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]          [,5]
##  [1,] 1.413040e-01  5.678242e-08  5.918907e-08  5.672798e-08  5.780301e-08
##  [2,] 2.507180e-07  1.823866e-01 -5.151694e-08 -5.192415e-08 -4.773454e-08
##  [3,] 2.646992e-07 -1.719967e-07  1.808716e-01 -5.529544e-08 -5.006357e-08
##  [4,] 2.597271e-07 -1.727238e-07 -1.834903e-07  1.808871e-01 -4.830552e-08
##  [5,] 2.423702e-07 -1.548657e-07 -1.629256e-07 -1.589971e-07  1.845019e-01
##  [6,] 2.715011e-07 -1.803784e-07 -1.880450e-07 -1.891093e-07 -1.714338e-07
##  [7,] 2.755309e-07 -1.849901e-07 -1.886999e-07 -1.903366e-07 -1.768260e-07
##  [8,] 2.779733e-07 -1.852205e-07 -1.966657e-07 -1.959678e-07 -1.792376e-07
##  [9,] 2.763165e-07 -1.876099e-07 -1.927455e-07 -1.909353e-07 -1.716600e-07
## [10,] 8.986119e-07 -2.653827e-07 -2.749393e-07 -2.766843e-07 -2.509904e-07
##                [,6]          [,7]          [,8]          [,9]
##  [1,]  5.794547e-08  5.733799e-08  5.572179e-08  5.747487e-08
##  [2,] -5.353827e-08 -5.456973e-08 -5.345037e-08 -5.575109e-08
##  [3,] -5.533367e-08 -5.442740e-08 -5.689569e-08 -5.628361e-08
##  [4,] -5.594502e-08 -5.531544e-08 -5.668502e-08 -5.554254e-08
##  [5,] -5.149345e-08 -5.296665e-08 -5.296672e-08 -5.049447e-08
##  [6,]  1.797324e-01 -5.959559e-08 -5.852159e-08 -5.810101e-08
##  [7,] -2.033618e-07  1.787948e-01 -6.033327e-08 -5.860511e-08
##  [8,] -2.039811e-07 -2.099545e-07  1.777783e-01 -5.892563e-08
##  [9,] -2.003368e-07 -2.034899e-07 -2.063762e-07  1.789836e-01
## [10,] -2.893221e-07 -2.944453e-07 -3.010367e-07 -2.968315e-07
##               [,10]
##  [1,]  3.238057e-07
##  [2,] -4.208164e-08
```

