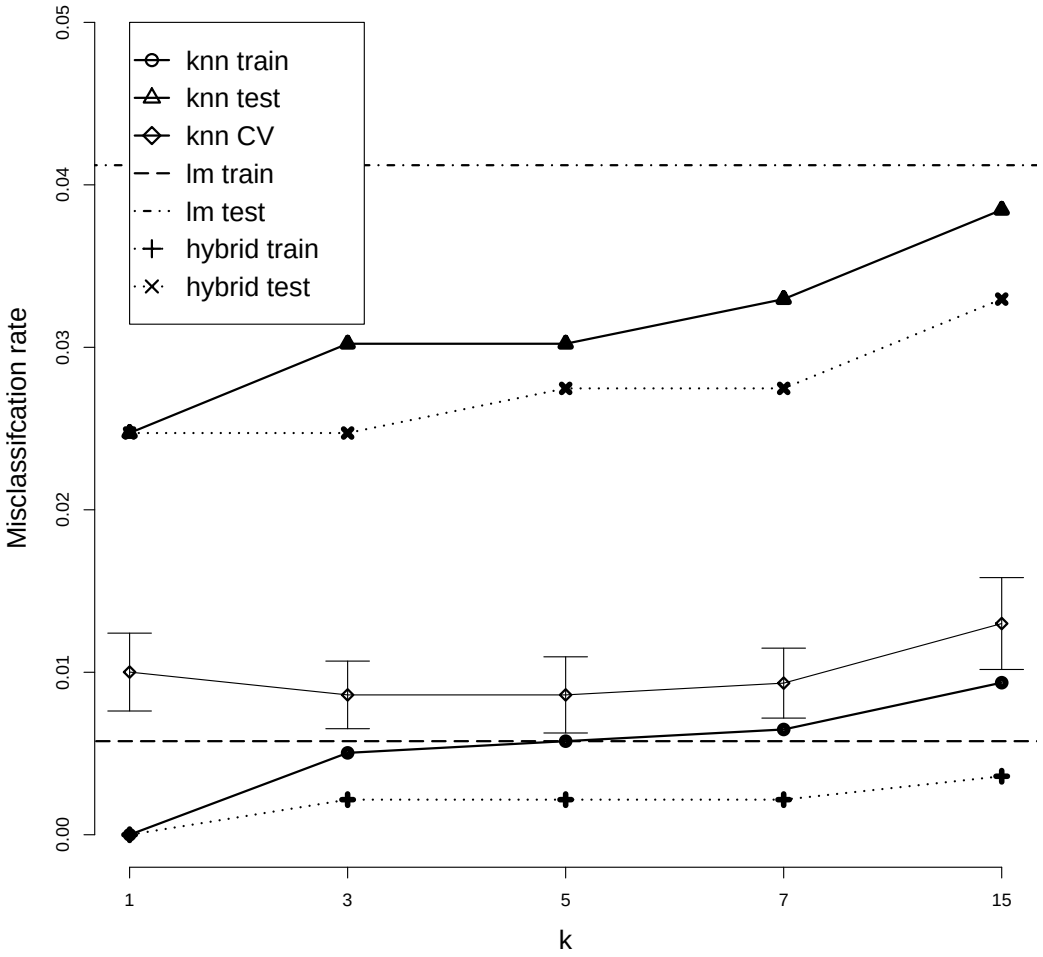
Problem Set 1 Solutions

Statistics 315A

Problem 1

(a)

Both the k-nearest neighbor (KNN) and linear regression classifiers are implemented in the code included in the appendix. The performance of the methods, along with the performance of the hybrid method described below, are shown in the following figure.



The value of k was selected by 5-fold cross-validation. The results are shown in the previous figure. An apparent minimum was reached (by an approximate tie) at $k = 3$ and $k = 5$ and so $k = 5$ was selected. One could also consider opting for a "simpler" model with higher k because of the significant uncertainty in the cross-validation estimates - perhaps using the "one standard error rule".

(b)

Examples of digits misclassified by each method and by both methods are shown in Figure 1. Indeed, these are difficult cases and many resemble both a "2" and a "3".

(c)

There are many heuristics that can be used to combine the methods. For example, try using KNN and if the neighbors do not unanimously agree then use linear regression. This method was implemented and shows a modest improvement over KNN in the first figure.

Code

```
p1 <- function(train_mat,test_mat){

#Compute k-nn cross-validation errors
kvec = c(1,3,5,7,15)
#Define K for K-fold CV
K = 10
cv = cross_validate_knn(K,kvec,train_mat[,-1],train_mat[,1])
print(dim(cv))
cv_mean = apply(cv,2,mean)
cv_sd = apply(cv,2,sd)/sqrt(K)
#Compute k-nn and hybrid train/test error
knn_test_err = rep(0,length(kvec))
knn_train_err = rep(0,length(kvec))
hybrid_train_err = rep(0,length(kvec))
hybrid_test_err = rep(0,length(kvec))

#Loop over different values of k and compute errors
for (i in seq(length(kvec))){
knn_pred = knn(kvec[i],train_mat[,-1],test_mat[,-1],train_mat[,1])
knn_test_err[i] = mean(knn_pred != test_mat[,1])
if (kvec[i]==5)
knn_missed = which(knn_pred != test_mat[,1])
knn_train_err[i] = mean(knn(kvec[i],train_mat[,-1],train_mat[,-1],train_mat[,1]) != train_mat[
hybrid_pred_train = hybrid(kvec[i],train_mat[,-1],train_mat[,-1],train_mat[,1])
hybrid_train_err[i] = mean(hybrid_pred_train!=train_mat[,1])
hybrid_pred_test = hybrid(kvec[i],train_mat[,-1],test_mat[,-1],train_mat[,1])
hybrid_test_err[i] = mean(hybrid_pred_test!=test_mat[,1])
```

