Stat 432 Homework 6

Giang Le (gianghl2)

Assigned: Sep 27, 2021; Due: 11:59 PM CT, Oct 5, 2021

Contents

Question 1: Fitting KNN

We are going to use the `caret` package to do a full tuning. Use the Handwritten Digit Data in the lecture note. The data contains two sets: `zip.train` and `zip.test`. The first column is the true digit. For this question, [5 points] subset the data to include only two digits: 4 and 9. Hence, this is a two-class classification problem. You need to setup the following tuning method using the `caret` package. Apply this tuning to the training dataset.

- [5 points] Use repeated 5-fold cross-validation, with 3 repeats.
- [5 points] For k, use all integers from 1 to 10

After completing the cross-validation of your training data, [10 points] report the best tuning and produce a plot that shows k against the cross-validation error. Predict the class label in the testing data.

- [5 points] Present the confusion table for the testing data
- [5 points] Calculate and report the testing error

Bonus Question [5 points]: Have you noticed that when k is an even number, the performance is worse than odd numbers. What could be the cause?

```
# Installing the library
install.packages("caret", repos = "http://cran.us.r-project.org")
```

```
##
## The downloaded binary packages are in
##  /var/folders/9c/3_mgdyf12z7dvb8rt4d60nt80000gn/T//RtmpAwbrNe/downloaded_packages
```

```
library("caret")
```

```
## Loading required package: ggplot2
```

```
## Loading required package: lattice
```

```
# Load data from my local dir.
load("/Users/gianghale/Desktop/fall-2021/stat-432/zipdata.RData")

# Look at the dimensions of train and test data
dim(zip.train)
```

```
## [1] 7291  257
```

```
## [1] 7291  257
dim(zip.test)
```

```
## [1] 2007  257
```

```
## [1] 2007  257

df.train <- data.frame(zip.train)
df.test <- data.frame(zip.test)

# I use subset to select only columns where values are 4 and 9 for the digits.

df.train.filtered <- subset(df.train, df.train$X1 == 4 | df.train$X1 == 9)
df.train.filtered$X1 <- as.factor(df.train.filtered$X1)
df.test.filtered <- subset(df.test, df.test$X1 == 4 | df.test$X1 == 9)
df.test.filtered$X1 <- as.factor(df.test.filtered$X1)
```

```
# Use repeated 5-fold cross-validation, with 3 repeats.
install.packages("kknn", repos = "http://cran.us.r-project.org")
```

```
##
## The downloaded binary packages are in
##  /var/folders/9c/3_mgdyf12z7dvb8rt4d60nt80000gn/T//RtmpAwbrNe/downloaded_packages
```

