Statistical Learning

Lecture 03a

ANU - RSFAS

Last Updated: Tue Mar 15 12:13:00 2022

Continuation of Classification - Back to the Default Data

```
library(ISLR)
head(Default)
```

```
##   default student   balance    income
## 1      No      No  729.5265 44361.625
## 2      No     Yes  817.1804 12106.135
## 3      No      No 1073.5492 31767.139
## 4      No      No  529.2506 35704.494
## 5      No      No  785.6559 38463.496
## 6      No     Yes  919.5885  7491.559
```

```r
library(MASS)
mod <- lda(default ~ balance + income + student,
           data=Default)
mod
```

```
## Call:
## lda(default ~ balance + income + student, data = Default)
##
## Prior probabilities of groups:
##     No    Yes
## 0.9667 0.0333
##
## Group means:
##       balance   income studentYes
## No   803.9438 33566.17  0.2914037
## Yes 1747.8217 32089.15  0.3813814
##
## Coefficients of linear discriminants:
##                      LD1
## balance     2.243541e-03
## income      3.367310e-06
## studentYes -1.746631e-01
```

Let's Examine the "Confusion" Matrix

```r
pred.train <- predict(mod)$class
y.obs <- Default$default

table(pred.train, y.obs)
```

```
##           y.obs
## pred.train   No  Yes
##        No  9645  254
##        Yes   22   79
```

- $(22 + 254)/10000$ errors - 2.76% misclassification rate!

Some Caveats

- This is training error, and we may be overfitting. Not a huge concern here since $n = 10000$ and $p = 3$! (n is large compared to p).
- The prior probability of **default = Yes** is 3.33%, while for **No** it is 96.67%. If we classify according to the mode of the categories we should predict that no one will default. In this case only 3.33% misclassification rate.
- We can break down the errors. Of the observed No's, we make $22/(22{+}9645) = 0.2\%$ errors; of the observed Yes's, we make $254/(254{+}79) = 76.3\%$ errors!

Types of Errors

- False positive rate: The fraction of negative examples that are classified as positive — 0.2% in example.
- False negative rate: The fraction of positive examples that are classified as negative — 75.7% in example.
- For the previous table, it was produced by classifying to class **Yes** if

$$\hat{P}(Default = Yes | Balance, Student, Income) \geq 0.5$$

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

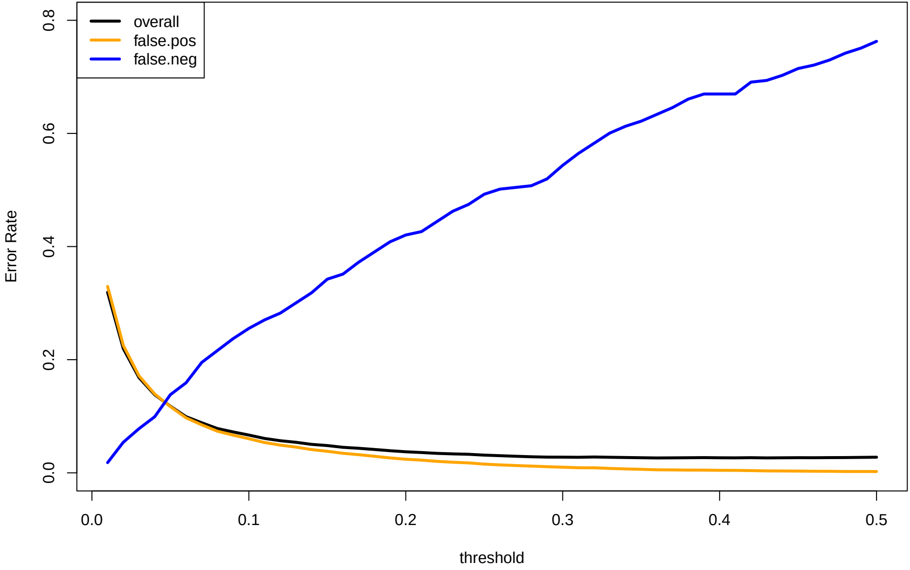
$$\hat{P}(Default = Yes | Balance, Student, Income) \geq \mathrm{threshold}$$

And we can vary the threshold.

