STT481: Homework 3 solution

100 points total

1(a).

```
library(ISLR)
data("Weekly")
### Model (i)
error <- 0
for(i in 1:nrow(Weekly)){
  glm.fit <- glm(Direction ~ Lag1 + Lag2, data = Weekly[-i,], family = binomial)
  pred.prob <- predict(glm.fit, Weekly[i,,drop=FALSE], type = "response")
  pred.class <- ifelse(pred.prob < 0.5, "Down", "Up")
  error <- error + as.numeric(pred.class !=  Weekly[i,"Direction"])
}
print(error/nrow(Weekly))
```

```
## [1] 0.4499541
```

```
### Model (ii)
error <- 0
for(i in 1:nrow(Weekly)){
  glm.fit <- glm(Direction ~ Lag1 + Lag2 + I(Lag1^2) + I(Lag2^2),
                 data = Weekly[-i,], family = binomial)
  pred.prob <- predict(glm.fit, Weekly[i,,drop=FALSE], type = "response")
  pred.class <- ifelse(pred.prob < 0.5, "Down", "Up")
  error <- error + as.numeric(pred.class !=  Weekly[i,"Direction"])
}
print(error/nrow(Weekly))
```

```
## [1] 0.4582185
```

1(b). Model (i) has smaller LOOCV estimate, so Model (i) is better result on this data based on the LOOCV estimates.

1(c). Yes, they are the same as the results in (a).

```
library(boot)
# Since the response is a binary variable an
# appropriate cost function for glm.cv is
cost <- function(r, pi = 0) mean(abs(r - pi) > 0.5)

glm.fit <- glm(Direction ~ Lag1 + Lag2, data = Weekly, family = binomial)
cv.error.1 <- cv.glm(Weekly, glm.fit, cost, K = nrow(Weekly))$delta[1]
print(cv.error.1)
```

```
## [1] 0.4499541
```

```
glm.fit <- glm(Direction ~ Lag1 + Lag2 + I(Lag1^2) + I(Lag2^2),
               data = Weekly, family = binomial)
cv.error.2 <- cv.glm(Weekly, glm.fit, cost, K = nrow(Weekly))$delta[1]
print(cv.error.2)
```

```
## [1] 0.4582185
```

1(d).

```
set.seed(1)  ## the seed can be arbitrary but we use 1 for the sake of consistency
fold.index <- cut(sample(1:nrow(Weekly)), breaks=10, labels=FALSE)

### Model (i)
error <- rep(0,10)
for(i in 1:10){
  glm.fit <- glm(Direction ~ Lag1 + Lag2, data = Weekly[fold.index != i,], family = binomial)
  pred.prob <- predict(glm.fit, Weekly[fold.index == i,,drop=FALSE], type = "response")
  pred.class <- ifelse(pred.prob < 0.5, "Down", "Up")
  error[i] <- mean(pred.class !=  Weekly[fold.index == i,"Direction"])
}
print(mean(error))
```

```
## [1] 0.4471882
```

```
### Model (ii)
error <- rep(0,10)
for(i in 1:10){
  glm.fit <- glm(Direction ~ Lag1 + Lag2 + I(Lag1^2) + I(Lag2^2),
                 data = Weekly[fold.index != i,], family = binomial)
  pred.prob <- predict(glm.fit, Weekly[fold.index == i,,drop=FALSE], type = "response")
  pred.class <- ifelse(pred.prob < 0.5, "Down", "Up")
  error[i] <- mean(pred.class !=  Weekly[fold.index == i,"Direction"])
}
print(mean(error))
```

```
## [1] 0.4490146
```

1(e). Both have the same CV estimates, so both are equally accurate. Because Model (i) is a simpler model (linear) so Model (i) will be preferred given that they have the same prediction accuracy.

1(f). They are not the same as the ones in (d) because the 10-fold divisions of data are different in the function `cv.glm`. This example shows that 10-fold CV involves randomness of the data split, whereas LOOCV doesn't involve the randomness.

```
library(boot)
# Since the response is a binary variable an
# appropriate cost function for glm.cv is
cost <- function(r, pi = 0) mean(abs(r - pi) > 0.5)

glm.fit <- glm(Direction ~ Lag1 + Lag2, data = Weekly, family = binomial)
cv.error.1 <- cv.glm(Weekly, glm.fit, cost, K = 10)$delta[1]
print(cv.error.1)
```

```
## [1] 0.4517906
```

```
glm.fit <- glm(Direction ~ Lag1 + Lag2 + I(Lag1^2) + I(Lag2^2), data = Weekly, family = binomial)
cv.error.2 <- cv.glm(Weekly, glm.fit, cost, K = 10)$delta[1]
print(cv.error.2)
```

```
## [1] 0.4527089
```

1(g). 10-fold CV is much faster, because LOOCV involves 1089 iterations while 10-fold CV only involves 10 iterations.