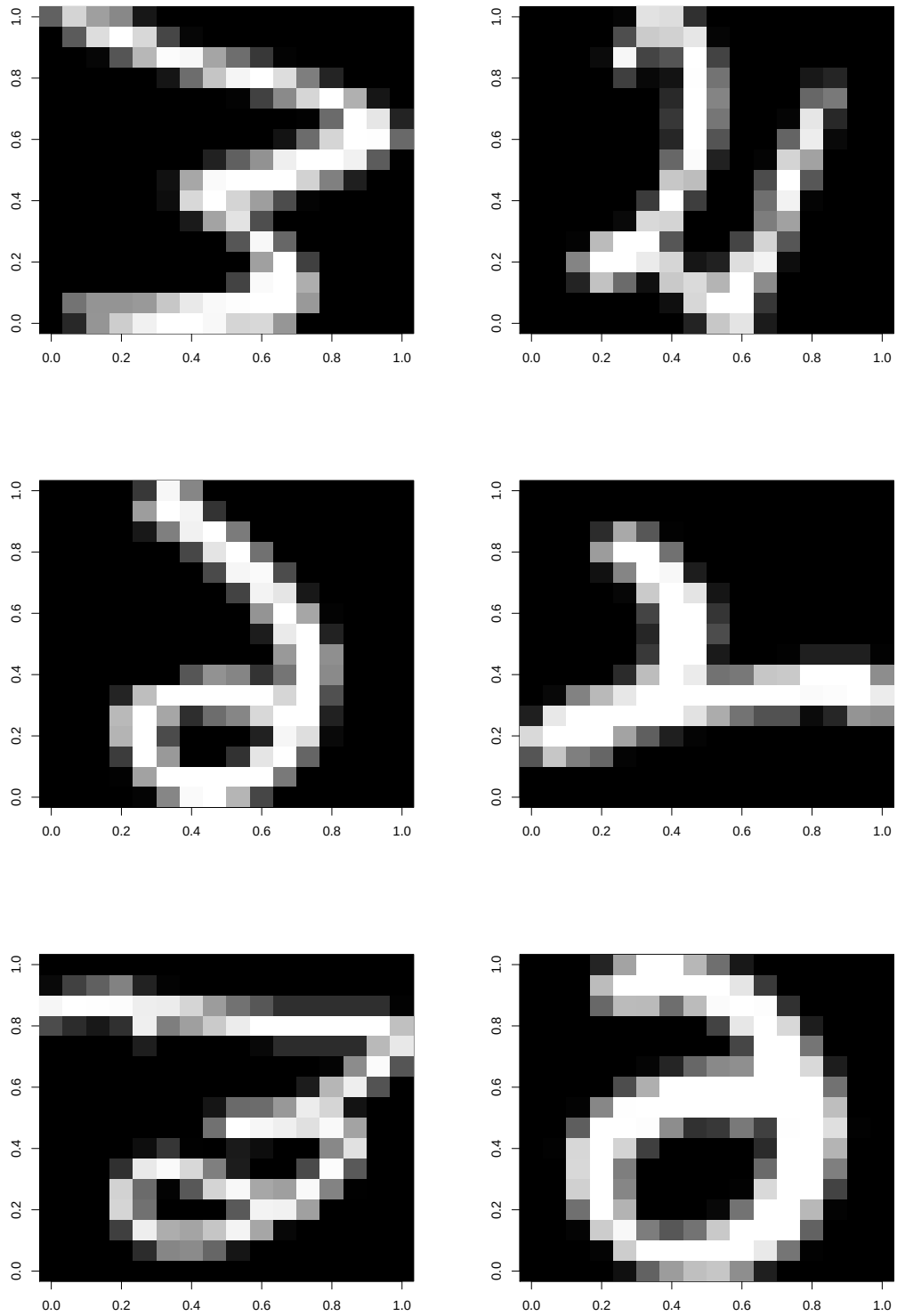


Figure 1: Examples misclassified by linear regression (61 and 148, top row), K–NN (54 and 161, middle row), and both methods (27 and 82, bottom row)

```
}



#Compute linear regression
m = lm(train_mat[,1]~train_mat[,-1])
pred_train = round(m$fitted)
pred_train[pred_train>2.5]=3
pred_train[pred_train<=2.5]=2
# A better way to write this is
# pred_train = sapply(round(m$fitted), function(x) min(max(x,2),3))
train_err = mean(pred_train!=train_mat[,1])

beta = m$coefficients
pred_test = beta[1] + test_mat[,-1]%*%beta[-1]
pred_test[pred_test>2.5]=3
pred_test[pred_test<=2.5]=2
test_err = mean(pred_test!=test_mat[,1])
lm_missed = which(pred_test!=test_mat[,1])


#Print missed cases
print(knn_missed)
print(lm_missed)

    #Plot results
    plot(seq(length(kvec)),knn_test_err,ylim=c(0,.05),axes=FALSE,
    xlab="k",ylab="Misclassifcation rate",cex.lab=1.5,cex.axis=1.5)
    print(length(kvec))
print(length(cv_mean))
points(seq(length(kvec)),cv_mean,lwd=2,pch=5)
lines(seq(length(kvec)),cv_mean,lwd=1)

plot_err_bar(seq(length(kvec)),cv_mean,.1,cv_sd)

axis(1,at=c(1,2,3,4,5),labels=c("1","3","5","7","15"))
axis(2)
points(seq(length(kvec)),knn_train_err,lwd=5,pch=1)
lines(seq(length(kvec)),knn_train_err,lwd=2,pch=1)
points(seq(length(kvec)),knn_test_err,lwd=5,pch=2)
lines(seq(length(kvec)),knn_test_err,lwd=2,pch=2)
points(seq(length(kvec)),hybrid_train_err,lwd=5,pch=3)
lines(seq(length(kvec)),hybrid_train_err,lwd=2,lty=3)
points(seq(length(kvec)),hybrid_test_err,lwd=5,pch=4)
lines(seq(length(kvec)),hybrid_test_err,lwd=2,lty=3)
```

```
lines(c(0,6),rep(train_err,2),lwd=2,lty=2)
lines(c(0,6),rep(test_err,2),lwd=2,lty=3)

 legend(1,0.05,c("knn train","knn test","knn CV","lm train","lm test","hybrid train","hybrid te
 pch=c(1,2,5,-1,-1,3,4),lty=c(1,1,1,3,4,3,3),cex=1.5)

}

plot_examples <-function(test,indices=c(1,2,3)){
for (i in seq(length(indices))){
image(matrix(test[indices[i],],16)[,16:1],col=gray(seq(0,1,len=256)))
}
}

plot_err_bar <- function(xlocations,ylocations,widths,heights){
 #Plot error bars at locations, with given widths and heights
 segments(xlocations,ylocations+heights,xlocations,ylocations-heights)
 segments(xlocations-widths,ylocations+heights,xlocations+widths,ylocations+heights)
 segments(xlocations-widths,ylocations-heights,xlocations+widths,ylocations-heights)
}


