STAT 730 Chapter 11: Discriminant analysis

Timothy Hanson

Department of Statistics, University of South Carolina

Stat 730: Multivariate Data Analysis

Basic idea

Clustering is called "unsupervised learning" in the machine learning literature; discriminant analysis (or classification) is termed "supervised learning." Really discriminant analysis and classification are slightly different actions, but they are used interchangeably.

Now the number of groups g is known, as is *the group membership* of each object. For example, in the dental data we know whether each child is male or female ($g = 2$); for the iris data we know which of the three species ($g = 3$) each flower belongs to. The data are $\{(\mathbf{x}_i, z_i)\}_{i=1}^n$ where $\mathbf{x}_i$ is a vector of (initially continuous) variables and z_i tells us which group $\mathbf{x}_i$ belongs to.

We ultimately want to classify an object with $\mathbf{x}_{n+1}$, i.e. find $\hat{z}_{n+1}$.

Model-based classification

By model-based, here we mean the same mixtures of Gaussian models are used as in model-based clustering.

We will start with classification that follows a generalization of the parametric MANOVA model:

$$\mathbf{x}_i \overset{ind.}{\sim} N_p(\boldsymbol{\mu}_{z_i}, \boldsymbol{\Sigma}_{z_i}),$$

where $i = 1, \ldots, n$ and $z_i \in \{1, \ldots, g\}$. Note that the "usual" MANOVA model occurs when $\boldsymbol{\Sigma}_1 = \cdots = \boldsymbol{\Sigma}_g = \boldsymbol{\Sigma}$. Also note that the $z_1, \ldots, z_n$ are *known* in contrast to clustering.

Marginal distribution

In the population governing $\{(\mathbf{x}_i, z_i)\}_{i=1}^n$, a pair $(\mathbf{x}, z)$ is generated according to the joint pdf/pmf $f(\mathbf{x}, z) = f(\mathbf{x}|z)p(z)$ where $(\mathbf{x}, z) \in \mathcal{X} \times \{1, \ldots, g\}$. Define $\pi_j = P(z = j)$; π_j is the proportion of vectors $\mathbf{x}$ that come from sub-population j. By the LTP, the marginal distribution of (an *unclassified* $\mathbf{x}$ with unknown z) is

$$\mathbf{x} \sim f(\mathbf{x}) = \sum_{j=1}^{g} f_j(\mathbf{x})\pi_j,$$

where $f_j(\mathbf{x})$ is the density of the jth sub-population.

For different covariance matrices, $f_j(\mathbf{x}) = \phi_p(\mathbf{x}; \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$, for a common covariance matrix $f_j(\mathbf{x}) = \phi_p(\mathbf{x}; \boldsymbol{\mu}_j, \boldsymbol{\Sigma})$.

Fitting

Let $n_j = \sum_{i=1}^{n} I\{z_i = j\}$; note $n_1 + \cdots + n_g = n$. As usual, $\hat{\boldsymbol{\mu}}_j = \frac{1}{n_j} \sum_{i:z_i=j} \mathbf{x}_i$. When the covariance matrices are different, $\hat{\boldsymbol{\Sigma}}_j = \frac{1}{n_j} \sum_{i:z_i=j} (\mathbf{x}_i - \hat{\boldsymbol{\mu}}_j)(\mathbf{x}_i - \hat{\boldsymbol{\mu}}_j)'$. If $\boldsymbol{\Sigma}_1 = \cdots = \boldsymbol{\Sigma}_g = \boldsymbol{\Sigma}$ then $\hat{\boldsymbol{\Sigma}} = \frac{1}{n} \sum_{i=1}^{n} (\mathbf{x}_i - \hat{\boldsymbol{\mu}}_{z_i})(\mathbf{x}_i - \hat{\boldsymbol{\mu}}_{z_i})'$.

Replacing population moments with sample estimates gives $\hat{f}_j(\mathbf{x})$.

If the data $\{(\mathbf{x}_i, z_i)\}_{i=1}^{n}$ are randomly drawn from the overall population, then $\hat{\pi}_j = \frac{n_j}{n}$, otherwise they must be estimated in some other way.

