STA314 Summer 2023

Week 3: Classification

Ziang Zhang

Department of Statistics, University of Toronto

Overview

What is classification?

In the previous lectures, we focused on predicting a *quantitative* or *continuous* outcome variable y (examples: housing price, height, weight ...).

This type of prediction method is called regression, which includes: OLS regression, KNN regression...

What is classification?

However, in many applications, we need to predict a *qualitative* or *categorical* outcome variable y (examples: eye color, sex, education degree ...).

Unlike quantitative variables that can take a range of values, qualitative variables can only take a few levels (categories), which may or may not be *ordinal*.

Classification refers to the process of predicting or assigning a qualitative variable y, to a certain level $\hat{y}$ (category).
The classification algorithm (method) is called a *classifier*.

In other words, a classifier is a way to determine the level of $\hat{y}$, based on the feature $\mathbf{x}$.

Different types of classifiers

Assume the outcome variable Y takes K levels, encoded as $1, 2, ..., K$. The feature vector is denoted as $\mathbf{X}$.

- A *probabilistic* classifier will compute $P(Y = k|\mathbf{X} = \mathbf{x})$ for each level k, and then predict based on that probability.
- The *classification threshold* is a pre-defined value, used to determine the class of an observation. If $P(Y = k|\mathbf{X} = \mathbf{x})$ is greater than the classification threshold, we classify the observation as class k.
- A *Bayes* classifier is a probabilistic classifier that predicts by $\hat{Y} = \text{argmax}_k \, P(Y = k|\mathbf{X} = \mathbf{x})$.
- A *non-probabilistic* classifier will directly predict Y without computing the probability.

Different types of classifiers

There are two types of *probabilistic* classifiers, depending on how $P(Y = k|\mathbf{X} = \mathbf{x})$ is computed.

- A discriminative approach computes $P(Y = k|\mathbf{X} = \mathbf{x})$ for each $k = 1, ..., K$ directly.
- The distribution of the feature $\mathbf{X}$ is neither considered nor utlized in the classification.
- A generative approach models the conditional distribution of $\mathbf{X}$ in each outcome category k, as well as the marginal distribution of Y (i.e. $P(Y = k)$). Then flips them into $P(Y = k|\mathbf{X} = \mathbf{x})$ using Bayes theorem.
- The (conditional) distribution of the feature becomes an important ingredient to be utilized.

Bayes Rule

$$P(Y = k|\mathbf{X} = \mathbf{x}) = \frac{f(\mathbf{X} = \mathbf{x}, Y = k)}{f(\mathbf{X} = \mathbf{x})}$$
$$= \frac{f_k(\mathbf{x}) \cdot \pi_k}{\sum_{i=1}^{K} f_i(\mathbf{x}) \cdot \pi_i}$$

where:

- $f_k(\mathbf{x}) = f(\mathbf{X} = \mathbf{x}|Y = k)$ is the conditional density of $\mathbf{X}$ given $Y = k$.
- $f(\mathbf{X} = \mathbf{x}) = \sum_{i=1}^{K} f(\mathbf{X} = \mathbf{x}, Y = k)$ is the marginal density of $\mathbf{X}$ at $\mathbf{x}$.
- $\pi_k = P(Y = k)$ is the marginal probability (prevalence) of Y at level k.
- In a generative method, all of $\{\pi_i\}_{i=1}^{K}$ and $\{f_i(\mathbf{x})\}_{i=1}^{K}$ need to be estimated.

Example: `Default` Data

- We are interested in predicting whether an individual will default on his credit card payment (two levels of Y: YES or NO).
- We collect the information from $n = 10,000$ individuals.
- There are two features `balance` (X_1) and `income` (X_2).

Example: Default Data

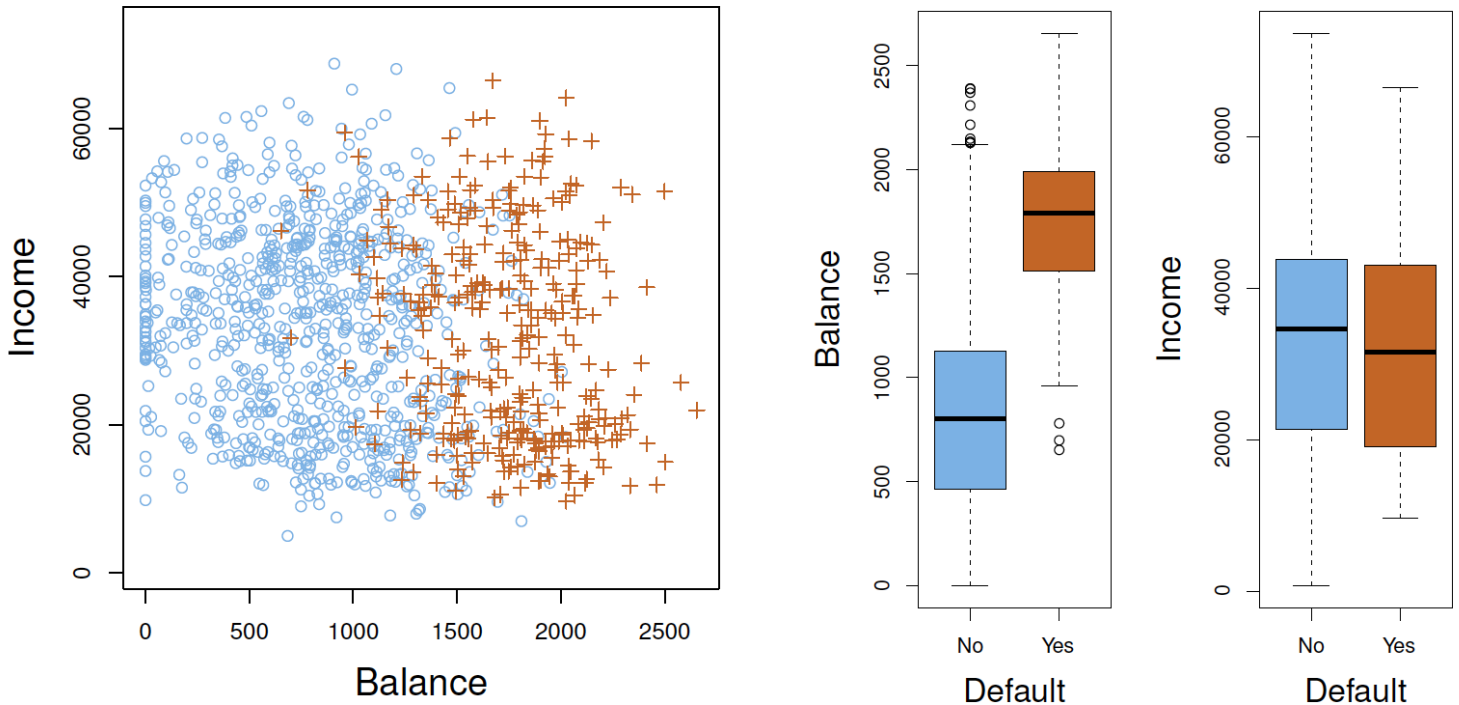


FIGURE 4.1. *The* `Default` *data set.* Left: *The annual incomes and monthly credit card balances of a number of individuals. The individuals who defaulted on their credit card payments are shown in orange, and those who did not are shown in blue.* Center: *Boxplots of* `balance` *as a function of* `default` *status.* Right: *Boxplots of* `income` *as a function of* `default` *status.*

Why not OLS?

We will first focus on the discriminative classifiers.
For simplicity, assume Y takes $K = 2$ levels, coded as 0 or 1.

