Parallel Implementation of Cox Regression for High-dimensional Massive Censored Dataset

Hsuan Chiu
UCLA
cherylautumn@cs.ucla.edu

1. INTRODUCTION

With the opportunity of collecting massive electronic health records (EHR), devoting medical knowledge discovery on these longitudinal observational database turns more promising than before because these post-marketing observations contains more events, such as the occurrence of adverse events with specific drug use, that cannot be observed from pre-marketing clinical trial. Longitudinal observational databases store patients' time-stamped medical information, such as periods of drug exposure and dates of diagnoses. One important feature of longitudinal observational data is *censored*, meaning the outcome of a patient may not be observed. A widely used approach for analyzing the censored data, known as time-to-event data, is survival analysis.

Survival analysis is aim to discover the relationship between event distribution and a set of explanatory variables. In clinical researches, the event can be defined as biological death or readmission, and explanatory variables can be age, tumor size, drug or gene. One of the most commonly-used method in survival analysis is the *Cox proportional hazard regression model* [1]. Traditional applications applied the Cox model in the past few decades usually consider data with a handful of predictors and a few hundreds or thousands of observations. Even though some researchers discussed high-dimensional data [3], which may extend the predictor to 10^5, the number of observations in their studies reduced to less than 10^3. Nevertheless, with the help of EHR, it is easy to obtain very-high-dimensional dataset with the massive number of observations ranging more than 10^6. This high-dimensional massive dataset present computational challenges to the Cox regression.

Cox regression is similar to the generalized linear models (GLM), such as linear regression and logistic regression, which require multiple times of iterative computations to complete the Maximum Likelihood Estimation (MLE). With the trend of big data, machine-learning communities identified there is a need to efficiently parallelizing the iterative computation in multi-core and distributed environment. In recent years, several novel parallel frameworks designed for machine learning tasks are proposed such as GraphLab [7], Piccolo [10], and Spark [12].

So far, the Cox regression is overlooked by the machine learning community and current statistic tools, such as R and SPSS, implementing the Cox regression are greatly limited by the dataset size. In this project, we leverage Spark to implement the distributed algorithm for the Cox regression. Our goal is to develop the Cox regression suite as part of Spark package. We expect our implementation not only to break the limitation of high dimensional massive dataset encountered by current tools but also to provide an alternative option for those who need to apply the Cox regression on large-scale data.

2. RELATED WORK

Cloud computing brings a new opportunity for programmers to parallel process large-scaled data. With several new data-centric programming models proposed [2][6], writing parallel applications turns easier and safer than traditional parallel in-memory implementations, such as MPI [9], because a simplified data accessing interface is provided. MapReduce [2] and Dryad [6] are pioneers to support data-flow programming model for bulk-processing of on-disk data but they are identified as not appropriate for machine learning application. Therefore, several new frameworks such as GraphLab [7], Piccolo [10], and Spark [12] are proposed to support with more expressive ability and higher efficiency for machine learning tasks. Generally, these new framework shift the disk-based computation to in-memory execution for reducing I/O cost and expediting iteration.

Spark use the Resilient Distributed Datasets (RDD) to efficiently offer fault tolerance and fast computation for applications required iterative algorithms and interactive data mining tools. RDD is a distributed memory abstraction that is designed for data reuse applications. Conventional cluster computing framework, such as MapReduce and Dryad, is inefficient for reusing the intermediate results. Data reuse is common in iterative machine learning and interactive data mining. The main feature of RDD is that it is a coarse-grained transformations and result can be rebuild following a series of transformation actions. By the first time loading data into memory, the following computations can be applied on the same dataset in the memory, which reduce the I/O cost significantly. RDD is also expressive enough to capture a wide class of computations. In this project, we use Spark as the platform for implementing Cox regression in distributed computation.

