Model Fitting with Distributed Data

Balasubramanian Narasimhan

Department of Biomedical Data Sciences
and
Department of Statistics
Stanford University

January 9, 2018

Introduction

Combining data from distributed institutional databases promises many advantages in personalizing medical care.

- Reliable and stable modeling of outcomes
  - Larger N
  - Precision of estimates
  - Power for detecting differences
- Richer feature sets for use in models
- More chance for finding "patients like me"

Introduction...

Registries also essentially perform this centralization.

- Data is aggreated at a central site, typically at periodic intervals
- Generally, data is anonymized for analysis
- Examples: Surveillance, Epidemiology, and End Results (SEER) Program; The Center for International Blood and Marrow Transplant Research (CIBMTR); etc.

The Problem

There are high (and growing) barriers to aggregation of medical data, particularly between centers/researchers.

- Lack of standardization of ontologies
- Privacy concerns
- Reluctance to cede control (once flown...)
- Proprietary attitude towards institution's data

Some efforts have been very costly and yielded few results.

Data Aggregation is not always necessary

- Data can stay at site (so sharing can be turned on/off)
- Computations can be distributed (also can be turned on/off)

Many computations that form the crux of model fitting can be so implemented:

- Maximizing a likelihood. Intermediate computations break up into sums of quantities computed on local data at sites.
- Singular Value Decomposition. Iterative algorithms are available for computing singular values using quantities computed on local data at sites.
- And more.

Indeed, sometimes distribution of the calculation among sites is necessary to share a heavy computational burden.

Some Approaches

- Jiang et. al.(*Bioinformatics* 2013) describe the **WebGLORE**: Web-based Grid LOgistic REgression service that enables privacy-preserving logistic model fit from distributed datasets. Only transfers aggregated local statistics (from participants) through Hypertext Transfer Protocol Secure (HTTPS) to a trusted server, where the global model is constructed.
  URL:
  `http://dbmi-engine.ucsd.edu/webglore3/WebGLORE`
- Wolfson, et. al.(*IJE, 2010*) describe fitting generalized linear models (GLMs) by aggregating anonymous summary-statistics from harmonized individual-level databases (DataSHIELD). The project is part of the OPAL software suite of stand-alone applications that support various study's data management activities.

Open Distributed Computation

Advances in open source software have made open distributed computing very accessible

- R is a widely used platform that has a rich set of libraries for all manner of statistical computation and model fitting. So distributed algorithms can be implemented in R packages in a straighforward manner
- The **opencpu** R package exposes R's functions over a REST Application Programming Interface: R functions can be invoked using a URL.
- The `shiny` R package enables one to provide friendly user interface to users

Assumptions

Our initial assumptions are:

- Transmitting summaries between is ok
- Some degree of trust between sites (via agreements between CIOs etc.)

Note that the master process can make an unlimited number of function calls on the worker sites. Therefore, this needs to be combined with some auditing mechanism.

Row-partitioned data

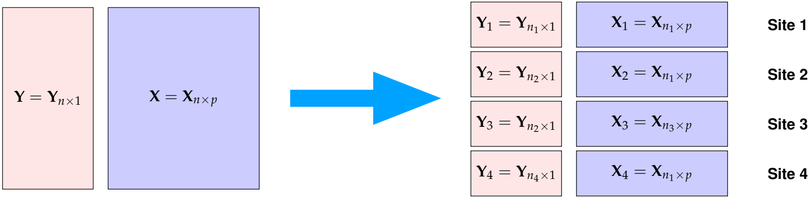
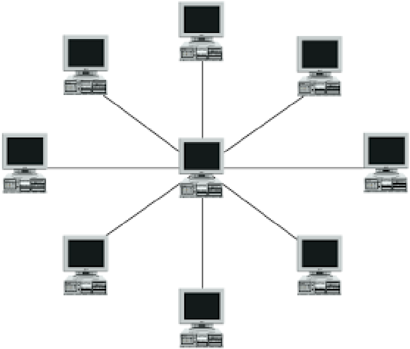

$\mathbf{Y} = \mathbf{Y}_{n \times 1}$
$\mathbf{X} = \mathbf{X}_{n \times p}$
$\mathbf{Y}_1 = \mathbf{Y}_{n_1 \times 1}$
$\mathbf{X}_1 = \mathbf{X}_{n_1 \times p}$
Site 1
$\mathbf{Y}_2 = \mathbf{Y}_{n_2 \times 1}$
$\mathbf{X}_2 = \mathbf{X}_{n_1 \times p}$
Site 2
$\mathbf{Y}_3 = \mathbf{Y}_{n_2 \times 1}$
$\mathbf{X}_3 = \mathbf{X}_{n_3 \times p}$
Site 3
$\mathbf{Y}_4 = \mathbf{Y}_{n_4 \times 1}$
$\mathbf{X}_4 = \mathbf{X}_{n_1 \times p}$
Site 4

Figure: Left: Aggregated data. Right: Distributed Data

Communication Topology

A commonly used topology is the star network.



In a star network, a single node runs a master process that communicates with other nodes in the network that are clients. The final results are available at the master.

Possible Computations

It turns out that many computations can be done in a star network using row-partitioned data.

Generalized Linear Models The overall likelihood, score and information matrices are all sums of the respective quantities over sites. Therefore a master process can perform the iterations for maximization.

Iterative Computations It may be necessary for intermediate results to be stored in the clients (*stateful*) iterative process.

And more.

