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")
```

Users may change the `repos` options depending on their locations and preferences. Alternatively, users can install the package using Github repository:

```r
library(devtools)
install_github("atarkhan/bigSurvSGD")
```

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  0.9980137 0.9864041 1.0096233 168.6917       0
#> X2  1.0485366 1.0385673 1.0585059 206.3923       0
#> X3  0.9172486 0.9067546 0.9277426 171.5217       0
#> X4  1.0581417 1.0452101 1.0710734 160.5699       0
#> X5  0.9518663 0.9386264 0.9651061 141.0806       0
#> X6  1.0324599 1.0221969 1.0427229 197.4113       0
#> X7  1.0236293 1.0126140 1.0346445 182.3565       0
#> X8  0.9870397 0.9725566 1.0015227 133.7358       0
#> X9  0.9235595 0.9139307 0.9331883 188.2202       0
#> X10 0.9180988 0.9084034 0.9277942 185.8220       0
#>
#> $coef.exp
#>     estimate lower.CI upper.CI   t-stat p-value
#> X1  2.712888 2.681574 2.744567 168.6917       0
#> X2  2.853472 2.825167 2.882062 206.3923       0
#> X3  2.502396 2.476273 2.528794 171.5217       0
#> X4  2.881012 2.843996 2.918510 160.5699       0
#> X5  2.590540 2.556468 2.625066 141.0806       0
#> X6  2.807965 2.779294 2.836931 197.4113       0
#> X7  2.783278 2.752787 2.814106 182.3565       0
#> X8  2.683279 2.644697 2.722424 133.7358       0
#> X9  2.518238 2.494107 2.542603 188.2202       0
#> X10 2.504524 2.480359 2.528925 185.8220       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 coefficents 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  0.9969852 0.9847186 1.0092517 159.4929       0
#> X2  1.0418111 1.0265665 1.0570557 134.1059       0
#> X3  0.9180651 0.9078106 0.9283196 175.6845       0
#> X4  1.0586586 1.0461546 1.0711625 166.1431       0
#> X5  0.9601850 0.9503239 0.9700462 191.0744       0
#> X6  1.0308922 1.0194509 1.0423335 176.8121       0
#> X7  1.0187692 1.0041930 1.0333454 137.1529       0
#> X8  0.9884350 0.9746343 1.0022356 140.5472       0
#> X9  0.9237685 0.9139756 0.9335614 185.1087       0
#> X10 0.9171348 0.9071419 0.9271278 180.1000       0
#>
#> $coef.exp
#>     estimate lower.CI upper.CI   t-stat p-value
#> X1  2.710099 2.677059 2.743547 159.4929       0
#> X2  2.834346 2.791465 2.877885 134.1059       0
#> X3  2.504440 2.478889 2.530254 175.6845       0
#> X4  2.882502 2.846683 2.918771 166.1431       0
#> X5  2.612180 2.586547 2.638066 191.0744       0
#> X6  2.803566 2.771672 2.835827 176.8121       0
#> X7  2.769784 2.729703 2.810452 137.1529       0
#> X8  2.687026 2.650198 2.724366 140.5472       0
#> X9  2.518765 2.494219 2.543552 185.1087       0
#> X10 2.502111 2.477232 2.527240 180.1000       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,]  0.8089366148
#>   [2,]  1.2715596726
#>   [3,]  0.6419496240
#>   [4,]  0.3619343929
#>   [5,]  1.4895524564
#>   [6,]  0.7680156042
#>   [7,]  1.0908969559
#>   [8,]  0.9677840304
#>   [9,]  0.7628536267
#>  [10,]  0.7564271085
#>  [11,]  0.3853607764
#>  [12,] -0.4811946717
#>  [13,]  0.0857531743
#>  [14,] -0.1005080253
#>  [15,]  0.0002515461
#>  [16,]  0.5752570813
#>  [17,]  0.1973205252
#>  [18,]  0.1240298428
#>  [19,] -0.2133152635
#>  [20,]  0.0573726981
