MCMVR Example

Aaron J. Molstad (amolstad@ufl.edu)

7/18/2019

In this document, we provide a short tutorial on how to use the `MCMVR` R package. We download this package from GitHub.

```
install.packages("devtools")
library(devtools)
devtools::install_github("ajmolstad/MCMVR")
library(MCMVR)
```

First, we generate data from the "errors-in-variables" data generating model described in Section 3.1 of the article.

```
set.seed(1)
p <- 50
q <- 10
n <- 100

beta <- matrix(rnorm(p*q)*sample(c(0,1), p*q, prob = c(.9, .1), replace=TRUE), nrow=p, ncol=q)

Z <- matrix(rnorm(n*p), nrow=n, ncol=p)
Y <- tcrossprod(Z, t(beta)) + matrix(rnorm(n*q, sd=1), nrow=n)
X <- Z + matrix(rnorm(n*p, sd=sqrt(0.5)), nrow=n)

Znew <- matrix(rnorm(n*p), nrow=n, ncol=p)
Xnew <- Znew + matrix(rnorm(n*p, sd=sqrt(0.5)), nrow=n)
Ynew <- tcrossprod(Znew, t(beta)) + matrix(rnorm(n*q, sd=1), nrow=n)
```

We fit the model using the cross-validation function. There are a number of key arguments: the first is `tau.vec`, where a user must specify a vector of candidate tuning parameters τ over which to fit the model:

$$\arg\min_{\beta} \operatorname{tr}\left\{n^{-1}(Y - X\beta)(\beta'\beta + \tau I_q)^{-1}(Y - X\beta)'\right\} + \frac{\lambda}{\tau}\operatorname{Pen}(\beta).$$