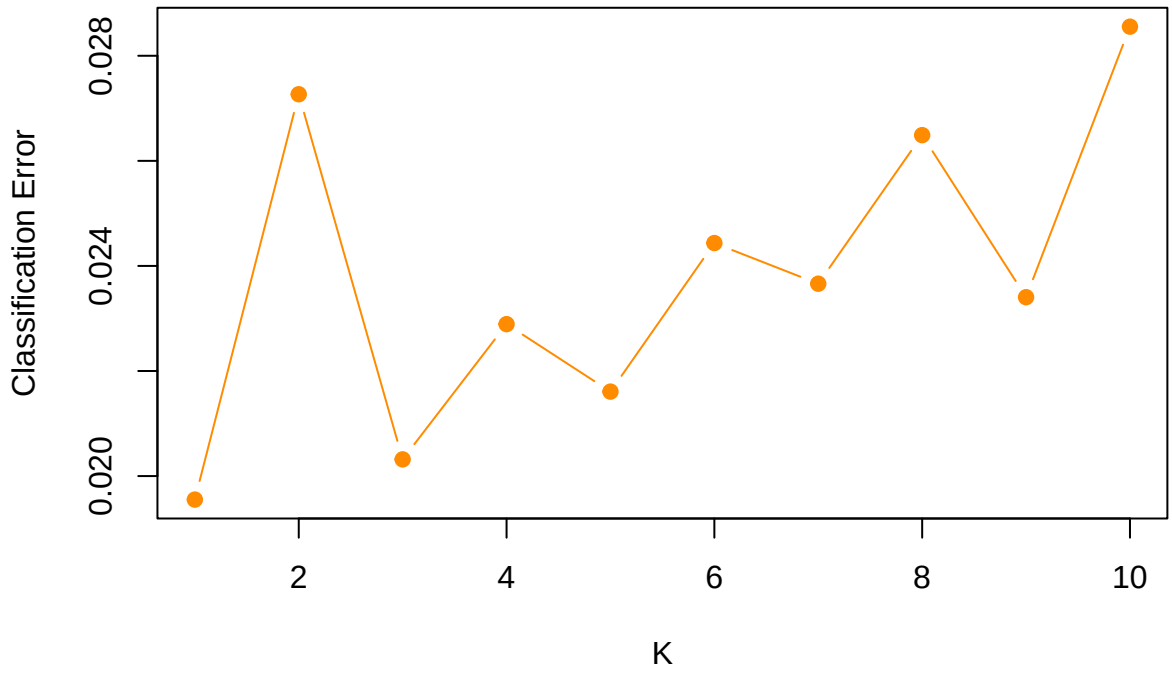
```
library("kknn")
```

```
##
## Attaching package: 'kknn'
```

```
## The following object is masked from 'package:caret':
##
##     contr.dummy
```

```
set.seed(662095561)

# I set the parameters of trainControl to be method ~ repeatedcv, 5 folds,
# and 3 repeats.
control <- trainControl(method = "repeatedcv", number = 5, repeats=3)

# Here I train the model using knn, and k values from 1 to 10.
knn.cvfit <- train(X1 ~ ., method = "knn",
                     data = df.train.filtered,
                     tuneGrid = data.frame(k = seq(1, 10, 1)),
                     trControl = control)
```

```
# report the best tuning and produce a plot that shows $k$ against the
# cross-validation error.

# according to this plot, the best k is when k = 1
 plot(knn.cvfit$results$k, 1-knn.cvfit$results$Accuracy,
       xlab = "K", ylab = "Classification Error", type = "b",
       pch = 19, col = "darkorange")
```




```
# I predict the label using k = 1 and output a confusion matrix
# According to the confusion matrix table, true positives are 189 for digit 4,
# and 171 for digit 9. There are 17 errors made in total.
library("class")
knn.fit <- knn(df.train.filtered[, 2:257], df.test.filtered[, 2:257], df.train.filtered[,1], k=1)
table(knn.fit, df.test.filtered[,1])
```

```
##
## knn.fit   4   9
##       4 189   6
##       9  11 171
```

```
# Here are the labels, predicted on the test data.
knn.fit
```

```
##   [1] 9 9 4 9 4 9 9 4 9 9 4 9 9 9 4 9 9 4 9 9 9 9 4 9 9 9 4 9 9 9 4 9 4 4 4 4 4
##  [38] 4 9 9 4 9 4 4 4 4 4 4 4 4 9 9 4 9 9 9 9 9 9 9 9 9 4 9 4 4 4 4 9 4 9 9 4 9 9
##  [75] 9 9 9 4 9 4 4 4 4 4 4 4 9 9 4 9 4 9 4 9 4 4 4 4 4 4 4 9 9 4 4 9 4 9 9 4 9 9
## [112] 4 4 4 9 4 4 4 4 4 4 9 4 9 4 4 4 4 4 4 9 4 9 4 4 4 4 4 9 4 4 4 4 9 4 4 4 4 9 9 4
## [149] 4 9 9 9 9 4 9 9 9 4 9 9 4 4 4 4 9 9 9 4 9 9 9 9 9 4 9 4 4 4 9 4 4 9 9 9 4 9
## [186] 9 9 9 4 4 4 4 4 4 9 9 4 9 4 4 4 4 9 4 9 9 9 9 9 4 9 9 9 9 9 9 9 4 4 4 9 9 4 9 4
## [223] 4 4 9 9 4 4 9 4 4 4 4 4 4 4 4 4 4 4 9 4 4 4 4 4 4 4 4 9 4 4 4 4 9 9 4 4 4 4 4 4
## [260] 4 4 4 4 4 4 4 4 4 4 4 9 4 9 4 4 4 9 4 4 4 4 4 4 9 9 4 9 4 4 4 4 4 4 4 4 9 4 4 4 9 4
## [297] 4 4 9 9 4 9 9 9 9 4 9 9 9 9 9 9 4 4 4 9 9 4 4 9 9 9 9 9 9 9 9 4 9 9 9 9 4
## [334] 9 4 4 4 4 9 9 9 9 9 4 4 4 9 4 4 4 4 9 9 9 9 4 9 4 9 4 9 9 9 9 9 4 9 9 9 9 4 9
## [371] 9 4 4 4 4 9 9 9
## Levels: 4 9
```

```
# Calculate and report the testing (prediction) error
mean(knn.fit != df.test.filtered[,1])
```

```
## [1] 0.04509284
```

