Distribution-free Approaches to Classification

- The classification methods we previously discussed (linear discriminants, quadratic discriminants, logistic regression) are based on models.

- Model-free approaches such as tree-based classification also exist and are popular for their intuitive appeal.

- In this section, we discuss two model-free approaches to classification
  - nearest-neighbor methods
  - classification trees (CART)

Nearest-Neighbor Classification

- Perhaps the simplest and most intuitive of all classification procedures is \( k \)-nearest-neighbor classification.

- The strategy for predicting the class of a new case is to identify the \( k \) closest neighbors from among the training samples and assign the case to the class that has the greatest representation among its \( k \) nearest neighbors.

- Ties may be broken at random or using some other approach.

Example 11.11 Business School Applicants

- A business school admissions officer has records on admission decisions of 87 previous applicants to the graduate program
  - Group 1: admit \( (n_1 = 31) \)
  - Group 2: do not admit \( (n_2 = 28) \)
  - Group 3: borderline \( (n_3 = 26) \)

- The officer has the following information for each student
  - GPA: undergraduate grade point average (0-4 scale)
  - GMAT: graduate management aptitude test score score (0-720)

- Good idea to standardize the GPA and GMAT scores
Example 11.11 Business School Applicants
Scaled scores: kNN classification: $k = 2, 3, 5$

- Classify a new applicant with GPA=2.85 and GMAT=457
- First standardize the scores GPA: \((2.85 - 2.97)/0.190 = -0.632\)
  and GMAT: \((457 - 488.45)/60.464 = -0.5201\)
- For $k=3$ neighbors, find the three closest cases in the training samples with respect to Euclidean distance
- If two of these 3 neighbors are in the training sample for group 3 and one is in the training sample for group 2, this case is classified into group 3.

Choosing $k$ in Nearest-neighbor Approaches

- The choice of $k$, the number of neighbors, to be included in the classification is important.
- A common choice is to take $k = 1$ but this can give rise to very irregular and jagged regions with high variances in prediction.
- Larger choices of $k$ lead to larger and more stable regions with smoother boundaries, but may not capture local details and could have larger misclassification rates if $k$ is too large.

Choosing $k$ by Cross-Validation

- Since this is a predictive problem, a choice of $k$ may be made using cross-validation. Cross-validation was used on the GMAT dataset to obtain $k = 3$ as the most optimal choice in terms of minimizing predicted misclassification error. The cross-validated error was 2.35%
- The distance function used here was Euclidean.
- The 3-nearest neighbor classification predicted the test score (after scaling) to be in category 3.

CV errors in kNN classification

The cross-validated estimates of error rates using 3 nearest neighbors

<table>
<thead>
<tr>
<th>Classification</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>31</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>28</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>24</td>
</tr>
</tbody>
</table>

The cross-validation estimate of the proportion of misclassified cases is 0.0235
**Issues with $k$-NN classification**