filter_data <- function(train_mat,test_mat){
#Select only 2's and 3's
train_ind = which(train_mat[,1]==2 | train_mat[,1]==3)
test_ind = which(test_mat[,1]==2 | test_mat[,1]==3)
return(list(as.matrix(train_mat[train_ind,]),as.matrix(test_mat[test_ind,])))
}

hybrid <- function(k,train,test,ytrain){
#Implement hybrid method described in assignment
knn_pred = knn(k,train,test,ytrain,round=FALSE)
ind = which(round(knn_pred)!=knn_pred)

m = lm(ytrain~train)
beta = m$coefficients
pred_lm = beta[1] + test%*%beta[-1]
pred_lm[pred_lm>2.5]=3
pred_lm[pred_lm<=2.5]=2

knn_pred[ind] = pred_lm[ind]
return(knn_pred)
}
```

```
knn <- function(k=5,train,test,ytrain,round=TRUE){
#Apply knn classification with k neighbors
n_train = dim(train)[1]
    n_test = dim(test)[1]
prediction = rep(0,n_test)

#Compute distances using (a-b)**2 = a**2 + b**2 - 2*a*b
#Note - a better implementation could precompute these since
#most distances are reused during CV but this function is more
#general
inner_products = train %*% t(test)
tr_squares = apply(train**2,1,sum)
te_squares = apply(test**2,1,sum)
distances = apply(-2*inner_products,2,'+',tr_squares)
distances = apply(distances,1,'+',te_squares)
for (i in seq(n_test)){
closest = order(distances[i,])[seq(k)]
if (round){
prediction[i] = round(mean(ytrain[closest]))
}
else{
prediction[i] = mean(ytrain[closest])
}
}
return(prediction)
}



cross_validate_knn <- function(K=10,kvec,train,ytrain){
#Perform K-fold cross-validation
n = dim(train)[1]
num_in_folds = floor(n/K)
#Define indices for folds
shuffle = sample(c(rep(seq(K-1),num_in_folds),rep(K,n-num_in_folds*(K-1))))
error = matrix(rep(0,K*length(kvec)),nrow=K)
for (i in seq(K)){
ind = which(shuffle==i)
for (j in seq(length(kvec))){
pred = knn(kvec[j],train[-ind,],train[ind,],ytrain[-ind])
error[i,j] = mean(pred!=ytrain[ind])
}
}
return(error)
}
```

Problem 2

Both approaches are based on the fact that $\hat{\beta} \sim N(\beta, \sigma^2(\mathbf{X}^T\mathbf{X})^{-1})$ (Equation (3.10)). The conceptual difference between the two methods' confidence intervals (denote them by CI_i^a and CI_{ii}^a) is the following:

$$\begin{aligned} P(a^T\beta \in CI_i^a) \quad &\geq 1-\alpha \qquad \forall a \\ P(a^T\beta \in CI_{ii}^a \ \forall a) &\geq 1-\alpha. \end{aligned}$$

We would expect CI_{ii}^a to be wider since its notion of coverage is simultaneous for *all* a.