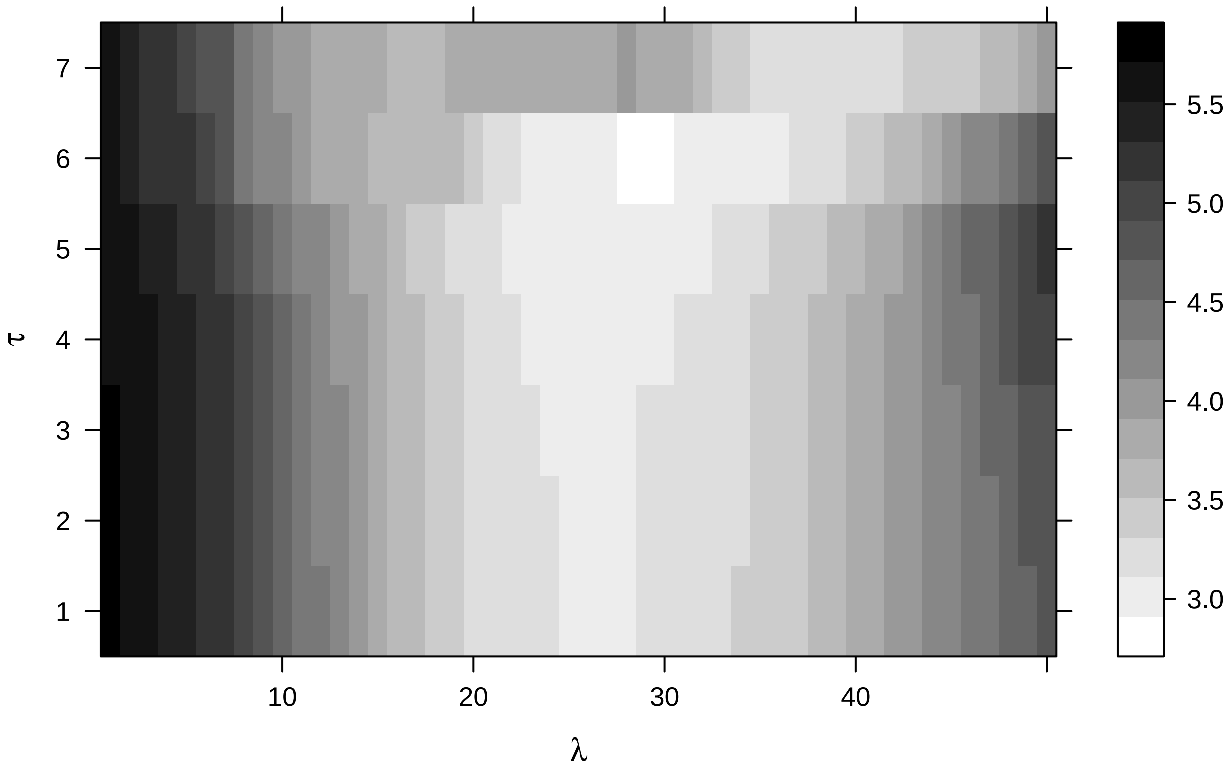
Another important argument is `nfolds`. If set to NULL, cross-validation is not performed, the model is fit to the complete data without cross-validation. Finally, a user must also decide which penalty to use. The current options are `penalty="L1"` and `penalty="NN"` which set $\operatorname{Pen}(\beta) = \sum_{j,k} |\beta_{j,k}|$ and $\operatorname{Pen}(\beta) = \sum_{j=1}^{\min(p,q)} \varphi_j(\beta)$, respectively, where $\varphi_j(\beta)$ is the jth largest singular value of β. Note that one only needs to input the number of candidate λ, `nlambda`, and the ratio of max to min lambda `delta`.

```
# ---------------------------------------------------
# Perform 5-fold CV for a grid of tuning parameters
# ---------------------------------------------------
tau.vec <- 10^seq(3, 0, by=-.5)
fit <- MCMVR.cv(X = X, Y = Y, tau.vec = tau.vec, nlambda = 50, nfolds = 5,
  delta = .01, tol = 1e-8, quiet= TRUE, inner.quiet= TRUE, penalty="L1")
str(fit)
```

```
## List of 10
##  $ beta        : num [1:3500, 1:50] 0 0 0 0 0 0 0 0 0 0 ...
##  $ sparsity.mat: num [1:50, 1:7] 0 1 2 2 2 2 5 6 8 8 ...
##  $ err.pred    : num [1:50, 1:7, 1:5] 6.66 6.58 6.51 6.45 6.4 ...
```

```
##  $ err.wpred   : num [1:50, 1:7, 1:5] 0.969 0.964 0.959 0.955 0.951 ...
##  $ Y.offset    : num [1:10] 0.2821 -0.1201 0.0626 -0.1167 -0.2071 ...
##  $ X.offset    : num [1:50] -0.03591 0.00857 -0.18562 -0.15828 -0.01975 ...
##  $ lambda.vec  : num [1:50] 6.12 5.57 5.07 4.62 4.2 ...
##  $ tau.vec     : num [1:7] 1000 316.2 100 31.6 10 ...
##  $ tau.min     : num 3.16
##  $ lam.min     : num 0.44
##  - attr(*, "class")= chr "EIVMR"
```

```r
# ---------------------------------------------------
# visualize CV error
# ---------------------------------------------------
library(lattice)
levelplot(apply(fit$err.pred, c(1,2), mean), col.regions=grey(100:0/100), xlab=expression(lambda), ylab=
```




```r
# ---------------------------------------------------
# get coefs from model which minimized CV error
# ---------------------------------------------------
betas <- MCMVR.coef(fit)$beta

par(mfrow=c(1,2))
image(betas!=0, col=grey(100:0/100), main=expression(paste("Nonzero entries of ", hat(beta), "\n")))
```

