Stat 427/627 Statistical Machine Learning

In-class Lab 3 and 4: Classification

Contents

1 KNN and introduction to classfication rate, confusion matrix, etc.

- KNN is a nonparametric, discriminative, supervised learning algorithm for classification.

1.1 Prepare the data set: Fuel Economy (`Auto` in `ISLR2` Package)

Consider the `Auto` data set in the `ISLR2` package. This data frame has 392 observations on the following 9 variables.

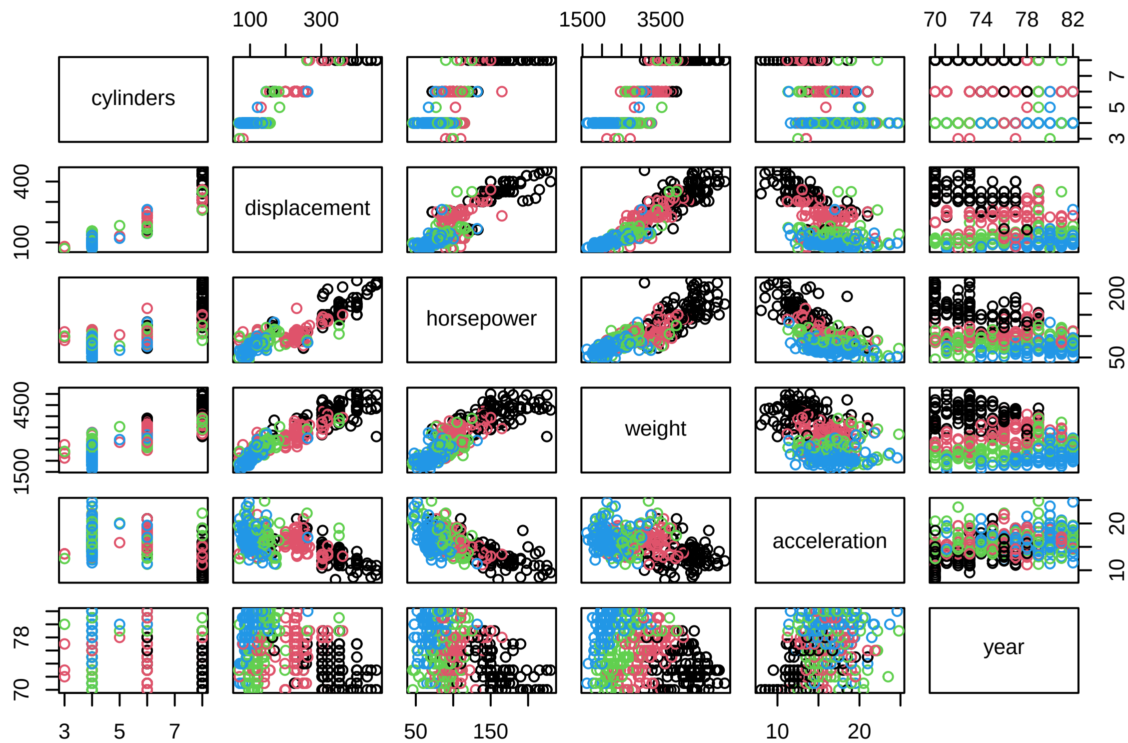
- `mpg`: miles per gallon
- `cylinders`: Number of cylinders between 4 and 8
- `displacement`: Engine displacement (cu. inches)
- `horsepower`: Engine horsepower
- `weight`: Vehicle weight (lbs.)
- `acceleration`: Time to accelerate from 0 to 60 mph (sec.)
- `year`: Model year (modulo 100)
- `origin`: Origin of car (1. American, 2. European, 3. Japanese)
- `name`: Vehicle name

We will create a new categorical variable `econ` (fuel economy) based on `mpg` by the quarantines.

```r
library(ISLR2)

auto.data <- ISLR2::Auto
summary(auto.data$mpg)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    9.00   17.00   22.75   23.45   29.00   46.60
```

```r
auto.data$econ <- cut(auto.data$mpg, breaks=quantile(auto.data$mpg), include.lowest=TRUE,
                      labels=c("Poor", "OK", "Good", "Excellent"))
pairs(auto.data[, 2:7], col=auto.data$econ)
```



We will also split the data set at 60%-40% for training and validation (or testing).

```r
set.seed(12345)  # Only for example purpose.
training_pct <- 0.6
Z <- sample(nrow(auto.data), floor(training_pct*nrow(auto.data)))
auto.training <- auto.data[Z, ]
auto.testing <- auto.data[-Z, ]
c(nrow(auto.data), nrow(auto.training), nrow(auto.testing))
```

```
## [1] 392 235 157
```

1.2 R function `knn()`

Function `knn()` is in package {`Class`}. Install the package once (for the first time), and load the package.

```r
# install.packages("class")
library(class)
```

This function requires at least 4 input arguments:

- train: Xs (i.e., predictors) in the training set.
- test: Xs (i.e., predictors) in the testing set.
- cl: Observed classification (i.e., response) in the training set.
- k: number of neighbors.

The output of the function is the predicted classification of the testing set.