Unlike clustering, the z_i are known; the EM algorithm is not needed!

Classifying via Bayes' rule

For an unclassified $\mathbf{x}$ from one of the g groups, Bayes' rule gives us the probability of being in group j:

$$P(z=j|\mathbf{x}) = \frac{f(\mathbf{x}|z=j)P(z=j)}{f(\mathbf{x})} \approx \frac{\hat{f}_j(\mathbf{x})\hat{\pi}_j}{\sum_{s=1}^{g} \hat{f}_s(\mathbf{x})\hat{\pi}_s}.$$

Classification rule: Let $d_j(\mathbf{x}) = \hat{f}_j(\mathbf{x})\hat{\pi}_j$. Then $\mathbf{x}$ is classified as coming from sub-population j when $d_j(\mathbf{x}) > d_s(\mathbf{x})$ for all $s \neq j$. That is, an observation is estimated to belong to the group with the highest posterior probability.

This classification scheme assumes that the cost of misclassification is the same across all possibilities. J & W define c_{js} to be the cost of saying $\mathbf{x}$ comes from group j when it really comes from s. Considering the expected cost of misclassification, they modify this rule to $d_j(\mathbf{x})c_{sj} > d_s(\mathbf{x})c_{js}$. Throughout MKB, the costs for misclassification are assumed equal. MKB pp. 305–308 place classification within the realm of decision theory.

Linear classification rules

The simplifying case where $\pi_1 = \cdots = \pi_g$ yields so-called ML discriminant rules. If $\boldsymbol{\Sigma}_1 = \cdots = \boldsymbol{\Sigma}_g$, then the ML discriminant rule allocates $\mathbf{x}$ to group j instead of group k when

$$\phi_p(\mathbf{x}; \hat{\boldsymbol{\mu}}_j, \hat{\boldsymbol{\Sigma}}) > \phi_p(\mathbf{x}; \hat{\boldsymbol{\mu}}_k, \hat{\boldsymbol{\Sigma}}).$$

Taking logs, this boils down to

$$(\hat{\boldsymbol{\mu}}_j - \hat{\boldsymbol{\mu}}_k)\hat{\boldsymbol{\Sigma}}^{-1}\{\mathbf{x} - \tfrac{1}{2}(\hat{\boldsymbol{\mu}}_j + \hat{\boldsymbol{\mu}}_k)\} > 0.$$

This results in g $(p-1)$-dimensional hyperplanes partitioning $\mathbb{R}^p$; each of the g regions classifies an $\mathbf{x}$ to one of the g groups.

This corresponds to Fisher's linear discriminant rule, see pp. 318–320. Fisher's rule was not originally derived considering normality though.

Quadratic classification rules

If we rather assume different covariance matrices, then the ML discriminant rule allocates $\mathbf{x}$ to group j instead of group k when

$$\phi_p(\mathbf{x}; \hat{\boldsymbol{\mu}}_j, \hat{\boldsymbol{\Sigma}}_j) > \phi_p(\mathbf{x}; \hat{\boldsymbol{\mu}}_k, \hat{\boldsymbol{\Sigma}}_k).$$

This results in a partitioning of $\mathbb{R}^p$ into g portions separated by quadratic surfaces. See Marden p. 230.

Generalizing so that the $\pi_1, \ldots, \pi_g$ are allowed to be different, as we first discussed, also gives linear and quadratic classification rules that are only slightly more complex (Marden p. 225 and p. 230).

Classification error

An error occurs when $\mathbf{x}_i$ is not classified correctly; recall we are considering that the cost of misclassification is the same over all possibilities. Let z_i be the truth and $\hat{z}_i$ be classification of $\mathbf{x}_i$ from a model. An estimate of the probability of misclassification is

$$E_r = \tfrac{1}{n} \sum_{i=1}^{n} I\{z_i \neq \hat{z}_i\}.$$

This underestimates the actual error; a better estimate is the leave-one-out, or cross-validated estimate

$$E_c = \tfrac{1}{n} \sum_{i=1}^{n} I\{z_i \neq \hat{z}_{i(i)}\},$$

where $\hat{z}_{i(i)}$ is the classification for $\mathbf{x}_i$ based on the $n-1$ pairs $\{(\mathbf{x}_j, z_j)\}_{j\neq i}$.