Varying the Threshold

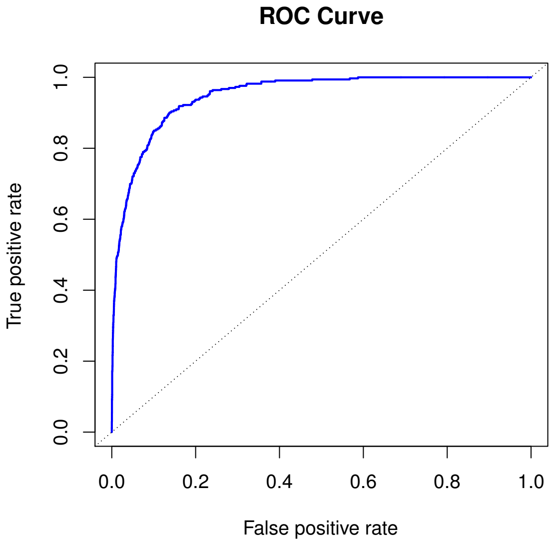
```r
threshold <- seq(0.01, 0.5, by=0.01)
n.t <- length(threshold)

overall <- rep(0, n.t)
false.pos <- rep(0, n.t)
false.neg <- rep(0, n.t)

## second column is the probability for Yes
pred.train <- predict(mod)$posterior[,2]

for(i in 1:n.t){

  pred.i <- rep("No", length(y.obs))
  pred.i[pred.train >= threshold[i]] <- "Yes"

  tab <- table(pred.i, y.obs)
  overall[i] <- (tab[1,2]+tab[2,1])/sum(tab)
  false.pos[i] <- tab[2,1]/(tab[2,1]+tab[1,1])
  false.neg[i] <- tab[1,2]/(tab[1,2]+tab[2,2])
}
```

```r
plot(threshold, overall, lwd=3, type="l", ylim=c(0, 0.8), ylab="Error Rate")
lines(threshold, false.pos, col="orange", lwd=3)
lines(threshold, false.neg, col="blue", lwd=3)
legend("topleft", legend=c("overall", "false.pos", "false.neg"), col=c("black", "orange", "blue"),
       lty=c(1,1,1), lwd=3 )
```



- In order to reduce the false negative rate, we may want to reduce the threshold to 0.1 or less.

Receiver Operating Curve (ROC)



- "The curious name stems from the method's origins in radio signal detection during WW II.' ' - Walter Piegorsch - *Statistical Data Analytics*.
- The ROC plot displays both **False Positive Rate** and the **True Positive Rate = 1 - False Negative Rate**.
- Sometimes we use the AUC (area under the curve) to summarize the overall performance.
- Higher AUC is good.
- An R package for constructing ROC curves is `pROC` (this plots the **Specificity = 1-False Positive Rate** against the **Sensitivity = True Positive Rate**).

Discriminant Analysis

- 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 $Pr(Y|X)$.
- 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.

Bayes Theorem for Classification

- Thomas Bayes was a famous mathematician whose name represents a big field of statistical and probabilistic modeling. Here we focus on a simple result, known as Bayes theorem:

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

- In terms of discriminant analysis we have:

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

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 have this issue.
- 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

Recall, the Gaussian density has the form:

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

- Let's first assume $\sigma_k^2 = \sigma^2 \;\forall\; k$.
- Plugging this into Bayes formula:

$$Pr(Y = k|X = x) = p_k(x) = \frac{\pi_k \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{ \frac{-1}{2\sigma^2} (x - \mu_k)^2 \right\}}{\sum_{\ell=1}^{K} \pi_\ell \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{ \frac{-1}{2\sigma^2} (x - \mu_\ell)^2 \right\}}$$

Discriminant Functions

- To classify at the value $X = x$, we need to see which of the $Pr(Y = k|X = x)$ is largest. Based on the numerator, 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:

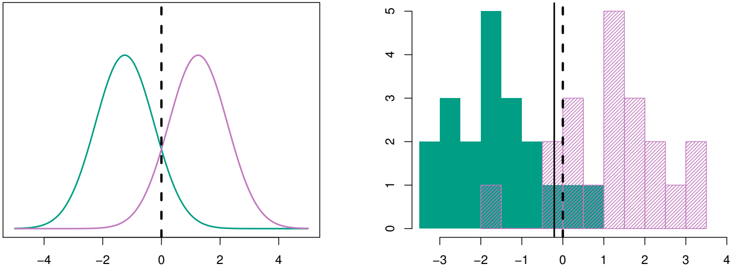
$$\delta_k(x) = x\frac{\mu_k}{\sigma^2} - \frac{\mu_k^2}{2\sigma^2} + log(\pi_k)$$

- We can see that $\delta_k(x)$ is a linear function of x.

- If $K = 2$ and $\pi_1 = \pi_2$ and we examine the decision boundary by setting $\delta_1(x) = \delta_2(x)$ and solving for x we find:

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

Example

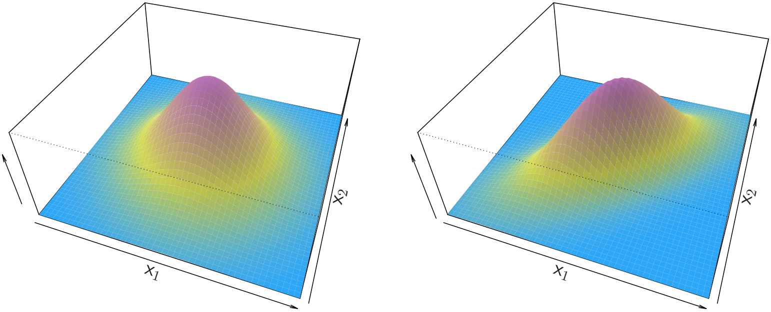


- If $p = 1$, $K = 2$, $\pi_1 = \pi_2 = 0.5$, $\mu_1 = -1.5$, $\mu_2 = 1.5$, and $\sigma^2 = 1$.
- Typically we don't know these parameters; we just have the training data. In that case we simply estimate the parameters and plug them into the rule.