- $k$-nearest neighbor classification appears to use only one parameter — the number of neighbors to be included in deciding on the majority-vote classification.
- However, the effective number of parameters to be fit is not really $k$ but more like $n/k$ neighborhoods, where $n$ is the total number of cases in the training samples.
- This approach roughly divides the region containing the training set data into approximately $n/k$ neighborhoods and each neighborhood is classified according to the majority voter.
- The performance may degrade if the number of traits measured on each case increases. The $n$ required to maintain the same density of sample points increases exponentially with $p$. This is the **curse of dimensionality**.

```
# Enter the business school admissions data
gmat <- read.table(file = "c:/stat501/data/gmat.dat", header=F, col.names=c("Applicant", "Admit", "GPA", "GMAT"))
```

```
gmat
  Applicant Admit GPA GMAT
1    1     1  2.96  596
2    1     1  3.14  473
3    1     1  3.22  482
4    1     1  3.29  527
5    1     1  3.69  505
6    1     1  3.46  493
```

```
# Do one-way MANOVA on the business school data.  # First create a factor from the variable that identifies the populations

gmat$Admit <- as.factor(gmat$Admit)
fit.lm <- lm(cbind(GPA, GMAT)~Admit, data = gmat)
fit.manova <- manova(fit.lm)
summary(fit.manova, , test=c("Wilks"))
```

```
Df Wilks approx F num Df den Df Pr(>F)
Admit 2 0.12588 73.651 4 162 < 2.2e-16 ***
Residuals 82
```

```
# Attach the MASS library. After doing this you can use the 
# lda and qda functions. use help(lda) and help(qda) to get
# more information on the lda and qda functions. You can also
# use help.search("discriminant analysis") to get more information.
library(MASS)

# First apply linear discriminant analysis with equal priors

gmat.lda <- lda(Admit ~., data=gmat[, -1], prior=c(1, 1, 1)/3)
```

```
# Input data for new cases to be classified

test <- data.frame(matrix(c(2.85, 457, 3.75, 597), ncol=2, byrow=T)
names(test) <- c("GPA", "GMAT")
```
# Plot the data
plot(x=gmat$GPA, y = gmat$GMAT, pch = as.character(gmat$Admit),
col = as.character(gmat$Admit))
dev.copy2pdf(file = "c:/stat501/Examples/GMATDATA.pdf")

# Classify the new cases
predict(gmat.lda, newdata=test)
$class
[1] 3 1
Levels: 1 2 3
$posterior
 1 2 3
1 0.001206032 1.481734e-01 8.506206e-01
2 0.999985574 1.620882e-12 1.442575e-05

# obtain the cross-validated misclassification rates
gmat.lda.cv <- lda(Admit ~., data=gmat[, -1], prior=c(1, 1, 1)/3,
CV = TRUE)
gmat.lda.cv$class
table(gmat$Admit, gmat.lda.cv$class)

1 2 3
1 1 7 0 4
2 20 2 6
3 1 2 4

# cross-validated estimate of overall proportion
# of cases that will be misclassified
mean(gmat$Admit!=gmat.lda.cv$class)
[1] 0.09411765

# Use quadratic discriminant analysis

gmat.qda<-qda(Admit ~., data=gmat[, -1], prior=c(1, 1, 1)/3)

# Classify new cases:
predict(gmat.qda, newdata=test)
$class
[1] 3 1
Levels: 1 2 3
$posterior
 1 2 3
1 0.005565686 9.393509e-02 9.004992e-01
2 1.000000000 2.088337e-12 3.249404e-18
# Obtain the cross-validated error rate

```r
gmat.qda.cv <- qda(Admit ~., data=gmat[,-1], prior=c(1, 1, 1)/3, CV = TRUE)
table(gmat$Admit, gmat.qda.cv$class)
```

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>30</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0</td>
<td>27</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>24</td>
</tr>
</tbody>
</table>

# Cross-validated estimate of overall misclassification rate for QDA

```r
mean(gmat$Admit!=gmat.qda.cv$class)
```

[1] 0.04705882

# Use the k-nearest neighbors classification procedure

# Note the the function knn needs a "test" dataset to come up with a classification rule. Attach the R library "class" for k-nearest neighbor classification.

```r
library(class)

# GPA and GMAT scores are on vastly different scales, so we should scale them

