Supervised1
Memorization methods

資料科學 Data Science
張家銘 Jia-Ming Chang
政治大學資訊科學系

Copyright declaration 版權說明

- Some of the figures in this presentation are taken from "An Introduction to Statistical Learning, with applications in R" (Springer, 2013) with permission from the authors: G. James, D. Witten, T. Hastie and R. Tibshirani.
- The web site of the book
- The credit of individual is indicated in the bottom part of the slide.
  - ie.,

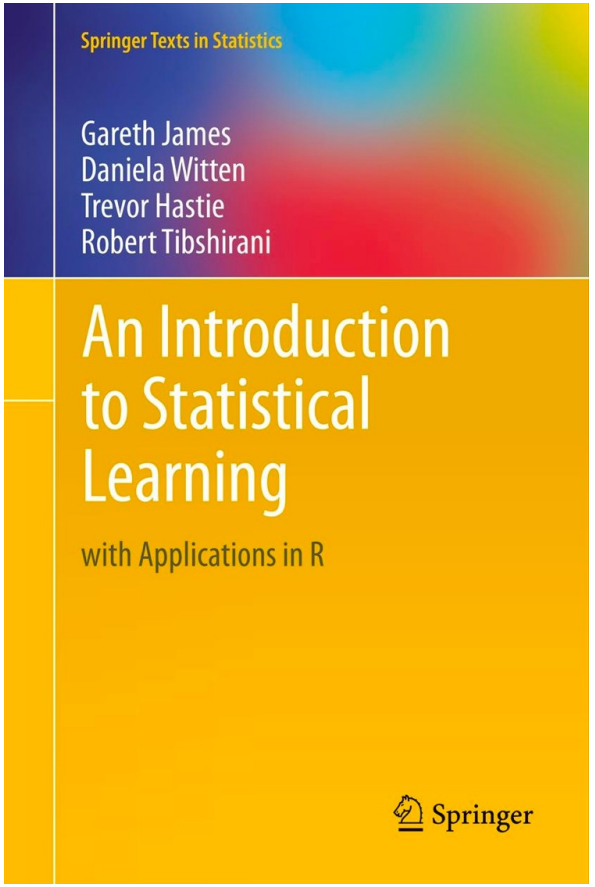


Figure 3.18, *An Introduction to Statistical Learning with Applications in R,* by Gareth James, Daniela Witten, Trevor Hastie and Robert Tibshirani

Copyright declaration 版權說明

- Some of the figures in this presentation are taken from "Practical Data Science with R (Manning, 2019)"
- The web site of the book
- The credit of individual is indicated in the bottom part of the slide.
  - ie.,

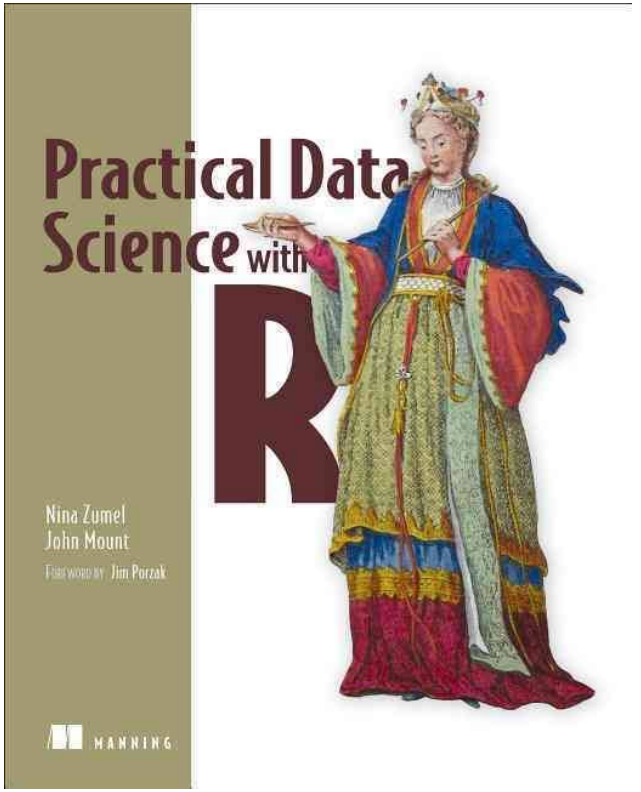
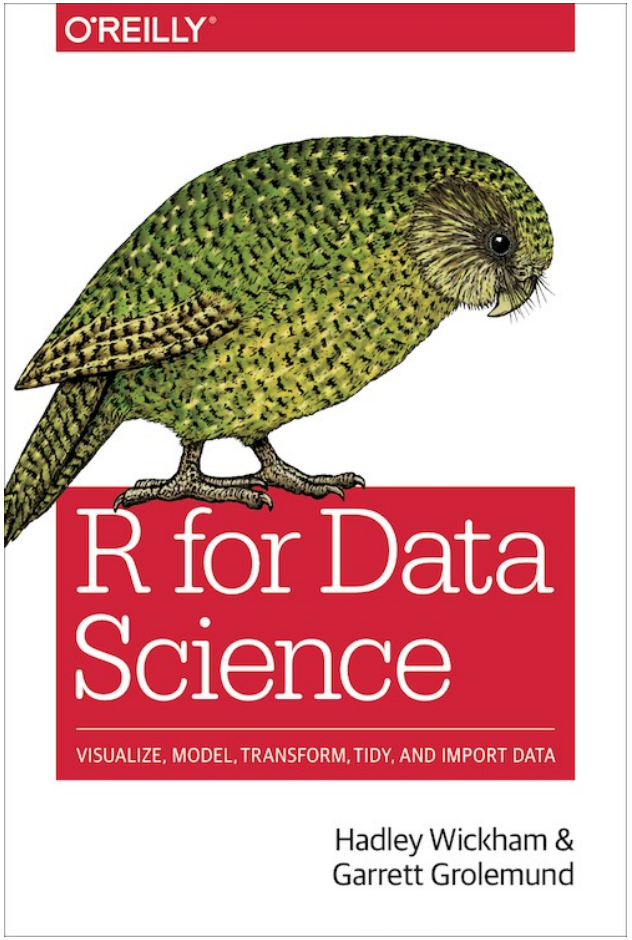


Figure 7.6, *Practical Data Science with R* by Nina Zumel and John Mount

Copyright declaration 版權說明

- Some of the figures in this presentation are taken from "R for Data Science" under the Creative Commons Attribution-NonCommercial-NoDerivs 3.0 License.
- The web site of the book
- The credit of individual is indicated in the bottom part.
  - ie.,

R for Data Science by Garrett Grolemund, Hadley Wickham



Recap for the last week

Unsupervised methods

- we'll look at methods to discover unknown relationships in data
- two classes of unsupervised methods
  - Cluster analysis : finds groups in your data with similar characteristics
    - Hierarchical clustering
    - k-means
  - Association rule : mining finds elements or properties in the data that tend to occur together.

