STATS415hw3

Yunguo Cai

1/29/2018

Load the 'ISLR' package to get Carseats data set and use the first 90% of the observations for training and the remaining 10% for testing.

```
library('ISLR')
#Divide the data into training and test sets
set.seed(100)
train_id = 1:floor(nrow(Carseats)*.9)
trainCarseats <- Carseats[train_id,]
testCarseats <- Carseats[-train_id,]
```

1.

```
mse <- function(model, y, data) {
  # model is an lm object (a linear regression)
  # y is the response variable from model
  # data is the dataset we want to use to compute fitted values using our model

  # The predict function computes fitted values when given a model and predictor data
  yhat <- predict(model, data)
  mean((y - yhat)^2)
}
```

```
#multiple regression model to predict Sales using all other variables
model1 = lm(Sales ~., data = trainCarseats)
#a reduced model with everything except for Population, Education, Urban, and US
model2 = lm(Sales ~.- Population - Education -Urban - US , data = trainCarseats)
summary(model1)
```

```
##
## Call:
## lm(formula = Sales ~ ., data = trainCarseats)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.8526 -0.7041  0.0389  0.6773  3.3814
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)      5.4852852  0.6408115   8.560 3.68e-16 ***
## CompPrice        0.0931519  0.0043767  21.284  < 2e-16 ***
## Income           0.0166264  0.0019639   8.466 7.18e-16 ***
## Advertising      0.1233377  0.0118123  10.441  < 2e-16 ***
## Population       0.0003420  0.0003888   0.879    0.380
## Price           -0.0955215  0.0027942 -34.186  < 2e-16 ***
## ShelveLocGood    4.8423763  0.1625583  29.789  < 2e-16 ***
## ShelveLocMedium  1.9507951  0.1329724  14.671  < 2e-16 ***
## Age             -0.0468736  0.0033666 -13.923  < 2e-16 ***
## Education       -0.0152832  0.0205585  -0.743    0.458
## UrbanYes         0.1213664  0.1188935   1.021    0.308
```

```
## USYes           -0.1511358  0.1572477  -0.961    0.337
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.023 on 348 degrees of freedom
## Multiple R-squared:  0.8733, Adjusted R-squared:  0.8693
## F-statistic:   218 on 11 and 348 DF,  p-value: < 2.2e-16
```

```
summary(model2)
```

```
##
## Call:
## lm(formula = Sales ~ . - Population - Education - Urban - US,
##     data = trainCarseats)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.7784 -0.6893  0.0373  0.6837  3.3272
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)       5.433534   0.532752  10.199  < 2e-16 ***
## CompPrice         0.092813   0.004346  21.357  < 2e-16 ***
## Income            0.016485   0.001948   8.461 7.19e-16 ***
## Advertising       0.118163   0.008320  14.202  < 2e-16 ***
## Price            -0.095437   0.002790 -34.204  < 2e-16 ***
## ShelveLocGood     4.826474   0.161733  29.842  < 2e-16 ***
## ShelveLocMedium   1.947009   0.132282  14.719  < 2e-16 ***
## Age              -0.046879   0.003357 -13.965  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.022 on 352 degrees of freedom
## Multiple R-squared:  0.872,  Adjusted R-squared:  0.8694
## F-statistic: 342.5 on 7 and 352 DF,  p-value: < 2.2e-16
```

```
#training and test error for model 1 and 2
train_mse_1 = mse(model1,trainCarseats$Sales,trainCarseats)
test_mse_1 = mse(model1,testCarseats$Sales,testCarseats)
train_mse_1
```

```
## [1] 1.010792
```

```
test_mse_1
```

```
## [1] 0.9903956
```

```
train_mse_2 = mse(model2,trainCarseats$Sales,trainCarseats)
test_mse_2 = mse(model2,testCarseats$Sales,testCarseats)
train_mse_2
```

```
## [1] 1.021013
```

```
test_mse_2
```

```
## [1] 1.0041
```

The MSE using the training data for model1 is 1.010792, while the MSE using the test data for model1 is

0.9903956. The two values are very close to each other, which means that model1 is not overfitted. The MSE using the training data for model2 is 1.021013, while the MSE using the test data for model2 is 1.0041. The two values are very close to each other, which means that model2 is not overfitted.

In comparison with model1, both training error and test error for model2 increase a little bit though they???re still close. It shows that though model1 has higher flexibility with 4 more variables, model2 performs almost as well as model1 in prediction. Model2 is a better choice with high accuracy but fewer variables. For both model1 and model2, traing error is slightly bigger than test error. It might be caused by the observations in the training data set is 90%, which is 9 times more than the observations in the test data set and thus results in bigger error.

2.