Using E_c instead of E_r has the same spirit as using PRESS instead of SSE in regression to estimate prediction error.

Cross-validated error estimate approaches

- Random subsampling takes K random splits of data into training and test portions, computes the classification error for each split $E_1, \ldots, E_K$, and computes $E_s = \frac{1}{K}\sum_{i=1}^{K} E_i$.
- K-fold cross-validation partitions data into $C_1 \cup \cdots \cup C_K$ of sizes roughly n/K each. Use $\cup_{j \neq i} C_j$ for training, C_i for testing giving E_i. $E_k = \frac{1}{K}\sum_{i=1}^{K} E_i$. Every obsrvation is predicted only once & all obs. are used for training. Most common: $K = 10$.
- E_c (previous slide) is K-fold using $K = n$.

Large K increases computation time, slightly increases variance of misclassification estimator, but decreases bias. Small K decreases computation time, increases bias, but decreases the variance. For random subsampling, variance is low and bias can be moderate. The bootstrap can also be used (although appears to be biased for large p). Note: there are *tons* of papers on this. Molinaro, Simon and Pfeiffer (2005, *Bioinformatics*) suggest leave-one-out cross validation unless the computational burden is too great, then they suggest 10-fold cross-validation.

Visualizing results

A common plot to look at are the first two canonical covariates (from MANOVA) along with the classification of object i. Let $\mathbf{W} = \sum_{i=1}^{n}(\mathbf{x}_i - \hat{\boldsymbol{\mu}}_{z_i})(\mathbf{x}_i - \hat{\boldsymbol{\mu}}_{z_i})'$ and $\mathbf{B} = \sum_{i=1}^{n}(\hat{\boldsymbol{\mu}}_{z_i} - \tilde{\boldsymbol{\mu}})(\hat{\boldsymbol{\mu}}_{z_i} - \tilde{\boldsymbol{\mu}})'$. Here, $\tilde{\boldsymbol{\mu}} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i$. Then take $\mathbf{a}_j = \hat{\gamma}_{(j)}/\sqrt{\hat{\gamma}_{(j)}'\hat{\boldsymbol{\Sigma}}^{-1}\hat{\gamma}_{(j)}}$ where $\mathbf{W}^{-1}\mathbf{B} = \hat{\boldsymbol{\Gamma}}\hat{\boldsymbol{\Lambda}}\hat{\boldsymbol{\Gamma}}'$. Then $y_{ij} = \mathbf{a}_j'\mathbf{x}_i$ is the jth canonical covariate on the ith object. Here, $j = 1, \dots, \min\{g-1, p\}$.

Canonical covariates maximize variability *among groups* in each of p directions $\hat{\gamma}_{(1)}, \dots, \hat{\gamma}_{(p)}$, iteratively, much like PCA. A default plot from the `lda` and `qda` functions from the `MASS` package provides this type of plot.

Variation is maximized assuming $\boldsymbol{\Sigma}_1 = \cdots = \boldsymbol{\Sigma}_g$, the usual MANOVA model.

Classifying gender using dental data

```
library(reshape) # to create data frame from dental data
library(MASS)    # has lda and qda functions
library(heavy)   # has dental data
data(dental)
d2=cast(melt(dental,id=c("Subject","age","Sex")),Subject+Sex~age)
names(d2)[3:6]=c("d8","d10","d12","d14")

# these functions estimate pi_i using sample proportions
# you can provide other pi_i if needed
f1=lda(x=as.matrix(d2[,3:6]),grouping=d2[,2],CV=T)
f1=lda(Sex~d8+d10+d12+d14,data=d2,CV=T)
f2=qda(Sex~d8+d10+d12+d14,data=d2,CV=T)
sum(f1$class!=d2$Sex)/length(d2$Sex) # CV error linear
sum(f2$class!=d2$Sex)/length(d2$Sex) # CV error quadratic
f1=lda(Sex~d8+d10+d12+d14,data=d2)   # refit without CV
f2=qda(Sex~d8+d10+d12+d14,data=d2)   # refit without CV
plot(f1) # uses disciminant functions

library(klaR) # provides panel of bivariate plots w/ regions
partimat(Sex~d8+d10+d12+d14,data=d2,method="lda")
partimat(Sex~d8+d10+d12+d14,data=d2,method="qda")
partimat(Sex~d8+d10+d12+d14,data=d2,method="svmlight") # needs additional insta
partimat(Sex~d8+d10+d12+d14,data=d2,method="rpart") # classification tree
```

