STAT 542: Homework 2

Name: Arka Mitra (arkam2)

Due: Monday 11:59 PM CT, Feb 16

Contents

About HW2

For this HW, we mainly try to understand the KNN method in both classification and regression settings and use it to perform several real data examples. Tuning the model will help us understand the bias-variance trade-off. A slightly more challenging task is to code a KNN method yourself. For that question, you cannot use any additional package to assist the calculation.

There is an important package, `ElemStatLearn`, which is the package associated with the ESL textbook for this course. Unfortunately the package is currently discontinued on CRAN. You can install an earlier version of this package by using

```
#Could not get devtools to work, so installed the ElemStatLearn from source
#  require(devtools)
#  install_version("ElemStatLearn", version = "2015.6.26.2", repos = "http://cran.us.r-project.org")
```

```
packageurl <- "https://github.com/cran/ElemStatLearn/archive/2015.6.26.2.tar.gz"
install.packages(packageurl, repos=NULL, type="source")
```

And of course you will have to install the `devtools` package if you don't already have it.

Question 1 [20 Points] KNN Classification (Diabetes)

Load the Pima Indians Diabetes Database (`PimaIndiansDiabetes`) from the `mlbench` package. If you don't already have this package installed, use the following code. It also randomly split the data into training and testing. You should preserve this split in the analysis.

Read the documentation of this dataset here and make sure that you understand the goal of this classification problem.

Use a grid of k values (every integer) from 1 to 20. Fit the KNN model using `Diab.train` and calculate both training and testing errors. For the testing error, use `Diab.test`. Plot the two errors against the corresponding k values. Make sure that you differentiate them using different colors/shapes and add proper legends. Does the plot match (approximately) our intuition of the bias-variance trade-off in terms of having an U-shaped error? What is the optimal k value based on this result?

```
# install.packages("mlbench") # run this line if you don't have the package
library(mlbench)
library(class)
data(PimaIndiansDiabetes)

set.seed(2)
trainid = sample(1:nrow(PimaIndiansDiabetes), nrow(PimaIndiansDiabetes)/2)
testid = -trainid
Diab.train = PimaIndiansDiabetes[trainid, ]
Diab.test = PimaIndiansDiabetes[-trainid, ]
allk = c(1:20); accuracies = matrix(nrow=2,ncol=20)
for (i in 1:20){
  knn_fit = knn(Diab.train[,1:8], Diab.test[,1:8], Diab.train[,9], k=allk[i])
  accuracy = mean((as.numeric(knn_fit) - as.numeric(Diab.test[,9]))^2)
  accuracies[1,i] <- accuracy
  accuracy = mean((as.numeric(knn_fit) - as.numeric(Diab.train[,9]))^2)
  accuracies[2,i] <- accuracy
}
plot(allk,accuracies[1,], type="l", col="red", lwd=2, xlab="No. of K-neighbors", ylab="Accuracy", mai
par(new=TRUE)
plot(allk,accuracies[2,], type="l", col="blue", lwd=2, ylab='',xlab='')
legend(12, 0.44, legend=c("Testing Error", "Training Error"),col=c("red", "blue"), lty=1, cex=1.5)
```

Although it is slightly difficult to ascertain as the number of k-neighbors is still very low, the bias-variance trade-off evolution does show familiar U-shaped nature in the testing error.

From visual examination, optimal $k = 12$

Question 2 [20 Points] KNN Classification (Handwritten Digit)

Load the Handwritten Digit Data (`zip.train` and `zip.test`) from the `ElemStatLearn` package. Use a grid of k values (every integer) from 1 to 20. And we need to select the best tuning using the `caret` package. To

perform this automatic tuning, you need to

- Specify the type of cross-validation using the `trainControl()` function. We need to use 3 fold cross-validation.
- Specify a grid of tuning parameters. This can be done using `expand.grid(k = c(1:20))`
- Train the cross-validation using the `train()` function

For details, read the documentation at here to learn how to use the `trainControl()` and `train()` functions. Some examples can also be found at here. Apply the function to the `zip.train` data with our choice of k grid.