- In the `default` data, we can code YES $= 1$ and NO $= 0$.
- The target is then $P(Y = 1|X_1, X_2) = \mathbb{E}(Y|X_1, X_2)$.
- Recall OLS also aims to estimate $\mathbb{E}(Y|X_1, X_2)$ as

$$\hat{\beta}_0 + \hat{\beta}_1 X_1 + \hat{\beta}_2 X_2.$$

- Why not just use OLS to estimate $P(Y = 1|X_1, X_2)$?

Problems with OLS

1. As probability, $P(Y = 1|X_1, X_2)$ should be within $[0, 1]$.
2. However, the OLS estimate $\hat{\beta}_0 + \hat{\beta}_1 X_1 + \hat{\beta}_2 X_2$, can be anywhere in the real line...
3. This is not unexpected, as the OLS regression was designed for predicting a continuous variable with support in the entire real line $\mathbb{R}$.
4. But we can come up with a transformation that maps from $[0, 1]$ to $\mathbb{R}$, and then apply regression on the *transformed* probability...

Logistic regression: Motivation

Let $P(A)$ denotes the probability of an event A, the **odd** of event A is defined as:

$$\text{odd}(A) = \frac{P(A)}{1 - P(A)}.$$

The **logit** function from $[0, 1] \to \mathbb{R}$, is defined as:

$$\text{logit}(p) = \log\left(\frac{p}{1-p}\right).$$

The log odd of A can be computed as

$$\text{logit}[P(A)] = \log\left(\frac{P(A)}{1 - p(A)}\right).$$

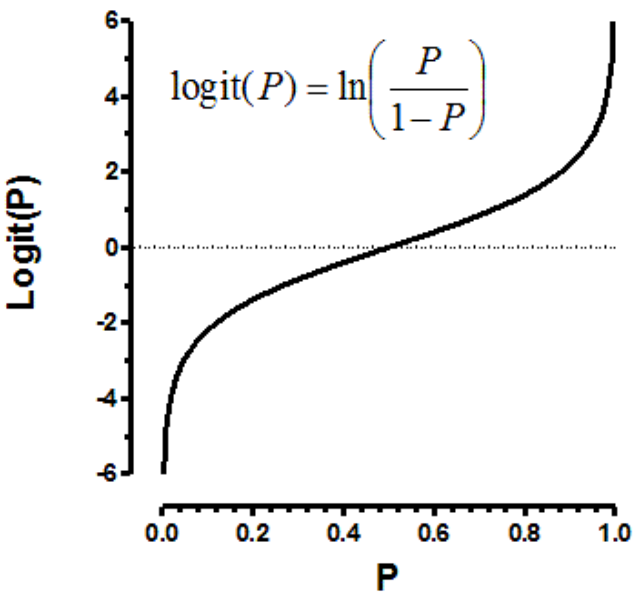
- As $P(A) \to 1$, $\text{odd}(A) \to \infty$ and $\text{logit}[P(A)] \to \infty$.
- As $P(A) \to 0$, $\text{odd}(A) \to 0$ and $\text{logit}[P(A)] \to -\infty$.

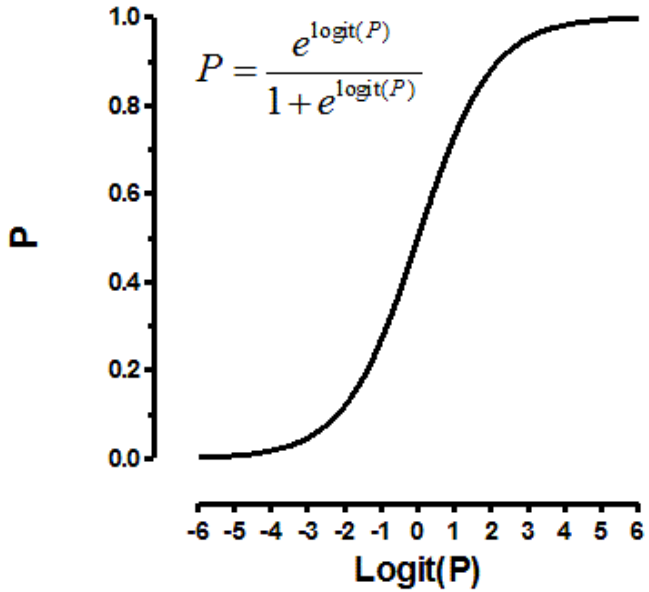
Logistic function

Given the logit of $P(A)$, the original probability can be computed from the inverse of the logit function:

$$P(A) = \frac{\exp(\text{logit}[P(A)])}{1 + \exp(\text{logit}[P(A)])}.$$

This function is called Logistic function or the inverse-logit.





Logistic regression

Given $Y = 0$ or 1, and feature $\mathbf{x} \in \mathbb{R}^p$, the logistic regression is defined as following:

$$\text{logit}[P(Y = 1|\mathbf{x})] = \mathbf{x}^T\boldsymbol{\beta}.$$

- This is similar to linear regression, since both models have $\mathbf{x}^T\boldsymbol{\beta}$ on the RHS.
- This is different from linear regression, since the LHS in linear regression is $\mathbb{E}(Y|\mathbf{x})$ instead of $\text{logit}[\mathbb{E}(Y|\mathbf{x})]$.
- Also, an additional parameter σ is required to specify $\text{Var}(Y|\mathbf{x})$ in linear regression.
- Whereas in logistic regression, $\text{Var}(Y|\mathbf{x})$ is already specified by $\boldsymbol{\beta}$ (why?)

Logistic regression: Prediction

Given the fitted logistic regression model:

$$\text{logit}[\hat{P}(Y = 1|\mathbf{x})] = \mathbf{x}\hat{\boldsymbol{\beta}}.$$

The log-odd at a new feature $\mathbf{x}_{new} \in \mathbb{R}^p$ is predicted as

$$\text{logit}[\hat{P}(Y_{new} = 1|\mathbf{x}_{new})] = \mathbf{x}_{new}^T\hat{\boldsymbol{\beta}}.$$

The predicted probability is therefore

$$\hat{P}(Y_{new} = 1|\mathbf{x}_{new}) = \frac{\exp(\mathbf{x}_{new}^T\hat{\boldsymbol{\beta}})}{1 + \exp(\mathbf{x}_{new}^T\hat{\boldsymbol{\beta}})}.$$

The classifier $\hat{Y}_{new}$ is **Bayes** if:
$\hat{Y}_{new} = 1$ when $\hat{P}(Y_{new} = 1|\mathbf{x}_{new}) \geq 0.5$, and zero otherwise.

Logistic regression: Computation

Like linear regression, the logistic regression model can be fitted with maximum likelihood.

- Recall the density (pmf) of Bernoulli random variable Y with success probability p: $P(Y = y) = p^y(1-p)^{1-y}$.
- Given independent observations $\{y_i, \mathbf{x}_i\}_{i=1}^n$, the probability of each observation is

$$P(Y_i = y_i|\mathbf{x}_i) = \left[\frac{\exp(\mathbf{x}_i^T\boldsymbol{\beta})}{1+\exp(\mathbf{x}_i^T\boldsymbol{\beta})}\right]^{y_i}\left[\frac{1}{1+\exp(\mathbf{x}_i^T\boldsymbol{\beta})}\right]^{1-y_i}.$$

- The log-likelihood of the samples is: *(derive as an exercise)*

$$l(\boldsymbol{\beta}) = \sum_{i=1}^{N}\left[y_i\mathbf{x}_i^T\boldsymbol{\beta} - \log\left(1+\exp(\mathbf{x}_i^T\boldsymbol{\beta})\right)\right]$$

Logistic regression: Computation

Given the log-likelihood:

$$l(\boldsymbol{\beta}) = \sum_{i=1}^{N} \left[y_i \mathbf{x}_i^T \boldsymbol{\beta} - \log\left(1 + \exp(\mathbf{x}_i^T \boldsymbol{\beta})\right) \right]$$

- It can be shown that $l(\boldsymbol{\beta})$ is concave.
- This implies the MLE $\hat{\boldsymbol{\beta}}$ can be obtained by solving

$$S(\boldsymbol{\beta}) = \nabla l(\boldsymbol{\beta}) = \mathbf{0}.$$

- Unlike in linear regression, the MLE $\hat{\boldsymbol{\beta}}$ in logistic regression does not have an analytical form...