Method i

By definition of the multivariate normal distribution, we know that $a^T\hat{\beta}$ is normal. Also,

$$\begin{aligned} \mathbb{E}[a^T\hat{\beta}] &= a^T\beta \\ \mathbb{V}ar[a^T\hat{\beta}] &= a^T\mathbb{C}ov[\hat{\beta}]a = \sigma^2 a^T(\mathbf{X}^T\mathbf{X})^{-1}a. \end{aligned}$$

So $a^T\hat{\beta} \sim N(a^T\beta, \sigma^2 a^T(\mathbf{X}^T\mathbf{X})^{-1}a)$. It follows that

$$P(-z_{1-\alpha/2} \leq \frac{a^T\hat{\beta} - a^T\beta}{\sigma\sqrt{a^T(\mathbf{X}^T\mathbf{X})^{-1}a}} \leq z_{1-\alpha/2}) = 1-\alpha.$$

Thus, a 95% confidence interval at the point a is given by

$$CI_i^a = \left[a^T\hat{\beta} - z_{0.975}\sigma\sqrt{a^T(\mathbf{X}^T\mathbf{X})^{-1}a}, \quad a^T\hat{\beta} + z_{0.975}\sigma\sqrt{a^T(\mathbf{X}^T\mathbf{X})^{-1}a}\right]$$

if σ is known, or else by

$$CI_i^a = \left[a^T\hat{\beta} - t_{n-4}^{0.975}\hat{\sigma}\sqrt{a^T(\mathbf{X}^T\mathbf{X})^{-1}a}, \quad a^T\hat{\beta} + t_{n-4}^{0.975}\hat{\sigma}\sqrt{a^T(\mathbf{X}^T\mathbf{X})^{-1}a}\right].$$

Method ii

In method 2, we begin with the confidence set for the vector $\hat{\beta}$ given in Equation (3.15):

$$C_\beta = \left\{\beta : (\hat{\beta}-\beta)^T\mathbf{X}^T\mathbf{X}(\hat{\beta}-\beta) \leq \sigma^2\chi_4^2(1-0.05)\right\}.$$

This follows because

$$(\hat{\beta}-\beta)^T\mathbf{X}^T\mathbf{X}(\hat{\beta}-\beta) \sim \sigma^2\chi_4^2.$$

If σ^2 is unknown, then technically it should be

$$C_\beta = \left\{\beta : (\hat{\beta}-\beta)^T\mathbf{X}^T\mathbf{X}(\hat{\beta}-\beta) \leq \hat{\sigma}^2 F_{4,n-4}(1-0.05)\right\},$$

but for large n, this makes little difference. Now, $CI_{ii}^a = \{a^T\beta : \beta \in C_\beta\}$, so

$$0.95 = P(\beta \in C_\beta) \leq P(a^T\beta \in CI_{ii}^a \ \forall a)$$

Geometrically, C_β is an ellipse in p-dimensional space, and CI_{ii}^a is the projection of this ellipse onto the subspace spanned by a. To find the endpoints of the interval CI_{ii}^a, we wish to solve

$$\min_\beta / \max_\beta a^T\beta \text{ subject to } (\hat\beta-\beta)^T\mathbf{X}^T\mathbf{X}(\hat\beta-\beta) \le \sigma^2\chi_4^2(0.95).$$

By recentering and sphering, we may transform this to a simpler problem: Let $z = (\mathbf{X}^T\mathbf{X})^{1/2}(\beta - \hat\beta)$. Then, noting that $\beta = \hat\beta + (\mathbf{X}^T\mathbf{X})^{-1/2}z$, we rewrite the problem as

$$\min_z / \max_z a^T(\hat\beta + (\mathbf{X}^T\mathbf{X})^{-1/2}z) \text{ subject to } ||z||^2 \le \sigma^2\chi_4^2(0.95).$$

By Cauchy-Schwarz, this is minimized/maximized at $z = \pm c(\mathbf{X}^T\mathbf{X})^{-1/2}a$, where $c = \frac{\sigma\chi_4^2(0.95)}{||(\mathbf{X}^T\mathbf{X})^{-1/2}a||}$. Therefore, the endpoints are given by

$$\begin{aligned} a^T(\hat\beta \pm (\mathbf{X}^T\mathbf{X})^{-1/2}c(\mathbf{X}^T\mathbf{X})^{-1/2}a)) &= a^T\hat\beta \pm \frac{\sigma\sqrt{\chi_4^2(0.95)}a^T(\mathbf{X}^T\mathbf{X})^{-1}a}{\sqrt{a^T(\mathbf{X}^T\mathbf{X})^{-1}a}} \\ &= a^T\hat\beta \pm \sigma\sqrt{\chi_4^2(0.95)a^T(\mathbf{X}^T\mathbf{X})^{-1}a} \end{aligned}$$

Thus, we see that the bands of this method are

$$|CI_{ii}^a|/|CI_i^a| = \sqrt{\chi_4^2(0.95)}/z_{.975} = 1.57$$

times wider than those of the pointwise intervals. This is the price paid for simultaneity. Figure 2 shows an example of the two confidence bands.