Bonus question:

For k that are even, there is a higher chance of the same number of neighbors being close to the target

point and the majority vote won't work. For k that are odd, we can determine the closest neighbor without ambiguity. Therefore when k is even, the performance tends to be worse.

Question 2: Intrinsic Low Dimension

For this question, let's setup a simulation study. We will consider two underlying settings:

- [Setting 1] All covariate values are independently generated from a standard normal distribution
- [Setting 2] Generate the first variable X_1 from a standard normal distribution. And then for all other covariates, generate them by adding independent noise variables to X_1, i.e., $X_j = X_1 + Z_j$ where all Z_js follows iid normal with mean 0 and sd $= 0.5$.

For both settings, the outcome Y is generated using $Y = X_1 \times 0.5 + \epsilon$, with ϵ follows iid standard normal. Hence, the expected outcome depends only on the first variable. The goal of this experiment is to observe how the kNN model could be affected by the dimensionality and how an intrinsically low dimensional structure may help in this case. Complete the following questions:

a) [10 points] Write a code to generate $n = 100$ observations and $p = 2$ for each setting separately. Make your code robust such that you can easily change p to a different number without modifying other parts of the code.

```
# Under this setting, I generate all covariate values using a standard norm distribution.
# The last variable is the response variable y, defined as 0.5*table[,1] + rnorm(n)
# or the same as  $Y = X_1 \times 0.5 + \epsilon$, with $\epsilon$ follows iid standard normal.
# The other covariates are generated by rnorm(n) where n is the number of observations.
setting_1 <- function(n, p) {
  table = data.frame(matrix(ncol = p + 1, nrow = n))
  for (i in 1:p) {
    table[,i] <- rnorm(n)
  }
  table[,p+1] <- 0.5*table[,1] + rnorm(n)
  return(table)
}
```

```
# Under this setting, I generate the first variable $X_1$ from a standard normal distribution.
# And then for all other covariates, I generate them by adding independent noise variables to $X_1$,
# i.e., $X_j = X_1 + Z_j$ where all $Z_j$s follows iid normal with mean 0 and sd $= 0.5$.
# The last variable is the response variable y, defined as 0.5*table[,1] + rnorm(n)
setting_2 <- function(n, p) {
  table = data.frame(matrix(ncol = p + 1, nrow = n))
  table[,1] <- rnorm(n)
  for (i in 2:p) {
    table[,i] <- table[,1] + rnorm(n, 0, 0.5)
  }
  table[,p+1] <- 0.5*table[,1] + rnorm(n)
  return(table)
}
```

```
# Data generated
set.seed(662095561)
n=100
p=2
train_1 <- setting_1(n,p)
train_2 <- setting_2(n,p)
```

b) [10 points] Fit a 5NN regression using the generated data under each setting, and predict the same target point $x^* = c(0.5, 0.5, \ldots, 0.5)$, i.e., all covariates are 0.5. What is the true expected outcome in

this case?

```
# Generate the test data where all covariates are 0.5
x0 = matrix(rep(0.5, p), nrow = 1, ncol = 2)
install.packages("FNN", repos = "http://cran.us.r-project.org")
```

```
##
## The downloaded binary packages are in
##  /var/folders/9c/3_mgdyf12z7dvb8rt4d60nt80000gn/T//RtmpAwbrNe/downloaded_packages
```

```
library(FNN)
```

