bigSurvSGD: Big Survival Data Analysis via Stochastic Gradient Descent

Aliasghar (Arash) Tarkhan and Noah Simon

2020-07-22

Introduction

We give a short tutorial on using bigSurvSGD package. This package fits Cox Model via stochastic gradient descent (SGD). This implementation avoids computational instability of the standard Cox Model when datasets are large. Furthermore, it scales up with very large datasets that do not fit the memory. It also handles large sparse datasets using the Proximal stochastic gradient descent algorithm.

Installing and loading package

Intallating package

Like many other R packages, the simplest way to obtain `bigSurvSGD` is to install it directly from CRAN:

```r
install.packages("bigSurvSGD", repos = "http://cran.us.r-project.org")
#> Installing package into '/home/arash/R/x86_64-pc-linux-gnu-library/3.6'
#> (as 'lib' is unspecified)
#> Warning: package 'bigSurvSGD' is not available (for R version 3.6.1)
```

Users may change the `repos` options depending on their locations and preferences.

Loading package

We load the packages `bigSurvSGD` and `survival` as folowing

```r
library(bigSurvSGD)
#> Loading required package: foreach
#> Loading required package: parallel
library(survival)
```

Fitting Cox model

Loading data

We first load dataset `survData` included in package:

```r
data("survData")
```

Now we use `bigSurvSGD` to estimate coefficients as following

```r
fitBigSurvSGD <- bigSurvSGD(formula=Surv(time, status)~.,data=survData,
                            parallel.flag=TRUE, num.cores = 2)
fitBigSurvSGD
```

```
#> $coef
#>      estimate  lower.CI  upper.CI   t-stat p-value
#> X1  1.0014332 0.9913426 1.0115237 194.7520       0
#> X2  1.0478321 1.0368870 1.0587772 187.8648       0
#> X3  0.9201694 0.9104090 0.9299298 185.0015       0
#> X4  1.0617439 1.0509978 1.0724900 193.8846       0
#> X5  0.9547198 0.9429559 0.9664836 159.2575       0
#> X6  1.0316996 1.0208529 1.0425462 186.6519       0
#> X7  1.0201680 1.0067283 1.0336076 148.9561       0
#> X8  0.9969134 0.9868500 1.0069769 194.3946       0
#> X9  0.9224735 0.9123849 0.9325621 179.4305       0
#> X10 0.9160031 0.9059468 0.9260593 178.7453       0
#>
#> $coef.exp
#>     estimate lower.CI upper.CI   t-stat p-value
#> X1  2.722180 2.694850 2.749788 194.7520       0
#> X2  2.851463 2.820423 2.882844 187.8648       0
#> X3  2.509716 2.485339 2.534331 185.0015       0
#> X4  2.891409 2.860504 2.922648 193.8846       0
#> X5  2.597942 2.567560 2.628685 159.2575       0
#> X6  2.805830 2.775561 2.836430 186.6519       0
#> X7  2.773661 2.736633 2.811189 148.9561       0
#> X8  2.709905 2.682770 2.737313 194.3946       0
#> X9  2.515505 2.490254 2.541011 179.4305       0
#> X10 2.499281 2.474274 2.524541 178.7453       0
#>
#> $lambda
#> [1] 0
#>
#> $alpha
#> [1] 0
#>
#> $features.mean
#>          X1          X2          X3          X4          X5          X6
#> -0.02081123 -0.03844643 -0.01947003  0.02845033 -0.01993854  0.01782694
#>          X7          X8          X9         X10
#> -0.01202580  0.04699838 -0.00599775  0.01309362
#>
#> $features.sd
#>        X1        X2        X3        X4        X5        X6        X7        X8
#> 1.0170818 1.0161579 1.0041936 0.9956090 1.0160725 0.9824744 0.9993097 1.0000475
#>        X9       X10
#> 0.9905759 1.0180216
```

For comparison, we can use standard Cox Model `coxph` to estimate coefficients as following

```
fitCox <- coxph(formula=Surv(time, status)~.,data=survData)
fitCox
#> Call:
#> coxph(formula = Surv(time, status) ~ ., data = survData)
#>
#>        coef exp(coef) se(coef)     z      p
#> X1  0.98008   2.66467  0.04374 22.41 <2e-16
#> X2  1.01072   2.74757  0.04430 22.82 <2e-16
#> X3  0.90612   2.47471  0.04297 21.09 <2e-16
#> X4  1.01932   2.77131  0.04460 22.85 <2e-16
#> X5  0.95135   2.58920  0.04401 21.62 <2e-16
#> X6  1.00906   2.74303  0.04490 22.48 <2e-16
#> X7  0.98892   2.68834  0.04421 22.37 <2e-16
#> X8  0.97359   2.64743  0.04449 21.88 <2e-16
#> X9  0.91224   2.48990  0.04505 20.25 <2e-16
#> X10 0.90848   2.48054  0.04359 20.84 <2e-16
#>
#> Likelihood ratio test=1748  on 10 df, p=< 2.2e-16
#> n= 1000, number of events= 823
```