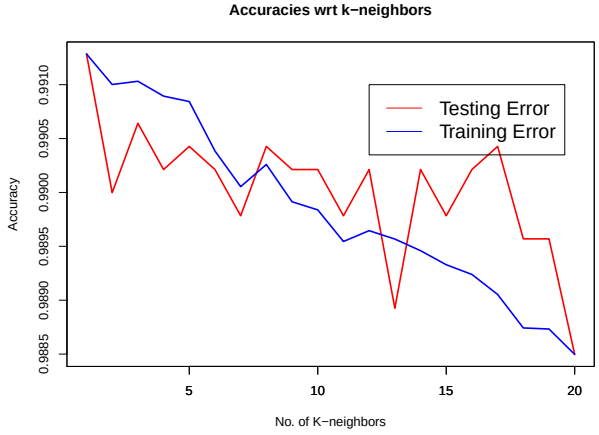
Fit the KNN model using **a randomly selected subset of `zip.train`, with size 500** as the training data and calculate both training and testing errors. Make sure to set random seed so that your result can be replicated. Plot the two errors against the corresponding k values. Make sure that you differentiate them using different colors/shapes and add proper legends. What is the optimal k value? Does the plot match our intuition of the bias-variance trade-off in terms of having an U-shaped error? If not, which theoretical result we introduced in the lecture can be used to explain it? Provide your explanation of the results.

```
# Handwritten Digit Recognition Data
library(ElemStatLearn)
# the first column is the true digit
# dim(zip.train)
library(caret)

set.seed(70)
rind = as.integer(runif(500)*dim(zip.test)[1])
train.x = zip.train[rind, -1]
test.x = zip.test[-rind,-1]
train.y = as.factor(zip.train[rind, 1])

#cross-validation using caret
TrainData=data.frame(train.x)
knn.fit = train(TrainData, train.y,
                method = "knn",
                tuneGrid = expand.grid(k = c(1:20)),
                trControl = trainControl(method = "cv",number = 3))
knn.fit
```

```
## k-Nearest Neighbors
##
## 500 samples
## 256 predictors
##  10 classes: '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold)
## Summary of sample sizes: 333, 334, 333
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    1  0.9299714  0.9204953
##    2  0.8919871  0.8772315
##    3  0.8979751  0.8840279
##    4  0.8839430  0.8678785
```

```
##     5  0.8819710  0.8655533
##     6  0.8719549  0.8540843
##     7  0.8699709  0.8518499
##     8  0.8579708  0.8382172
##     9  0.8599668  0.8405117
##    10  0.8479427  0.8267120
##    11  0.8520068  0.8313552
##    12  0.8419787  0.8199328
##    13  0.8239906  0.7993034
##    14  0.8159825  0.7900241
##    15  0.8039944  0.7762052
##    16  0.8060025  0.7786281
##    17  0.7959863  0.7668897
##    18  0.7779621  0.7462871
##    19  0.7739822  0.7415446
##    20  0.7659741  0.7324331
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 1.
```

```r
#Testing sample
TestData = data.frame(zip.test[,-1])
TestClasses = as.factor(zip.test[,1])
# the confusion matrix
pred.y=predict(knn.fit, newdata = TestData)

  allk = c(1:20); accuracies = matrix(nrow=2,ncol=20)
  for (i in 1:20){
    knn_fit = knn(train.x, test.x, zip.train[rind, 1], k=allk[i])
    accuracy = mean(knn_fit != zip.train[rind, 1])
    accuracies[1,i] <- accuracy
    accuracy = mean(knn_fit != test.x)
    accuracies[2,i] <- accuracy
  }
  plot(allk,accuracies[1,], type="l", col="red", lwd=2, xlab="No. of K-neighbors", ylab="Accuracy", mai
  par(new=TRUE)
  plot(allk,accuracies[2,], type="l", col="blue", lwd=2, ylab='',xlab='')
  legend(12, 0.991, legend=c("Testing Error", "Training Error"),col=c("red", "blue"), lty=1, cex=1.5)
```



The U-shaped curve is not observed here. The variation of local variance around a given point is very small in this problem, so low bias allows for a good estimator. Hence, 1NN is a good choice.