Estimating the Parameters

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

Linear Discrimanent Analysis When $p > 1$



$$f_k(x) = \frac{1}{(2\pi)^{p/2}} |\Sigma|^{-1/2} \exp\left\{ -\frac{1}{2} (x - \mu_k)^t \, \Sigma^{-1} (x - \mu_k) \right\}$$

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

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

```r
mod <- lda(default ~ balance + income,
           data=Default)
mod
```

```
## Call:
## lda(default ~ balance + income, data = Default)
##
## Prior probabilities of groups:
##     No    Yes
## 0.9667 0.0333
##
## Group means:
##       balance   income
## No   803.9438 33566.17
## Yes 1747.8217 32089.15
##
## Coefficients of linear discriminants:
##                  LD1
## balance 2.230835e-03
## income  7.793355e-06
```

```r
X <- Default[, 3:4]
head(X)
```

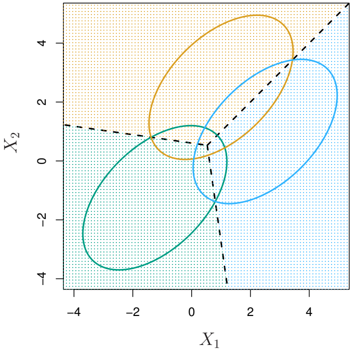
```
##     balance    income
## 1  729.5265 44361.625
## 2  817.1804 12106.135
## 3 1073.5492 31767.139
## 4  529.2506 35704.494
## 5  785.6559 38463.496
## 6  919.5885  7491.559
```

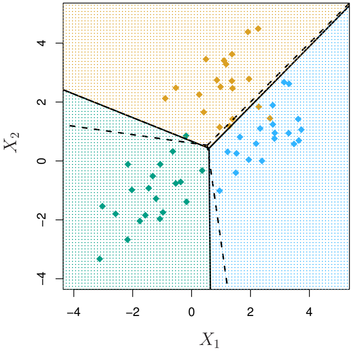
```r
g.1 <- X[Default$default == "No",]
g.2 <- X[Default$default == "Yes",]

cov.1 <- cov(g.1)
cov.2 <- cov(g.2)

n.1 <- nrow(g.1)
n.2 <- nrow(g.2)
```

```r
n <- n.1+n.2
K <- 2

Sigma.hat <- 1/(n - K) * (cov.1*(n.1-1) + cov.2*(n.2-1))
mu.1.hat <- apply(g.1, 2, mean)
mu.2.hat <- apply(g.2, 2, mean)

mu.hat.diff <- as.matrix(mu.2.hat -  mu.1.hat)
mu.hat.sum <- as.matrix(mu.2.hat +  mu.1.hat)

coeff.vec <-  solve(Sigma.hat) %*% mu.hat.diff
scale <-  as.numeric(sqrt(t(mu.hat.diff)%*%
                             solve(Sigma.hat)%*%mu.hat.diff))
coeff.vec/scale
```

```
##                 [,1]
## balance 2.230835e-03
## income  7.793355e-06
```

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



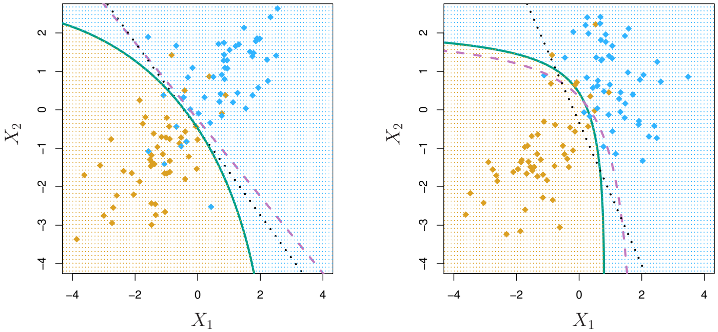


Other forms of Discriminant Analysis

$$P(Y = k|X) = \frac{\pi_k f_k(x)}{\sum_\ell \pi_\ell f_\ell(x)}$$

- When $f_k(x)$ are Gaussian densities, with the same covariance matrix $\mathbf{\Sigma}$ in each class, this leads to linear discriminant analysis. By altering the forms for $f_k(x)$, we get different classifiers.
- With Gaussians but different $\mathbf{\Sigma}_k$ in each class, we get quadratic discriminant analysis.
- $f_k(x) = \prod_j^p f_{kj}(x_j)$ (conditional independence model) in each class we get naive Bayes. For Gaussian this means the $\mathbf{\Sigma}_k$ are diagonal.
- Many other forms, by proposing specific density models for $f_k(x)$, including nonparametric approaches.