1.3 Fit KNN model to the fuel economy data.

Recall that column 2 - 7 are the predictors.

```
colnames(auto.training)[2:7]
```

```
## [1] "cylinders"    "displacement" "horsepower"   "weight"       "acceleration"
## [6] "year"
```

```
X.train <- auto.training[, 2:7]
Y.train <- auto.training$econ

X.test <- auto.testing[, 2:7]
Y.test <- auto.testing$econ

econ.knn <- knn(train=X.train, test=X.test, cl=Y.train, k = 3)

table(Y.test, econ.knn)  # Confusion matrix on the testing data set.
```

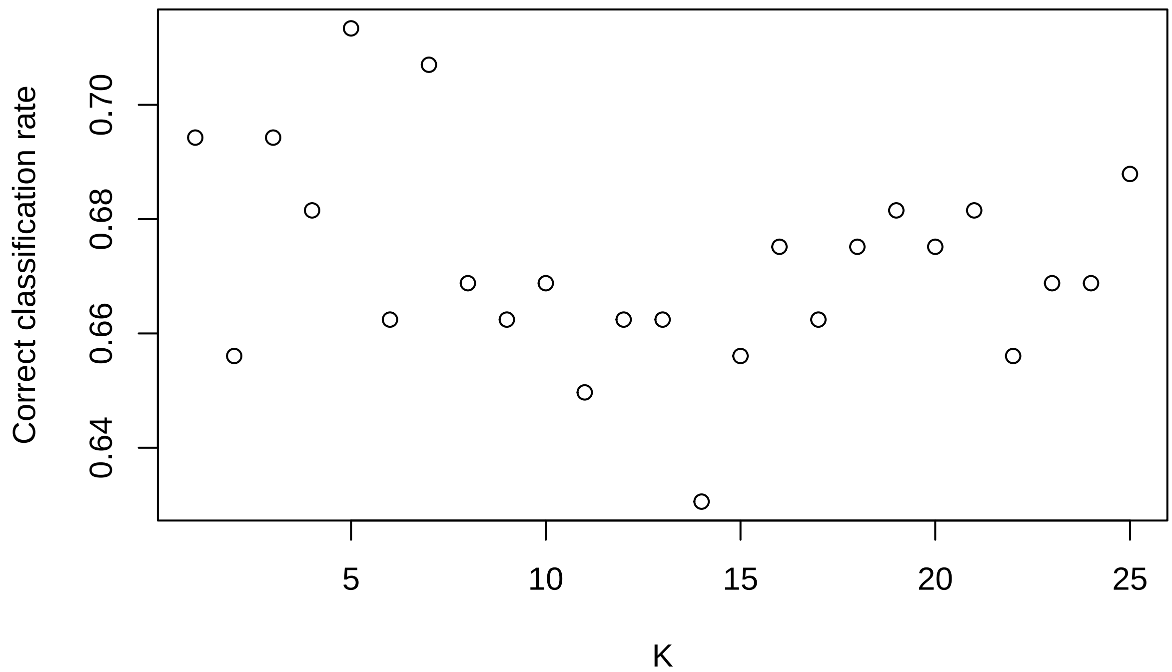
```
##            econ.knn
## Y.test      Poor OK Good Excellent
##   Poor        31  3    0         0
##   OK           6 23    5         0
##   Good         1  8   30         5
##   Excellent    0  2   19        24
```

```
mean(Y.test == econ.knn) # correct classification rate on the testing data.
```

```
## [1] 0.6878981
```

1.4 Tuning K to maximize correct classification rate.

```
Kmax <- 25  # Set the largest K I would consider for this study.
class.rate <- rep(0, Kmax)
for (i in 1:Kmax) {
  knn.out <- knn(train=X.train, test=X.test, cl=Y.train, k = i)
  class.rate[i] <- mean(Y.test == knn.out)
}

plot(c(1:Kmax), class.rate, xlab="K", ylab="Correct classification rate")
```




```r
k.opt <- which.max(class.rate)
c(k.opt, class.rate[which.max(class.rate)])  # Optimal K.
```

```
## [1] 5.0000000 0.7133758
```

```r
econ.knnOpt <- knn(train=X.train, test=X.test, cl=Y.train, k = k.opt)
table(Y.test, econ.knnOpt)  # Confusion matrix on the testing data set.
```

```
##            econ.knnOpt
## Y.test      Poor OK Good Excellent
##   Poor        30  4    0         0
##   OK           5 25    4         0
##   Good         1 11   31         1
##   Excellent    0  4   17        24
```

```r
mean(Y.test == econ.knnOpt) # correct classification rate on the testing data.
```

```
## [1] 0.7006369
```

Remark

- Note that, in this example, the classification rate is above 0.6 for $K = 1, 2, ..., 25$, and the change is relatively small. If you change the random seed, or change the training percentage, you may get a very different K.
- Instead of splitting the data set once, a better approach is conduct *cross-validation*. We'll discuss it next week.

2 Logistic regression

- Logistic regression is a parametric, discriminative, supervised learning algorithm for classification and inference.

2.1 The Depression data (read data from a file)

A study a 3,189 high school students has been concluded in order to find socioeconomic and family factors that my be associated with stress and depression. Data Set **depression.csv** contains some variables obtained from this study.