The progress of the *K*-means algorithm

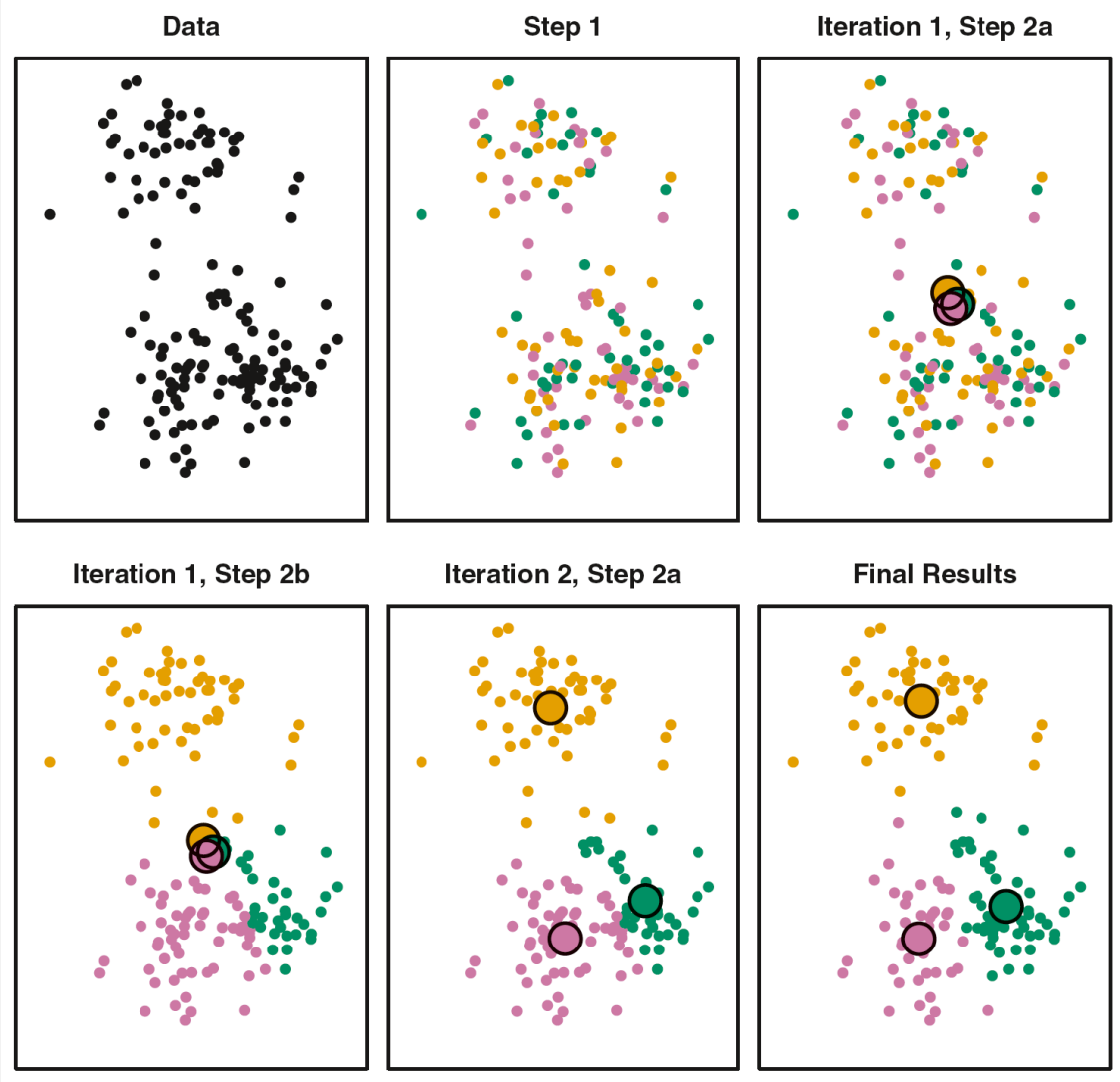


Figure10.6, *An Introduction to Statistical Learning with Applications in R* by Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani

K-means clustering performed 6 times

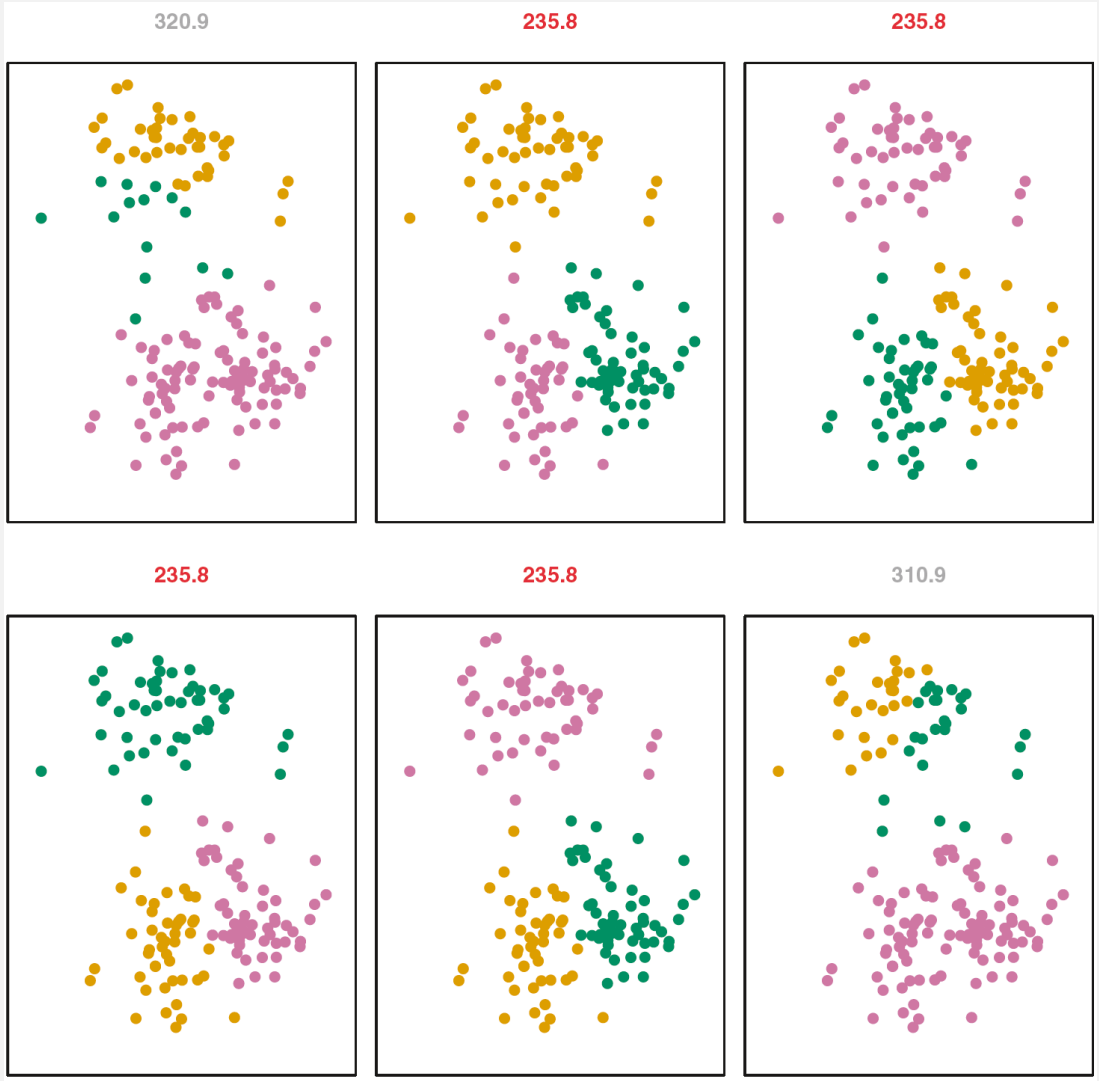


Figure10.7, *An Introduction to Statistical Learning with Applications in R* by Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani

Whether a given cluster is *real*?