Site Stratified Cox Model

The Cox PH model assumes a hazard function of the form

$$\boldsymbol{\lambda}_{n\times 1}(t) = \lambda_0(t)\exp(\boldsymbol{X}_{n\times p}\boldsymbol{\beta}_{p\times 1}),$$

Model fitting and inference is accomplished by maximizing a partial likelihood function (see Therneau and Grambsch) of the form:

$$l(\boldsymbol{\beta}|\boldsymbol{X}) = \sum_{i=1}^{n}\int_0^{\infty}\left[Y_i(t)\boldsymbol{X}_i(t)\boldsymbol{\beta} - \log\left(\sum_j Y_j(t)r_j(\boldsymbol{\beta}, t)\right)\right]dN_i(t).$$

In multicenter studies, the *stratified* Cox model is often used where each site is a stratum. This allows for different a baseline hazard for each site, yet a single $\boldsymbol{\beta}$ is fit.

It turns out again that the overall log-likelihood is a sum over the strata. Same for score, information matrix etc. So once again the computation can be distributed.

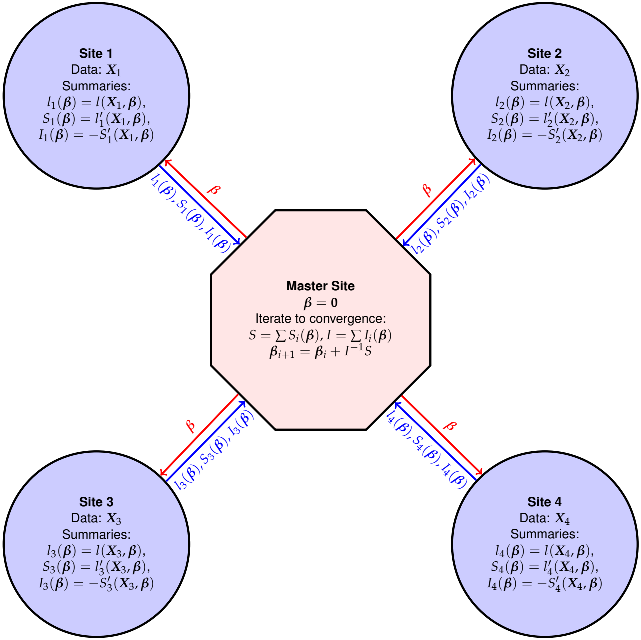
Maximization via Newton-Raphson

K sites, $l_k(\boldsymbol{\beta})$, $S_k(\boldsymbol{\beta})$, $I_k(\boldsymbol{\beta})$ are site-specific likelihood, score and information matrix.

0. Set $i = 0$, $\boldsymbol{\beta}_0 = 0$, a tolerance ϵ and a maximum number of iterations B.
1. Transmit $\boldsymbol{\beta}_i$ to each site
2. Each site k sends back $l_k(\boldsymbol{\beta}_i)$, $S_k(\boldsymbol{\beta}_i)$ and $I(\boldsymbol{\beta}_i)$
3. Compute $l(\boldsymbol{\beta}_i) = \sum_{k=1}^{K} l(\boldsymbol{\beta}_{)}$, $S(\boldsymbol{\beta}_i) = \sum_{k=1}^{K} S_k(\boldsymbol{\beta}_i)$, $I(\boldsymbol{\beta}_i) = \sum_{k=1}^{K} I_k(\boldsymbol{\beta}_i)$,
4. Set
$$\boldsymbol{\beta}_{i+1} = \boldsymbol{\beta}_i + I^{-1}(\boldsymbol{\beta}_i)S(\boldsymbol{\beta}_i)$$
5. Stop if converged or iteration count exceeded. Else increment i and repeat step 1.

For the Cox Model, the convergence is very fast.

Schematic



Site 1
Data: X_1
Summaries:
$l_1(\boldsymbol{\beta}) = l(\boldsymbol{X}_1, \boldsymbol{\beta})$,
$S_1(\boldsymbol{\beta}) = l_1'(\boldsymbol{X}_1, \boldsymbol{\beta})$,
$I_1(\boldsymbol{\beta}) = -S_1'(\boldsymbol{X}_1, \boldsymbol{\beta})$

Site 2
Data: X_2
Summaries:
$l_2(\boldsymbol{\beta}) = l(\boldsymbol{X}_2, \boldsymbol{\beta})$,
$S_2(\boldsymbol{\beta}) = l_2'(\boldsymbol{X}_2, \boldsymbol{\beta})$,
$I_2(\boldsymbol{\beta}) = -S_2'(\boldsymbol{X}_2, \boldsymbol{\beta})$

Master Site
$\boldsymbol{\beta} = \boldsymbol{0}$
Iterate to convergence:
$S = \sum S_i(\boldsymbol{\beta}), I = \sum I_i(\boldsymbol{\beta})$
$\boldsymbol{\beta}_{i+1} = \boldsymbol{\beta}_i + I^{-1}S$

Site 3
Data: X_3
Summaries:
$l_3(\boldsymbol{\beta}) = l(\boldsymbol{X}_3, \boldsymbol{\beta})$,
$S_3(\boldsymbol{\beta}) = l_3'(\boldsymbol{X}_3, \boldsymbol{\beta})$,
$I_3(\boldsymbol{\beta}) = -S_3'(\boldsymbol{X}_3, \boldsymbol{\beta})$

