Homework 1 Solutions

Alexander McLain

1. **(15 points)** Suppose $\boldsymbol{A}$ and $\boldsymbol{B}$ are symmetric $J \times J$ matricies. Suppose $\boldsymbol{A}$ and $\boldsymbol{B}$ have eigen values $\{\lambda_j(\boldsymbol{A})\}$ and $\{\lambda_j(\boldsymbol{B})\}$. The Hoffman-Wielandt Theorem (Hoffman and Wielandt, 1953) states that

$$\sum_{j=1}^{J}\{\lambda_j(\boldsymbol{A}) - \lambda_j(\boldsymbol{B})\}^2 \leq tr\{(\boldsymbol{A}-\boldsymbol{B})(\boldsymbol{A}-\boldsymbol{B})^\top\}$$

Prove this result. Hint: Use the spectral decomposition theorem on $\boldsymbol{A}$ and on $\boldsymbol{B}$; express $tr\{(\boldsymbol{A}-\boldsymbol{B})(\boldsymbol{A}-\boldsymbol{B})^\top\}$ in terms of the decomposition matrices of $\boldsymbol{A}$ and $\boldsymbol{B}$, and simplify; then, show that the result is minimized by $\sum_{j=1}^{J}\{\lambda_j(\boldsymbol{A}) - \lambda_j(\boldsymbol{B})\}^2$.

Solution

By the SVD we can suppose that $A = P\Lambda_A P^\top$ and $B = Q\Lambda_B Q^\top$ where P and Q are orthogonal matrices, Λ_A is a diagonal matrix with values $\{\lambda_j(A)\}$, and Λ_B is a diagonal matrix with values $\{\lambda_j(B)\}$.

$$\begin{aligned}
\operatorname{tr}\left\{(A-B)(A-B)^\top\right\} &= \operatorname{tr}\left\{(A-B)^\top(A-B)\right\} \\
&= \operatorname{tr}\left\{\left(P\Lambda_A P^\top - Q\Lambda_B Q^\top\right)^\top \left(P\Lambda_A P^\top - Q\Lambda_B Q^\top\right)\right\} \\
&= \operatorname{tr}\left\{P\Lambda_A^2 P^\top + Q\Lambda_B^2 Q^\top - 2P\Lambda_A P^\top Q\Lambda_B Q^\top\right\} \\
&= \operatorname{tr}\left\{P\Lambda_A^2 P^\top\right\} + \operatorname{tr}\left\{Q\Lambda_B^2 Q^\top\right\} - 2\operatorname{tr}\left\{\Lambda_A\Lambda_B\right\} \quad (\text{ since } \operatorname{tr}(AB) = \operatorname{tr}(BA)) \\
&= \Lambda_A^2 + \Lambda_B^2 - 2\Lambda_A\Lambda_B \\
&= (\Lambda_A - \Lambda_B)^\top(\Lambda_A - \Lambda_B) \\
&= \sum_{j=1}^{J}\{\lambda_j(A) - \lambda_j(B)\}^2
\end{aligned}$$

2. Below is the code contained in the question:

```r
set.seed(1234)
n <- 2000
p <- 10

A <- matrix(runif(n^2)*2-1, ncol=n)
Z <- matrix(rnorm(n*p),n,p)

f1 <- function(A,Z){ t(Z) %*% (t(A)%*%A) %*% Z }

f2 <- function(A,Z){
  Sigma <- (t(A)%*%A)
  temp1 <- t(Z) %*% Sigma
  temp1 %*% Z
  }
```

1a. **(10 points)**

```
library(bench)
mark( f1(A,Z), f2(A,Z) , min_iterations = 10)
```

expression	min	median	itr/sec	mem_alloc	gc/sec
f1(A, Z)	5.63s	5.67s	0.1764632	61.3MB	0.8823161
f2(A, Z)	5.73s	5.79s	0.1722299	61.3MB	0.2583449

For my computer, f1 and f2 took approximately the same amount of time. The memory allocation is identical.

1b. **(10 points)** The RcppArmadillo function "Question_1.cpp" is attached

Loading it in

```
library(Rcpp)
library(RcppArmadillo)
sourceCpp("Question_1.cpp")
```

Lets test to see if they give the same results:

```
r_ans <- f1(A,Z)
rcpp_ans <- Question_1(A,Z)$D
table( round(r_ans,6) == round(rcpp_ans,6) )
```

TRUE
100

(There will be some differences due to rounding error, which is why I use the *round* function)

1c. **(10 points)**

```
mark( f1(A,Z), f2(A,Z), Question_1(A,Z)$D , min_iterations = 10)
```

expression	min	median	itr/sec	mem_alloc	gc/sec
f1(A, Z)	5.84s	5.88s	0.1681111	61.34MB	0.3922592
f2(A, Z)	5.61s	5.64s	0.1772278	61.34MB	0.2658417
Question_1(A, Z)$D	173.45ms	174.28ms	5.7192685	9.55KB	0.0000000

On my computer, the RcppArmadillo function was over 30 times faster than f1 and f2. Also, the amount of memory that was used was 5,000 times less! This indicates that if we were to change n or p the difference would get even bigger.