- whether a cluster represents true structure is to see if the cluster holds up under plausible variations in the dataset?
- `clusterboot` (fpc package)
  - use bootstrap resampling to evaluate how stable a given cluster is
  - Jaccard coefficient
    - The Jaccard similarity between two sets A and B
    - the number of elements in the intersection of A and B / the number of elements in the union of A and B.

Cluster stability

- The cluster stability of each cluster in the original clustering
  - the mean value of its Jaccard coefficient over all the bootstrap iterations.
- stability value
  - 0.6 <= : unstable.
  - 0.6 ~ 0.75 : measure a pattern in the data, but there isn't high certainty about which points should be clustered together.
  - 0.85 >= : highly stable (they're likely to be real clusters).

Calinski-Harabasz index – CH.R

- B / W : should be maximized at the optimal *k*
  - B = BSS(*k*)/(*k*-1)
  - W= WSS(*k*)/(*n*-*k*)
  - the between-cluster variance (which is essentially the variance of all the cluster centroids from the dataset's grand centroid) / the total within-cluster variance (the average WSS of the clusters in the clustering)

Interpreting a Dendrogram

the vertical axis, indicates how different the two observations are.

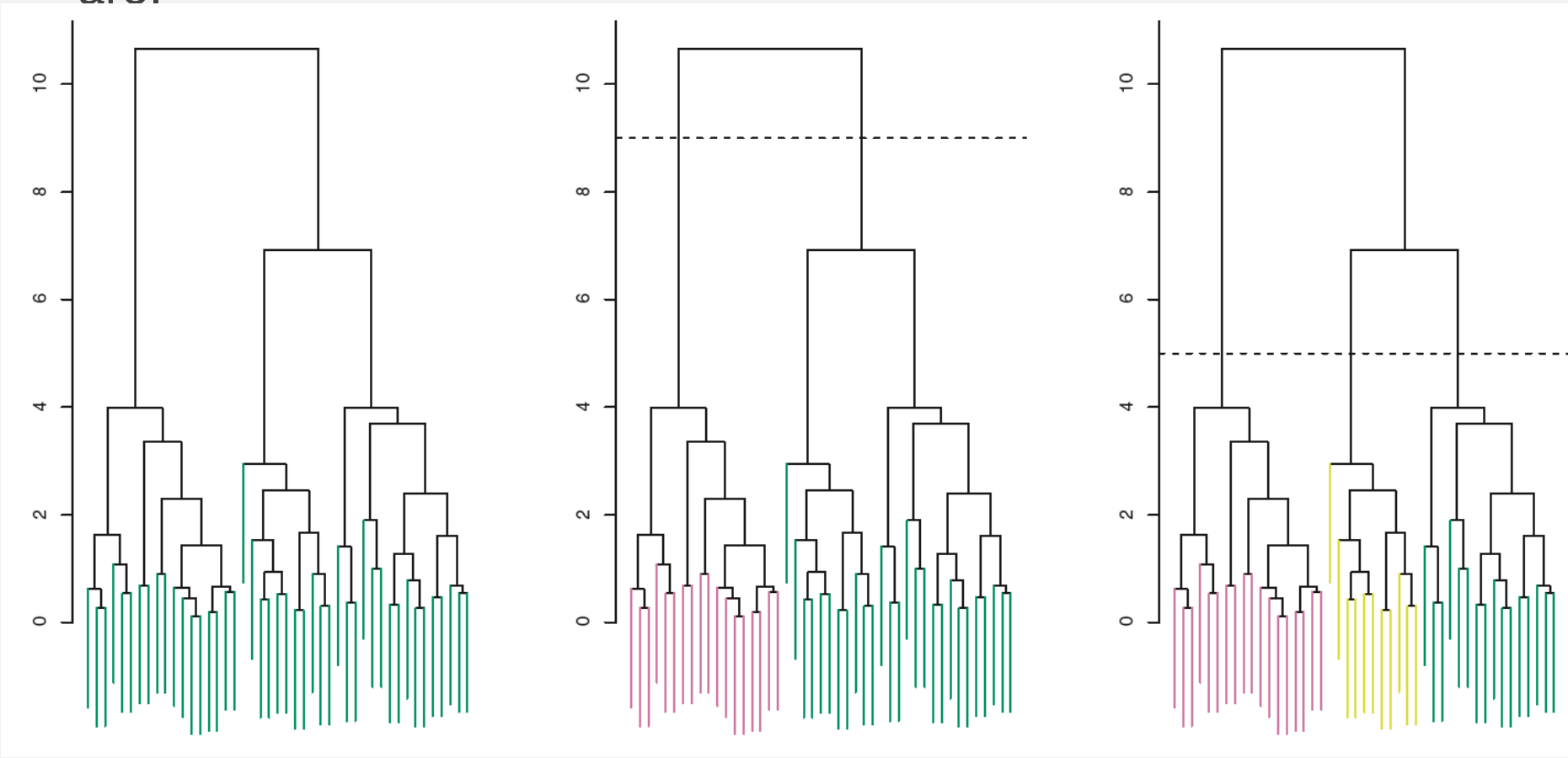


Figure10.9, *An Introduction to Statistical Learning with Applications in R* by Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani

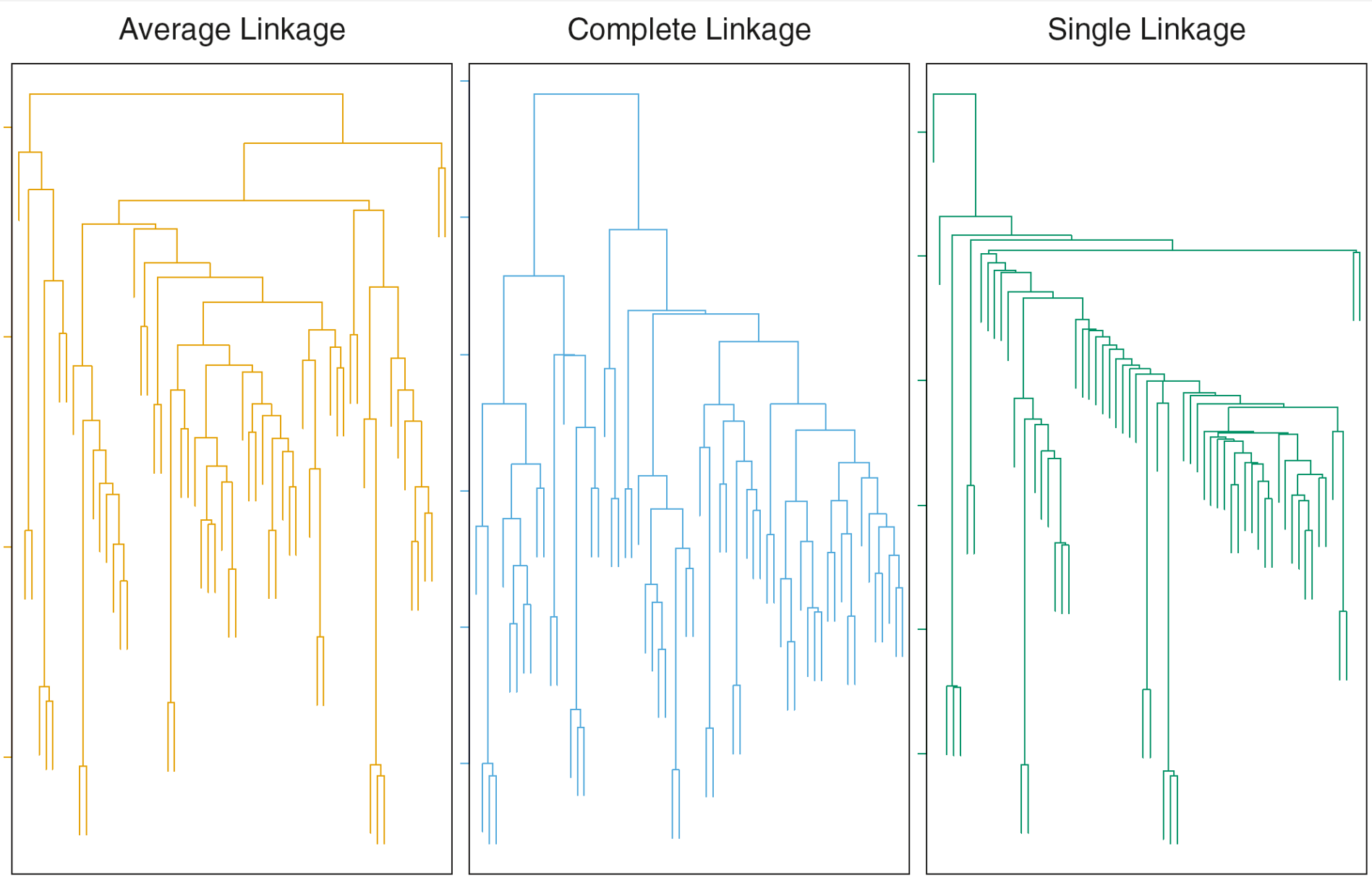


Figure10.12, *An Introduction to Statistical Learning with Applications in R* by Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani

Table 8.1 A database of library transactions

Transaction ID	Books checked out
1	*The Hobbit*, *The Princess Bride*
2	*The Princess Bride*, *The Last Unicorn*
3	*The Hobbit*
4	*The Neverending Story*
5	*The Last Unicorn*
6	*The Hobbit*, *The Princess Bride*, *The Fellowship of the Ring*
7	*The Hobbit*, *The Fellowship of the Ring*, *The Two Towers*, *The Return of the King*
8	*The Fellowship of the Ring*, *The Two Towers*, *The Return of the King*
9	*The Hobbit*, *The Princess Bride*, *The Last Unicorn*
10	*The Last Unicorn*, *The Neverending Story*

Table 8.1 A database of library transactions

Transaction ID	Books checked out
1	*The Hobbit*, *The Princess Bride*
2	*The Princess Bride*, *The Last Unicorn*
3	*The Hobbit*
4	*The Neverending Story*
5	*The Last Unicorn*
6	*The Hobbit*, *The Princess Bride*, *The Fellowship of the Ring*
7	*The Hobbit*, *The Fellowship of the Ring*, *The Two Towers*, *The Return of the King*
8	*The Fellowship of the Ring*, *The Two Towers*, *The Return of the King*
9	*The Hobbit*, *The Princess Bride*, *The Last Unicorn*
10	*The Last Unicorn*, *The Neverending Story*

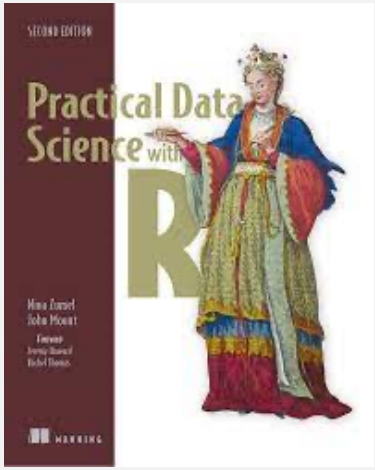
support of the itemset { The Hobbit, The Princess Bride} = 3/10

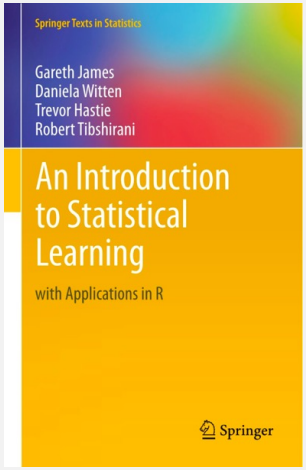
confidence : “People who check out The Hobbit also check out The Princess Bride” = 3/5

confidence : “People who check out The Princess Bride also check out The Hobbit ” = 3/4

Today

- *Practical Data Science with R* by Zumel, N. & Mount, J (Manning)
  - Chp. 6: Memorization methods
- *An Introduction to Statistical Learning with Applications in R* by Gareth James, Daniela Witten, Trevor Hastie and Robert Tibshirani
  - Chp. 2.2.3 The classification setting





Outline - Memorization methods

1. Building single-variable models
   - categorical features
   - numeric features
2. Cross-validated variable selection
3. Building basic multivariable models
   - nearest neighbor
   - naive Bayes

The Conference on Knowledge Discovery and Data Mining (KDD) Cup 2009

- customer relationship management.
  - Input: The contest supplied 230 facts about 50,000 credit card accounts
  - Goals: predict
    - churn : predict account cancellation
    - appetency : the innate tendency to use new products and services
    - upselling : willingness to respond favorably to marketing pitches

Preparing the KDD data for analysis

- KDD2009/KDD2009.R

```
d <- read.table('orange_small_train.data.gz',header=T, sep='\t',
na.strings=c('NA',''))

churn <- read.table('orange_small_train_churn.labels.txt',
header=F,sep='\t')

d$churn <- churn$V1
```

Preparing the KDD data for analysis

```
appetency <-
read.table('orange_small_train_appetency.labels.txt',
header=F,sep='\t')

d$appetency <- appetency$V1

upselling <-
read.table('orange_small_train_upselling.labels.txt',
header=F,sep='\t')

d$upselling <- upselling$V1
```

Preparing the KDD data for analysis

- Split data into train and test subsets

```
set.seed(729375)
d$rgroup <- runif(dim(d)[[1]])
dTrainAll <- subset(d,rgroup<=0.9)
dTest <- subset(d,rgroup>0.9)
```

Preparing the KDD data for analysis

- Identify features

```
outcomes=c('churn','appetency','upselling')
vars <- setdiff(colnames(dTrainAll), c(outcomes,'rgroup'))
catVars <- vars[sapply(dTrainAll[,vars],class) %in%
c('factor','character')]
numericVars <- vars[sapply(dTrainAll[,vars],class) %in%
c('numeric','integer')]
rm(list=c('d','churn','appetency','upselling'))
```

Preparing the KDD data for analysis

- Identify outputs

```
outcome <- 'churn'
pos <- '1'
```

Preparing the KDD data for analysis

- Further split training data into training and calibration.

```
useForCal <-
rbinom(n=dim(dTrainAll)[[1]],size=1,prob=0.1)>0
dCal <- subset(dTrainAll,useForCal)
dTrain <- subset(dTrainAll,!useForCal)
```

Preparing the KDD data for analysis

```
load('KDD2009.Rdata')
```

Subsample to prototype quickly

- Often the data scientist will be so engrossed with the business problem, math, and data that they forget how much trial and error is needed.
- It's often an excellent idea to first work on a small subset of your training data, so that it takes seconds to debug your code instead of minutes.
- Don't work with expensive data sizes until you have to.