We have presented here an analytical approach to this problem. One could also have "simulated" the Method 2 confidence bands by—for example—generating a large number of β-vectors on the boundary of C_β and then for each a, finding the maximum/minimum values of $a^T\beta$ observed.

R code for Problem 2:

```
set.seed(123)
n=75
x = sort(rnorm(n))
beta = rnorm(4)
X = cbind(1,poly(x,3,raw=T))
y = X%*%beta + 4*rnorm(n)

bhat = solve(t(X)%*%X,t(X))%*%y
yhat = X%*%bhat
sighat = sqrt(sum((y-yhat)^2)/(n-4))
cov.bhat = solve(t(X)%*%X)*sighat^2
z = qnorm(1-0.025)

# Method 1
width = z *  sqrt(diag(X%*%cov.bhat%*%t(X)))
lower = yhat - width
upper = yhat + width
```

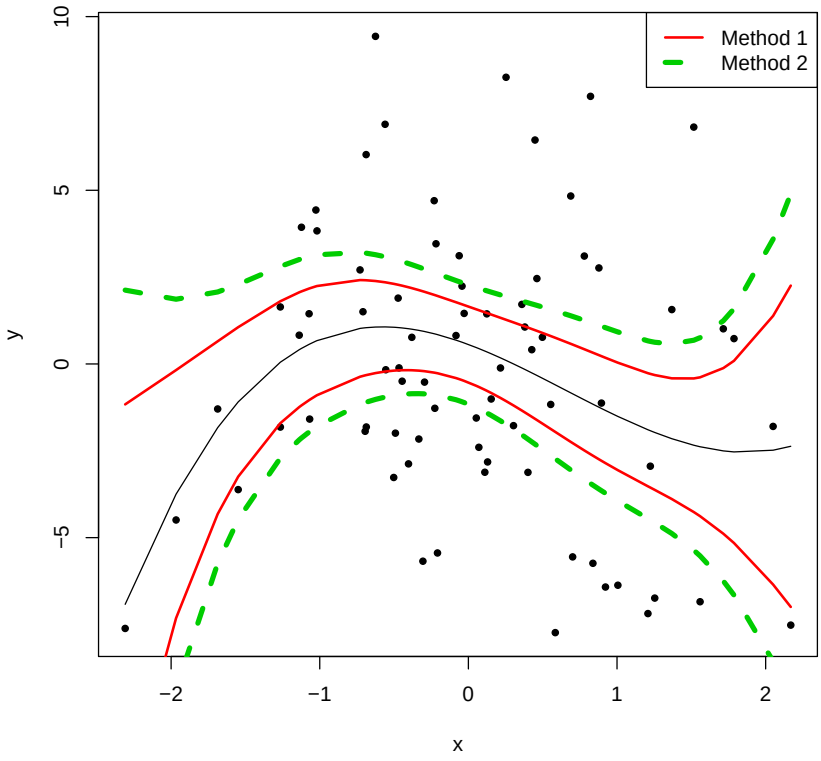


Figure 2: Problem 2

```
pdf("p2.pdf")
plot(x,y,pch=20)
lines(x,yhat)
lines(x,lower,lwd=2,col=2)
lines(x,upper,lwd=2,col=2)

# Method 2
qchi2 = qchisq(.95,df=4)
width = sqrt(qchi2 * diag(X%*%cov.bhat%*%t(X)))
lower = yhat - width
upper = yhat + width

lines(x,lower,lty=2,lwd=4,col=3)
lines(x,upper,lty=2,lwd=4,col=3)
legend("topright",legend = c("Method 1","Method 2"),col=2:3,
       lty=1:2,lwd=c(2,4))
dev.off()
```

Problem 3

First note

$$E(R_{te}(\hat{\beta})) \quad = \quad \frac{1}{M}\sum_{i=1}^{M} E((\tilde{y}_i - \hat{\beta}\tilde{x}_i)^2) = \frac{1}{N}\sum_{i=1}^{N} E((\tilde{y}_i - \hat{\beta}\tilde{x}_i)^2)$$

because $\hat{\beta}$ has no dependence on $(\tilde{x}_i, \tilde{y}_i)$ or M. Thus, without loss of generality assume that $M = N$. Then let $\tilde{\beta}$ be the least squares estimate of β based on the test data such that by symmetry

$$E(R_{te}(\hat{\beta})) = E(R_{tr}(\tilde{\beta})).$$

By construction $E(R_{tr}(\hat{\beta})) \leq E(R_{tr}(\tilde{\beta}))$ and combined with the previous equation this gives the desired result. Many students asserted that

$$E(R_{tr}(\hat{\beta})) = E(R_{te}(\tilde{\beta}))$$

but this is only true when $M = N$ since the usual regression arguments give

$$E(R_{tr}(\hat{\beta})) = \frac{N-p}{N}\sigma^2$$

which depends on N.