```r
gmat.scaled <- scale(gmat[, c(3, 4)])
```

# Plot the scaled data

```r
plot(x=gmat.scaled[, c("GPA")], y = gmat.scaled[, c("GMAT")], pch = as.character(gmat$Admit), col = as.character(gmat$Admit))
```
# Apply knn classification with three nearest neighbors

gmat.knn.sc <- knn(train = gmat.scaled[, c("GPA", "GMAT")], test  
gmat.scaled[, c("GPA", "GMAT")], cl = gmat$Admit, k = 3)

# New cases need to be centered and scaled to match the data points

test.sc <- scale(test, center=attr(gmat.scaled, "scaled:center"),  
scale=attr(gmat.scaled, "scaled:scale"))

test.sc <- scale(test, center=attr(gmat.scaled, "scaled:center"),  
scale=attr(gmat.scaled, "scaled:scale"))

test.sc <- scale(test, center=attr(gmat.scaled, "scaled:center"),  
scale=attr(gmat.scaled, "scaled:scale"))

knn(train = gmat.scaled[, c("GPA", "GMAT")], test = test.sc,  
prob = T, cl = gmat$Admit, k = 3)

[1] 3 1
attr("prob")
[1] 1 1
Levels: 1 2 3

# Get the cross-validated error rate using k-nearest neighbors.
# Note that the function knn.cv provides the class for each
# case using a leave-out-1 cross-validation.
# We will do this for k = 1, 2, ..., 10 neighbors and choose
# the k with the smallest misclassification error rate.
# If this takes too long try reducing the 10000 replications
down to 1000 or so
# We replicate because ties are broken at random.

knn.cv.err<-NULL
knn.cv.sd<-NULL
for (i in seq(1, 10, by=1)) {
  temp<NULL
  for (j in 1:10000)
    temp <- c(temp,mean(knn.cv(gmat.scaled[, c("GPA", "GMAT")],  
                            cl = gmat$Admit, k = i) != gmat$Admit))
  knn.cv.err<-c(knn.cv.err,mean(temp))
  knn.cv.sd<-c(knn.cv.sd,sd(temp))
  cat("\n Done i= ",i)
}

plot(y=knn.cv.err, x=seq(1,10,by=1), xlim = c(1, 10),  
ylim=c(min(knn.cv.err - 1.96 * knn.cv.sd),  
      max(knn.cv.err + 1.96 * knn.cv.sd)), type = "n")
lines(y=knn.cv.err + 1.96 * knn.cv.sd, x=seq(1,10,by=1),  
lty = 2, col = "blue")
lines(y=knn.cv.err - 1.96 * knn.cv.sd, x=seq(1,10,by=1),  
lty = 2, col = "green")
lines(y=knn.cv.err, x=seq(1,10,by=1), col = "red")
# Show how cases are classified

gmat.knn.sc <- knn(train = gmat.scaled[, c("GPA", "GMAT")],
                   test = gmat.scaled[, c("GPA", "GMAT")], cl = gmat$Admit, k = 3)

  gmat.knn.sc
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
  [29] 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
  [57] 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
  [85] 3
  Levels: 1 2 3

knn(train = gmat.scaled[, c("GPA", "GMAT")], test = test.sc,
     prob = T, cl = gmat$Admit, k = 3)

  [1] 3 1
  attr("prob")
  [1] 1
  Levels: 1 2 3

# Print the cross-validated classification table for k=3

gmat.knn.cv <- knn.cv(gmat.scaled[, c("GPA", "GMAT")],
                      cl = gmat$Admit, k = 3)
table(gmat$Admit, gmat.knn.cv)

  gmat.knn.cv
  1 2 3
  1 31 0 0
  2 0 28 0
  3 1 1 24

  mean(gmat.knn.cv != gmat$Admit)
  [1] 0.02352941

Classification and Regression Trees (CART)

- CART is a simple and frequently used technique for classification.
- The CART algorithm generates a classification tree by sequentially doing binary splits on the data.
- The simplest case is when splits are made on individual variables.
Basic Details of CART

- The basic form of the procedure partitions the space of measurements on \( p \) traits into a set of disjoint \( p \)-dimensional hyper-rectangles denoted by \( R_1, R_2, \ldots, R_k \).
- The expected or predicted response is constant within each hyper-rectangle.
- For the classification situation, the response is a class indicator while for the regression case, it is a continuous numerical value.

For regression, minimize the sum of squared residuals within the \( k \) hyper-rectangles \( R_1, R_2, \ldots, R_k \).

- Compute the sample mean response within \( R_i \)
  
  \[
  \bar{Y}_i = \sum_{j \in R_i} Y_j
  \]

- Compute \( RSS_i = \sum_{j \in R_i} (Y_j - \bar{Y}_i)^2 \)