Quadratic Discriminant Analysis



$$\delta_k(x) = -\frac{1}{2}\log|\Sigma_k| - \frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_k)^t \boldsymbol{\Sigma}_k^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_k) + \log(\pi_k)$$

- Because the $\boldsymbol{\Sigma}_k$ are different, the quadratic terms matter.

Naive Bayes

- Assumes 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 $\boldsymbol{\Sigma}_k$ is diagonal:

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

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

Logistic Regression versus LDA

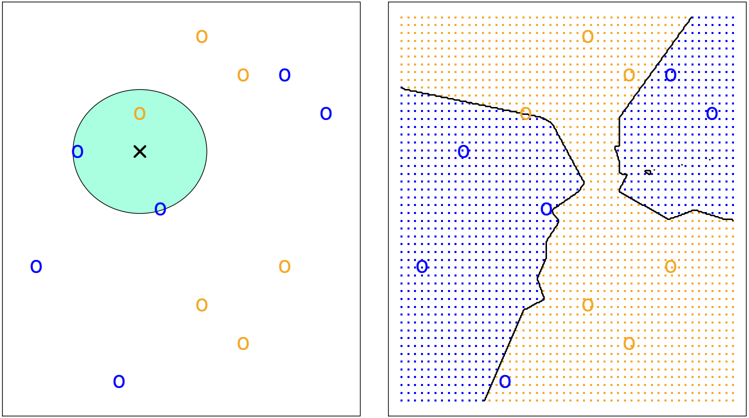
- For a two-class problem, one can show that for LDA:

$$\log\left(\frac{p_1(x)}{1-p_1(x)}\right) = \log\left(\frac{p_1(x)}{p_2(x)}\right) = c_0 + c_1 x_1 + \cdots + c_p x_p$$

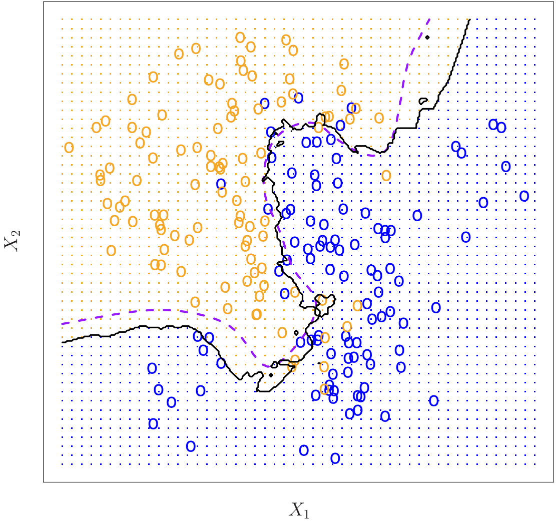
- So it has the same form as logistic regression.
- The difference is in how the parameters are estimated.
- Logistic regression uses the conditional likelihood based on $P(Y|X)$ (known as discriminative learning in computer science) – X is actually fixed or we assume we use just part of the conditional distribution.
- LDA uses the full likelihood based on $P(X, Y)$ (known as generative learning). We us the full joint distribution.
- Despite these differences, in practice the results are often very similar.

Nearest Neighbor *KNN* - Back to Basics

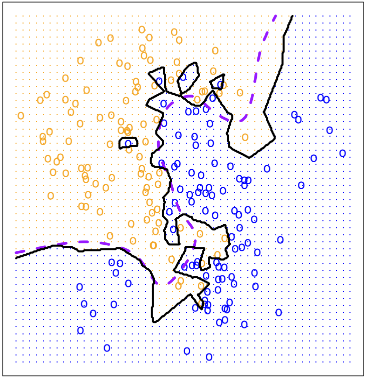
- A simpler approach - pick a test point x, and classify it based on its nearest neighbors k.



