Chapter 2

Supervised machine learning (I)

Supervised machine learning is an important analytical tool for high-throughput genomic data. Assume that the input data in population (stochastic) version include a G-dimensional random vector $\vec{\mathbf{X}} = (\mathbf{x_1}, \cdots, \mathbf{x_G})$ as covariates (e.g. the gene expression of G genes) and a random variable $\mathbf{Y}$ that takes values on $\{1, 2, \cdots, K\}$ as the class label (e.g. whether the patient is metastatic or not as a K=2 binary case). In a given application, the observed data contain S samples (patients): $\mathbf{D} = ((\mathbf{y_1}, \vec{\mathbf{x}}_1), \cdots, (\mathbf{y_S}, \vec{\mathbf{x}}_S))$ where $(\mathbf{y_s}, \vec{\mathbf{x}}_s) \sim (\mathbf{Y}, \vec{\mathbf{X}})$ for $1 \leq s \leq S$. Taking genomic applicla-tions as an example, Y may represent labels for "disease vs control", "metastatic vs non-metastatic", "short patient survival vs long patient survival", "drug respondants vs non-respondants" or "multiple disease subtypes". X may contain G features of one type or multi-type mixture of clinical variables, gene expression intensities, miRNA expression, protein expression, methylation intensities, genotype mutations and many more. The goal of supervised machine learning is to "learn" from the observed data $\mathbf{D}$ to construct a prediction model $C(\vec{\mathbf{x}}|\mathbf{D}) \in \{1, 2, \cdots, K\}$ when the observed covariates of a future patient $\vec{\mathbf{x}}$ is given. In the population (stochastic) version (i.e. pretending that the entire underlying distribution is known), the overall classification error rate is $Pr(\mathbf{Y} \neq C(\vec{\mathbf{X}}|\mathbf{D}))$. If a validation set is available with known class labels $\mathbf{D}^{(test)} = ((\mathbf{y}_1^{(test)}, \vec{\mathbf{x}}_1^{(test)}), \cdots, (\mathbf{y}_{S'}^{(test)}, \vec{\mathbf{x}}_{S'}^{(test)}))$, the validation error is assessed as $\left[\sum_{s=1}^{S'} \chi\left(\mathbf{y}_s^{(test)} \neq C(\vec{\mathbf{x}}_s^{(test)}|\mathbf{D})\right)\right] / S'$, where $\chi(\cdot)$ is an indicator function.

In Chapter 2.1, we start from an ideal machine learning principle when the entire underlying data distribution is known. Under this situation, the optimal solution called Bayes classification rule should be pursued. In real practice, the underlying distribution is certainly unknown and is impossible to estimate well in the high-dimensional situation as previously mentioned in Chapter 1. We will introduce several popular machine learning methods including logistic regression (Chapter 2.2), linear (quadratic) discriminant analysis (Chapter 2.3), classification and regression tree (CART) (Chapter 2.4), random forest (Chapter 2.6) and support vector machines (Chapter 2.7). Understanding formulations and algorithms behind different methods is only the first step. There are many other fundamental machine learning concepts to consider when performing machine learning in a real data set. They will be discussed in the next chapter.

2.1 Bayes classification rule

From Bayes rule, we have the following formula:

$$f(Y=k|X)=\frac{f(Y=k)f(X|Y=k)}{f(X)}=\frac{f(Y=k)f(X|Y=k)}{\sum_{l=1}^{K}f(Y=l)f(X|Y=l)}$$

Let c_{ij} be the cost of inaccurately assigning an individual of group i to group j. We usually assume $c_{ij}\geq 0$ if $i\neq j$ and $c_{ii}=0$, $1\leq i\leq K$. Assume that the prior $f(Y=k)$, conditional density $f(X|Y=k)$ and the cost function are known, the optimal (or Bayes rule) classifier that minimizes the expected loss is:

$$C(x)=i \text{ if } x\in R_i$$

where $R_i=\{x:\sum_{1\leq h\leq K}f(Y=h)f(x|Y=h)c_{hi}<\sum_{1\leq h\leq K}f(Y=k)f(x|Y=h)c_{hj},\forall 1\leq j\leq K, j\neq i\}$. In other words, we assign an observation x to group i if $\sum_{1\leq h\leq K}f(Y=h)f(X|Y=h)c_{hi}$ is minimized (i.e. $C(x)=\arg\min_i\sum_{1\leq h\leq K}f(Y=h)f(X|Y=h)c_{hi}$).
In the special case that $c_{ij}=1$ $(\forall i\neq j)$ and $c_{ii}=0$ $(1\leq i\leq K)$, the classification rule becomes

$$\begin{aligned}C_{Bayes}(X)&=\arg\min_i\sum_{h\neq i}f(Y=h)f(X|Y=h)=\arg\min_i\sum_{h\neq i}f(Y=h|X)\\&=\arg\min_i 1-f(Y=i|X)=\arg\max_i f(Y=i|X)\\&=\arg\max_i f(Y=i)f(X|Y=i).\end{aligned}$$

One can show that $C_{Bayes}(X)$ is optimal in that $C_{Bayes}(X) = \arg\max_{C(x)} P(Y = C(X))$ (See Exercise 1).