Gradient Descent

To solve $S(\boldsymbol{\beta}) = \nabla l(\boldsymbol{\beta}) = \mathbf{0}$ numerically, we can use the method of **gradient descent** (GD).

Given an initial guess $\boldsymbol{\beta}_0$, the GD method iterates as the following:

$$\boldsymbol{\beta}_{i+1} = \boldsymbol{\beta}_i + \alpha S(\boldsymbol{\beta}_i).$$

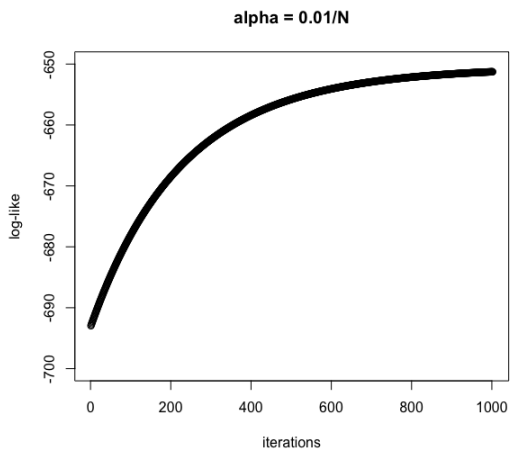
- The tuning parameter $\alpha \in \mathbb{R}^+$ is called the *learning rate*.
- When this algorithm converges: $\boldsymbol{\beta}_{i+1} \approx \boldsymbol{\beta}_i$, which implies $S(\boldsymbol{\beta}_i) \approx \mathbf{0}$.
- Recall the gradient $S(\boldsymbol{\beta})$ is the direction of the steepest ascent of l.
- Here we are maximizing l, but typical GD aims to minimize a function (i.e. $-l$).

Gradient Descent: Example

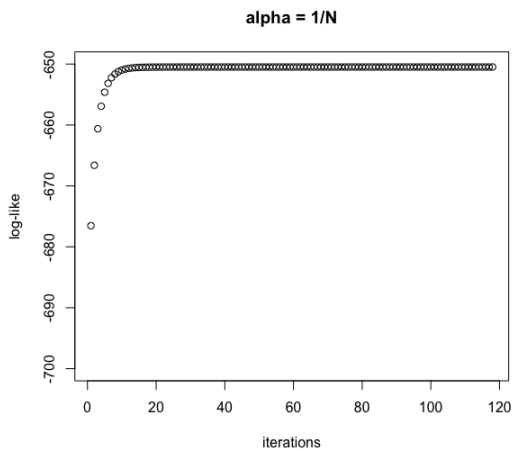
```r
## Log-likelihood
l <- function(beta, train_data_X, train_data_Y){
  sum(-log(1 + exp(train_data_X %*% beta)) + (train_data_Y * (train_data_X %*% beta)))
}

## Score
s <- function(beta, train_data_X, train_data_Y){
  # plogis(q = ...) is the logistic function
  y_hat <- plogis(train_data_X %*% beta)
  S <- t(train_data_X) %*% (train_data_Y - y_hat)
  as.vector(S)
}

GD <- function(beta0, alpha, train_data_X, train_data_Y, threshold = 1e-8, maxit = 1000){
  ## Initialize
  beta_old <- beta0
  score <- s(beta = beta0, train_data_X = train_data_X, train_data_Y = train_data_Y)
  beta_new <- beta_old + alpha * score
  result <- list(l = l(beta = beta_new, train_data_X = train_data_X, train_data_Y = train_data_Y), beta = NULL)
  iter <- 0
  ## Check the convergence
  while(sqrt(sum(score^2)) >= threshold & iter <= maxit){
    iter <- iter + 1
    beta_old <- beta_new
    ## Compute the new score
    score <- s(beta = beta_old, train_data_X = train_data_X, train_data_Y = train_data_Y)
    ## Update the estimate
    beta_new <- beta_old + alpha * score
    ## Report the log-likelihood
    result$l <- c(result$l, l(beta = beta_new, train_data_X = train_data_X, train_data_Y = train_data_Y))
  }
  result$beta <- beta_new
  result
}
```

Gradient Descent: Example

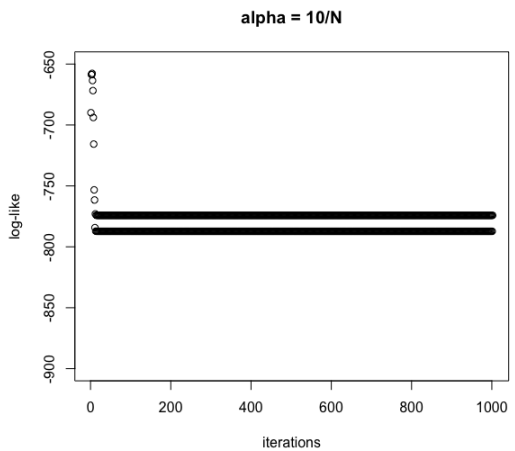
```
> ### Simulate N data:
> set.seed(123)
> N <- 1000
> ## Simulate some features
> train_data_X <- cbind(rnorm(n = N), rnorm(n = N))
> ## Simulate some outcomes
> train_data_Y <- rbinom(n = N, size = 1, prob = plogis(train_data_X %*% c(-0.5, 0.3)))
> ## Apply GD
> result1 <- GD(beta0 = c(0,0), train_data_X = train_data_X, train_data_Y = train_data_Y, alpha = 1/N)
> result2 <- GD(beta0 = c(0,0), train_data_X = train_data_X, train_data_Y = train_data_Y, alpha = 0.01/N)
> result3 <- GD(beta0 = c(0,0), train_data_X = train_data_X, train_data_Y = train_data_Y, alpha = 10/N)
>
>
> ## Plot GD:
> plot(result2$l, xlab = "iterations", ylab = "log-like", main = "alpha = 0.01/N", ylim = c(-700, -650))
> plot(result1$l, xlab = "iterations", ylab = "log-like", main = "alpha = 1/N", ylim = c(-700, -650))
> plot(result3$l, xlab = "iterations", ylab = "log-like", main = "alpha = 10/N", ylim = c(-900, -650))
```



(a) small α



(b) medium α



(c) large α

Figure: Gradient Descent for different learning rates

First order Method

Simple GD only computes the first order information, hence is a very simple method. However:

- It uses the same α for all the parameters and iterations.
- This becomes more of a problem in high-dimensional problem.
- One way to generalize the simple GD is to make α *adaptive* in each iteration.