Site 4
Data: X_4
Summaries:
$l_4(\boldsymbol{\beta}) = l(\boldsymbol{X}_4, \boldsymbol{\beta})$,
$S_4(\boldsymbol{\beta}) = l_4'(\boldsymbol{X}_4, \boldsymbol{\beta})$,
$I_4(\boldsymbol{\beta}) = -S_4'(\boldsymbol{X}_4, \boldsymbol{\beta})$

$\boldsymbol{\beta}$
$l_1(\boldsymbol{\beta}), S_1(\boldsymbol{\beta}), I_1(\boldsymbol{\beta})$
$\boldsymbol{\beta}$
$l_2(\boldsymbol{\beta}), S_2(\boldsymbol{\beta}), I_2(\boldsymbol{\beta})$
$\boldsymbol{\beta}$
$l_3(\boldsymbol{\beta}), S_3(\boldsymbol{\beta}), I_3(\boldsymbol{\beta})$
$\boldsymbol{\beta}$
$l_4(\boldsymbol{\beta}), S_4(\boldsymbol{\beta}), I_4(\boldsymbol{\beta})$

Singular Value Decomposition

The SVD of $\boldsymbol{X}_{n\times p}$, is $\boldsymbol{U}$, $\boldsymbol{V}$, $\boldsymbol{D}$ such that

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^\top,\ \boldsymbol{U}^\top\boldsymbol{U} = I,\ \boldsymbol{V}^\top\boldsymbol{V} = I,\ \text{and } \boldsymbol{D} \text{ is diagonal.}$$

- Decomposes the variance of $\boldsymbol{X}$ into what are called principal components.
- v_1, the first column of $\boldsymbol{V}$, the first principal component of X maximizes of $\mathrm{var}(\boldsymbol{X}v)$.
- u_1 indicates how much of factor v_1 is present in each observation.
- $d_1^2/\sum_j d_j^2$ is the proportion of the variance of $\boldsymbol{X}$ that can be explained by v_1.
- The first k vectors can used to get a k approximation to $\boldsymbol{X}$.

Efficient Implementations in LAPACK, which much software builds upon.

There is a well-known power method for computing a singular vector corresponding to the largest eigenvalue.

```
**Data:** $\boldsymbol{X} \in \mathcal{R}^{n \times p}$
**Result:** $u \in \mathcal{R}^n,\ v \in \mathcal{R}^p$, and $d > 0$
$u \leftarrow (\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}, \ldots, \frac{1}{\sqrt{n}})$;
**repeat**
    $v \leftarrow \boldsymbol{X}^\top u$;
    $v \leftarrow v / \|v\|$;
    $u \leftarrow \boldsymbol{X} v$;
    $d \leftarrow \|u\|$;
    $u \leftarrow u / \|u\|$;
**until** *convergence*;
```

Note that the operations involve inner products and sums and therefore distribute over sites.

Singular vectors can be found successively by removing the effect of the top singular vector and then finding the rank $k - 1$ approximation again.

Privacy-preserving rank-k SVD

```
Data: each site has private data $\boldsymbol{X}_j \in \mathcal{R}^{n_j \times p}$
Result: $V \in \mathcal{R}^{p \times k}$, and $d_1 \geq \ldots d_k \geq 0$
$V \leftarrow 0,\ \ d \leftarrow 0$ foreach site j do
    $U^{[j]} = 0$;
    transmit $n_j$ to master;
end
for $i \leftarrow 1$ to $k$ do
    foreach site j do $u^{[j]} \leftarrow (1, 1, \ldots, 1)$ of length $n_j$;
    $\|u\| \leftarrow \sqrt{\sum_j n_j}$;
    transmit $\|u\|$, $V$, and $D$ to sites;
    repeat
        foreach site j do
            $u^{[j]} \leftarrow u^{[j]} / \|u\|$;
            calculate $v^{[j]} \leftarrow (\boldsymbol{X}^{[j]} - U^{[j]} D V^\top)^\top u^{[j]}$;
            transmit $v^{[j]}$ to master;
        end
        $v \leftarrow \sum_j v^{[j]}$; $v \leftarrow v / \|v\|$;
        transmit $v$ to sites;
        foreach site j do
            calculate $u^{[j]} \leftarrow \boldsymbol{X}^{[j]} v$;
            transmit $\|u^{[j]}\|$ to master;
        end
        $\|u\| \leftarrow \sum_j \|u^{[j]}\|$;
        transmit $\|u\|$ to sites;
        $d_i \leftarrow \|u\|$;
    until convergence;
    $V \leftarrow \texttt{cbind}(V, v)$;
    foreach site j do $U^{[j]} \leftarrow \texttt{cbind}(U^{[j]}, u^{[j]})$;
end
```

The Tools

To enable such computations (and others that might be developed) in a distributed way, one needs

- Readily available computing power (a unix server box).
- An extensible, open source environment (R) to implement such tools and algorithms (our package **distcomp**)
- Tools to propose, define and refine computation tasks (R package **shiny**, R package **opencpu**)
- Secure means of exposing computation to site—we've all become HIPAA-etrified! (SSL)
- Auditing mechanisms to provide reports to satisfy center CIO so that there is a trail (Logs/Dashboards)

There are several social aspects of the collaboration need to be engineered.

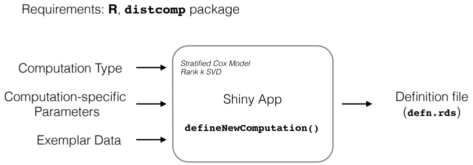
Workflow

The main steps are the following.

1. Define the Computation
2. Set up a Worker Process for the Computation
3. Set up a Master Process for the Computation
4. Run the Computation

We address each in turn.