I would expect a better training error with K = 1. When K = 1, the model is more flexible than the model when K = 10. For the training data, the predicted value are usually just itself. Thus the training error will be very small. By contrast, I would expect a better test error with K = 10. When K = 1, the model is very likely to be overfitted and thus its performance to predict test data will be worse than the model with K = 10.

3.

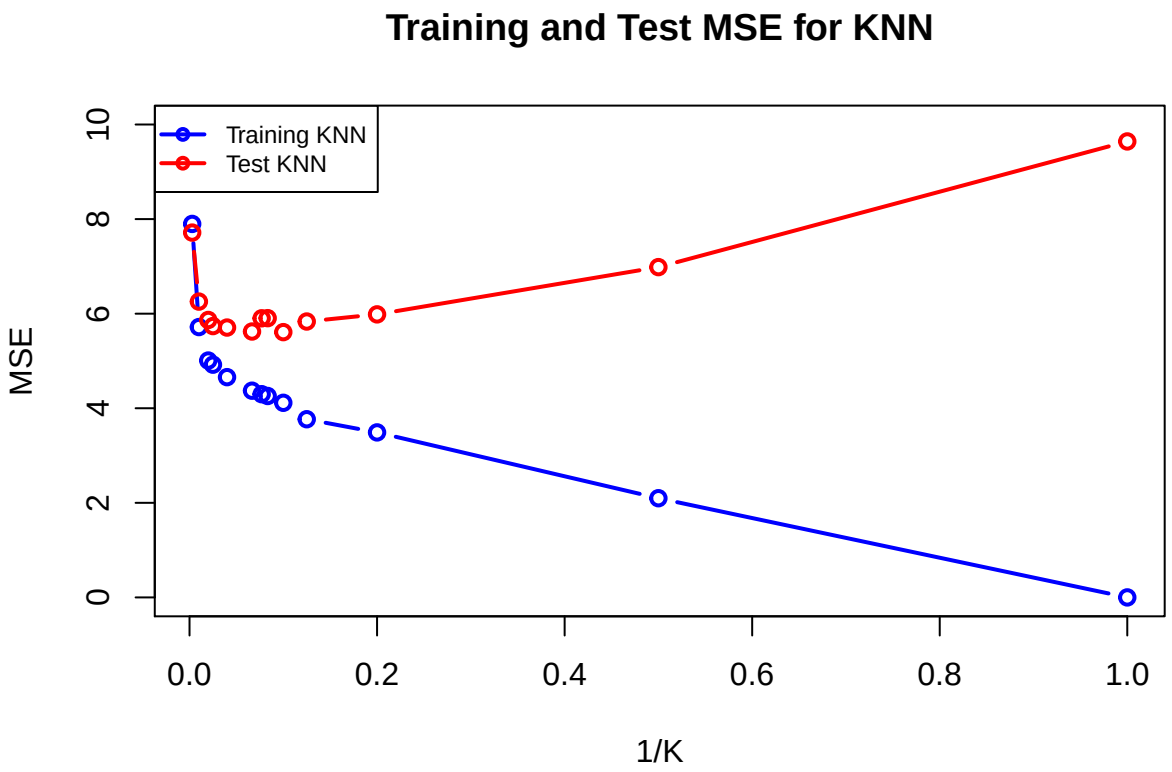
I would standardize the variables in the dataset first. The ranges and units of different variables are quite different. Weighted distances have been used to emphasize some variables over others. For example, CompPrice and Price contain much larger numbers than Advertising. However, CompPrice and Price are not necessarily more important than Advertising. So without standardization, the KNN model will be influenced a lot. When not knowing the importance of each variable, standardize all the variables at first so that all the variables will have the same weight in the KNN model is the best practice.

```
library("FNN")
knn_train_carseats = trainCarseats[c(1,2,3,4,6,8)]
knn_test_carseats = testCarseats[c(1,2,3,4,6,8)]
n=50
k_range  = c(1,2,5,8,10,12,13,15,25,40,50,100,353)
trainMSE = c() #creating null vector

for(i in 1:length(k_range)){
  knnTrain <- knn.reg(train = scale(knn_train_carseats[,-1]),
                      test = scale(knn_train_carseats[,-1]),
                      y = knn_train_carseats$Sales,
                      k = k_range[i])
  trainMSE[i] <- mean((knn_train_carseats$Sales-knnTrain$pred)^2)
}
```

```
testMSE = c() #creating null vector
for(i in 1:length(k_range)){
  knnTest <- knn.reg(train = scale(knn_train_carseats[,-1]),
                     test = scale(knn_test_carseats[,-1]),
                      y = knn_train_carseats$Sales,
                     k = k_range[i])
  testMSE[i] <- mean((knn_test_carseats$Sales - knnTest$pred)^2)
}
```

```
k_inverse=1/k_range
plot(trainMSE ~ k_inverse, type = "b", lwd = 2, col = "blue",
main = "Training and Test MSE for KNN", xlab = "1/K", ylab = "MSE",ylim = c(0,10))
# Add the test MSE
lines(testMSE ~ k_inverse, type = "b", lwd = 2, col = "red")
legend("topleft", legend = c("Training KNN", "Test KNN"), col = c("blue", "red"),
cex = .75, lwd = c(2, 2), pch = c(1, 1), lty = c(1, 1))
```




```
which(trainMSE==min(trainMSE))
```

