Lab 2: k-Nearest Neighbors

PSTAT 131/231, Fall 2023

Learning Objectives

- k-Nearest Neighbors
- Training/Test split
- Run k-Nearest Neighbors methods in classification and regression problems
- Compute Training and Test error rates (in classification) and MSE (in regression)

---

1. Install packages and obtatin `Carseats` dataset

In this section, we will primarily focus on performing k-Nearest Neighbors for classification (using package `class`, which contains various functions for classification, including k-Nearest Neighbor) and regression (using package `FNN`).

The following packages are needed to assist our analysis:

```
# install.packages("ISLR")
# install.packages("tidyverse")
# install.packages("class")
# install.packages("FNN")

# Load libraries
library(ISLR)
library(tidyverse)
library(class)
library(FNN)
```

The dataset `Carseats` in package `ISLR` is adopted to make an example task.

```
# Obtain Carseats from ISLR using data()
data(Carseats)
# Check the structure by str()
str(Carseats)
# Get dataset info
?Carseats
```

2. k-Nearest Neighbors (a.k.a. kNN, k-NN, knn) classification

As a reminder, `Carseats` is a simulated data set containing sales of child car seats at 400 different stores on 11 features (3 discrete and 8 numerical). For the classification goal, we create a new feature `High` as the response variable following the rule:

$$High = \begin{cases} \text{No}, & \text{if Sales} \leq \text{median(Sales)} \\ \text{Yes}, & \text{if Sales} > \text{median(Sales)} \end{cases}$$

In this section, we will work on the binary response variable `High`, as well as other continuous variables except for `Sales`. The goal is to investigate the relationship between `High` and all continuous variables but `Sales`.

To achieve this goal, we will delete three categorical variables (`ShelveLoc`, `Urban` and `US`) and the continuous `Sales`[1] from the original data. We call the resulting dataset `seats`:

```
# Create the binary response variable High, drop Sales and 3 discrete independent
# variables as well. Call the new dataset seats
seats = Carseats %>%
    mutate(High=as.factor(ifelse(Sales <= median(Sales), "Low", "High"))) %>%
    select(-Sales, -ShelveLoc, -Urban, -US)

# Check column names of seats
colnames(seats)
str(seats)

# Another way to create seats using quantile()
seats = Carseats %>%
   mutate(High=as.factor(ifelse(Sales <= quantile(Sales, probs=0.5), "Low", "High"))) %>%
   select(-Sales, -ShelveLoc, -Urban, -US)
```

(a). Training/Testing split

Given a data set, the use of a particular statistical learning method is warranted if it results in a low test (not training) error. The test error can be easily calculated if a designated test set is available. Therefore, before performing kNN on the data, we first sample 50% of the observations as a training set, and the other 50% as a test set. Note that we set a random seed before applying `sample()` to ensure reproducibility of results.

```
# Set random seed
set.seed(333)

# Sample 50% observations as training data
train = sample(1:nrow(seats), 200)
seats.train = seats[train,]

# The rest 50% as test data
seats.test = seats[-train,]
```

For later convenience purposes, we create `XTrain`, `YTrain`, `XTest` and `YTest`. `YTrain` and `YTest` are response vectors from the training set and the test set. `XTrain` and `XTest` are design matrices[2].

```
# YTrain is the true labels for High on the training set
# XTrain is the standardized design matrix
YTrain = seats.train$High
XTrain = seats.train %>% select(-High) %>% scale(center = TRUE, scale = TRUE)
?scale

# YTest is the true labels for High on the test set, Xtest is the design matrix
YTest = seats.test$High
XTest = seats.test %>% select(-High) %>% scale(center = TRUE, scale = TRUE)
```

(b). Train a kNN classifier and calculate error rates

- `knn()` function in the package `class` can be used to train a kNN classifier. This function works rather differently from the other model-fitting functions that we have encountered thus far. Rather than a

[1] Note: we delete `Sales` from the explanatory variables is due to the fact that `High` is derived from it.

[2] Design matrix, also known as regressor matrix or model matrix, is a matrix of values of explanatory variables, often denoted by X. Each row represents an individual observation, with the successive columns corresponding to the variables and their specific values for that object.

two-step approach in which we first fit the model and then we use the model to make predictions, `knn()` forms predictions using a single command. The function requires at least four inputs.

  - *train*: a matrix containing the predictors associated with the training data, i.e., design matrix of training set, labeled `Xtrain` below
  - *test*: a matrix containing the predictors associated with the data for which we wish to make predictions, i.e., `Xtest`
  - *cl*: a vector containing the class labels for the training observations, labeled `Ytrain` below.
  - k: the number of nearest neighbors to be used by the classifier.