1. Defining the Computation



Requirements: **R**, **distcomp** package
Computation Type
Computation-specific Parameters
Exemplar Data
Stratified Cox Model
Rank k SVD
Shiny App
defineNewComputation()
Definition file (**defn.rds**)

A computation may be defined on any machine where R and the **distcomp** package are installed.

The function `defineNewComputation()` launches a **shiny** app that leads the user through the process.

The end result is an R data file unambiguously defining the computation instance for **distcomp**.

Example: UIS dataset

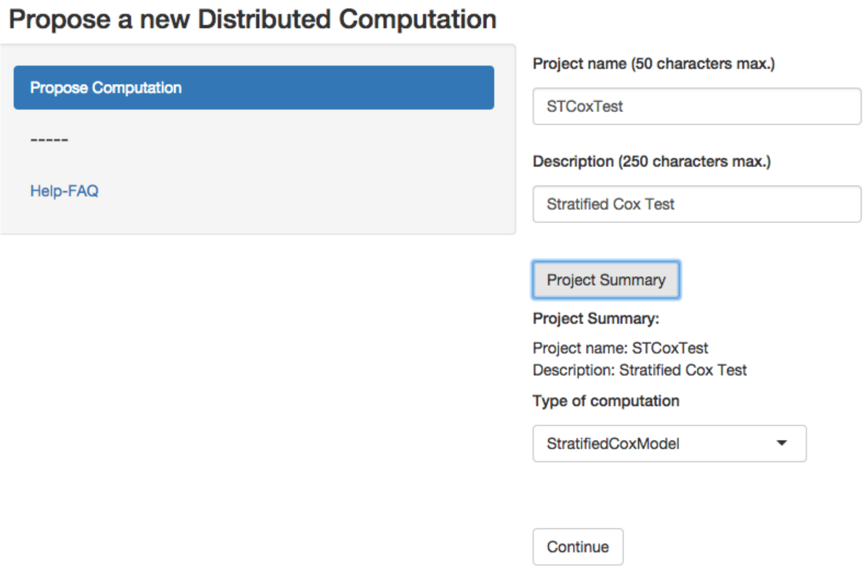
Hosmer and Lemeshow data on time until return to drug use for patients enrolled in two different residential treatment programs. Aggregated fit is:

```
> uis <- readRDS("uis.RDS")
> coxOrig <- coxph(formula = Surv(time, censor) ~ age + becktota +
+                       ndrugfp1 + ndrugfp2 + ivhx3 +
+                       race + treat + strata(site), data = uis)
> summary(coxOrig)
Call:
coxph(formula = Surv(time, censor) ~ age + becktota + ndrugfp1 +
ndrugfp2 + ivhx3 + race + treat + strata(site), data = uis)

n= 575, number of events= 464
(53 observations deleted due to missingness)

               coef exp(coef)  se(coef)      z Pr(>|z|)
age       -0.028076  0.972315  0.008131 -3.453 0.000554 ***
becktota   0.009146  1.009187  0.004991  1.832 0.066914 .
ndrugfp1  -0.521973  0.593349  0.124424 -4.195 2.73e-05 ***
ndrugfp2  -0.194178  0.823512  0.048252 -4.024 5.72e-05 ***
ivhx3TRUE  0.263634  1.301652  0.108243  2.436 0.014868 *
race      -0.240021  0.786611  0.115632 -2.076 0.037920 *
treat     -0.212616  0.808466  0.093747 -2.268 0.023331 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
...
```

The Distributed Analog

Assuming that the UIS dataset is row-partitioned as noted earlier into the two sites, we show how one would perform the distributed model fit in the next few screenshots.

1.1 Screenshot



Propose a new Distributed Computation

Project name (50 characters max.)

STCoxTest

Description (250 characters max.)

Stratified Cox Test

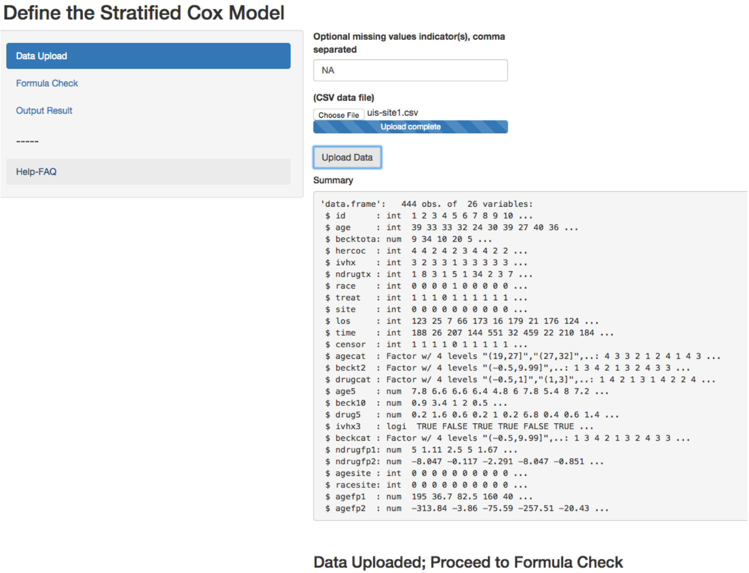
Project Summary:

Project name: STCoxTest
Description: Stratified Cox Test

Type of computation

StratifiedCoxModel

1.2 Screenshot



Define the Stratified Cox Model

Optional missing values indicator(s), comma separated

NA

(CSV data file)

