Question 1

a)  
In figure 1 we can see the error curves for KNN as well as the linear regression classification (constant w.r.t. $k$). The linear regression method has a higher test error than the K-NN method for all values of $K$. For KNN, $k = 3$ is optimal when it is chosen by doing cross-validation. However, from the test error curves we can see that $k = 1$ actually yields a lower test error.

![Error Plots](image)

Figure 1: Error curves for KNN and linear regression classification.

b)  
If figure 2 are some digits that are being misclassified by the methods. The first row contains 2 images that are misclassified by KNN, but not linear regression, the second row misclassified by linear regression, but not KNN and the third row is misclassified by both methods.

c)  
Here, several different solutions are possible. One particularly simple one is the following.

If KNN and the linear model based classification agree, use that classification. If they don’t agree we want to make the decision based on how “convinced” the model is of its prediction. For this, define the following:

- Let $c^{KNN}_i$ be the classification of the $i$-th digit with KNN and $p^{KNN}_i$ be the proportion of neighbours that agree with $c^{KNN}_i$.
Figure 2: Misclassified digits

- Let $c_i^{LM}$ be the classification of the linear model and $\hat{y}_i$ be the prediction of the linear model (response coded as 2 or 3). Then

$$p_i^{LM} = \begin{cases} 
\hat{y}_i - 2 & \text{if } c_i^{LM} = 3 \\
3 - \hat{y}_i & \text{if } c_i^{LM} = 2 
\end{cases}$$

Here, especially $p_i^{LM}$ can be larger than 1 which is not intuitive for a probability. This model is better suited for a logistic regression approach, however here we will stick to it nonetheless. Given $p_i^{LM}$ and $p_i^{KNN}$, the disagreement is decided in favour of the model with the larger probability.

The error curves for this new model compared to the old ones can be seen in figure 3. Indeed, the combined model yields lower error rates than either of the models alone.

The code for this question can be found here in the appendix.

Question 2

First, we look at the right hand side of the inequality and note

$$E(R_{te}(\hat{\beta})) = E(E(\frac{1}{M} \sum_{i=1}^{M} (\hat{y}_i - \hat{\beta}\hat{x}_i)^2 | X, y) =$$

$$= E(\frac{1}{M} \sum_{i=1}^{M} E((\hat{y}_i - \hat{\beta}\hat{x}_i)^2 | X, y)) =$$

$$= E(E((\hat{y}_1 - \hat{\beta}\hat{x}_1)^2 | X, y)) = E((\hat{y}_1 - \hat{\beta}\hat{x}_1)^2)$$

where we use that given $X, y, \hat{\beta}$ is a constant and also that all $\hat{x}_i, \hat{y}_i$ are i.i.d. Using this we see that

$$E(E(\frac{1}{M} \sum_{i=1}^{M} (\hat{y}_i - \hat{\beta}\hat{x}_i)^2 | X, y) = E(E(\frac{1}{N} \sum_{i=1}^{N} (\hat{y}_i - \hat{\beta}\hat{x}_i)^2 | X, y)$$
so that without loss of generality we can assume that \( M = N \). However, if \( M = N \), and if \( \hat{\beta} \) is the least squares estimate over the test set, then \( R_{tr}(\hat{\beta}) \) and \( R_{te}(\hat{\beta}) \) have the same distribution. Therefore

\[
E(R_{tr}(\hat{\beta})) = E(R_{te}(\hat{\beta})) \leq E(R_{te}(\tilde{\beta}))
\]

where the last inequality follows as \( \tilde{\beta} \) is the least squares estimate over the test dataset and therefore

\[
\frac{1}{M} \sum_{i=1}^{M} (\tilde{y}_i - \tilde{\beta} \tilde{x}_i)^2 \leq \frac{1}{M} \sum_{i=1}^{M} (\tilde{y}_i - \hat{\beta} \tilde{x}_i)^2
\]

**Question 3**

As noted in the question, the squared distance of any sample point from the origin has a \( \chi^2 \) distribution with mean \( p \). Therefore, the target point has expected squared distance \( p \) from the origin.