Second order Method

- Second order method will also compute the Hessian matrix, and use that to define the learning rate at each iteration.
- It also allows different parameters to have different learning rate, which is crucial in high-dimension.
- A famous second order method is the Newton-Raphson: Given an initial guess $\boldsymbol{\beta}_0$, it iterates as the following:

$$\boldsymbol{\beta}_{i+1} = \boldsymbol{\beta}_i - \boldsymbol{H}^{-1}(\boldsymbol{\beta}_i) S(\boldsymbol{\beta}_i),$$

where $\boldsymbol{H}$ is the hessian matrix of the log-likelihood function.

Logistic Regression: Example

In R, the logistic regression can be fitted simply using the function `glm`.
It uses a more sophisticated type of gradient descent called the *Fisher scoring*, which uses the second order information.

```
> data <- ISLR2::Default
> ## Set "Yes" as 1, "No" as 0; Note that 0 means reference level.
> data$default <- relevel(data$default, ref = "No")
> mod <- glm(formula = default~., data = data, family = binomial(link = "logit"))
> mod$coefficients
  (Intercept)    studentYes       balance        income
-1.086905e+01 -6.467758e-01  5.736505e-03  3.033450e-06
```

Default example

Now suppose we want to predict whether a student with $1000 balance and $45000 income will default on his credit card or not.

```
> new_data <- data.frame(student = "Yes", balance = 1000, income = 45000)
> ## Predicted log odd of default
> predict(mod, newdata = new_data, type = "link")
       1
-5.64281
> ## Predicted probability of default
> predict(mod, newdata = new_data, type = "response")
           1
0.003530389
> ## Predicted outcome (Bayes)
> as.numeric(predict(mod, newdata = new_data, type = "response") >= 0.5)
[1] 0
```

Logistic regression for multiple levels

- So far we have assumed Y to take only $K = 2$ levels.
- In reality, it is important to predict Y that could take $K > 2$ levels.
- For example, the education level could be high school, university, or graduate school...
- The disease status could be Type I diabetes, Type II diabetes, or no diabetes.
- $Y \sim \text{Multinomial}(p_1, p_2, ..., p_{K-1})$, instead of $\text{Bernoulli}(p)$.

Question: Why Y could take K levels, but there are only $K - 1$ probabilities?

Logistic regression for multiple levels

Now assume Y can take $K > 2$ levels.

- We code these levels as $0, 1, .., K-1$, where 0 is the *baseline* (reference) level.
- Multinomial logistic regression specifies the following log-odd ratio (for $k = 1, 2, ..., K-1$):

$$\log \frac{P(Y=k|\mathbf{x})}{P(Y=0|\mathbf{x})} = \beta_{k0} + \sum_{j=1}^{p} \beta_{kj} x_j = \mathbf{x}^T \boldsymbol{\beta}_k.$$

- This implies for $k = 1, 2, ..., K-1$: (*derive as an exercise*)

$$P(Y=k|\mathbf{x}) = \frac{\exp(\mathbf{x}^T \boldsymbol{\beta}_k)}{1 + \sum_{j=1}^{K-1} \exp(\mathbf{x}^T \boldsymbol{\beta}_j)},$$

$$P(Y=0|\mathbf{x}) = \frac{1}{1 + \sum_{j=1}^{K-1} \exp(\mathbf{x}^T \boldsymbol{\beta}_j)}$$

Softmax Coding

The Multinomial logistic regression can also be coded without setting a baseline level, which is called **softmax coding**:

$$P(Y = k|\mathbf{x}) = \frac{\exp(\mathbf{x}^T\boldsymbol{\beta}_k)}{\sum_{j=0}^{K-1}\exp(\mathbf{x}^T\boldsymbol{\beta}_j)}, \tag{1}$$

for $k = 0, 1, 2, ..., K-1$. It is straightforward to show:

$$\begin{aligned}\log\frac{P(Y = k|\mathbf{x})}{P(Y = m|\mathbf{x})} &= (\beta_{k0} - \beta_{m0}) + \sum_{j=1}^{p}(\beta_{kj} - \beta_{mj})x_j \\ &= \mathbf{x}^T(\boldsymbol{\beta}_k - \boldsymbol{\beta}_m).\end{aligned} \tag{2}$$

- In terms of prediction, the two types of coding are equivalent.
- In terms of parameter estimation, the softmax coding is *overparametrized*, hence no unique solution (why?)

Why use generative Methods for Classification?

Although (multinomial) Logistic regression provides a convenient way to construct probabilistic classifiers, sometimes generative classifiers will be more advantageous:

- When the two classes are substantially separated, the fitted logistic regression model is surprisingly unstable.
- When the two classes are perfectly separated (for example: $Y_i = 0$ for $X_i > 0$ and $Y_i = 1$ for $X_i < 0$ for all i), the MLE is **undefined** for logistic regression.
- When the distribution of $\mathbf{x}$ is correctly specified, generative methods tend to outperform discriminative methods for *small sample*.

Linear discriminative analysis

Recall the Bayes rule:

$$P_k(\mathbf{x}) = P(Y = k|\mathbf{x}) = \frac{f_k(\mathbf{x}) \cdot \pi_k}{\sum_{i=1}^{K} f_i(\mathbf{x}) \cdot \pi_i}$$

- Linear discriminative analysis (LDA) assumes

$$\boldsymbol{x}|Y = k \sim N_p(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}).$$

- Within different categories, the feature $\boldsymbol{x}$ has different mean vectors but the same covariance matrix.
- Therefore

$$f_k(\boldsymbol{x}) = \frac{1}{\sqrt{(2\pi)^p |\boldsymbol{\Sigma}|}} \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{x} - \boldsymbol{\mu}_k)\right)$$

LDA continued

Now, the probability $P_k(\mathbf{x})$ can be computed as:

$$P_k(\mathbf{x}) = \frac{\pi_k \cdot \frac{1}{\sqrt{(2\pi)^p |\mathbf{\Sigma}|}} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_k)^T \mathbf{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_k)\right)}{\sum_{j=1}^{K} \pi_j \cdot \frac{1}{\sqrt{(2\pi)^p |\mathbf{\Sigma}|}} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_j)^T \mathbf{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_j)\right)}.$$

Take logarithm and do some algebra, it can be shown (*exercise*):

$$\underset{k}{\operatorname{argmax}}\, P_k(\mathbf{x}) = \underset{k}{\operatorname{argmax}}\, \delta_k(\mathbf{x}),$$

where

$$\delta_k(\mathbf{x}) = \underbrace{\mathbf{x}^T \Sigma^{-1} \boldsymbol{\mu}_k}_{\text{linear in } \mathbf{x}} - \underbrace{\frac{1}{2}\mu_k^T \Sigma^{-1} \boldsymbol{\mu}_k + \log \pi_k}_{\text{constant}}.$$

The function δ_k is called a discriminant function.

Decision boundary

The Bayes decision boundary between two categories i, j is

$$\{\mathbf{x} \in \mathbb{R}^p : P(Y = i|\mathbf{x}) = P(Y = j|\mathbf{x})\}.$$

For a probabilistic classifier, the (estimated Bayes) *decision boundary* between two categories i, j is defined as

$$\{\mathbf{x} \in \mathbb{R}^p : \hat{P}(Y = i|\mathbf{x}) = \hat{P}(Y = j|\mathbf{x})\}.$$

- Note that, $P(Y = i|\mathbf{x}) = P(Y = j|\mathbf{x}) \Rightarrow \delta_i(\mathbf{x}) = \delta_j(\mathbf{x})$.
- A decision boundary is linear if it is defined by a *linear equation*;
  and quadratic if it is defined by a *quadratic equation*.

LDA in practice

In practice, all the parameters $\{\boldsymbol{\mu}_k, \boldsymbol{\Sigma}, \pi_k\}_{k=1}^K$ are unknown, hence will be the function $\delta_k(\mathbf{x})$.