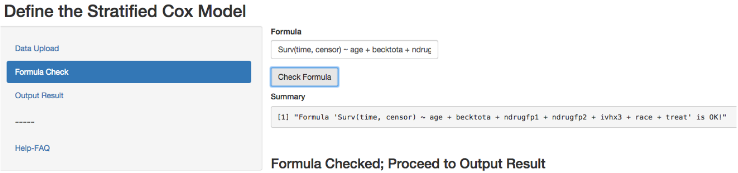
uis-site1.csv

Summary

```
'data.frame':   444 obs. of  26 variables:
 $ id      : int  1 2 3 4 5 6 7 8 9 10 ...
 $ age     : int  39 33 33 32 24 30 39 27 40 36 ...
 $ becktota: num  9 34 10 20 5 ...
 $ hercoc  : int  4 4 2 4 2 3 4 4 2 2 ...
 $ ivhx    : int  3 2 3 3 1 3 3 3 3 3 ...
 $ ndrugtx : int  1 8 3 1 5 1 34 2 3 7 ...
 $ race    : int  0 0 0 0 1 0 0 0 0 0 ...
 $ treat   : int  1 1 1 0 1 1 1 1 1 1 ...
 $ site    : int  0 0 0 0 0 0 0 0 0 0 ...
 $ los     : int  123 25 7 66 173 16 179 21 176 124 ...
 $ time    : int  188 26 207 144 551 32 459 22 210 184 ...
 $ censor  : int  1 1 1 1 0 1 1 1 1 1 ...
 $ agecat  : Factor w/ 4 levels "(19,27]","(27,32]",..: 4 3 3 2 1 2 4 1 4 3 ...
 $ beckt2  : Factor w/ 4 levels "(-0.5,9.99]",..: 1 3 4 2 1 3 2 4 3 3 ...
 $ drugcat : Factor w/ 4 levels "(-0.5,1]","(1,3]",..: 1 4 2 1 3 1 4 2 2 4 ...
 $ age5    : num  7.8 6.6 6.6 6.4 4.8 6 7.8 5.4 8 7.2 ...
 $ beck10  : num  0.9 3.4 1 2 0.5 ...
 $ drug5   : num  0.2 1.6 0.6 0.2 1 0.2 6.8 0.4 0.6 1.4 ...
 $ ivhx3   : logi  TRUE FALSE TRUE TRUE FALSE TRUE ...
 $ beckcat : Factor w/ 4 levels "(-0.5,9.99]",..: 1 3 4 2 1 3 2 4 3 3 ...
 $ ndrugfp1: num  5 1.11 2.5 5 1.67 ...
 $ ndrugfp2: num  -8.047 -0.117 -2.291 -8.047 -0.851 ...
 $ agesite : int  0 0 0 0 0 0 0 0 0 0 ...
 $ racesite: int  0 0 0 0 0 0 0 0 0 0 ...
 $ agefp1  : num  195 36.7 82.5 160 40 ...
 $ agefp2  : num  -313.84 -3.86 -75.59 -257.51 -20.43 ...
```

Data Uploaded; Proceed to Formula Check

1.3 Screenshot



Define the Stratified Cox Model

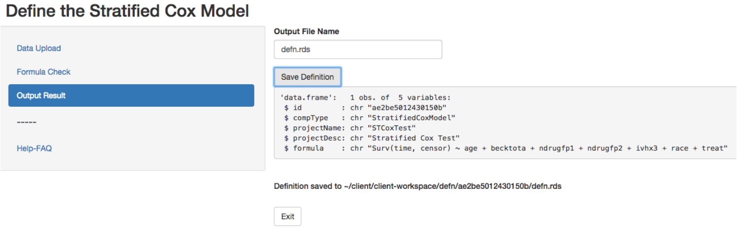
Formula

Surv(time, censor) ~ age + becktota + ndruç

Summary

```
[1] "Formula 'Surv(time, censor) ~ age + becktota + ndrugfp1 + ndrugfp2 + ivhx3 + race + treat' is OK!"
```

Formula Checked; Proceed to Output Result



Define the Stratified Cox Model

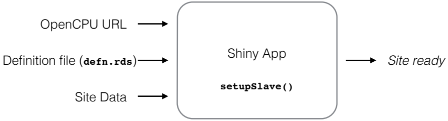
Output File Name

defn.rds

```
'data.frame':	1 obs. of  5 variables:
 $ id         : chr "ae2be5012430150b"
 $ compType   : chr "StratifiedCoxModel"
 $ projectName: chr "STCoxTest"
 $ projectDesc: chr "Stratified Cox Test"
 $ formula    : chr "Surv(time, censor) ~ age + becktota + ndrugfp1 + ndrugfp2 + ivhx3 + race + treat"
```

Definition saved to ~/client/client-workspace/defn/ae2be5012430150b/defn.rds

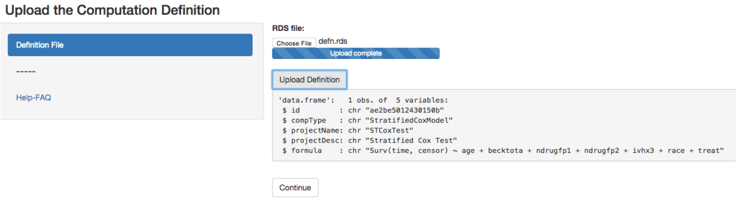
2. Setting up Worker



- Requires a one-time configuration of an **opencpu** server
- R package **distcomp** and a writable workspace
- All interaction is through a **shiny** app that configures the worker for the computation