- Now we apply `knn()` function to train the kNN classifier on the training set and make predictions on training and test sets.

  Notice that we set a random seed before applying `knn()` to ensure reproducibility of results. The random component in `knn()` is that if several observations are tied as nearest neighbors, then R will randomly break the tie, so a fixed seed instructs R to break the tie in the same way as we run the function multiple times.

  - **Calculate training error rate** Recall that there are at least four arguments in `knn()` that should be specified. In order to get the training error, we have to train the kNN classifier on the **training** set and predict `High` on the same **training** set, then we can construct the 2 by 2 confusion matrix to get the training error rate. Based on this idea, we should have `train=XTrain`, `test=XTrain`, and `cl=YTrain` in `knn()`. We assume `k=2` is the best number of nearest neighbors for now, so we train a 2-NN classifier.

```
set.seed(444)

# knn - train the classifier and make predictions on the TRAINING set!
pred.YTtrain = knn(train=XTrain, test=XTrain, cl=YTrain, k=2)

# Get confusion matrix
conf.train = table(predicted=pred.YTtrain, true=YTrain)
conf.train
```

```
##          true
## predicted High Low
##      High  100  32
##      Low     0  68
```

```
# Trainning error rate
1 - sum(diag(conf.train)/sum(conf.train))
```

```
## [1] 0.16
```

  - **Calculate test error rate** To get the test error, we have to train the kNN classifier on the **training** set and predict `High` on the **test** set, then again we can construct the 2 by 2 confusion matrix to get the test error rate. Based on this idea, we should have `train=XTrain`, `test=XTest`, and `cl=YTrain` in `knn()`. Similarly as above, we set a random seed and try to train the 2-NN classifier.

```
set.seed(555)

# knn - train the classifier on TRAINING set and make predictions on TEST set!
pred.YTest = knn(train=XTrain, test=XTest, cl=YTrain, k=2)

# Get confusion matrix
```

```
conf.test = table(predicted=pred.YTest, true=YTest)
conf.test
```

```
##          true
## predicted High Low
##      High   83  71
##      Low    16  30
```

```
# Test error rate
1 - sum(diag(conf.test)/sum(conf.test))
```

```
## [1] 0.435
```

The first thing to note is that the test error rate is higher than the training error rate, which is expected! But the test error rate obtained by 2-NN classifier is not ideal enough, since 43.5% of the test observations are incorrectly predicted, which is closed to a coin-tossing result. It will be no surprise to us that changing the number of neighbours (k) in the above `knn()` function will lead us to different training/test error rates. Among all possible values of neighbours, which would be the best one and what is the strategy to find this optimal value? The answer, in short, is by Cross-validation.

3. k-Nearest Neighbors regression

k-Nearest Neighbors method can also be used for regression problem. In this section, we still again consider the `Carseats` dataset, which is a simulated data set containing sales of child car seats at 400 different stores on 11 features (3 discrete and 8 numerical). For the regression goal, we are interested in investigating the relationship between the response `Sales` and predictors `Income`, `Advertising`, and `Price`.

To achieve this goal, we will construct a new dataset by selecting corresponding predictors and response, and we call the resulting dataset `sales`:

```
# Create the selected dataset from Carseats, named sales
sales = Carseats %>%
    select(Sales, Income, Advertising, Price)

# Check column names of seats
colnames(sales)
str(sales)
```

(a). Training/Testing split

Following earlier steps, we again carry out the training/test split

```
# Set random seed
set.seed(333)

# Sample 50% observations as training data
train = sample(1:nrow(sales), 200)
sales.train = sales[train,]

# The rest 50% as test data
sales.test = sales[-train,]
```

Following the naming conventions in Section 2, we create `XTrain`, `YTrain`, `XTest` and `YTest`. `YTrain` and `YTest` are response vectors from the training set and the test set. `XTrain` and `XTest` are design matrices.

```
# YTrain is the true labels for High on the training set, XTrain is the standardized design matrix
YTrain = sales.train$Sales
```

```
XTrain = sales.train %>% select(-Sales) %>% scale(center = TRUE, scale = TRUE)

# YTest is the true labels for High on the test set, Xtest is the design matrix
YTest = sales.test$Sales
XTest = sales.test %>% select(-Sales) %>% scale(center = TRUE, scale = TRUE)
```