#>  [21,]  0.1035677608
#>  [22,] -0.3463930565
#>  [23,]  0.0080984365
#>  [24,]  0.0001439110
#>  [25,]  0.0302199050
#>  [26,]  0.0094110360
#>  [27,]  0.4917954798
#>  [28,] -0.0236165719
#>  [29,] -0.0064579561
#>  [30,] -0.3081721403
#>  [31,] -0.0080616492
#>  [32,] -0.1100318146
#>  [33,]  0.0406763688
#>  [34,] -0.0887400972
#>  [35,]  0.0001565026
#>  [36,]  0.4630150088
#>  [37,]  0.0000000000
#>  [38,] -0.1874326712
#>  [39,]  0.0006407360
#>  [40,] -0.1272590495
#>  [41,]  0.0000000000
#>  [42,]  0.1100825798
#>  [43,]  0.1989075369
#>  [44,]  0.2017220949
#>  [45,] -0.1489729560
#>  [46,]  0.0278351149
#>  [47,] -0.2558661660
#>  [48,] -0.0593011998
#>  [49,]  0.0037766340
#>  [50,] -0.1307343026
#>  [51,] -0.0013542682
#>  [52,]  0.0203296745
#>  [53,]  0.0000000000
#>  [54,] -0.1586016370
#>  [55,]  0.2294775536
#>  [56,] -0.0009828585
#>  [57,] -0.7723479238
#>  [58,] -0.0097985966
#>  [59,]  0.0121749804
#>  [60,]  0.2315136194
#>  [61,]  0.0022577939
#>  [62,]  0.2813407795
#>  [63,] -0.0810434117
#>  [64,]  0.0457737308
#>  [65,]  0.0354105409
#>  [66,]  0.0000000000
#>  [67,]  0.0695542352
#>  [68,]  0.0170346234
#>  [69,] -0.0275946880
#>  [70,]  1.0377516021
#>  [71,] -0.0559274907
#>  [72,]  0.5301564883
#>  [73,]  0.1300650837
#>  [74,]  0.0023884097
#>  [75,]  0.0082017416
#>  [76,] -0.0814512955
#>  [77,] -0.0011166663
#>  [78,]  0.0000000000
#>  [79,]  0.0033950655
#>  [80,]  0.0537391128
#>  [81,]  0.1478113994
#>  [82,]  0.0014915315
#>  [83,]  0.1589698506
#>  [84,]  0.1129173326
#>  [85,]  0.0001245675
#>  [86,] -0.1655042203
#>  [87,] -0.2590369070
#>  [88,]  0.0070838829
#>  [89,]  0.0031760429
#>  [90,]  0.3213389139
#>  [91,] -0.2060658543
#>  [92,]  0.1541432184
#>  [93,]  0.3914761294
#>  [94,] -0.0037302390
#>  [95,] -0.0027484040
#>  [96,]  0.0000000000
#>  [97,] -0.0008572930
#>  [98,]  0.5597009597
#>  [99,]  0.2447947138
#> [100,] -0.0001281835
#> [101,]  0.1648651203
#> [102,]  0.0630844106
#> [103,] -0.1316311000
#> [104,] -0.1463243480
#> [105,] -0.1546126517
#> [106,]  0.0013974756
#> [107,]  0.0007699179
#> [108,]  0.4290332032
#> [109,]  0.0783236051
#> [110,]  0.0010046573
#> [111,] -0.3723193033
#> [112,] -0.0859389232
#> [113,] -0.3423885063
#> [114,]  0.4341478816
#> [115,] -0.2366685716
#> [116,] -0.0888845055
#> [117,] -0.0018762729
#> [118,] -0.3403597629
#> [119,] -0.1173020762
#> [120,]  0.0000000000
#> [121,] -0.1707853403
#> [122,]  0.0064603006
#> [123,]  0.0296627738
#> [124,] -0.0414488637
#> [125,] -0.0008525683
#> [126,]  0.1043190867
#> [127,]  0.0006886184
#> [128,] -0.3450512023
#> [129,] -0.1535677494
#> [130,]  0.0105589871
#> [131,] -0.0480329961
#> [132,] -0.0570280699
#> [133,]  0.0322069645
#> [134,]  0.1042326835
#> [135,]  0.0534383550
#> [136,] -0.0060633780
#> [137,] -0.0093968639
#> [138,]  0.0493624938
#> [139,]  0.0090919110
#> [140,] -0.0025420926
#> [141,] -0.1205097056
#> [142,] -0.0608209821
#> [143,]  0.2369959798
#> [144,] -0.0160429579
#> [145,] -0.0049880939
#> [146,]  0.0173997659
#> [147,] -0.2058461804
#> [148,]  0.2245154692
#> [149,] -0.2287768177
#> [150,]  0.0062104762
#> 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.