Some extentions

The g populations need not be normal. Nonparametric estimates $\hat{f}_j(\mathbf{x})$ can be used instead of $\phi_p(\mathbf{x}; \hat{\boldsymbol{\mu}}_j, \hat{\boldsymbol{\Sigma}}_j)$. These include kernel-smoothed estimates, mixtures of normals within each population, etc. Something that has not been tried: Polya trees. Very easy and fast to fit.

If p is too large, e.g. $p > g$, then one can instead use PCA and take the first few principal components for use in model-based classification. One can also use so-called discriminant functions, i.e. canonical covariates from a few slides back.

We will now consider alternatives to the model-based classification rules, and are entering the realms of machine learning and data mining. A classic, free book is *The Elements of Statistical Learning* by Hastie, Tibshirani, and Friedman. You can download the PDF from

`http://statweb.stanford.edu/~tibs/ElemStatLearn/`

There is also a newer R companion book.

Logistic regression

When $g = 2$, an alternative to model-based classification is logistic regression; assume now that $z_i \in \{0, 1\}$. There are two classification probabilities $\pi(\mathbf{x})$ $(z = 1)$ and $1 - \pi(\mathbf{x})$ $(z = 0)$. The model is

$$\log \frac{\pi(\mathbf{x})}{1 - \pi(\mathbf{x})} = \beta_0 + \boldsymbol{\beta}'\mathbf{x},$$

where $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_p)'$, commonly fit using maximum likelihood treating $z_i \overset{ind.}{\sim} \text{Bern}\{\pi(\mathbf{x}_i)\}$.

Note then, that $\mathbf{x} \in \mathbb{R}^p$ is allocated to the $z = 1$ group when $\hat{\beta}_0 + \hat{\boldsymbol{\beta}}'\mathbf{x} > 0$: the two classification regions are separated by a $(p - 1)$-dimensional hyperplane in $\mathbb{R}^p$, similar to Fisher's linear rule.

Logistic regression

A more powerful version is additive logistic regression

$$\log \frac{\pi(\mathbf{x})}{1-\pi(\mathbf{x})} = \beta_0 + g_1(x_1) + \cdots + g_p(x_p).$$

This can provide nonlinear boundaries in $\mathbb{R}^p$. Additive logistic regression is available in `gam`. There is also locally-weighted logistic regression and kernel logistic regression for nonlinear classification.

Logistic regression can be thought of as providing a "nonparametric" linear discriminant rule. The model-based approaches can be more efficient when the assumptions on $f_0(\cdot)$ and $f_1(\cdot)$ are reasonable. Logistic regression makes no assumptions on these densities and can work better when assumptions on the densities are false.

$\pi_1 = P(z_i = 1)$ and $\pi_0 = P(z_i = 0)$

If we use the classification as is, we are implicitly assuming that the $(\mathbf{x}_i, z_i)$ are arriving *iid* from $f(\mathbf{x}, z)$ and $\hat{\pi}_1 = \frac{1}{n}\sum_{i=1}^{n} I\{z_i = 1\}$ estimates π_1. If, instead, the true probabilities in the population (π_1, π_0) are different (and known) from the proportions $(\hat{\pi}_0, \hat{\pi}_1)$ in the data set, we need to modify the intercept to rather be $\tilde{\beta}_0 = \hat{\beta}_0 + \log(\pi_1/\pi_0) - \log(\hat{\pi}_1/\hat{\pi}_0)$ in our decision rule.

Groups are sampled differently from cross-sectional (*iid*) in the case of rare events or planned experiments, e.g. case-control sampling. This is also called product-binomial sampling.

