610 HW7

Joe Stoica and Corey Maxedon

11/10/2019

1

No output required.

2

```
get_inv_cov <- function(lambda, data=df) {
  p = ncol(data)
  theta = Variable(rows = p, cols = p)
  S <- cov(data)
  objective = Minimize(-log_det(theta) + matrix_trace(S %*% theta) + lambda * sum(abs(theta)))
  problem = Problem(objective)
  result = psolve(problem)
  return(result$getValue(theta))
}

get_inv_cov(5)
```

```
##                [,1]          [,2]          [,3]          [,4]
##  [1,]  1.412874e-01  7.447805e-09  1.209544e-08  1.007046e-08
##  [2,] -6.737036e-08  1.823873e-01 -1.392103e-09 -1.649375e-09
##  [3,] -6.062954e-08  1.575565e-08  1.808693e-01 -2.478687e-09
##  [4,] -6.750301e-08  1.471612e-08  1.355898e-08  1.808849e-01
##  [5,] -5.806431e-08  1.672345e-08  1.692370e-08  1.954892e-08
##  [6,] -6.533892e-08  1.403304e-08  1.522966e-08  1.452630e-08
##  [7,] -6.490164e-08  1.250972e-08  1.657699e-08  1.548003e-08
##  [8,] -6.696487e-08  1.396312e-08  1.277300e-08  1.327701e-08
##  [9,] -6.380677e-08  1.056801e-08  1.348590e-08  1.475121e-08
## [10,]  7.120130e-07 -5.793081e-08 -5.718386e-08 -6.124532e-08
##                [,5]          [,6]          [,7]          [,8]
##  [1,]  6.805407e-09  1.331933e-08  1.514234e-08  1.645114e-08
##  [2,]  8.070226e-11 -2.322041e-09 -3.051204e-09 -2.953724e-09
##  [3,] -2.934052e-10 -2.513627e-09 -2.440874e-09 -3.860981e-09
##  [4,]  5.406973e-10 -2.611368e-09 -2.647360e-09 -3.607362e-09
##  [5,]  1.845077e-01 -1.630892e-09 -2.463860e-09 -2.807626e-09
##  [6,]  1.358756e-08  1.797281e-01 -3.606193e-09 -3.533720e-09
##  [7,]  1.158220e-08  1.364055e-08  1.787890e-01 -3.961936e-09
##  [8,]  1.137944e-08  1.511312e-08  1.453560e-08  1.777709e-01
##  [9,]  1.461628e-08  1.548015e-08  1.689347e-08  1.879351e-08
## [10,] -4.393149e-08 -6.433685e-08 -6.497760e-08 -6.909148e-08
##                [,9]         [,10]
##  [1,]  1.531273e-08  3.246072e-07
##  [2,] -3.552775e-09 -2.376171e-08
##  [3,] -3.274152e-09 -2.349445e-08
```

```
##  [4,] -2.807571e-09 -2.525355e-08
##  [5,] -1.553929e-09 -1.751630e-08
##  [6,] -3.034285e-09 -2.661286e-08
##  [7,] -2.885832e-09 -2.698175e-08
##  [8,] -2.605611e-09 -2.891187e-08
##  [9,]  1.789782e-01 -2.632586e-08
## [10,] -6.373390e-08  1.382238e-01
```