Users need to use `formula` to specify `time-to-event` and `status` variables if they were named deifferently from `time` and `status`. User may also need to specify a subset of features they need to include in the model. For example, suppose that variable `t` and `s` represent `time-to-event` and `status` variables, and a user wants to only include features $X1$, $X2$, and $X3$:

```
colnames(survData)[c(1,2)] <- c("t", "s")
fitBigSurvSGD <- bigSurvSGD(formula=Surv(time=t, status=s)~X1+X2+X3, data=survData,
                            parallel.flag=TRUE, num.cores=2)
```

For comparison, we can estimate coeffcents using standard Cox Model `coxph`:

```
fitCox <- coxph(formula=Surv(t, s)~X1+X2+X3, data=survData)
fitCox
#> Call:
#> coxph(formula = Surv(t, s) ~ X1 + X2 + X3, data = survData)
#>
#>       coef exp(coef) se(coef)      z      p
#> X1 0.37111   1.44934  0.03628 10.229 <2e-16
#> X2 0.32047   1.37778  0.03542  9.049 <2e-16
#> X3 0.34446   1.41123  0.03655  9.424 <2e-16
#>
#> Likelihood ratio test=252.2  on 3 df, p=< 2.2e-16
#> n= 1000, number of events= 823
```

Note that the current version only supports numerical variables. Users need to convert non-numerical valriables into numerical variables beforehand.

Fitting large dataset of the hard drive

Package `bigSurvSGD` can handle large datasets that do not fit the memory. For an example, suppose that dataset `bigSurvData` is very big and we saved it with path `/home/arsh/bigSurvData.csv`:

```
data("survData")
write.csv(survData, file = "/home/arash/bigSurvData.csv", row.names = F)
```

Suppose that memory can only handle up to 100 rows of `bigSurdData`. In practice, this number is a maximum number of rows for which R can run without "lack of memory" error. This number would be big if there few features (columns) in dataset and it wold be small if there are many featurs. Note that number of rows per chunk must be at least equal to strata size specified as `strata.size` in this package. For our example, we ask `bigSurvSGD` to use `bigmemory` package (by defining `bigmemory.flag=T`) to read data chunk-by-chunk off the memory with chunk size of 100.

```
fitBigSurvSGD <- bigSurvSGD(formula=Surv(time=time, status=status)~.,
                            data="/home/arash/bigSurvData.csv",
                              bigmemory.flag = T, num.rows.chunk = 100)
#> Warning in read.big.matrix(filename = data, sep = ",", skip = 0, header = T):
#> Because type was not specified, we chose double based on the first line of data.
fitBigSurvSGD
#> $coef
#>      estimate  lower.CI  upper.CI   t-stat p-value
#> X1  1.0012419 0.9912880 1.0111959 197.3867       0
#> X2  1.0452106 1.0329031 1.0575181 166.6510       0
#> X3  0.9154692 0.9039534 0.9269850 156.0001       0
#> X4  1.0587589 1.0461936 1.0713242 165.3473       0
#> X5  0.9498494 0.9347962 0.9649027 123.8220       0
#> X6  1.0283928 1.0161501 1.0406355 164.8376       0
#> X7  1.0228705 1.0116945 1.0340466 179.5998       0
#> X8  0.9880933 0.9741181 1.0020684 138.7443       0
#> X9  0.9234496 0.9131186 0.9337806 175.4065       0
#> X10 0.9133173 0.9010727 0.9255619 146.3696       0
#>
#> $coef.exp
#>     estimate lower.CI upper.CI   t-stat p-value
#> X1  2.721660 2.694703 2.748886 197.3867       0
#> X2  2.843997 2.809209 2.879216 166.6510       0
#> X3  2.497947 2.469346 2.526879 156.0001       0
#> X4  2.882791 2.846794 2.919243 165.3473       0
#> X5  2.585320 2.546694 2.624532 123.8220       0
#> X6  2.796568 2.762539 2.831016 164.8376       0
#> X7  2.781167 2.750257 2.812424 179.5998       0
#> X8  2.686108 2.648830 2.723910 138.7443       0
#> X9  2.517961 2.492082 2.544109 175.4065       0
#> X10 2.492578 2.462243 2.523286 146.3696       0
#>
#> $lambda
#> [1] 0
#>
#> $alpha
#> [1] 0
#>
#> $features.mean
#>          X1          X2          X3          X4          X5          X6
#> -0.02081123 -0.03844643 -0.01947003  0.02845033 -0.01993854  0.01782694
#>          X7          X8          X9         X10
#> -0.01202580  0.04699838 -0.00599775  0.01309362
#>
#> $features.sd
#>        X1        X2        X3        X4        X5        X6        X7        X8
#> 1.0170818 1.0161579 1.0041936 0.9956090 1.0160725 0.9824744 0.9993097 1.0000475
#>        X9       X10
#> 0.9905759 1.0180216
```

Note that the current package only supports a path to a `.csv` (comma separated values) files.