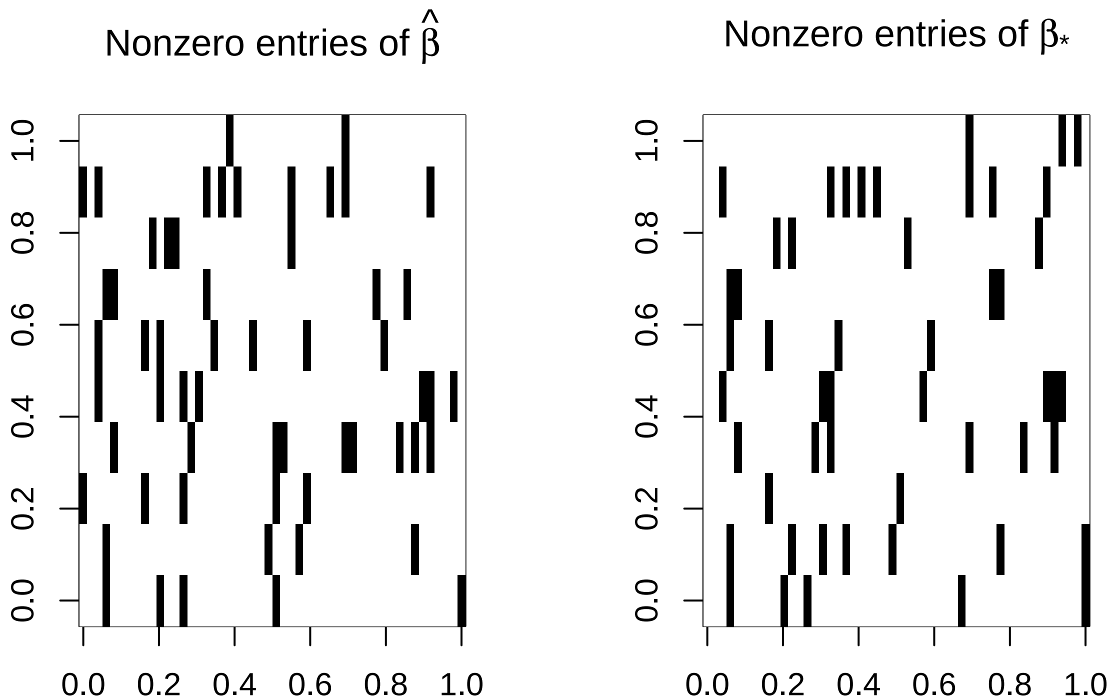
```
## Warning in title(...): font metrics unknown for character 0xa

## Warning in title(...): font metrics unknown for character 0xa
```

```r
image(beta!=0, col=grey(100:0/100), main=expression(paste("Nonzero entries of ", beta["*"], "\n")))
```

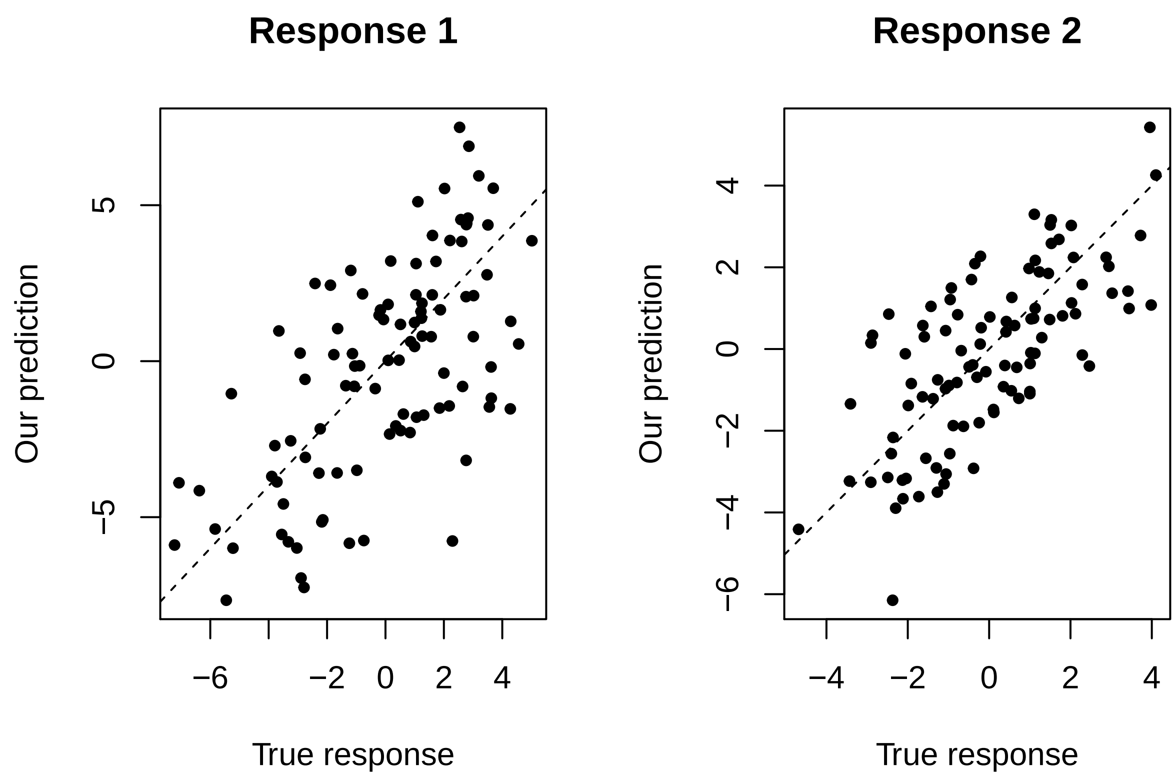
```
## Warning in title(...): font metrics unknown for character 0xa
```

```
## Warning in title(...): font metrics unknown for character 0xa
```