- To go from \( k \) to \( k+1 \) rectangles split one of the previous rectangles to minimize \( RSS_{k+1} = \sum_{j=1}^{k+1} RSS_j \)

The regression tree is built in a hierarchical manner and the best additional split depends on the \( k \) hyperrectangles available from the previous step.

For classification, a node (or hyper-rectangle) impurity measure is often minimized.

- A hyper-rectangle is pure if only cases from just one of the populations exist in the hyper-rectangle.
- Let \( (\pi_{j1}, \pi_{j2}, \ldots, \pi_{ji}) \) denote the probability that a randomly selected case from \( R_j \) is a member of the \( i \)-th population.
- Then, \( R_j \) is pure if \( \pi_{ji} = 1 \) for some \( i \) and zero for all other populations.

Define an objective function to minimize within \( R_j \) that attains its minimum when \( R_j \) is pure, e.g.

- the Gini index
- an information kernel

To go from \( k \) to \( k+1 \) rectangles split one of the previous rectangles to minimize the sum of the impurity measures for the new set of \( k+1 \) rectangles.

The classification tree is built in a hierarchical manner and the best additional split depends on the \( k \) hyperrectangles available from the previous step.
Basic Details of CART

- The **key question** is how to use training samples to determine the number of splits $k$ and the actual splits that define the rectangles $R_1, R_2, \ldots, R_k$.
- Start with a training sample cases in one large rectangle
- At each step increase the number of rectangles by one, splitting one of the previous rectangles into two parts

Once we have the first split, we fix it and apply the splitting procedure to each of the two resulting rectangles.

As a result, we will not get the optimal partitioning of the space into $k = 3$ rectangles, but we will split one of the rectangles from the first split to get three rectangles.

We thus get a tree, with each node being associated with a rectangular region $R_i$.

As more rectangles are split and the tree grows, the rectangles (terminal nodes) become more pure.

Note that estimates of population proportions become more unstable as we move farther down the tree because the estimates are based on fewer cases from the training samples.

Choose the split that optimizes the objective function for splitting a previous rectangle into two parts
- Must select one of the $k$ previous rectangles to split
- For each of the previous rectangles you must select one of the $p$ measured traits to use to make the split
- Must also find the best boundary value to use to make the split for the selected variable

This hierarchical approach generally will not find a globally optimal set of hyper-rectangles.
Example 11.11 (CART on GMAT Data)

- To classify case with measured traits $x_i$, we simply run it down the tree until it falls into a terminal node.
- Use the training sample data to estimate the proportion of cases that belong to each of the populations.
- Classify the new case into the population estimated to have the highest probability for that terminal node.
- Most tree building algorithms use some type of cross validation procedure to get the estimated proportions for the terminal nodes.
- Cross validation is also used to determine the number of terminal nodes. The basic idea is to make a tree that is too big and prune it back.

Terminating the Tree

- One option is to start with a very large tree and use bottom-up recombination.
- For the $t$-th node, denote the left and right daughters by $l(t)$ and $r(t)$.
- Define the cost of the $t$-th node as $c(t) = \text{Gini index}$ if it is a terminal node.
- Define the cost of splitting the $t$-th node as $c(t) = c(l(t)) + c(r(t)) + \lambda$.
- We consider splitting the $t$-th node worthwhile if the cost incurred in splitting it is smaller than the cost of making it a terminal node.
- The parameter $\lambda$ is called the complexity parameter and it specifies the price we pay for splitting a node and thereby creating a more complex model.