Fitting Cox Model with sparse data

Now we consider dataset `sparseSurvData` with number of features (columns) larger than number of observations (rows). `bigSurvSGD` package handles sparse datasets. The following fits a regularized Cox Model with the elastic net penalty where the penalty coefficients of l_1 and l_2 norms are $alpha * lambda = 0.09$ and $(1 - alpha) * lambda = 0.01$, respectively.

```
data("sparseSurvData")
fitBigSurvSGD <- bigSurvSGD(formula=Surv(time=time, status=status)~.,data=sparseSurvData,
                            alpha = 0.9, lambda = 0.1)
fitBigSurvSGD$coef
#>                 [,1]
#>   [1,]  9.833215e-01
#>   [2,]  1.236850e+00
#>   [3,]  5.620576e-01
#>   [4,]  4.887365e-01
#>   [5,]  1.500420e+00
#>   [6,]  9.688801e-01
#>   [7,]  1.086457e+00
#>   [8,]  1.058125e+00
#>   [9,]  7.597682e-01
#>  [10,]  7.047179e-01
#>  [11,]  3.553877e-01
#>  [12,] -5.181407e-01
#>  [13,]  9.921142e-03
#>  [14,] -1.371236e-01
#>  [15,] -8.627290e-03
#>  [16,]  5.641366e-01
#>  [17,]  1.697434e-01
#>  [18,]  4.355986e-02
#>  [19,] -1.912925e-01
#>  [20,]  4.019959e-02
#>  [21,]  1.427245e-01
#>  [22,] -3.254819e-01
#>  [23,]  7.956402e-04
#>  [24,]  5.114463e-02
#>  [25,]  1.775065e-02
#>  [26,] -6.785533e-03
#>  [27,]  4.493350e-01
#>  [28,] -1.615560e-02
#>  [29,]  3.501832e-03
#>  [30,] -3.240669e-01
#>  [31,]  0.000000e+00
#>  [32,] -6.997445e-02
#>  [33,]  1.329800e-02
#>  [34,] -2.711686e-03
#>  [35,]  8.216685e-04
#>  [36,]  5.721543e-01
#>  [37,] -6.179943e-03
#>  [38,] -1.345703e-01
#>  [39,]  8.440543e-02
#>  [40,] -1.419525e-01
#>  [41,] -1.935782e-03
#>  [42,]  1.284617e-02
#>  [43,]  9.067068e-02
#>  [44,]  5.564311e-02
#>  [45,] -3.254866e-01
#>  [46,]  3.258191e-02
#>  [47,] -1.550116e-01
#>  [48,] -7.088588e-02
#>  [49,]  0.000000e+00
#>  [50,] -1.609095e-01
#>  [51,] -2.405919e-03
#>  [52,]  0.000000e+00
#>  [53,] -6.801934e-03
#>  [54,] -1.116882e-01
#>  [55,]  6.842079e-02
#>  [56,] -2.465876e-03
#>  [57,] -7.208869e-01
#>  [58,] -5.652801e-02
#>  [59,] -6.648701e-03
#>  [60,]  9.623186e-02
#>  [61,] -6.364723e-03
#>  [62,]  2.421893e-01
#>  [63,] -1.089100e-01
#>  [64,]  3.087937e-02
#>  [65,]  2.276635e-02
#>  [66,]  0.000000e+00
#>  [67,]  4.978472e-04
#>  [68,]  1.340537e-02
#>  [69,] -2.295164e-03
#>  [70,]  8.405551e-01
#>  [71,] -1.039229e-01
#>  [72,]  4.634097e-01
#>  [73,]  2.439762e-01
#>  [74,] -1.423271e-03
#>  [75,]  5.804132e-04
#>  [76,] -6.617824e-02
#>  [77,]  0.000000e+00
#>  [78,]  6.412053e-04
#>  [79,]  1.740016e-03
#>  [80,]  4.076892e-03
#>  [81,]  1.534972e-01
#>  [82,]  1.289462e-03
#>  [83,]  1.435075e-01
#>  [84,]  1.537703e-01
#>  [85,]  5.430385e-02
#>  [86,] -1.129325e-01
#>  [87,] -2.859895e-01
#>  [88,]  0.000000e+00
#>  [89,]  0.000000e+00
#>  [90,]  1.330452e-01
#>  [91,] -1.980532e-01
#>  [92,]  1.756017e-01