1d. **(5 point bonus)** The RcppEigen function "Question_1_eigen.cpp" is attached.

Below I run in the function and test if it gives the same result:

```
library(RcppEigen)
sourceCpp("Question_1_eigen.cpp")
rcpp_eigen_ans <- Question_1_eigen(A,Z)
table( round(r_ans,6) == round(rcpp_eigen_ans,6) )
```

TRUE
100

Now I'll test it against the RcppArmadillo:

```
mark( Question_1(A,Z)$D, Question_1_eigen(A,Z) , min_iterations = 10)
```

expression	min	median	itr/sec	mem_alloc	gc/sec
Question_1(A, Z)$D	178.1ms	180.1ms	5.462839	3.32KB	0
Question_1_eigen(A, Z)	17.5ms	17.8ms	56.359123	7.45KB	0

From this we can see that Eigen works around 10 times faster than Armadillo! However, Eigen comes with a cavet that it can cripple your computer while using it. For example, when using Armadillo I can easily run 24 simulations at the same time. While using Eigen I can only run 2. This happens because Eigen has a way of using much more of your CPU drive.

2a.**(7 points)**

```
base_dat <- read.csv("baseball.csv",as.is=TRUE)
base_df <- data.frame(base_dat)
full_lm <- lm(Salary~.,data=base_df)
summary(full_lm)
```

```
##
## Call:
## lm(formula = Salary ~ ., data = base_df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1908.3  -463.0    10.9   340.7  3181.7
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   223.115    332.717   0.671 0.502970
## BatAvg       3043.192   2712.536   1.122 0.262746
## OBP         -3528.013   2376.084  -1.485 0.138581
## Runs            7.100      5.643   1.258 0.209259
## Hits           -2.698      3.312  -0.815 0.415788
## Doubles         1.368      8.611   0.159 0.873846
## Triples       -17.922     21.647  -0.828 0.408339
## HR             19.483     12.583   1.548 0.122506
## RBI            17.415      5.068   3.436 0.000668 ***
## Walks           5.815      4.523   1.285 0.199548
## SO             -9.586      2.151  -4.457 1.15e-05 ***
## SB             13.044      4.714   2.767 0.005988 **
## Err            -9.553      7.500  -1.274 0.203693
## FA           1372.886    108.594  12.642  < 2e-16 ***
## Prior_FA     -280.790    137.640  -2.040 0.042168 *
## Arb           783.592    118.289   6.624 1.48e-10 ***
## Prior_Arb     352.114    241.829   1.456 0.146361
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 694.3 on 320 degrees of freedom
## Multiple R-squared:  0.7014, Adjusted R-squared:  0.6865
## F-statistic: 46.99 on 16 and 320 DF,  p-value: < 2.2e-16
```

From the output we can see that the estimated $\hat{MSE} = \hat{\sigma}^2 = \frac{1}{n-p}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$ is $694.3^2 = 482031.8$. The mean squared error, i.e., the average squared error $\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$, is

```
MSE <- mean((full_lm$residuals)^2)
MSE
```

```
## [1] 457715.6
```

2b. **(7 points)** Neither, definition of MSE is a good estimate of the EPE. Here $M\hat{S}E = \hat{\sigma}^2$ is not good since the EPE combines residual error, i.e., the σ^2 from the linear regression model, and the model error. This MSE estimates σ^2, thus it's an estimate of the EPE assuming the model is perfectly right, which we know isn't true. The "other" MSE, i.e., $\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$, uses the same data to estimate the model and estimate the prediction error (so called "double dipping"), which will result in bias.

2c-d. **(16 points total)** How you order the variables in terms of what's "best" will differ. For simplicity, here I just use the significance tests to order the varibles. This yields the following list of the top 8 (in order): FA, Arb, SO, RBI, SB, Prior_FA, HR, and OBP.