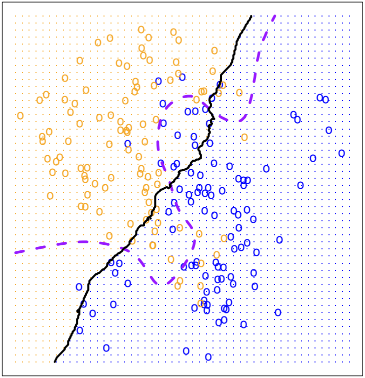
KNN: K=10



KNN: K=1



KNN: K=100



Stock Market Data

- We will examine daily **stock market data** for the **S&P 500 stock index** from 2001 - 2005 (1,250 days).
- We have the following variables:
    - Year
    - Lag1, Lag2, Lag3, Lag4, Lag5: **percent return for each of the last 5 days**
    - 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 today**

- Let's load in the data.

```
library(ISLR)
data(Smarket)
n <- nrow(Smarket)
head(Smarket)
```

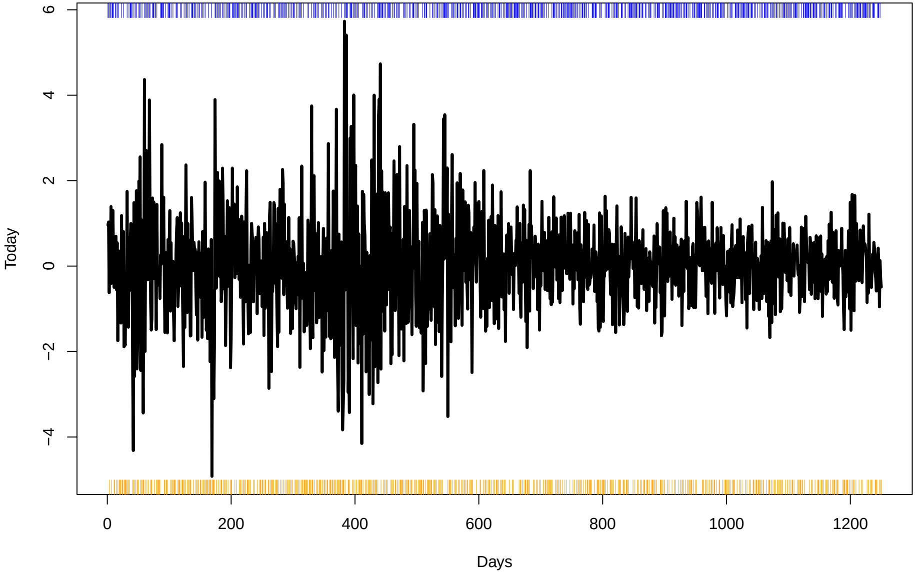
```
##   Year   Lag1   Lag2   Lag3   Lag4   Lag5 Volume  Today Direction
## 1 2001  0.381 -0.192 -2.624 -1.055  5.010 1.1913  0.959        Up
## 2 2001  0.959  0.381 -0.192 -2.624 -1.055 1.2965  1.032        Up
## 3 2001  1.032  0.959  0.381 -0.192 -2.624 1.4112 -0.623      Down
## 4 2001 -0.623  1.032  0.959  0.381 -0.192 1.2760  0.614        Up
## 5 2001  0.614 -0.623  1.032  0.959  0.381 1.2057  0.213        Up
## 6 2001  0.213  0.614 -0.623  1.032  0.959 1.3491  1.392        Up
```

- I will **attach** the data so we can call variables directly.

```
attach(Smarket)
```

- Let's first consider a trace plot for the variable **Today**. I also added rug plots for **Down (orange)** and **Up (blue)**.

```
Days <- 1:n
plot(Days, Today, type="l", lwd=3, xlab="Days")
rug(Days[Direction=="Down"], col="orange")
rug(Days[Direction=="Up"], col="blue", side=3)
```



Today
Days
6
4
2
0
−2
−4
0
200
400
600
800
1000
1200

- Let's examine the correlation among the continuous variables. None of the correlations are large, except for **Year** and **Volume**.

```
cor(Smarket[,1:8])
```

```
##              Year         Lag1         Lag2         Lag3         Lag4
## Year   1.00000000  0.029699649  0.030596422  0.033194581  0.035688718
## Lag1   0.02969965  1.000000000 -0.026294328 -0.010803402 -0.002985911
## Lag2   0.03059642 -0.026294328  1.000000000 -0.025896670 -0.010853533
## Lag3   0.03319458 -0.010803402 -0.025896670  1.000000000 -0.024051036
## Lag4   0.03568872 -0.002985911 -0.010853533 -0.024051036  1.000000000
## Lag5   0.02978799 -0.005674606 -0.003557949 -0.018808338 -0.027083641
## Volume 0.53900647  0.040909908 -0.043383215 -0.041823686 -0.048414246
## Today  0.03009523 -0.026155045 -0.010250033 -0.002447647 -0.006899527
##                Lag5      Volume        Today
## Year    0.029787995  0.53900647  0.030095229
## Lag1   -0.005674606  0.04090991 -0.026155045
## Lag2   -0.003557949 -0.04338321 -0.010250033
## Lag3   -0.018808338 -0.04182369 -0.002447647
## Lag4   -0.027083641 -0.04841425 -0.006899527
## Lag5    1.000000000 -0.02200231 -0.034860083
## Volume -0.022002315  1.00000000  0.014591823
## Today  -0.034860083  0.01459182  1.000000000
```