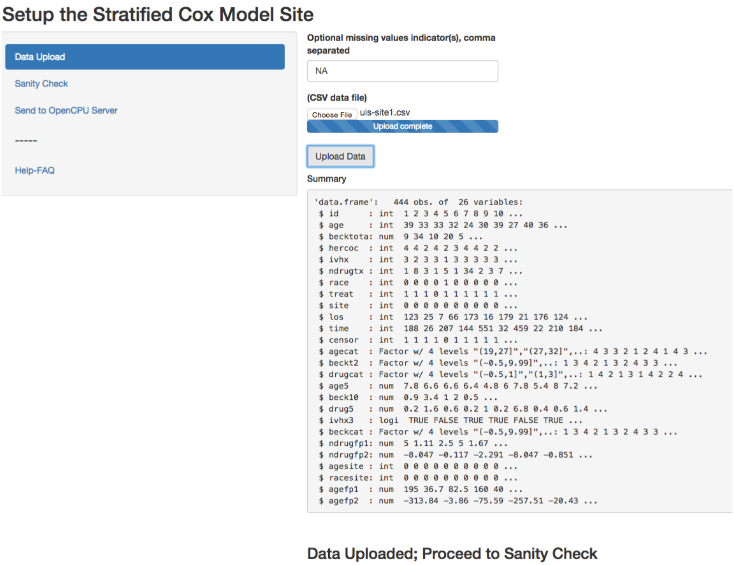
Repeat for each site.

2.1 Screenshot



```
'data.frame':	1 obs. of  5 variables:
 $ id         : chr "ae2be5012430150b"
 $ compType   : chr "StratifiedCoxModel"
 $ projectName: chr "STCoxTest"
 $ projectDesc: chr "Stratified Cox Test"
 $ formula    : chr "Surv(time, censor) ~ age + becktota + ndrugfp1 + ndrugfp2 + ivhx3 + race + treat"
```

2.2 Screenshot



Setup the Stratified Cox Model Site

Optional missing values indicator(s), comma separated

NA

(CSV data file)

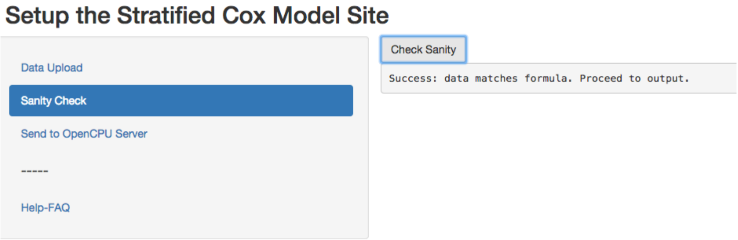
uis-site1.csv

Summary

```
'data.frame':   444 obs. of  26 variables:
 $ id      : int  1 2 3 4 5 6 7 8 9 10 ...
 $ age     : int  39 33 33 32 24 30 39 27 40 36 ...
 $ becktota: num  9 34 10 20 5 ...
 $ hercoc  : int  4 4 2 4 2 3 4 4 2 2 ...
 $ ivhx    : int  3 2 3 3 1 3 3 3 3 3 ...
 $ ndrugtx : int  1 8 3 1 5 1 34 2 3 7 ...
 $ race    : int  0 0 0 0 1 0 0 0 0 0 ...
 $ treat   : int  1 1 1 0 1 1 1 1 1 1 ...
 $ site    : int  0 0 0 0 0 0 0 0 0 0 ...
 $ los     : int  123 25 7 66 173 16 179 21 176 124 ...
 $ time    : int  188 26 207 144 551 32 459 22 210 184 ...
 $ censor  : int  1 1 1 1 0 1 1 1 1 1 ...
 $ agecat  : Factor w/ 4 levels "(19,27]","(27,32]",..: 4 3 3 2 1 2 4 1 4 3 ...
 $ beckt2  : Factor w/ 4 levels "(-0.5,9.99]",..: 1 3 4 2 1 3 2 4 3 3 ...
 $ drugcat : Factor w/ 4 levels "(-0.5,1]","(1,3]",..: 1 4 2 1 3 1 4 2 2 4 ...
 $ age5    : num  7.8 6.6 6.6 6.4 4.8 6 7.8 5.4 8 7.2 ...
 $ beck10  : num  0.9 3.4 1 2 0.5 ...
 $ drug5   : num  0.2 1.6 0.6 0.2 1 0.2 6.8 0.4 0.6 1.4 ...
 $ ivhx3   : logi  TRUE FALSE TRUE TRUE FALSE TRUE ...
 $ beckcat : Factor w/ 4 levels "(-0.5,9.99]",..: 1 3 4 2 1 3 2 4 3 3 ...
 $ ndrugfp1: num  5 1.11 2.5 5 1.67 ...
 $ ndrugfp2: num  -8.047 -0.117 -2.291 -8.047 -0.851 ...
 $ agesite : int  0 0 0 0 0 0 0 0 0 0 ...
 $ racesite: int  0 0 0 0 0 0 0 0 0 0 ...
 $ agefp1  : num  195 36.7 82.5 160 40 ...
 $ agefp2  : num  -313.84 -3.86 -75.59 -257.51 -20.43 ...
```