Logistic regression with large p

The LASSO (least absolute shrinkage and selection operator) and elastic net are two regularized regression approaches. "Regularization" broadly means to impose contraints to solve overparameterized problems. LASSO and elastic net both shrink logistic regression coefficients toward zero, and both are formulated as penalized regression. The LASSO maximizes

$$L_\lambda(\boldsymbol{\beta}) = \tfrac{1}{n} \log \left\{ \prod_{i=1}^{n} \pi(\mathbf{x}_i)^{z_i} [1 - \pi(\mathbf{x}_i)]^{1-z_i} \right\} - \lambda \sum_{j=1}^{p} |\beta_j|,$$

where $\pi(\mathbf{x}) = \frac{e^{\beta_0 + \boldsymbol{\beta}'\mathbf{x}}}{1 + e^{\beta_0 + \boldsymbol{\beta}'\mathbf{x}}}$ and $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_p)'$.

Often λ is chosen through k-fold cross-validation, i.e. chosen to minimize prediction error. Typically covariates are standardized to have unit variance before using LASSO.

Classifying gender using dental data

```
library(glmnet)
lambda=cv.glmnet(as.matrix(d2[,3:6]),d2[,2],family="binomial")$lambda.min
f=glmnet(as.matrix(d2[,3:6]),d2[,2],family="binomial")
p=predict(f,newx=as.matrix(d2[,3:6]),s=lambda,type="class")
plot(f,xvar="lambda") # log(lambda)=-2.9; only one measurement is used!
cverror=0 # cross-validated prediction error
for(i in 1:dim(d2)[1]){
 lambda=cv.glmnet(as.matrix(d2[-i,3:6]),d2[-i,2],family="binomial")$lambda.min
 f=glmnet(as.matrix(d2[-i,3:6]),d2[-i,2],family="binomial")
 p=predict(f,newx=as.matrix(d2[i,3:6]),s=lambda,type="class")
 if(d2[i,2]!=p){cverror=cverror+1}
}
cverror/dim(d2)[1]
```

Uses cross-validation to find $\hat{\lambda}$. Note that LASSO can shrink coefficients to zero! That is why it is also useful for variable selection. It provides an "automatic" stepwise procedure. As $\lambda \to 0^+$, the usual MLE estimates from logistic regression are obtained. Another option instead of LASSO would be a best subset approach if p is of moderate size.

Boosting

Boosting is a recent classification scheme first put forth in the machine learning literature, popularly termed AdaBoost. Friedman, Hastie, and Tibshirani (2000) show that boosting is equivalent to a particular additive logistic regression,

$$\log \frac{\pi(\mathbf{x})}{1-\pi(\mathbf{x})} = \sum_{m=1}^{M} h_m(\mathbf{x}).$$

The h_m functions have special structure (otherwise the model is over-parameterized).

Can be performed using `LogitBoost` in the `caTools` package, `ada` in the `ada` package. Also see the `mboost` package for a more recent treatment and `gbm`. Logit-boost typically performs better that AdaBoost on noisy data and/or misclassified training data.

Boosting can automatically perform "feature detection," i.e. figure out which elements of **x** are really needed to classify, whereas SVM cannot.

Support vector machines

Consider data $\{(\mathbf{x}_i, z_i)\}_{i=1}^n$ where $z_i \in \{-1, +1\}$. SVM, due to V. Vapnik, are classifiers that split a p-dimensional feature space into two portions $\mathbb{R}^p = R_p \cup R_n$, separated by a $(p-1)$-dimensional hyperplane, each portion classifies a $\mathbf{x}$ falling into it as $+1$ or -1. For data that are truly linearly separated, two maximally separating parallel hyperplanes are visualized

$$\mathbf{w}'\mathbf{x} - b_p = 1, \quad \mathbf{w}'\mathbf{x} - b_n = -1,$$

that separate the $+1$ from the -1; I'll draw a picture. The separating hyperplane solving $\mathbf{w}'\mathbf{x} - b = 0$ is in the middle of these two.