- `ID`: Participant's identification number
- `Gender`: Female or Male
- `Guardian_status`: 0 = does not live with both natural parents. 1 = with both parents.
- `Cohesion_score`: 16-80, large value indicates strong connection to the community.
- `Depression_score`: 0 - 60
- `Diagnosis`: Clinic diagnosis of major depression. 0 = negative (no), 1 = positive (yes)

```r
depr <- read.csv("../Data/depression_data.csv", header=T)
summary(depr)
```

```
##        ID          Gender          Guardian_status  Cohesion_score
##  Min.   :   1   Length:3189        Min.   :0.0000   Min.   :16.00
##  1st Qu.: 800   Class :character   1st Qu.:0.0000   1st Qu.:48.00
##  Median :1597   Mode  :character   Median :1.0000   Median :58.00
##  Mean   :1597                      Mean   :0.5177   Mean   :56.24
##  3rd Qu.:2394                      3rd Qu.:1.0000   3rd Qu.:66.00
##  Max.   :3191                      Max.   :1.0000   Max.   :80.00
##
##  Depression_score   Diagnosis
##  Min.   : 0.00    Min.   :0.0000
##  1st Qu.: 8.00    1st Qu.:0.0000
##  Median :14.00    Median :0.0000
##  Mean   :15.56    Mean   :0.1572
##  3rd Qu.:21.00    3rd Qu.:0.0000
##  Max.   :54.00    Max.   :1.0000
##                   NA's   :2731
```

```r
# Remove missing values in Diagnosis
depr <- na.omit(depr)

# Since we'll use logistic regression for this data set. Convert the response as 0-1 or a factor.
depr$Diagnosis <- 1*(depr$Diagnosis == 1)
head(depr)
```

```
##    ID Gender Guardian_status Cohesion_score Depression_score Diagnosis
## 10 10 Female               0           69.0               30         0
## 13 13   Male               0           72.0                3         0
## 28 28   Male               1           78.0                9         0
## 30 30   Male               0           56.5               12         0
## 38 40 Female               0           28.0               46         0
## 43 45   Male               0           62.0               15         1
```

```r
# Split data (optional in real life)
set.seed(23456)  # Only for example purpose.
training_pct <- 0.6
Z <- sample(nrow(depr), floor(training_pct*nrow(depr)))
```

```r
depr.training <- depr[Z, ]
depr.testing <- depr[-Z, ]
```

For now, we'll only consider logistic regression for binary (0-1, yes-no, success-failure, etc) response.

```r
head(depr)
```

```
##    ID Gender Guardian_status Cohesion_score Depression_score Diagnosis
## 10 10 Female               0           69.0               30         0
## 13 13   Male               0           72.0                3         0
## 28 28   Male               1           78.0                9         0
## 30 30   Male               0           56.5               12         0
## 38 40 Female               0           28.0               46         0
## 43 45   Male               0           62.0               15         1
```

2.2 glm() function and its output

```r
depr.glm <- glm(Diagnosis ~ Gender + Guardian_status + Cohesion_score, family = binomial, data=depr.trai
summary(depr.glm)
```

```
##
## Call:
## glm(formula = Diagnosis ~ Gender + Guardian_status + Cohesion_score,
##     family = binomial, data = depr.training)
##
## Coefficients:
##                 Estimate Std. Error z value Pr(>|z|)
## (Intercept)      0.78591    0.67763   1.160  0.24613
## GenderMale      -0.79113    0.36369  -2.175  0.02961 *
## Guardian_status -0.96344    0.37050  -2.600  0.00931 **
## Cohesion_score  -0.03484    0.01358  -2.565  0.01032 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 247.98  on 273  degrees of freedom
## Residual deviance: 225.87  on 270  degrees of freedom
## AIC: 233.87
##
## Number of Fisher Scoring iterations: 5
```

```r
depr.glm$fitted.value[1:5]
```

```
##       2852        279        797       3030       1115
## 0.13165959 0.22538084 0.27076827 0.08531709 0.33688550
```

2.3 Predict the probability and the log(odds)

Predict the probability using `predict()` function with argument `type="response"`.

```r
p1 <- predict(depr.glm, newdata=data.frame(Gender="Female", Guardian_status=0, Cohesion_score=50),
        type = "response")
p1
```

```
##         1
```

```
## 0.2777014
```

```r
odds1 <- p1/(1-p1)  # probability -> odds.

p2 <- predict(depr.glm, newdata=data.frame(Gender="Female", Guardian_status=1, Cohesion_score=50),
        type = "response")
odds2 <- p2/(1-p2)

cbind(p1, odds1, p2, odds2)
```

```
##          p1    odds1        p2     odds2
## 1 0.2777014 0.384469 0.1279357 0.1467045
```

Predicting log-odds (i.e., the linear function) is not used often. The example is to show how to compute the probability from log-odds.

```r
logOdds1 <- predict(depr.glm, newdata=data.frame(Gender="Female", Guardian_status=0, Cohesion_score=50)
        type = "link")
exp(logOdds1)/(1+exp(logOdds1))  # log-odds -> probability
```

```
##         1
## 0.2777014
```