The Cox regression requires computing maximum partial likelihood estimator by solving a non-linear optimization problem. Standard statistic tools, such as R or SPSS, do not work well in the presence of high-dimensional massive dataset because these tools implement Newton-Raphson method requiring a matrix

inversion during the course of iteration. The main problem with matrix inversion is the memory limitation. One possible remedy is to apply feature selection as a pre-processing step, such as [3][13], and to reduce the number of dimensions. However, Genkin et al. [4] argued that the feature selection may choose redundant or ineffective combinations of features and the statistical consequences made by feature selection is hard to explain. Another solution is to modify the data structure of the input data. For example, Mittal et al. [8] leverage the sparsity of high-dimensional data and they implement a specialized column-wise data structures to reduce the memory requirements. However, to modify the input data structure does not actually solve the problem. Other methods [11] propose to use penalty functions, such as elastic net penalty, to improve the prediction accuracy for high-dimensional data but these implementations still do not solve the memory requirements problem. In this project, we argue that the actual effective solution is to distribute dataset and run the maximum partial likelihood in parallel.

3. METHOD

3.1 Cox Regression

Survival analysis aims to study the relationship between the survival time T and a set of explanatory factors. Two important notions for survival analysis are the survival function $S(t)$ and the hazard function $h(t)$. The survival function gives the probability that the individual is still alive at some specified time t while the hazard function indicates the rate for the event to occur per unit time, conditioned on that the individual has survived up to time t. Survival time T is a random variable with cumulative distribution function $P(t) = \Pr(T \leq t)$ and probability density function $p(t) = dP(t)/dt$. Thus, the survival function is defined as:

$$S(t) = \Pr(T > t) = 1 - P(t), \quad (1)$$

and the hazard function is defined as:

$$h(t) = \lim_{\Delta t \to 0} \frac{\Pr[(t \leq T < t + \Delta t) | T \geq t]}{\Delta t}. \quad (2)$$

The relationship between hazard function and survival function is

$$S(t) = \exp\left(-\int_0^t h(u)du\right). \quad (3)$$

As mentioned, survival analysis is to discover the relationship between survival distribution and one or more explanatory variables. Thus, it is more general to define the hazard function as depending not only on time T but also on a set of covariates vector $X = (x_1, x_2, \ldots, x_d)^T$, where d stands for the number of dimensions in the covariate space. In clinical researches, covariates may be demographic attributes, such as age, diagnosis variables, such as tumor size, or experimental treatment, such as drug or procedure. One of the most commonly-used method to model this scenario is the Cox proportional hazard model [1]:

$$h(t, X) = h_0(t)\exp(\beta^T X). \quad (4)$$

For each individual, this model provides the hazard at time t given a certain set of predictor variables X associated with that individual. The Cox model shows that the hazard at time t depends on the baseline hazard function, $h_0(t)$, and on time-independent covariates X. Here $\beta = (\beta_1, \beta_2, \ldots, \beta_d)^T$ is the regression parameter vector and $\beta \in \mathbb{R}^d$. Each individual is also associated with an outcome, $O = \{Y, \delta\}$, composed by the observed time $Y = \min\{T, C\}$ and the censoring indicator $\delta = I(T \leq C)$ where C is the censoring time and T is the survival time. Cox introduced partial likelihood for estimating β. Given a right-censored dataset contains n individuals and K events, where $K \leq n$, if only one event occurred at each time point and these event time points are ordered as $t_1 < t_2 < \cdots < t_K$, at each time point t_j, a risk set R_j represents a set of individuals whose event may be observed at t_j. The Cox's partial likelihood is expressed as

$$L(\beta) = \prod_{j=1}^{n} \left(\frac{\exp(\beta^T X_j)}{\sum_{l \in R_j} \exp(\beta^T X_l)}\right)^{\delta_j}. \quad (5)$$

The heuristics idea is to compare the risk of the failed individual to the risk of other people in the risk set at each time point. By maximizing $L(\beta)$, estimator parameters $\hat{\beta}$ can be derived. Solving maximization of the partial likelihood function $L(\beta)$ is a non-linear optimization problem. Two approaches, *Newton-Raphson* and *Cyclical Coordinate Descent*, are commonly applied in the Cox regression coefficients estimation procedure. In this project, we implement Cyclical Coordinate Descent by leveraging the distributed power provided by Spark.