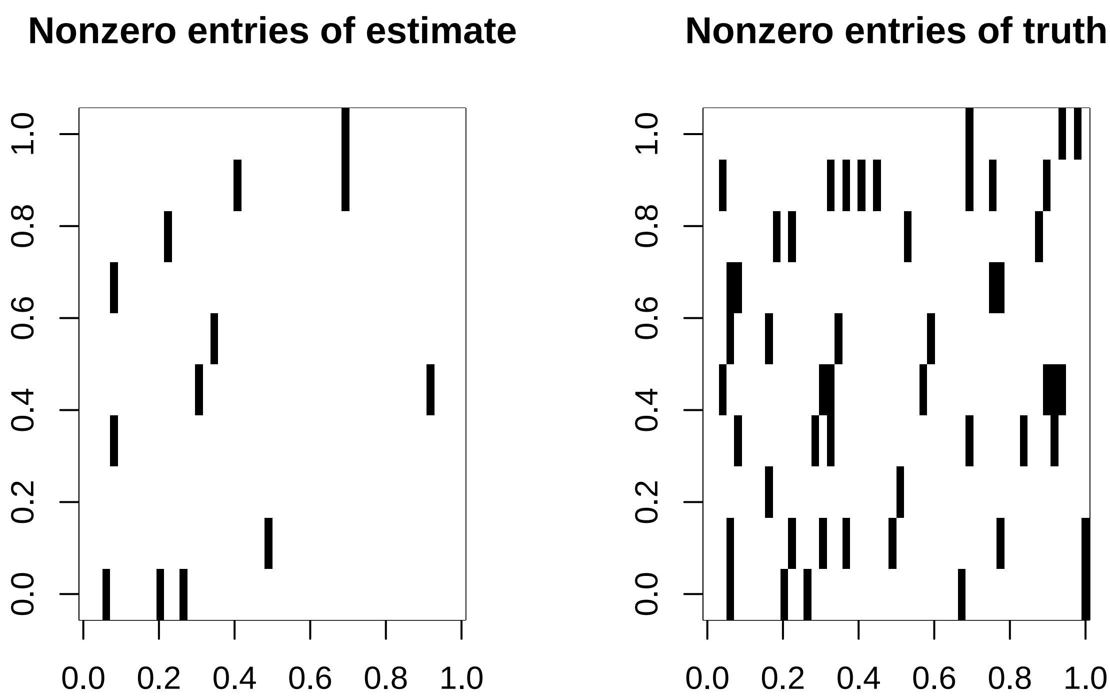

```r
# ------------------------------------------------
# make predictions for Xnew
# ------------------------------------------------
preds <- MCMVR.predict(Xnew, fit)

par(mfrow=c(1,2))
plot(preds$preds[,1], Ynew[,1], pch=20, main="Response 1", ylab="Our prediction", xlab="True response")
abline(0,1, lty=2)
plot(preds$preds[,2], Ynew[,2], pch=20, main="Response 2", ylab="Our prediction", xlab="True response")
abline(0,1, lty=2)
```




```
# ---------------------------------------------------
# get coefs from model with seperate tuning parameter
# ---------------------------------------------------
betas <- MCMVR.coef(fit, tau = fit$tau.vec[1], lambda=fit$lambda.vec[12])$beta

par(mfrow=c(1,2))
image(betas!=0, col=grey(100:0/100), main=paste("Nonzero entries of estimate"))
image(beta!=0, col=grey(100:0/100), main=paste("Nonzero entries of truth"))
```