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

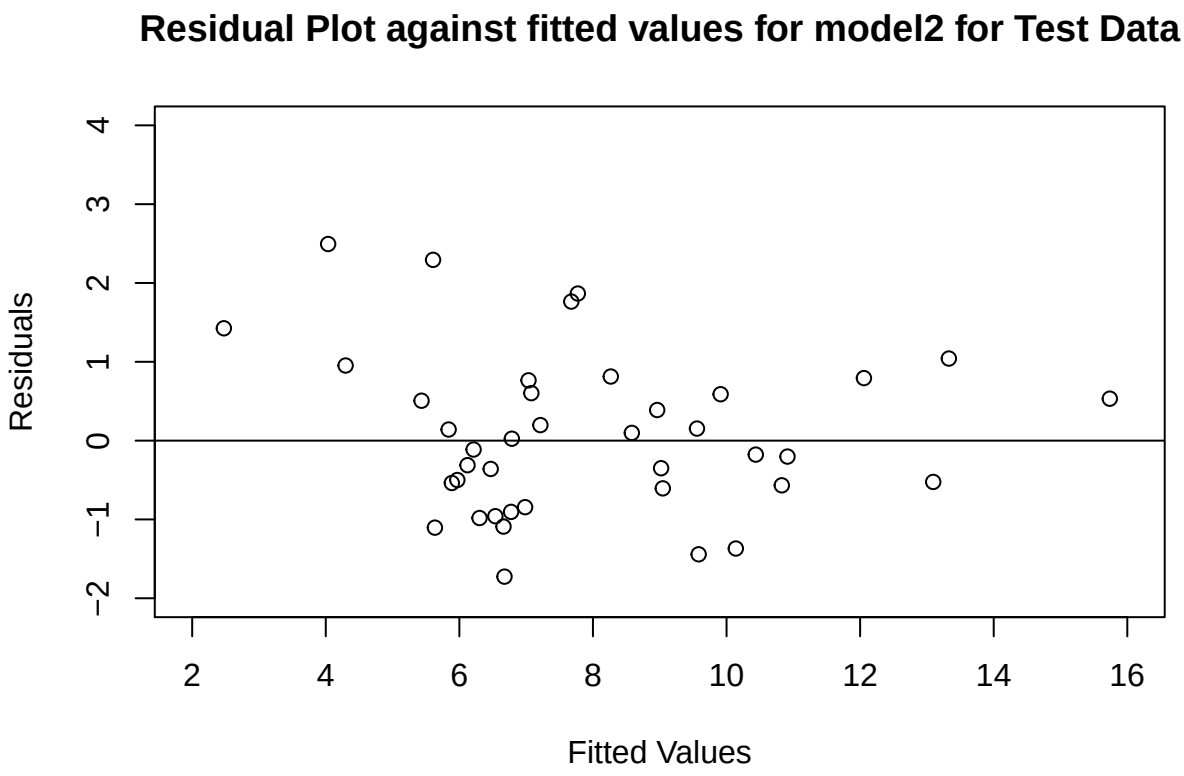
```
which(testMSE==min(testMSE))
```

```
## [1] 5
```