Logistic Regression

- Let's fit a logistic regression model to predict **Direction** (Down $Y = 0$, Up $Y = 1$) given the covariates.

```
glm.fit <- glm(Direction ~ Lag1 + Lag2 + Lag3 + Lag4 +
                   Lag5 + Volume, family = binomial)
summary(glm.fit)
```

```
##
## Call:
## glm(formula = Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 +
##     Volume, family = binomial)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.446  -1.203   1.065   1.145   1.326
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.126000   0.240736  -0.523    0.601
## Lag1        -0.073074   0.050167  -1.457    0.145
## Lag2        -0.042301   0.050086  -0.845    0.398
## Lag3         0.011085   0.049939   0.222    0.824
## Lag4         0.009359   0.049974   0.187    0.851
## Lag5         0.010313   0.049511   0.208    0.835
## Volume       0.135441   0.158360   0.855    0.392
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1731.2  on 1249  degrees of freedom
## Residual deviance: 1727.6  on 1243  degrees of freedom
## AIC: 1741.6
```

Let's look at the **Prediction matrix/Confusion matrix** for the **Training data**.

```
glm.probs <- predict(glm.fit, type = "response")
glm.probs[1:5]
```

```
##         1         2         3         4         5
## 0.5070841 0.4814679 0.4811388 0.5152224 0.5107812
```

- Let's classify our prediction $Y^* = 1$ if $P(Y = 1|X) > 0.5$, and $Y^* = 0$ otherwise.

```
glm.pred <- rep ("Down", n)
glm.pred[glm.probs > 0.5]="Up"
glm.pred[1:5]
```

```
## [1] "Up"   "Down" "Down" "Up"   "Up"
```

```r
tab <- table(glm.pred, Direction)
tab
```

```
##          Direction
## glm.pred Down  Up
##     Down  145 141
##     Up    457 507
```

- From the table we can see that the *Training Error Rate* is 47.84%.

```r
round((tab[1,2]+tab[2,1])/sum(tab)*100,2)
```

```
## [1] 47.84
```

Testing data

- What about the *Testing Error Rate*? Let's fit the model without using the 2005 data and try to predict that!

```
train <- (Year < 2005)
Smarket.2005 <-  Smarket[!train,]
Direction.2005 <- Direction[!train]
```

- Let's fit the model to the training data.

```
glm.fit <- glm(Direction ~ Lag1 + Lag2 + Lag3 + Lag4 +
                 Lag5 + Volume, data=Smarket,
               family=binomial, subset=train)
summary(glm.fit)
```

```
##
## Call:
## glm(formula = Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 +
##     Volume, family = binomial, data = Smarket, subset = train)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.302  -1.190   1.079   1.160   1.350
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.191213   0.333690   0.573    0.567
## Lag1        -0.054178   0.051785  -1.046    0.295
## Lag2        -0.045805   0.051797  -0.884    0.377
## Lag3         0.007200   0.051644   0.139    0.889
## Lag4         0.006441   0.051706   0.125    0.901
## Lag5        -0.004223   0.051138  -0.083    0.934
## Volume      -0.116257   0.239618  -0.485    0.628
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1383.3  on 997  degrees of freedom
## Residual deviance: 1381.1  on 991  degrees of freedom
## AIC: 1395.1
```

```r
glm.probs <- predict (glm.fit, Smarket.2005,
                      type= "response")

glm.pred <- rep("Down", 252)
glm.pred[glm.probs > 0.5] <- "Up"
tab <- table(glm.pred, Direction.2005)
tab
```

```
##          Direction.2005
## glm.pred Down Up
##     Down   77 97
##     Up     34 44
```

- So the *Test Error Rate* is 51.98%. A bit worse than random guessing!

- Let's try to remove some of the "noise" by dropping the weaker predictors. We will just use the first two lags.

```
glm.fit <- glm(Direction ~ Lag1 + Lag2,
               data=Smarket, family=binomial, subset=train)
summary(glm.fit)
```

```
##
## Call:
## glm(formula = Direction ~ Lag1 + Lag2, family = binomial, data = Smarket
##     subset = train)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.345  -1.188   1.074   1.164   1.326
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.03222    0.06338   0.508    0.611
## Lag1        -0.05562    0.05171  -1.076    0.282
## Lag2        -0.04449    0.05166  -0.861    0.389
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1383.3  on 997  degrees of freedom
## Residual deviance: 1381.4  on 995  degrees of freedom
## AIC: 1387.4
##
## Number of Fisher Scoring iterations: 3
```

```r
glm.probs <- predict (glm.fit, Smarket.2005, type= "response")

glm.pred <- rep("Down", 252)
glm.pred[glm.probs > 0.5] <- "Up"
table(glm.pred, Direction.2005)
```

```
##          Direction.2005
## glm.pred Down  Up
##     Down   35  35
##     Up     76 106
```

```r
round(mean(glm.pred!= Direction.2005)*100,2)
```

```
## [1] 44.05
```

- Hmmmmmm . . . We got the training error rate down to 44%!