2(a).

```
library(ISLR)
library(class)
data("Default")
X <- Default[, c("student", "balance", "income")]
X[,"student"] <- ifelse(X[,"student"] == "Yes", 1, 0)
X <- scale(X)
y <- Default[, "default"]

K.vt <- c(1,5,10,15,20,25,30)
error.k <- rep(0, length(K.vt))
counter <- 0
for(k in K.vt){
  counter <- counter + 1 # counter for error.k
  error <- 0 # reset the error when running each k
  for(i in 1:nrow(Default)){
    pred.class <- knn(X[-i,], X[i,], y[-i], k=k)
    error <- error + as.numeric(pred.class !=  y[i])
  }
  error.k[counter] <- error/nrow(Default)
}
print(error.k)
```

```
## [1] 0.0419 0.0322 0.0284 0.0278 0.0278 0.0274 0.0276
```

2(b). $K = 25$ appears to provide the best results because it has the lowest LOOCV estimates. But $K = 15, 20, 25, 30$ have very similar LOOCV so we also expect that $K = 15, 20, 30$ will result in similar prediction accuracy.

```
print(K.vt[which.min(error.k)])
```

```
## [1] 25
```

2(c). Yes, they are the same as the results in (a).

```
loocv.rate <- rep(0,7)
counter <- 0
for(k in c(1,5,10,15,20,25,30)){
  counter <- counter + 1 # counter for k
  cvknn <- knn.cv(X, y, k = k)  ## the little k here is the number of nearest neighbors not k-fold
  loocv.rate[counter] <- mean(cvknn != y)
}
print(loocv.rate)
```

```
## [1] 0.0419 0.0322 0.0286 0.0278 0.0278 0.0274 0.0274
```

2(d).

```r
set.seed(10)  ## the seed can be arbitrary but we use 10 for the sake of consistency
fold.index <- cut(sample(1:nrow(Default)), breaks=10, labels=FALSE)
K.vt <- c(1,5,10,15,20,25,30)
error.k <- rep(0, length(K.vt))
counter <- 0
for(k in K.vt){
  counter <- counter + 1 # counter for error.k
  error <- 0 # reset the error when running each k
  for(i in 1:10){
    pred.class <- knn(X[fold.index!=i,], X[fold.index==i,], y[fold.index!=i], k=k)
    error <- error + sum(pred.class !=  y[fold.index==i])
  }
  error.k[counter] <- error/nrow(Default)
}
print(error.k)
```

```
## [1] 0.0429 0.0318 0.0290 0.0281 0.0282 0.0278 0.0283
```

2(e). The best K appears to be 25, which gives the lowest error 0.0278.

3(a). The best K appears to be 1.

```
train.dat <- read.csv("zipcode_train.csv")
train.dat$Y <- as.factor(train.dat$Y)
test.dat <- read.csv("zipcode_test.csv")
test.dat$Y <- as.factor(test.dat$Y)

set.seed(9999)
fold.index <- cut(sample(1:nrow(train.dat)), breaks=10, labels=FALSE)

K.vt <- 1:30
error.k <- rep(0, length(K.vt))
counter <- 0
for(k in K.vt){
  counter <- counter + 1 # counter for error.k
  error <- 0 # reset the error when running each k
  for(i in 1:10){
    pred.class <- knn(train.dat[fold.index!=i,1:256], train.dat[fold.index==i,1:256],
                      train.dat[fold.index!=i,"Y"], k=k)
    error <- error + sum(pred.class !=  train.dat[fold.index==i,"Y"])
  }
  error.k[counter] <- error/nrow(train.dat)
}
print(error.k)
```

```
##  [1] 0.001728111 0.002880184 0.002880184 0.004032258 0.002880184 0.003456221
##  [7] 0.002880184 0.004608295 0.004032258 0.004608295 0.005184332 0.005760369
## [13] 0.006336406 0.006336406 0.007488479 0.007488479 0.008064516 0.008640553
## [19] 0.008640553 0.008064516 0.009216590 0.010368664 0.010368664 0.010368664
## [25] 0.010944700 0.010944700 0.010944700 0.010944700 0.010944700 0.010944700
```

3(b). The KNN with the best K performs better than the logistic regression and LDA since it has lower test error. The result might be diiferent than yours because we may use different random seeds.