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

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

```
# Fit a $5$NN regression using the generated data under setting_1.
# The true expected outcome should be 0.5*0.5 = 0.25 (ignoring the noise).

knn.reg.fit.1 = knn.reg(train = train_1[,1:2], y = train_1[,3],
                    test = x0,
                    k = 5, algorithm = "brute")
test.pred.1 = knn.reg.fit.1$pred
test.pred.1
```

```
## [1] -0.09521057
```

```
# Fit a $5$NN regression using the generated data under setting_2
knn.reg.fit.2 = knn.reg(train = train_2[,1:2], y = train_2[,3],
                    test = x0,
                    k = 5, algorithm = "brute")
test.pred.2 = knn.reg.fit.2$pred
test.pred.2
```

```
## [1] 0.3006636
```

c) [10 points] For a simulation study, we need to repeat step b) many times to obtain the mean prediction error under each setting. Hence setup a simulation with `nsim = 300` and record the squared prediction error of each simulation run. After the simulation is completed, calculate the mean prediction error. At the end of this question, you should have two numbers, one for each setting. Which setting has a lower prediction error?

```
n = 100
p = 2
nsim = 300
allerror.1 = rep(NA, nsim)
allerror.2 = rep(NA, nsim)

for (l in 1:nsim)
  {
    # generate data
    train_1 <- setting_1(n,p)
    train_2 <- setting_2(n,p)

    knn.fit.1 = knn.reg(train = train_1[,1:2], y = train_1[,3],
```

```
                    test = x0,
                    k = 5, algorithm = "brute")

    knn.fit.2 = knn.reg(train = train_2[,1:2], y = train_2[,3],
                    test = x0,
                    k = 5, algorithm = "brute")

    # record the prediction error of this run
    # the expected response value is 0.25
    allerror.1[l] = (knn.fit.1$pred - 0.25)^2
    allerror.2[l] = (knn.fit.2$pred - 0.25)^2
  }

mean(allerror.1)
```

```
## [1] 0.2226093
```

```
mean(allerror.2)
```

```
## [1] 0.2050299
```

Setting 2 has a slightly smaller prediction error here. This is surprising to me as I was expecting setting 1 to have a smaller prediction error. Setting 1 emulates a scenario where our covariates are independent whereas there is some linear dependence between covariates in setting 2. This should lead to our model's typical regression assumptions to break down in setting 2 and normally we might not be able to apply our linear regression method to the problem. The fact that setting 2 has a smaller prediction error might be showing that KNN is a good non-parametric solution as it doesn't make many assumptions as in parametric models.

d) [5 points] Now, let's investigate the effect of dimensionality by ranging p from 2 to 50 with every integer. Before completing this question, think about what would happen to the mean prediction error for each setting? Will they increase or decrease? Which setting would increase more dramatically? Write a short paragraph to describe your expectation and why do you expect such a behavior? If you don't know the answer, then perform the next question and come back to this one once you have the result.

Increasing p would increase the mean prediction error as the number of dimensionality increases. This would reflect the curse of dimensionality. The higher the dimension (more predictors), the harder it is for kNN model to perform. I think the increase would be more dramatic for setting 2. This is because there are also random errors built into the predictors, making it more likely that the model would perform worse.

e) [20 points] Setup the simulation study to obtain the estimated prediction errors for p ranging from 2 to 50 under each setting. You have a double-loop for this simulation with one looping on p and the other one looping on `nsim`. Be careful that your target point also needs to increase its dimension. If you need more understandings of this double loop simulation, review HW4 Q1. In that question, λ is an analog to our p here, and the Bias^2/Variance/Sum are analogs to our two different settings. At the end of this question, you should again provide a plot of prediction errors with changing values of p. Does that plot matches your expectation in part d)?