Problem 4

Since linear combination of Normal random variables is still normal. We know $a^T z$ follows a normal distribution and we only need to work out its mean and variance. Note that

$$\mathbb{E}[a^T z] = a^T \mathbb{E}[z] = 0 \tag{1}$$

$$\mathbb{V}ar[a^T z] = a^T \mathbb{C}ov(z) a = ||a||^2 = 1 \tag{2}$$

Therefore $a^T z \sim N(0,1)$ and $\mathbb{E}[(a^T z)^2] = 1$. Since $\mathbb{E}[||z||^2] = p$, we clearly see that the prediction point is away from the projections of the data.

Problem 5

(a)

Since $p \gg N$, *except for those cases where the system* $(X^T X)\hat{\beta} = X^T Y$ *is inconsistent*, the system will have more variables than the number of equations, which implies infinit number of solutions. The solutions form a linear space. For any $\hat{\beta}$, the residuals will be all zero.

(b)

$X^T X$ is clearly positive semi-definite. Therefore as long as $\lambda > 0$, the matrix $X^T X + \lambda I$ is positive definite, which implies that it is invertible. Hence the ridge regression always have unique solution for $\lambda > 0$.

(c) &(d)

As $\lambda \to 0$, the ridge regression solution will converges to the solution in the solution space of part(a) which has the smallest 2-norm. To see that, suppose the full(fat) SVD of X is $U\Sigma V^T$. The ridge regression solution

$$\begin{aligned} \hat{\beta}_\lambda^{ridge} =& (X^T X + \lambda I)^{-1} X^T Y = (V\Sigma^T \Sigma V^T + \lambda I)^{-1} V \Sigma^T U^T Y & (3) \\ =& (V(\Sigma^T \Sigma + \lambda I)^{-1} V^T) V \Sigma^T U^T Y = V(\Sigma^T \Sigma + \lambda I)^{-1} \Sigma^T U^T Y & (4) \end{aligned}$$

Note that $(\Sigma^T \Sigma + \lambda I)^{-1} \Sigma^T$ is a p by N diagonal matrix where its ith diagnonal entry is

$$\frac{\sigma_i}{\sigma_i^2 + \lambda} \xrightarrow{\lambda \to 0} \begin{cases} \frac{1}{\sigma_i} & \sigma_i > 0 \\ 0 & \sigma_i = 0 \end{cases} \tag{5}$$

Note that in (5) *the limit cannot be calculated by just input* $\lambda = 0$ *if some* $\sigma_i = 0$. *But the limit still exists*

The limit can also be seen using the generalized inverse of Σ, denoted as Σ^- Therefore we have

$$\hat{\beta}_\lambda^{ridge} \xrightarrow{\lambda \to 0} V \Sigma^- U^T Y \tag{6}$$

The solution could also be derived using the reduced SVD of X. If the reduced SVD of X is $X = U\Sigma V^T$ where now U and V is no longer orthogonal matrix but Σ is now N by N diagonal matrix. We have

$$\hat{\beta}_\lambda^{ridge} \xrightarrow{\lambda \to 0} V \Sigma^{-1} U^T Y \tag{7}$$

Problem 6: All-subsets regression.

The following are the 21 coefficients (intercept included) generated for the underlying model $Y = X^T\beta + \epsilon$ (Note that it's acceptable to either include or exclude the intercept in the generating model):

```
-1.4341  1.3120  1.2430 -1.2044 -0.9844  0.7913  0.6248 -0.4941 -0.4835
0.4493 -0.3621 -0.2932  0.2650  0.2336 -0.2216 -0.1532 -0.1520 -0.1174
0.0717  0.0099  0.0093
```

To generate the ith observation $(x_1^{(i)}, x_2^{(i)}, ..., x_{20}^{(i)}; y^{(i)})$, each $x_t^{(i)} \overset{i.i.d}{\sim} N(0,1)$ and $\epsilon^{(i)} \sim N(0, \beta^T\beta)$ independently to ensure the signal to noise variance ratio to be approximately 1.

Fixing k, for each of the 30 training sets generated, we have an estimated regression function $\hat{f}_j$ (based on either the best subset or simply the first k variables). The squared prediction error (SPE), the bias^2 and the variance can be computed as follows:

$$SPE = \frac{1}{10000}\frac{1}{30}\Sigma_{i=1}^{10000}\Sigma_{j=1}^{30}(y^{(i)} - \hat{f}_j(\mathbf{x}^{(i)}))^2$$

$$Bias^2 = \frac{1}{10000}\Sigma_{i=1}^{10000}(f(\mathbf{x}^{(i)}) - \bar{\hat{f}}(\mathbf{x}^{(i)}))^2$$

$$Var = \frac{1}{10000}\frac{1}{30}\Sigma_{i=1}^{10000}\Sigma_{j=1}^{30}(\hat{f}_j(\mathbf{x}^{(i)}) - \bar{\hat{f}}(\mathbf{x}^{(i)}))^2$$

where

$$\bar{\hat{f}}(\mathbf{x}) = \frac{1}{30}\Sigma_{j=1}^{30}\hat{f}_j(\mathbf{x})$$

and $(\mathbf{x}^{(i)}, y^{(i)})$ are from the test set of size 10000.