(b). Train a kNN regressor and calculate MSE

- `knn.reg()` function in the package `FNN` can be used to train a kNN regressor. Similar to the `knn()` function in classification setting, the function `knn.reg()` requires at least four inputs.
  - *train*: a matrix containing the predictors associated with the training data, i.e., design matrix of training set, labeled `Xtrain` below
  - *test*: a matrix containing the predictors associated with the data for which we wish to make predictions, i.e., `Xtest`
  - y: responses for the training observations, labeled `Ytrain` below.
  - k: the number of nearest neighbors to be used by the regressor.
- The output of `knn.reg()` is a list (i.e., a combination of different objects in R). You can access the certain object inside the list by using the `$` operator. In particular, `$pred` returns the predictions on the test dataset.
- Now we apply `knn.reg()` function to train the kNN regressor on the training set and make predictions on training and test sets.
  - **Calculate training MSE** Recall that there are at least four arguments in `knn.reg()` that should be specified. In order to get the training error, we have to train the kNN regressor on the **training** set and predict `Sales` on the same **training** set. Based on this idea, we should have `train=XTrain`, `test=XTrain`, and `y=YTrain` in `knn()`. We assume `k=2` is the best number of nearest neighbors for now, so we train a 2-NN regressor.

```
set.seed(444)

# knn - train the regressor and make predictions on the TRAINING set!
pred.YTtrain = knn.reg(train=XTrain, test=XTrain, y=YTrain, k=2)

# Get confusion matrix
head(pred.YTtrain)
```

```
## $call
## knn.reg(train = XTrain, test = XTrain, y = YTrain, k = 2)
##
## $k
## [1] 2
##
## $n
## [1] 200
##
## $pred
##   [1]  6.900  7.440  6.130  8.065  4.660 11.350  7.510  6.960  4.325  5.320
##  [11]  5.990  2.880  3.250  5.270 10.625  4.770  4.220  7.905 11.465  8.670
##  [21] 10.195  4.510  5.535  5.150  4.900  3.185  9.780  6.525  6.300 10.790
##  [31]  6.515  8.470 11.465  7.255  3.580  5.315  4.940 11.315  7.980  6.020
##  [41]  4.105  5.090  6.490  8.255  2.840 10.260  7.660  4.740  2.180 10.350
##  [51]  2.880  5.550  4.105  6.400  8.000 10.625  1.465  8.950 10.390  6.650
```

```
##  [61]  3.555  7.470  8.730  8.950  5.610  6.610  6.725 10.970  6.010  8.885
##  [71] 11.235 10.350  9.080  6.045  8.920  9.335  4.890  6.865  6.130  4.220
##  [81]  6.515  8.510  2.730 10.540  6.050  6.595  9.975 11.585  7.030  6.180
##  [91]  7.890  8.090  7.480  7.890  7.300  5.175 11.785  9.975  7.775  9.925
## [101]  6.960  9.080  5.550  7.895  8.920  5.520  7.030 11.405  7.740  8.920
## [111] 10.960  6.165 11.595  5.535  6.050  7.770  3.250  6.135 11.085 11.340
## [121]  4.900  5.370  7.480  7.795  7.895  5.385 10.050  6.380  9.725  7.540
## [131]  6.955  7.470  9.335 11.585  9.085  7.535  6.300  9.300  6.050 11.350
## [141] 11.085  6.065  6.975  4.510  5.385  8.755  6.045  8.755  5.370  4.700
## [151]  8.475  6.315 10.540  8.895  7.905  7.180  2.730  9.885 10.425  5.090
## [161]  7.500  9.925  7.500  5.315  8.475  5.440 10.970  4.105  7.905 10.790
## [171]  5.155  6.900  9.725  7.180  5.655  6.650  9.690  9.750  4.940  8.625
## [181]  6.525  8.735  7.510 11.595 10.425  7.035  6.030  9.690  6.725 10.390
## [191]  3.540  7.740  7.300  5.125  7.890  6.915  6.975  8.255 11.225  7.795
##
## $residuals
## NULL
##
## $PRESS
## NULL
```

```r
# Training MSE
mean((pred.YTtrain$pred - YTrain)^2)
```

```
## [1] 3.214146
```

– **Calculate test MSE** To get the test MSE, we have to train the kNN regressor on the **training** set and predict `Sales` on the **test** set. Based on this idea, we should have `train=XTrain`, `test=XTest`, and `y=YTrain` in `knn.reg()`. Similarly as above, we set a random seed and try to train the 2-NN regressor.

```r
set.seed(555)

# knn - train the regressor on TRAINING set and make predictions on TEST set!
pred.YTest = knn.reg(train=XTrain, test=XTest, y=YTrain, k=2)

head(pred.YTest$pred)
```

```
## [1]  9.120 10.425  8.510  6.430  6.430 12.420
```

```r
# Test MSE
mean((pred.YTest$pred - YTest)^2)
```

```
## [1] 7.044295
```

Just as we discovered in the classification settting, the test MSE obtained by 2-NN regressor is not ideal enough. Again, it will be no surprise to us that changing the number of neighbours (k) in the above `knn.reg()` function will lead us to different training/test MSE. Can you try different values of (k) and see what is going on? Among all possible values of neighbours, which would be the best one and what is the strategy to find this optimal value? The answer, in short, is by Cross-validation. We will talk about Cross-Validation in the future.