- The classification should be done using the estimated discriminant function:

$$\hat{\delta}_k(\mathbf{x}) = \mathbf{x}^T \hat{\boldsymbol{\Sigma}}^{-1} \hat{\boldsymbol{\mu}}_k - \frac{1}{2} \hat{\boldsymbol{\mu}}_k^T \hat{\boldsymbol{\Sigma}}^{-1} \hat{\boldsymbol{\mu}}_k + \log \hat{\pi}_k$$

- All the parameters are replaced by their MLEs, let N_k denotes the number of observations with $Y = k$:
  - $\hat{\boldsymbol{\mu}}_k = \frac{1}{N_k} \sum_{i:y_i=k} \boldsymbol{x}_i$
  - $\hat{\pi}_k = \frac{N_k}{N}$
  - $\hat{\boldsymbol{\Sigma}}_k = \frac{1}{N_k - 1} \sum_{i:y_i=k} (\boldsymbol{x}_i - \hat{\boldsymbol{\mu}}_k)(\boldsymbol{x}_i - \hat{\boldsymbol{\mu}}_k)^T$
  - $\hat{\boldsymbol{\Sigma}} = \frac{1}{K} \sum_{k=1}^K \hat{\boldsymbol{\Sigma}}_k$

Illustration when $p = 1$

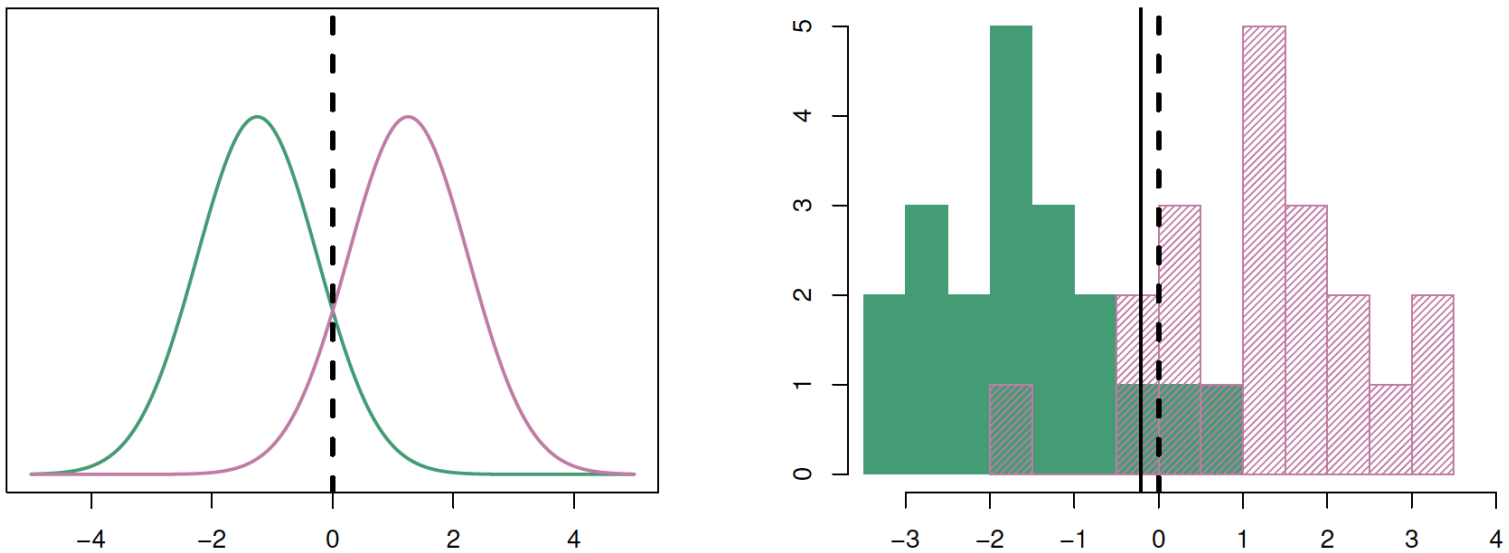


FIGURE 4.4. Left: *Two one-dimensional normal density functions are shown. The dashed vertical line represents the Bayes decision boundary.* Right: *20 observations were drawn from each of the two classes, and are shown as histograms. The Bayes decision boundary is again shown as a dashed vertical line. The solid vertical line represents the LDA decision boundary estimated from the training data.*

Illustration when $p = 2$

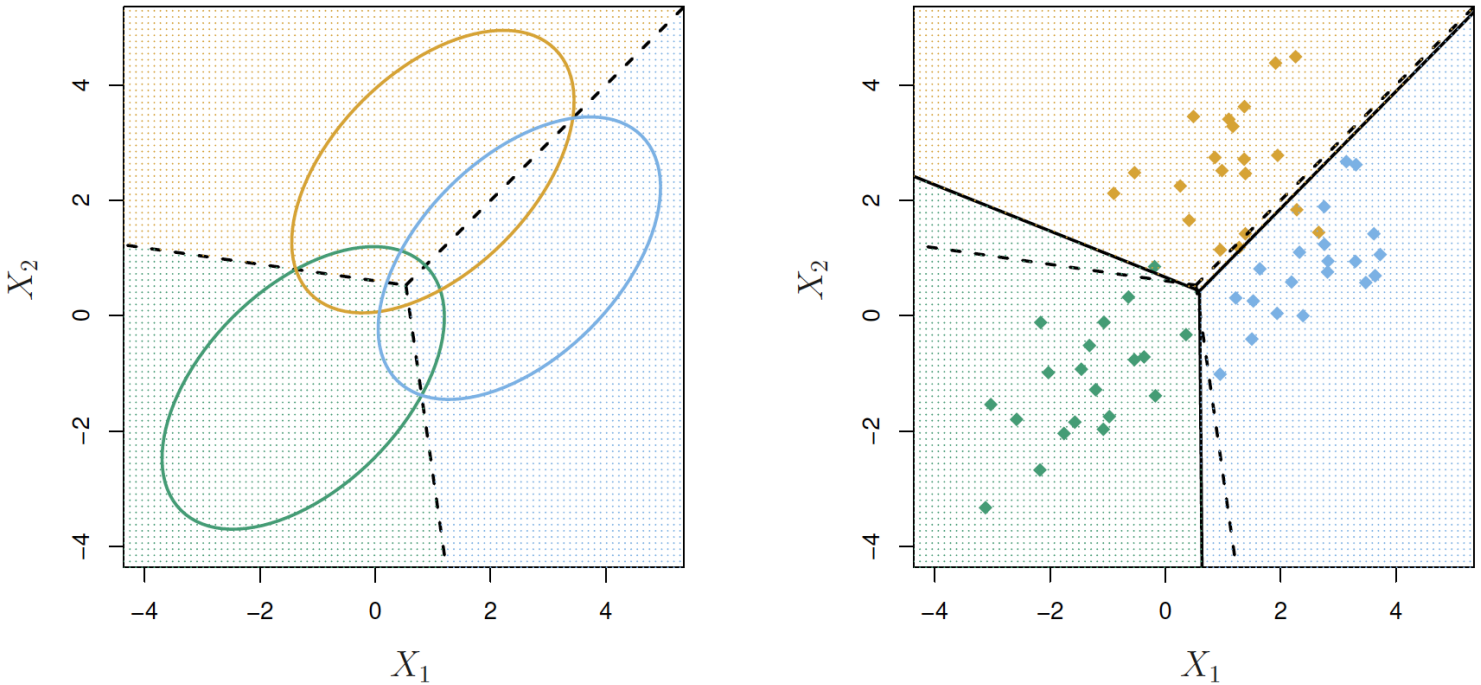


FIGURE 4.6. *An example with three classes. The observations from each class are drawn from a multivariate Gaussian distribution with $p = 2$, with a class-specific mean vector and a common covariance matrix.* Left: *Ellipses that contain* 95 % *of the probability for each of the three classes are shown. The dashed lines are the Bayes decision boundaries.* Right: 20 *observations were generated from each class, and the corresponding LDA decision boundaries are indicated using solid black lines. The Bayes decision boundaries are once again shown as dashed lines.*

