# Stat 432 Homework 6

Sharvi Tomar (stomar2)

10/02/21

## 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.

```
# Loading the Handwritten Digit Data
load("/Users/sharvitomar/Desktop/Sem1/432/zipdata.RData")

# subset the data to include only two digits: 4 and 9
train = data.frame(zip.train)
train = subset(train, X1 == 9 | X1 == 4)

# subset the data to include only two digits: 4 and 9
test = data.frame(zip.test)
test = subset(test, X1 == 9 | X1 == 4)
```

- [5 points] Use repeated 5-fold cross-validation, with 3 repeats.

```
# Using repeated 5-fold cross-validation, with 3 repeats
library(caret)
```

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

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

```
control <-
  trainControl(method = "repeatedcv",
               repeats = 3,
               number = 5)
```

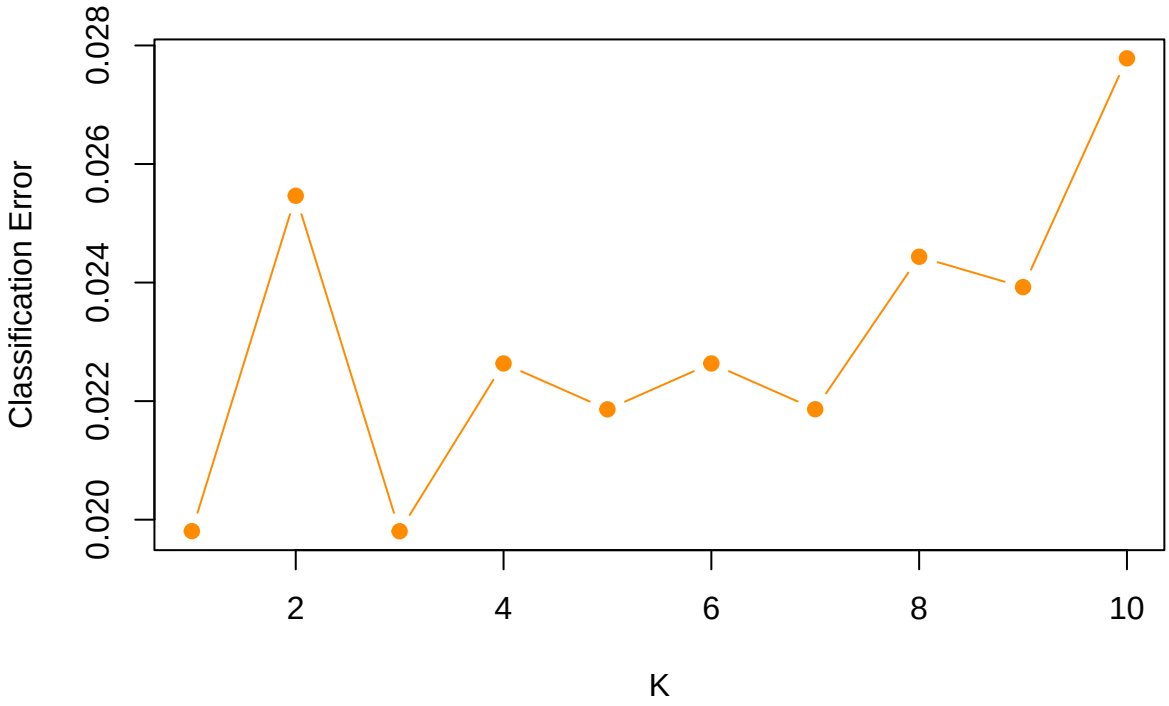
- [5 points] For $k$, use all integers from 1 to 10

```
library(e1071)
set.seed(667346304)

# Performing cross-validation of training data
knn.cvfit <- train(
  y ~ .,
  method = "knn",
  data = data.frame("x" = train[, 2:257], "y" = as.factor(train[, 1])),
  tuneGrid = data.frame(k = seq(1, 10, 1)),      # Setting values for k to be all integers from 1-10
  trControl = control
)
```

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.

```
# Plot that shows k against the cross-validation error
plot(
  knn.cvfit$results$k,
  1 - knn.cvfit$results$Accuracy,
  xlab = "K",
  ylab = "Classification Error",
  type = "b",
  pch = 19,
  col = "darkorange"
)
```




The best tuning parameter is $k$=1 and $k$=3 as these are values of $k$ corresponding to the least classification error.