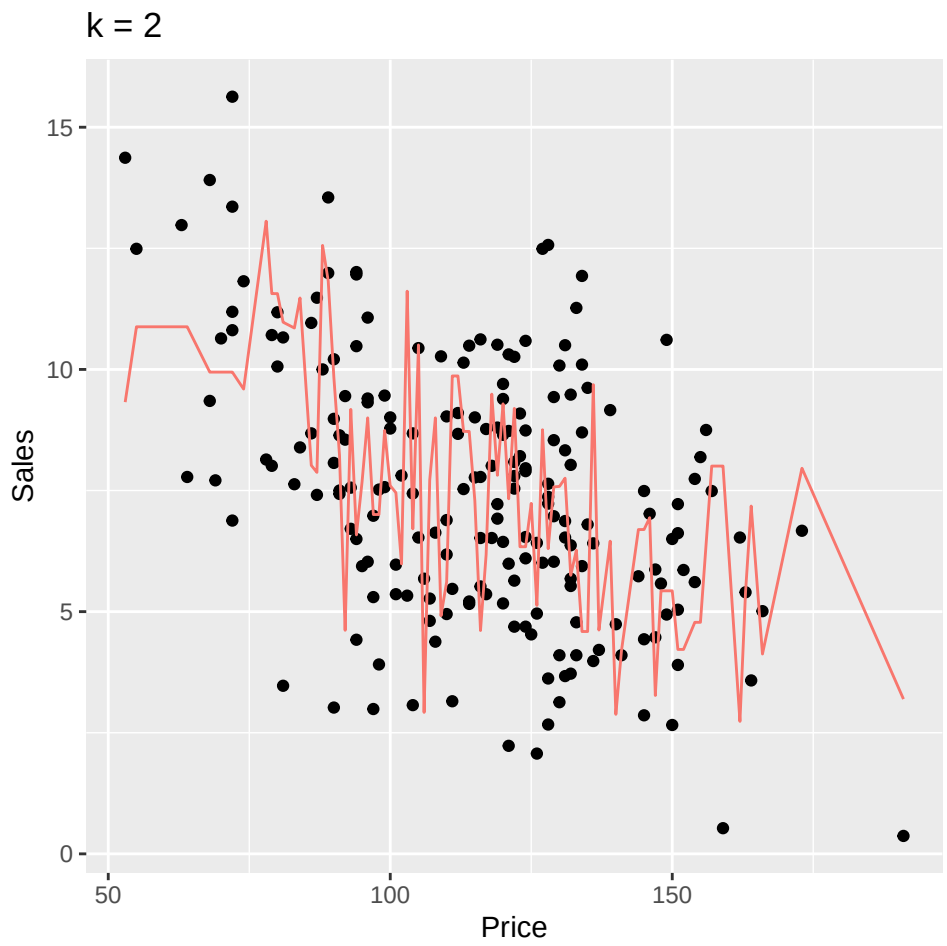
4. Drawing the test error in k-Nearest Neighbors regression

We can draw the actual responses and the prediction in the test dataset, and visually evaluate how the k-NN regressor performs.