- Note that if $\lambda \geq \lambda'$, then a tree built using $\lambda$ can be obtained by pruning the tree built using $\lambda'$.
- For $\lambda \to \infty$, the tree consists of only one node.
- Finally, if the tree has $k$ terminal nodes, there are at most $k$ different subtrees that can be obtained by choosing larger values of $\lambda$. 

Terminating the Tree
Cross-validation to determine tree size

# Now do the classification using CART. First install the # tree package and then attach the tree library
library(tree)

# Use all columns of the data frame except the first column
gmat.tree <- tree(Admit ~ ., data = gmat[, -1])

summary(gmat.tree)

classification tree:
tree(formula = Admit ~ ., data = gmat[, -1])
Number of terminal nodes:  7
Residual mean deviance:  0.2589 = 20.19 / 78
Misclassification error rate:  0.07059 = 6 / 85

gmat.tree
node, split, n, deviance, yval, (yprob)  * denotes terminal node
1) root 85 186.30 1 ( 0.36471 0.32941 0.30588 )
   2) GPA < 2.725 26 0.00 2 ( 0.00000 1.00000 0.00000 ) *
      3) GPA > 2.725 59 96.05 1 ( 0.52542 0.03390 0.44068 )
         6) GPA < 3.17 29 33.39 3 ( 0.10345 0.06897 0.82759 )
            12) GMAT < 415 5 6.73 3 ( 0.00000 0.40000 0.60000 ) *
            13) GMAT > 415 24 18.08 3 ( 0.12500 0.00000 0.87500 )
               26) GMAT < 472 14 0.00 3 ( 0.00000 0.00000 1.00000 ) *
               27) GMAT > 472 10 12.22 3 ( 0.30000 0.00000 0.70000 )
                  54) GPA < 2.875 5 0.00 3 ( 0.00000 0.00000 1.00000 ) *
                  55) GPA > 2.875 5 6.73 1 ( 0.60000 0.00000 0.40000 ) *
         7) GPA > 3.17 30 14.70 1 ( 0.93333 0.00000 0.06667 )
            14) GMAT < 474.5 5 6.73 1 ( 0.60000 0.00000 0.40000 ) *
            15) GMAT > 474.5 25 0.00 1 ( 1.00000 0.00000 0.00000 ) *

Our prediction using the best cross-validated tree is that the new applicant belongs to the first category.
# Plot the classification tree
plot(gmat.tree)
text(gmat.tree, cex=0.7)

# Now let us decide on getting the optimal tree using cross-validation. This is done using the function cv.tree.
# Note that the argument K denotes the number of folds of the data in the cross-validation procedure. If we wanted leave-one-out cross-validation, we would set K to be the number of observations in the data set.

gmat.tree.cv <- cv.tree(gmat.tree, K = nrow(gmat))

# Plot the results.
plot(gmat.tree.cv)

# Try k=10 fold cross-validation

gmat.tree.cv <- cv.tree(gmat.tree, K = 10)

# Plot the results.
plot(gmat.tree.cv)
It appears that a tree with 4 nodes has the lowest cross-validated misclassification rate.

# Prune the branches of the tree

gmat.prune.tree <- prune.tree(gmat.tree, best = 4)

plot(gmat.prune.tree)
text(gmat.prune.tree)

# Create a data frame with test cases

test <- data.frame(matrix(c(2.85, 457, 3.75, 597),
                          ncol=2, byrow=T))

names(test) <- c("GPA", "GMAT")

test

GPA  GMAT
1  2.85  457
2  3.75  597
# You can only run one case down the tree at a time
# with tree "predict" function

predict(gmat.prune.tree, newdata=test[1, ], type=c("vector"))
   1 2 3
1 0.125 0 0.875

predict(gmat.prune.tree, newdata=test[2, ], type=c("vector"))
   1 2 3
2 0.9333333 0 0.06666667

# Find the overall misclassification rate on the dataset

mean(apply(predict(gmat.prune.tree), 1, which.max)!=gmat$Admit)
[1] 0.08235294

Summary

- It appears that QDA does the best for the Business Graduate Program admissions data (misclassification rate = 4.71%)
- Followed by CART (misclassification rate = 8.23%)
  - LDA (misclassification rate = 9.41%)
  - Logistic Regression (could do perfect separation??)
- The estimate of the overall misclassification rate for the k-nearest neighbors procedure with k=3 neighbors indicated a misclassification rate of 2.35%, but this may be biased.
- CART may be useful for discovering interactions that could be included in a logistic regression or an LDA or QDA approach