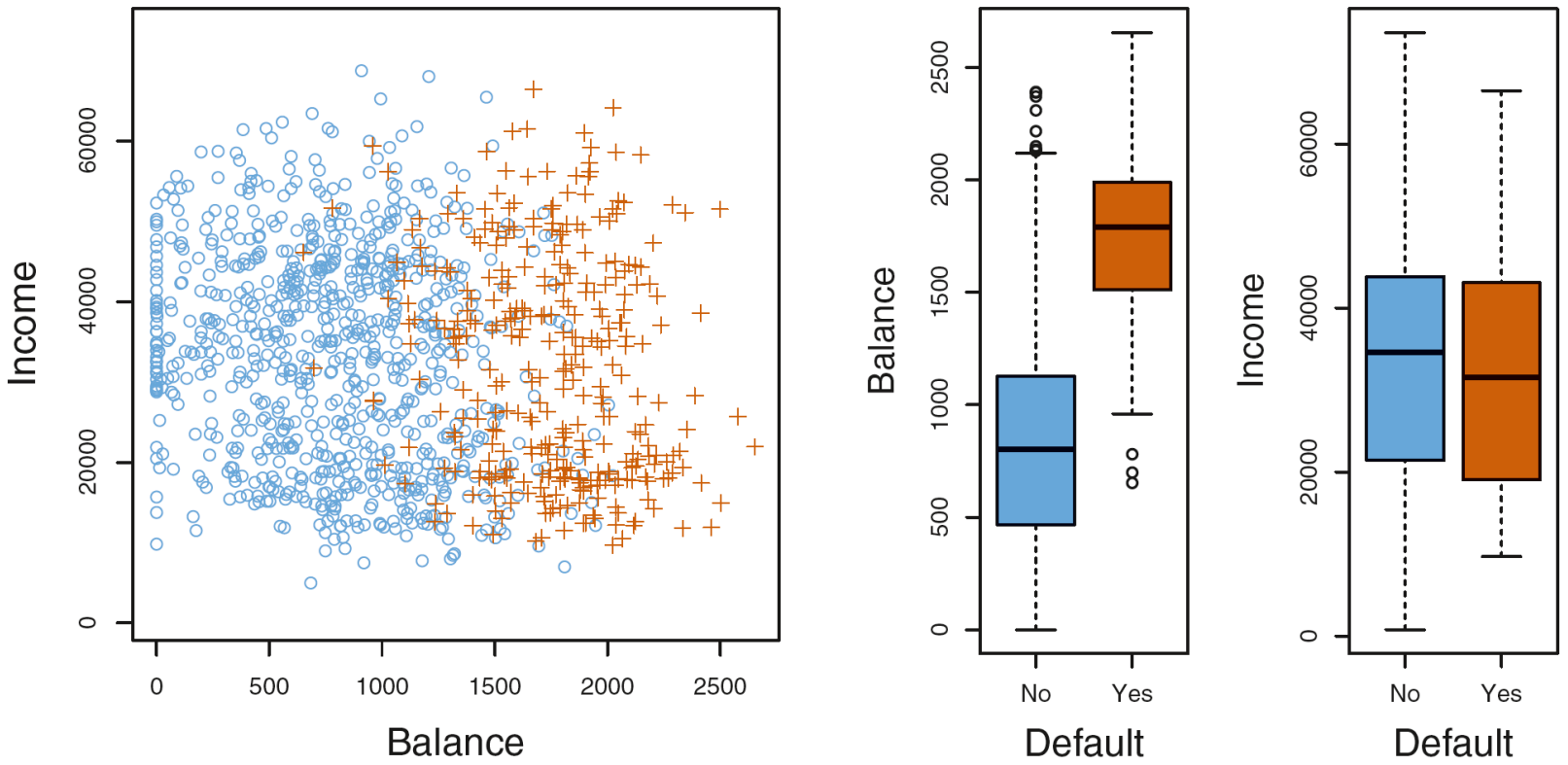
03. Classification Problems

Table of Contents

Classification

- Here the response variable Y is qualitative or categorical.
  - eye color: {brown, blue, green}
  - email: {spam, ham(not-spam)}
  - digit calss: $\{0, 1, \ldots, 9\}$
- Given a feature vector X and a qualitative response Y taking values in the set $\mathcal{C}$, the classification task is to build a function $C(X)$ that takes as input the feature vector X and predicts its value for Y, i.e., $C(X) \in \mathcal{C}$.
- Often we are more interested in estimating the probabilities that X belongs to each category in $\mathcal{C}$.
  - For example, it is more valuable to have an estimate of the probability that an insurance claim is fraudulent, than a classification fraudulent or not.

Example: Credit Card Default Data



- Left: The annual incomes and monthly credit card balances of a number of individuals. Default (orange) and Not-default (blue)
- Right: Boxplots of either balance or income as a function of default status

```
library(ISLR)
data(Default)
summary(Default)
attach(Default)
```

```
plot(income ~ balance, xlab="Balance", ylab="Income",
     pch=c(1,3)[unclass(default)],
     col=c("lightblue","red")[unclass(default)])
```

```
set.seed(1234)
ss <- sample(which(default=="No"), sum(default=="Yes"))
ss <- c(ss, which(default=="Yes"))
us <- unclass(default[ss])
plot(income[ss] ~ balance[ss], xlab="Balance", pch=c(1,3)[us],
     col=c("lightblue","red")[us], ylab="Income")
```

```
par(mfrow=c(1,2))
boxplot(balance~default, col=c("lightblue","red"), boxwex=0.5,
        xlab="Default", ylab="Balance")
boxplot(income~default, col=c("lightblue","red"), boxwex=0.5,
        xlab="Default", ylab="Income")
```

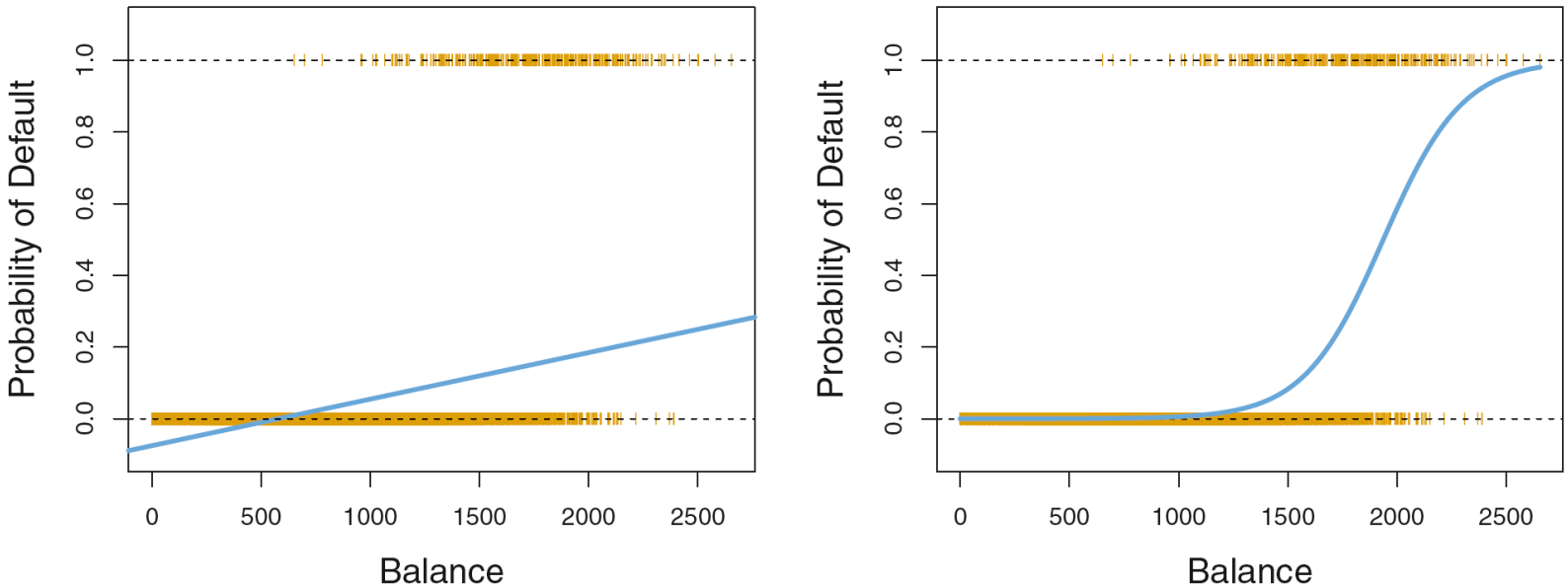
Linear Regression

- Suppose for the Default classification task that we code

$$Y = \begin{cases} 0, & \text{if No} \\ 1, & \text{if Yes} \end{cases}$$

- Can we simply perform a linear regression of Y on X and classify as Yes if $\hat{Y} > 0.5$?
  - In this case of a binary outcome, linear regression does a good job as a classifier, and is equivalent to linear discriminant analysis which we discuss later.
  - Since in the population $E(Y|X = x) = Pr(Y = 1|X = x)$, we might think that regression is perfect for this task.
  - However, linear regression might produce probabilities less than zero or bigger than one. Logistic regression is more appropriate.

Linear versus Logistic Regression



- The orange marks indicate the response Y, either 0 or 1.
- Linear regression does not estimate $Pr(Y = 1|X)$ well in the left while logistic regression seems well suited to the task in the right.

```
ndef <- rep(0, length(default))
ndef[default=="Yes"] <- 1
```

```
g1 <- glm(ndef ~ balance)
g2 <- glm(default ~ balance, family="binomial")
```

```
par(mfrow=c(1,2))
plot(balance, ndef, pch="|", col="orange", xlab="Balance",
     ylab="Probability of Default",ylim=c(-0.1,1.1))
abline(h=c(0,1), lty=2)
lines(balance, g1$fit, col="lightblue", lwd=2)
```

```
plot(balance, as.numeric(default)-1, pch="|", col="orange",
     xlab="Balance", ylab="Probability of Default",
     ylim=c(-0.1,1.1))
abline(h=c(0,1), lty=2)
u <- order(balance)
lines(balance[u], g2$fit[u], col="lightblue", lwd=3)
```

Logistic Regression

- Let's write $p(X) = Pr(Y = 1|X)$ for short and consider using balance to predict default. Logistic regression uses the form

$$p(X) = \frac{e^{\beta_0+\beta_1 X}}{1 + e^{\beta_0+\beta_1 X}}$$

- It is easy to see that no matter what values β_0, β_1 or X take, $p(X)$ will have values between 0 and 1.
- A bit of rearrangement gives

$$\log\left(\frac{p(X)}{1 - p(X)}\right) = \beta_0 + \beta_1 X.$$

- This monotone transformation is called the log odds or logit transformation of $p(X)$.
- Logistic regression ensures that our estimate for $p(X)$ lies between 0 and 1.