2.4 Predict the class. Confusion matrix, classfication rate, TPR, TNR, and error rate.

```r
cutoff <- 0.4  # Try cutoff <-0.5 and see what happens.
predicted.class <- 1*(depr.glm$fitted.values > cutoff)

# Confusion matrix
table(depr.training$Diagnosis, predicted.class)
```

```
##    predicted.class
##       0   1
##   0 219   9
##   1  44   2
```

```r
# Correct classification rate
mean(depr.training$Diagnosis == predicted.class)
```

```
## [1] 0.8065693
```

```r
# Error rate
1 - mean(depr.training$Diagnosis == predicted.class)
```

```
## [1] 0.1934307
```

```r
# TPR, FNR, TNR and FPR
prop.table(table(depr.training$Diagnosis, predicted.class), 1)
```

```
##    predicted.class
##              0          1
##   0 0.96052632 0.03947368
##   1 0.95652174 0.04347826
```

Note that:

- Although the training error rate is 19% (i.e., classification rate 81%), the False Negative rate, $P(\widehat{Y} = 0|Y = 1)$, is very high (95.6%).

- If we increase the cutoff value, we'll have even higher FNR. (Why?)
- Checking the prediction error within the training set can be misleading. We will use the testing data set soon.

2.5 ROC curve and cutoff tuning

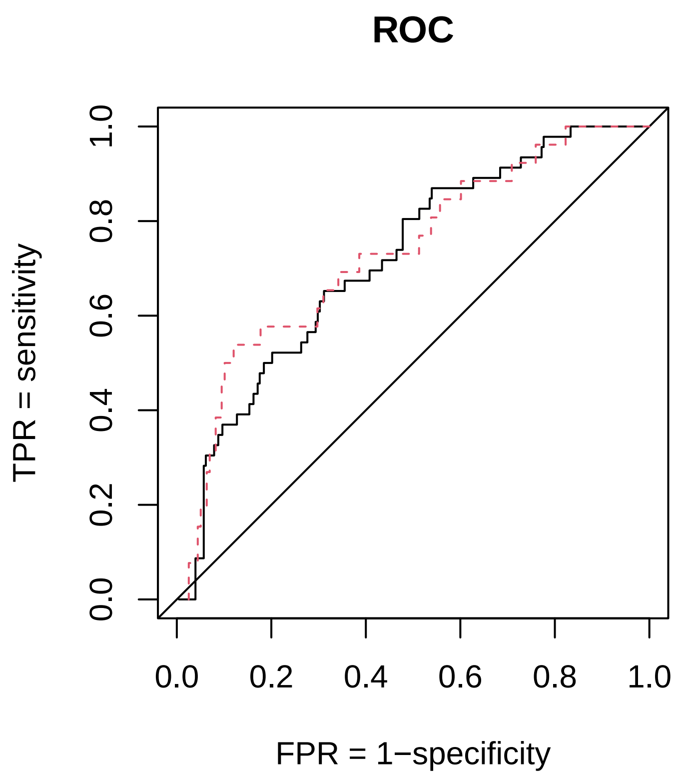
Here is a function that can draw ROC.

- The function optimal cut-off that maximizes $sensitivity + specificity$. I.e, it maximized $TPR + TNR$, or $TPR + (1 - FPR)$,
- **Be sure to convert the response variable to 0-1 and call it `y` in the data frame.**

```
roc.analysis <-function (object, newdata = NULL, newplot=TRUE)
{
  if (is.null(newdata)) {
    pi.tp <- object$fitted[object$y == 1]
    pi.tn <- object$fitted[object$y == 0]
  }
  else {
    pi.tp <- predict(object, newdata, type = "response")[newdata$y == 1]
    pi.tn <- predict(object, newdata, type = "response")[newdata$y == 0]
  }

  pi.all <- sort(c(pi.tp, pi.tn))
  sens <- rep(1, length(pi.all)+1)
  specc <- rep(1, length(pi.all)+1)
  for (i in 1:length(pi.all)) {
    sens[i+1] <- mean(pi.tp >= pi.all[i], na.rm = T)
    specc[i+1] <- mean(pi.tn >= pi.all[i], na.rm = T)
  }

  npoints <- length(sens)
  area <- sum(0.5 * (sens[-1] + sens[-npoints]) * (specc[-npoints] -
        specc[-1]))

  lift <- (sens - specc)[-1]
  cutoff <- pi.all[lift == max(lift)][1]
  sensopt <- sens[-1][lift == max(lift)][1]
  specopt <- 1 - specc[-1][lift == max(lift)][1]

  par(pty="s")
  if (newplot){
  plot(specc, sens, xlim = c(0, 1), ylim = c(0, 1), type = "s",
          xlab = "FPR = 1-specificity", ylab = "TPR = sensitivity", main="ROC")
  abline(0, 1)
  }
  else lines(specc, sens, type="s", lty=2, col=2)

  list(pihat=as.vector(pi.all), sens=as.vector(sens[-1]),
  spec=as.vector(1-specc[-1]), area = area, cutoff = cutoff,
  sensopt = sensopt, specopt = specopt)
}
```

Draw ROC curves for both training and testing data sets.

```r
train.ROC <- roc.analysis(depr.glm)

depr.testing$y <- 1*(depr.testing$Diagnosis == 1)
test.ROC <- roc.analysis(depr.glm, newdata=depr.testing, newplot=F)
```