3

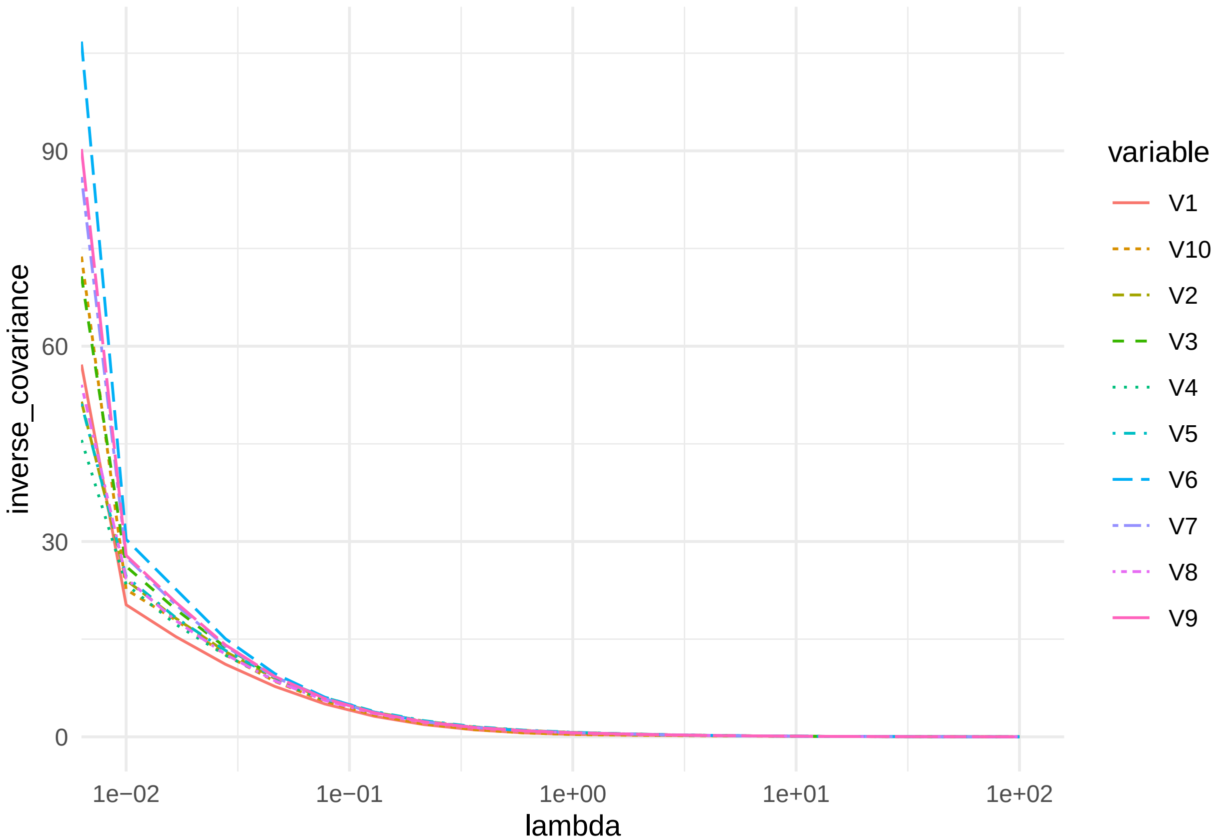
```r
lambda_search = c(0, 10^(seq(-2, 2, length.out = 19))) # TODO rerun this chunk
inv_cov_matrices = aaply(lambda_search, 1, function(x) diag(get_inv_cov(x)))
colnames(inv_cov_matrices) = colnames(df)
inv_cov_matrices = cbind(lambda = lambda_search, inv_cov_matrices)

inv_cov_melted = reshape2::melt(data.frame(inv_cov_matrices),
                                id.vars = "lambda",
                                value.name = "inverse_covariance")

write.csv(inv_cov_melted, "inv_cov_melted.csv", row.names = FALSE)
```

```r
inv_cov_melted <- read_csv("inv_cov_melted.csv")

inv_cov_melted %>%
  ggplot(aes(x = lambda, y = inverse_covariance,
             color = variable, lty = variable)) +
  geom_line()+
  scale_x_log10()+
  theme_minimal()
```



Here we can see that as lambda is increased, the values for our inverse covariance approaches zero.