Finding $(\mathbf{w}, b)$ reduces (not obvious) to minimizing $\frac{1}{2}||\mathbf{w}||^2$ subject to $z_i(\mathbf{w}'\mathbf{x}_i - b) \geq 1$. If the $+1$ and -1 points overlap, the soft-margin classifier instead minimizes $\frac{1}{2}||\mathbf{w}||^2 + c\sum_{i=1}^n \xi_i$, where $\xi_i \geq 0$ with many $\xi_i = 0$, subject to $z_i(\mathbf{w}'\mathbf{x}_i - b) \geq 1 - \xi_i$.

Support vectors & nonlinear classification via SVM

The support vectors are those $\mathbf{x}_i$ where $\xi_i > 0$; these are vectors that are misclassified or close to the boundary. For points that are linearly separated, the support vectors lie exactly on the separating parallel hyperplanes.

Nonlinear classification proceeds by using kernel functions. Dot products $\mathbf{w}'\mathbf{x}$ are replaced by, e.g. polynomial functions $k(\mathbf{w}, \mathbf{x}) = (\mathbf{x}'\mathbf{w})^k$ or Gaussian functions $k(\mathbf{w}, \mathbf{x}) = e^{-\theta||\mathbf{w}-\mathbf{x}||^2}$.

This trick can solve very hard classification problems, such as when the separating hyper-surface is elliptical.

SVM can do as well or better than logistic regression with fewer features. SVM has been very successful in classifying, for example disease using using medical imaging or genetics data.

Example: dental data using SVMs

This is a *very* simple example. You will typically want to `tune` the SVM using a training data set in real applications. The function `svm` can handle factors as predictors too.

```
library(e1071)
f=svm(Sex~d8+d10+d12+d14,data=d2) # default kernel is radial not linear!
summary(f)
plot(f,d2,d12~d14,slice=list(d8=22,d10=23))

f=svm(Sex~d8+d10+d12+d14,data=d2,kernel="linear") # classical linear SVM
summary(f)
plot(f,d2,d12~d14,slice=list(d8=22,d10=23))
```

Classification and regression trees (CART)

The linear (non-kernel) versions of logistic regression and SVM produce a hyperplane that slices $\mathbb{R}^p$ into two portions, each portion classifying one of the two groups. A method that produces regions that are simple rectangles in $\mathbb{R}^p$ are classification trees. The data are $\{(\mathbf{x}_i, z_i)\}_{i=1}^n$.

The tree is grown iteratively. First, variable $v_1 \in \{1, \dots, p\}$ is chosen to split x_{iv_1} at value t_1 yielding two portions $P_0 = \{i : x_{iv_1} \le t_1\}$ and $P_1 = \{i : x_{iv_1} > t_1\}$. Note then $\{1, \dots, n\} = P_0 \cup P_1$. The tree now has two branches and we add a third by picking either P_0 or P_1 and dividing this into two portions similarly, i.e. using x_{iv_2} and cutpoint t_2. Say we pick P_1 to split into $P_1 = P_{10} \cup P_{11}$. Note then that $\{1, \dots, n\} = P_0 \cup \underbrace{P_{10} \cup P_{11}}_{P_1}$.

The tree has three branches; I'll draw it on the board.

Example

At the kth split, there are $k+1$ branches, i.e. we have partitioned $\{1,\ldots,n\}$ into $k+1$ sets. We have also partitioned $\mathbb{R}^p$ into $k+1$ rectangles. The tree is given by a series of sets that have been split. Say we split 5 times, yielding a partition with 6 portions, according to $\mathbf{e}_1 = \emptyset$, $\mathbf{e}_2 = 1$, $\mathbf{e}_3 = 11$, $\mathbf{e}_4 = 0$, $\mathbf{e}_5 = 110$. The final tree has

$$\{1,\ldots,n\} = P_{00} \cup P_{01} \cup P_{10} \cup P_{1100} \cup P_{1101} \cup P_{111}.$$

Let $\mathbf{e} = e_1 \cdots e_k$; the region associated with portion $P_{\mathbf{e}}$ is

$$R_{\mathbf{e}} = R_{e_1\cdots e_j} = \cap_{i=1}^{j} \left[\{\mathbf{x} : x_{v_i} \le t_i, e_i = 0\} \cup \{\mathbf{x} : x_{v_i} > t_i, e_i = 1\} \right].$$

For the example,

$$\mathbb{R}^p = R_{00} \cup R_{01} \cup R_{10} \cup R_{1100} \cup R_{1101} \cup R_{111}.$$

Along with each split is a variable $v_1,\ldots,v_k$ and cutpoint $t_1,\ldots,t_k$. The splits are added sequentially to maximize a Bernoulli likelihood.