In Figure 3, we include the SPE, bias^2 and variance curves as a function of k, for both best subset procedure and the simple model using only the first k variables. Interestingly, the simple model has a lower SPE and variance for all $k = 1$ through 19. The simple model has bigger bias at lower k, probably because the best subset model may give different best subsets of variables for different training set and the bias from the 30 trained models cancel each other. At around $k = 11$, the simple model starts to have a small bias as well because the variables are sorted by their coefficients. Overall, the best subset model seems to perform worse. The best subset procedure searches through a model space that is too big and it is quite possible that some subset of variables far from the true model happens to win under training, but does not have much explanatory power with the test data.

Another interesting observation is that both procedures share similar patterns in all three plots. Again, the "dip" in SPE curves reflects the bias and variance trade off, as SPE can be decomposed into:

$$\text{Expected PE} = \sigma^2 + \text{Variance} + \text{Bias}^2.$$

Bias decreases as the model gets more and more comprehensive with larger k. Indeed, when all the 20 variables are included (as in the true model), the bias approaches zero. On the other hand, variance gets bigger as more variables are included.

The fact that the simple model with the first k variables wins big here is largely because we pick the first k variables with the biggest "true" coefficients. In other words, the most influential

variables get to be included first. However, we should keep in mind that this is cheating in the sense that in practice we wouldn't have known the true ordering

The R code for this problem is included below:

```
### Problem 6 All subset regression

#Generate beta and test set, one for intercept
set.seed(80)
library(leaps)
beta <- rnorm(21)
beta <- beta[order(abs(beta),decreasing=T)]
#fn simulates pairs of X and Y
sim <- function(n){
  X <- matrix(NA,n,21); rids <- rep(NA,n); fX <- rep(NA,n);  Y <- rep(NA,n)
  for (i in 1:n){
    X[i,] <- c(1,rnorm(20))  #include 1 for intercept
    rids[i] <- rnorm(1,mean=0,sd=sqrt(beta[-1]%*%beta[-1]))
    fX[i] <- X[i,]%*%beta
    Y[i] <- fX[i]+rids[i]
  }
  return(list(X=X[,-1],Y=Y,fX=fX))
}

data.test <- sim(10000)
#Generate 30 training sets of size  50 each
data.train <- list()
for (i in 1:30){
  data.train[[i]] <- sim(50)
}

#Compute the predicted values of the best submodel of size k, given training set
pred <- function(data.train,k){
  leap.which <- leaps(x=data.train$X,y=data.train$Y,method="r2",nbest=1)$which
  X.k <- data.train$X[,(leap.which[k,])]
  beta.hat <- lm(data.train$Y~X.k)$coefficients
  pred <-cbind(1, data.test$X[,(leap.which[k,])])%*%beta.hat
  return(pred)
}

#Using first k variables
pred2 <- function(data.train,k){
  X.k <- data.train$X[,1:k]
  beta.hat <- lm(data.train$Y~X.k)$coefficients
  pred <-cbind(1, data.test$X[,1:k])%*%beta.hat
  return(pred)
}


#Compute SPE, Bias^2 and Variance
SPE <- rep(NA,20); Bias2 <- rep(NA,20); Var <- rep(NA,20)
for (k in 1:20){
   preds <- matrix(NA,30,10000)
   for (i in 1:30){
     preds[i,] <- pred(data.train[[i]],k)
   }
   preds.bar <- colMeans(preds)
```

```
   SPE[k] <- mean((t(preds)-data.test$Y)^2)
   Bias2[k] <- mean((data.test$fX-preds.bar)^2)
   Var[k] <- mean((t(preds)-preds.bar)^2)
 }

#Compute SPE, bias^2 and variance for the simple model using first k variables
SPE2 <- rep(NA,20); Bias22 <- rep(NA,20); Var2 <- rep(NA,20)
for (k in 1:20){
   preds <- matrix(NA,30,10000)
   for (i in 1:30){
     preds[i,] <- pred2(data.train[[i]],k)
   }
   preds.bar <- colMeans(preds)
   SPE2[k] <- mean((t(preds)-data.test$Y)^2)
   Bias22[k] <- mean((data.test$fX-preds.bar)^2)
   Var2[k] <- mean((t(preds)-preds.bar)^2)
 }

postscript("fig5a.ps",height=6,width=6,horizontal=F,family='Times')
plot(SPE,xlab="k",ylab="Squared prediction error",main="Squared prediction error",
type='b',lty=2,pch=19,ylim=c(min(SPE,SPE2)-2,max(SPE,SPE2)+2),col='red',cex=0.5)
points(SPE2,type='b',lty=1,pch=19,col='blue',cex=0.5)
legend(15,max(SPE,SPE2),legend=c("best-size-k model","first-k model"),
col=c("red","blue"),lty=c(2,1),cex=0.7)
dev.off()

postscript("fig5b.ps",height=6,width=6,horizontal=F,family='Times')
plot(Bias2,xlab="k",ylab="Bias^2",main="Bias square",type='b',lty=2,
pch=19,ylim=c(min(Bias2,Bias22)-2,max(Bias2,Bias22)+2),col='red',cex=0.5)
points(Bias22,type='b',lty=1,pch=19,col='blue',cex=0.5)
legend(15,max(Bias2,Bias22),legend=c("best-size-k model","first-k model"),
col=c("red","blue"),lty=c(2,1),cex=0.7)
dev.off()

postscript("fig5c.ps",height=6,width=6,horizontal=F,family='Times')
plot(Var,xlab="k",ylab="Variance",main="Variance",type='b',lty=2,pch=19,
ylim=c(min(Var,Var2)-2,max(Var,Var2)+2),col='red',cex=0.5)
points(Var2,type='b',lty=1,pch=19,col='blue',cex=0.5)
legend(15,min(Var,Var2),legend=c("best-size-k model","first-k model"),
col=c("red","blue"),lty=c(2,1),cex=0.7)
dev.off()
```