Quadratic discriminative analysis

Continuing from the Bayes rule, let's look at a more generalized model:

$$P_k(\mathbf{x}) = P(Y = k|\mathbf{x}) = \frac{f_k(\mathbf{x}) \cdot \pi_k}{\sum_{i=1}^{K} f_i(\mathbf{x}) \cdot \pi_i}$$

- Quadratic discriminative analysis (QDA) also assumes

$$\boldsymbol{x}|Y = k \sim N_p(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k).$$

- Unlike LDA, QDA allows the feature $\boldsymbol{x}$ to have a different covariance matrix, $\boldsymbol{\Sigma}_k$, for each class.
- Specifically

$$f_k(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^p |\boldsymbol{\Sigma}_k|}} \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1} (\boldsymbol{x} - \boldsymbol{\mu}_k)\right)$$

QDA continued

For QDA, the probability $P_k(\mathbf{x})$ can be computed as:

$$P_k(\mathbf{x}) = \frac{\pi_k \cdot \frac{1}{\sqrt{(2\pi)^p |\mathbf{\Sigma}_k|}} \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_k)^T \mathbf{\Sigma}_k^{-1} (\boldsymbol{x} - \boldsymbol{\mu}_k)\right)}{\sum_{j=1}^K \pi_j \cdot \frac{1}{\sqrt{(2\pi)^p |\mathbf{\Sigma}_j|}} \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_j)^T \mathbf{\Sigma}_j^{-1} (\boldsymbol{x} - \boldsymbol{\mu}_j)\right)}.$$

Again, we can just maximize the discriminant function δ_k:

$$\delta_k(\mathbf{x}) = -\frac{1}{2}\log|\mathbf{\Sigma}_k| - \underbrace{\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^T \mathbf{\Sigma}_k^{-1} (\mathbf{x} - \boldsymbol{\mu}_k)}_{\text{quadratic in } \mathbf{x}} + \log \pi_k.$$

Since $\operatorname{argmax}_k P_k(\mathbf{x}) = \operatorname{argmax}_k \delta_k(\mathbf{x})$.

The function δ_k is quadratic instead of linear, hence the name Quadratic Discriminant Analysis.

QDA in practice

Similarly to LDA, the QDA in practice uses the estimated discriminant function $\hat{\delta}_k$.

- Again, all the parameters are replaced by their MLEs:
  - $\hat{\boldsymbol{\mu}}_k = \frac{1}{N_k} \sum_{i:y_i=k} \boldsymbol{x}_i$
  - $\hat{\pi}_k = \frac{N_k}{N}$
  - $\hat{\boldsymbol{\Sigma}}_k = \frac{1}{N_k - 1} \sum_{i:y_i=k} (\boldsymbol{x}_i - \hat{\boldsymbol{\mu}}_k)(\boldsymbol{x}_i - \hat{\boldsymbol{\mu}}_k)^T$
- Unlike in LDA, the samples covariances $\hat{\boldsymbol{\Sigma}}_k$ will *not* be aggregated together.

QDA vs LDA: Pros and Cons

- Pros of QDA:
  - **More flexible:** QDA allows for a different covariance matrix for each class and can model complex decision boundaries.
  - As a result, QDA method will have lower bias than LDA.
- Cons of QDA:
  - **Requires more data:** Estimating individual covariance matrices means more parameters ($Kp(p+1)/2$) to estimate, which requires more data to avoid overfitting.
  - As a result, QDA method tends to have higher variance than LDA.
  - Also inverting high-dimensional covariance matrices can be computationally expensive and unstable.
- If the common variance assumption holds or n is small, LDA is less likely to overfit than QDA.
- If n is large and the common variance assumption is clearly violated, QDA may be a better choice.

Comparing LDA with QDA

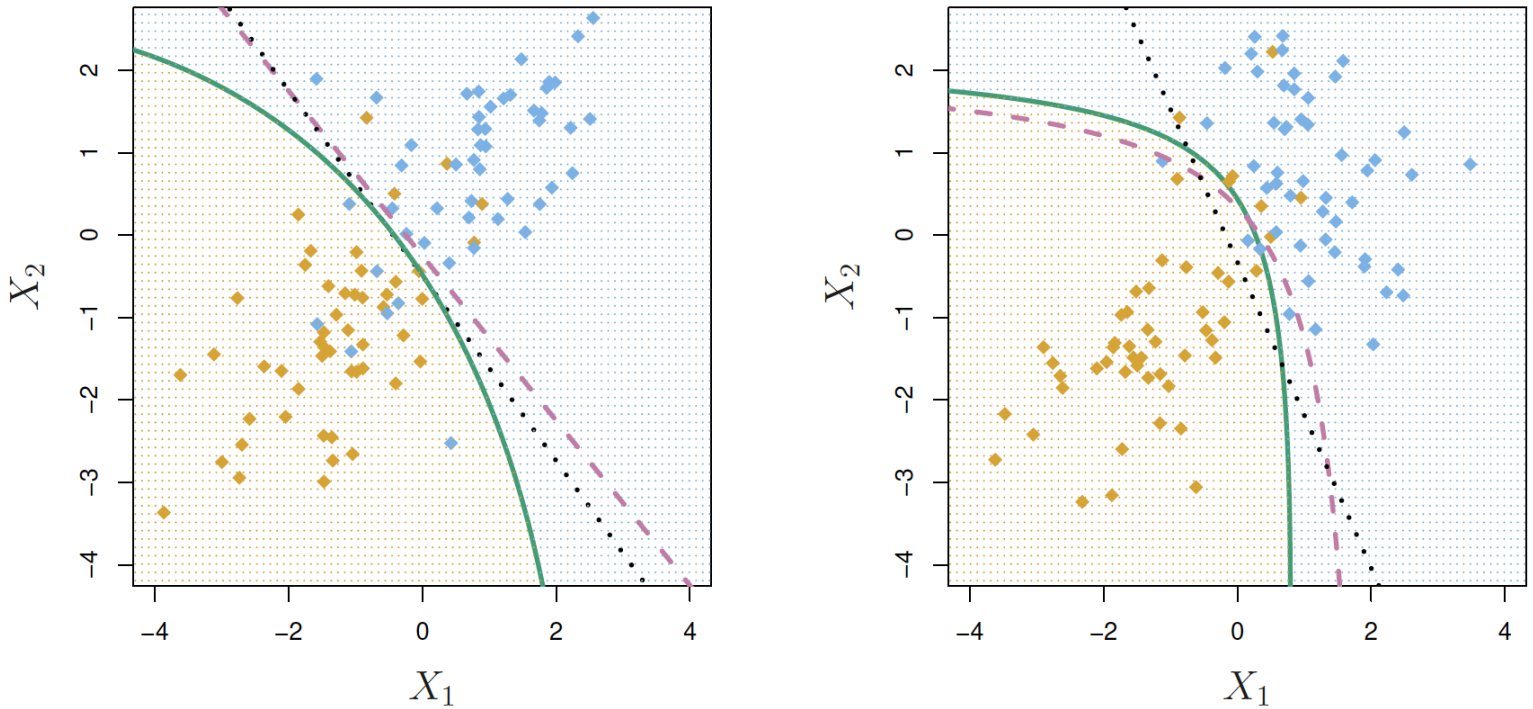


FIGURE 4.9. Left: *The Bayes (purple dashed), LDA (black dotted), and QDA (green solid) decision boundaries for a two-class problem with* $\mathbf{\Sigma}_1 = \mathbf{\Sigma}_2$. *The shading indicates the QDA decision rule. Since the Bayes decision boundary is linear, it is more accurately approximated by LDA than by QDA.* Right: *Details are as given in the left-hand panel, except that* $\mathbf{\Sigma}_1 \neq \mathbf{\Sigma}_2$. *Since the Bayes decision boundary is non-linear, it is more accurately approximated by QDA than by LDA.*