```
##  [3,] -4.187177e-08
##  [4,] -4.316316e-08
##  [5,] -4.064925e-08
##  [6,] -4.459324e-08
##  [7,] -4.469775e-08
##  [8,] -4.549216e-08
##  [9,] -4.582821e-08
## [10,]  1.382470e-01
```

3

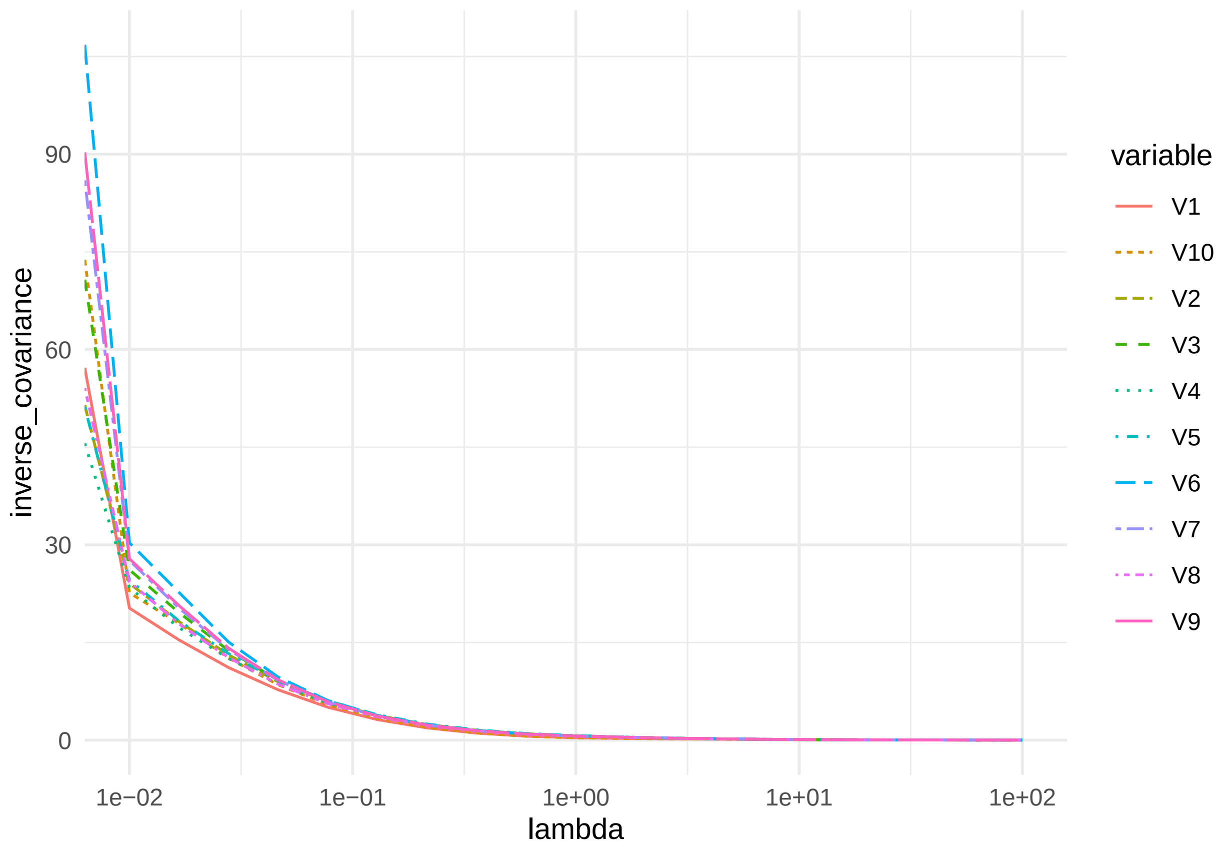
```r
lambda_search = c(0, 10^(seq(-2, 2, length.out = 19))) # TODO rerun this chunk
inv_cov_matrices = aaply(lambda_search, 1, function(x) diag(get_inv_cov(x)))
colnames(inv_cov_matrices) = colnames(df)
inv_cov_matrices = cbind(lambda = lambda_search, inv_cov_matrices)

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
```

```r
    train = df[-test_rows, ]

    # 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,]    0.00000000    -214.38992
##  [2,]    0.01000000    -216.26470
##  [3,]    0.01668101    -200.41873
##  [4,]    0.02782559    -180.57728
##  [5,]    0.04641589    -155.38860
##  [6,]    0.07742637    -120.23174
##  [7,]    0.12915497     -84.11231
##  [8,]    0.21544347     -39.85269
##  [9,]    0.35938137       2.87995
## [10,]    0.59948425      43.75154
## [11,]    1.00000000      88.93703
## [12,]    1.66810054     118.49892
## [13,]    2.78255940     150.62103
## [14,]    4.64158883     185.25581
## [15,]    7.74263683     225.00968
## [16,]   12.91549665     267.98175
## [17,]   21.54434690     314.96410
## [18,]   35.93813664     362.91954
## [19,]   59.94842503     412.14234
## [20,]  100.00000000     462.28262
```

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.

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 triangle 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,] 40.621   3.686 18.292   4.188 15.981  17.391  13.962   6.844  16.746
##  [2,]  3.686  44.087  0.000   0.000  0.000   0.000   0.000   0.000   0.000
##  [3,] 18.292   0.000 56.014   0.000  0.000   0.000   0.000   0.000   0.000
##  [4,]  4.188   0.000  0.000  40.669  0.000   0.000   0.000   0.000   0.000
##  [5,] 15.981   0.000  0.000   0.000 43.665   0.000   0.000   0.000   0.000
##  [6,] 17.391   0.000  0.000   0.000  0.000  94.131   0.000   0.000   0.000
##  [7,] 13.962   0.000  0.000   0.000  0.000   0.000  63.506   0.000   0.000
##  [8,]  6.844   0.000  0.000   0.000  0.000   0.000   0.000  47.007   0.000
##  [9,] 16.746   0.000  0.000   0.000  0.000   0.000   0.000   0.000  67.436
## [10,] -8.142 -16.496 -9.237 -15.319 -3.032 -30.229 -18.936 -17.909 -18.111
##         [,10]
##  [1,]  -8.142
##  [2,] -16.496
##  [3,]  -9.237
##  [4,] -15.319
##  [5,]  -3.032
##  [6,] -30.229
##  [7,] -18.936
##  [8,] -17.909
##  [9,] -18.111
## [10,]  56.173
```

Here is the rounded estimate from the data.

6

```r
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.

```r
# 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)
```

```r
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.