Example:
Suppose a disease diagnostic tool gives values from N(3,1) for diseased patients and N(0, 2) for normal patients (i.e. $f(X|Y = 0) \sim N(0, 2)$ and $f(X|Y = 1) \sim N(3, 1)$. Figure 2.1 shows the density plot of the two conditional distributions. When the cost of making a type-I and type-II errors are equivalent and the prior probabilities are the same, the decision boundary is made at the point $x_0 = 1.597$ where $f(X_0|Y = 1) = f(X_0|Y = 0)$. In this case, the resulting sensitivity is 92%, specificity is 87.1% and overall accuracy is 85.4%.

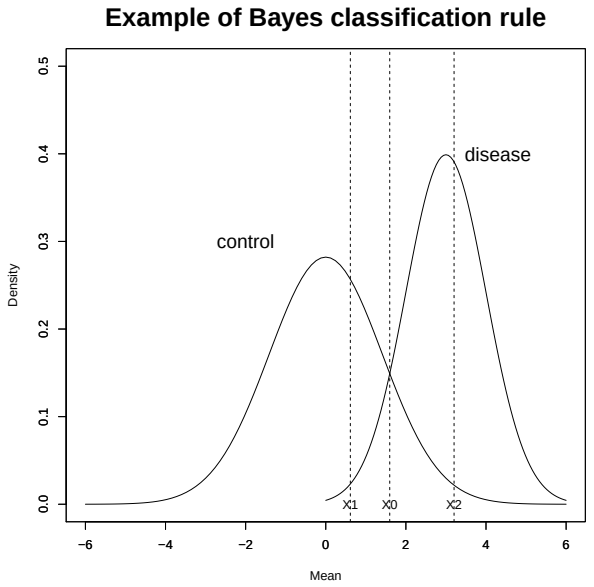

Figure 2.1: Density plot of conditional distributions of control N(0,2) and disease N(3,1) and the three Bayes decision thresholds x_0, x_1 and x_2.

If the prevalence of the disease is 10%, f(Y=0)=0.9 and f(Y=1)=0.1. Consider a cheap diagnostic tool that is meant to be a simple first-line screen test. We may want to allow smaller cost function c_{01} for false positives so that these false positive patients can be followed up by more expensive and accurate diagnosic tools. For example, we can assigin $c_{01} = 0.01$ and $c_{10} = 1$ and the decision boundary will be made at $x_1 = 0.613$ where $f(X_1|Y = 1) \cdot 0.1 \cdot 1 = f(X_1|Y = 0) \cdot 0.9 \cdot 0.01$. Under

this cost function and prior probability, the resulting best specificity is as low as 66.8% and the sensitivity is high at 99.2%. On the other hand, if the diagnostic tool is expensive and accurate, we usually require large cost function for false positives (so that not to scare normal patients). If we assigin $c_{01} = 1$ and $c_{10} = 0.5$, the decision boundary will be made at $x_2 = 3.2$ where $f(X_2|Y = 1) \cdot 0.1 \cdot 0.5 = f(X_2|Y = 0) \cdot 0.9 \cdot 1$. The resulting specificity is high (98.8%) and the sensitivity is 42.1%. Exercise 2 will request the reader to repeat the analysis of this simple example. As we will see in Chapter 2.3, Bayes classification rule under Gaussian assumption motivates a class of linear and quadratic discriminant analysis (LDA and QDA) methods.

2.2 Logistic regression

Logistic regresion is the simplest machine learning method for binary classification. It extends from simple linear regression using a logistic function. Suppose $Pr(\mathbf{Y} = 1) = \pi(\vec{\mathbf{X}})$.

$$g(\vec{\mathbf{x}}_s) = log\frac{\pi(\vec{\mathbf{x}}_s)}{1 - \pi(\vec{\mathbf{x}}_s)} = \beta_0 + \sum_{g=1}^{G} \beta_g \cdot \mathbf{x_{gs}} + \epsilon_{gs}$$

Logistic regression performs well when sample size S is large, number of covariates G is small and the assumptions (linear association, independence and homoscedasticity) are true in the data. In most genomic applications, S is small and G is large. The linear association also may not be true. Nevertheless, one can still perform proper gene filtering (to be discussed in the next chapter) and PCA eigen-decomposition and apply logistic regression for binary prediction. It can serve as a quick first-line machine learning method.

2.3 Linear and quadratic discriminant analysis

Under Gaussian assumptioins, assume $X|Y = k \sim N(\mu_k, \Sigma_k)$. We have

$$f(X|Y = k) = \frac{1}{(2\pi)^{G/2}|\Sigma_k|^{1/2}} \exp\left(-\frac{1}{2}(X - \mu_k)^T \Sigma_k^{-1}(X - \mu_k)\right)$$

The Bayes classifier (under uniform cost function) becomes

$$C_{Bayes}(X) = \arg\max_k f(Y = k|X)$$
$$= \arg\min_k -2\log f(Y = k) + \log|\Sigma_k| + (X - \mu_k)^T\Sigma_k^{-1}(X - \mu_k)$$

Since $f(Y = k|X)$ is continuous in X for a given k, the decision boundary of selecting among any pair of l and m is

$$f(Y = l|X) = f(Y = m|X) \text{ (think about why?)}$$

Or equivalently,
$-2\log f(Y = l) + \log|\Sigma_l| + (X - \mu_l)^T\Sigma_l^{-1}(X - \mu_l)$
$= -2\log f(Y = m) + \log|\Sigma_m| + (X - \mu_m)^T\Sigma_m^{-1}(X - \mu_m)$.
We can easily see that the discriminant boudary above is of quadratic form in X.