Find the optimal *cutoff from the training* data set. Then apply it to the *testing data to compute the prediction error rate.*

```r
train.ROC[(4:7)]
```

```
## $area
## [1] 0.71496
##
## $cutoff
##      1307
## 0.1871962
##
## $sensopt
## [1] 0.6521739
##
## $specopt
## [1] 0.6798246
```

```r
cutoff <- train.ROC$cutoff
pred.prob <- predict(depr.glm, newdata=depr.testing, type="response")
predicted.class <- 1*(pred.prob > cutoff)
```

```r
# Confusion matrix
table(depr.testing$Diagnosis, predicted.class)
```

```
##    predicted.class
##       0   1
##   0 109  49
##   1  10  16
```

```r
# Correct classification rate
mean(depr.testing$Diagnosis == predicted.class)
```

```
## [1] 0.6793478
```

```r
# Error rate
1 - mean(depr.testing$Diagnosis == predicted.class)
```

```
## [1] 0.3206522
```

2.6 Extra: logistic regression for Binomial counts data

Sometimes, data may be aggregated according to unique X-combination. I.e., Each row is a summary of n_i observations that have the same X-values. The response variable(s) are the **counts** of successes (1) and failures (0). The logistic regression model can still be applied. However, note that:

- The R syntax need to be modified. One way is to use `glm(c(count.of.success, count.of.failures) ~ x1 + x2 ......)`
- We can still study the association, and predict the probabilities.
- We will NOT predict individual's classification.

Here is an example using `menarche` data in `MASS` package.

```r
library(MASS)
data(menarche)

summary(menarche)
```

```
##       Age            Total           Menarche
##  Min.   : 9.21   Min.   :  88.0   Min.   :   0.00
##  1st Qu.:11.58   1st Qu.:  98.0   1st Qu.:  10.00
##  Median :13.08   Median : 105.0   Median :  51.00
##  Mean   :13.10   Mean   : 156.7   Mean   :  92.32
##  3rd Qu.:14.58   3rd Qu.: 117.0   3rd Qu.:  92.00
##  Max.   :17.58   Max.   :1049.0   Max.   :1049.00
```

```r
## fit the logistic regression model
logit.fit <- glm(cbind(Menarche,Total-Menarche)~Age, family=binomial(link=logit),data=menarche)
summary(logit.fit)
```

```
##
## Call:
## glm(formula = cbind(Menarche, Total - Menarche) ~ Age, family = binomial(link = logit),
##     data = menarche)
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -21.22639    0.77068  -27.54   <2e-16 ***
## Age           1.63197    0.05895   27.68   <2e-16 ***
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 3693.884  on 24  degrees of freedom
## Residual deviance:   26.703  on 23  degrees of freedom
## AIC: 114.76
##
## Number of Fisher Scoring iterations: 4
```

3 LDA and QDA (Generative Models)

- LDA and QDA are parametric, generative, supervised learning algorithm for classification.

3.1 Recall the fuel economy data from Example 1.

For now, I'll only use 2-predictors: `horsepower` and `year`. This will make it easier to plot the data.

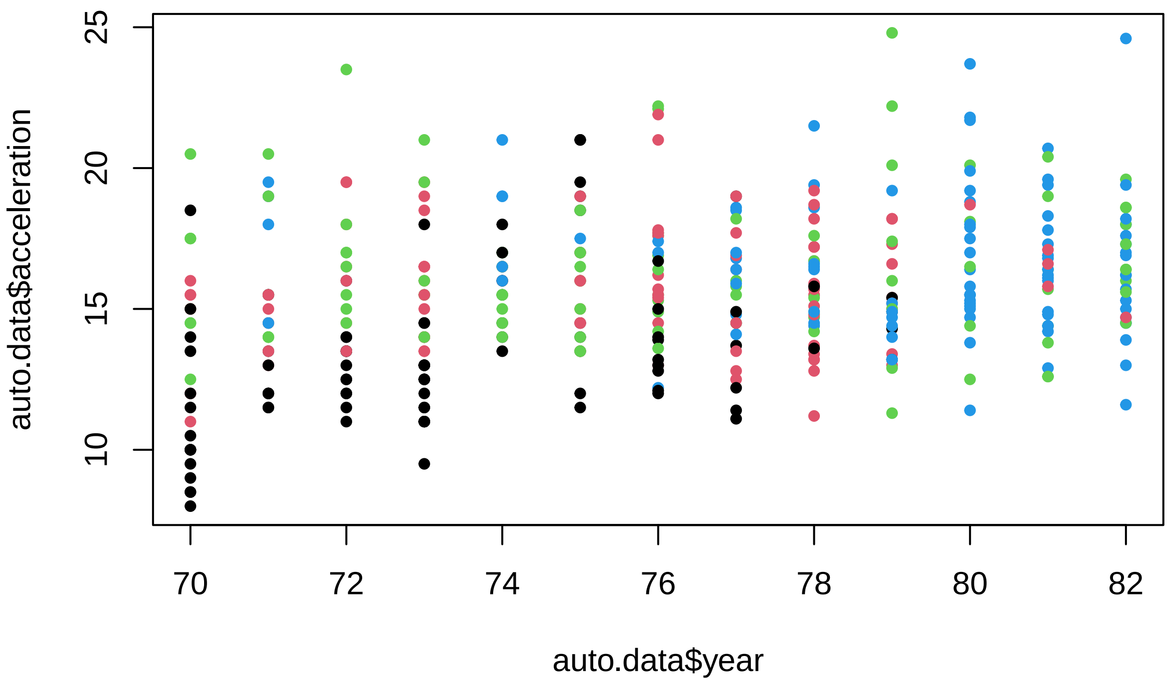
```
colnames(auto.data)
```

```
##  [1] "mpg"          "cylinders"    "displacement" "horsepower"   "weight"
##  [6] "acceleration" "year"         "origin"       "name"         "econ"
```

```
c(nrow(auto.data), nrow(auto.training), nrow(auto.testing))
```

```
## [1] 392 235 157
```

```
plot(auto.data$year, auto.data$acceleration, col=auto.data$econ, pch=20)
```