4

```
kfold_cv <- function(lambda, df, folds) {
  n = nrow(df)

  # assign rows from df to different samples
  samples = sample(n, n)

  # these labels will match up to i in the for loop
  labels = rep(1:folds, each = n / folds)
  fold_labels = cbind(samples, labels)

  # pre-allocate space
  nll_vec = numeric(folds)

  for (i in 1:folds) {

    # find the rows that match i, these will be for our testing df
    hold_out <- which(fold_labels[, 2] == i)
    test_rows = fold_labels[hold_out, 1]

    # training data
    train = df[-test_rows, ]
```

```r
    # testing (held out) data
    test = df[test_rows, ]

    # fit the model excluding the indices I_i
    theta_hat_i = get_inv_cov(lambda, train)

    # covariance computed just on the hold out
    S = cov(test)

    # calculate the negative log likelihood
    nll_vec[i] = -log(det(theta_hat_i)) + sum(diag(S %*% theta_hat_i))
  }

  # computing overall negative log likelihood
  return(sum(nll_vec))
}
```

```r
# This also takes forever and I just saved it as a CSV so I don't have to wait
# for RMD compilation
neglik_vector = sapply(lambda_search, kfold_cv, df = df, folds = 10)
neg_lik_df <- cbind(lambda_search, neglik_vector)
write.csv(neg_lik_df, "neg_lik_df.csv", row.names = FALSE)
```

```r
read_csv("neg_lik_df.csv",) %>% as.matrix()
```

```
##       lambda_search neglik_vector
##  [1,]    0.01000000    -220.12629
##  [2,]    0.01623777    -205.00578
##  [3,]    0.02636651    -184.75296
##  [4,]    0.04281332    -157.54507
##  [5,]    0.06951928    -129.26951
##  [6,]    0.11288379     -90.68051
##  [7,]    0.18329807     -52.99702
##  [8,]    0.29763514     -14.68367
##  [9,]    0.48329302      30.16608
## [10,]    0.78475997      67.71028
## [11,]    1.27427499     106.75086
## [12,]    2.06913808     126.12286
## [13,]    3.35981829     163.24567
## [14,]    5.45559478     196.93215
## [15,]    8.85866790     236.21338
## [16,]   14.38449888     277.95707
## [17,]   23.35721469     322.47378
## [18,]   37.92690191     367.65386
## [19,]   61.58482111     414.83989
## [20,]  100.00000000     462.22302
```

Here we can see that the lower the value of lambda, the lower the negative log likelihood. Basically. choosing a lambda value that's 0 will result in the best results. The reason zero doesn't have the best negative log-likelihood is most likely due to random chance.

5

```r
get_inv_cov_update <- function(data) {

  p = ncol(data)
  theta = Variable(rows = p, cols = p)
  S <- cov(data)
  objective = Minimize(-log_det(theta) + matrix_trace(S %*% theta))

  # Creating constraints
  i <- 2:9
  j <- 2:9

  grid <- expand.grid(i = i, j = j) %>%
    filter(i < j)

  constraint_list <- alply(grid, 1, function(row){
    # this sets everything in the upper triangule to 0
    # besides the diagonal and the upper and rightmost border of the matrix
    theta[row[, 1], row[, 2]] == 0
  }
  )

  problem = Problem(objective, constraint_list)
  result = psolve(problem)
  return(result$getValue(theta))
}

round(get_inv_cov_update(df), 3)
```

```
##         [,1]    [,2]   [,3]    [,4]   [,5]    [,6]    [,7]    [,8]    [,9]
##  [1,] 40.614   3.681 18.274   4.182 15.957  17.400  13.969   6.847  16.766
##  [2,]  3.681  44.081  0.000   0.000  0.000   0.000   0.000   0.000   0.000
##  [3,] 18.274   0.000 55.976   0.000  0.000   0.000   0.000   0.000   0.000
##  [4,]  4.182   0.000  0.000  40.655  0.000   0.000   0.000   0.000   0.000
##  [5,] 15.957   0.000  0.000   0.000 43.606   0.000   0.000   0.000   0.000
##  [6,] 17.400   0.000  0.000   0.000  0.000  94.176   0.000   0.000   0.000
##  [7,] 13.969   0.000  0.000   0.000  0.000   0.000  63.570   0.000   0.000
##  [8,]  6.847   0.000  0.000   0.000  0.000   0.000   0.000  46.990   0.000
##  [9,] 16.766   0.000  0.000   0.000  0.000   0.000   0.000   0.000  67.511
## [10,] -8.143 -16.498 -9.236 -15.318 -3.031 -30.246 -18.961 -17.897 -18.129
##         [,10]
##  [1,]  -8.143
##  [2,] -16.498
##  [3,]  -9.236
##  [4,] -15.318
##  [5,]  -3.031
##  [6,] -30.246
##  [7,] -18.961
##  [8,] -17.897
##  [9,] -18.129
## [10,]  56.195
```

Here is the rounded estimate from the data.