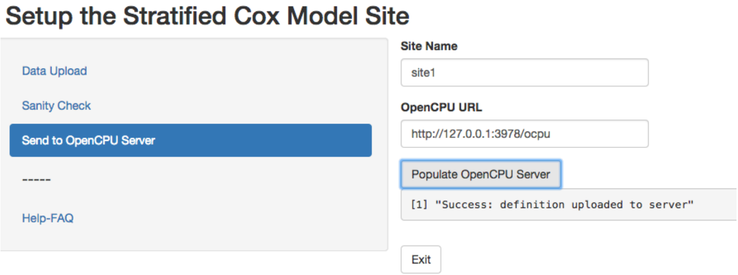
Data Uploaded; Proceed to Sanity Check

2.3 Screenshot



Setup the Stratified Cox Model Site

```
Success: data matches formula. Proceed to output.
```



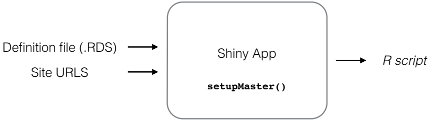
Setup the Stratified Cox Model Site

Site Name: site1

OpenCPU URL: http://127.0.0.1:3978/ocpu

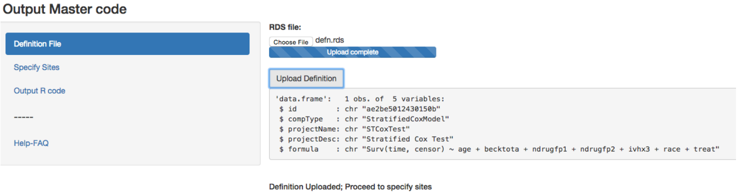
```
[1] "Success: definition uploaded to server"
```

3. Setting up Master



- Specification of computation definition
- Specification of Worker URLs
- Writes out R code that can be executed

3.1 Screenshot

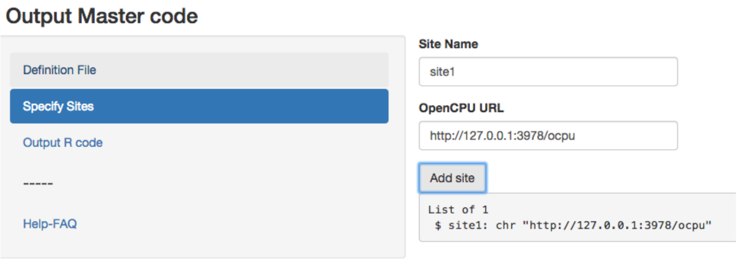


defn.rds

```
'data.frame':   1 obs. of  5 variables:
 $ id         : chr "ae2be5012430150b"
 $ compType   : chr "StratifiedCoxModel"
 $ projectName: chr "STCoxTest"
 $ projectDesc: chr "Stratified Cox Test"
 $ formula    : chr "Surv(time, censor) ~ age + becktota + ndrugfp1 + ndrugfp2 + ivhx3 + race + treat"
```

Definition Uploaded; Proceed to specify sites

3.2 Screenshot



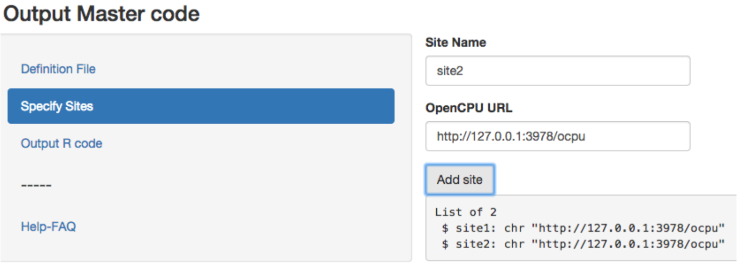
Output Master code

Site Name: site1

OpenCPU URL: http://127.0.0.1:3978/ocpu

```
List of 1
 $ site1: chr "http://127.0.0.1:3978/ocpu"
```

3.3 Screenshot



Output Master code

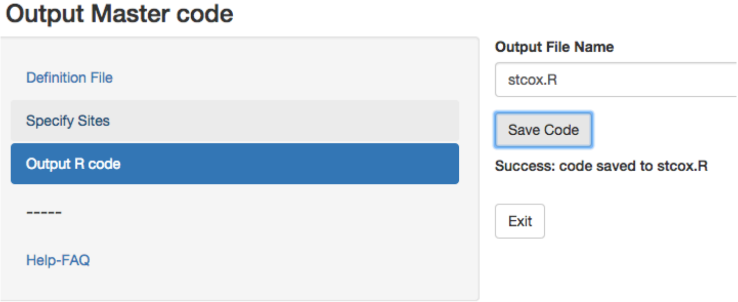
Site Name

site2

OpenCPU URL

http://127.0.0.1:3978/ocpu

```
List of 2
 $ site1: chr "http://127.0.0.1:3978/ocpu"
 $ site2: chr "http://127.0.0.1:3978/ocpu"
```



Output Master code

Output File Name

stcox.R

Success: code saved to stcox.R

The Distributed Cox Fit

The code produced in the master setup is:

```
library(distcomp)
defn <- structure(list(id = "ae2be5012430150b", compType = "StratifiedCoxModel",
    projectName = "STCoxTest", projectDesc = "Stratified Cox Test",
    formula = "Surv(time, censor) ~ age + becktota + ndrugfp1 + ndrugfp2 + ivhx3 + race + treat"),
.Names = c("id",
"compType", "projectName", "projectDesc", "formula"), row.names = c(NA,
-1L), class = "data.frame")
sites <-
structure(c("http://127.0.0.1:3978/ocpu", "http://127.0.0.1:3978/ocpu"
), .Names = c("site1", "site2"))
siteDataFiles <-
c("site1.rds", "site2.rds")
siteNames <-
c("site1", "site2")
master <- coxMaster$new(defnId = defn$id, formula=defn$formula, localServer=TRUE)
for (i in seq.int(length(sites))) {
   master$addSite(siteNames[i], sites[i], dataFileName=siteDataFiles[i])
}
result <- master$run()
print(master$summary())
```

