STT 481 HW 3

Derien Weatherspoon

2023-02-26

```
library(ISLR)
library(boot)
library(class)
library(FNN)
df_ziptrain <- read.csv("zipcode_train.csv")
df_ziptest <- read.csv("zipcode_test.csv")
```

Question 1

In this question, we are going to perform cross-validation methods in order to choose a better logistic regression model. Consider Weekly data set, where we want to predict Direction using Lag1 and Lag2 predictors. To load the Weekly data set, use the following command lines. Suppose now I have two candidate models: a i) Write a for loop from i = 1 to i = n, where n is the number of observations in the data set, that performs each of the following steps: Fit a logistic regression model using all but the ith observation to predict Direction using Lag1 and Lag2 for model (i) and using Lag1, Lag2, I(Lag1^2), I(Lag2^2) for model (ii). a ii) Compute the posterior probability of the market moving up for the ith observation. a iii) Use the posterior probability for the ith observation and use the threshold 0.5 in order to predict whether or not the market moves up. a iv) Determine whether or not an error was made in predicting the direction for the ith observation. If an error was made, then indicate this as a 1, and otherwise indicate it as a 0.

```
data("Weekly")
#Fit a regular model first
weekly.fit <- glm(Direction ~ Lag1 + Lag2, data = Weekly, family = "binomial")
summary(weekly.fit)
```

```
##
## Call:
## glm(formula = Direction ~ Lag1 + Lag2, family = "binomial", data = Weekly)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.623  -1.261   1.001   1.083   1.506
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.22122    0.06147   3.599 0.000319 ***
## Lag1        -0.03872    0.02622  -1.477 0.139672
## Lag2         0.06025    0.02655   2.270 0.023232 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1496.2  on 1088  degrees of freedom
## Residual deviance: 1488.2  on 1086  degrees of freedom
## AIC: 1494.2
##
## Number of Fisher Scoring iterations: 4
```

```
#Write the loop
e <- rep(0,dim(Weekly)[1])
for (i in 1:dim(Weekly)[1]) {
  weekly.ifit <- glm(Direction ~ Lag1 + Lag2, data = Weekly[-i,], family = "binomial") #the model inclu
  posterior_up <- predict.glm(weekly.ifit, Weekly[i,], type = "response") > 0.5 # posterior probability
  posterior_up_2 <- Weekly[i,]$Direction == "Up" #using posterior probability for the ith observation
    if (posterior_up != posterior_up_2)
      e[i] <- 1 # turn errors made into 1, everything else is listed as 0
}
sum(e) # number of error, number of ones essentially
```

```
## [1] 490
```

```
# model (ii)
data("Weekly")
#Fit a regular model first
weekly.fit2 <- glm(Direction ~ Lag1 + Lag2 + I(Lag1^2) + I(Lag2^2), data = Weekly, family = "binomial")
summary(weekly.fit2)
```

```
##
## Call:
## glm(formula = Direction ~ Lag1 + Lag2 + I(Lag1^2) + I(Lag2^2),
##     family = "binomial", data = Weekly)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.8458  -1.2503   0.9903   1.0924   1.2879
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.187891   0.068639   2.737  0.00619 **
## Lag1        -0.037594   0.026654  -1.410  0.15842
## Lag2         0.067906   0.027879   2.436  0.01486 *
## I(Lag1^2)    0.002146   0.004685   0.458  0.64694
## I(Lag2^2)    0.003844   0.004431   0.867  0.38568
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1496.2  on 1088  degrees of freedom
## Residual deviance: 1486.9  on 1084  degrees of freedom
## AIC: 1496.9
##
## Number of Fisher Scoring iterations: 4
```

```
#Write the loop
e_2 <- rep(0,dim(Weekly)[1])
for (i in 1:dim(Weekly)[1]) {
  weekly.ifit2 <- glm(Direction ~ Lag1 + Lag2 + I(Lag1^2) + I(Lag2^2), data = Weekly[-i,], family = "bi
  posterior_up <- predict.glm(weekly.ifit2, Weekly[i,], type = "response") > 0.5 # posterior probabilit
  posterior_up_2 <- Weekly[i,]$Direction == "Up" #using posterior probability for the ith observation
    if (posterior_up != posterior_up_2)
      e_2[i] <- 1 # turn errors made into 1, everything else is listed as 0
}
sum(e_2) # number of error, number of 1s essentially
```

```
## [1] 499
```