6

```
bootstrap <- function(data) {
  sample = sample(1:50, 50, replace = TRUE)
  temp = data[sample, ]
  theta_b <- get_inv_cov_update(temp)
}

# This takes forever to run, so I ran it once and saved the final result in a
# csv down below
B <- 100
results <- replicate(B, bootstrap(df))
```

The bootstrap runs above. It really takes a long time to run so there won't be any output above.

```
# Constructing a grid to use to get all of the quantiles
grid <- expand.grid(x = 1:10, y = 1:10)

# this creates a dataframe for each layer of the bootstrap results
ci_df <- adply(grid, 1, function(r){
  quantile(results[r[,1], r[,2],], c(0.025, 0.975))
})

ci_df <- ci_df %>%
  # filter out the values that are 0
  filter(`2.5%` != 0 & `97.5%` != 0 ) %>%
  arrange(x, y)

# save the csv
write.csv(ci_df, "ci_df.csv", row.names = FALSE)
```

```
read_csv("ci_df.csv") %>% as.matrix()
```

```
##         x  y        2.5%      97.5%
##   [1,]  1  1  37.053375  65.959350
##   [2,]  1  2  -6.430375  16.595400
##   [3,]  1  3   7.480150  32.502775
##   [4,]  1  4  -4.722975  17.018775
##   [5,]  1  5   9.280400  27.824325
##   [6,]  1  6   2.439400  32.165050
##   [7,]  1  7   5.586575  26.272525
##   [8,]  1  8   1.059775  19.012875
##   [9,]  1  9   6.298225  30.485350
##  [10,]  1 10 -15.715675   8.709675
##  [11,]  2  1  -6.430375  16.595400
##  [12,]  2  2  34.302400  68.435675
##  [13,]  2 10 -36.038175  -9.021975
##  [14,]  3  1   7.480150  32.502775
##  [15,]  3  3  43.584575  98.534650
##  [16,]  3 10 -23.896775   0.548025
##  [17,]  4  1  -4.722975  17.018775
##  [18,]  4  4  29.108550  59.391500
##  [19,]  4 10 -25.726375  -9.178525
##  [20,]  5  1   9.280400  27.824325
```

```
## [21,]  5  5  34.718450  69.420650
## [22,]  5 10 -11.711900   3.161775
## [23,]  6  1   2.439400  32.165050
## [24,]  6  6  68.942775 152.979375
## [25,]  6 10 -51.110275 -19.339050
## [26,]  7  1   5.586575  26.272525
## [27,]  7  7  49.598250 100.966325
## [28,]  7 10 -35.423775 -11.591575
## [29,]  8  1   1.059775  19.012875
## [30,]  8  8  38.827350  72.232350
## [31,]  8 10 -31.192125 -11.085925
## [32,]  9  1   6.298225  30.485350
## [33,]  9  9  50.620650 103.782775
## [34,]  9 10 -31.619475  -8.047350
## [35,] 10  1 -15.715675   8.709675
## [36,] 10  2 -36.038175  -9.021975
## [37,] 10  3 -23.896775   0.548025
## [38,] 10  4 -25.726375  -9.178525
## [39,] 10  5 -11.711900   3.161775
## [40,] 10  6 -51.110275 -19.339050
## [41,] 10  7 -35.423775 -11.591575
## [42,] 10  8 -31.192125 -11.085925
## [43,] 10  9 -31.619475  -8.047350
## [44,] 10 10  50.612475  87.058075
```

Here are all of the confidence intervals for the portion of the upper triangle that aren't zero.