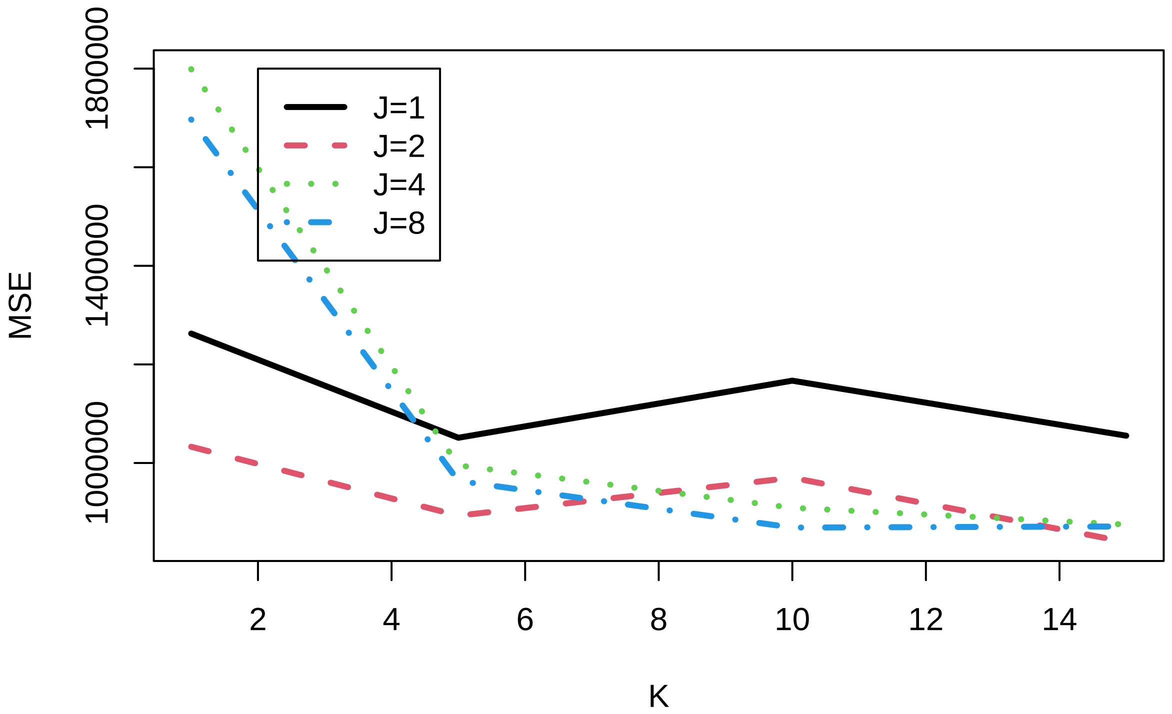
```
library(FNN)
K <- c(1,5,10,15)
J <- c(1, 2, 4, 8)
best_8 <- subset(base_df,select=c("FA", "Arb", "SO", "RBI", "SB", "Prior_FA", "HR","OBP","Salary"))
MSE <- matrix(0,4,4)
i <- 0
for(k in K){
i <- i+1
l <- 0
for(j in J){
l <- l+1
  MSE[l,i] <- mean(knn.reg(train=best_8[,1:j],y=best_8$Salary,test=NULL,k=k)$residuals^2)
}
}
colnames(MSE) <- K
row.names(MSE) <- J
MSE
```

	1	5	10	15
1	1262568	1051166.2	1166994.0	1055321.6
2	1032929	893166.8	969995.5	839553.9
4	1798868	995099.0	909079.4	875059.4
8	1696772	963614.9	869019.0	871262.7

2e. **(6 points)** Again there are many ways to do this plot.

Here is a (rather) simple way:

```
matplot(K,t(MSE),type = "l",ylab = "MSE",lwd=3)
legend(2,1800000,c("J=1","J=2","J=4","J=8"),lty = 1:4,col=1:4,lwd=3)
```



2f. **(5 points)** Varied answers acceptable. Overall, $J = 1$ performs the worst and $J = 2$ probably performs the best but 4 and 8 are close when $K > 1$. However, since these results are based on using the same training and test data they don't mean too much. This is why overfitting (i.e., using $J = 8$) works pretty well. Interestingly, using $K = 1$ (which would also be overfitting) performs worse then using $K = 5$ for all J. This is not as hypothesized.

If we were to estimate the EPE using test data overfitting would be appropriately penalized. I don't think we would see $J = 8$ performing as well as it does here.

2g. **(5 points)** The bias should decrease as the model gets more complex. Thus, as K decreases or J increases the bias, and hence the $bias^2$, should become smaller.

3a. **(5 points)** The form for the NN estimator given in the class notes clearly show it is a part of this family with $\gamma_i(x_0|X) = k^{-1}I\{X_i \in N_k(x_0)\}$ where $N_k(x_0)$ is an index of the k NN of x_0.

To see that the LR estimator is, note that

$$\hat{y} = \hat{f}(x_0) = x_0(X'X)^{-1}X'Y$$

Let $H(x_0) = x_0(X'X)^{-1}X'$ with elements $H(x_0) = (h_1(x_0), h_2(x_0), \ldots, h_n(x_0))$. Then

$$\hat{f}(x_0) = \sum_{i=1}^{n} h_i(x_0)y_i$$

which is part of the family where $\gamma_i(x_0|X) = h_i(x_0)$.

3b-c. **(10 points total)** These calculations are ones we have gone over in class. The only difference here is that in 3c the X's are random (along with the Y). For this case, we can use the law of total expectation to get:

$$E_{XY}\{(f(x_0) - \hat{f}(x_0))^2\} = E_X\left[E_{Y|X}\{(f(x_0) - \hat{f}(x_0))^2\}\right]$$

You can see that the inner part of the right hand side is the same as was done in class. Thus we have

$$E_X\left[E_{Y|X}\{(f(x_0)-\hat{f}(x_0))^2\}\right] = E_X\left[bias^2\{f(x_0)\}+Var\{f(x_0)\}\right] = \int \left[bias^2\{f(x)\}+Var\{f(x)\}\right] dF_X(x)$$

where $F_X(x)$ is the distribution function of X.

(5 points) In c) we have a similar term as b), but we take the expectation over X. This makes this a more meaningful measure of the expected error of the model.