We now also fit the model for the nuclear norm penalized version. Note that it is sometimes useful to relax the convergence tolerance for this version. We also turn off the `quiet` option. Note that `inner.quiet` should only be used for determining an appropriate value of `tol`.

```
# --------------------------------------------------
# Perform 5-fold CV for a grid of tuning parameters
# --------------------------------------------------
tau.vec <- 10^seq(3, 0, by=-1)
fit <- MCMVR.cv(X = X, Y = Y, tau.vec = tau.vec, nlambda = 10, nfolds = 5,
  delta = .01, tol = 1e-10, quiet = FALSE, inner.quiet= TRUE, penalty="NN")
```

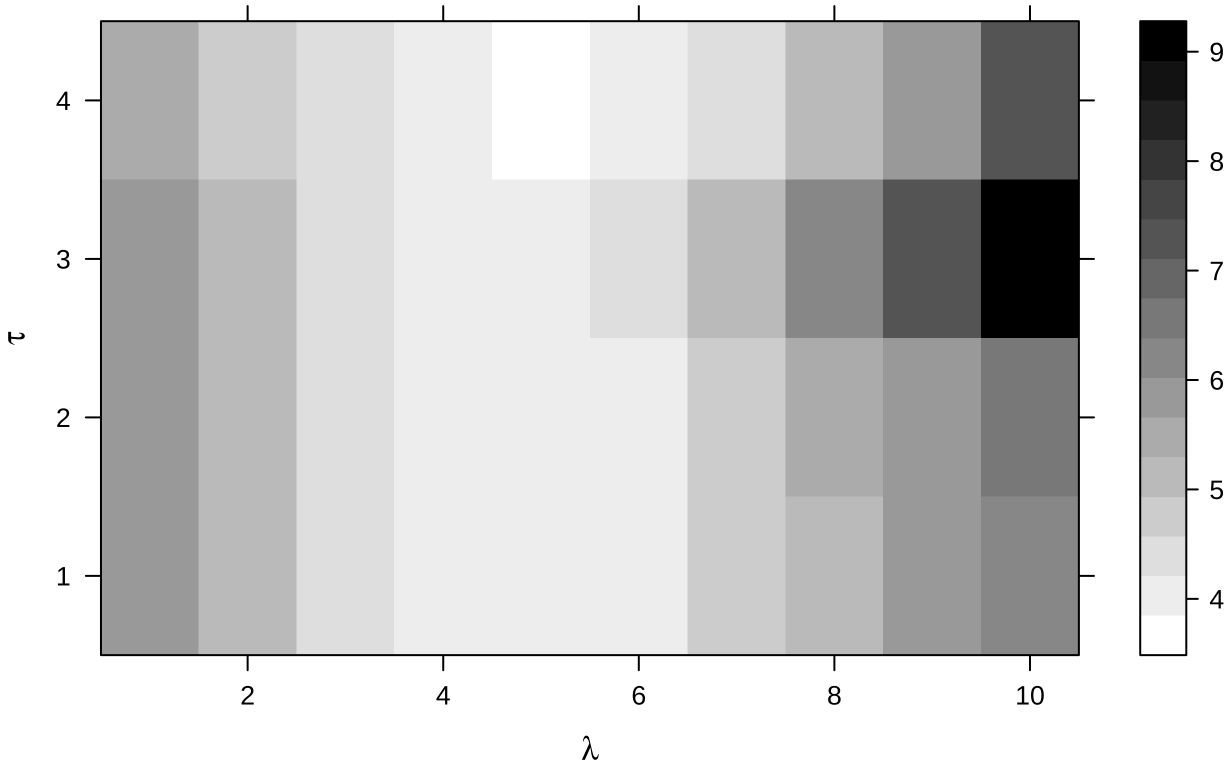
```
## 1 1 ; Nuclear norm =  0
## 1 2 ; Nuclear norm =  1.630469
## 1 3 ; Nuclear norm =  4.750991
## 1 4 ; Nuclear norm =  8.122151
## 1 5 ; Nuclear norm =  10.97407
## 1 6 ; Nuclear norm =  13.17827
## 1 7 ; Nuclear norm =  14.83245
## 1 8 ; Nuclear norm =  16.01548
## 1 9 ; Nuclear norm =  16.82551
## 1 10 ; Nuclear norm =  17.35355
## 2 1 ; Nuclear norm =  0
## 2 2 ; Nuclear norm =  1.661005
## 2 3 ; Nuclear norm =  4.792185
## 2 4 ; Nuclear norm =  8.16899
## 2 5 ; Nuclear norm =  11.03848
## 2 6 ; Nuclear norm =  13.27934
## 2 7 ; Nuclear norm =  14.98198
## 2 8 ; Nuclear norm =  16.22102
## 2 9 ; Nuclear norm =  17.07895
## 2 10 ; Nuclear norm =  17.64561
## 3 1 ; Nuclear norm =  0
## 3 2 ; Nuclear norm =  1.935984
```

```