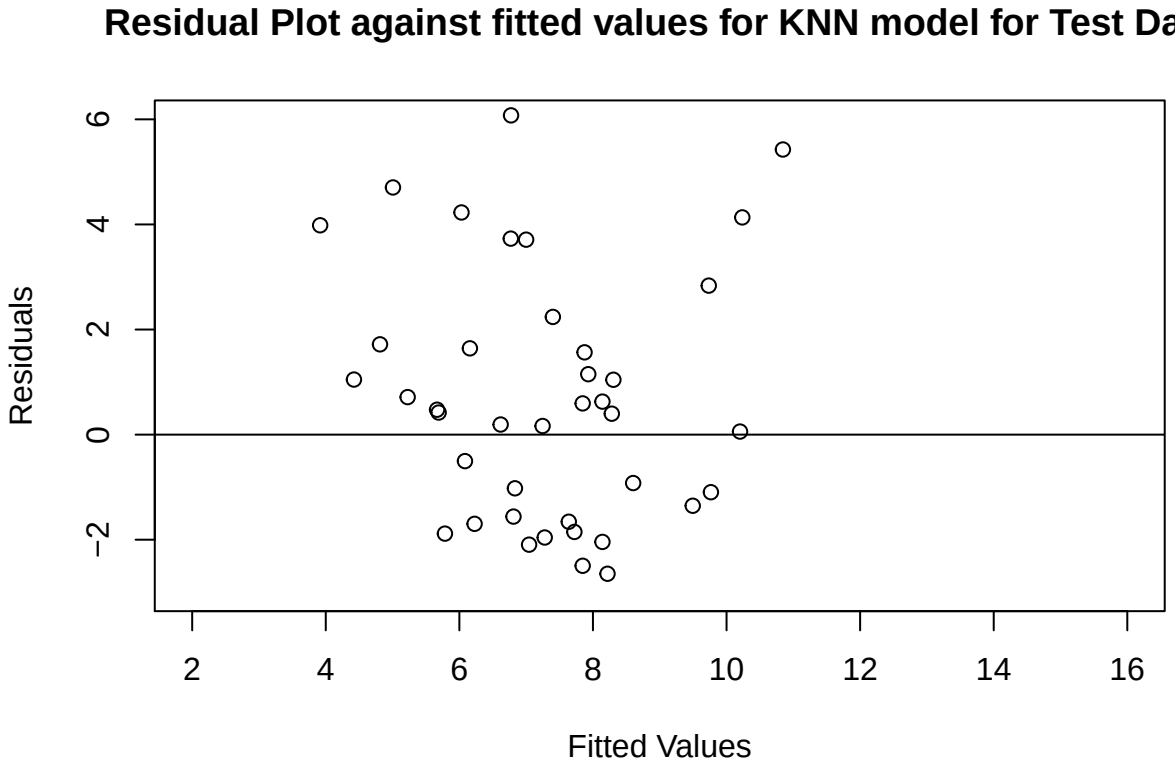
The training error is the smallest when K=1. The test error is the smallest when K=5.
The training error keeps decreasing when K decreases. By contrast, as K decreases, the test error first decreases, reaches the smallest at some point (in this case, K=5), and then turns to increase. This case of MSE plot of test error agrees to the theory that a model with too much flexibility may overfit so that it gets smaller training error but larger test error as K decreases.

5

```
pred_y_2 = predict(model2, testCarseats)
model2_residuals = testCarseats$Sales - pred_y_2
plot(model2_residuals~pred_y_2, xlab = 'Fitted Values', xlim = c(2,16), ylim = c(-2,4),
ylab = 'Residuals', main = 'Residual Plot against fitted values for model2 for Test Data')
abline(0,0)
```




```
knnTest <- knn.reg(train = scale(knn_train_carseats[,-1]),
                   test = scale(knn_test_carseats[,-1]),
                   y = knn_train_carseats$Sales,
                   k = 5)
test_residuals <- testCarseats$Sales - knnTest$pred
plot(test_residuals~knnTest$pred, xlab = 'Fitted Values', xlim = c(2,16), ylim = c(-3,6),
ylab = 'Residuals', main = 'Residual Plot against fitted values for KNN model for Test Data')
abline(0,0)
```



The two plots for linear regression model and KNN model separately are quite different. The similarity of the two plots is that most points are between the fitted values 6-8. The diffrence is that the linear model has a wider range in fitted values while a smaller range in residuals. For model2, the linear regression model, the residuals ranges roughly from -2 to 3, and the fitted values ragnes roughly from 2 to 16. For the KNN model with K=5, the residuals ranges roughly from -3 to 6, and the fitted values ragnes roughly from 4 to 12. It shows that the linear regression model has smaller test error than the KNN model with the optimal K=5 which has the smallest test error among all the KNN models with different K values.

```
test_mse_2
```

```
## [1] 1.0041
```

```
knnTest_mse = mean((knn_test_carseats$Sales - knnTest$pred)^2)
knnTest_mse
```

```
## [1] 5.985387
```

The test MSE of the linear regression model2 is only 1.0041, while the test MSE of the KNN regression model with K=5 is 5.985387, which proves that KNN has bigger MSE. The fitted value of KNN model seems to be more concentrated than the linear regression model2. The reason for this might be that KNN model considers K closest data points and take the average of their responses, so extreme values are less likely to occur and the fitted values are thus more concentrated.
From the residual plots, it shows that model2 performs better than the knn model with k=5 in prediction with smaller residuals. In general, the KNN model should perform better. As long as n/K>p, KNN is more flexible than a linear model with p predictors. However, in this case KNN model performs worse and the MSE is much bigger than the linear model2. The reason for this might be the choice of training data. We use the first 90% of the observations for training data and the remaining 10% for testing. We haven't randomly chosen the training data, which might cause some bias. It may also caused by the overfitting of the KNN model. In all, linear regression model is eaiser to interpret as a less flexible method.