Chapter 4 Problem 13

Andira Putri

Using the `Boston` data set, fit classification models in order to predict whether a given suburb has a crime rate (`crim`) above or below the median. Explore logistic regression, LDA, and KNN models using various subsets of predictors. Describe your findings.

```
#Import Boston data
library(MASS)
#create dummy variables for crim based on median
#1 if above median
#0 otherwise
med=median(Boston$crim)
crim01=ifelse(Boston$crim>med,1,0)
#Combine Boston data set with new dummy variables
df=data.frame(Boston,crim01)
```

- Quick review of variables in `Boston` data set:

crim-per capita crime rate by town.

zn-proportion of residential land zoned for lots over 25,000 sq.ft.

indus-proportion of non-retail business acres per town.

chas-Charles River dummy variable (= 1 if tract bounds river; 0 otherwise).

nox-nitrogen oxides concentration (parts per 10 million).

rm-average number of rooms per dwelling.

age-proportion of owner-occupied units built prior to 1940.

dis-weighted mean of distances to five Boston employment centres.

rad-index of accessibility to radial highways.

tax-full-value property-tax rate per $10,000.

ptratio-pupil-teacher ratio by town.

black- $1000(B_k - 0.63)^2$ where B_k is the proportion of blacks by town.

lstat-lower status of the population (percent).

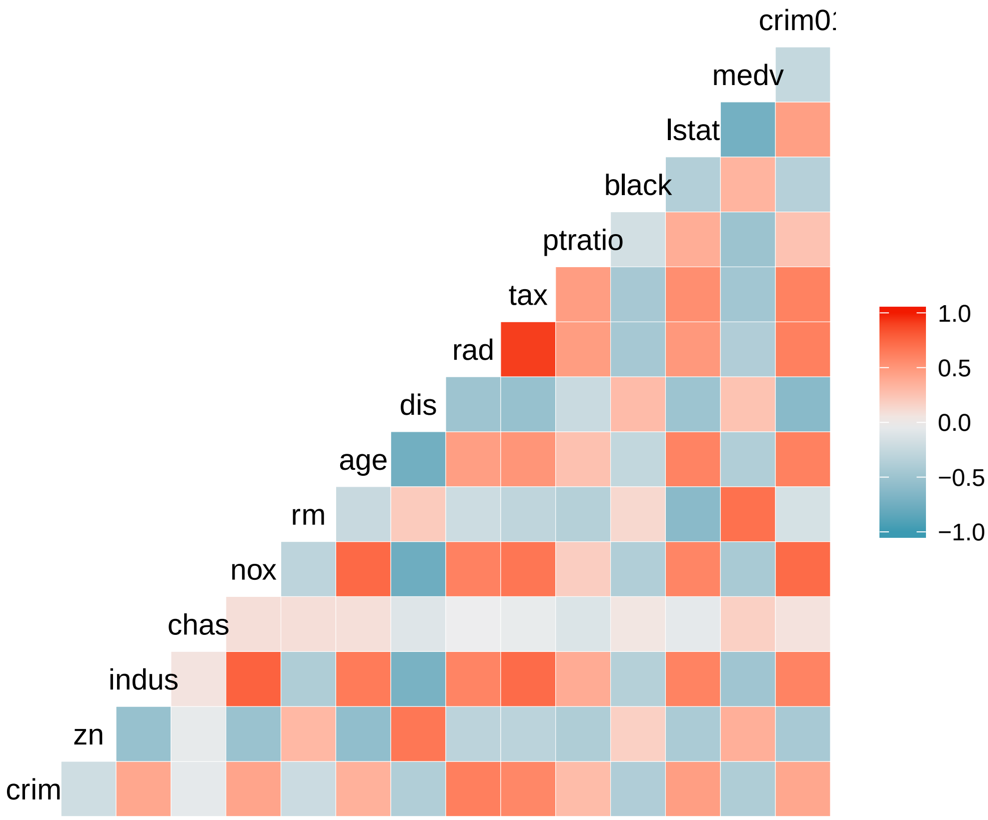
medv-median value of owner-occupied homes in $1000s.

```
#Getting an idea of which predictors I would like to use
require(GGally)
```

```
## Loading required package: GGally
```

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

```
#in heatmap, red -> neg corr, blue -> pos corr
ggcorr(data=df)
```



We look at the right-most side of the heatmap triangle since the `crim01` variable is examined with all other ones there. The heatmap suggests that `rad`,`tax`,`age`,`nox`, and `dis` would be interesting predictors to use, based on the color saturation of the squares.

Before starting any model-fitting, I split the data into a training and test set.

```
set.seed(1)
#split data
#I have 506 total observations, I want to train using 400 of them
train=sample(506,400)
train.set=df[train,]
test.set=df[-train,] #106 observations for testing
```

Logistic Regression

In our supervised learning studies, we shift from linear regression methods to logistic regression (among others) when our response is qualitative and binary. Note that $p(x) \in [0, 1]$.

The logistic model is given by:

$p(x) = \frac{exp(\beta_0+\beta_1 x)}{1+exp(\beta_0+\beta_1 x)}$

When we do some rearranging to the previous equation, we get something called "the odds":

$\frac{p(x)}{1-p(x)} = exp(\beta_0 + \beta_1 x)$

The values of odds close to 0 or ∞ indicate very low or high probabilities, respectively. Finally, we rearrange the equation one last time to get the log-odds, or logit formula.