On the other hand, we know that \( ||a||^2 = 1 \) and it is independent of any \( x_i \). Therefore, as \( x_i \) is \( N(0, I_p) \), we have that \( z_i | a = a^T x_i | a \sim N(0, a^T I_p a) \). Thus

\[
E(z_i^2) = E(E(z_i^2 | a)) = E(\chi_1^2) = 1.
\]

**Question 4**

a)

When \( p \gg N \), we can solve the \( y = X\beta \) system exactly in many ways, by identifying a nonsingular subset of \( N \) of the columns of \( X \) and solving for the corresponding elements of \( \beta \), and setting the rest to zero. Because any of these ways solve for \( y \) exactly, the residuals will be zero.

[In fact, we can go further. Using the SVD, let \( X = UDV^T = ZV^T \), with the \( N \) columns of the \( p \times N \) matrix \( V \) orthogonal, and \( Z = UD \) is \( N \times N \) nonsingular. Let \( V_{\perp} (p \times (p-N)) \) ]
span the orthogonal complement to $V$ in $\mathbb{R}^p$. Let $\hat{\theta}$ solve $Z\theta = y$, i.e. $\hat{\theta} = Z^{-1}y$. Let $\hat{\beta}(\gamma) = V\hat{\theta} + V_\perp\gamma$. Then for any value of $\gamma$, $X\hat{\beta}(\gamma) = ZV^T(V\hat{\theta} + V_\perp\gamma) = Z\theta = y$. Hence for all $\gamma \in \mathbb{R}^{(p-N)}$, $\hat{\beta}(\gamma)$ has zero residuals.

b) The ridge solution is unique. The normal equations $(X^TX + \lambda I)\beta = X^Ty$ are nonsingular, so the inverse exists and defines this unique solution.

c) Here we find useful the view of ridge regression as solving the problem

$$\min ||y - X\beta||^2 \text{ subject to } ||\beta|| < s,$$

with $s$ inversely proportional to $\lambda$. As we let $s$ grow, we restrict the solution less, until eventually we touch the subspace where $r = 0$. Because of the way in which we let the solutions grow, this will be the zero-residual solution with minimum $||\beta||$.

d) It is easy to show that the ridge regression solution has the form

$$\hat{\beta}_\lambda = V(D^2 + \lambda I)^{-1}DU^Ty.$$

When $\lambda = 0$, we have $\hat{\beta}_0 = VD^{-1}U^Ty$.

[Note that this is $\hat{\beta}_0 = VZ^{-1}y = V\hat{\theta}$, which is unique. From $\hat{\beta}(\gamma) = V\theta + V_\perp\gamma$, we see that any zero-residual solution is the ridge solution plus a piece in $V_\perp$, which makes clear that ridge is the minimum-norm solution.]

Question 5

Before we can simulate data from the model, the first step is to draw $\beta$ from a $N(0, I_p)$ as stated in the question. Then we have to find the $\sigma^2$, the variance of $\varepsilon \sim N(0, \sigma^2)$ such that the signal to noise variance ratio is 1. As $E(x_i\beta) = E(E(x_i\beta|\beta)) = 0$ b/c $x_i \sim N(0, I_p)$, we get for the signal variance

$$Var(x_i\beta) = E((x_i\beta)^2) = E(E(x_i\beta^2|\beta)) = E(\beta^T\beta) = p.$$ 

Therefore, in order to have a signal to noise variance ratio of 1, we choose

$$\sigma^2 = p.$$ 

Let $y_{i,j,k}$ be the $k$-subset prediction on the $j$-th observation in the $i$-th simulation ($1 \leq k \leq p$, $1 \leq j \leq N$, $1 \leq i \leq 30$), then the prediction bias for the $k$-subset regression on the $j$-th observation is

$$\text{Bias}_{j,k} = y_{\cdot,j,k} - y_j$$

where

$$y_{\cdot,j,k} = \frac{1}{30} \sum_{i=1}^{30} y_{i,j,k}$$
and prediction variance or the $k$-subset regression on the $j$-th observation is

$$\text{Var}_{j,k} = \frac{1}{29} \sum_{i=1}^{30} (y_{i,j,k} - y_{.,j,k})^2$$

Then, the

$$\text{bias}^2_k = \frac{1}{N} \sum_{j=1}^{N} \text{Bias}^2_{j,k}$$

and

$$\text{var}_k = \frac{1}{N} \sum_{j=1}^{N} \text{Var}^2_{j,k}$$

and

$$\text{SPE}_k = \text{bias}^2_k + \text{var}_k$$

After generating a training dataset with $N = 10000$ observations, and $p = 20$ variables we make 30 simulations of the prediction process. In each simulation, we generate a training sample with $n = 50$ observations.