Naive Bayes method

Going back to the Bayes rule:

$$P_k(\mathbf{x}) = P(Y = k|\mathbf{x}) = \frac{f_k(\mathbf{x}) \cdot \pi_k}{\sum_{i=1}^{K} f_i(\mathbf{x}) \cdot \pi_i}$$

- The essence of generative classifiers is to estimate $\{P_k(\mathbf{x})\}_{k=1}^K$, by estimating $\{\pi_k\}_{k=1}^K$ and $\{f_k(\mathbf{x})\}_{k=1}^K$.
- $\{\hat{\pi}_k\}_{k=1}^K$ can be simply constructed by sample proportions.
- $\{\hat{f}_k(\mathbf{x})\}_{k=1}^K$ is much harder to obtain.
- LDA and QDA simplify the task by assuming $\{f_k(\mathbf{x})\}_{k=1}^K$ being multivariate normal with equal or unequal variance.
- Naive Bayes simplifies the task by a *single assumption*:

$$f_k(\mathbf{x}) = \prod_{i=1}^{p} f_{ki}(x_i).$$

Naive Bayes continued

The Naive Bayes assumption:

$$f_k(\mathbf{x}) = \prod_{i=1}^{p} f_{ki}(x_i).$$

Why is this assumption powerful?

- Directly estimating $f_k(\mathbf{x})$ not only involves estimating all the p *marginal distributions*.
- The entire *joint distribution* also contains the information of association between different features.
- For example, if the joint distribution $f_k(\mathbf{x})$ is normal, it is specified by p means and $p(p+1)/2$ covariances.
- By assuming the feature is independent of each other, $f_k(\mathbf{x})$ is then parametrized by p mean values and p elements of the covariance.

Naive Bayes continued

$$f_k(\mathbf{x}) = \prod_{i=1}^{p} f_{ki}(x_i).$$

Different methods can be used to estimate the marginal $f_{ki}(x_i)$.

- If x_i is quantitative, can assume $x_i|Y = k \sim N(\mu_{ki}, \sigma_{ki}^2)$ (parametric).
- Alternatively, estimate $f_{ki}(x_i)$ with its histogram or kernel density estimator (nonparametric).
- If x_i is qualitative, one can estimate $f_{ki}(x_i)$ by the sample proportion.

Naive Bayes: Example

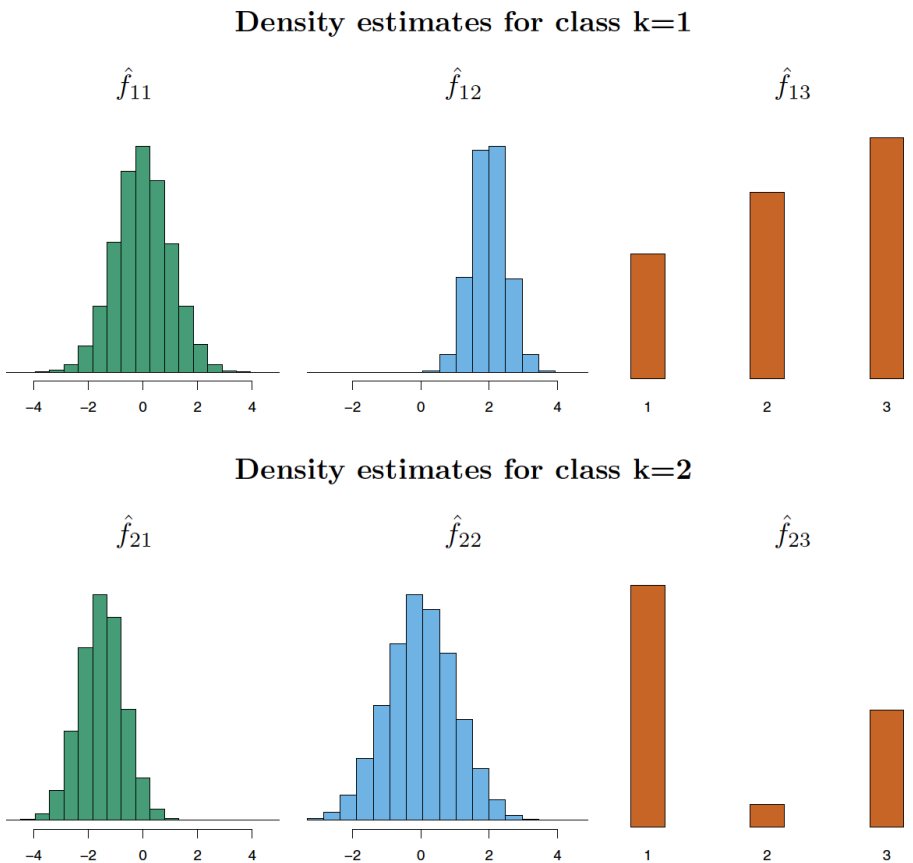


FIGURE 4.10. *In the toy example in Section 4.4.4, we generate data with $p = 3$ predictors and $K = 2$ classes. The first two predictors are quantitative, and the third predictor is qualitative with three levels. In each class, the estimated density for each of the three predictors is displayed. If the prior probabilities for the two classes are equal, then the observation $x^* = (0.4, 1.5, 1)^T$ has a 94.4% posterior probability of belonging to the first class.*

Summary of the three methods

Linear Discriminant Analysis (LDA):

- $f_k(\mathbf{x})$ normal with the *same* covariance matrix across k.
- Has Linear discriminant function and decision boundary.
- Less complex, less likely to overfit, especially with small n.

Quadratic Discriminant Analysis (QDA):

- $f_k(\mathbf{x})$ normal with the *different* covariance matrix across k.
- Has quadratic discriminant function and decision boundary.
- More flexible, better for non-linearly separable classes at the cost of more data and computation.

Naive Bayes:

- Assume $f_k(\mathbf{x})$ factorizes to marginals (independence).
- Highly scalable, and handle high-dimensional datasets well.
- Although the Naive Bayes assumption likely introduces bias, it reduces the variance and simplifies the computation.
- In practice, this classifier is often robust to the Naive Bayes assumption.

Assessment of classifiers

The performance of a classifier can be summarized with the **confusion matrix**:

		True class		
		− or Null	+ or Non-null	Total
Predicted class	− or Null	True Neg. (TN)	False Neg. (FN)	N*
	+ or Non-null	False Pos. (FP)	True Pos. (TP)	P*
	Total	N	P	

Similar to MSE, the confusion matrix is more informative when it is computed using an independent test set.