## 3 3 ; Nuclear norm =  5.096542
## 3 4 ; Nuclear norm =  8.453984
## 3 5 ; Nuclear norm =  11.37895
## 3 6 ; Nuclear norm =  13.82129
## 3 7 ; Nuclear norm =  15.87913
## 3 8 ; Nuclear norm =  17.60286
## 3 9 ; Nuclear norm =  19.0213
## 3 10 ; Nuclear norm =  20.16546
## 4 1 ; Nuclear norm =  0
## 4 2 ; Nuclear norm =  2.816803
## 4 3 ; Nuclear norm =  5.69757
## 4 4 ; Nuclear norm =  8.106116
## 4 5 ; Nuclear norm =  10.49734
## 4 6 ; Nuclear norm =  12.68873
## 4 7 ; Nuclear norm =  14.86626
## 4 8 ; Nuclear norm =  17.09687
## 4 9 ; Nuclear norm =  19.38221
## 4 10 ; Nuclear norm =  21.69832
## Through CV fold 1
## Through CV fold 2
## Through CV fold 3
## Through CV fold 4
## Through CV fold 5
```

```
str(fit)
```

```
## List of 10
##  $ beta        : num [1:2000, 1:10] 0 0 0 0 0 0 0 0 0 0 ...
##  $ sparsity.mat: num [1:10, 1:4] 0 500 500 500 500 500 500 500 500 500 ...
##  $ err.pred    : num [1:10, 1:4, 1:5] 6.61 6.02 5.02 4.39 4.29 ...
##  $ err.wpred   : num [1:10, 1:4, 1:5] 0.979 0.899 0.764 0.678 0.676 ...
##  $ Y.offset    : num [1:10] 0.2821 -0.1201 0.0626 -0.1167 -0.2071 ...
##  $ X.offset    : num [1:50] -0.03591 0.00857 -0.18562 -0.15828 -0.01975 ...
##  $ lambda.vec  : num [1:10] 9.99 5.99 3.59 2.15 1.29 ...
##  $ tau.vec     : num [1:4] 1000 100 10 1
##  $ tau.min     : num 1
##  $ lam.min     : num 1.29
##  - attr(*, "class")= chr "EIVMR"
```

```
levelplot(apply(fit$err.pred, c(1,2), mean), col.regions=grey(100:0/100), xlab=expression(lambda), ylab=
```



4
3
2
1
τ
2
4
6
8
10
λ
9
8
7
6
5
4