Then we use the function `leaps` (from a contributed package from CRAN web) to do the all subset regression and calculate the optimal combination of variables in each sample size and make predictions on the testing sample.

Let $y_{i,j,k}$ be the $k$-subset prediction on the $j$-th observation in the $i$-th simulation ($1 \leq k \leq p$, $1 \leq j \leq N$, $1 \leq i \leq 30$), then the prediction bias for the $k$-subset regression on the $j$-th observation is

$$\text{Bias}_{j,k} = y_{.,j,k} - x_j \beta$$

where

$$y_{.,j,k} = \frac{1}{30} \sum_{i=1}^{30} y_{i,j,k}$$

and prediction variance or the $k$-subset regression on the $j$-th observation is

$$\text{Var}_{j,k} = \frac{1}{29} \sum_{i=1}^{30} (y_{i,j,k} - y_{.,j,k})^2$$

Then, the

$$\text{bias}^2_k = \frac{1}{N} \sum_{j=1}^{N} \text{Bias}^2_{j,k}$$

and

$$\text{var}_k = \frac{1}{N} \sum_{j=1}^{N} \text{Var}^2_{j,k}$$

and

$$\text{SPE}_k = \frac{1}{30 \cdot N} \sum_{i,j}^{(\hat{y}_{i,j,k} - y_{i,j,k})^2}$$

In figure we can clearly see the bias-variance tradeoff. For increasing model size, the bias is decreasing for the simple model as well as the best subset model. At the same time, for increasing model size, the variance is increasing for both models. Furthermore, from the prediction error curves of both models, we can see that the best subset selection model
Figure 4: Error curves for the simple and the best-subset selection models.

Error is minimal at $k = 20$, but decreasing slowly at the end so that lower values could be chosen as well. For the simple model, the minimal prediction error is at $k = 10$, so using the 10 first and therefore best predictors.

The code used for this question can be found in the appendix.

### Appendix

#### Code for question 1:

In "hw1Funcs.R":

```r
getClassAndProb = function(pred, labels=c(2,3))
{
    class = pred
    class[pred< 1.5]=labels[1]
    class[pred>= 1.5]=labels[2]
    class = as.factor(class)

    ### infer a measure of how good the prediction is
    pred[pred>2]=2
    pred[pred<1]=1
    classProb = 1-abs(round(pred) - pred)

    return(list(class=class, prob=classProb))
}
```

The code for the questions:
library(class)
source("hw1Funcs.R")
seed(1)

training = read.table("zip.train", colClasses="numeric")
test = read.table("zip.test", colClasses="numeric")
### first column is the truth; pick out only the twos and threes
training = training[training[,1]==2 | training[,1]==3,]
test = test[test[,1]==2 | test[,1]==3,]
training[,1] = as.factor(training[,1])
test[,1] = as.factor(test[,1])

numTrain = dim(training)[1]
numTest = dim(test)[1]
numFolds = 10
permuted = sample(numTrain)
training = training[permuted,]

### part (a)
### apply the linear regression
linModForm = as.formula(paste("V1 ~ " , paste(paste("V",2:257, sep=""),collapse="+"),")))
trainingNumeric = training
trainingNumeric[,1] = as.numeric(trainingNumeric[,1])
linMod = lm(linModForm, data=trainingNumeric)
linModPred = predict(linMod, newdata=test)
linModPredTrain = predict(linMod, newdata=training)
linModClass = getClassAndProb(linModPred)$class
linModClassProb = getClassAndProb(linModPred)$prob
linModClassTrain = getClassAndProb(linModPredTrain)$class
linModClassTrainProb = getClassAndProb(linModPredTrain)$prob

### calculate the error
errorTrainLM = mean(linModClassTrain != training[,1])
errorTestLM = mean(linModClass != test[,1])

### apply the knn regression
### do the CV; divide training dataset into folds
foldLength = rep(floor(numTrain/numFolds), numFolds)
leftover = numTrain - sum(foldLength)
foldIndexVec = c(0,cumsum(foldLength))