2.3.1 LDA

If we further assume that the covariance structures are the same for all classes (i.e. $\Sigma_l = \Sigma$, $1 \le l \le K$), we can easily show that

$$C_{LDA}(X) = \arg\min_k -2\log f(Y = k) + (X - \mu_k)^T\Sigma^{-1}(X - \mu_k)$$
$$= \arg\min_k -2\log f(Y = k) + \mu_k'\Sigma^{-1}\mu_k - 2\mu_k'\Sigma^{-1}X$$

The decision boundary becomes

$$-2\log f(Y = l) - 2\mu_l^T\Sigma^{-1}X + \mu_l^T\Sigma^{-1}\mu_l$$
$$= -2\log f(Y = m) - 2\mu_m^T\Sigma^{-1}X + \mu_m^T\Sigma^{-1}\mu_m$$

$$2(\mu_l - \mu_m)^T\Sigma^{-1}X - (\mu_l - \mu_m)^T\Sigma^{-1}(\mu_l + \mu_m)$$
$$+ 2\left(\log f(Y = l) - \log f(Y = m)\right) = 0$$

The decision boundary is a linear hyperplane.
If further assum uniform prior $\log f(Y = l) = \log f(Y = m)$, the hyperplane simplifies to $2(\mu_l - \mu_m)^T\Sigma^{-1}X - (\mu_l - \mu_m)^T\Sigma^{-1}(\mu_l + \mu_m) = 0$.

Define transformation $X^* = \Sigma^{-1/2}X$, then $\mu_l^* = \Sigma^{-1/2}\mu_l$, $\mu_m^* = \Sigma^{-1/2}\mu_m$. The hyperplane becomes

$$2(\mu_l^* - \mu_m^*)^T X^* - (\mu_l^* - \mu_m^*)^T(\mu_l^* + \mu_m^*) = 0$$

The hyperplane is perpendicular to $\mu_l^* - \mu_m^*$ and it passes through the mid-point of the two class centers $\frac{\mu_l^* + \mu_m^*}{2}$.

Implementation in real data:

(1) estimate $\widehat{\Sigma}$ by combining K groups

(2) $\widehat{\Sigma} = UDU^T$ (eigen-decomposition)
$\widehat{\Sigma}^{-1/2} = UD^{-1/2}U^T$

(3) sphere the data: transform the entire data by $\widehat{\Sigma}^{-1/2}$

$$x^* = \widehat{\Sigma}^{-1/2}x$$

$$\mu_k^* = \widehat{\Sigma}^{-1/2}\mu_k$$

(4) For a given x, classify x^* into the closest centroid μ_k^*, i.e. $C(x) = \arg\min_k |x^* - \mu_k^*|$

?? Add a figure to show the concept of transformation and LDA implementation. (the discrimination is simple by nearest distance in the transformed space)

We note that the total number of parameters for general QDA is $K \cdot G + K \cdot \frac{G(G+1)}{2}$ (the former term is for class means and the latter term is for covariance matrixes). This is usually intractable especially when K is high. LDA somewhat reduces the number of parameters to $K \cdot G + \frac{G(G+1)}{2}$.

2.3.2 DLDA

In microarray application, G is large. Estimation of general Σ may be unstable. Strong assumption may be imposed to LDA. One example is to assume independence between covariates when conditioned on each class. The covariance matrixes become identical and diagonal.

$$\Sigma_k = \Sigma = \begin{pmatrix} \sigma_1^2 & & 0 \\ & \ddots & \\ 0 & & \sigma_G^2 \end{pmatrix} \Rightarrow \Sigma^{-1/2} = \begin{pmatrix} \sigma^{-1} & & 0 \\ & \ddots & \\ 0 & & \sigma_G^{-1} \end{pmatrix}$$

$$\Rightarrow \sum_{g=1}^{G} \frac{(\mu_{lg} - \mu_{mg})}{\sigma_g^2} \left[x_g - \frac{\mu_{lg} + \mu_{mg}}{2} \right] = 0$$

$$C_{DLDA}(X) = \arg\max_k \sum_{g=1}^{G} \left\{ \frac{(x_g - \mu_{kg})^2}{\sigma_g^2} \right\}$$

The number of parameters for DLDA is greatly reduced to $G \cdot K + G$.

2.3.3 DQDA

Assume $\Sigma_k = \begin{pmatrix} \sigma_{1k}^2 & & 0 \\ & \ddots & \\ 0 & & \sigma_{gk}^2 \end{pmatrix}$ and $f(Y = l) = 1/K$ (uniform prior), $\forall 1 \leq l \leq K$.

The classification rule reduces to:

$$C_{DQDA}(X) = \arg\min_k \sum_{g=1}^{G} \left\{ \frac{(x_g - \mu_{kg}^2)^2}{\sigma_{kg}^2} + \log \sigma_{kg}^2 \right\}$$

The decision boundary becomes:

$$\log|\Sigma_l| + (X - \mu_l)^T \Sigma_l^{-1} (X - \mu_l) = \log|\Sigma_m| + (X - \mu_m)^T \Sigma_m^{-1} (X - \mu_m)$$

$\Rightarrow \sum_{g=1}^{G} \left(\frac{1}{\sigma_{gl}^2} - \frac{1}{\sigma_{gm}^2} \right) X_g^2 - 2 \sum_{g=1}^{G} \left(\frac{\mu_{gl}}{\sigma_{gl}^2} - \frac{\mu_{gm}}{\sigma_{gm}^2} \right) x_g + \sum_{g=1}^{G} \left(\frac{\mu_{gl}^2}{\sigma_{gl}^2} - \frac{\mu_{gm}^2}{\sigma_{gm}^2} \right) + 2 \sum_{g=1}^{G} \left(\log \sigma_{gl}^2 - \log \sigma_{gm}^2 \right) = 0$, a simple quadratic surface.