Here I loop p from 1 to 49 (as indices), so when the data are generated, I have to add 1 to p. Another difference compared to the previous part is that in this case, the errors have to be stored in a matrix instead of a vector, where each column is a p value and each row is a simulation.

```
n = 100
nsim = 300

# Setting up the prediction error matrix. Each row is a simulation.
# Each column corresponds to a p value.
allerror.matrix.1 = matrix(ncol = 49, nrow = nsim)
allerror.matrix.2 = matrix(ncol = 49, nrow = nsim)
```

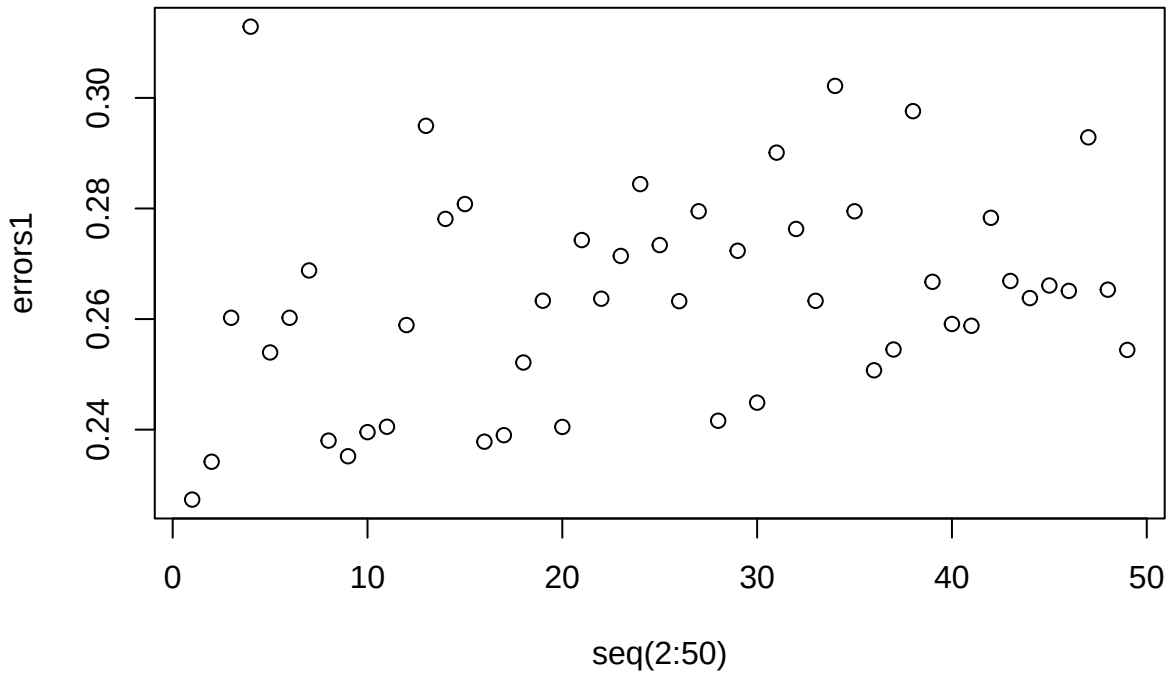
```
for (l in 1:nsim)
  for (p in 1:49) {
    {
    # generate data
    train_1 <- setting_1(n,p+1)
    train_2 <- setting_2(n,p+1)

    knn.fit.1 = knn.reg(train = train_1[,1:2], y = train_1[,3],
                        test = x0,
                        k = 5, algorithm = "brute")

    knn.fit.2 = knn.reg(train = train_2[,1:2], y = train_2[,3],
                        test = x0,
                        k = 5, algorithm = "brute")

    # record the prediction error of this run
    # the truth is 0.25
    allerror.matrix.1[l,p] = (knn.fit.1$pred - 0.25)^2
    allerror.matrix.2[l,p] = (knn.fit.2$pred - 0.25)^2
  }
}
```

```
# Here are the mean prediction errors for every p values
errors1 = rep(NA, 49)
errors2 = rep(NA, 49)
for (i in 1:49) {
  errors1[i] = mean(allerror.matrix.1[,i])
}

for (i in 1:49) {
  errors2[i] = mean(allerror.matrix.2[,i])
}

dim(errors1)
```