3.2 The `lda()` functions from `MASS` package

In practice, we often include `CV=TRUE` inside the the `lda()` function. See the last section for more details.

```
library(MASS)
```

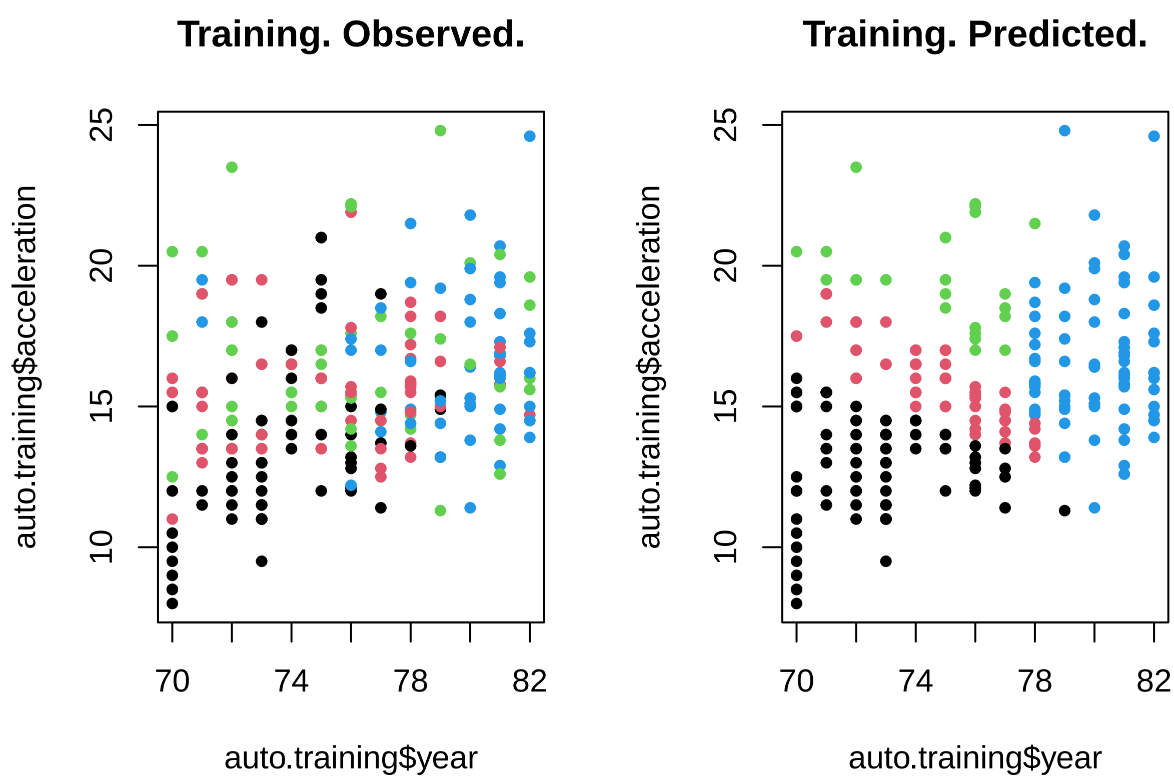
```
lda.train <- lda(econ ~ acceleration + year, data=auto.training)
lda.train
```

```
## Call:
## lda(econ ~ acceleration + year, data = auto.training)
##
## Prior probabilities of groups:
##      Poor        OK      Good Excellent
## 0.2765957 0.2680851 0.2425532 0.2127660
##
## Group means:
##           acceleration     year
## Poor          13.41846 73.55385
## OK            15.65397 75.04762
## Good          16.41579 75.68421
## Excellent     16.67800 79.34000
##
## Coefficients of linear discriminants:
##                    LD1        LD2
## acceleration 0.1920260 -0.3363007
## year         0.2621598  0.1963846
```