The number of parameters for DQDA is $G \cdot K + G \cdot K = 2 \cdot G \cdot K$

Example
Figure 2.2 shows a simulated 2D example for LDA, DLDA and QDA. In the first LDA example, the 100 observatioins are simulated from each of the two classes: $N\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 & 0.5 \\ 0.5 & 1 \end{pmatrix}\right)$ and $N\left(\begin{pmatrix} 0 \\ 4 \end{pmatrix}, \begin{pmatrix} 1 & 0.5 \\ 0.5 & 1 \end{pmatrix}\right)$. By applying "lda" function in R, the LDA classification rule is determined and the green and blue regions represent the areas that are predicted to class 1 or class 2. In the second DLDA example, there is zero correlation between the two dimensions in the conditional distributions in each class: $N\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}\right)$ and $N\left(\begin{pmatrix} 0 \\ 4 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}\right)$. It is easily seen that the classification boundaries generated in both examples are linear. In the third example, diagonal and distinct covariance matrixes in the

two classes are used: $N\left(\begin{pmatrix}0\\0\end{pmatrix}, \begin{pmatrix}1 & 0\\0 & 2\end{pmatrix}\right)$ and $N\left(\begin{pmatrix}0\\4\end{pmatrix}, \begin{pmatrix}2 & 0\\0 & 1\end{pmatrix}\right)$. In this situation, the decision boundary becomes quadratic as suggested by QDA theorem. Exercise 3 goes through steps to repeat the simulation and generate the prediction plot.

?? change the blue/green colors to gray scale to show posterior probabilities.

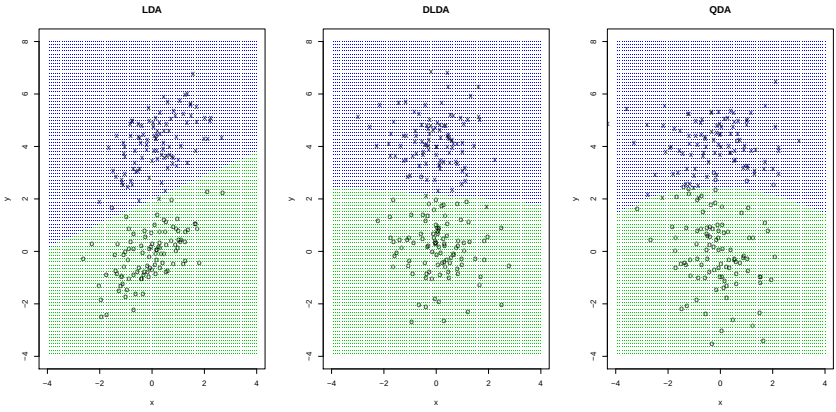

Figure 2.2: Simulation of three examples for LDA, DLDA and QDA and their regions of prediction.

2.4 Classification and regression tree

As before, assume data from distribution (Y, X) where $Y \in \{1, 2, \ldots, K\}$ is the class label and $X = (x_1, \ldots, x_G) \in \mathbb{R}^G$ is the $G-$ dimensional covariates. The classification and regression tree proposed by Brieman et al (1984) perform the following tree construction. A feature x chosen from one of the G dimensions and a cutoff $c \in \mathbb{R}$ are used to split the root node into t_L and t_R such that a specific goodness of split (GOS) criterion is achieved. The splitting procedure is applied recursively until some stopping rule is satisfied. Denote by T the tree constructed and $\widetilde{T}$ the set of terminal nodes of T. Below we start from determining the classification rule when the tree structure T is given (Chapter 2.4.1). Chapter 2.4.2 introduces goodness of split (GOS) criteria for deciding the branch splits and construct the tree. Finally, Chapter 2.4.3 describes pruning procedures to balance cost and complexity of the tree.

2.4.1 Bayes classification rule when tree structure is given

For a given terminal node $t \in \widetilde{T}$, the Bayes classification rule is

$$\begin{aligned} C_{Bayes}(X) &= \arg \max_{1 \le i \le K} P(Y = i | X \in t) \\ &= \arg \max_{1 \le i \le K} \frac{P(Y = i) P(X \in t | Y = i)}{P(X \in t)} \\ &= \arg \max_{1 \le i \le K} P(Y = i) P(X \in t | Y = i) \end{aligned}$$

The misclassification rate of $C_{Bayes}(X)$ in the terminal node t is

$$\begin{aligned} P(C_{Bayes}(X) \neq Y | X \in t) &= 1 - \max_{1 \le i \le K} P(Y = i | X \in t) \\ &= 1 - \frac{\max_{1 \le i \le K} P(Y = i) P(X \in t | Y = i)}{P(X \in t)} \end{aligned}$$

$$\begin{aligned} \therefore P(C_{Bayes}(X) \neq Y) &= \sum_{t \in \widetilde{T}} P(X \in t) P(C_{Bayes}(X) \neq Y | X \in t) \\ &= \sum_{t \in \widetilde{T}} \left[P(X \in t) - \max_{1 \le i \le K} P(Y = i) P(X \in t | Y = i) \right] \end{aligned}$$