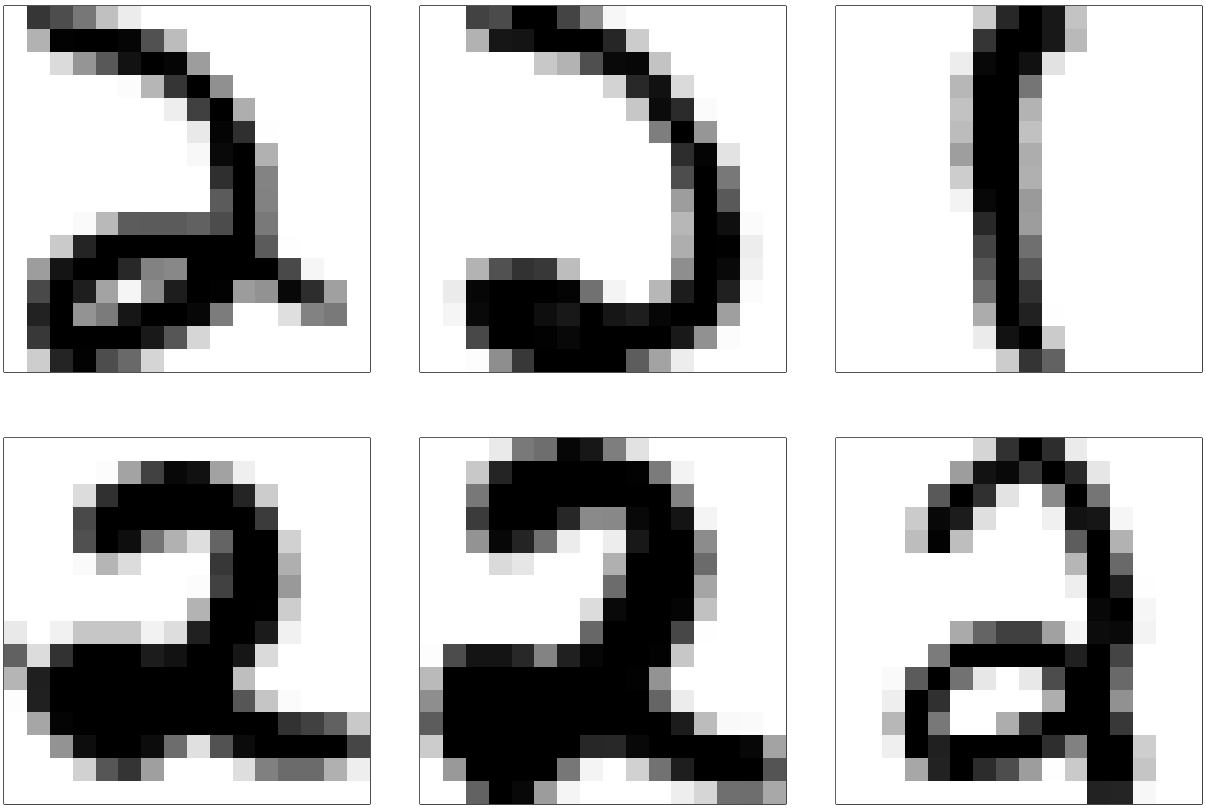
```
y <- knn(train.dat[,1:256],test.dat[,1:256], train.dat[,"Y"],k = K.vt[which.min(error.k)])
print(mean(y!=test.dat[,"Y"]))
```

```
## [1] 0.01298701
```

3(c).

```
COLORS <- c("white", "black")
CUSTOM_COLORS <- colorRampPalette(colors = COLORS)
vis <- function(i){
  par(pty = "s", mar = c(1, 1, 1, 1), xaxt = "n", yaxt = "n")
  z <- matrix(as.numeric(test.dat[i,1:256]), 16, 16)
  image(1:16,1:16,z[,16:1], col = CUSTOM_COLORS(256))
}

false.index <- which(y!=test.dat[,"Y"])
par(mfrow=c(2,3))
for(i in 1:length(false.index))  vis(i)
```



```
par(mfrow=c(1,1))
```