- What if we just assumed the **market will increase every day?**

```
naive.pred <- rep("Up", 252)
tab <- table(naive.pred, Direction.2005)
tab
```

```
##            Direction.2005
## naive.pred Down  Up
##         Up  111 141
```

```
round((tab[1,1])/sum(tab)*100,2)
```

```
## [1] 44.05
```

- No real work needed. A larger study should be used before you try to bet on the market.

Linear Discriminant Analysis

```
library(MASS)
lda.fit <- lda(Direction ~ Lag1 + Lag2,
               data=Smarket, subset=train)
lda.fit
```

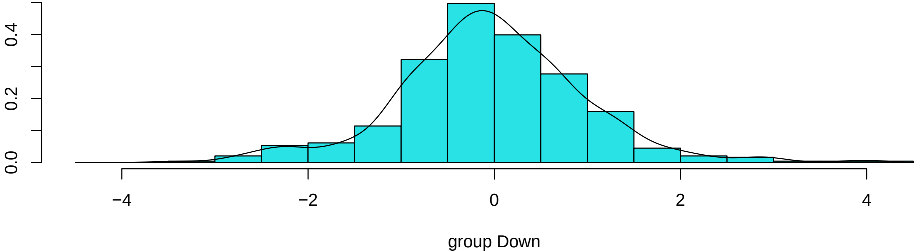
```
## Call:
## lda(Direction ~ Lag1 + Lag2, data = Smarket, subset = train)
##
## Prior probabilities of groups:
##     Down       Up
## 0.491984 0.508016
##
## Group means:
##             Lag1        Lag2
## Down  0.04279022  0.03389409
## Up   -0.03954635 -0.03132544
##
## Coefficients of linear discriminants:
##             LD1
## Lag1 -0.6420190
## Lag2 -0.5135293
```

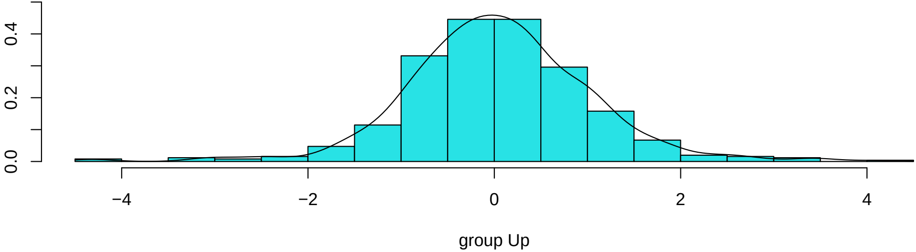
- The coefficients are the multipliers for each of the covariates.

$$-0.6420190 \times Lag1 - 0.5135293 \times Lag2$$

- The *plot()* function produces plots of the linear discriminants, obtained by computing $-0.6420190 \times Lag1 - 0.5135293 \times Lag2$ for each of the training observations.

```
plot(lda.fit, type="both")
```


0.0
0.2
0.4
−4
−2
0
2
4
group Down


0.0
0.2
0.4
−4
−2
0
2
4
group Up

```r
lda.pred <- predict(lda.fit, Smarket.2005)

##
head(lda.pred$class)
```

```
## [1] Up Up Up Up Up Up
## Levels: Down Up
```

```r
head(lda.pred$posterior)
```

```
##           Down        Up
## 999  0.4901792 0.5098208
## 1000 0.4792185 0.5207815
## 1001 0.4668185 0.5331815
## 1002 0.4740011 0.5259989
## 1003 0.4927877 0.5072123
## 1004 0.4938562 0.5061438
```

```r
head(lda.pred$x)
```

```
##              LD1
## 999   0.08293096
## 1000  0.59114102
## 1001  1.16723063
## 1002  0.83335022
## 1003 -0.03792892
## 1004 -0.08743142
```

- The **class** predictions are based on the **posterior probabilities** (which ever is highest). We have a test error rate of 44%. The same as logistic regression!

```
lda.class <- lda.pred$class
table(lda.class, Direction.2005)
```

```
##           Direction.2005
## lda.class Down  Up
##      Down   35  35
##      Up     76 106
```

```
round(mean(lda.class!= Direction.2005)*100, 2)
```

```
## [1] 44.05
```

- We can pick other cut-off values. Let's use 60%.

```
lda.class.60 <- rep("Up", 252)
lda.class.60[lda.pred$posterior[,1]< 0.60] <- "Down"
table(lda.class.60, Direction.2005)
```

```
##              Direction.2005
## lda.class.60 Down  Up
##         Down  111 141
```

```
round(mean(lda.class.60!= Direction.2005)*100, 2)
```

```
## [1] 55.95
```

- The results are not as good!

