The initial state $X_0 = 0$, i.e., we start off having sampled none of the indices. Of course, after the first sample, we transition from state 0 to state 1 with probability 1. Then, from state 1, we have $1/n$ probability of picking the same index (stay in the same state), and $\frac{n-1}{n}$ probability of picking a different a different index (transitioning to state 2). Generalizing this notion yields a chain that graphically

$$\begin{tikzcd}
{} \arrow[d] \\
\boxed{0} \arrow[r, "1"] & \boxed{1} \arrow[loop above, "\frac{1}{n}"] \arrow[r, "\frac{n-1}{n}"] & \boxed{2} \arrow[loop above, "\frac{2}{n}"] \arrow[r, "\frac{n-2}{n}"] & \boxed{3} \arrow[loop above, "\frac{3}{n}"] \arrow[r, "\frac{n-3}{n}"] & \boxed{4} \arrow[loop above, "\frac{4}{n}"] \arrow[r, "\frac{n-4}{n}"] & \cdots \arrow[r, "\frac{1}{n}"] & \boxed{n} \arrow[loop above, "1"]
\end{tikzcd}$$

looks like:
Now the expected number of samples required to sample every index at least once corresponds to the expected number of iterations it takes for the chain, starting from state 0, to reach state n. Denote $T_{i,j}$ as the number of iterations to go from state i to state j. Then clearly we have

$$\begin{aligned}
E\{T_{0,n}\} &= E\{T_{0,1} + T_{1,2} + \cdots T_{n-1,n}\} \\
&= E\{T_{0,1}\} + \cdots + E\{T_{n-1,n}\}
\end{aligned}$$

Since $T_{i,i+1} \sim \mathit{Geometric}(p = \frac{n-i}{n})$ we have $E\{T_{i,i+1}\} = \frac{n}{n-i}$. Hence we have

$$\begin{aligned}
E\{T_{0,n}\} &= 1 + \frac{n}{n-1} + \frac{n}{n-2} + \cdots + n \\
&= n(\frac{1}{n} + \frac{1}{n-1} + \cdots 1) \\
&= n \times H_n \\
&\sim n \log n
\end{aligned}$$

where H_n is the n^{th} harmonic number and the last line follows from the result that $H_n \sim \log n$. This is a stronger result than $\Theta(n \log n)$. [2] $\square$

[2]This is a well-known result and and perhaps its proof is just as well-known. I first saw this proof when it was presented by Professor Luc Devroye in the graduate course: Probabilistic Analysis of Algorithms at McGill University.

Acknowledgements

I would like to thank my supervisor and mentor, Professor Yi Yang, who shared his knowledge of statistical learning, exposed me to modern machine learning techniques, and guided me throughout this project. I would also like to thank Jean Djamen for sharing his knowledge of R and C++. Without Jean, I doubt I would have made as much progress as I did with the code which ultimately enabled the simulation results.

References

[1] A. Tarkhan and N. Simon, “Bigsurvsgd: Big survival data analysis via stochastic gradient descent,” 2020.

[2] N. Simon, J. H. Friedman, T. Hastie, and R. Tibshirani, “Regularization paths for cox’s proportional hazards model via coordinate descent,” *Journal of Statistical Software, Articles*, vol. 39, no. 5, pp. 1–13, 2011.

[3] T. M. Therneau, *A Package for Survival Analysis in R*, 2020. R package version 3.2-3.

[4] Terry M. Therneau and Patricia M. Grambsch, *Modeling Survival Data: Extending the Cox Model*. New York: Springer, 2000.

[5] A. Raj and S. U. Stich, “k-svrg: Variance reduction for large scale optimization,” 2018.

[6] A. Nitanda, “Stochastic proximal gradient descent with acceleration techniques,” vol. 2, pp. 1574–1582, 12 2014.

[7] J. Borucka, “Methods of handling tied events in the cox proportional hazard model,” 2014.