3.2 Newton-Raphson Algorithm

Newton-Raphson is a common strategy to solve non-linear optimization problem. Compare to gradient descent approach, Newton-Raphson requires less number of steps to converge, which is very suitable to implement on a single machine. However, Newton-Raphson requires constructing the Hessian matrix and performing matrix inversion during the course of estimation. These two properties make Newton-Raphson algorithm hard to scale especially when having high-dimensional massive data.

Maximizing the partial likelihood is equivalent to maximizing a scaled log partial likelihood,

$$l(\beta) = \sum_{j=1}^{n} \delta_j [\beta^T X_j - \log(\sum_{l \in R_j} \exp(\beta^T X_l)]. \quad (6)$$

The first derivative of $l(\beta)$ is defined as $U(\beta)$. $U(\beta)$ is a vector and each element β_i in $U(\beta)$ is defined as

$$\frac{\partial l(\beta)}{\partial \beta_i} = \sum_{j=1}^{n} \delta_j [X_i - (\frac{\sum_{l \in R_j} X_{li} \exp(\beta^T X_l)}{\sum_{l \in R_j} \exp(\beta^T X_l)})]. \quad (7)$$

The second derivative of $l(\beta)$ is defined as $H(\beta)$, also known as the Hessian matrix. $H(\beta)$ is a $d \times d$ matrix and each element β_{ik} is defined as

$$\frac{\partial^2 l(\beta)}{\partial \beta_i \partial \beta_k} = \sum_{j=1}^{n} \delta_j [- \left(\frac{\sum_{l \in R_j} X_{li} X_{lk} \exp(\beta^T X_l)}{\sum_{l \in R_j} \exp(\beta^T X_l)} \right) + \left(\frac{\sum_{l \in R_j} X_{li} \exp(\beta^T X_l)}{\sum_{l \in R_j} \exp(\beta^T X_l)} \right) \left(\frac{\sum_{l \in R_j} X_{lk} \exp(\beta^T X_{jl})}{\sum_{l \in R_j} \exp(\beta^T X_l)} \right)]. \quad (8)$$

Constructing $H(\beta)$ is computation intensive because it requires $O(nd^2)$ for time cost and $O(d^2)$ for memory space. Numerical instability is another problem because of the matrix inversion. Distributed computation for elements in $H(\beta)$ is not simple because the summation of riskSet R_j requires seeing individuals who survive longer than individual j. In this project, we still implement the function to build Hessian matrix for estimating the p-value of coefficients. However, for iterative model training, we take the Cyclical Coordinate Descent algorithm as the main approach to break the memory limitation of high-dimensional massive dataset.

ALGORITHM 1: Newton-Raphson Algorithm

Procedure: Model construction
Input: A censored dataset *D*, *lreThreshold* and *iterMax*.
Output: Estimated coefficients β for Cox regression model.
1. Set an initial value $\hat{\beta}^{(0)} = 0$, *iter*=0, *LRE*=∞
2. **While** (*iter* < *iterMax* and *LRE* > *lreThreshold*)
3. $\hat{\beta}^{(iter++)} = \hat{\beta}^{(iter)} - H^{-1}(\hat{\beta}^{(iter)}) U(\hat{\beta}^{(iter)})$
4. LRE=$\left| \frac{\hat{\beta}^{(iter)} - \hat{\beta}^{(iter-1)}}{\hat{\beta}^{(iter-1)}} \right|$
5. **end**

3.3 Cyclical Coordinate Descent Algorithm

Seeing the limitation of Newton-Raphson algorithm caused by high-dimensional massive data, another group of researches turn to use *cyclic coordinate descent*, CLG, as a replacement. Genkin et al. [4] leveraged CLG on the logistic regression and Mittal et al. [8] applied CLG on the Cox regression. Mittal et al. designed a specialized data structure for solving the memory insufficient problem pertaining to large-scale data and their program is not designed for distributed computation. In this project, we re-implement the CLG for the Cox Regression on Spark framework and we expect the memory limitation can be truly solved by distributed computation.

A cyclic coordinate descent algorithm starts from setting all variables to some initial value. Next, it set the first variable to a value that minimizes the objective function while holding all other variables constant. For the second variable, CLG applies the same strategy and keeps holding all other variables constant (including the new value of the first variable). CLG applies the same approach until all variables have been traversed. Then, CLG starts the next iteration until the maximum number of iterations achieved or the convergence criterion is met. Compare to Hessian matrix, CLG does not require to construct, store or invert the Hessian matrix.