KVec = 1:15
trainClassCV = matrix(numeric(numTrain * length(KVec)), ncol=length(KVec))
trainClassCVProb = trainClassCV
testClass = matrix(numeric(numTest * length(KVec)), ncol = length(KVec))
testClassProb = testClass

trainClass = matrix(numeric(numTrain * length(KVec)), ncol = length(KVec))
trainClassProb = trainClass

for(j in 1:length(KVec))
{
    res = knn(training[,2:257], test[2:257], training[,1], k=KVec[j], prob=T)
    testClass[,j] = res
    testClassProb[,j] = attr(res,"prob")

    res = knn(training[,2:257], training[2:257], training[,1], k=KVec[j], prob=T)
    trainClass[,j] = res
    trainClassProb[,j] = attr(res,"prob")

    for(i in 1:numFolds)
    {
        trainFolds = training[-((foldIndexVec[i]+1):foldIndexVec[i+1]),]
        testFold = training[(foldIndexVec[i]+1):foldIndexVec[i+1],]

        res = knn(trainFolds[,2:257], testFold[,2:257], trainFolds[,1], k=KVec[j], prob=T)
        trainClassCV[(foldIndexVec[i]+1):foldIndexVec[i+1],j] = res
        trainClassCVProb[(foldIndexVec[i]+1):foldIndexVec[i+1],j] = attr(res,"prob")
    }
}

### calculate the CV training error for KNN
whichFalseCV = (trainClassCV != as.numeric(training[,1]))
classErrorCVKNN = apply(whichFalseCV, 2 , mean)

whichFalseTest = (testClass != as.numeric(test[,1]))
classErrorTestKNN = apply(whichFalseTest,2,mean)

whichFalseTrain = (trainClass != as.numeric(training[,1]))
classErrorTrainKNN = apply(whichFalseTrain,2,mean)

### now plot the error
pdf("ErrorCurvesQues1a.pdf")
plot(1:15, classErrorCVKNN, main="Error Plots", xlab="K", ylab="Error", type="b", ylim=c(0,0.1))
lines(1:15, classErrorTestKNN, col="red")
points(1:15, classErrorTestKNN, col="red")
lines(1:15, classErrorTrainKNN, col="brown")
points(1:15, classErrorTrainKNN, col="brown")
abline(h=errorTrainLM, col="blue")
abline(h=errorTestLM, col="green")
### optimal number of class neighbours

K=5
falseKNN = whichFalseTest[,K]
falseLM = (linModClass != test[,1])

falseKNNNotLM = (falseKNN & !falseLM)
falseLMNotKNN = (falseLM & !falseKNN)
falseBoth = (falseLM & falseKNN)

pdf("DigitsMisclassifiedQues1b.pdf")
par(mfrow=c(3,2))
image(1:16,1:16, matrix(as.numeric(training[which(falseKNNNotLM)[1],2:257]), ncol=16)[,16:1], main="KNN wrong, LM right")
image(1:16,1:16, matrix(as.numeric(training[which(falseKNNNotLM)[2],2:257]), ncol=16)[,16:1], main="KNN wrong, LM right")
image(1:16,1:16, matrix(as.numeric(training[which(falseLMNotKNN)[1],2:257]), ncol=16)[,16:1], main="KNN right, LM wrong")
image(1:16,1:16, matrix(as.numeric(training[which(falseLMNotKNN)[2],2:257]), ncol=16)[,16:1], main="KNN right, LM wrong")
image(1:16,1:16, matrix(as.numeric(training[which(falseBoth)[1],2:257]), ncol=16)[,16:1], main="Both wrong")
image(1:16,1:16, matrix(as.numeric(training[which(falseBoth)[2],2:257]), ncol=16)[,16:1], main="Both wrong")
dev.off()