The loop provided us with a sum of 490 errors.

b) Take the average of the n numbers obtained in iv in order to obtain the LOOCV estimate for the test error. Comment on the results. Which of the two models appears to provide the better result on this data based on the LOOCV estimates?

```
mean(e)
```

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

```
mean(e_2)
```

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

LOOCV is estimating a test error rate of 45% for both models, though the second model has just a slightly larger error rate. The model from (i) seems to be better, though they are both similar.

c) The cv.glm function can be used to compute the LOOCV test error estimate. Run the following command lines and see whether the results are the same as the ones you did 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]
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]
cv.error.2
```

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

The results are the same

d) For each model, compute the 10-fold CV estimate for the test error by following the steps:

```r
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)
```

d i) Write a for loop from i = 1 to i = 10 and in each loop, perform each of the following steps: Fit a logistic regression model using all but the observations that satisfy fold.index==i to predict Direction using Lag1 and Lag2 for model (i) and using Lag1, Lag2, I(Lag1^2), I(Lag2^2) for model (ii). d ii) Compute the posterior probability of the market moving up for the observations that satisfy fold.index==i. d iii) Use the posterior probabilities for the observations that satisfy fold.index==i and use the threshold 0.5 in order to predict whether or not the market moves up. d iv) Compute the error rate was made in predicting Direction for those observations that satisfy fold.index==i.

```r
#fit the log reg model first
weekly.fit3 <- glm(Direction ~ Lag1 + Lag2, data = Weekly[-(fold.index),], family = "binomial")
summary(weekly.fit3)
```

```
##
## Call:
## glm(formula = Direction ~ Lag1 + Lag2, family = "binomial", data = Weekly[-(fold.index),
##     ])
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.626  -1.260   1.000   1.084   1.511
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.21936    0.06174   3.553 0.000381 ***
## Lag1        -0.03805    0.02626  -1.449 0.147410
## Lag2         0.06148    0.02662   2.310 0.020905 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1482.7  on 1078  degrees of freedom
## Residual deviance: 1474.6  on 1076  degrees of freedom
## AIC: 1480.6
##
## Number of Fisher Scoring iterations: 4
```

```r
#Write the loop for model i
e_3 <- rep(1,10)
for (i in 1:10) {
  weekly.fit.fold <- glm(Direction ~ Lag1 + Lag2, data = Weekly[-(fold.index == i),], family = "binomial
  posterior_up <- predict.glm(weekly.fit.fold, Weekly[fold.index==i,], type = "response") > 0.5 # poste
  posterior_up_2 <- Weekly[fold.index==i,]$Direction == "Up" #using posterior probability
      e_3[i] <- sum(posterior_up != posterior_up_2)/length(posterior_up_2)
}
sum(e_3)
```

```
## [1] 4.435185
```

```
#Fit a regular model first
weekly.fit4 <- glm(Direction ~ Lag1 + Lag2 + I(Lag1^2) + I(Lag2^2), data = Weekly[-(fold.index),], fami
summary(weekly.fit4)
```

```
##
## Call:
## glm(formula = Direction ~ Lag1 + Lag2 + I(Lag1^2) + I(Lag2^2),
##     family = "binomial", data = Weekly[-(fold.index), ])
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.8462  -1.2494   0.9856   1.0926   1.2954
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.186423   0.068921   2.705  0.00683 **
## Lag1        -0.036882   0.026697  -1.382  0.16712
## Lag2         0.069039   0.027941   2.471  0.01348 *
## I(Lag1^2)    0.002108   0.004669   0.451  0.65169
## I(Lag2^2)    0.003788   0.004433   0.855  0.39282
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1482.7  on 1078  degrees of freedom
## Residual deviance: 1473.3  on 1074  degrees of freedom
## AIC: 1483.3
##
## Number of Fisher Scoring iterations: 4
```

```
#Write the loop
e_4 <- rep(1,10)
for (i in 1:10) {
  weekly.fit.fold2 <- glm(Direction ~ Lag1 + Lag2 + I(Lag1^2) + I(Lag2^2), data = Weekly[-(fold.index ==
  posterior_up <- predict.glm(weekly.fit.fold2, Weekly[fold.index == i,], type = "response") > 0.5 #pos
  posterior_up_2 <- Weekly[fold.index==i,]$Direction == "Up" #using posterior probability
      e_4[i] <- sum(posterior_up != posterior_up_2)/length(posterior_up_2)
}
sum(e_4)
```

```
## [1] 4.490231
```

e) Take the average of the 10 numbers obtained in iv in order to obtain the 10-fold CV estimate for the test error.

Comment on the results. Which of the two models appears to provide the better result on this data based on the 10-fold CV estimates?