Maximum Likelihood

- We use maximum likelihood to estimate the parameters.

$$l(\beta_0, \boldsymbol{\beta}) = \prod_{i:y_i=1} p(x_i) \prod_{i:y_i=0} (1 - p(x_i))$$
$$= \prod_{i=1}^{n} \left(\frac{e^{\beta_0 + x_i^{\mathrm{T}} \beta_1}}{1 + e^{\beta_0 + x_i^{\mathrm{T}} \beta_1}} \right)^{y_i} \left(\frac{1}{1 + e^{\beta_0 + x_i^{\mathrm{T}} \beta_1}} \right)^{1 - y_i}$$

- This likelihood gives the probability of the observed zeros and ones in the data. We pick β_0 and β_1 to maximize the likelihood of the observed data.
- Most statistical packages can fit linear logistic regression models by maximum likelihood. In R we use the glm function.

```
g2 <- glm(default ~ balance, family="binomial")
summary(g2)$coef
```

```
## Fitted values
g2$fit
```

```
## inverse logistic function
ilogit <- function(x, coef) {
    exp(cbind(1, x) %*% coef) / (1 + exp(cbind(1, x) %*% coef))
}
cbind(g2$fit, ilogit(balance, g2$coef))
ilogit(1000, g2$coef)
```

```
g3 <- glm(default ~ student, family="binomial")
summary(g3)$coef
```

```
## Student "Yes"
ilogit(1, g3$coef)
## Student "No"
ilogit(0, g3$coef)
```

Logistic Regression with Several Variables

- When we have p predictors,

$$\log\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p,$$

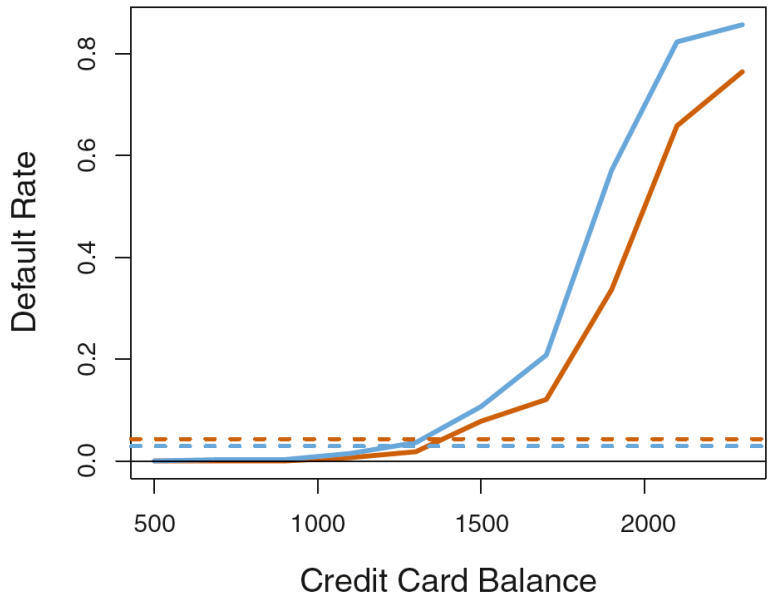
where

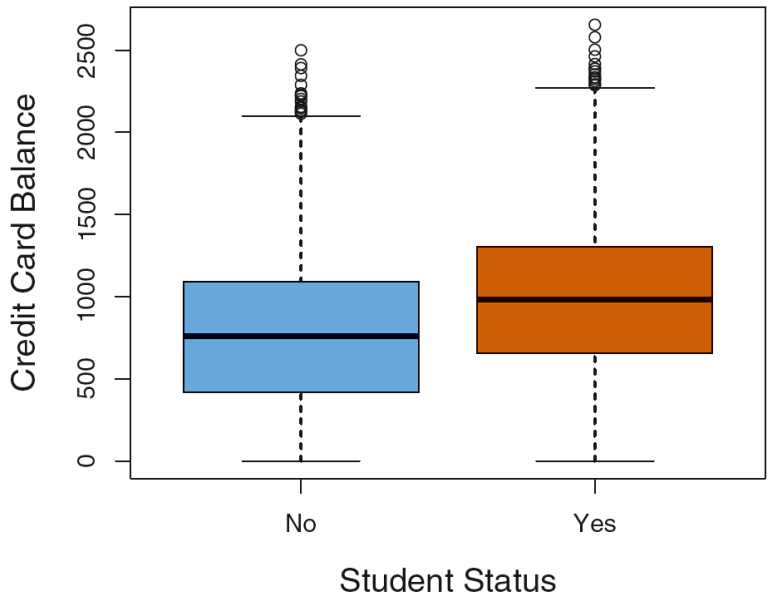
$$p(X) = \frac{e^{\beta_0+\beta_1 X_1+\cdots+\beta_p X_p}}{1+e^{\beta_0+\beta_1 X_1+\cdots+\beta_p X_p}}$$

```
g4 <- glm(default~ balance + income + student, family="binomial")
round(summary(g4)$coef, 4)
```

- Why is coefficient for student negative, while it was positive before?

Confounding





- Students tend to have higher balances than non-students, so their marginal default rate is higher than for non-students.
- But for each level of balance, students default less than non-students.
- Multiple logistic regression can tease this out.

```
yst <- g4$fit[student=="Yes"]
nst <- g4$fit[student=="No"]
plot(balance, g2$fit, col="white", xlab="Credit Card Balance",
     ylab="Default Rate")
abline(h=0)
abline(h=mean(yst), lty=2, col="orange")
abline(h=mean(nst), lty=2, col="lightblue")
u1 <- order(balance[student=="Yes"])
u2 <- order(balance[student=="No"])
lines(balance[student=="Yes"][u1], yst[u1], col="orange", lwd=2)
lines(balance[student=="No"][u2], nst[u2], col="lightblue", lwd=2)
```

```
boxplot(balance ~ student, col=c("lightblue","orange"),
    xlab="Student Status", ylab="Credit Card Balance", boxwex=0.6)
```

```
ilogit(cbind(1500, 40, 1), g4$coef)
ilogit(cbind(1500, 40, 0), g4$coef)
xb <- predict(g4, data.frame(balance=1500, income=40,
                student="Yes"))
exp(xb)/(1+exp(xb))
predict(g4, data.frame(balance=1500, income=40, student="Yes"),
        type="response")
```

```
set.seed(1111)
n <- nrow(Default)
train <- sample(1:n, n*0.7)
test <- setdiff(1:n, train)
```

```
g1 <- glm(default ~ balance, family="binomial", subset=train)
g2 <- glm(default ~ student, family="binomial", subset=train)
g3 <- glm(default ~ income, family="binomial", subset=train)
g4 <- glm(default~ balance+student+income, family="binomial",
          subset=train)
```

```
miss <- NULL
for (k in 1:4) {
    g <- get(paste("g", k, sep=""))
    pred <- predict(g, Default, type="response")[test]
    yhat <- rep(0, length(test))
    yhat[pred > 0.5] <- 1
    miss[k] <- mean(yhat!=as.numeric(default[test])-1)
}
miss
```

Logistic Regression with More Than Two Classes

- So far we have discussed logistic regression with two classes, but it is easily generalized to more than two classes.
- When $Y \in \{1, 2, \ldots, K\}$,

$$Pr(Y = k|X) = \frac{e^{\beta_{0k}+\beta_{1k}X_1+\cdots+\beta_{pk}X_p}}{\sum_{l=1}^{K} e^{\beta_{0l}+\beta_{1l}X_1+\cdots+\beta_{pl}X_p}}$$

- There is a linear function for each class. Note that some cancellation is possible, and only $K - 1$ linear functions are needed as in K-class logistic regression.
- Multiclass logistic regression is also referred to as multinomial regression.

```
library(remotes)
install_github("cran/rattle.data")
```

```
library(rattle.data)
library(nnet)
data(wine)
```

```
str(wine)
summary(wine)
plot(wine[, -1], col=as.numeric(wine$Type) + 1)
plot(wine[, 2:7], col=as.numeric(wine$Type) + 1)
plot(wine[, 8:14], col=as.numeric(wine$Type) + 1)
```

```
fit <- multinom(Type ~ ., data=wine, trace=FALSE)
summary(fit)
```

```
z <- coef(summary(fit))/summary(fit)$standard.errors
pnorm(abs(z), lower.tail=FALSE)*2
```

```
set.seed(1)
u <- sort(sample(1:nrow(wine), 10))
fitted(fit)[u,]
predict(fit, wine, type="prob")[u,]
```

```
prob0 <- predict(fit, wine, type="prob")
pred0 <- apply(prob0, 1, which.max)
table(pred0, wine$Type)
```

```
pred0a <- predict(fit, wine, type="class")
table(pred0a, wine$Type)
```

```
set.seed(1111)
n <- nrow(wine)
train <- sample(1:n, round(n*0.7))
test <- setdiff(1:n, train)
```

```
fit1 <- multinom(Type ~ Alcohol + Color, data=wine,
                 subset=train)
summary(fit1)
```

```
pred1 <- predict(fit1, wine, type="class")
tab1 <- table(pred1[test], wine$Type[test])
1-sum(diag(tab1))/sum(tab1)
```

```
fit2 <- multinom(Type ~ ., data=wine, subset=train)
summary(fit2)
```

```
pred2 <- predict(fit2, wine, type="class")
tab2 <- table(pred2[test], wine$Type[test])
1-sum(diag(tab2))/sum(tab2)
```

```
set.seed(12345)
miss <- NULL
for (k in 1:100) {
    train <- sample(1:n, round(n*0.7))
    test <- setdiff(1:n, train)
    g <- multinom(Type ~ ., data=wine, subset=train, trace=FALSE)
    pred <- predict(g, wine, type="class")
    tab <- table(pred[test], wine$Type[test])
    miss[k] <- 1-sum(diag(tab))/sum(tab)
}
summary(miss)
hist(miss, main="Classification Error Rate", col="orange")
```

Bayes Theorem in Classification

- Here the approach is to model the distribution of X in each of the classes separately, and then use Bayes theorem to flip things around and obtain $Pr(Y|X)$.

$$Pr(Y = k|X = x) = \frac{Pr(X = x|Y = k) \cdot Pr(Y = k)}{Pr(X = x)}$$
$$= \frac{Pr(X = x|Y = k) \cdot Pr(Y = k)}{\sum_k Pr(X = x|Y = k) \cdot Pr(Y = k)}$$

- When we use normal (Gaussian) distributions for each class, this leads to linear or quadratic discriminant analysis.
- However, this approach is quite general, and other distributions can be used as well. We will focus on normal distributions.

Discriminant Analysis

- One writes Bayes theorem slightly differently for discriminant analysis:

$$Pr(Y = k|X = x) = \frac{\pi_k f_k(x)}{\sum_{l=1}^{K} \pi_l f_l(x)}$$

- $f_k(x) = Pr(X = x|Y = k)$ is the density for X in class k. We use normal densities for these, separately in each class.

$$f_k(x) = \frac{1}{\sqrt{2\pi}\sigma_k} e^{-\frac{(x-\mu_k)^2}{2\sigma_k^2}} \quad \text{for} \quad k = 1, \ldots, K$$

- $\pi_k = Pr(Y = k)$ is the marginal or prior probability for class k. It can be easily estimated from the sample proportion of class k among n observations.

Why Discriminant Analysis?

- When the classes are well-separated, the parameter estimates for the logistic regression model are surprisingly unstable. Linear discriminant analysis does not suffer from this problem.
- If n is small and the distribution of the predictors X is approximately normal in each of the classes, the linear discriminant model is again more stable than the logistic regression model.
- Linear discriminant analysis is popular when we have more than two response classes, because it also provides low-dimensional views of the data.

Linear Discriminant Analysis when $p = 1$

- The Gaussian density has the form

$$f_k(x) = \frac{1}{\sqrt{2\pi}\sigma_k} e^{-\frac{1}{2}\left(\frac{x-\mu_k}{\sigma_k}\right)^2},$$

  where μ_k is the mean, and σ_k^2 the variance (in class k).
- We assume that the $\sigma = \sigma_1 = \sigma_2 = \ldots = \sigma_K$ are the same.
- Plugging this into Bayes formula, we get a rather complex expression for $p_k(x) = Pr(Y = k|X = x)$:

$$p_k(x) = \frac{\pi_k \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{x-\mu_k}{\sigma}\right)^2}}{\sum_{l=1}^{K} \pi_l \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{x-\mu_l}{\sigma}\right)^2}}$$

- Fortunately, there are simplifications and cancellations.