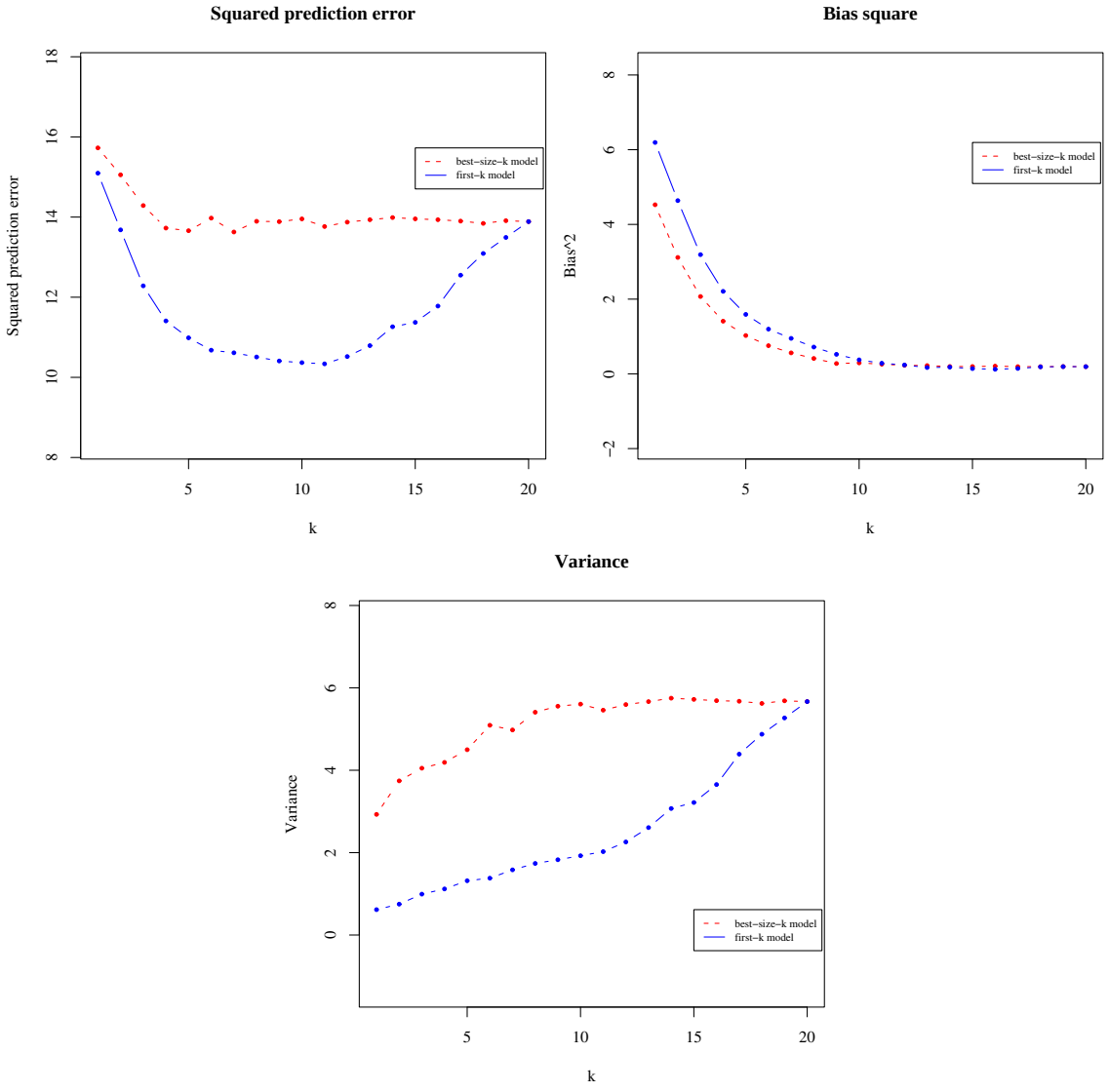


Figure 3: Squared prediction error, bias^2 and variance curves as a function of k. The red curve is based on best subset of size k, and the blue curve is based on the first k variables.