### as a new method, take the average of the probabilities of the first 2

combPredTest = testClass
combPredTrain = trainClass
for(k in 1:15)
{
    combPredTest[瓾testClassProb[,k]<linModClassProb,k] = linModClass[瓾testClassProb[,k]<linModClass]
    combPredTrain[瓾trainClassProb[,k]<linModClassTrainProb,k] = linModClassTrain[瓾trainClassProb]
}

whichFalseCombTest = (combPredTest != as.numeric(test[,1]))
classErrorCombTest = apply(whichFalseCombTest,2,mean)

whichFalseCombTrain = (combPredTrain != as.numeric(training[,1]))
classErrorCombTrain = apply(whichFalseCombTrain,2,mean)

pdf("ErrorCurvesQues1c.pdf")
plot(1:15, classErrorTestKNN, main="Error Plots", xlab="K", ylab="Error", type="b", ylim=c(0,0.1))
points(1:15, classErrorTestKNN, col="red")
lines(1:15, classErrorTrainKNN, col="red", lty=2, lwd=2)
points(1:15, classErrorTrainKNN, col="red")
lines(1:15, classErrorCombTest, col="black")
points(1:15, classErrorCombTest, col="black")
lines(1:15, classErrorCombTrain, col="black", lty=2, lwd=2)
points(1:15, classErrorCombTrain, col="black")
abline(h=errorTrainLM, col="blue")
Code for question 5:

```r
library(leaps)
seed(1)

### set the necessary parameters
p=20
sigma=sqrt(p)
numSims=30
trainSize=50
testSize=10000

### generate the beta
beta = rnorm(p)
orderBeta = order(abs(beta), decreasing=T)
beta=beta[orderBeta]

### generate the test dataset
test.X = matrix(rnorm(p*testSize), ncol=p)
test.y = test.X %*% beta + rnorm(testSize, sd=sigma)

### prepare an array that holds all predictions
test.pred = array(numeric(testSize*p*numSims), dim=c(testSize,p,numSims))
test.pred.simple = array(numeric(testSize*p*numSims), dim=c(testSize,p,numSims))

### run the simulations
for(i in 1:numSims)
{
    ### generate the training dataset
    train.X = matrix(rnorm(trainSize*p),ncol=p)
    train.y = train.X %*% beta + rnorm(trainSize, sd=sigma)
    train.mod = leaps(x=train.X, y=train.y, nbest=1, int=F)
    for(j in 1:p)
    {
        ### now for each size fix a model on the training set and predict on the test set
        train.X.select = train.X[,train.mod$which[j,], drop=F]
        model = lm.fit(x=train.X.select,y=train.y)
        test.X.select = test.X[,train.mod$which[j,], drop=F]
        test.pred[,j,i] = test.X.select %*% model$coeff
    }
    ### now for the simple model
    train.X.select = train.X[,1:j, drop=F]
    model = lm.fit(x=train.X.select,y=train.y)
}
```

test.X.select = test.X[,1:j, drop=F]
test.pred.simple[,j,i] = test.X.select %*% model$coeff
}

### now calculate bias, variance and prediction error curves
biasAveOverSim = apply(test.pred, c(1,2), mean) - as.vector(test.X %*% beta)
bias = apply(biasAveOverSim^2,2,mean)
varAveOverSim = apply(test.pred,c(1,2),var)
variance = apply(varAveOverSim,2,mean)
predError = apply((test.pred-as.vector(test.y))^2, MARGIN=2, FUN=mean)

### now the same for the simple model
biasAveOverSimSimple = apply(test.pred.simple, c(1,2), mean) - as.vector(test.X %*% beta)
biasSimple = apply(biasAveOverSimSimple^2,2,mean)
varAveOverSimSimple = apply(test.pred.simple,c(1,2),var)
varianceSimple = apply(varAveOverSimSimple,2,mean)
predErrorSimple = apply((test.pred.simple-as.vector(test.y))^2, MARGIN=2, FUN=mean)

pdf("ErrorCurvesQues5.pdf")
plot(predErrorSimple, type="l", main="Error curves", xlab="k", ylab="error", ylim=c(0, max(predError)), lwd=2)
lines(biasSimple, lty=2, lwd=2)
lines(varianceSimple, lty=3, lwd=2)
lines(predError, col="red", lwd=2)
lines(bias, col="red", lty=2, lwd=2)
lines(variance, col="red", lty=3,lwd=2)
legend("bottomright", legend=c("Pred Error Simple", "Bias Simple", "Variance Simple", "Pred Error AS", "Bias AS", "Variance AS"), col=c("black","black","black","red","red","red"),lty=c(1,2,3,1,2,3),lwd=c(2,2,2,2,2,2,2))
dev.off()