When $K = 2$,

$$P(C_{Bayes}(X) \neq Y | X \in t) = \frac{\min_{1 \le i \le 2} P(Y = i) P(X \in t | Y = i)}{P(X \in t)}$$

$$P(C_{Bayes}(X) \neq Y) = \sum_{t \in \widetilde{T}} \min_{1 \le i \le 2} P(Y = i) P(X \in t | Y = i)$$

Theorem: (Brieman, Friedman, Olshen, Stone, 1984)
For any given tree T, $C_{Bayes}(X)$ gives the lowest classification error rate among all possible group assignment in the terminal nodes $\widetilde{T}$.

Example: Consider a given training data set $(Y_s, X_s), 1 \le s \le S$, a new test sample X_{new} and prior $P(Y = i) = p_i$. If X_{new} falls to terminal node t^* (i.e. $X_{new} \in t^*$), the new test sample should be predicted to

$$C_{Bayes}(X_{new}) = \arg \max_{1 \le i \le K} p_i \cdot \frac{\sum_{1 \le s \le S} \chi(Y_s = i, X_s \in t^*)}{\sum_{1 \le s \le S} \chi(Y_s = i)},$$

where $\chi(\cdot)$ is an indicator function that takes value one if the statement is true and zero if false. When uniform prior is used, the prediction should be based on

$$C_{Bayes}(X_{new}) = \arg \max_{1 \le i \le K} \frac{\sum_{1 \le s \le S} \chi(Y_s = i, X_s \in t^*)}{\sum_{1 \le s \le S} \chi(Y_s = i)},$$

When the prior $P(Y = i)$ is estimated from the training data, the prediction reduces to looking for the class with majority vote in node t^*:

$$C_{Bayes}(X_{new}) = \arg \max_{1 \le i \le K} \sum_{1 \le s \le S} \chi(Y_s = i, X_s \in t^*)$$

This classification decision is quite frequently used but the readers should be aware of the underlying assumption. The assumption is often not true; for example, the disease prevalence is 10% but 50 control and 50 diseases samples are analyzed as the training data. See the next chapter (on machine learning performance evaluation) for more detail.

2.4.2 Goodness of split (GOS) criteria

For a given terminal node $t \in \widetilde{T}$, an impurity measure can be defined as $M(t) = \phi(P(Y = 1|X \in t), P(Y = 2|X \in t), \ldots, P(Y = K|X \in t))$. Normally, we require $\phi(P_1, \ldots, P_K)$, $P_i \ge 0$ and $\sum_{i=1}^{K} P_i = 1$, such that ϕ is maximized when $P_i = 1/K$, $1 \le i \le K$ (the most impure case) and minimized when $P_i = 1$ and $P_j = 0$ $(j \ne i)$ (the most pure case). We also require ϕ symmetry in $(P_1, \ldots, P_K)$. Two impurity functions are commonly used:
Gini index: $M_{Gini}(t) = 1 - \sum_{i=1}^{K} (P(Y = i|X \in t))^2$
Entropy: $M_{entropy}(t) = -\sum_{i=1}^{K} P(Y = i|X \in t) \log P(Y = i|X \in t)$

Given an impurity function $M(t)$, the GOS criterion is to find the split t_L and t_R of node t such that the impurity measure is maximally decreased:

$$\arg \max_{t_R, t_L} M(t) - [P(X \in t_L|X \in t)M(t_L) + P(X \in t_R|X \in t)M(t_R)]$$
$$= \arg \min_{t_R, t_L} \frac{P(X \in t_L)M(t_L) + P(X \in t_R)M(t_R)}{P(X \in t)}$$

Example For a given node t, suppose seven observations are inside the node: covariate $X = \{1.1, 1.5, 1.7, 1.8, 2.0, 2.5, 3\}$ and class label

$Y = \{0, 1, 0, 0, 1, 1, 1\}$. Denote by $G(t) = P(X \in t_L | X \in t)M(t_L) + P(X \in t_R | X \in t)M(t_R)$. It can be shown that the threshold at $X = 1.9$ gives the best split under either Gini or Entropy impurity function ($t_L = ((0, 1.1), (1, 1.5), (0, 1.7), (0, 1.8))$ and $t_R = ((1, 2.0), (1, 2.5), (1, 3))$). Under Gini index, the minimized score is $G_{Gini}(1.9) = (4/7) \cdot (1 - 0.25^2 - 0.75^2) + (3/7) \cdot (1 - 1^2 - 0^2) = 0.214$. For entropy measure, the minimized score becomes $G_{entropy} = (4/7) \cdot (-0.25 \cdot \log(0.25) - 0.75 \cdot \log(0.75)) + (3/7) \cdot 0 = 0.321$. The optimization is a linear search (the same as the number of observations in a node) and is very fast.

Theorem: Monotone invariant property
Consider $Z = G(X)$ a monotone transformation. Based on either impurity function and GOS criterion, the tree T_X generated by X and the tree T_Z generated by Z have the same structure. That is, the selected features are the same and the cutpoints $C_{T_Z} = G(C_{T_X})$ are equivalent.