Question 3 [40 Points] Write your own KNN for regression

For this question, you **cannot** use (load) any additional `R` package. Complete the following steps.

a. [25 points] Generate the covariate of $n = 1000$ training data, with $p = 5$ from independent standard Normal distribution. Then, generate Y from

$$Y = X_1 + 0.5 \times X_2 - X_3 + \epsilon,$$

with i.i.d. standard normal error ϵ. Write a function `myknn(xtest, xtrain, ytrain, k)` that fits a KNN model and predict multiple target points `xtest`. Here `xtrain` is the training dataset covariate value, `ytrain` is the training data outcome, and `k` is the number of nearest neighbors. Use the euclidean distance to evaluate the closeness between two points.

Test your code using the first 500 observations as the training data and the rest as testing data. Predict the Y values using your KNN function with `k = 5`. Evaluate the prediction accuracy using mean squared error

$$\frac{1}{N}\sum_i (y_i - \widehat{y}_i)^2$$

b. [15 Points] Consider k being all integers from 1 to 10. Use the degrees of freedom as the horizontal axis. Demonstrate your results in a single, easily interpretable figure with proper legends. What is your optimal tuning parameter and the associated degrees of freedom?

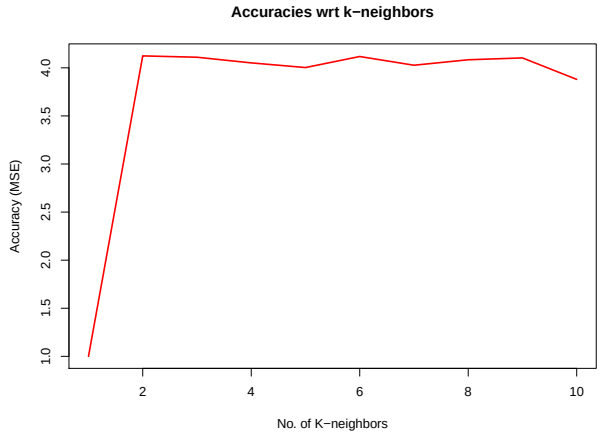
```r
set.seed(300)

#Generating covariate given by Y =  = X1 + 0.5*X2 - X3 + e
n=1000; p=5
X = matrix( rnorm(n*p,mean=0,sd=1), n, p)
e = rnorm(1000)
Y = X[,1] + 0.5*X[,2] - X[,3] + e

#Generating testing and training datasets.
xtrain = X[1:500,]
ytrain = Y[1:500]
xtest = X[500:1000,]

# This is my kNN model

# Euclidean Distance
euc_dist <- function(y1, y2) {
    return(sqrt(sum((y1 - y2)^2)))
}


# knn_model()
myknn <- function(xtest, xtrain, ytrain, k) {

  # Create return value, predictions
    predictions <- rep(NA, nrow(xtest))
  #LOOP-1
```

```
  for(i in c(1:nrow(xtest))){   #looping over each record of test data
    eu_dist =c()          #eu_dist empty  vector

    #LOOP-2-looping over train data
    for(j in c(1:nrow(xtrain))){

      #adding euclidean distance b/w test data point and train data to eu_dist vector
      eu_dist <- c(eu_dist, euc_dist(xtest[i,], xtrain[j,]))

    }
        # Sort distances maintaining index from min to max
        distances_sorted <- sort(eu_dist, index.return=T)

        # Save types from xtrain using the indexes in distances_sorted
        nn_k <- xtrain[distances_sorted$ix[0:k], ]

        # Create frequency table of types in nn_k
        freq_table <- table(nn_k[ , ncol(nn_k)])

        # Sort frequency table from min to max
        freq_table_sorted <- sort(freq_table, decreasing=T)

        # If there is a tie for most frequent type, take a sample of the types
        # in in nn_k; else, enter the type of first element of sorted frequency
        # table in return value
        if(freq_table_sorted[1] == freq_table_sorted[2]) {
            predictions[i] <- sample(nn_k[ , ncol(nn_k)])
        } else {
            predictions[i] <- names(freq_table_sorted[1])
        }
  }
  mse_base = 0
  for (i in 1:length(ytrain)) mse_base = mse_base+(ytrain[i]-predictions[i])^2
  acc = 1/length(ytrain)*mse_base
  return(acc)
  #return(predictions)
} # end knn_model()

# accuracy <- function(ytest, pred) {
#   mse_base = 0
#   for (i in 1:length(ytest)) mse_base = mse_base+(ytest[i]-pred[i])^2
#   acc = 1/length(ytest)*mse_base
#   return(acc)
# }

print(myknn(xtest, xtrain, ytrain, k=5))
```