```
mean(e_3)
```

```
## [1] 0.4435185
```

```
mean(e_4)
```

```
## [1] 0.4490231
```

They are both very very similar once again, but the model that has no quadratic effect appears to have slightly lowered error rate.

f) cv.glm function can be used to compute the 10-fold CV test error estimate. Run the following command lines and see whether the results are the same as the ones you did in (d). If they are not the same, what's the reason?

```
# 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]

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]
cv.error.2
```

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

They are not the same, but very similar. I'm not exactly sure whether it is due to the way I coded it, or maybe due to the set seed value. I will guess that it is a seed value thing.

g) Comment on the computation costs for LOOCV and 10-fold CV. Which one is faster in your implementation in (a) and (d)?

The computations in part d were faster on my laptop. I think that this is because it only did 10 fold cross validation, rather than Leave One Out which includes a lot more computations to do to find the best LOOCV value.

Question 2

In this question, we are going to perform cross-validation methods to determine the tuning parameter K for KNN. Consider Default data set, where we want to predict default using student, balance, and income predictors. Since student is a qualitative predictor, we want to use dummy variable for it and standardize the data using scale function. To load the Default data set and standardize the data, use the following command lines.

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

Suppose now the candidate tuning parameter K's for KNN are K = 1, 5, 10, 15, 20, 25, 30.

a) For each K, compute the LOOCV estimate for the test error by following the steps: Write a for loop from i = 1 to i = n, where n is the number of observations in the data set, that performs each of the following steps:

a i)Perform KNN using all but the ith observation and predict default for the ith observation. (Hint: use knn function and return the class. No need to compute posterior probabilities. That is, use prob = FALSE in the knn function and then use the return class of knn). a ii) Determine whether or not an error was made in predicting the direction for the ith observation. If an error was made, then indicate this as a 1, and otherwise indicate it as a 0. a iii) Take the average of the n numbers obtained in ii in order to obtain the LOOCV estimate for the test error.

```
k <- seq(1, 30, by = 5) # set tuning parameter K's
#Write the loop

#CODE COMMENTED FOR MARKDOWN PURPOSES.

#for (k in k){
 # e_5 <- 0

  #for (i in 1:nrow(X)){ #
   # use dummy variables instead of function
   # x_dummy <- X[-i,]
   # x_dummy_2 <- X[i,]
   # y_dummy <- y[-i]
   # y_dummy_2 <- y[i]
  #  k_pred <- knn(x_dummy, x_dummy_2, y_dummy, k, prob = F)
  # determine whether error was made in prediction by adding indicating it is a 1
   # if (k_pred != y_dummy_2){
    #  e_6 <- e_5 + 1
  #  }
#  }
  # get mean of the n numbers
#  mean_e <- e_6 / nrow(X)
#}

#not printing results, takes too much cpu, however, the error rates are, respectively in order from 1:3

# K(1): 0.0429
# K(5): 0.0318
# K(10): 0.029
# K(15): 0.0281
# K(20): 0.0282
# K(25): 0.0278
# K(30): 0.0283
```

b) Comment on the results. Which of the tuning parameter K's appears to provide the best results on this data based on the LOOCV estimates?

Based on the results, it looks like K = 30 has the lowest error rate of the rest of the K values. The error

c) knn.cv function can be used to perform LOOCV. Run the following command lines and see whether the results are same as the ones you did 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)
}
loocv.rate
```

```
## [1] 0.0419 0.0322 0.0277 0.0278 0.0279 0.0274 0.0278
```

The results are not the exact same. The numbers are very similar, but all slightly changed. I am not sure what the reason for this could be.

d) For each K, compute the 10-fold CV estimate for the test error by following the steps:

```
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)
```