Misclassification rate estimates
Recall the following equation for misclassification rate:

$$R(T) = \sum_{t \in \widetilde{T}} P(X \in t) - \max_{1 \le i \le K} (P(Y = i) P(X \in t | y = i))$$

We may estimate the error rate by empirical distribution

$$\widehat{R}(T) = \sum_{t \in \widetilde{T}} \widehat{P}(X \in t) - \max_{1 \le i \le K} [P(Y = i) \widehat{P}(X \in t | y = i)]$$

where $\widehat{P}(X \in t) = \frac{\sum_{j=1}^{N} 1\{X_j \in t\}}{N}$, $\widehat{P}(X \in t | Y = i) = \frac{\sum_{j=1}^{N} 1\{Y_j = i \ \& \ X_i \in t\}}{\sum_{j=1}^{N} 1\{Y_j = i\}}$
Note that if the same data is used to generate T and to estimate the error rate, the error rate is overfitting and is often too optimistic. V-fold cross validation is often used to avoid the bias (see next chapter for more details).

2.4.3 Minimal cost-complexity pruning

Suppose T_{max} is the largest tree developed until all observations in a terminal node are pure or when the number of observation is smaller than a pre-defined threshold. The tree T_{max} is often overfitted to the training data and cannot generalize to independent test data (see "generalizability" concept in the next chapter). Brieman et al (1984) proposed a backward node recombination strategy called minimal cost-complexity

pruning. The cost-complexity of T is

$$C_\alpha(T) = \widehat{R}(T) + \alpha|T| \tag{2.1}$$

where $\widehat{R}(T)$ is the error rate of tree T estimated from empirical distribution and $|T|$ is the number of terminal nodes of T. α is a positive complexity parameter that balance between (overfitted) error and model complexity. In practice, α is estimated from cross-validation.

How do we perform the optimization? Since the number of nodes is finite, the optimization can be performed within limited time for a given α. The following weakest-link cutting method provides an efficient and concise algorithm to find optimal subtrees for any α. The algorithm sequentially generate a nested sequence of optimal subtrees: $T_{max} = T_1 \subset T_2 \subset \ldots \subset T_m$, where T_m is the top node containing all observations. The nested sequence of optimal subtrees is generated as the following. At iteration i, the tree is created by removing a subtree from T_{i-1}. The choice of the subtree s removed from T_{i-1} is based on the cost-complexity measure: $s^* = \arg\min_{s \in T_{i-1}} \frac{R(prune(T_{i-1},s)) - R(T_{i-1})}{|T_{i-1}| - |prune(T_{i-1},s)|} = \frac{R(prune(T_{i-1},s)) - R(T_{i-1})}{|s|-1}$ and $T_i = prune(T_{i-1}, s^*)$, where $prune(T_{i-1}, s)$ is the tree after pruning s from T_{i-1} and $|s|$ is the number of leaves for subtree s. Note that the optimization searches all possible subtrees in T_{i-1} so it is possible that large subtrees are pruned in the iterations and normally $m << |T_{max}|$. The following theorem states that the resulting nested sequence of subtrees provide solutions for all possible α. Note that this algorithm simplifies the computation in the estimation of α through cross-validation. The proof of the theorem is left for exercise (Exercise 6)

Theorem Denote by $\alpha_1 = 0$ and $\alpha_i = \frac{R(T_i) - R(T_{i-1})}{|T_{i-1}| - |T_i|}$. It can be shown that $\alpha_{i-1} < \alpha_i$, $\forall i > 1$. Furthermore, for a given α satisfying $\alpha_{i-1} \leq \alpha < \alpha_i$, the smallest optimal subtree from equation 2.1 is T_{i-1}.

Example: ??Add one more example to illustrate the tree pruning and the weakest-link cutting algorithm.

Example: Figure 2.3 (A)-(C) shows two simulated examples for demonstrating the CART method. In the first example, 200 points are simulated in [-6,6]×[-6,6]. When $x > 0$ and $y > 0$, the observations belong to class II and otherwise class I. This example exactly matches the searching space behind CART and Figure 2.3(A) shows that CART can perform almost perfectly. The resulting CART classifier (Figure 2.3 (B)) has 99.8% prediction accuracy on an additional 10,000 simulated observations. In Figure 2.3 (C)-(F), we simulate an example that fits the underlysing assumption of LDA. 100 observations of class I are simu-

lated from $N\left(\begin{pmatrix}1\\-1\end{pmatrix}, \begin{pmatrix}2 & 1\\1 & 2\end{pmatrix}\right)$ and another 100 observations of class II are from $N\left(\begin{pmatrix}-1\\1\end{pmatrix}, \begin{pmatrix}2 & 1\\1 & 2\end{pmatrix}\right)$. When LDA is applied to the data and the resulting classifier is used to predict an additional 10,000 simulated observations, 91.8% accuracy is achieved (Figure 2.3 (C); regions of different predictions are colored by green and blue). Since LDA is the Bayes classifier in this case, the accuracy is the best we can do among all possible classification methods. When we use CART, the prediction accuracy drops to 83.5% and the classification rule is not desirable (Figure 2.3 (D)).

2.5 Bagging and boosting

In this chapter and next chapter, we will introduce several resampling-based ensemble methods to improve CART. CART is well-known to be instable in high-dimensional complex systems. These ensemble methods help provide more stable classifiers and improve the accuracy. The concepts also applies to other instable machine learning methods.