- [5 points] Present the confusion table for the testing data

```
library(class)
knn.fit <- knn(train[, 2:257], test[, 2:257], train[, 1], k = 5)
xtab = table(knn.fit, test[, 1])

library(caret)
# Confusion table for the testing data
confusionMatrix(xtab)
```

```
## Confusion Matrix and Statistics
##
##
## knn.fit   4   9
##       4 186   3
##       9  14 174
##
##                Accuracy : 0.9549
##                  95% CI : (0.9288, 0.9735)
##     No Information Rate : 0.5305
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.9098
```

```
##
##  Mcnemar's Test P-Value : 0.01529
##
##             Sensitivity : 0.9300
##             Specificity : 0.9831
##          Pos Pred Value : 0.9841
##          Neg Pred Value : 0.9255
##              Prevalence : 0.5305
##          Detection Rate : 0.4934
##    Detection Prevalence : 0.5013
##       Balanced Accuracy : 0.9565
##
##        'Positive' Class : 4
##
```

- [5 points] Calculate and report the testing error

For evaluating classification models, we can use metrics such as Classification Accuracy.

```
confusionMatrix(xtab)$overall["Accuracy"]
```

```
##  Accuracy
## 0.9549072
```

The Classification Accuracy for the model is 95.49%.

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?

The above problem is a binary classification problem using KNN, hence, it is advisable to take odd values of $k$ to avoid the ties i.e. two classes labels achieving the same score.

If you are using K and you have an even number of classes (e.g. 2) it is a good idea to choose a K value with an odd number to avoid a tie. And the inverse, use an even number for K when you have an odd number of classes.

## Question 2: Intrinsic Low Dimension

For this question, let's setup a simulation study. We will consider two settings. For both settings, the outcome $Y$ is generated using $Y = X_1 \times 0.5 + \epsilon$, with $\epsilon$ follows iid standard normal. Hence, the expected outcome depends only on the first variable. The goal of this experiment is to observe how the $k$NN model could be affected by the dimensionality and how an intrinsically low dimensional structure may help in this case. Complete the following questions:

- [Setting 1] All covariate values are independently generated from a standard normal distribution

```
# Setting-1 Data generation
set.seed(667346304)

library(MASS)
setting1 <- function(n, p) {
  X <- matrix(mvrnorm(n, rep(0, p), diag(p)), nrow = n, ncol = p)
  epsilon = rnorm(n, 0, 1)
```

```
  y = X[, 1] * 0.5 + epsilon
  X<- cbind(y, X)
  return(X)
}
```

- [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 $Z_j$s follows iid normal with mean 0 and sd = 0.5.

```
# Setting-2 Data generation
set.seed(667346304)

setting2 <- function(n, p) {
  x_1 = rnorm(n, 0, 1)
  if (p == 1) {
    X = as.matrix(x_1)
  }
  else if (p == 2) {
    X = matrix(x_1 + mvrnorm(n, rep(0, p - 1), 0.5),
               nrow = n,
               ncol = p - 1)
    X <- cbind(x_1, X)
  }
  else{
    X = matrix(x_1 + mvrnorm(n, rep(0, p - 1), diag(rep(0.5, p - 1))),
               nrow = n,
               ncol = p - 1)
    X <- cbind(x_1, X)
  }
  epsilon = rnorm(n, 0, 1)
  y = X[, 1] * 0.5 + epsilon
  X<- cbind(y, X)
  return(X)
}
```

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.

```
# Generating data from setting-1
data_1 = setting1(100, 2)
# Generating data from setting-2
data_2 = setting2(100, 2)
```

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?

```
library(FNN)
```

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

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

```r
# Test data
test_data = rep(0.5, 2)

# Fitting a 5NN regression model on setting-1 data
knn.fit1 = knn.reg(
  train = data_1[1:100,2:3],
  test = test_data,
  y = data_1[1:100, 1],
  k = 5,
  algorithm = "brute"
)

# Fitting a 5NN regression model on setting-2 data
knn.fit2 = knn.reg(
  train = data_2[1:100,2:3],
  test = test_data,
  y = data_2[1:100, 1],
  k = 5,
  algorithm = "brute"
)

# The true expected outcome in the case
y_true= 0.5*0.5+0
```

The true expected outcome in the case $x^* = c(0.5, 0.5, \dots, 0.5)$ would be $Y = X_1 \times 0.5 + \epsilon$, 0.25