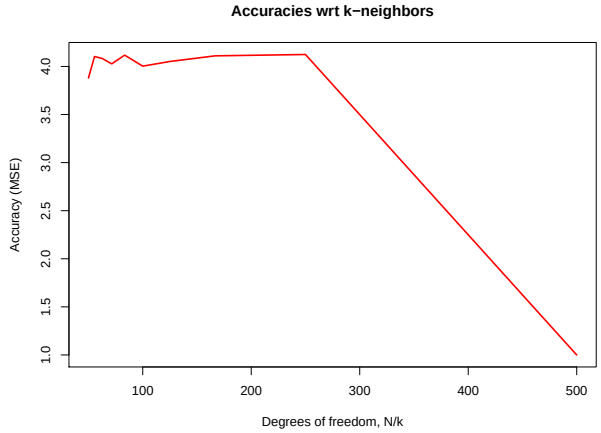
```
## [1] 4.055872
```

```
dof = c(1:10); acc = c(1:10)
for (i in 2:length(dof)){
  #pred.y = myknn(xtest, xtrain, ytrain, k=i)
  acc[i] = myknn(xtest, xtrain, ytrain, k=i)
```

```
}

plot(dof,acc, type="l", col="red", lwd=2, xlab="No. of K-neighbors", ylab="Accuracy (MSE)", main="Accura
```




```
plot(500/dof,acc, type="l", col="red", lwd=2, xlab="Degrees of freedom, N/k", ylab="Accuracy (MSE)", ma
```



The optimal tuning parameter is k, with degrees of freedom, n/k = 500/k.

Question 4 [20 Points] Curse of Dimensionality

Let's consider a high-dimensional setting. Keep the model the same as question 3. We consider two cases that both generate an additional set of 95 covariates:

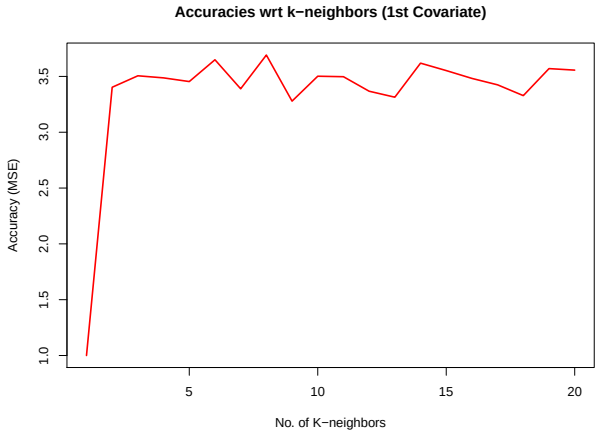
- Generate another 95-dimensional covariate with all independent standard Gaussian entries
- Generate another 95-dimensional covariate using the formula $X^T A$, where X is the original 5-dimensional vector, and A is a 5×95 dimensional (fixed) matrix that remains the same for all observations. You should generate A only once using i.i.d. uniform $[0, 1]$ entries.