CLG algorithm needs to find $\hat{\beta}^{(new)}$, the value of the i^{th} entry of β that maximized $l(\beta)$, assuming that the other β_d's are held at their current values.

The value of updated coefficient $\hat{\beta}^{(new)}$ is defined as

$$\hat{\beta}^{(new)} = \beta_i - \frac{g'(\beta_i)}{g''(\beta_i)}, \quad (9)$$

where $g'(\beta_i) = -U(\beta)$ and $g''(\beta_i) = -\frac{\partial^2 l(\beta)}{\partial {\beta_i}^2}$. Computing $g'(\beta_i)$ and $g''(\beta_i)$ can be implemented in parallel. We partition data by rows, and we calculate $\exp(\beta^T X_j)$, $X_{li} \exp(\beta^T X_j)$ and ${X_{li}}^2 \exp(\beta^T X_j)$ for each individual j in parallel. Next, we collect these values back and order them by survival time in descending. Lastly, we iterate the ordered sequence at once to aggregate values and to obtain $g'(\beta_i)$ and $g''(\beta_i)$.

ALGORITHM 2: CLG Algorithm

Procedure: Model construction
Input: A censored dataset *D*, *lreThreshold* and *iterMax*.
Output: Estimated coefficients β for Cox regression model.
1. Set an initial value $\hat{\beta}^{(0)} = 0$, *iter*=0, *LRE*=∞
2. **While** (*iter* < *iterMax* and *LRE* > *lreThreshold*)
3. **For** i=1,…,d
4. $\Delta\beta_i = \min(\max\left(\frac{g'(\beta_i)}{g''(\beta_i)}, -\Delta i\right), \Delta i)$
5. $\hat{\beta}^{(new)} = \beta_i + \Delta\beta_i$
6. $\Delta i = \max(2|\Delta\beta_i|, \Delta j/2)$
7. LRE=$\left| \frac{\hat{\beta}^{(iter)} - \hat{\beta}^{(iter-1)}}{\hat{\beta}^{(iter-1)}} \right|$
8. **end**
9. **end**

3.4 Regularization

To avoid over-fitting, penalty function is often added into the objective function. In this project, we also implement L1 and L2 penalty as alternative options to train the Cox regression model. The penalized log partial likelihood of β in the L1 case, $l_{L1}(\beta)$, can be written as

$$l_{L1}(\beta) = l(\beta) - \sum_{i=1}^{d} (log 2 - log\sqrt{\gamma_i} + \sqrt{\gamma_i}|\beta_i|). \quad (10)$$

The penalized log partial likelihood of β in the L2 case, $l_{L2}(\beta)$, can be written as

$$l_{L2}(\beta) = l(\beta) - \sum_{i=1}^{d} (log\sqrt{\tau_i} - 0.5 log 2\pi + \frac{\beta_i}{2\tau_i}). \quad (11)$$

Typically, γ_i and τ_i are tuning parameters.