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?

```r
# Number of simulations
nsim = 300

# Vector to store predicted values
allerror_1 = rep(NA, nsim)
allerror_2 = rep(NA, nsim)

for (l in 1:nsim)
{
  # Generating data from setting-1
  data_1 = setting1(100, 2)
  X_1 = data_1[1:100,2:3]
  y_1 = data_1[1:100, 1]

  # Generating data from setting-2
  data_2 = setting2(100, 2)
  X_2 = data_2[1:100,2:3]
```

```
  y_2 = data_2[1:100, 1]

  # "5-NN" regression using the FNN package for setting-1 data
  knn.fit_1 = knn.reg(
    train = X_1,
    test = test_data,
    y = y_1,
    k = 5,
    algorithm = "brute"
  )

  # "5-NN" regression using the FNN package for setting-2 data
  knn.fit_2 = knn.reg(
    train = X_2,
    test = test_data,
    y = y_2,
    k = 5,
    algorithm = "brute"
  )

  # Recording prediction error of this run, the truth is 0.25
  allerror_1[l] = (knn.fit_1$pred - 0.25) ^ 2
  allerror_2[l] = (knn.fit_2$pred - 0.25) ^ 2
}

# the prediction error
mean(allerror_1)
```

```
## [1] 0.2255192
```

```
mean(allerror_2)
```

```
## [1] 0.1994046
```

The mean prediction error is lower for setting-2 in comparison to prediction error for setting-1 data.

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.

I would expect an increase in the prediction errors with increase in dimensionality.

The mean prediction error of setting-1 data should vary more drastically than that of setting-2. This is due to the fact that all the variates in setting 1 are independent whereas for setting-2 the variates are correlated and hence there possibly exists a lower dimension(than the present dimension) where all the variates are independent.Since there exists a lower dimensional sub-space for setting-2 but not for setting-1, setting-2 data prediction errors should vary less drastically.

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, $\lambda$ 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)?

```
# number of simulations
nsim = 300
p_grid = seq(2, 50, 1)

# vector to store predicted values
allerror_1 = matrix(NA, nsim, 49)
allerror_2 = matrix(NA, nsim, 49)

for (p_val in p_grid) {
  for (l in 1:nsim)
  {
    # generate data
    test_data = rep(0.5, p_val)
    data_1 = setting1(100, p_val)
    X_1 = data_1[1:100,2:(p_val+1)]
    y_1 = data_1[1:100, 1]

    data_2 = setting2(100, p_val)
    X_2 = data_2[1:100,2:(p_val+1)]
    y_2 = data_2[1:100, 1]

  # "5-NN" regression using the FNN package for setting-1 data
    knn.fit_1 = knn.reg(
      train = X_1,
      test = test_data,
      y = y_1,
      k = 5,
      algorithm = "brute"
    )

  # "5-NN" regression using the FNN package for setting-2 data
      knn.fit_2 = knn.reg(
      train = X_2,
      test = test_data,
      y = y_2,
      k = 5,
      algorithm = "brute"
    )

    # record the prediction error of this run, the truth is 0.25
    allerror_1[l, p_val-1] = (knn.fit_1$pred - 0.25) ^ 2
    allerror_2[l, p_val-1] = (knn.fit_2$pred - 0.25) ^ 2
  }
}
# the mean prediction error for each value of p
colMeans(allerror_1)
```