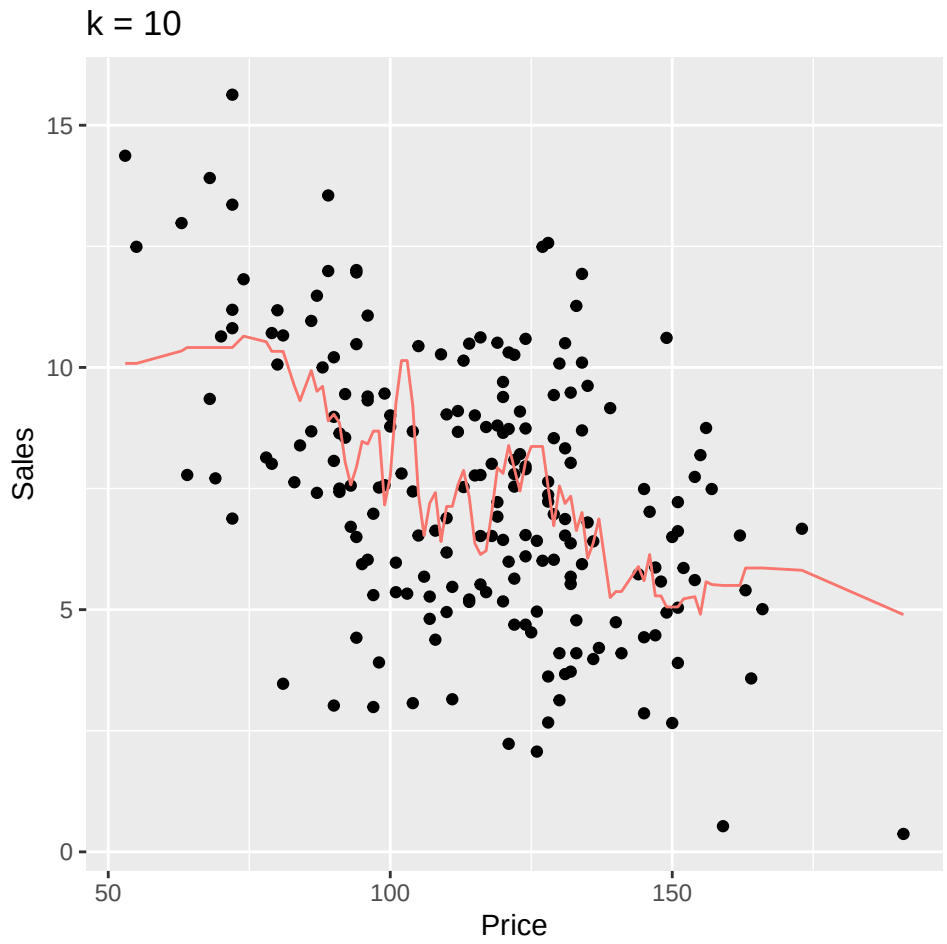
In this section, we will learn the relationship betweent `Sales` (as response) and `Price` (as the only predictor).

The reason that we only consider one predictor is for visualization purpose.

```
sales = Carseats %>% select(Sales, Price)
set.seed(333)

# Sample 50% observations as training data
train = sample(1:nrow(sales), 200)
sales.train = sales[train,]

# The rest 50% as test data
sales.test = sales[-train,]

# Get XTrain and YTrain
YTrain = sales.train$Sales
XTrain = sales.train %>% select(-Sales) %>% scale(center = TRUE, scale = TRUE)

# YTest is the true labels for High on the test set, Xtest is the design matrix
YTest = sales.test$Sales
XTest = sales.test %>% select(-Sales) %>% scale(center = TRUE, scale = TRUE)

# knn - train the regressor on TRAINING set and make predictions on TEST set!
# we consider two different values of k, k = 2 and k = 10, and
# visually look at their difference
pred.YTest.k2 = knn.reg(train = XTrain, test = XTest, y = YTrain, k = 2)
pred.YTest.k10 = knn.reg(train = XTrain, test = XTest, y = YTrain, k = 10)

# you can wrap the code manually like this, in order to prevent
# having a very long command sentence in Rmd output file.
data.plot.k2 <- data.frame(Sales = sales.test$Sales,
                           Price = sales.test$Price,
                           Pred = pred.YTest.k2$pred)

data.plot.k10 <- data.frame(Sales = sales.test$Sales,
                           Price = sales.test$Price,
                           Pred = pred.YTest.k10$pred)

# black dots represent the actual data points in test set
# red line represents the 2-nn prediction
data.plot.k2 %>% ggplot(aes(x=Price, y=Sales)) + geom_point() +
    geom_line(aes(x = Price, y=Pred, color="red")) + theme(legend.position = "none") +
  ggtitle("k = 2")
```




```
# draw the same plot for 10-nn
data.plot.k10 %>% ggplot(aes(x=Price, y=Sales)) + geom_point() +
    geom_line(aes(x = Price, y=Pred, color="red")) + theme(legend.position = "none") +
  ggtitle("k = 10")
```



We observe that `2-nn` regressor yields more wiggly prediction curve than `10-nn`. Recall in the lecture that `2-nn` is more flexible than `10-nn`, meaning that `2-nn` could have smaller bias and higher variance than `10-nn`. This is reflected in these two plots. The prediction curve for `2-nn` fits the data points better (low bias), but is more wiggly (higher variance).

We get a conceptual and visual undertanding how `2-nn` and `10-nn` perform in comparison, but how do we know exactly which is better? We need to compute the test MSE for both methods on the test dataset. We have shown how to do it in the previous section.