4. IMPLEMENTATION

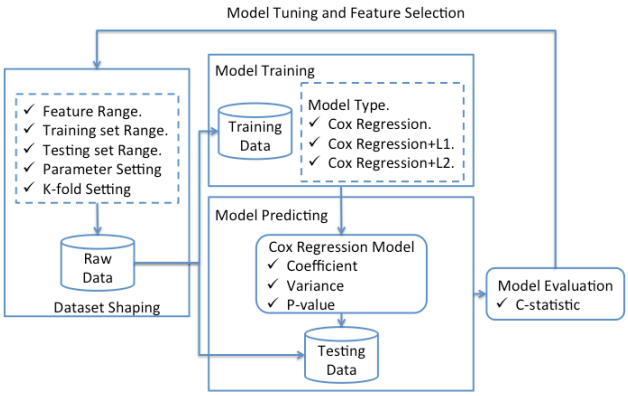

Figure 1. System Flow

In this project, we implement the Cox regression as a statistic package on Spark and we provide several functions that often offered by current statistic tool, such as R and SAS. Figure 1 shows the overflow for model training in our implementation. Model training is an iterative process including several steps. Initially, Dataset Shaping allows user to setup a set of training features, training data, testing data and parameter setting. Next, Dataset Shaping generate the training data and testing data associated with the Cox regression parameter setting on the fly. Then, in the model training process, we offer not only the Cox regression model but also models with L1 and L2 regularization. We also implement the Breslow's Approximation if the input dataset has tied. The output of a Cox Regression Model contains information, such as coefficient, variance and p-value. The model information allows users to evaluate the significance of attributes and to select the training feature set in the next round. The testing data is used to evaluate the model accuracy. K-fold cross validation is also provided in this package. So far, we provide the C-statistic to measure the Cox regression accuracy given testing data. With model information and model evaluation results, users can execute model tuning or feature selection in the next round by setting configurations in the Dataset Shaping.

5. EVALUATION

We evaluate CLG implemented on Spark through a series of experiments. We evaluate runtime on a cluster with 16 machines. Each machine has 8 processors (Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz) and 30GB RAM memory, and runs Linux Ubuntu 14.04. We use Spark 1.3.1 and Apache Hadoop 1.0.4 as framework.

The real dataset is randomly sampled 42,365 patients from total 1,027,339 patients who already diagnosed with Congestive Heart Failure (CHF) since 2003/01/01 to 2013/03/31. We use treatment patterns discovered by sequential pattern mining as patients' attributes.

Table 1. The number of dimensions in dataset

Dataset	200M	400M	2G
Dimensions	2,766	5,184	22,924

5.1 Efficiency Results

We evaluate efficiency of Cox regression implemented iCLG algorithm, and we leverage RDD and the parallel framework provided by Spark to break the limitation of high dimensions. We run experiments on a cluster and a single machine respectively.

5.1.1 Runtime on a cluster

In Table 1, given 3 datasets with the same number of patients, 42,365, we evaluate the absolute runtime of different number of dimensions. We run this experiment on a 16-node cluster, and we list results in Table 2.

From the result, we find that even though the implementation of CLG on Spark breaks the high dimensional problem that cannot be solved in current statistic tool, the runtime performance is not satisfied. The main reason is that the inherent nature of the Cox model coefficient estimation relies risk set at each time point. The risk set can be generated by either using Cartesian patients set or by sorting patient on survival time. Because the Cartesian is expensive even though for only 42,365 patients, we use sorting as an alternative option. Therefore, when we estimate one coefficient in one iteration, we have to sort and to collect RDDs. The cost of sorting and collecting is 0.2 seconds in our environment. Because CLG pick one variable to a value that minimizes the objective function while holding all other variables constant, the number of sort and collections is proportional to the number of dimensions. Compare to logistic regression or linear regression, the current coefficient estimation algorithm for the Cox regression cannot fully utilized the power of distributed computing. So far, we have not found an appropriate algorithm to break dimensions in parallel yet.

Table 2. Duration of 5 iterations for Cox regression using various dataset on a 16-node cluster.

Dataset	Runtime (hh:mm:ss)
200M	1:17:03
400M	2:37:16
2G	15:26:28

Next, We evaluate the runtime by fixing the dataset size, 200M, and we run the Cox model estimation on 2, 4, 8 and 16 working machines. Table 3 shows that the increasing number of nodes only demonstrates the distributed effect at the initial stage. When the number of nodes is larger than 8, there is no significant improvement. The reason is because we did not fully utilize the

distributed power of Spark. The bottleneck at sorting and collecting severely hamper the efficiency. However, we know that the limitation of high dimensional problem can be break in this paradigm, at least the memory limitation on a single machine does not exist anymore.

Table 3. Duration of 5 iterations for Cox regression using various number of nodes on 200M.

Number of Nodes	Runtime (hh:mm:ss)
2	2:00:00
4	1:36:33
8	1:15:23
16	1:17:03

5.1.2 Runtime on a single machine

To compare with current Cox regression package, such as survival package and glmnet package in R, we conduct an experiment on a single machine. The single machine experiment leverage 8 EC2 Compute Units (4 virtual cores with 2 EC2 Compute Units each) with 15GB of main memory, running on Linux 64-bit platform. The input dataset is 42,365×717 because 717 dimensions are almost the computation limit of the Cox regression in R.