Discriminant Functions

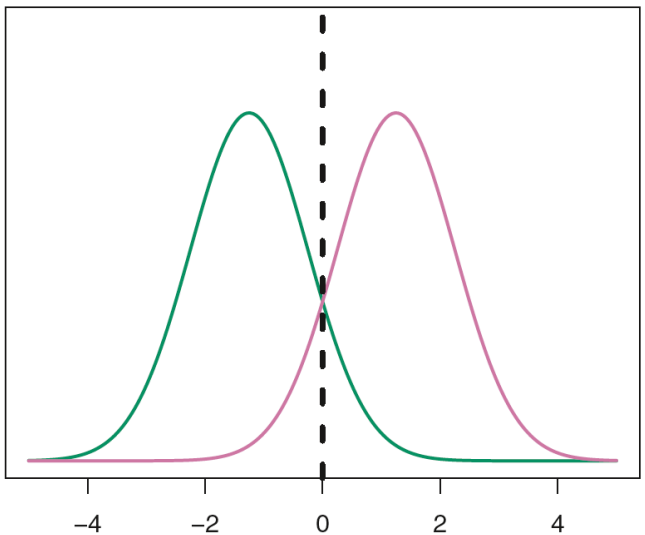
- To classify at the value $X = x$, we need to see which of the $p_k(x)$ is largest. Taking logs, and discarding terms that do not depend on k, we see that this is equivalent to assigning x to the class with the largest discriminant score:

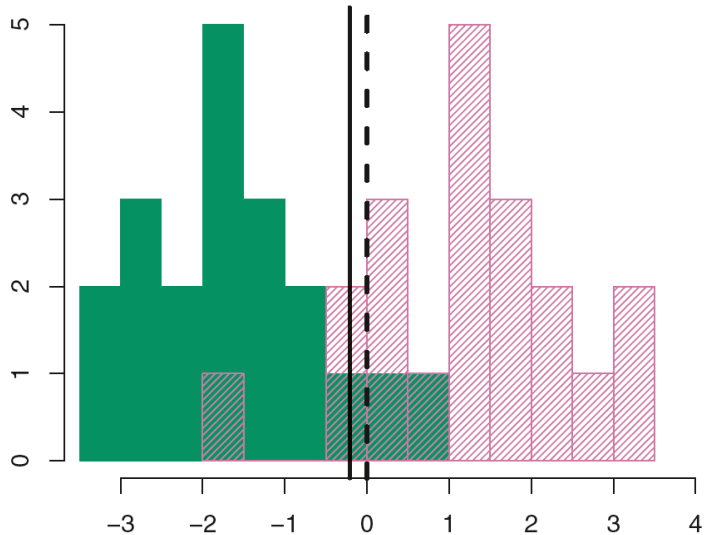
$$\delta_k(x) = x \cdot \frac{\mu_k}{\sigma^2} - \frac{\mu_k^2}{2\sigma^2} + \log(\pi_k)$$

- Note that $\delta_k(x)$ is a linear function of x.
- If there are $K = 2$ classes and $\pi_1 = \pi_2 = 0.5$, then one can see that the decision boundary is at

$$x = \frac{\mu_1 + \mu_2}{2}$$

Decision Boundary





- Example with $\mu_1 = -1.5$, $\mu_2 = 1.5$, $\pi_1 = \pi_2 = 0.5$ and $\sigma^2 = 1$.
- Typically we don't know these parameters; we just have the training data. In that case we simply estimate the parameters and plug them into the rule.

Decision Boundary

```
x <- seq(-5, 5, 0.1)
plot(x, dnorm(x, -1.5, 1), type="l", col="darkgreen", xlab="",
     ylab="", yaxt="n", xlim=c(-5,5), lwd=2)
lines(x, dnorm(x, 1.5, 1), col="red", lwd=2)
abline(v=0, lwd=3, lty=2)
```

```
x1 <- rnorm(20, -1.5, 1)
x2 <- rnorm(20, 1.5, 1)
hist(x1, col=rgb(0.4,1,0,0.8), breaks=10, xlab="", ylab="",
     main="", xlim=c(-4,4))
hist(x2, col=rgb(0.7,0,0,0.7), breaks=10, add=T)
abline(v=0, lwd=3, lty=2)
abline(v=(mean(x1)+mean(x2))/2, lwd=3, lty=1)
legend("topleft", c("True", "Estimated"), lty=c(2,1), lwd=1,
       bty="n", cex=0.9)
```

Estimating The Parameters

- Parameters are estimated using the following formula

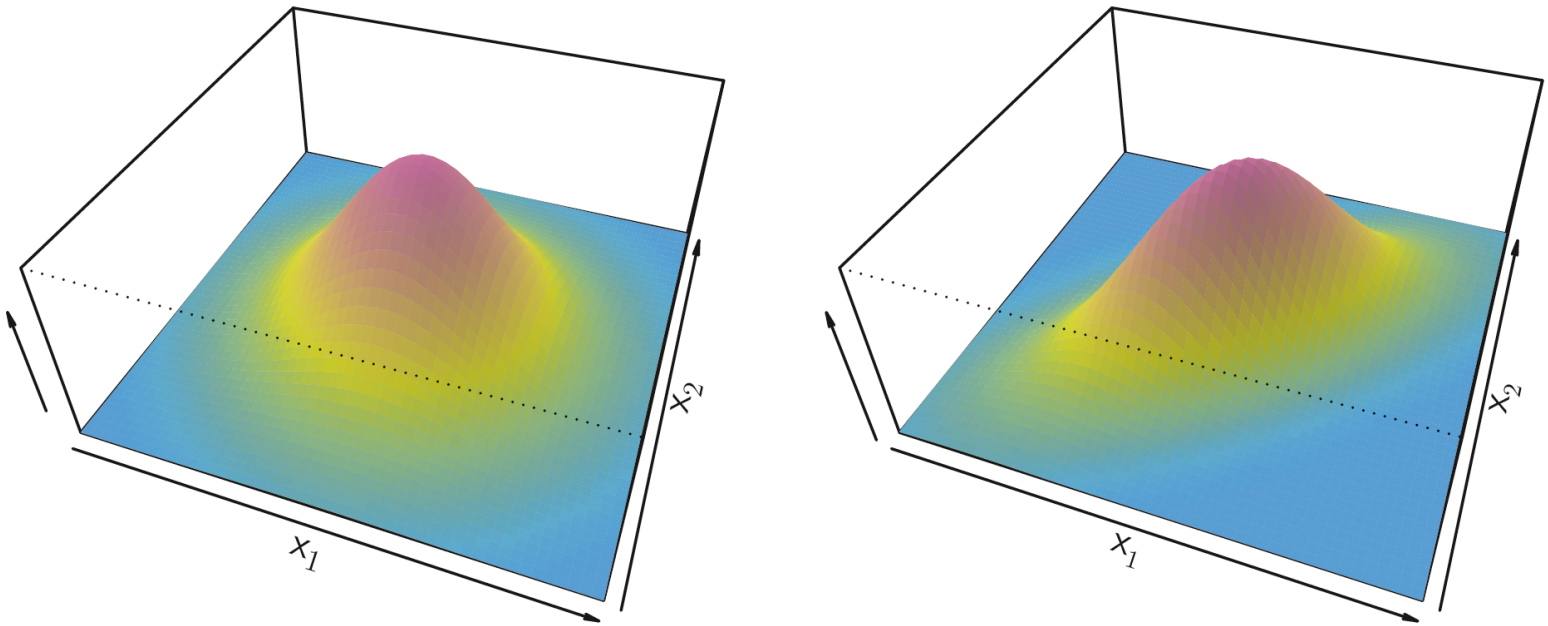
$$\begin{aligned}
\hat{\pi}_k &= \frac{n_k}{n}, \qquad \hat{\mu}_k = \frac{1}{n_k} \sum_{i:y_i=k} x_i \\
\hat{\sigma}^2 &= \frac{1}{n-K} \sum_{k=1}^{K} \sum_{i:y_i=k} (x_i - \hat{\mu}_k)^2 \\
&= \sum_{k=1}^{K} \frac{n_k - 1}{n-K} \hat{\sigma}_k^2
\end{aligned}$$

where

$$\hat{\sigma}_k^2 = \frac{1}{n_k - 1} \sum_{i:y_i=k} (x_i - \hat{\mu}_k)^2$$

is the the usual formula for the estimated variance in the kth class.

Linear Discriminant Analysis when $p > 1$



- Density: a multivariate Gaussian distribution

$$f(x) = \frac{1}{(2\pi)^{p/2}|\Sigma|^{1/2}} e^{-\frac{1}{2}(x-\mu)^{\mathrm{T}}\Sigma^{-1}(x-\mu)}$$

- Observations in the kth class are drawn from a multivariate Gaussian distribution $N(\mu_k, \Sigma)$, where μ_k is a class-specific mean vector, and Σ is a covariance matrix that is common to all K classes.

Linear Discriminant Analysis when $p > 1$

- Discriminant function

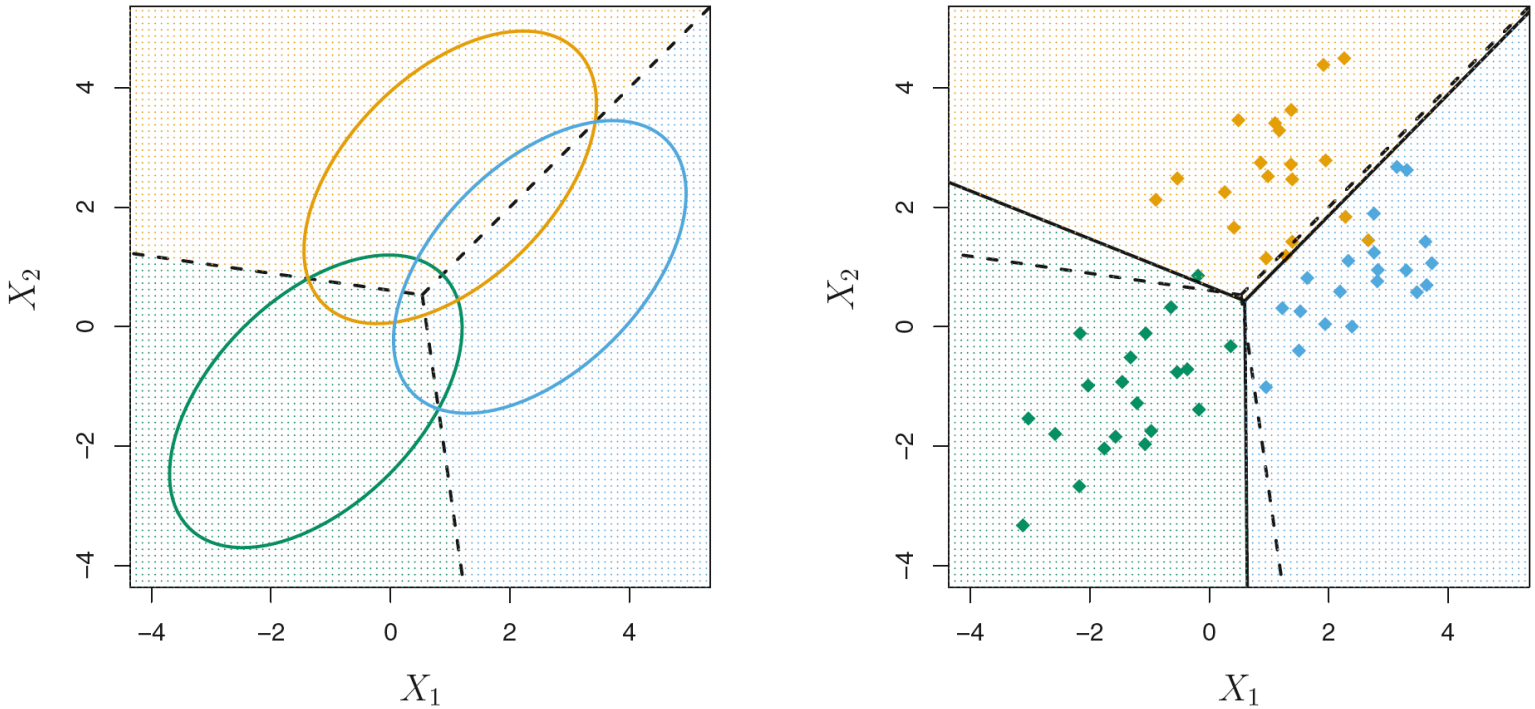
$$\delta_k(x) = x^{\mathrm{T}}\Sigma^{-1}\mu_k - \frac{1}{2}\mu_k^{\mathrm{T}}\Sigma^{-1}\mu_k + \log \pi_k$$