#>  [93,]  4.908763e-01
#>  [94,]  1.243001e-02
#>  [95,] -1.367657e-02
#>  [96,] -1.367067e-02
#>  [97,] -3.349094e-03
#>  [98,]  5.235237e-01
#>  [99,]  2.810796e-01
#> [100,]  0.000000e+00
#> [101,]  1.049011e-01
#> [102,] -2.731232e-04
#> [103,] -2.159474e-01
#> [104,] -8.279002e-02
#> [105,] -9.353724e-02
#> [106,]  8.226880e-04
#> [107,]  0.000000e+00
#> [108,]  2.868932e-01
#> [109,]  1.290147e-01
#> [110,]  0.000000e+00
#> [111,] -3.933387e-01
#> [112,] -1.155195e-01
#> [113,] -3.448321e-01
#> [114,]  3.969045e-01
#> [115,] -3.297246e-01
#> [116,]  0.000000e+00
#> [117,] -5.865355e-02
#> [118,] -4.203928e-01
#> [119,] -6.268132e-03
#> [120,] -1.998739e-02
#> [121,] -2.649363e-01
#> [122,]  0.000000e+00
#> [123,]  1.903078e-02
#> [124,] -1.146917e-01
#> [125,]  2.764774e-03
#> [126,]  1.230062e-01
#> [127,]  0.000000e+00
#> [128,] -3.465598e-01
#> [129,] -2.310705e-01
#> [130,] -9.291870e-06
#> [131,] -5.011697e-03
#> [132,] -4.145086e-02
#> [133,]  1.478242e-02
#> [134,]  9.835482e-02
#> [135,]  3.803453e-03
#> [136,] -1.960436e-03
#> [137,]  0.000000e+00
#> [138,]  2.908115e-02
#> [139,]  0.000000e+00
#> [140,]  0.000000e+00
#> [141,] -2.076465e-01
#> [142,] -6.150218e-02
#> [143,]  2.748722e-01
#> [144,] -1.922117e-02
#> [145,]  3.515583e-03
#> [146,]  4.488588e-02
#> [147,] -2.592750e-01
#> [148,]  2.120511e-01
#> [149,] -1.734547e-01
#> [150,]  6.942456e-04
#> attr(,"names")
#>   [1] "X1"    "X2"    "X3"    "X4"    "X5"    "X6"    "X7"    "X8"    "X9"    "X10"
#>  [11] "X11"  "X12"  "X13"  "X14"  "X15"  "X16"  "X17"  "X18"  "X19"  "X20"
#>  [21] "X21"  "X22"  "X23"  "X24"  "X25"  "X26"  "X27"  "X28"  "X29"  "X30"
#>  [31] "X31"  "X32"  "X33"  "X34"  "X35"  "X36"  "X37"  "X38"  "X39"  "X40"
#>  [41] "X41"  "X42"  "X43"  "X44"  "X45"  "X46"  "X47"  "X48"  "X49"  "X50"
#>  [51] "X51"  "X52"  "X53"  "X54"  "X55"  "X56"  "X57"  "X58"  "X59"  "X60"
#>  [61] "X61"  "X62"  "X63"  "X64"  "X65"  "X66"  "X67"  "X68"  "X69"  "X70"
#>  [71] "X71"  "X72"  "X73"  "X74"  "X75"  "X76"  "X77"  "X78"  "X79"  "X80"
#>  [81] "X81"  "X82"  "X83"  "X84"  "X85"  "X86"  "X87"  "X88"  "X89"  "X90"
#>  [91] "X91"  "X92"  "X93"  "X94"  "X95"  "X96"  "X97"  "X98"  "X99"  "X100"
#> [101] "X101" "X102" "X103" "X104" "X105" "X106" "X107" "X108" "X109" "X110"
#> [111] "X111" "X112" "X113" "X114" "X115" "X116" "X117" "X118" "X119" "X120"
#> [121] "X121" "X122" "X123" "X124" "X125" "X126" "X127" "X128" "X129" "X130"
#> [131] "X131" "X132" "X133" "X134" "X135" "X136" "X137" "X138" "X139" "X140"
#> [141] "X141" "X142" "X143" "X144" "X145" "X146" "X147" "X148" "X149" "X150"
```

Note that if the dataset is very large, users may want to specify maximum number of rows per chunk specified by `max.rows.chunk` to read data chunk-by-chunk off the memory. This avoids lack-of-memory issue, as we discussed in the previous section.