Table 4. Comparison of model training duration between Spark Cox, Coxph and Coxnet.

Package	Runtime (min)
Spark_Cox	68.5
R_Coxph	23.43
R_Coxnet	43.12

Table 4 shows the Cox regression training time, including the time to load dataset into the memory. R_Coxph is the Cox regression implemented in the R survival package. R_Coxph has shortest training time because it leverages Newton-Raphson as the optimization approach that has faster converging ability than CLG. R_Coxnet use CLG as their core algorithm and the penalty function they use is elastic net penalty, which can better handle the correlated covariates especially in high-dimensional data. Both packages in R are targeted on a single machine and high dimensions, such as datasets in Table 1 cannot be processed. Our implementation is denoted as Spark_Cox, and we set 10 iterations because typical learning algorithms need tens of iterations to converge. Spark_Cox has longest training time among the three on a single machine. We think it is mainly because of the nature of CLG. If we run this dataset, 42,365×717, with Spark_Cox on the 16-node cluster, the runtime can be reduced to around 25 minutes, which is comparable to the R_Coxph.

5.2 Accuracy Results

In this project, we implement the Harrell's c-statistic [5] as the accuracy metric of Cox regression. Harrell's c-statistic is an extension of AUC (area under the receiver-operator curve) and it is a common approach to measure the discriminative power of various survival models by comparing the estimated and observed ordering of risk scores between pairs of comparable subjects. Two subjects (i, j) are comparable if at least one of the subjects in the pair encounters the event, and the follow-up time duration for that subject is less than that of the other, i.e. $y_i < y_j$ and $\delta_i = 1$. If a pair of comparable subjects (i, j) is concordant only when the estimated risk of the subject who develops the event first is higher than that of the other subject, i.e. $y_i < y_j$, $\delta_i = 1$ and $\beta^T x_i > \beta^T x_j$. Given a test set of n subjects, the c-statistic is

$$c - statistic = \frac{the\ number\ of\ concordant\ pairs}{the\ number\ of\ comparable\ pairs} \quad (12)$$

Using the Cartesian, filter, map and reduce API provided in Spark, the c-statistic can be computed in parallel.

We also implement k-fold cross validation in our package. Through configuration setting, users can run the cross validation associated with different input parameters. For example, we ran an accuracy experiment to compare the c-statistic of three models: the Cox regression model (None), the Cox regression model with L1 regularization (L1) and the Cox regression model with L2 regularization (L2). To place the regularization parameters from the L1 and L2 penalties on the same scale, we re-parameterize via $\gamma = \gamma_i = 2/\sigma^2$ and $\tau = \tau_i = \sigma^2$ respectively, for all coefficient i. We perform 5-fold cross validation on the dataset with size of 42,365×77. In Figure 2, we show how the accuracy change associated with the variation of parameter σ.

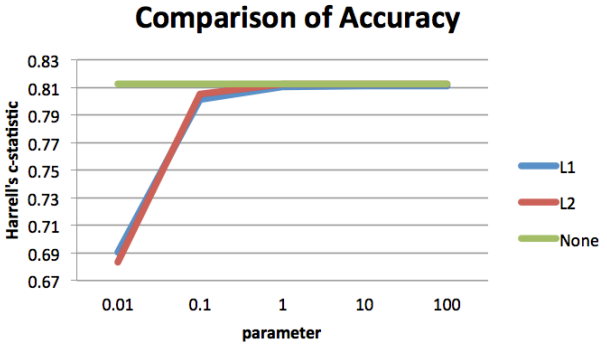

Figure 2. Comparison of Accuracy between L1, L2 and no regularization

6. CONCLUSION

In this project, we implement the Cox regression package on top of Spark framework. We leverage the distributed computation power of Spark and CLG algorithm to break the limitation of high-dimensional massive censored dataset, which cannot be completely solved by current statistical software, such as R and SAS. This implementation is truly scalable both in increasing the number of data points and dimensions because the memory limitation and matrix inversion issue does not exist anymore. In order to form this package as current statistical software, we develop several additional common used functions by statistical users in this package.