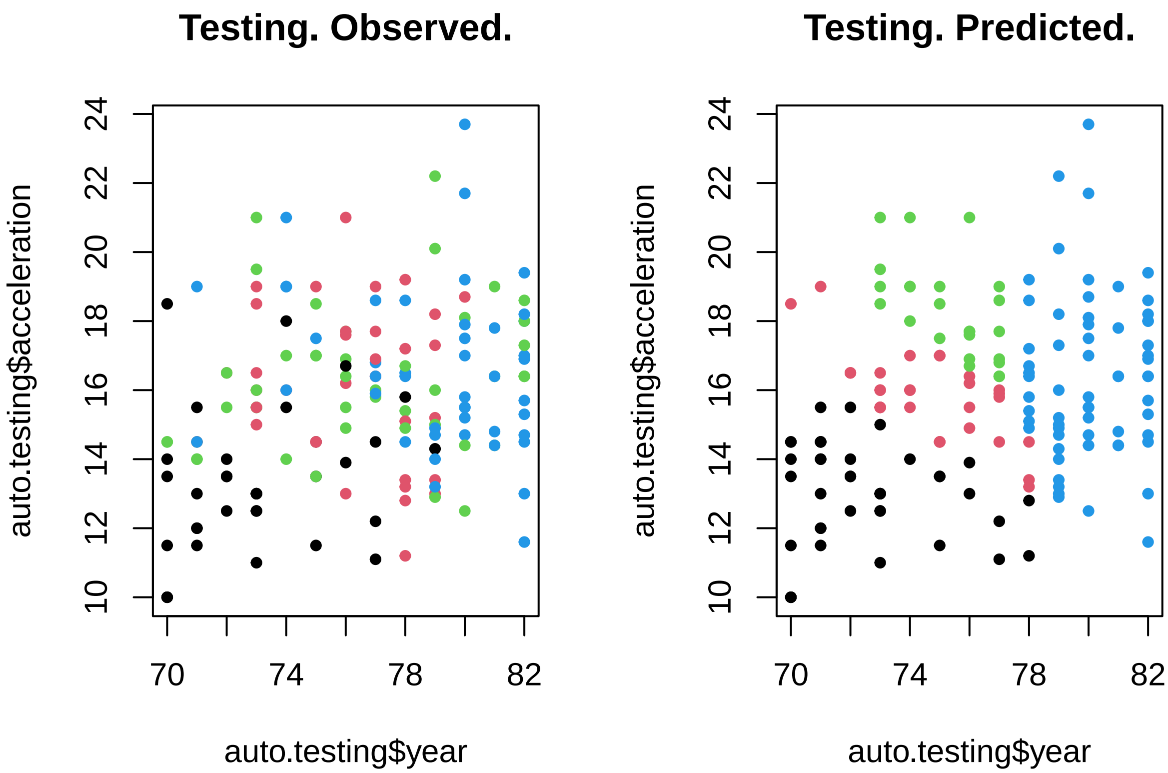
```
##
## Proportion of trace:
##    LD1    LD2
## 0.8831 0.1169
```

Demonstrate the "linear" boundary.

```r
par(mfrow=c(1, 2))
plot(auto.training$year, auto.training$acceleration, col=auto.training$econ, pch=20,
     main="Training. Observed.")
pred.train <- predict(lda.train)$class
plot(auto.training$year, auto.training$acceleration, col=pred.train , pch=20,
     main="Training. Predicted.")
```




```r
par(mfrow=c(1, 2))
plot(auto.testing$year, auto.testing$acceleration, col=auto.testing$econ, pch=20,
     main="Testing. Observed.")
pred.test <- predict(lda.train, newdata=auto.testing)$class
plot(auto.testing$year, auto.testing$acceleration, col=pred.test, pch=20,
     main="Testing. Predicted.")
```



Prediction classification rate and prediction error rate (on the testing data)

```
pred.test <- predict(lda.train, newdata=auto.testing)$class
mean(pred.test == auto.testing$econ)  # classification rate
```

```
## [1] 0.4713376
```

```
1- mean(pred.test == auto.testing$econ) # error rate
```

```
## [1] 0.5286624
```