- Despite its complex form, the discriminant function

$$\delta_k(x) = c_{k0} + c_{k1}x_1 + c_{k2}x_2 + \ldots + c_{kp}x_p$$

  is a linear function.
- LDA decision rule depends on x only through a linear combination of its elements.
- We need to estimate the unknown paramters $\mu_1, \ldots, \mu_K$, $\pi_1, \ldots, \pi_K$ and Σ.

Example: $p = 2$ and $K = 3$



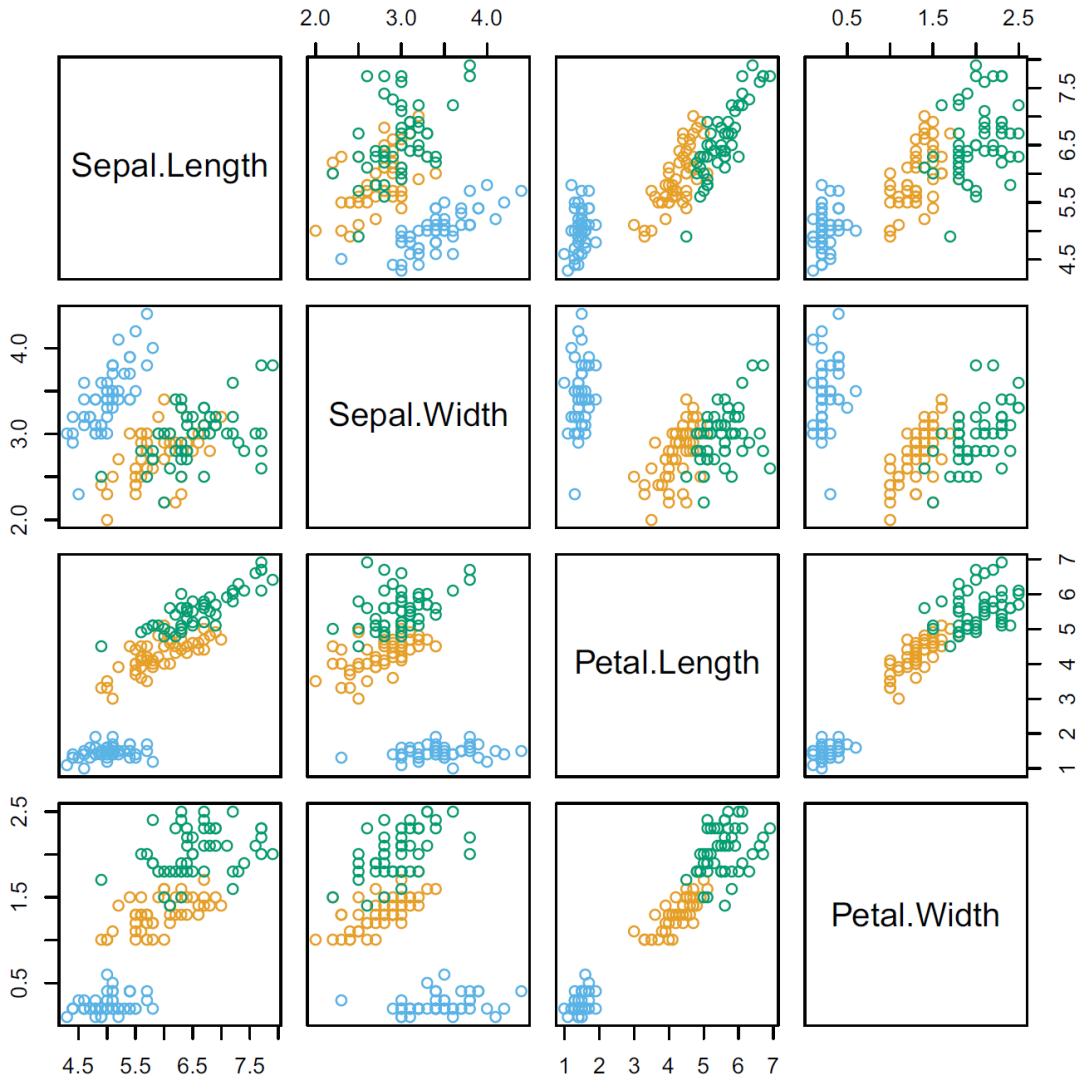
- In this example, $\pi_1 = \pi_2 = \pi_3 = 1/3$.
- The dashed lines are known as the Bayes decision boundaries. If they were known, they would yield the fewest misclassification errors, among all possible classifiers.

Fisher's Iris Data

4 variables
3 species
50 samples/class

- Setosa
- Versicolor
- Virginica

LDA classifies all but 3 of the 150 training samples correctly.



LDA: Fisher's Iris Data

```
## Open the iris dataset
data(iris)
?iris
str(iris)
summary(iris)
plot(iris[, -5], col=as.numeric(iris$Species) + 1)
```

```
## Apply LDA for iris data
library(MASS)
g <- lda(Species ~., data=iris)
plot(g)
plot(g, dimen=1)
```

```
## Compute misclassification error for training sets
pred <- predict(g)
table(pred$class, iris$Species)
mean(pred$class!=iris$Species)
```

```
## Randomly separate training sets and test sets
set.seed(1234)
tran <- sample(nrow(iris), size=floor(nrow(iris)*2/3))
g <- lda(Species ~., data=iris, subset=tran)
```

```
## Compute misclassification error for test sets
pred <- predict(g, iris)$class[-tran]
test <- iris$Species[-tran]
table(pred, test)
mean(pred!=test)
```

```
## Posterior probability
post <- predict(g, iris)$posterior[-tran,]
post[1:10,]
apply(post, 1, which.max)
as.numeric(pred)
```

LDA vs. Multinomial Regression

```
library(nnet)
```

```
set.seed(1234)
K <- 100
RES <- array(0, c(K, 2))
for (i in 1:K) {
    tran.num <- sample(nrow(iris), size=floor(nrow(iris)*2/3))
    tran <- as.logical(rep(0, nrow(iris)))
    tran[tran.num] <- TRUE
    g1 <- lda(Species ~., data=iris, subset=tran)
    g2 <- multinom(Species ~., data=iris, subset=tran, trace=FALSE)
    pred1 <- predict(g1, iris[!tran,])$class
    pred2 <- predict(g2, iris[!tran,])
    RES[i, 1] <- mean(pred1!=iris$Species[!tran])
    RES[i, 2] <- mean(pred2!=iris$Species[!tran])
}
```

```
apply(RES, 2, mean)
```

```
library(rattle.data)
data(wine)
```

```
set.seed(1111)
RES2 <- array(0, c(K, 2))
for (i in 1:K) {
    tran.num <- sample(nrow(wine), size=floor(nrow(wine)*2/3))
    tran <- as.logical(rep(0, nrow(wine)))
    tran[tran.num] <- TRUE
    g1 <- lda(Type ~., data=wine, subset=tran)
    g2 <- multinom(Type ~., data=wine, subset=tran, trace=FALSE)
    pred1 <- predict(g1, wine[!tran,])$class
    pred2 <- predict(g2, wine[!tran,])
    RES2[i, 1] <- mean(pred1!=wine$Type[!tran])
    RES2[i, 2] <- mean(pred2!=wine$Type[!tran])
}
```

```
apply(RES2, 2, mean)
```

Posterior Probabilities

- Once we have estimates $\hat{\delta}_k(x)$, we can turn these into estimates for class probabilities:

$$\hat{Pr}(Y = k|X = x) = \frac{e^{\hat{\delta}_k(x)}}{\sum_{l=1}^{K} e^{\hat{\delta}_k(x)}}.$$

- So classifying to the largest $\hat{\delta}_k(x)$ amounts to classifying to the class for which $\hat{Pr}(Y = k|X = x)$ is largest.
- When $K = 2$, we classify to class 2 if

$$\hat{Pr}(Y = 2|X = x) \geq 0.5,$$

else to class 1.

LDA on Default Data

```
library(ISLR)
data(Default)
attach(Default)
```

```
library(MASS)
g <- lda(default~., data=Default)
pred <- predict(g, default)
table(pred$class, default)
mean(pred$class!=default)
```

		True Default Status		
		No	Yes	Total
Predicted	No	9645	254	**9899**
Default Status	Yes	22	79	**101**
		9667	**333**	**10000**

LDA on Default Data

- The misclassification error is only 2.76%! However, this is training error, and we may be overfitting. Not a big concern here since $n = 10000$ and $p = 4$!
- If we classified to the prior – always to class "No". In this case, we would make 333/10000 errors, which is only 3.33%.
- The trivial null classifier achieve an error rate that null is only a bit higher than the LDA training set error rate.
- Two types of errors
  - Of the true No's, we make $22/9667 = 0.22\%$ errors
  - Of the true Yes's, we make $254/333 = 76.2\%$ errors

Two Types of Classification Errors

- False positive rate: The fraction of negative examples that are classified as positive. (0.22% in the example)
- False negative rate: The fraction of positive examples that are classified as negative. (76.2% in the example)
- We produced this table by classifying to class 'Yes' if

$$\hat{Pr}(\text{Default} = \text{"Yes"} | \text{Balance, Student}) \geq 0.5$$

- We can change the two error rates by changing the threshold from 0.5 to some other value in [0, 1]:

$$\hat{Pr}(\text{Default} = \text{"Yes"} | \text{Balance, Student}) \geq \alpha,$$

where $\alpha \in [0, 1]$ is a threshold.