Bootstrapping Bootstrapping was first proposed by Brad Efron in 1979, motivated from the "jackknife" concept. The term is ironic in nature since the act of lifting oneself up into the air by one's bootstrap (and by oneself) is physically impossible. It was the first intuition when facing resampling concepts from a classical statistician's point of view 30 years ago. How can one gain more information by resampling from the observed data? You cannot create more information!! However, resampling ideas turn out to be clever and have become popular in modern statistical science especially when computing is (to a large degree) no longer a problem.

The fundamental concept of bootstrapping is that the observed (empirical) data are realizations of the underlying unknown populatioin distribution. The best guess of the underlying cumulative distribution function (CDF) $F(t)$ is the empirical CDF $\hat{F}_S(t) = \frac{\sum_{s=1}^{S} \chi(x_s \le t)}{t}$, where $x_1, \cdots, x_S$ are iid observations from $F(t)$. In the bootstrap technique, the bootstrapped data are sampled with uniform probability with replacement in each iteration. The procedure is repeated for B times (e.g. B=1000). Suppose in iteration b ($1 \le b \le B$), the bootstrapped data are denoted as $x_1^{(b)}, \cdots, x_S^{(b)}$. Theoretically, $x_1^{(b)}, \cdots, x_S^{(b)}$ are iid generated from $\hat{F}_S(t)$. The empirical distribution function $\hat{F}_S(t)$ has nice properties that it asymptotically converges to $F(t)$ in several ways:

1. $\hat{F}_S(t) \to F(t)$ almost surely for every value t. That is, $\hat{F}_S(t)$ is consistent.

2. The central limit theorem shows that the pointwise distribution is asymptotically normal distributed with $\sqrt{S}$ convergence rate:

$$\sqrt{S}(\hat{F}_S(t) - F(t)) \to^d N(0, F(t)(1 - F(t)))$$

3. The convergence can be stronger in a uniform sense. That is,

$$sup_t|\hat{F}_S(t) - F(t)| \to 0.$$

With the nice convergence properties of $\hat{F}_S(t)$, it is hoped that (under some condition) inferences based on the bootstrapped samples reflects corresponding inferences based on the unknown underlying true distribution (at least asymptotically). Such a technique is useful when the theoretical distribution of the interested statistic is too complicated or the sample size is insufficient for direct inference (e.g. provide the variance or confidence interval estimate of a complicated estimator). It can be shown that when S is large, the number of unique samples in a bootstrapped data is expected to be (1-1/e)$\cdot S$=63.2%$\cdot S$ (See Exercise 7).

Bagging Bagging is a resampling type of machine learning ensemble algorithm. It is often used to improve CART but it theoretically can apply to any machine learning method. In the b^{th} $(1 \leq b \leq B)$ iteration, bootstrapped data $x_1^{(b)}, \cdots, x_S^{(b)}$ are sampled. Then CART is applied on the bootstrapped data and the prediction model is generated. After B iteration, B prediction models are available. For any new sample, the final prediction is based on majority vote of the B prediction models. Bagging is often used to improve unstable machine learning methods (e.g CART). The model averaging approach provides a more stable classifier and the classification rule also has higher complexity. If bagging is applied to already stable methods (e.g. K-nearest neighbor), it was shown that is might slightly decrease the prediction accuracy (Breiman, 1996).

Boosting Boosting is another resampling-based machine learning ensemble method. The method also resample data with S sample size. Different from iid sampling from the fixed empirical distribution in Bagging, boosting's sampling favors for those objects misclassified in previous classifications. The most famouse boosting algorithm is AdaBoost (Adaptive Boosting).

??More details of AdaBoost will be provided in the future.

2.6 Random forest

Random forest combines two major resampling techniques in the ensemble method: bagging and random selection of features. The algorithm goes with the following:

1. In the b^{th} iteration, generate a bootstrapped data $D^{(b)}$ from the original data D. Choose a constant $m << G$.

2. For each node of the tree, randomly select m variables to decide the split on that node. The tree is fully grown and not pruned.

3. The iteration is repeated for B times. A total of B decision trees $T^{(1)}, T^{(2)}, \cdots, T^{(B)}$ are recorded. To predict any new sample x_{new}, the decision is made by majority vote from the B tree predictions $\arg\max_i \frac{\sum_{1\leq b\leq B} \chi(T^{(b)}(x_{new})=i)}{B}$.

One merit of random forest is that it does not require prior feature filtering and it automatically generates an importance score of each feature that contributes to the classifier. Computation of random forest is very fast and it improves CART and bagging in almost all cases.

Example: Back to the previous example in CART, Bagging and random forest are applied to the simulated data. Bagging provides a more complex and more desirable classifier than CART due to its resampling and majority vote algorithm (Figure 2.3 (E)). The generalization accuracy improves from 83.5% to 87.9%. Random forest generated a further complex classifier that better match the underlying truth and the accuracy improves to 88.5%, which is already quite close to the Bayes classifier accuracy 91.8% (Figure 2.3 (F)).

2.7 Support vector machines

?? To be added later

2.8 Artificial neural network

?? To be added later

Reference:

Christopher J. C. Burges. (1998) A Tutorial on Support Vector Machines

for Pattern Recognition. Data Mining and Knowledge Discovery. 2: 121 - 167.