Building single-variable models

Using categorical features

Using numeric features

Building single-variable models – Using categorical features

Using categorical features

- KDD2009/sinVal_cat.R
- Plotting churn grouped by variable 218 levels

```
table218 <- table(
Var218=dTrain[,'Var218'],
churn=dTrain[,outcome],
useNA='ifany')
print(table218)
```

Using categorical features

- Churn rates grouped by variable 218 codes

```
print(table218[,2]/(table218[,1]+table218[,2]))
```

Take Var200 as example

```
v<-"Var200"
outCol<-dTrain[,outcome]
varCol<-dTrain[,v]
appCol<-dTest[,v]
```

Take Var200 as example

```
pPos <- sum(outCol==pos)/length(outCol)
naTab <- table(as.factor(outCol[is.na(varCol)]))
pPosWna <- (naTab/sum(naTab))[pos]

vTab <- table(as.factor(outCol),varCol)
pPosWv <- (vTab[pos,]+1.0e-3*pPos)/(colSums(vTab)+1.0e-3)

pred <- pPosWv[appCol]
pred[is.na(appCol)] <- pPosWna
pred[is.na(pred)] <- pPos
```

Function to build single-variable models for categorical variables

```
mkPredC <- function(outCol,varCol,appCol) {
    pPos <- sum(outCol==pos)/length(outCol)
    naTab <- table(as.factor(outCol[is.na(varCol)]))
    pPosWna <- (naTab/sum(naTab))[pos]
    vTab <- table(as.factor(outCol),varCol)
    pPosWv <- (vTab[pos,]+1.0e-3*pPos)/(colSums(vTab)+1.0e-3)
    pred <- pPosWv[appCol]
    pred[is.na(appCol)] <- pPosWna
    pred[is.na(pred)] <- pPos
    pred
}
```

Applying single-categorical variable models to all of our datasets

```
for(v in catVars) {
    pi <- paste('pred',v,sep='')
    dTrain[,pi] <- mkPredC(dTrain[,outcome],dTrain[,v],dTrain[,v])
    dCal[,pi] <- mkPredC(dTrain[,outcome],dTrain[,v],dCal[,v])
    dTest[,pi] <- mkPredC(dTrain[,outcome],dTrain[,v],dTest[,v])
}
```

Scoring categorical variables by AUC

```
library('ROCR')
calcAUC <- function(predcol,outcol) {
   perf <-
   performance(prediction(predcol,outcol==pos),'auc')
   as.numeric(perf@y.values)
}
```

Scoring categorical variables by AUC

```
for(v in catVars) {
    pi <- paste('pred',v,sep='')
    aucTrain <- calcAUC(dTrain[,pi],dTrain[,outcome])
    if(aucTrain>=0.8) {
        aucCal <- calcAUC(dCal[,pi],dCal[,outcome])
        print(sprintf("%s, trainAUC: %4.3f calibrationAUC:
        %4.3f", pi,aucTrain,aucCal))
    }
}
```

- What is the most promising variable?

Building single-variable models - Using numeric features

Building single-variable models : Using numeric features

- bin the numeric feature into a number of ranges

  quantile()

  cut()

Take Var7 as example

```
v<-"Var7"
outCol<-dTrain[,outcome]
varCol<-dTrain[,v]
appCol<-dTest[,v]
```

Take Var7 as example

```
cuts <- unique(as.numeric(quantile(varCol,
probs=seq(0, 1, 0.1),na.rm=T)))
varC <- cut(varCol,cuts,include.lowest=T)
appC <- cut(appCol,cuts, include.lowest=T)
mkPredC(outCol,varC,appC)
```

Building single-variable models : Using numeric features

- KDD2009/sinVal_num.R

```
mkPredN <- function(outCol,varCol,appCol) {
    cuts <- unique(as.numeric(quantile(varCol,
    probs=seq(0, 1, 0.1),na.rm=T)))
    varC <- cut(varCol,cuts)
    appC <- cut(appCol,cuts)
    mkPredC(outCol,varC,appC)
}
```

Scoring numeric variables by AUC

```
for(v in numericVars) {
    pi <- paste('pred',v,sep='')
    dTrain[,pi] <- mkPredN(dTrain[,outcome],dTrain[,v],dTrain[,v])
    dTest[,pi] <- mkPredN(dTrain[,outcome],dTrain[,v],dTest[,v])
    dCal[,pi] <- mkPredN(dTrain[,outcome],dTrain[,v],dCal[,v])
    aucTrain <- calcAUC(dTrain[,pi],dTrain[,outcome])
    if(aucTrain>=0.55) {
        aucCal <- calcAUC(dCal[,pi],dCal[,outcome])
        print(sprintf("%s, trainAUC: %4.3f calibrationAUC: %4.3f",
        pi,aucTrain,aucCal))
    }
}
```

- Which variable is the best? How to check?

Plotting variable performance

```
ggplot(data=dCal) +
geom_density(aes(x=predVar126,color=as.factor(churn)))
```

```
ggplot(data=dCal) +
geom_density(aes(x=predVar81,color=as.factor(churn)))
```

Dealing with missing values in numeric variables?

- two-step process
  1. for each numeric variable, introduce a new advisory variable
     - 1 : when the original variable had a missing value
     - 0 : otherwise.
  2. replace all missing values of the original variable with 0

Using cross-validation to estimate effects of overfitting

```
var <- 'Var217'
aucs <- rep(0,100)
for(rep in 1:length(aucs)) {
    useForCalRep <- rbinom(n=dim(dTrainAll)[[1]],size=1,prob=0.1)>0
    predRep <- mkPredC(dTrainAll[!useForCalRep,outcome],
    dTrainAll[!useForCalRep,var],
    dTrainAll[useForCalRep,var])
    aucs[rep] <- calcAUC(predRep,dTrainAll[useForCalRep,outcome])
}
mean(aucs)
sd(aucs)
```

Using cross-validation to estimate effects of overfitting

- without for() loop

```
fCross <- function() {
    useForCalRep <-
    rbinom(n=dim(dTrainAll)[[1]],size=1,prob=0.1)>0
    predRep <- mkPredC(dTrainAll[!useForCalRep,outcome],
    dTrainAll[!useForCalRep,var],
    dTrainAll[useForCalRep,var])
    calcAUC(predRep,dTrainAll[useForCalRep,outcome])
}
aucs <- replicate(100,fCross())
```

replicate v.s. for loop

```
ptm <- proc.time()
aucs <- replicate(100,fCross())
proc.time() - ptm

ptm <- proc.time()
aucs <- rep(0,100)
for(rep in 1:length(aucs)) {
    useForCalRep <- rbinom(n=dim(dTrainAll)[[1]],size=1,prob=0.1)>0
    predRep <- mkPredC(dTrainAll[!useForCalRep,outcome],
    dTrainAll[!useForCalRep,var],
    dTrainAll[useForCalRep,var])
    aucs[rep] <- calcAUC(predRep,dTrainAll[useForCalRep,outcome])
}
proc.time() - ptm
```

replicate v.s. for loop

```
ptm <- proc.time()
aucs <- replicate(1000,fCross())
proc.time() - ptm

ptm <- proc.time()
aucs <- rep(0,1000)
for(rep in 1:length(aucs)) {
    useForCalRep <- rbinom(n=dim(dTrainAll)[[1]],size=1,prob=0.1)>0
    predRep <- mkPredC(dTrainAll[!useForCalRep,outcome],
    dTrainAll[!useForCalRep,var],
    dTrainAll[useForCalRep,var])
    aucs[rep] <- calcAUC(predRep,dTrainAll[useForCalRep,outcome])
}
proc.time() - ptm
```

Using cross-validation to estimate effects of overfitting

- In some modeling circumstances, training set estimations are good enough (linear regression is often such an example).
- In many other circumstances, estimations from a single calibration set are good enough.
- In extreme cases (such as fitting models with very many variables or level values), you're well advised to use replicated cross-validation estimates of variable utilities and model fits.