4(a). Load data and standardization.

```
library(FNN)
```

```
##
## Attaching package: 'FNN'
```

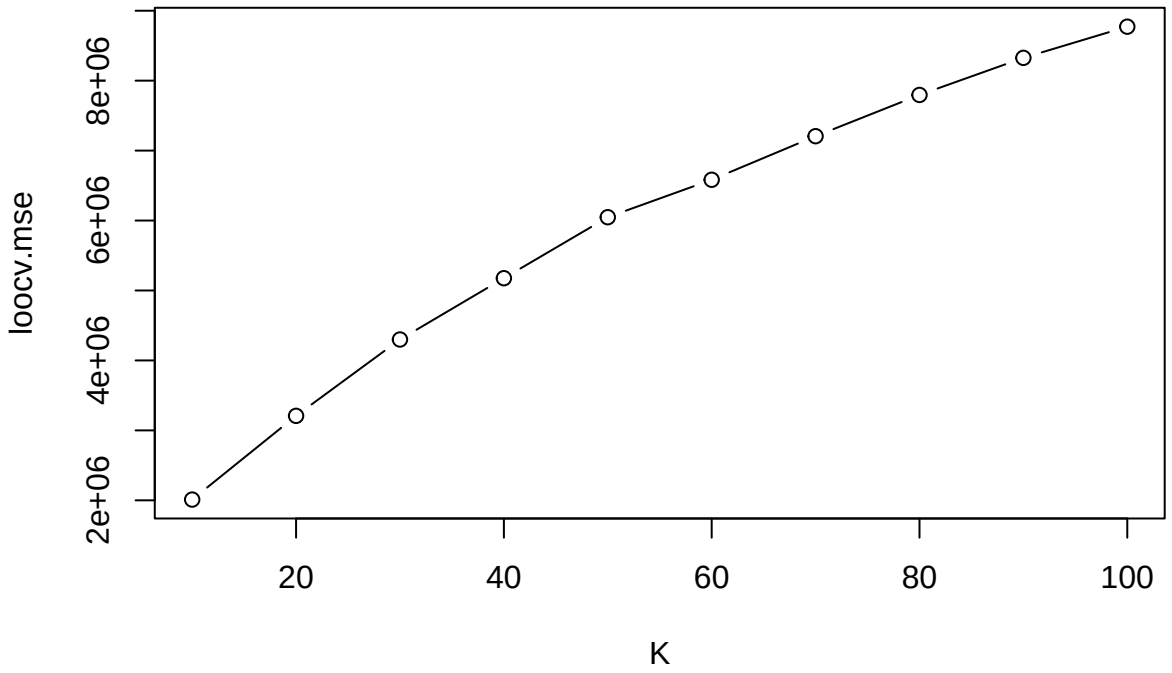
```
## The following objects are masked from 'package:class':
##
##     knn, knn.cv
```

```
train.df <- read.csv("toyota.csv")
test.df <- data.frame("model"=c("RAV4", "Camry"), "year"=c(2019, 2015),
                      "transmission"=c("Automatic","Automatic"), "mileage"=c(12345,50000),
                      "fuelType"=c("Diesel", "Hybrid"), "tax"=c(150,130),
                      "mpg"=c(25,50), "engineSize"=c(2,3))

# combine and then transform together. Will separate them later
toyata.all <- rbind(train.df[,-3], test.df) # -3 removing the output (Sales) column
X.all <- model.matrix( ~ ., data=toyata.all)[,-1]
X.all <- scale(X.all)
# Separate them to train and test (original size)
X.train <- X.all[1:nrow(train.df),]
```

Compute LOOCV MSE:

```
loocv.mse <- rep(0,10)
counter <- 0
for(k in c(10,20,30,40,50,60,70,80,90,100)){
  counter <- counter + 1 # counter for k
  cvknn <- knn.reg(X.train, NULL, train.df$price, k = k)
  ## the little k here is the number of nearest neighbors not k-fold
  ## X.train is the training input
  ## y.train is the training output
  loocv.mse[counter] <- mean(cvknn$residuals^2)
}
plot(c(10,20,30,40,50,60,70,80,90,100), loocv.mse, type="b", xlab="K")
```