The Cox Output

If you run that program, after a while, it spits out the following:

```
        coef exp(coef)  se(coef)       z          p
1 -0.0280495   0.97234 0.0081301 -3.4501 5.6041e-04
2  0.0091441   1.00919 0.0049918  1.8318 6.6979e-02
3 -0.5219296   0.59337 0.1244240 -4.1948 2.7315e-05
4 -0.1941709   0.82352 0.0482507 -4.0242 5.7168e-05
5  0.2636376   1.30166 0.1082448  2.4356 1.4868e-02
6 -0.2400609   0.78658 0.1156319 -2.0761 3.7887e-02
7 -0.2125720   0.80850 0.0937466 -2.2675 2.3359e-02
```

As can be seen, the results are similar to the original model fit. We have successfully tested examples and deployed `distcomp` at real sites for fitting survival models with breast cancer registries. Stanford, Vanderbilt, Mt. Sinai and Oxford with breast cancer data. We also have collaborations between Stanford and Palo Alto Medical Foundation underway.

Example: SVD Computation

Simulated dataset on three sites, 20 × 5 matrix at each site. Aggregated SVD is:

```
> set.seed(12345)
> x <- matrix(rnorm(100), nrow = 20))
> svd(x)$d
[1] 9.707537 8.199827 7.982888 7.257286 6.235182
> svd(x)$v
            [,1]        [,2]        [,3]        [,4]        [,5]
[1,] -0.17946375  0.08268613 -0.01644895 -0.98010572 -0.00883063
[2,] -0.78963831  0.34694371  0.34328503  0.16509457  0.33316749
[3,]  0.21305901  0.91839439 -0.25083926  0.04461477 -0.21505068
[4,]  0.54504905  0.16843629  0.53318714 -0.10009622  0.61663844
[5,] -0.04232602 -0.03120945 -0.73121540  0.01126215  0.68002329
```

The Distributed SVD Fit

The code produced in the master setup is:

```
library(distcomp)
defn <-
structure(list(id = structure(1L, .Label = "db19ec158c9d5218", class = "factor"),
    compType = structure(1L, .Label = "RankKSVD", class = "factor"),
    projectName = structure(1L, .Label = "SVDTest", class = "factor"),
    projectDesc = structure(1L, .Label = "SVD Test Example", class = "factor"),
    rank = 2L, ncol = 5L), .Names = c("id", "compType", "projectName",
"projectDesc", "rank", "ncol"), row.names = c(NA, -1L), class = "data.frame")
sites <-
structure(c("http://127.0.0.1:3978/ocpu", "http://127.0.0.1:3978/ocpu",
"http://127.0.0.1:3978/ocpu"), .Names = c("site1", "site2", "site3"
))
siteDataFiles <-
c("site1.rds", "site2.rds", "site3.rds")
siteNames <-
c("site1", "site2", "site3")
master <- svdMaster$new(defnId = defn$id, localServer=TRUE)
for (i in seq.int(length(sites))) {
   master$addSite(siteNames[i], sites[i], dataFileName=siteDataFiles[i])
}
result <- master$run(k=defn$rank)
print(result)
```

The SVD Output

If you run that program, after a while, it spits out the following:

```
$v
           [,1]        [,2]
[1,]  0.17947030  0.08275684
[2,]  0.78969198  0.34634459
[3,] -0.21294972  0.91875219
[4,] -0.54501407  0.16784298
[5,]  0.04229739 -0.03032954

$d
[1] 9.707451 8.200043
```

If you actually ask for $k = 5$, it gives:

```
> result$d
[1] 9.707451 8.200043 7.982650 7.257355 6.235351

> result$v
           [,1]        [,2]       [,3]        [,4]         [,5]
[1,]  0.17947030  0.08275684  0.0165604  0.98008722 -0.008933396
[2,]  0.78969198  0.34634459 -0.3437723 -0.16504730  0.333181988
[3,] -0.21294972  0.91875219  0.2496210 -0.04479619 -0.214978886
[4,] -0.54501407  0.16784298 -0.5334277  0.10025749  0.616612820
[5,]  0.04229739 -0.03032954  0.7312254 -0.01140918  0.680060781
```

Data in Databases

The examples shown here use CSV files for demonstration. But the package can use data in databases as well, for example Redcap. This approach can be replicated for any database, where instead of uploading a CSV file, one would specify database parameters.

- During setup, credentials need to be supplied
- Data can be directly updated in the databases any time
- Some additional checks and balances are needed

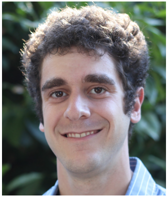
The approach can be replicated for any database the only difference being database URL and credentials will be needed during setup.

Some References

- *Software for Distributed Computation on Medical Databases: A Demonstration Project*, Journal of Statistical Software, Vol. 77 (2017).
- CRAN package `distcomp`

Collaborators and Acknowledgements

Marina Bendersky



Samuel Gross

Philip Lavori

Daniel Rubin

Research supported by Stanford Cancer Institute, NIH, NSF (Nos. P30 CA124435-01, UI1 RR025744, LM-07033)