3.3 The qda() functions from MASS package.

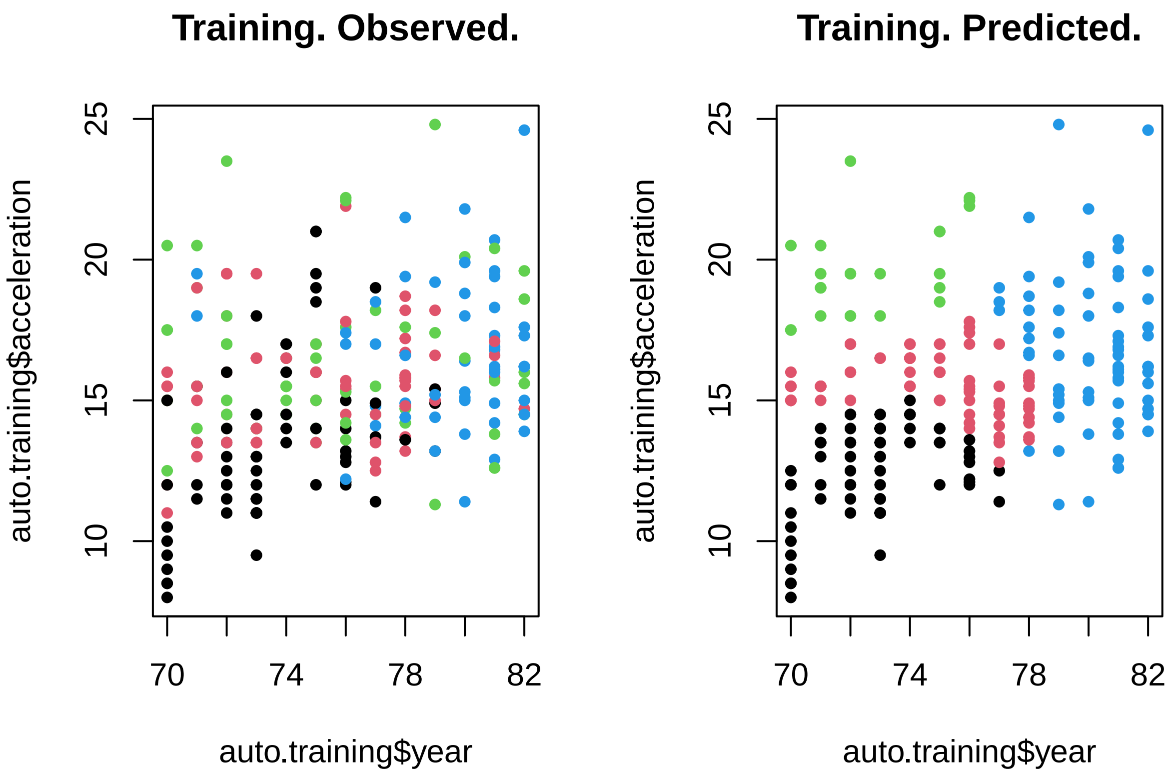
In practice, we often include `CV=TRUE` inside the the `qda()` function. See the last section for more details.

```
qda.train <- qda(econ ~ acceleration + year, data=auto.training)
qda.train
```

```
## Call:
## qda(econ ~ acceleration + year, data = auto.training)
##
## Prior probabilities of groups:
##      Poor        OK      Good Excellent
## 0.2765957 0.2680851 0.2425532 0.2127660
##
## Group means:
##           acceleration     year
## Poor          13.41846 73.55385
## OK            15.65397 75.04762
```

```
## Good           16.41579 75.68421
## Excellent      16.67800 79.34000
```

```
par(mfrow=c(1, 2))
plot(auto.training$year, auto.training$acceleration, col=auto.training$econ, pch=20,
     main="Training. Observed.")
pred.train <- predict(qda.train)$class
plot(auto.training$year, auto.training$acceleration, col=pred.train , pch=20,
     main="Training. Predicted.")
```



Prediction classification rate and error rate on the testing data.

```
pred.test <- predict(qda.train, newdata=auto.testing)$class
mean(pred.test == auto.testing$econ)   # classification rate
```

```
## [1] 0.477707
```

```
1- mean(pred.test == auto.testing$econ)  # error rate
```

```
## [1] 0.522293
```

3.4 More about the prior

By default, `lda()` and `qda()` use the **sample proportion** as the **prior**. We can set a different prior if needed.

- Use default

```r
qda.train <- qda(econ ~ acceleration + year, data=auto.training)
pred.test <- predict(qda.train, newdata=auto.testing)$class
mean(pred.test == auto.testing$econ)   # classification rate
```

```
## [1] 0.477707
```

- Use Uniform prior (equal prior probability)

```r
J <- nlevels(auto.data$econ)
qda.train2 <- qda(econ ~ acceleration + year, data=auto.training,
                  prior=rep(1/J, J))
pred.test <- predict(qda.train2, newdata=auto.testing)$class
mean(pred.test == auto.testing$econ)   # classification rate
```

```
## [1] 0.4649682
```

- Many other choices, as long as the prior probabilities add up to 1 and covers all response classes.

```r
qda.train3 <- qda(econ ~ acceleration + year, data=auto.training,
                  prior=c(0.4, 0.3, 0.1, 0.2),)
pred.test <- predict(qda.train3, newdata=auto.testing)$class
mean(pred.test == auto.testing$econ)   # classification rate
```

```
## [1] 0.4904459
```

3.5 Heads up: cross-validation

In previous R examples, we split the original data set into training and testing sets. The training data is used to fit the model, including tuning when applicable. Then the *trained* model is applied to the testing data and prediction accuracy measures are computed. The prediction accuracy measurements on the testing data provide a valid way of evaluating the model.

However, the above result is "random'': one researcher's split may be different from another researcher's split. Hence, the model accuracy assessments may be different even though people are using the same data and the same type of models.

A **Leave-One-Out** cross-validation (LOOCV) algorithm is used in `lda()` and `qda()` with `CV=TRUE`. This allows us to - Use the entire data set without splitting it *manually* - Get a valid assessment on the prediction accuracy.

```r
# Use the whole observed data with CV=TRUE
qda.cv <- qda(econ ~ acceleration + year, data=auto.data, CV=TRUE)
qda.cv$class[1:3]
```

```
## [1] Poor Poor Poor
## Levels: Poor OK Good Excellent
```

```r
qda.cv$posterior[1:3, ]
```

```
##        Poor         OK       Good    Excellent
## 1 0.8285736 0.07695698 0.09396443 0.0005049820
## 2 0.8603322 0.06744613 0.07188533 0.0003363048
## 3 0.9145679 0.03186362 0.05335148 0.0002170076
```

```r
mean(qda.cv$class == auto.data$econ)   # classification rate
```

```
## [1] 0.4719388
```