How to choose the variable and cutpoint?

For any $P_{\mathbf{e}} = P_{e_1 \cdots e_j} \subset \{1, \ldots, n\}$, let $y_{\mathbf{e}} = y_{e_1 \cdots e_j} = \sum_{i \in P_{e_1 \cdots e_j}} I\{z_i = 1\}$ and $n_{\mathbf{e}} = n_{e_1 \cdots e_j} = \sum_{i \in P_{e_1 \cdots e_k}} 1$. Let $v_1, \ldots, v_k$ be a series of variables chosen to split on, $t_1, \ldots, t_k$ their corresponding cutoffs, and $P_{\mathbf{e}_1}, \ldots, P_{\mathbf{e}_k}$ the associated portions. One can argue (using conditional probability) that the likelihood for any sequence is

$$L = \prod_{j=1}^{k} [\frac{y_{\mathbf{e}_j 0}}{n_{\mathbf{e}_j 0}}]^{y_{\mathbf{e}_j 0}} [1 - \frac{y_{\mathbf{e}_j 0}}{n_{\mathbf{e}_j 0}}]^{n_{\mathbf{e}_j 0} - y_{\mathbf{e}_j 0}} [\frac{y_{\mathbf{e}_j 1}}{n_{\mathbf{e}_j 1}}]^{y_{\mathbf{e}_j 1}} [1 - \frac{y_{\mathbf{e}_j 1}}{n_{\mathbf{e}_j 1}}]^{n_{\mathbf{e}_j 1} - y_{\mathbf{e}_j 1}}.$$

At iteration k, $(\mathbf{e}_k, v_k, t_k)$ is chosen to maximize this likelihood.

The tree is typically stopped when there are only a few observations in each branch. At this point the tree has overgrown (overfits the data) and must be pruned. Marden suggests AIC or BIC.

Dental data, bagging, & random forests

```
f=tree(Sex~d8+d10+d12+d14,data=d2)
plot(f); text(f)
f.aic=prune.tree(f,k=4)
plot(f.aic); text(f.aic)
f.bic=prune.tree(f,k=2*log(dim(d2)[1]))
plot(f.bic); text(f.bic)
```

Bagging simply grows a different tree for each of several bootstrapped (i.e. with replacement) samples. Classifications are made by majority vote, the most frequent classification across the boostrapped trees. A random forest is a bagged classifier, but where only a (random) subset of the elements of $\mathbf{x}$ (typically of size $\sqrt{p}$) are considered at each split.

Bagging can produce much better prediction, but the nice interpretability of one tree goes out the window. Essentially becomes a "black box" learner.

Logistic regression, boosting, SVM, and CART

- All methods can be generalized to more than $g = 2$ groups, but the most natural generalization occurs with logistic regression (ordinal regression or generalized logits for nominal outcomes).
- Logistic regression, boosting, and CART all naturally handle categorical predictors. However, `svm` seems to handle categorical predictors as well.
- Model-based approaches provide *probabilies* of group membership for a given $\mathbf{x}_{n+1}$. SVM, for example, only provides $\hat{z}_{n+1}$.

Other methods

There are a few other nonparametric approaches to classification and regression.

- Random forests (bagging).
- Neural networks.
- Gaussian process models (for classification and regression).
- Others, and tweaks on existing approaches, are churned out daily in the machine learning and statistics literature. Mastering the domain of machine learning is a daunting task.
- Scratches the surface, but gives a taste of the most popular classifiers in use.
- Many classifiers are immediately generalizable to regression too. For example "classification and **regression** trees." How do you think CART works in a regression context?