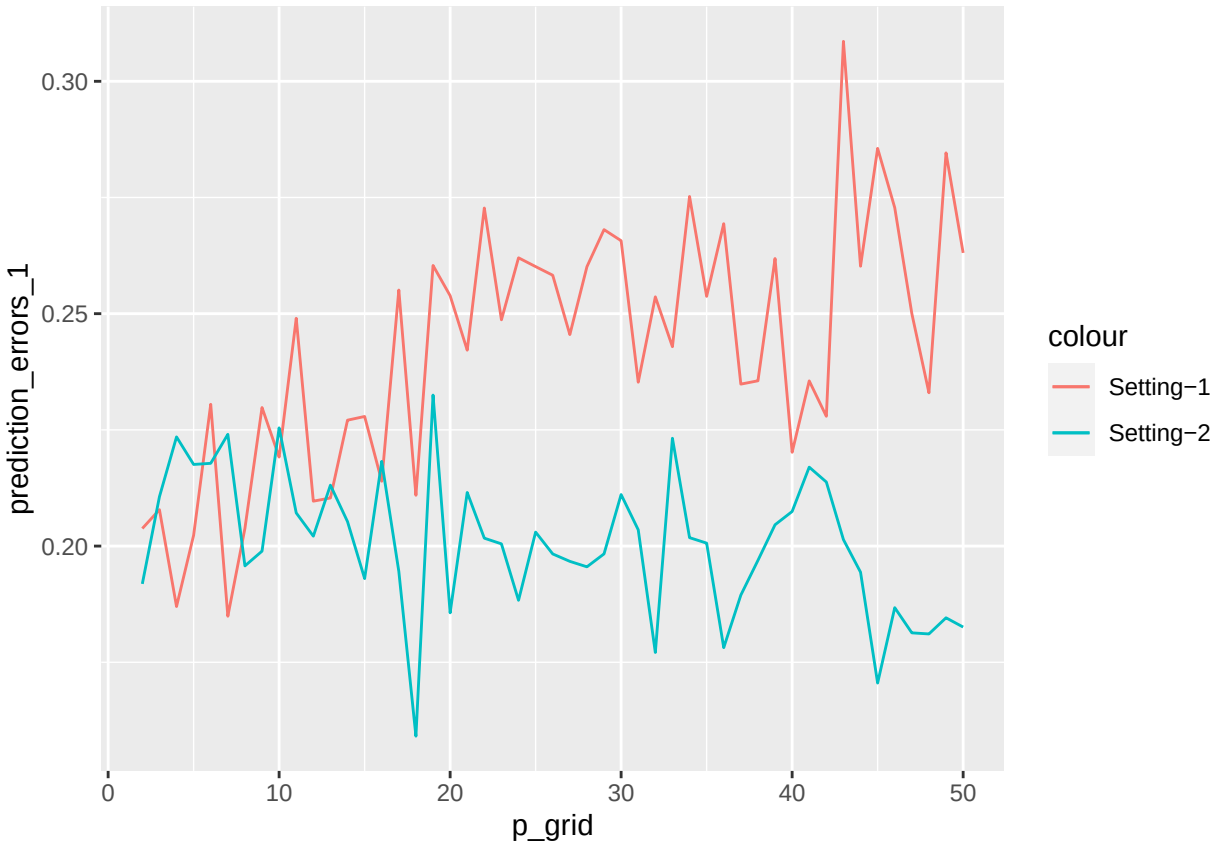
```
##  [1] 0.2037796 0.2078283 0.1869921 0.2023235 0.2304993 0.1848990 0.2038142
##  [8] 0.2298029 0.2191494 0.2490022 0.2096594 0.2103454 0.2270865 0.2278805
## [15] 0.2139447 0.2550864 0.2109280 0.2603734 0.2538677 0.2421475 0.2727303
## [22] 0.2486794 0.2620205 0.2601285 0.2582674 0.2454966 0.2601055 0.2680881
## [29] 0.2656974 0.2352554 0.2536253 0.2428931 0.2752228 0.2537266 0.2693566
## [36] 0.2348398 0.2355600 0.2618664 0.2201738 0.2355425 0.2279247 0.3085943
## [43] 0.2602104 0.2855660 0.2727951 0.2500146 0.2329829 0.2845995 0.2630740
```

```
colMeans(allerror_2)
```

```
##  [1] 0.1918634 0.2106135 0.2234879 0.2175601 0.2178013 0.2240386 0.1957330
##  [8] 0.1989042 0.2254298 0.2071317 0.2021309 0.2131006 0.2052567 0.1930005
## [15] 0.2182184 0.1945411 0.1590987 0.2324857 0.1856338 0.2115276 0.2016931
## [22] 0.2004723 0.1883268 0.2030017 0.1983070 0.1967093 0.1955362 0.1983187
## [29] 0.2110834 0.2034974 0.1771033 0.2231943 0.2018121 0.2006284 0.1781446
## [36] 0.1894445 0.1969209 0.2045820 0.2074399 0.2169799 0.2137804 0.2013840
## [43] 0.1943793 0.1705218 0.1867448 0.1813274 0.1810899 0.1845643 0.1825704
```

```
# plot of prediction errors with changing values of p
prediction_errors_1=colMeans(allerror_1)
prediction_errors_2=colMeans(allerror_2)

df=data.frame(p_grid,prediction_errors_1,prediction_errors_2)
library(ggplot2)
ggplot(df, aes(x=p_grid)) +
  geom_line(aes(y = prediction_errors_1, color= "Setting-1")) +
  geom_line(aes(y = prediction_errors_2, color= "Setting-2"))
```




From the plot, I can see an increasing trend for the mean prediction errors with increase in dimensionality for setting-1 data as per expectation. However, the similar trend cant be seen for setting-2 data.

The dramatic variations in prediction errors over the range of p values (dimensions of data) in setting-1 data as compared to setting-2 data is as per expectation. This is because the variates in setting-2 data are co-related and hence, a lower dimension exists where all variates are independent. There is possibly (approximately) a lower dimensional representation of the data so that when you evaluate the distance on the high-dimensional space, it is just as effective as working on the low dimensional space.

This is not the case for setting-1 data where all the variates are independent.