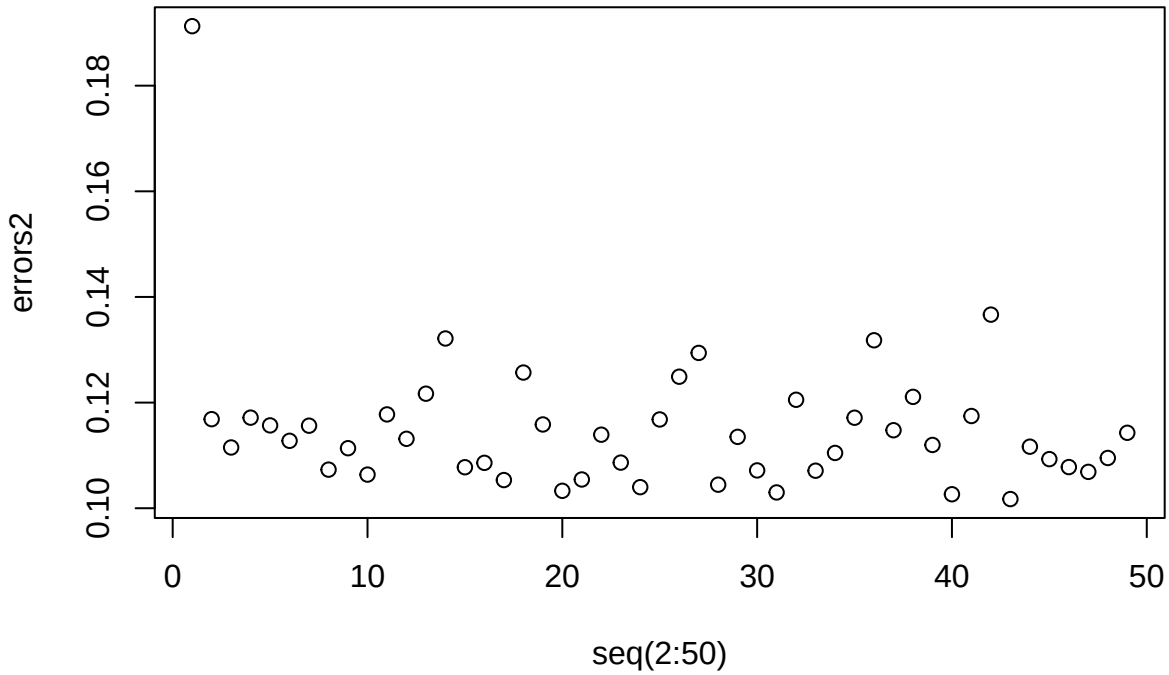
```
## NULL
```

Here are the plots of p values against mean prediction errors. There is no clear trend in either setting. This is not the same as my expectation, where I expected that the mean prediction error would increase for higher p. My regression line at the end also shows that the slopes in either case are very close to 0. So there is no evidence that increasing p leads to an increased mean prediction error.

```
#Plotting

plot(x=seq(2:50), y=errors1)
```




```
plot(x=seq(2:50), y=errors2)
```




```
check_trend_1 <- lm(errors1 ~ seq(2:50))
summary(check_trend_1)
```

```
##
## Call:
## lm(formula = errors1 ~ seq(2:50))
##
## Residuals:
##       Min        1Q    Median        3Q       Max
```

```
## -0.027289 -0.017582 -0.002245  0.011569  0.057068
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.2542513  0.0054157  46.947   <2e-16 ***
## seq(2:50)   0.0003906  0.0001885   2.072   0.0438 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.01867 on 47 degrees of freedom
## Multiple R-squared:  0.08367,    Adjusted R-squared:  0.06418
## F-statistic: 4.292 on 1 and 47 DF,  p-value: 0.04381
```

```
check_trend_2 <- lm(errors2 ~ seq(2:50))
summary(check_trend_2)
```

```
##
## Call:
## lm(formula = errors2 ~ seq(2:50))
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.012921 -0.008429 -0.002694  0.003937  0.070895
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.1205851  0.0039323  30.665   <2e-16 ***
## seq(2:50)   -0.0002184  0.0001369  -1.595    0.117
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.01355 on 47 degrees of freedom
## Multiple R-squared:  0.05137,    Adjusted R-squared:  0.03118
## F-statistic: 2.545 on 1 and 47 DF,  p-value: 0.1173
```