Changes in Errors along with Different Thresholds

```
thre <- seq(0,1,0.01)
res <- matrix(NA, length(thre), 3)
```

```
## Compute overall error, false positives, false negatives
for (i in 1:length(thre)) {
    decision <- rep("No", length(default))
    decision[pred$posterior[,2] >= thre[i]] <- "Yes"
    res[i, 1] <- mean(decision != default)
    res[i, 2] <- mean(decision[default=="No"]=="Yes")
    res[i, 3] <- mean(decision[default=="Yes"]=="No")
}
```

```
k <- 1:51
matplot(thre[k], res[k,], col=c(1,"orange",4), lty=c(1,4,2),
        type="l", xlab="Threshold", ylab="Error Rate", lwd=2)
legend("top", c("Overall Error", "False Positive",
       "False Negative"), col=c(1,"orange",4), lty=c(1,4,2),
       cex=1.2)
apply(res, 2, which.min)
```

Confusion Matrix

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

Name	Definition	Synonyms
False Pos. rate	FP/N	Type I error, 1−Specificity
True Pos. rate	TP/P	1−Type II error, power, sensitivity, recall
Pos. Pred. value	TP/P*	Precision, 1−false discovery proportion
Neg. Pred. value	TN/N*	

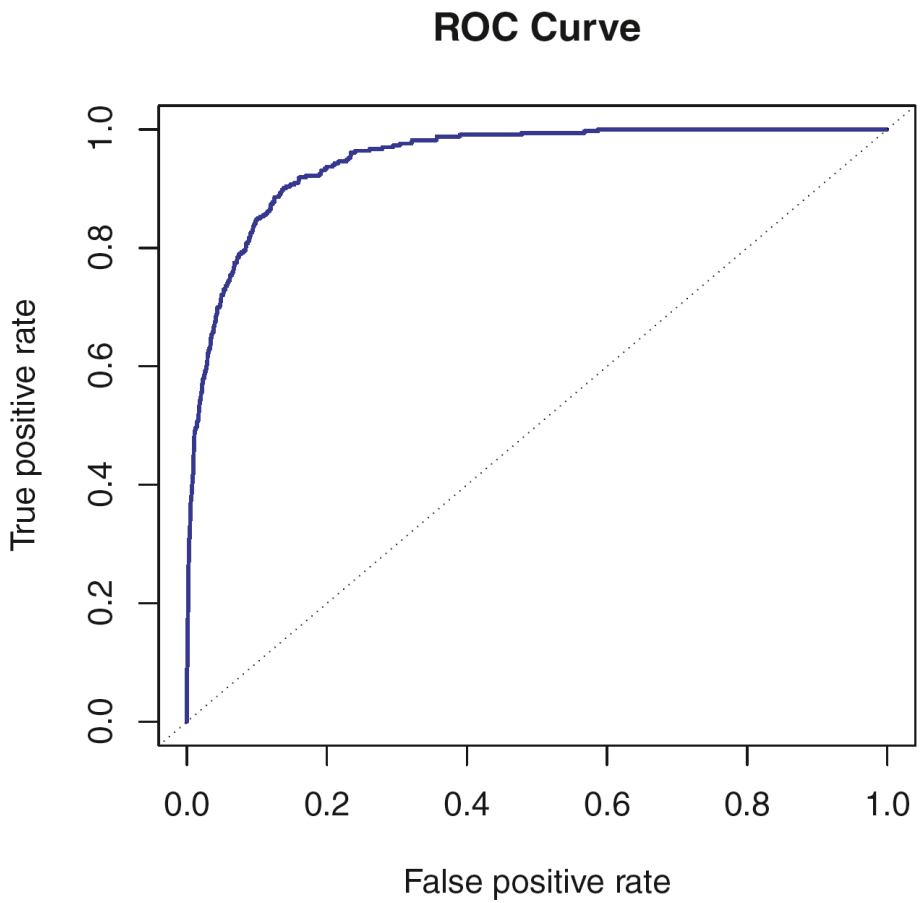
Variable selection in high-dimensional data

- H_0: Null $\approx$ Negative $(-)$ $\approx$ Not significant
- H_1: Non-null $\approx$ Positive $(+)$ $\approx$ Significant

Assessment of the Performance of Classifiers

- Class-specific performance is important in medicine and biology, where the terms sensitivity (true positive rate) and specificity (true negative rate) characterize the performance of a classifier or screening test.
- The ROC (Receiver Operating Characteristics) curve is a popular graphic for simultaneously displaying the two types of errors for all possible thresholds.
- The overall performance of a classifier, summarized over all possible thresholds, is given by the area under the (ROC) curve (AUC).
  - An ideal ROC curve will hug the top left corner, so the larger the AUC the better the classifier.
  - We expect a classifier that performs no better than chance to have an AUC of 0.5.

ROC curve



- The ROC plot displays both true positive rate and false positive rate simultaneously.

```
thre <- seq(0,1,0.001)
Sen <- Spe <- NULL
RES <- matrix(NA, length(thre), 4)
colnames(RES) <- c("TP", "TN", "FP", "FN")
```

```
for (i in 1:length(thre)) {
    decision <- rep("No", length(default))
    decision[pred$posterior[,2] >= thre[i]] <- "Yes"
    Sen[i] <- mean(decision[default=="Yes"] == "Yes")
    Spe[i] <- mean(decision[default=="No"] == "No")
    RES[i,1] <- sum(decision[default=="Yes"] == "Yes")
    RES[i,2] <- sum(decision[default=="No"] == "No")
    RES[i,3] <- sum(decision=="Yes") - RES[i,1]
    RES[i,4] <- sum(default=="Yes") - RES[i,1]
}
```

```
plot(1-Spe, Sen, type="b", pch=20, xlab="False positive rate",
     col="darkblue", ylab="True positive rate", main="ROC Curve")
abline(0, 1, lty=3, col="gray")
```

```
TPR <- RES[,1] / (RES[,1] + RES[,4])
TNR <- RES[,2] / (RES[,2] + RES[,3])
```

```
plot(1-TNR, TPR, type="b", pch=20, xlab="False positive rate",
     col="darkblue", ylab="True positive rate", main="ROC Curve")
abline(0, 1, lty=3, col="gray")
```

```
PPV <- RES[,1] / (RES[,1] + RES[,3])
NPV <- RES[,2] / (RES[,2] + RES[,4])
```

```
MAT <- cbind(TPR, TNR, PPV, NPV)
matplot(thre, MAT, type="l", lty=c(1,2,1,2), col=c(1,1,2,2),
        xlab="thresholds", ylab="")
legend("right", c("TPR", "TNR", "PPV", "NPV"),
       lty=c(1,2,1,2), col=c(1,1,2,2), bty="n", cex=1.2)
```

```
library(ROCR)
```

```
## Compute ROC curve
label <- factor(default, levels=c("Yes","No"),
                labels=c("TRUE","FALSE"))
preds <- prediction(pred$posterior[,2], label)
perf <- performance(preds, "tpr", "fpr" )
plot(perf, lwd=4, col="darkblue")
abline(a=0, b=1, lty=2)
```

```
slotNames(perf)
k <- 1:100
list(perf@x.name, perf@x.values[[1]][k])
list(perf@y.name, perf@y.values[[1]][k])
list(perf@alpha.name, perf@alpha.values[[1]][k])
```

```
## Compute AUC
performance(preds, "auc")@y.values
```

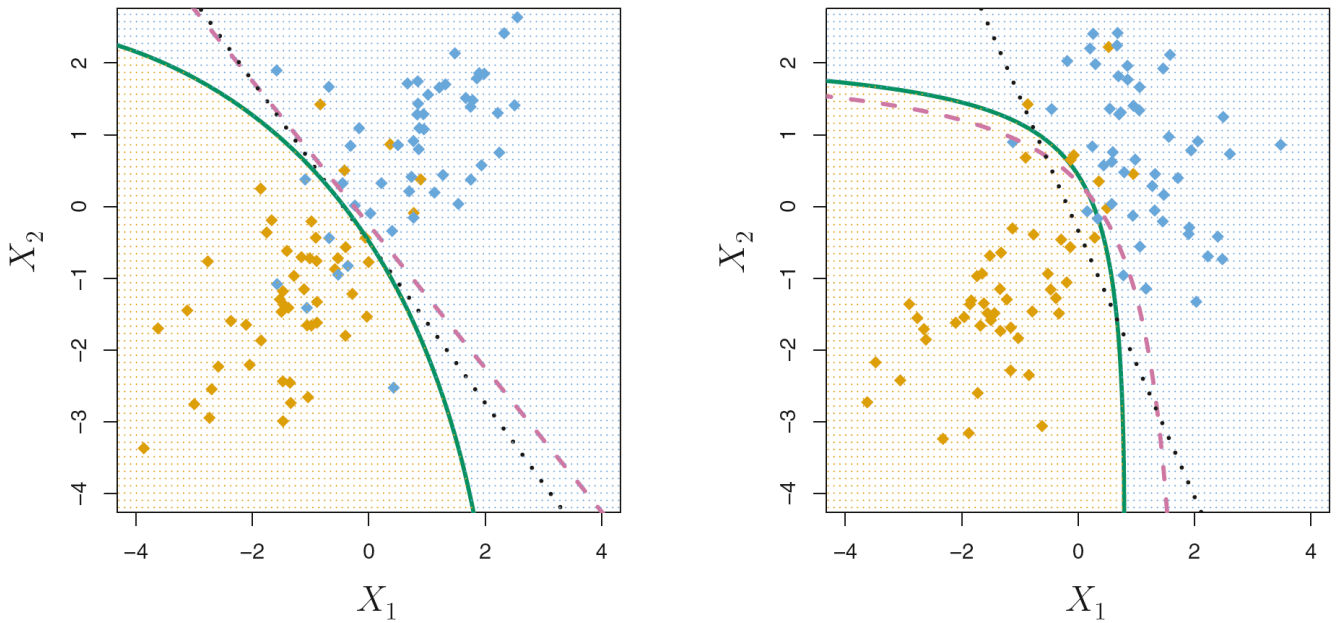
Quadratic Discriminant Analysis

- QDA assumes that each class has its own covariance matrix, i.e., $X \sim N(\mu_k, \Sigma_k)$.
- Under this assumption, the Bayes classifier assigns an observation $X = x$ to the class for which

$$\begin{aligned}\delta_k(x) &= -\frac{1}{2}(x-\mu_k)^{\mathrm{T}}\Sigma_k^{-1}(x-\mu_k) + \log \pi_k \\ &= -\frac{1}{2}x^{\mathrm{T}}\Sigma_k^{-1}x + x^{\mathrm{T}}\Sigma_k^{-1}\mu_k - \frac{1}{2}\mu_k^{\mathrm{T}}\Sigma_k^{-1}\mu_k + \log \pi_k\end{aligned}$$

  is largest.
- QDA estimates a separate covariance matrix for each class, for a total of $Kp(p+1)/2$ parameters.
- LDA is a much less flexible classifier than QDA, and so has substantially lower variance.