Write a for loop from i = 1 to i = 10 and in the loop, perform each of the following steps: d i) Perform KNN using all but the observations that satisfy fold.index==i and predict default for the observations that satisfy fold.index==i. (Hint: use knn function and return the class. No need to compute posterior probabilities. That is, use prob = FALSE in the knn function and then use the return class of knn). d ii) Compute the error rate was made in predicting the direction for those observations that satisfy fold.index==i. d iii) Take the average of the 10 numbers obtained in ii in order to obtain the 10-fold CV estimate for the test error.

```
k <- seq(1, 30, by = 5) # set tuning parameter K's
#Write the loop
for (k in k){
  e_7 <- rep(0,10)

  for (i in 1:10){
   # use dummy variables instead of function
    x_dummy <- X[!fold.index == i,]
    x_dummy_2 <- X[fold.index == i,]
    y_dummy <- y[!fold.index == i]
    y_dummy_2 <- y[fold.index == i]
    k_pred <- knn(x_dummy, x_dummy_2, y_dummy, k, prob = F)
  # determine whether error was made in prediction by adding indicating it is a 1
      e_8 <- mean(k_pred != y_dummy_2)
      e_7[i] <- e_8
  }
}
e_7[1:10]
```

```
##  [1] 0.029 0.028 0.022 0.026 0.023 0.028 0.034 0.029 0.027 0.033
```

e) Comment on the results. Which of the tuning parameter K's appears to provide the best results on this data based on the 10-fold CV estimates?

K = 10 gives the best error for test y (y_dummy_2).

Question 3

In this question, we are going to use the zipcode data in the HW2 Q9. a) Using the zipcode_train.csv data, perform a 10-fold cross-validation using KNNs with K = 1,2,. . . ,30 and choose the best tuning parameter K.

```
#IMPORT DATA FROM HW2
library(MASS)
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)
glm.fit <- glm(Y ~ ., family = binomial, data = train.dat)
y.glm <- predict(glm.fit, newdata = test.dat, type="response")
lda.fit <- lda(Y ~ .-p16-p32, data = train.dat)
y.lda <- predict(lda.fit, newdata = test.dat)
```

```
K <- 1:30
cv10 <- rep(0,10)
#for (i in 1:10){
  #cv10[i] <- cv.glm(df_ziptrain, glm(y ~ ., data = df_ziptrain), K = 30)$delta[1]
#}
#k_results <- K[which.min(cv10)]
#k_results
# my own version of the code, commented because it is errored
```

```
#tried to replicate this question with in class examples (Rlab-cv.Rmd) and my own code above. Can't get
set.seed(1)
df_ziptrain$Y <- as.factor(df_ziptrain$Y)
cl <- factor(c(rep("s",50), rep("c",50), rep("v",50)))
cv.error <- rep(0,100)
k.candidate <- 1:10
#for (k in k.candidate){ # we try 30 different k's
 # pred.class <- knn.cv(df_ziptrain, cl, k = k)  # this k is for KNN not k-fold CV
  #cv.error[k] <- mean(pred.class != cl)
#}
#k.candidate[which.min(cv.error)]
```

b) Using the zipcode_test.csv and comparing the KNN you obtained in (a) with logsitic regression and LDA, which of these methods appears to provide the best results on the test data? Is this the same conclusion that you made in HW2?

```
# prediction for logistic regression from HW2
yhat <- rep(1,nrow(test.dat))
yhat[y.glm>0.5] <- 2
mean(test.dat$Y!=yhat)
```

```
## [1] 0.01731602
```