Quadratic Discriminant Analsysis

```
qda.fit <- qda(Direction ~ Lag1 + Lag2,
               data=Smarket, subset=train)
qda.fit
```

```
## Call:
## qda(Direction ~ Lag1 + Lag2, data = Smarket, subset = train)
##
## Prior probabilities of groups:
##     Down       Up
## 0.491984 0.508016
##
## Group means:
##             Lag1        Lag2
## Down  0.04279022  0.03389409
## Up   -0.03954635 -0.03132544
```

- The output does not contain the coefficients of the linear discriminants, because the QDA classifier involves a quadratic, rather than a linear, function of the predictors.

- Let's get the predictions.

```
qda.class  <- predict(qda.fit, Smarket.2005)$class
table(qda.class, Direction.2005)
```

```
##          Direction.2005
## qda.class Down  Up
##      Down   30  20
##      Up     81 121
```

```
round(mean(qda.class!= Direction.2005)*100, 2)
```

```
## [1] 40.08
```

- Interestingly, the QDA predictions are accurate almost 60% of the time. This level of accuracy is quite impressive for stock market data, which is known to be quite hard to model accurately. This suggests that the quadratic form assumed by QDA may capture the true relationship more accurately than the linear forms assumed by LDA and logistic regression.

K-Nearest Neighbors

- Let's set up the data for knn.

```
library (class)

##
train.X <- cbind(Lag1, Lag2)[train,]
test.X <- cbind (Lag1, Lag2)[!train,]
train.Direction <- Direction[train]
```

- Let's fit the model.

```
set.seed(1)
knn.pred <- knn(train.X, test.X, train.Direction, k=1)
table(knn.pred, Direction.2005)
```

```
##          Direction.2005
## knn.pred Down Up
##     Down   43 58
##     Up     68 83
```

```
##
round(mean(knn.pred!= Direction.2005)*100, 2)
```

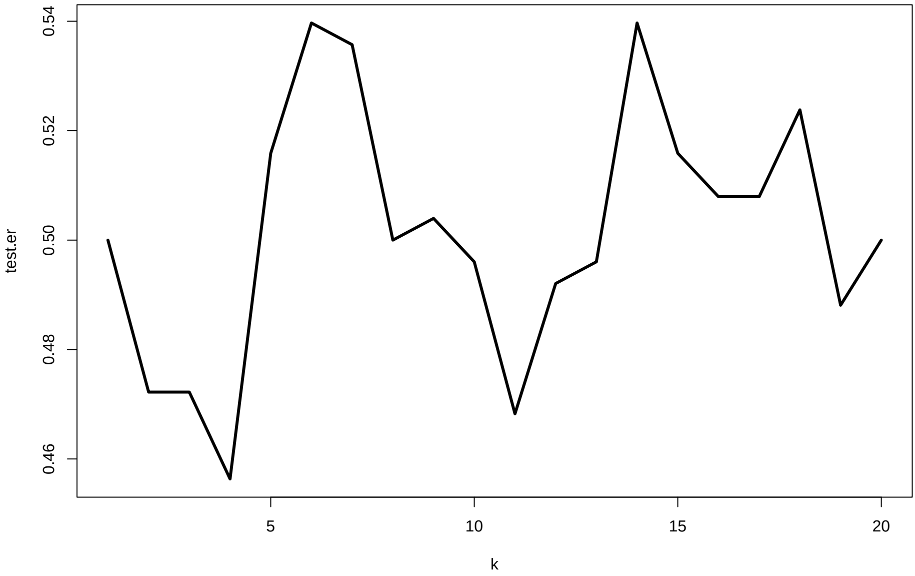
```
## [1] 50
```

- So a 50% testing error rate.

- Let's try $k = 1, \ldots, 20$ and plot the error rate.

```
k <- 1:20
test.er <- rep(0, length(k))

for(c in 1:20){
knn.pred <- knn(train.X, test.X, train.Direction, k=c)
test.er[c] <- mean(knn.pred!=Direction.2005)
}
```

```r
plot(k, test.er, type="l", lwd=3)
```



- It seems the best is $k = 4$.

```
set.seed(1)
knn.pred <- knn(train.X, test.X, train.Direction, k=4)
table(knn.pred, Direction.2005)
```

```
##          Direction.2005
## knn.pred Down Up
##     Down   45 58
##     Up     66 83
```

```
##
round(mean(knn.pred!= Direction.2005)*100, 2)
```

```
## [1] 49.21
```

- This still doesn't do that well!

- Overall QDA appears to do the best as it may be capturing some non-linearities. We could also try non-linear terms with logistic regression!