LDA vs. QDA



- The Bayes (purple dashed), LDA (black dotted), and QDA (green solid) decision boundaries for a two-class problem.
- In the left: $\Sigma_1 = \Sigma_2$ and in the right: $\Sigma_1 \neq \Sigma_2$

LDA vs. QDA: Default Data

```
library(MASS)
```

```
set.seed(1234)
n <- nrow(Default)
train <- sample(1:n, n*0.7)
test <- setdiff(1:n, train)
```

```
## Classification error rate of LDA
g1 <- lda(default~., data=Default, subset=train)
pred1 <- predict(g1, Default)
table(pred1$class[test], Default$default[test])
mean(pred1$class[test]!=Default$default[test])
```

```
## Classification error rate of QDA
g2 <- qda(default~., data=Default, subset=train)
pred2 <- predict(g2, Default)
table(pred2$class[test], Default$default[test])
mean(pred2$class[test]!=Default$default[test])
```

```
## AUC comparison between LDA and QDA
library(ROCR)
label <- factor(default[test], levels=c("Yes","No"),
                labels=c("TRUE","FALSE"))
preds1 <- prediction(pred1$posterior[test,2], label)
preds2 <- prediction(pred2$posterior[test,2], label)
performance(preds1, "auc")@y.values
performance(preds2, "auc")@y.values
```

```
set.seed(123)
N <- 100
CER <- AUC <- matrix(NA, N, 2)
```

```
for (i in 1:N) {
    train <- sample(1:n, n*0.7)
    test <- setdiff(1:n, train)
    y.test <- Default$default[test]
```

```
    g1 <- lda(default~., data=Default, subset=train)
    g2 <- qda(default~., data=Default, subset=train)
    pred1 <- predict(g1, Default)
    pred2 <- predict(g2, Default)
    CER[i,1] <- mean(pred1$class[test]!=y.test)
    CER[i,2] <- mean(pred2$class[test]!=y.test)

    label <- factor(default[test], levels=c("Yes","No"),
                    labels=c("TRUE","FALSE"))
    preds1 <- prediction(pred1$posterior[test,2], label)
    preds2 <- prediction(pred2$posterior[test,2], label)
    AUC[i,1] <- as.numeric(performance(preds1, "auc")@y.values)
    AUC[i,2] <- as.numeric(performance(preds2, "auc")@y.values)
}
```

```
apply(CER, 2, mean)
apply(AUC, 2, mean)
```

Naive Bayes Method

- Assumes that features are independent in each class.
- Useful when p is large, and so multivariate methods like QDA and even LDA break down.
- Gaussian naive Bayes assumes each Σ_k is diagonal:

$$\delta_k(x) \propto \log\left[\pi_k \prod_{j=1}^{p} f_{kj}(x_j)\right] = -\frac{1}{2}\sum_{j=1}^{p} \frac{(x_j - \mu_{kj})^2}{\sigma_{kj}^2} + \log \pi_k$$

- It can use for mixed features (qualitative and quantitative). If X_j is qualitative, replace $f_{kj}(x_j)$ with probability mass function over discrete categories.
- Despite strong assumptions, naive Bayes often produces good classification results.

```
data(iris)
library(e1071)
```

```
g1 <- naiveBayes(Species ~ ., data = iris)
g1 <- naiveBayes(iris[,-5], iris[,5])
pred <- predict(g1, iris[,-5])
table(pred, iris[,5])
mean(pred!=iris$Species)
```

```
## Randomly separate training sets and test sets
set.seed(1234)
tran <- sample(nrow(iris), size=floor(nrow(iris)*2/3))
```

```
## Compute misclassification error for test sets
g2 <- naiveBayes(Species ~ ., data=iris, subset=tran)
pred2 <- predict(g2, iris)[-tran]
test <- iris$Species[-tran]
table(pred2, test)
mean(pred2!=test)
```

```
data(Default)
```

```
set.seed(1234)
n <- nrow(Default)
train <- sample(1:n, n*0.7)
test <- setdiff(1:n, train)
```

```
g3 <- naiveBayes(default ~ ., data=Default, subset=train)
pred3 <- predict(g3, Default)[test]
table(pred3, Default$default[test])
mean(pred3!=Default$default[test])
```

```
## AUC of Naive Bayes
library(ROCR)
label <- factor(default[test], levels=c("Yes","No"),
                labels=c("TRUE","FALSE"))
pred4 <- predict(g3, Default, type="raw")
preds <- prediction(pred4[test, 2], label)
performance(preds, "auc")@y.values
```

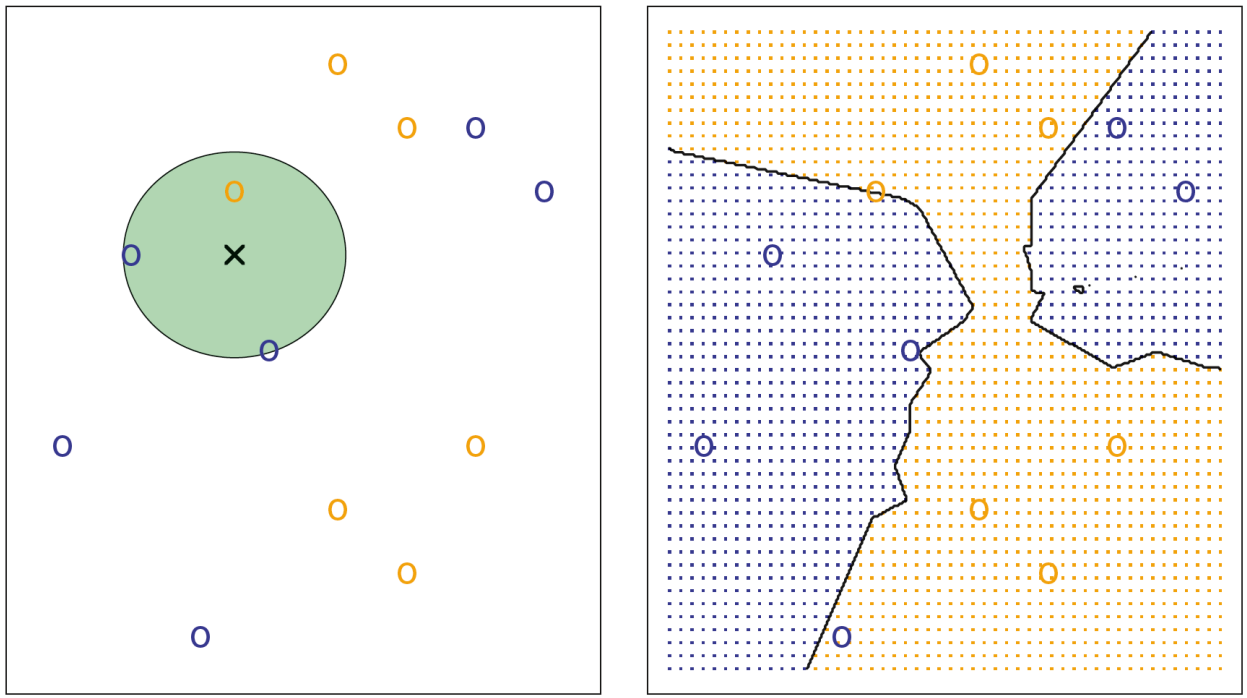
K-Nearest Neighbors

- In theory, to predict qualitative responses using the Bayes classifier is the best.
- For real data, the conditional distribution of Y given X is unknown, so computing the Bayes classifier is impossible.
- KNN (K-nearest neighbors) classifier estimates the conditional distribution of Y given X

$$Pr(Y = j|X = x_0) = \frac{1}{K} \sum_{i \in \mathcal{N}_0} I(y_i = j)$$

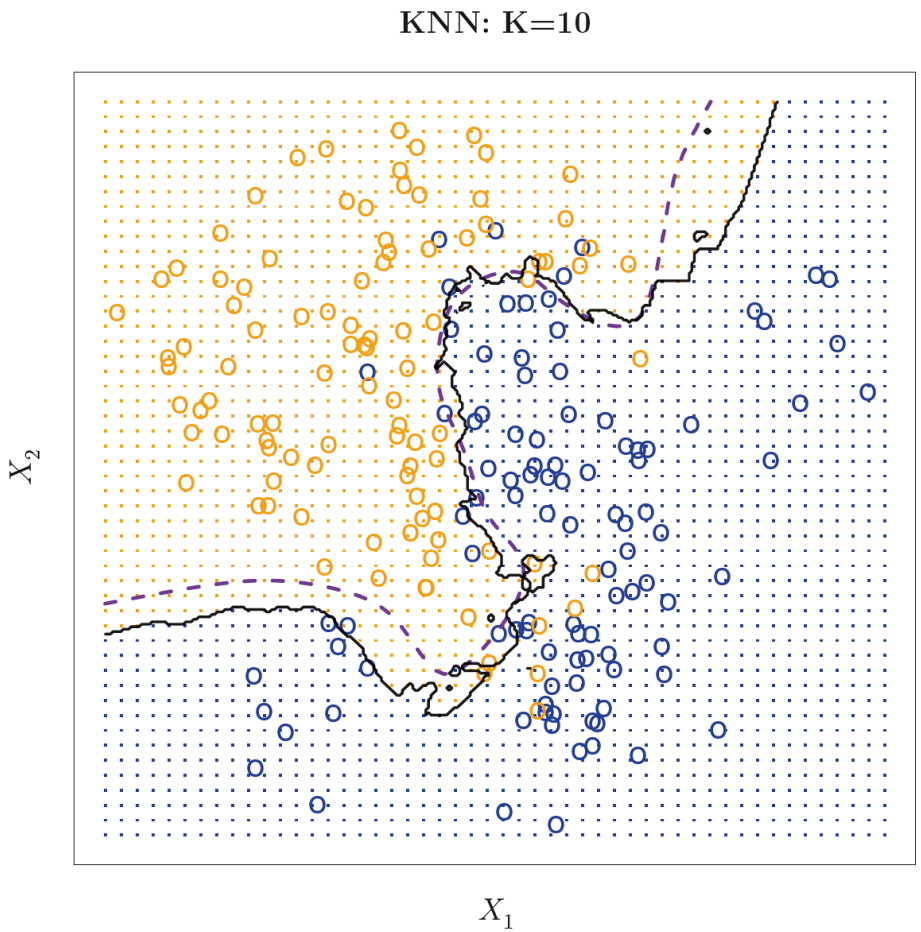
  where x_0 a test observation and $\mathcal{N}_0$ is a set of K points in the training data that are closest to x_0.
- KNN applies Bayes rule and classifies the test observation x_0 to the class with the largest probability.

KNN method



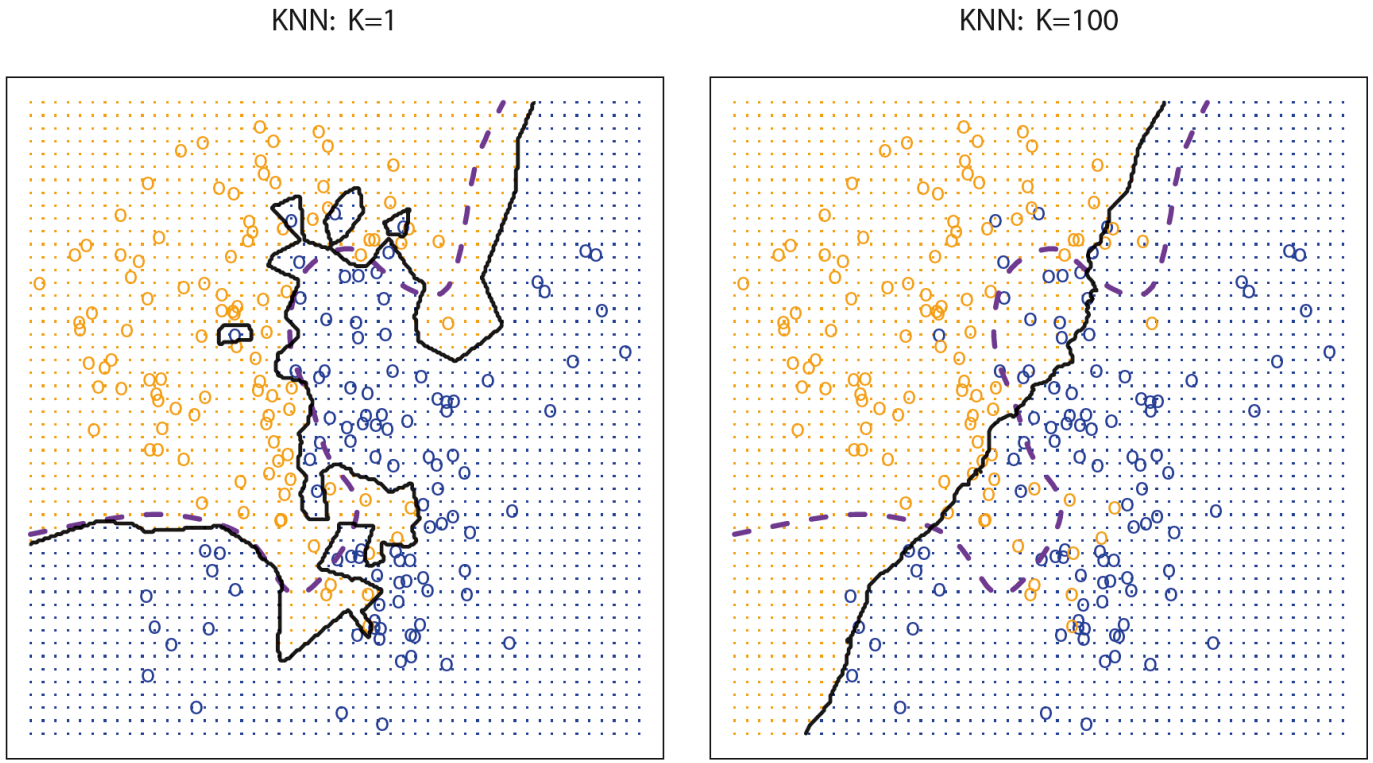
- The KNN approach, using $K = 3$.
- In the left: a test observation at which a predicted class label is desired is shown as a black cross.
- In the right: the KNN decision boundary for this example is shown in black.