Efron, B. (1979). "Bootstrap methods: Another look at the jackknife". The Annals of Statistics 7 (1): 126

Breiman, Leo (1996). "Bagging predictors". Machine Learning 24 (2): 123140.

Exercise:

1. Prove that the Bayes classification rule gives the smallest error rate when the cost function and prior are both uniformly non-informative.

2. (a) Draw density plot of conditional distributions for control N(0, 2) and disease N(3,1). (b) Calculate x_0, x_1 and x_2 under the prior probability and cost function specified in the example. (hint: use "uniroot" function in R) (c) Draw Figure 2.1 (d) Simulate 1000 normal and 1000 disease patients. Apply the LDA method to verify your answers of x_0, x_1 and x_2.

3. Follow the following steps to repeat the LDA, DLDA and QDA examples. (a) Simulate 100 observations from each class and draw scatter plots of all three examples. (b) Perform LDA and DLDA and QDA on each example. (c) Generate grid points on the space and perform prediction using uniform prior (hint: use "predict.lda" and "predict.qda" function in R). Draw the prediction results by different colors. (d) From the analysis output, derive the exact classification boundary function.

4. Use a real data to perform LDA:

   (1) Apply the "lda" function from "MASS" package to the data.

   (2) Write your own R code to implement LDA and compare the prediction result and accuracy to (1).

   (3) Apply "diagDA" in "sfsmisc" package or "dlda" in "supclust" package.

   (4) Write your own R code to implement DLDA and compare the prediction result and accuracy to (2).

5. Use a real data to perform CART:

   (1) Write your own R code to implement GOS criteria and perform tree splitting. Use Gini index to construct the maximal tree until all terminal nodes are pure or less than five objects.

(2) Use the cost-complexity pruning technique and cross validation to determine the complexity parameter α and prune the tree.

(3) Perform cross-validation to estimate an unbiased error rate for CART. (Note: Two loops of cross-validation are needed; One outer loop for error rate estimate and one inner loop to estimate α.)

6. Prove the theorem of weakest-link cutting algorithm for tree pruning.

7. Prove that when $S \to \infty$, the fraction of unique samples in a bootstrapped data is expected to be $(1 - 1/e)$ where e is the base of natural logarithm.

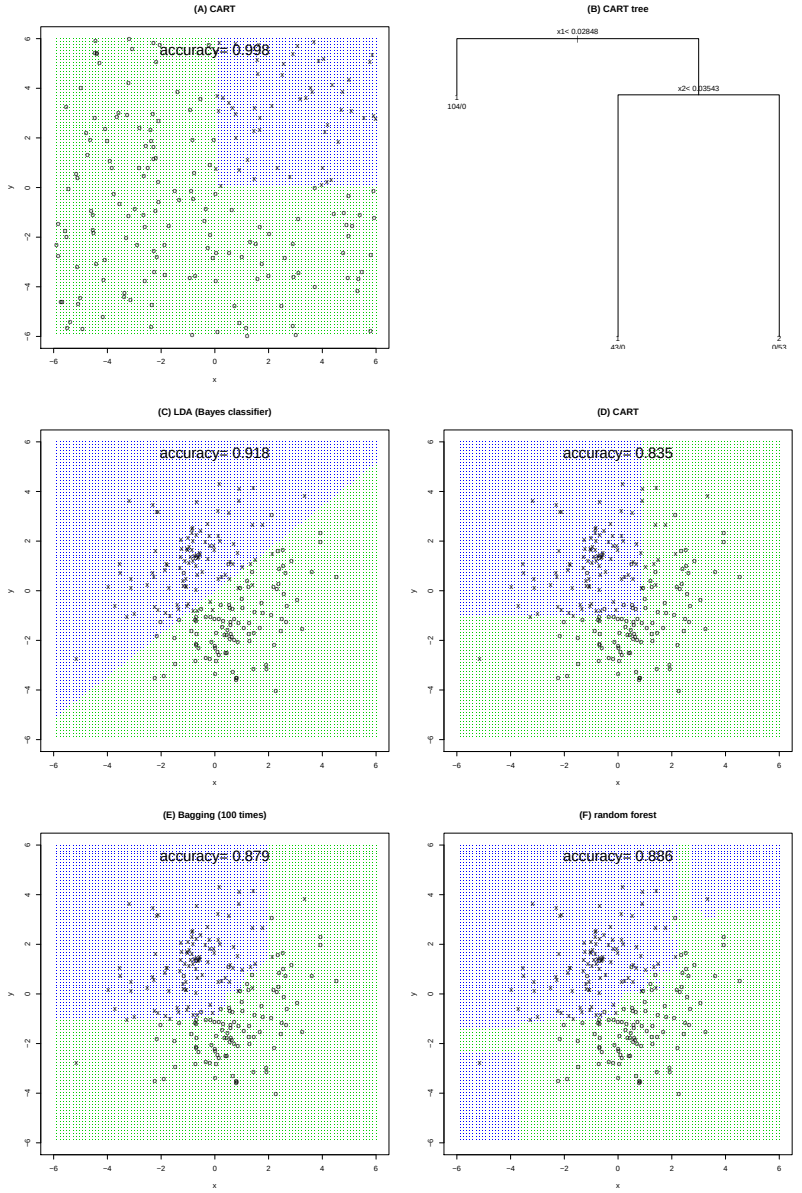

Figure 2.3: Simulation examples to demonstrate CART, Bagging and random forest methods.