$log(\frac{p(x)}{1-p(x)}) = \beta_0 + \beta_1 x$

This gives us a linear function of the predictors. Increasing x by one unit changes the log-odds by β_1. Logistic regression uses the maximum likelihood estimation to compute the approximate values of β_0 and β_1. Essentially, we seek to find coefficient values that corresponds as closely as possible to an observation. In other words, we find β_0 and β_1 that gives us $p(x) = 1$ when an observation belongs to a class and $p(x) = 0$ when an observation does not. Usually, the resulting $p(x)$ is not always 0 or 1, so we define a threshold. For example, say our threshold is 0.5 (which it usually is unless there is a special circumstance). If $p(x) < 0.5$, we round down to $p(x) = 0$ and say that the observation does not belong to the class. This methodology is reflected in the code below:

```
set.seed(1)
#fit logistic regression model with trainind data
log.fit=glm(crim01~rad+tax+age+nox+dis,train.set,family="binomial")
#make predictions with log reg model
#generates a vector of probabilities in the form P(Y=1|X)
log.prob=predict(log.fit,test.set,type="response")
#create vector of 1089 "0"" elements
log.pred=rep("0",106)
#change "0" to "1" if probability exceeds 0.5
log.pred[log.prob>0.5]="1"
#generates confusion matrix
table(log.pred,test.set$crim01)
```

```
##
## log.pred  0  1
##        0 52  7
##        1  7 40
```

```
#compute test MSE
(7+7)/106*100
```

```
## [1] 13.20755
```

Linear Discriminant Analysis (LDA)

Logistic regression featuring more than 2 response classes is possible, but there are better alternatives such as linear discriminant analysis. We also choose to perform LDA when the classes are well-separated and a small amount of training observations, as logistic regression can become unstable in these cases.

Let's first go over the assumptions of LDA:

- Data has normal/Gaussian distribution
- Equal variance among all classes (observations of each variable differ from the mean by the same amount)

LDA estimates the mean and variance for each class k, given by μ_k and σ_k respectively. We also have π_k, which denotes the prior probability or the probabilty of each class k in the training set. Then, LDA assigns an observation to the class which maximizes the equation:

$\delta_k(x) = x\frac{\mu_k}{\sigma^2} - \frac{\mu_k^2}{2\sigma^2} + log(\pi_k)$

```
set.seed(1)
#fit LDA model using training set
lda.fit=lda(crim01~rad+tax+age+nox+dis,train.set)
#make predictions using test set
lda.pred=predict(lda.fit,test.set)
#assign to classes
lda.class=lda.pred$class
```

```
#generate confusion matrix
table(lda.class,test.set$crim01)
```

```
##
## lda.class  0  1
##         0 57 15
##         1  2 32
```

```
#compute test MSE
(2+15)/106*100
```

```
## [1] 16.03774
```

Our test error rate is higher than that of logistic regression. Maybe this is because our response `crim01`$\in$\{0,1\} was binary to begin with, and we didn't have to use LDA.

K-Nearest Neighbors (KNN)

LDA is an extension of Bayes classifier, which is ideal but non-realistic for most data because of unknown conditional distributions. Instead, we estimate the distribution of Y|X, then classify the observation to the class with the highest estimated probability. This is the K-nearest neighbors (KNN) classifier. Say we have a test observation x_o and positive integer `k`.

1.) The classifier first identifies the k points in the training data that are closest to x_o, given by N_o.

2.) KNN estimates the conditional probability for class `j` as a fraction of points in N_o whose response values equal j. Probability is estimated by:

$P(Y=j|X=x_o)=\frac{1}{k}\sum_{i\in N_o}I(y_i=j)$

where $I(y_1=j)=0$ when an observation is not in the class and $I(y_1=j)=1$ when an observation is.

3.) KNN classifies the test observations x_o to the class with the largest probability.

To see the classifier in action, please look at problem #7 in the document: https://github.com/diramputri/Statistical-Learning/blob/master/Chapter%202%20Conceptual%20Exercises.pdf. Now, we implement KNN with k=1 and k=4.

```
set.seed(1)
library(class)
#initialize matrices
train.x=as.matrix(train.set$crim01)
test.x=as.matrix(test.set$crim01)
#fit KNN model with k=1
knn.pred=knn(train.x,test.x,train.set$crim01,k=1)
#make predictions on test data
table(knn.pred,test.set$crim01)
```

```
##
## knn.pred  0  1
##        0 59  0
##        1  0 47
```

```
#fit KNN model with k=4
knn.pred1=knn(train.x,test.x,train.set$crim01,k=4)
#make predictions on test data
table(knn.pred1,test.set$crim01)
```

```
##
## knn.pred1  0  1
```

```
##          0 59  0
##          1  0 47
```

For both k=1 and k=4, our error rate is 0%. That's suspicious… I tried again with k=100 below.

```
set.seed(1)
#fit KNN model with k=4
knn.pred10=knn(train.x,test.x,train.set$crim01,k=100)
#make predictions on test data
table(knn.pred10,test.set$crim01)
```

```
##
## knn.pred10  0  1
##          0 59  0
##          1  0 47
```

Again, our error rate with k=100 is 0%.

Conclusion

For logistic regression, we got an error rate of 13.21%. LDA gave us a higher error rate of 16.04%. However, for KNN using any value of `k`, we got 0% error! From these results, we might be able to conclude that our data does not follow a normal/Gaussian distribution because it breaks LDA's assumption about this, thus resulting in poorer performance. In addition, since I picked predictors with the highest correlations, the classes might have been well-separated, which causes logistic regression to be unstable. These are the reasons to believe why KNN performed the best, though the error rates for the other methods were not that bad.