KNN method



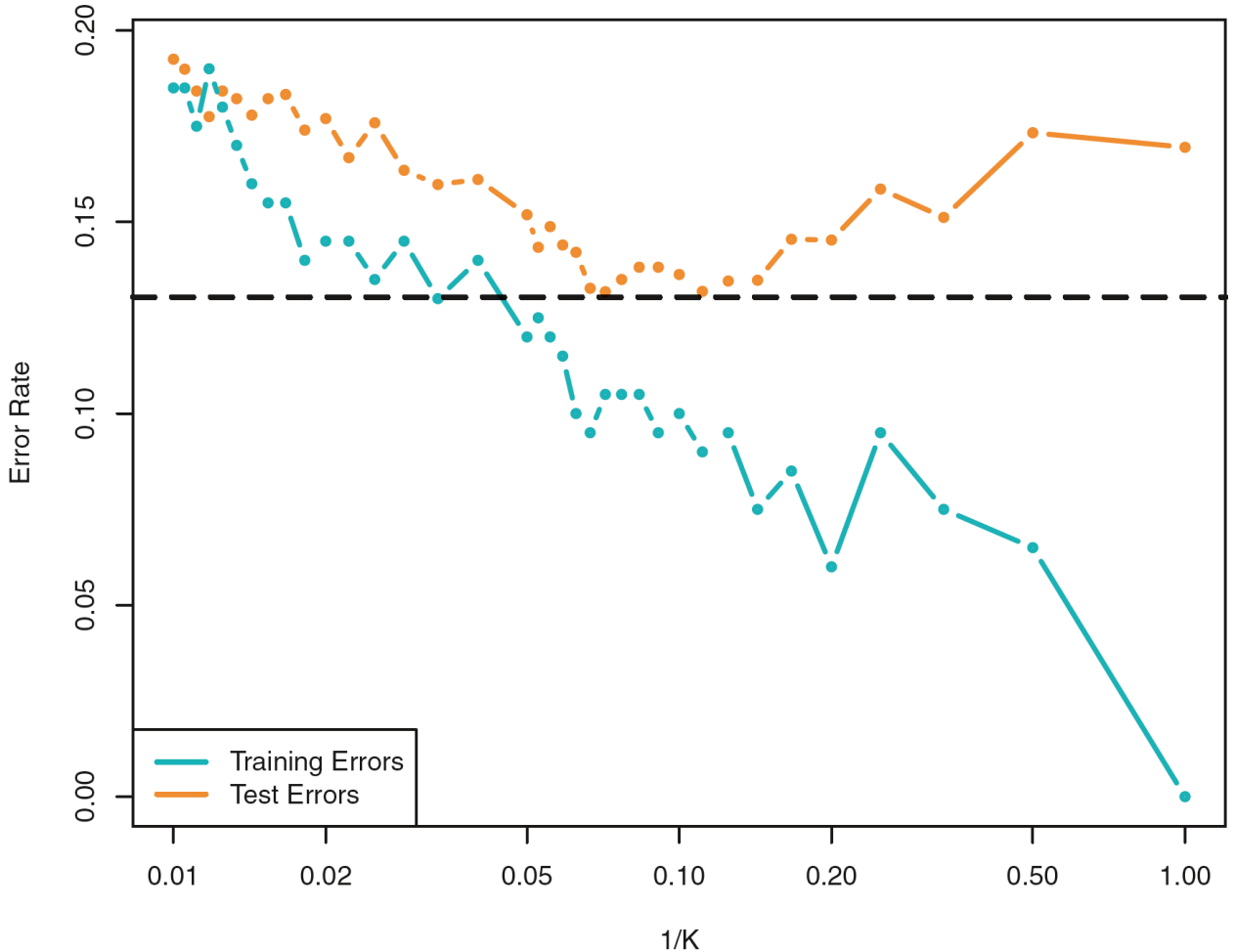
- KNN decision boundary (black line) and The Bayes decision boundary (purple dashed line)
- The choice of K has a drastic effect on the KNN classifier obtained.

KNN method



- When $K = 1$, the decision boundary is overly flexible (low bias but high variance).
- When $K = 100$, the decision boundary is close to linear so less flexible (low variance but high bias).

KNN method



- The KNN training error rate vs. test error rate

Caravan Insurance Data

```
library(ISLR)
data(Caravan)
dim(Caravan)
str(Caravan)
attach(Caravan)
```

```
# only 6% of people purchased caravan insurance.
summary(Purchase)
mean(Purchase=="Yes")
```

```
## Logistic regression
g0 <- glm(Purchase~., data=Caravan, family="binomial")
summary(g0)
```

```
library(glmnet)
y <- Purchase
x <- as.matrix(Caravan[,-86])
```

```
g1 <- glmnet(x, y, alpha=1, family="binomial")
plot(g1, "lambda")
```

```
set.seed(123)
g1.cv <- cv.glmnet(x, y, alpha=1, family="binomial")
plot(g1.cv)
```

```
g1.cv$lambda.min
g1.cv$lambda.1se
```

```
coef1 <- coef(g1.cv, s="lambda.min")
coef2 <- coef(g1.cv, s="lambda.1se")
cbind(coef1, coef2)
```

```
sum(coef1!=0)-1
sum(coef2!=0)-1
```

```
Car1 <- Caravan[ ,which(coef1[-1]!=0)]
Car2 <- Caravan[ ,which(coef2[-1]!=0)]
```

```
g1 <- glm(Purchase~., data=Car1, family="binomial")
g2 <- glm(Purchase~., data=Car2, family="binomial")
summary(g1)
summary(g2)
```

```
## Standardize data so that mean=0 and variance=1.
X <- scale(Caravan[,-86])
apply(Caravan[,1:5], 2, var)
apply(X[,1:5], 2, var)
```

```
## Separate training sets and test sets
test <- 1:1000
train.X <- X[-test, ]
test.X <- X[test, ]
train.Y <- Purchase[-test]
test.Y <- Purchase[test]
```

```
library(class)
```

```
## Classification error rate of KNN
set.seed(1)
knn.pred <- knn(train.X, test.X, train.Y, k=1)
mean(test.Y!=knn.pred)
mean(test.Y!="No")
table(knn.pred, test.Y)
```

```
knn.pred=knn(train.X, test.X, train.Y, k=3)
table(knn.pred, test.Y)
mean(test.Y!=knn.pred)
```

```
knn.pred=knn(train.X, test.X, train.Y, k=5)
table(knn.pred, test.Y)
mean(test.Y!=knn.pred)
```

```
knn.pred=knn(train.X, test.X, train.Y, k=10)
table(knn.pred, test.Y)
mean(test.Y!=knn.pred)
```

KNN: Simulation Data

```
library(mnormt)
set.seed(1010)
```

```
sigma <- matrix(c(1, 0.5, 0.5, 1), 2, 2)
x.tran1 <- rmnorm(100, c(0, 0.8), sigma)
x.tran2 <- rmnorm(100, c(0.8, 0), sigma)
x.test1 <- rmnorm(3430, c(0, 0.8), sigma)
x.test2 <- rmnorm(3430, c(0.8 ,0), sigma)
```

```
x.tran <- rbind(x.tran1, x.tran2)
x.test <- rbind(x.test1, x.test2)
y.tran <- factor(rep(0:1, each=100))
```

```
mn <- min(x.tran)
mx <- max(x.tran)
px1 <- seq(mn, mx, length.out=70)
px2 <- seq(mn, mx, length.out=98)
gd <- expand.grid(x=px1, y=px2)
```

```
g1 <- knn(x.tran, gd, y.tran, k = 1, prob=TRUE)
g2 <- knn(x.tran, gd, y.tran, k = 10, prob=TRUE)
g3 <- knn(x.tran, gd, y.tran, k = 100, prob=TRUE)
```

```
par(mfrow=c(1,3))
prob1 <- attr(g1, "prob")
prob1 <- ifelse(g1=="1", prob1, 1-prob1)
pp1 <- matrix(prob1, length(px1), length(px2))
contour(px1, px2, pp1, levels=0.5, labels="", xlab="", ylab="",
        main="KNN: K=1", axes=FALSE)
points(x.tran, col=ifelse(y.tran==1, "cornflowerblue", "coral"))
co1 <- ifelse(pp1>0.5, "cornflowerblue", "coral")
points(gd, pch=".", cex=1.2, col=co1)
box()
```

```
prob2 <- attr(g2, "prob")
prob2 <- ifelse(g2=="1", prob2, 1-prob2)
pp2 <- matrix(prob2, length(px1), length(px2))
contour(px1, px2, pp2, levels=0.5, labels="", xlab="", ylab="",
        main="KNN: K=10", axes=FALSE)
points(x.tran, col=ifelse(y.tran==1, "cornflowerblue", "coral"))
co2 <- ifelse(pp2>0.5, "cornflowerblue", "coral")
points(gd, pch=".", cex=1.2, col=co2)
box()
```

```
prob3 <- attr(g3, "prob")
prob3 <- ifelse(g3=="1", prob3, 1-prob3)
pp3 <- matrix(prob3, length(px1), length(px2))
contour(px1, px2, pp3, levels=0.5, labels="", xlab="", ylab="",
        main="KNN: K=100", axes=FALSE)
points(x.tran, col=ifelse(y.tran==1, "cornflowerblue", "coral"))
co3 <- ifelse(pp3>0.5, "cornflowerblue", "coral")
points(gd, pch=".", cex=1.2, col=co3)
box()
```