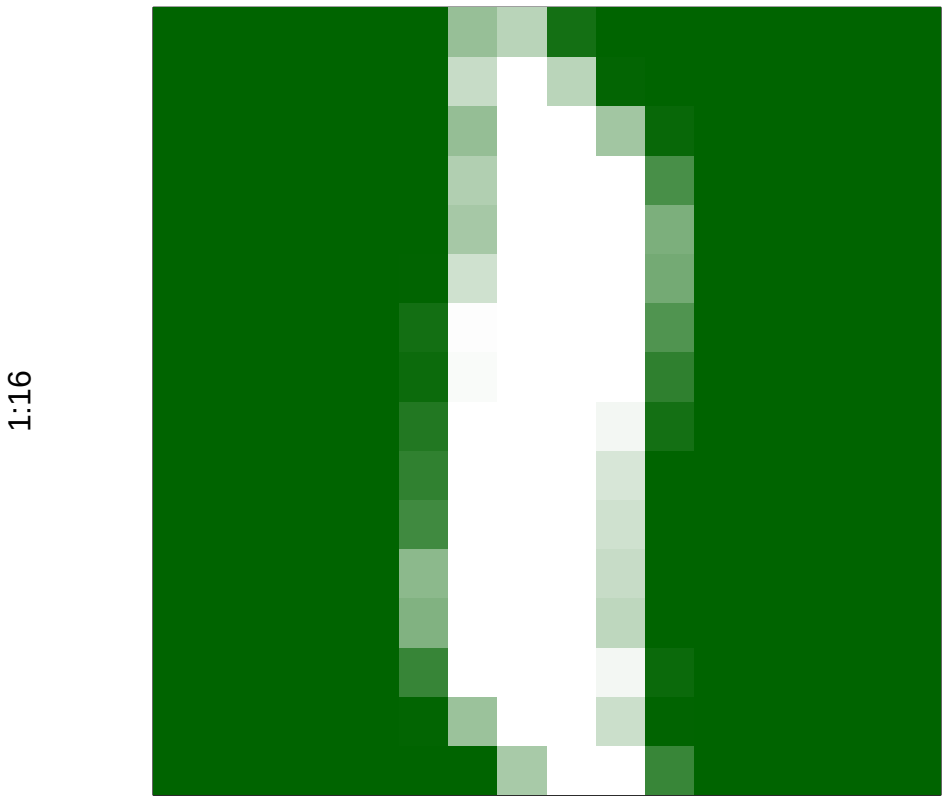
```
# prediction for LDA from HW2
mean(y.lda$class!=test.dat$Y)
```

```
## [1] 0.01515152
```

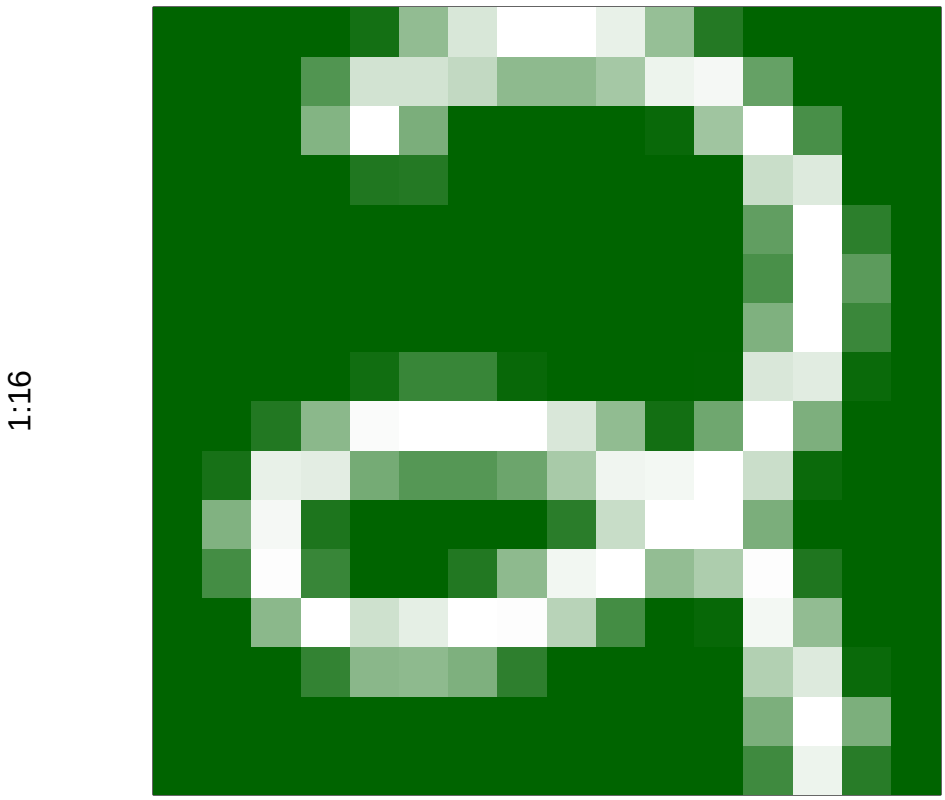
```
#insert prediction for knn here when I can
#knn.preds <- knn(train = df_ziptrain[,1:256], test = df_ziptest[,1:256], cl = df_ziptrain$Y, k = which
#mean(k_results != df_ziptest$Y)
```