We view the model training as an iterative process start from input data shaping, model training, model evaluation and feature selection. Data shaping allows users to determine the training feature sets, k-fold cross validation and parameters by setting configuration file, and Data shaping also preserves ability for extending to a better feature selection component in the future. The model training implements Cox regression coefficient estimation algorithm and regularization. Regularization is for avoiding model over-fitting and feature selection. We also implement Breslow's approximation for tied data. The output model information containing coefficient, variance and p-value that also supported by current statistical software. Further, we provide the c-statistic, specific accuracy metrics for survival models, to measure the discriminative ability of the Cox regression model. With these basic functions and the support from Spark framework, we expect that a scalable Cox regression model can be widely applied.

We address several limitations we encountered in this project and we expect we can break these constraints in the future. First, we implement CLG as our core coefficient estimation algorithm, but CLG cannot be fully distributed, especially if we want to partition dimensions. Therefore, we only utilize partial distributed power in Spark. So far, we have not found any related work to partition dimensions and run the estimation in parallel, especially for the partial maximum likelihood. Second, even though we did not implement the Newton-Raphson algorithm, the Hessian matrix is still required if we need the p-value of coefficients. Thus, how to efficiently calculate p-value is important because p-value is used for judging the significance of an explanatory variable and for feature selection. Third, to preserve the feature selection flexibility, we leverage the SQL API provided in Spark. However, the Spark SQL is still in development and several functions do not work correctly. For example, when we select more than 20,000 features to form the training dataset, Spark is halted. In the future, a scalable survival analysis framework need to be developed so that the high dimensional massive censored dataset can contribute more in medical knowledge discovery.

7. REFERENCES

[1] Cox, D.R. 1972. Regression models and life tables. *Journal of the Royal Statistical Society. Series B:*. 34, 2 (1972), 187–220.

[2] Dean, J. and Ghemawat, S. 2008. MapReduce : Simplified Data Processing on Large Clusters. *Communications of the ACM*. 51, 1 (2008), 1–13.

[3] Fan, J. et al. 2010. High-dimensional variable selection for Cox's proportioinal hazards model. *Institute of Mathematical Statistics Collections*. 6, (2010), 70–86.

[4] Genkin, A. et al. 2007. Large-Scale Bayesian Logistic Regression for Text Categorization. *Technometrics*.

[5] Harrell, F.E. et al. 1996. Multivariable prognostic models: Issues in developing models, evaluating assumptions and adequacy, and measuring and reducing errors. *Statistics in Medicine*. 15, 4 (1996), 361–387.

[6] Isard, M. et al. 2007. Dryad : Distributed Data-Parallel Programs from Sequential Building Blocks. (2007).

[7] Low, Y. et al. 2010. GraphLab: A New Framework for Parallel Machine Learning. *The 26th Conference on Uncertainty in Artificial Intelligence (UAI 2010)*. (2010), 8–11.

[8] Mittal, S. et al. 2014. High-dimensional, massive sample-size Cox proportional hazards regression for survival analysis. *Biostatistics*. 15, 2 (2014), 207–221.

[9] Passing, M. and Forum, I. 2009. MPI : A Message-Passing Interface Standard. *Forum American Bar Association*. 8, UT-CS-94-230 (2009), 647.

[10] Power, R. and Li, J. 2010. Piccolo: Building Fast, Distributed Programs with Partitioned Tables. *Proceedings of the 9th USENIX conference on Operating systems design and implementation - OSDI'10*. 1-14 (2010), 1–14.

[11] Simon, N. et al. 2011. Regularization paths for Cox's proportional hazards model via coordinate descent. *Journal of Statistical Software*. 39, 5 (2011), 1–13.

[12] Zaharia, M. et al. 2012. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. *NSDI'12 Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. (2012), 2–2.

[13] Zhu, L. et al. 2011. Model-Free Feature Screening for Ultrahigh Dimensional Data. *Journal of the American Statistical Association*. 106, (2011), 1464–1475.