Comparison of Classification Methods

- We compare classification error rate of 7 classification methods on 6 different scenarios.
- 7 classification methods
  - LDA and QDA
  - KNN(K=1), KNN(K=5), and KNN(K=20)
  - Logistic regression
  - Bayes Naive method
- 6 scenarios
  1. Gaussian model with a diagonal covariance
  2. Gaussian model with the same covariance
  3. t-distribution model
  4. Gaussian model with a different covariance
  5. Multiplicative model
  6. Complicated non-parametric model

```
library(mnormt);library(MASS)
library(class);library(e1071)
```

```
MissClassRate <- function(x.tran, x.test, y.test, y.tran) {
    nt <- nrow(x.tran)
    ldafit <- predict(lda(x.tran, y.tran), x.test)$class
    qdafit <- predict(qda(x.tran, y.tran), x.test)$class
    knn1 <- knn(x.tran, x.test, y.tran, k=1)
    knn5 <- knn(x.tran, x.test, y.tran, k=5)
    knn20 <- knn(x.tran, x.test, y.tran, k=20)
    data <- data.frame(x=rbind(x.tran,x.test),y=c(y.tran,y.test))
    g <- glm(y~., family="binomial", subset=1:nt, data)
    logit <- predict(g, data, type="response")[-c(1:nt)]
    logit[logit >= 0.5] <- 1
    logit[logit < 0.5] <- 0
    g2 <- naiveBayes(y~., subset=1:nt, data)
    NB <- predict(g2, data)[-c(1:nt)]
  c(mean(ldafit!=y.test), qda=mean(qdafit!=y.test),
    mean(knn1!=y.test), mean(knn5!=y.test), mean(knn20!=y.test),
    mean(logit!=y.test), mean(NB!=y.test))
}
```

Scenario 1

```
set.seed(12345)
K <- 100
RES1 <- matrix(NA, K, 7)
```

```
for (i in 1:K) {
    x.A <- rmnorm(150, rep(0, 2), diag(2))
    x.B <- rmnorm(150, rep(1, 2), diag(2))
    x.tran <- rbind(x.A[1:50, ], x.B[1:50, ])
    x.test <- rbind(x.A[-c(1:50), ], x.B[-c(1:50), ])
    y.tran <- factor(rep(0:1, each=50))
    y.test <- factor(rep(0:1, each=100))
    RES1[i,] <- MissClassRate(x.tran, x.test, y.test, y.tran)
}
```

```
boxplot(RES1, boxwex=0.5, col=2:8,names=c("LDA", "QDA", "KNN-1",
        "KNN-5", "KNN-20", "Logit", "NB"), main="Scenario 1",
        ylab="Test Error Rates")
```

Scenario 2

```
RES2 <- matrix(NA, K, 7)
```

```
for (i in 1:K) {
    x.A <- rmnorm(150, rep(0, 2), matrix(c(1,-0.5,-0.5,1),2))
    x.B <- rmnorm(150, rep(1, 2), matrix(c(1,-0.5,-0.5,1),2))
    x.tran <- rbind(x.A[1:50, ], x.B[1:50, ])
    x.test <- rbind(x.A[-c(1:50), ], x.B[-c(1:50), ])
    y.tran <- factor(rep(0:1, each=50))
    y.test <- factor(rep(0:1, each=100))
    RES2[i,] <- MissClassRate(x.tran, x.test, y.test, y.tran)
}
```

```
boxplot(RES2, boxwex=0.5, col=2:8,names=c("LDA", "QDA", "KNN-1",
        "KNN-5", "KNN-20", "Logit", "NB"), main="Scenario 2",
        ylab="Test Error Rates")
```

Scenario 3

```
RES3 <- matrix(NA, K, 7)
```

```
for (i in 1:K) {
    x.A <- cbind(rt(150, df=5, ncp=0), rt(150, df=5, ncp=0))
    x.B <- cbind(rt(150, df=5, ncp=0.5), rt(150, df=5, ncp=0.5))
    x.tran <- rbind(x.A[1:50, ], x.B[1:50, ])
    x.test <- rbind(x.A[-c(1:50), ], x.B[-c(1:50), ])
    y.tran <- factor(rep(0:1, each=50))
    y.test <- factor(rep(0:1, each=100))
    RES3[i,] <- MissClassRate(x.tran, x.test, y.test, y.tran)
}
```

```
boxplot(RES3, boxwex=0.5, col=2:8,names=c("LDA", "QDA", "KNN-1",
        "KNN-5", "KNN-20", "Logit", "NB"), main="Scenario 3",
        ylab="Test Error Rates")
```

Scenario 4

```
RES4 <- matrix(NA, K, 7)
```

```
for (i in 1:K) {
    x.A <- rmnorm(150, rep(0, 2), matrix(c(1,0.5,0.5,1),2))
    x.B <- rmnorm(150, rep(1, 2), matrix(c(1,-0.5,-0.5,1),2))
    x.tran <- rbind(x.A[1:50, ], x.B[1:50, ])
    x.test <- rbind(x.A[-c(1:50), ], x.B[-c(1:50), ])
    y.tran <- factor(rep(0:1, each=50))
    y.test <- factor(rep(0:1, each=100))
    RES4[i,] <- MissClassRate(x.tran, x.test, y.test, y.tran)
}
```

```
boxplot(RES4, boxwex=0.5, col=2:8,names=c("LDA", "QDA", "KNN-1",
        "KNN-5", "KNN-20", "Logit", "NB"), main="Scenario 4",
        ylab="Test Error Rates")
```

Scenario 5

```
RES5 <- matrix(NA, K, 7)
for (i in 1:K) {
    x.A <- rmnorm(150, rep(0, 2), diag(2))
    x.B <- rmnorm(150, rep(1, 2), diag(2))
    x.tran <- rbind(x.A[1:50, ], x.B[1:50, ])
    x.test <- rbind(x.A[-c(1:50), ], x.B[-c(1:50), ])
    tr.int <- x.tran[,1]*x.tran[,2]
    te.int <- x.test[,1]*x.test[,2]
    xb.tr <- cbind(x.tran,tr.int)%*%c(-0.5,0.5,1)
    xb.te <- cbind(x.test,te.int)%*%c(-0.5,0.5,1)
    y.tran <- rep(0, 100); y.test <- rep(0, 200)
    y.tran[xb.tr > 0] <- 1; y.tran <- factor(y.tran)
    y.test[xb.te > 0] <- 1; y.test <- factor(y.test)
    RES5[i,] <- MissClassRate(x.tran, x.test, y.test, y.tran)
}
```

```
boxplot(RES5, boxwex=0.5, col=2:8,names=c("LDA", "QDA", "KNN-1",
        "KNN-5", "KNN-20", "Logit", "NB"), main="Scenario 5",
        ylab="Test Error Rates")
```

Scenario 6

```
RES6 <- matrix(NA, K, 7)
for (i in 1:K) {
    x.A <- rmnorm(150, rep(0, 2), diag(2))
    x.B <- rmnorm(150, rep(1, 2), diag(2))
    x.tran <- rbind(x.A[1:50, ], x.B[1:50, ])
    x.test <- rbind(x.A[-c(1:50), ], x.B[-c(1:50), ])
    tr.int <- exp(x.tran[,1])/log(abs(x.tran[,2]))
    te.int <- exp(x.test[,1])/log(abs(x.test[,2]))
    xb.tr <- cbind(x.tran,tr.int)%*%c(-0.5,0.5,1)
    xb.te <- cbind(x.test,te.int)%*%c(-0.5,0.5,1)
    y.tran <- rep(0, 100); y.test <- rep(0, 200)
    y.tran[xb.tr > 0] <- 1; y.tran <- factor(y.tran)
    y.test[xb.te > 0] <- 1; y.test <- factor(y.test)
    RES6[i,] <- MissClassRate(x.tran, x.test, y.test, y.tran)
}
```

```
boxplot(RES6, boxwex=0.5, col=2:8,names=c("LDA", "QDA", "KNN-1",
        "KNN-5", "KNN-20", "Logit", "NB"), main="Scenario 5",
        ylab="Test Error Rates")
```

```
par(mfrow=c(2,3))
boxplot(RES1, boxwex=0.5, col=2:8, ylim=c(0,0.6),
        names=c("LDA", "QDA", "KNN-1", "KNN-5",
                "KNN-20", "Logit", "NB"),
        main="Scenario 1", ylab="Test Error Rates")
```

```
boxplot(RES2, boxwex=0.5, col=2:8, ylim=c(0,0.6),
        names=c("LDA", "QDA", "KNN-1", "KNN-5",
                "KNN-20", "Logit", "NB"),
        main="Scenario 2", ylab="Test Error Rates")
```

```
boxplot(RES3, boxwex=0.5, col=2:8, ylim=c(0,0.6),
        names=c("LDA", "QDA", "KNN-1", "KNN-5",
                "KNN-20", "Logit", "NB"),
        main="Scenario 3", ylab="Test Error Rates")
```

```
boxplot(RES4, boxwex=0.5, col=2:8, ylim=c(0,0.6),
        names=c("LDA", "QDA", "KNN-1", "KNN-5",
                "KNN-20", "Logit", "NB"),
        main="Scenario 4", ylab="Test Error Rates")
```

```
boxplot(RES5, boxwex=0.5, col=2:8, ylim=c(0,0.6),
        names=c("LDA", "QDA", "KNN-1", "KNN-5",
                "KNN-20", "Logit", "NB"),
        main="Scenario 5", ylab="Test Error Rates")
```

```
boxplot(RES6, boxwex=0.5, col=2:8, ylim=c(0,0.6),
        names=c("LDA", "QDA", "KNN-1", "KNN-5",
                "KNN-20", "Logit", "NB"),
        main="Scenario 6", ylab="Test Error Rates")
```

Stock Market Data

- `Smarket` data set consists of percentage returns for the S&P 500 stock index over 1∼250 days, from the beginning of 2001 until the end of 2005.
- For each date, we have recorded the percentage returns for each of the five previous trading days.
  - `Lag1`, `Lag2`, `Lag3`, `Lag4` and `Lag5`
- We have also recorded
  - `volume`: the number of shares traded on the previous day (in billions).
  - `Today`: the percentage return on the date in question.
  - `direction`: whether the market was Up or Down on this date.
- Our goal is to predict `direction` (a qualitative response) using the other features.

Stock Market Data

```
library(ISLR)
names(Smarket)
str(Smarket)
```

```
dim(Smarket)
summary(Smarket)
pairs(Smarket)
cor(Smarket[, -9])
```

```
attach(Smarket)
par(mfrow=c(2,4))
for (i in 1:8) {
    plot(Smarket[,i], pch=20, main=colnames(Smarket)[i],
         col=as.numeric(Smarket$Direction) + 1)
}
table(Year)
train <- (Year < 2005)
y.test <- Direction[!train]
Sdata <- Smarket[,-c(1,8)]
```

Stock Market Data

```
## Logistic Regression
g1 <- glm(Direction~., data=Sdata, family="binomial",
          subset=train)
p1 <- predict(g1, Sdata[!train,], type="response")
pred1 <- rep("Down", length(y.test))
pred1[p1 > 0.5] <- "Up"
mean(pred1!=y.test)
```

```
## LDA
library(MASS)
g2 <- lda(Direction~., data=Sdata, subset=train)
pred2 <- predict(g2, Sdata[!train,])$class
mean(pred2!=y.test)
```

```
## QDA
g3 <- qda(Direction~., data=Sdata, subset=train)
pred3 <- predict(g3, Sdata[!train,])$class
mean(pred3!=y.test)
```

Stock Market Data

```
## Naive Bayes
library(e1071)
g4 <- naiveBayes(Direction~., data=Sdata, subset=train)
pred4 <- predict(g4, Sdata[!train,])
mean(pred4!=y.test)
```

```
## KNN
library(class)
x.train <- Sdata[train, -7]
x.test <- Sdata[!train, -7]
y.train <- Sdata$Direction[train]
```

```
CER <- NULL
for (k in 1:200) {
    g5 <- knn(x.train, x.test, y.train, k=k)
    CER[k] <- mean(g5!=y.test)
}
summary(CER)
plot(1:200, CER, type="b", xlab="k", ylab="Error", col=2, pch=20)
```