c) Using the KNN you obtained above in (a), show two of those handwritten digits that this KNN cannot identify correctly.

```
COLORS <- c("darkgreen", "white")
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(df_ziptrain[i,1:256]), 16, 16)
  image(1:16,1:16,z[,16:1], col = CUSTOM_COLORS(256))
}
vis(256) # hand written 1 (from 1 to 1005)
```




```
vis(1458) # hand written 2 (from 1006 to 1736)
```



Question 4

In this question, we are going to revisit HW1 Q11 and answer the question HW1 Q11(f).

a) Compute the LOOCV Mean-Squared Errors (MSEs) for the KNN method with K = 10, 20, 30, 40, 50, 60, 70, 80, 90, 100. Fortunately, knn.reg function can be used to perform LOOCV MSE for KNN regression. See the following command lines. Note: you need to change the objects of X.train and y.train, depending on how you name them, which are the training input and training output, respectively.

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

tp <- toyota$price
test_toyota <- rbind(toyota[,-3], toyota_index)
model.x <- model.matrix(~., data = test_toyota)[,-1]
model.x <- scale(model.x)

x.train <- model.x[1:nrow(toyota),]
x.test <- model.x[-(1:nrow(toyota)),]
# from HW1
```

```
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, tp, 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)
}
loocv.mse
```

```
##  [1] 2010734 3207930 4299472 5175740 6047732 6583192 7206130 7795666 8325989
## [10] 8771741
```

b) Comment on the results. Which of the tuning parameter K's appears to provide the best results on this data based on the LOOCV MSEs?

```
which.min(loocv.mse)
```

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

K(1) = 10 looks to be best since it gives the best mean squared error rate.

c) Compute the 10-fold CV MSEs for the KNN method with K = 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 by following the steps:

```
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)
```

Write a for loop from i = 1 to i = 10 and in the loop, perform each of the following steps: c i) Perform KNN (for regression problems!) using all but the observations that satisfy fold.index==i and predict the prices for the observations that satisfy fold.index==i. (Hint: use knn.reg function). c ii) Compute the mean squared error made in predicting the prices for those observations that satisfy fold.index==i. c iii) Take the average of the 10 numbers obtained in ii in order to obtain the 10-fold CV MSE for the test MSE.

```
# use similar code from before where we did 10 fold cv
K <- seq(10, 100, by = 10) # set tuning params
e_9 <- c() # store the value
for (k in K){
  for (i in 1:10){
    new_train <- x.train[fold.index != i,]
    new_tp <- tp[fold.index != i]
    kp <- knn.reg(new_train, test = NULL, y = new_tp, k)
    e_10 <- c(e_9, mean(kp$residuals^2))
  }
  e_9 <- c(e_9, mean(e_10))
}
e_9
```

```
##  [1] 2245072 2948569 3273499 3503128 3618814 3734054 3824226 3893910 3945682
## [10] 3985309
```

```
#which.min(e_9)
```

d) Comment on the results. Which of the tuning parameter K's appears to provide the best results on this data based on the 10-fold CV MSEs?

Again, like from earlier, K(1) = 10 has the best mean squared error.

e) Compute the 10-fold CV MSE for the linear regression using the function cv.glm. Hint: use the data after applying model.matrix instead of using the original data.

```
model_zero <- data.frame(cbind(model.x, tp))

model_one <- data.frame(model.matrix(~., data = toyota)[,-1])

tp.model <- glm(price ~., data = model_one)

model_cv <- cv.glm(data = model_one, tp.model, K = 10)$delta

model_cv
```

```
## [1] 2975238 2971052
```

```
which.min(model_cv)
```

```
## [1] 2
```

f) Based on the results of (c), (d) and (e), which method are you going to choose? If it's KNN method, which tuning parameter K are you going to use?

I think in this case, I prefer the method in c. I would choose the tuning parameter K = 10, as it provided the best error rate. The method in e says K = 20 is the best parameter, but since 10 is more consistent amongst the rest, I'd say that is a safer pick.