Make sure that you set seed when generating these covariates. Fit KNN in both settings (with the total of 100 covariates) and select the best k value. Answer the following questions

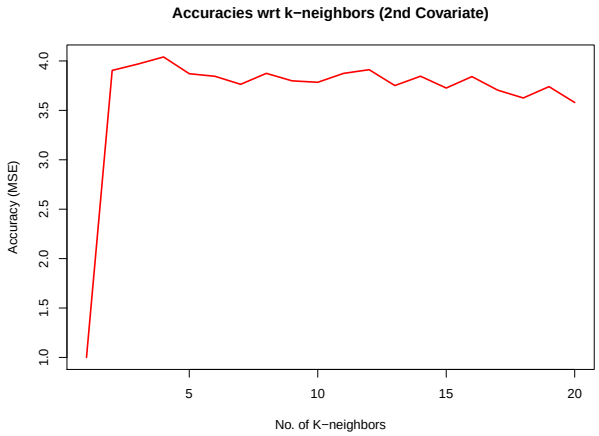
- For each setting, what is the best k and the best mean squared error for prediction?
- In which setting kNN performs better? Should this be expected? Why?

```
#Generating covariate given by Y =  = X1 + 0.5*X2 - X3 + e
set.seed(300)
n= 1000; p=5
X = matrix( rnorm(5*n,mean=0,sd=1), p, n)
N = matrix( rnorm(95*n,mean=0,sd=1), 95, n)
X1 = rbind(X,N)
Y = X[,1] + 0.5*X[,2] - X[,3] + e
X1 = t(X1)

A = matrix( runif(p*95), p, 95)
M = t(X) %*% A
X2 = rbind(X,t(M))
X2 = t(X2)

#Generating testing and training datasets.
xtrain1 = X1[1:500,]
xtest1 = X1[500:1000,]
xtrain2 = X2[1:500,]
xtest2 = X2[500:1000,]
ytrain = Y[1:500]

# This is my kNN model

# Euclidean Distance
euc_dist <- function(y1, y2) {
    return(sqrt(sum((y1 - y2)^2)))
}


# knn_model()
myknn <- function(xtest, xtrain, ytrain, k) {

  # Create return value, predictions
    predictions <- rep(NA, nrow(xtest))
  #LOOP-1
  for(i in c(1:nrow(xtest))){   #looping over each record of test data
    eu_dist =c()          #eu_dist empty  vector

    #LOOP-2-looping over train data
    for(j in c(1:nrow(xtrain))){

      #adding euclidean distance b/w test data point and train data to eu_dist vector
      eu_dist <- c(eu_dist, euc_dist(xtest[i,], xtrain[j,]))

    }
        # Sort distances maintaining index from min to max
        distances_sorted <- sort(eu_dist, index.return=T)

        # Save types from xtrain using the indexes in distances_sorted
        nn_k <- xtrain[distances_sorted$ix[0:k], ]
```

```
        # Create frequency table of types in nn_k
        freq_table <- table(nn_k[ , ncol(nn_k)])

        # Sort frequency table from min to max
        freq_table_sorted <- sort(freq_table, decreasing=T)

        # If there is a tie for most frequent type, take a sample of the types
        # in in nn_k; else, enter the type of first element of sorted frequency
        # table in return value
        if(freq_table_sorted[1] == freq_table_sorted[2]) {
            predictions[i] <- sample(nn_k[ , ncol(nn_k)])
        } else {
            predictions[i] <- names(freq_table_sorted[1])
        }
  }
  mse_base = 0
  for (i in 1:length(ytrain)) mse_base = mse_base+(ytrain[i]-predictions[i])^2
  acc = 1/length(ytrain)*mse_base
  return(acc)
  #return(predictions)
} # end knn_model()


dof = c(1:20); acc1 = c(1:20); acc2 = c(1:20)
for (i in 2:length(dof)){
  acc1[i] = myknn(xtest1, xtrain1, ytrain, k=i)
  acc2[i] = myknn(xtest2, xtrain2, ytrain, k=i)
}

plot(dof,acc1, type="l", col="red", lwd=2, xlab="No. of K-neighbors", ylab="Accuracy (MSE)", main="Accu
```




```
plot(dof,acc2, type="l", col="red", lwd=2, xlab="No. of K-neighbors", ylab="Accuracy (MSE)", main="Accu
```



1st case: Best k: 9, MSE: 3.32 2nd case: Best k: 20, MSE: 3.76

kNN works better in Case 2. This is to be expected. In higher dimensional cases, increasing model complexity (k) improves effective number of parameters (N/k).