Using cross-validation to estimate effects of overfitting

- Automatic cross-validation is extremely useful in all modeling situations, so it's critical you automate your modeling steps so you can perform cross-validation studies.

Building models using many variables

Naive Bayes

k-nearest neighbor

Building models using many variables – Naive Bayes

Using Naive Bayes

- *y*
  - *T* or *True* if the person is employed
  - *F* otherwise
- evidence (ev1) : the predicate education "High School"
- $P(\text{ev1}|y==T)$
  - conditional probability of ev1, given $y==T$
- $P(y==T\ |\text{ev1})$

Bayes' law

$$P(y == T|ev_1) = \frac{P(y == T) \times P(ev_1|y == T)}{P(ev_1)}$$

$$P(y == F|ev_1) = \frac{P(y == F) \times P(ev_1|y == F)}{P(ev_1)}$$

Two-class problem

- Class = 1
  - $\Pr(Y = 1 | X = x_0) > 0.5$
- Class = 2
  - otherwise

Two groups blue & orange

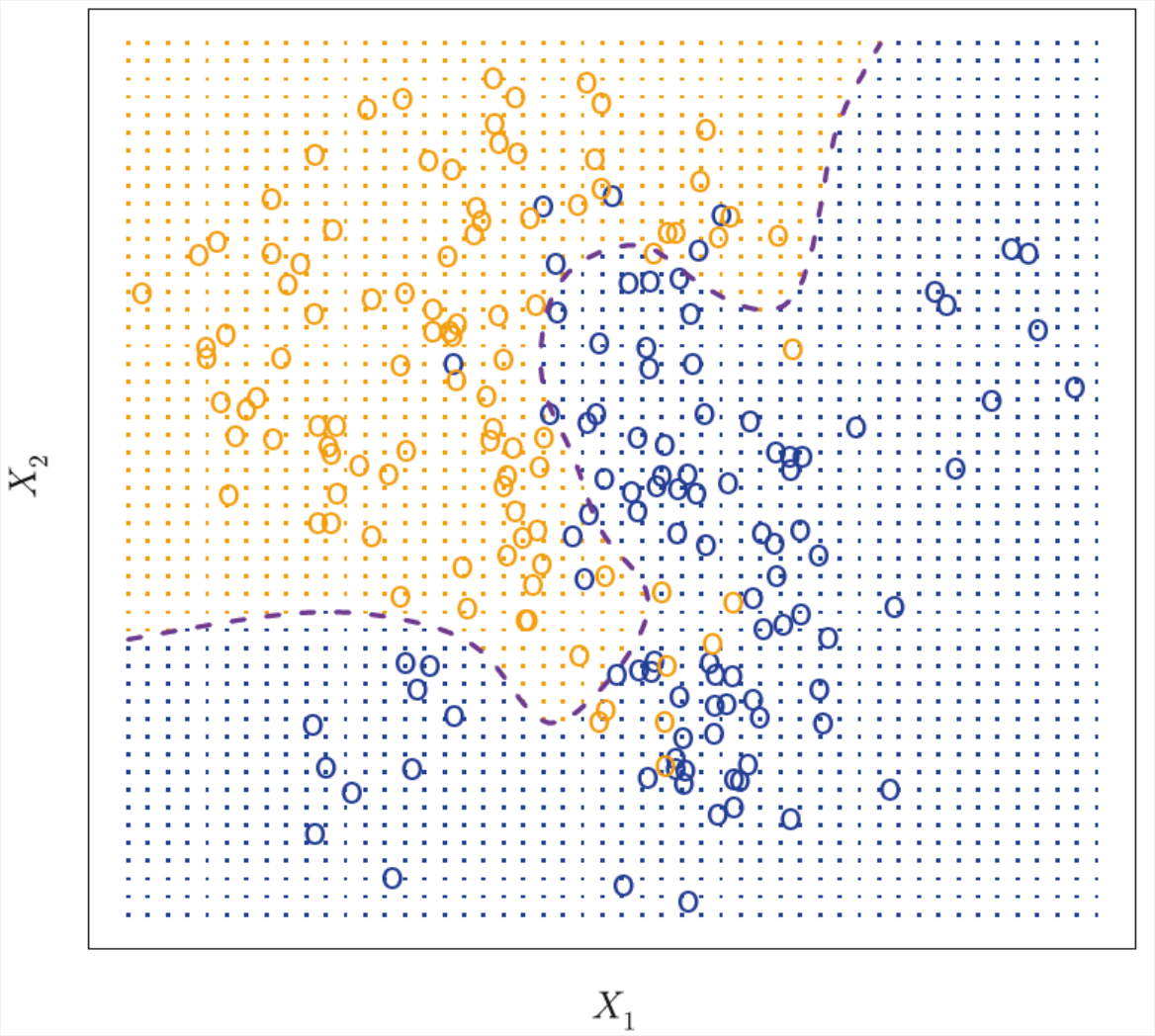


Figure2.13, *An Introduction to Statistical Learning with Applications in R* by Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani

Two groups blue & orange

- Orange shaded region
  - the set of points `Pr(Y=orange|X) > 50%`
- Blue shaded region
  - the set of points `Pr(Y=orange|X) < 50%`
- purple dashed line
  - `Pr(Y=orange|X) = 50%`
  - Bayes decision boundary

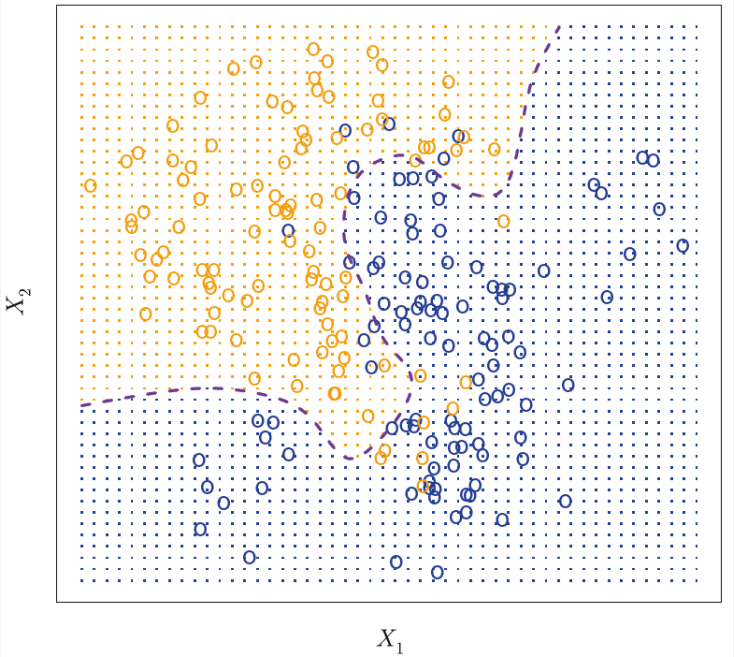


Figure2.13, *An Introduction to Statistical Learning with Applications in R* by Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani