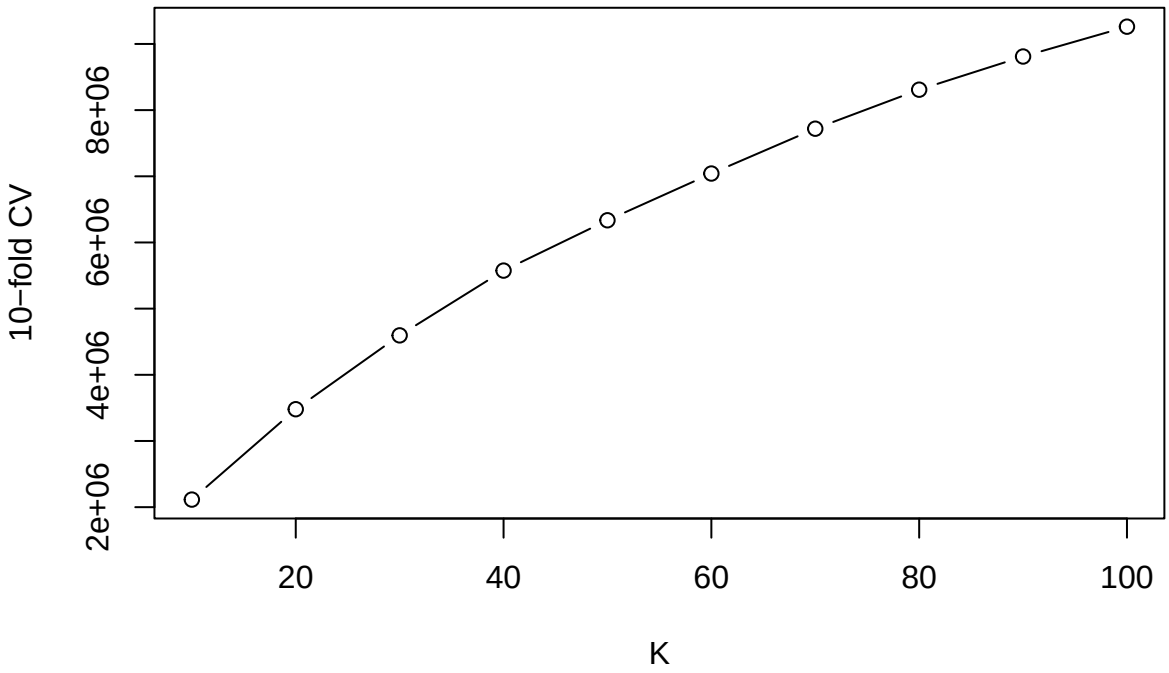


4(b). $K = 10$ appears to provide the best results because it has the lowest LOOCV MSEs.

4(c). Compute 10-fold CV MSEs:

```
set.seed(10)  ## the seed can be arbitrary but we use 10 for the sake of consistency
fold.index <- cut(sample(1:nrow(X.train)), breaks=10, labels=FALSE)
K.vt <- c(10,20,30,40,50,60,70,80,90,100)
error.k <- rep(0, length(K.vt))
counter <- 0
for(k in K.vt){
  counter <- counter + 1 # counter for error.k
  mse <- rep(0,10) # initialize an mse object to record the MSE for each fold
  for(i in 1:10){
    pred.out <- knn.reg(X.train[fold.index!=i,], X.train[fold.index==i,],
                        train.df$price[fold.index!=i], k=k)
    mse[i] <- mean((pred.out$pred -  train.df$price[fold.index==i])^2)
  }
  error.k[counter] <- sum(mse)/10
}

plot(c(10,20,30,40,50,60,70,80,90,100), error.k, type="b", xlab="K", ylab="10-fold CV")
```




```
print(min(error.k))
```

```
## [1] 2114478
```

4(d). $K = 10$ appears to provide the best results because it has the lowest CV MSEs, which is 2072074.

4(e). The 10-fold CV MSE is 2966045.

```
library(boot)
data.df <- data.frame(X.train, price = train.df$price)
lm.fit <- glm(price ~ ., data=data.df)
set.seed(100)
print(cv.glm(data.df, lm.fit, K = 10)$delta[1])
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading
```

```
## [1] 2969050
```

4(f). KNN with $K = 10$ has a lower CV MSE, so I would choose KNN with $K = 10$ for this problem.