Assessment of classifiers

Two useful measures can be derived from the confusion matrix:

- Sensitivity (also called True Positive Rate or Power): $\frac{\text{TP}}{\text{TP+FN}} = \frac{\text{TP}}{\text{P}}$. It measures the proportion of actual positives that are correctly identified.
- Specificity (also called True Negative Rate): $\frac{\text{TN}}{\text{TN+FP}} = \frac{\text{TN}}{\text{N}}$. It measures the proportion of actual negatives that are correctly identified.

They correspond to two types of errors:

- Type-I error rate (1−Specificity): $\frac{\text{FP}}{\text{N}}$.
- Type-II error rate (1−Sensitivity): $\frac{\text{FN}}{\text{P}}$. It measures the proportion of actual negatives that are correctly identified.

For a probabilistic classifier, there are many ways to choose the decision threshold. Assume $Y = 0$ (negative) or 1 (positive):

- The Bayes classifier uses the threshold 0.5, which means $\hat{Y} = 1$ if and only if $P(Y = 1|\boldsymbol{x}) > 0.5$.
- It can be shown that the Bayes classifier will minimize the *overall error rate* $P(\hat{Y} \neq Y|\boldsymbol{x})$.
- However, in many applications the overall error rate is less important, because the cost of making one type of error surpasses that of the other type.
- For example, in COVID-surveillance, misclassifying a positive patient as negative is much more serious than misclassifying a negative patient.

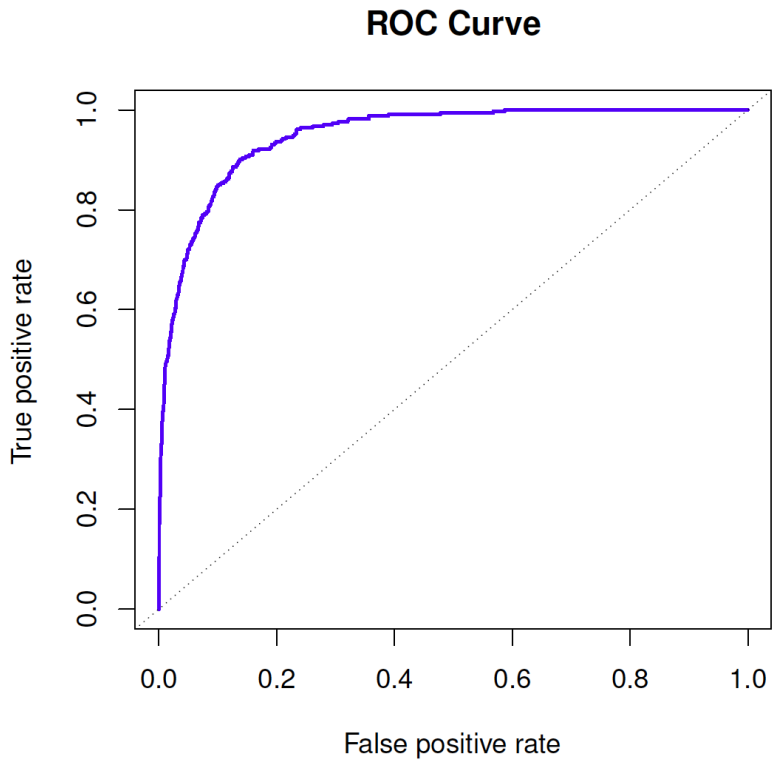
Depending on the application, the Bayes classifier isn't necessarily always the best way to choose the threshold.

- You need to determine which is more important between $P(\hat{Y} = 1|\boldsymbol{x}, Y = 0)$ and $P(\hat{Y} = 0|\boldsymbol{x}, Y = 1)$.
- *Question: which threshold will optimize the specificity of a classifier?*
- *Question: which threshold will optimize the sensitivity of a classifier?*

There is a trade-off between the specificity and the sensitivity of a classifier, based on the choice of the threshold.

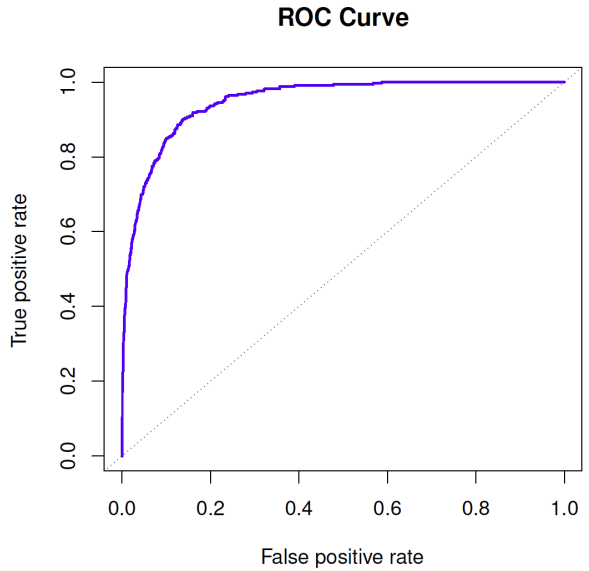
ROC and AUC

The ROC curve is a popular graphic for simultaneously displaying two types of errors for all possible thresholds.



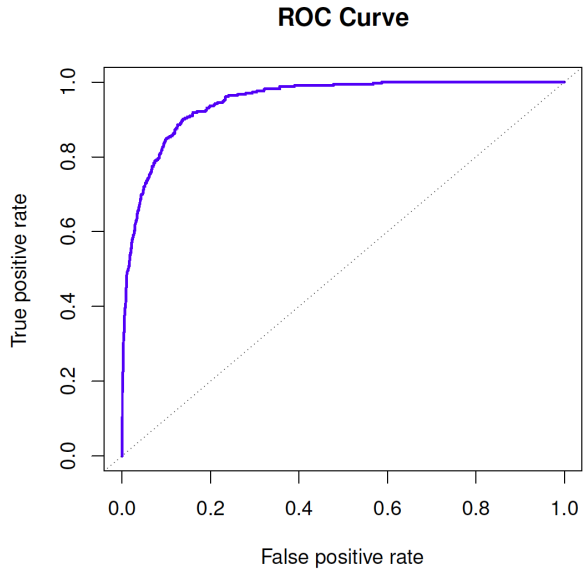
- The true positive rate is the sensitivity; The false positive rate is 1−specificity.

ROC and AUC



- The ideal ROC curve hugs the top left corner, indicating a high true positive rate and a low false positive rate.
- The dotted (45 degree) line represents the *no information* classifier, which is based on random guessing.
- In other words, the predicted probability $\hat{P}(Y = 1|\boldsymbol{x})$ is randomly generated from Unif$[0, 1]$.

ROC and AUC



- The overall performance of a classifier, summarized over all possible thresholds, is given by the area under the (ROC) curve (AUC).
- The larger the AUC, the better the classifier.
- The no-information classifier has an AUC of 0.5, which is entirely based on random guessing.
- Any reasonable classifier should have AUC larger than 0.5.

For next week:

- The midterm will be held on July 28th, during the regular lecture time (EX320).
- There will be no quiz during the midterm week. Extra office hours will be announced soon.
- There will be no problem set for this week. Instead, a practice midterm will be posted.
- The practice midterm has a representative *structure* for the midterm, but does not guarantee the same difficulty.

For next week:

- The midterm will cover lecture material from week 1 to week 3.
- The midterm is closed-book, and the only allowed item is a non-programmable calculator.
- The exercises from the problem sets, practice midterm and lecture slides should prepare you well for the midterm.
- You are not expected to write large chunks of R codes by hand, but you are expected to read and annotate R codes and outputs during the test.
- You might also be asked to write pseudo-code during the test, which does not have specific syntax requirements.
- Cheatsheet is neither allowed nor needed. Some long equations (i.e. normal density) will be provided whenever needed for that question.