Bayes error rate

- The error rate at $X = x_0$

$$1 - \max_j \Pr(Y = j | X = x_0)$$

- In general,

$$1 - E\left(\max_j \Pr(Y = j | X)\right)$$

Naive Bayes assumption

- $P(y == T \mid ev_1 \ldots ev_N)$?
- all the evidence is conditionally independent of each other for a given outcome
  - $P(ev_1 \& \ldots \& ev_N \mid y==T) \approx P(ev_1 \mid y==T) \times \ldots \times P(ev_N \mid y==T)$
  - $P(ev_1 \& \ldots \& ev_N \mid y==F) \approx P(ev_1 \mid y==F) \times \ldots \times P(ev_N \mid y==F)$

Naive Bayes assumption

- P(*y*==*T* | evidence) + P(*y*==*F* | evidence) = 1

$$P(y == T|ev_1 \& \ldots ev_N) \approx \frac{P(y == T) \times \left(P(ev_1|y == T) \times \cdots \times P(ev_N|y == T)\right)}{P(ev_1 \& \ldots \& ev_N)}$$

$$P(y == F|ev_1 \& \ldots ev_N) \approx \frac{P(y == F) \times \left(P(ev_1|y == F) \times \cdots \times P(ev_N|y == F)\right)}{P(ev_1 \& \ldots \& ev_N)}$$

Naive Bayes

- For numerical reasons, it's better to convert the products into sums

$$score(T|ev_1 \& \ldots \& e_N) = \log(P(y == T)) + \log\big(P(ev_1|y == T)\big) + \cdots + \log(P(ev_N|y == T))$$

$$score(F|ev_1 \& \ldots \& e_N) = \log(P(y == F)) + \log\big(P(ev_1|y == F)\big) + \cdots + \log(P(ev_N|y == F))$$

$$P(ev_1|y == T) = \frac{P(ev_1 \,\&\, y == T)}{P(y == T)}$$

Building a Naive Bayes model

- KDD2009/mulVal_NaiveBayes.R

$$P(y == T)$$

```
pPos <- sum(dTrain[,outcome]==pos)/length(dTrain[,outcome])
```

$$P(ev_1|y == T) = \frac{P(ev_1 \,\&\, y == T)}{P(y == T)}$$

```
log(pf/pPos + smoothingEpsilon)
```

$$score(T|ev_1 \& \ldots \& e_N) = \log(P(y == T)) + \log\big(P(ev_1|y == T)\big) + \cdots + \log(P(ev_N|y == T))$$

```
scorePos <- log(pPos + smoothingEpsilon) + rowSums(log(pf/pPos + smoothingEpsilon))
```

Building a Naive Bayes model

- KDD2009/mulVal_NaiveBayes.R

```
pPos <- sum(dTrain[,outcome]==pos)/length(dTrain[,outcome])

nBayes <- function(pPos,pf) {
      pNeg <- 1 - pPos
      smoothingEpsilon <- 1.0e-5
      scorePos <- log(pPos + smoothingEpsilon) + rowSums(log(pf/pPos + smoothingEpsilon))
      scoreNeg <- log(pNeg + smoothingEpsilon) + rowSums(log((1-pf)/(1-pPos) +
      smoothingEpsilon))
      m <- pmax(scorePos,scoreNeg)
      expScorePos <- exp(scorePos-m)
      expScoreNeg <- exp(scoreNeg-m)
      expScorePos/(expScorePos+expScoreNeg)
}
```

Applying and evaluating a Naive Bayes model

```
pVars <- paste('pred',c(numericVars,catVars),sep='')
dTrain$nbpredl <- nBayes(pPos,dTrain[,pVars])
dCal$nbpredl <- nBayes(pPos,dCal[,pVars])
dTest$nbpredl <- nBayes(pPos,dTest[,pVars])
```

Applying and evaluating a Naive Bayes model

```
print(calcAUC(dTrain$nbpredl,dTrain[,outcome]))
print(calcAUC(dCal$nbpredl,dCal[,outcome]))
print(calcAUC(dTest$nbpredl,dTest[,outcome]))
```

Using a Naive Bayes package

```
library('e1071')

lVars <- c(catVars,numericVars)

ff <- paste('as.factor(',outcome,'>0) ~ ', paste(lVars,collapse=' +
'),sep='')

nbmodel <- naiveBayes(as.formula(ff),data=dTrain)


dTrain$nbpred <- predict(nbmodel,newdata=dTrain,type='raw')[,'TRUE']

dCal$nbpred <- predict(nbmodel,newdata=dCal,type='raw')[,'TRUE']

dTest$nbpred <- predict(nbmodel,newdata=dTest,type='raw')[,'TRUE']
```

Using a Naive Bayes package

```
calcAUC(dTrain$nbpred,dTrain[,outcome])
calcAUC(dCal$nbpred,dCal[,outcome])
calcAUC(dTest$nbpred,dTest[,outcome])
```

The property of Naive Bayes

- Naive Bayes doesn't perform any clever optimization, so it can be outperformed by methods like logistic regression and support vector machines.
- Naive Bayes is particularly useful when you have a very large number of features that are rare and/or nearly independent.
  - Document classification: bag-of-words or bag-of-k-grams

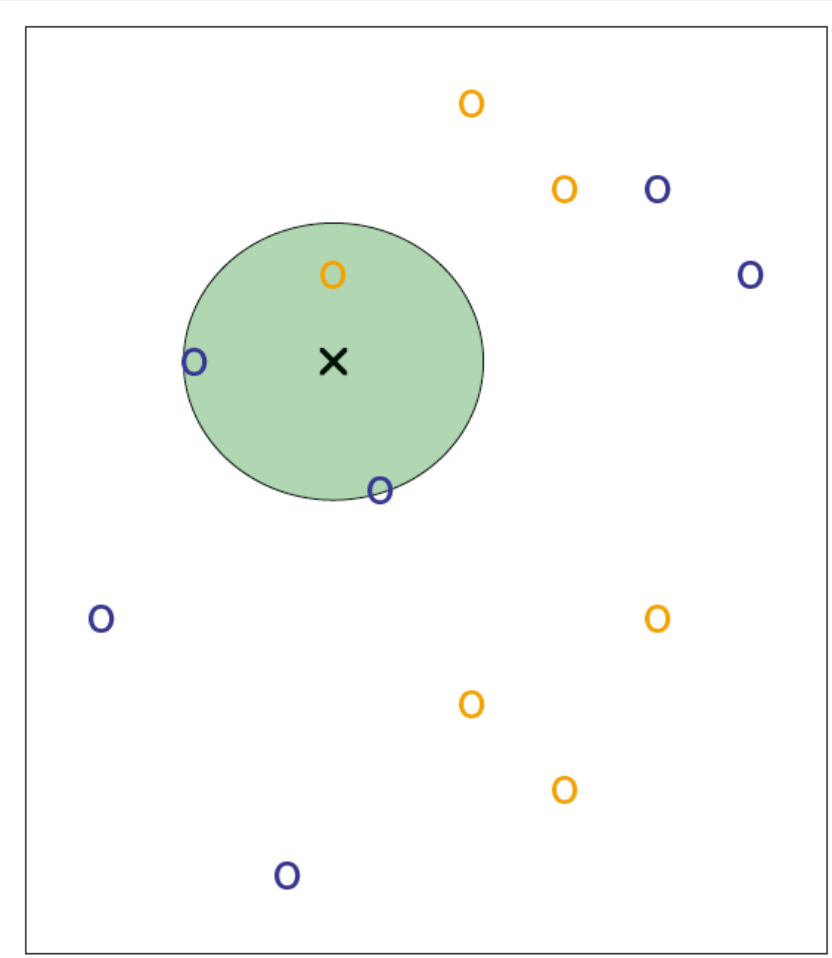
Building models using many variables - *k*-nearest neighbor

How to conditional distribution of Y given X?

- a test observation x_0
  - first identifies the K points in the training data that are closest to x_0, represented by N_0.
- then estimates the conditional probability for class j as the fraction of points in N_0 whose response values equal j

$$\Pr(Y = j|X = x_0) = \frac{1}{k}\sum_{i \in N_0} I(y_i = j)$$

$K = 3$



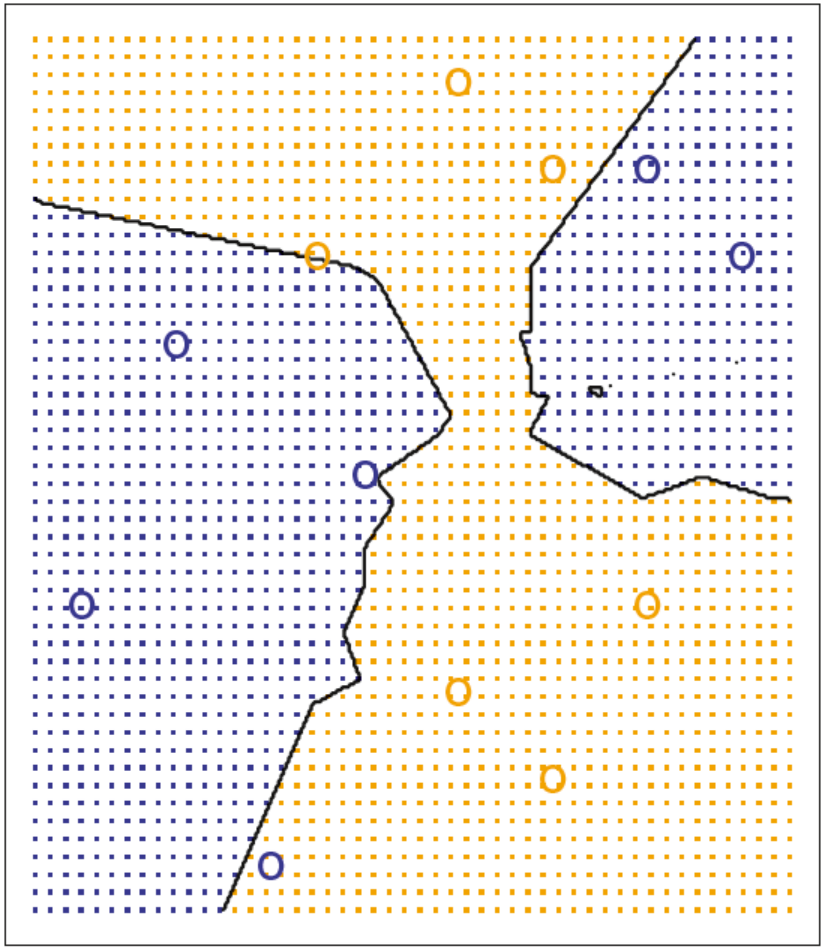


Figure2.14, *An Introduction to Statistical Learning with Applications in R* by Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani

Simulation data by $K = 10$

- error rate
  - KNN : 0.1363
  - Bayes: 0.1304

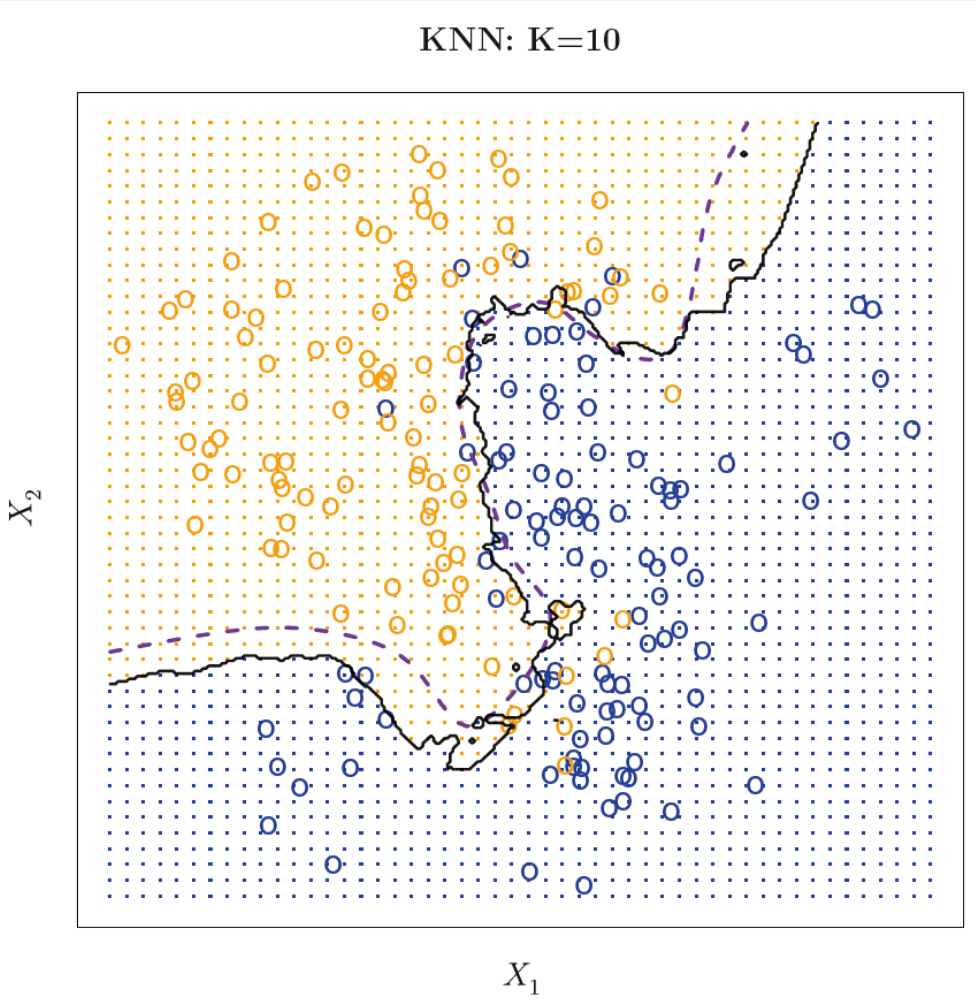
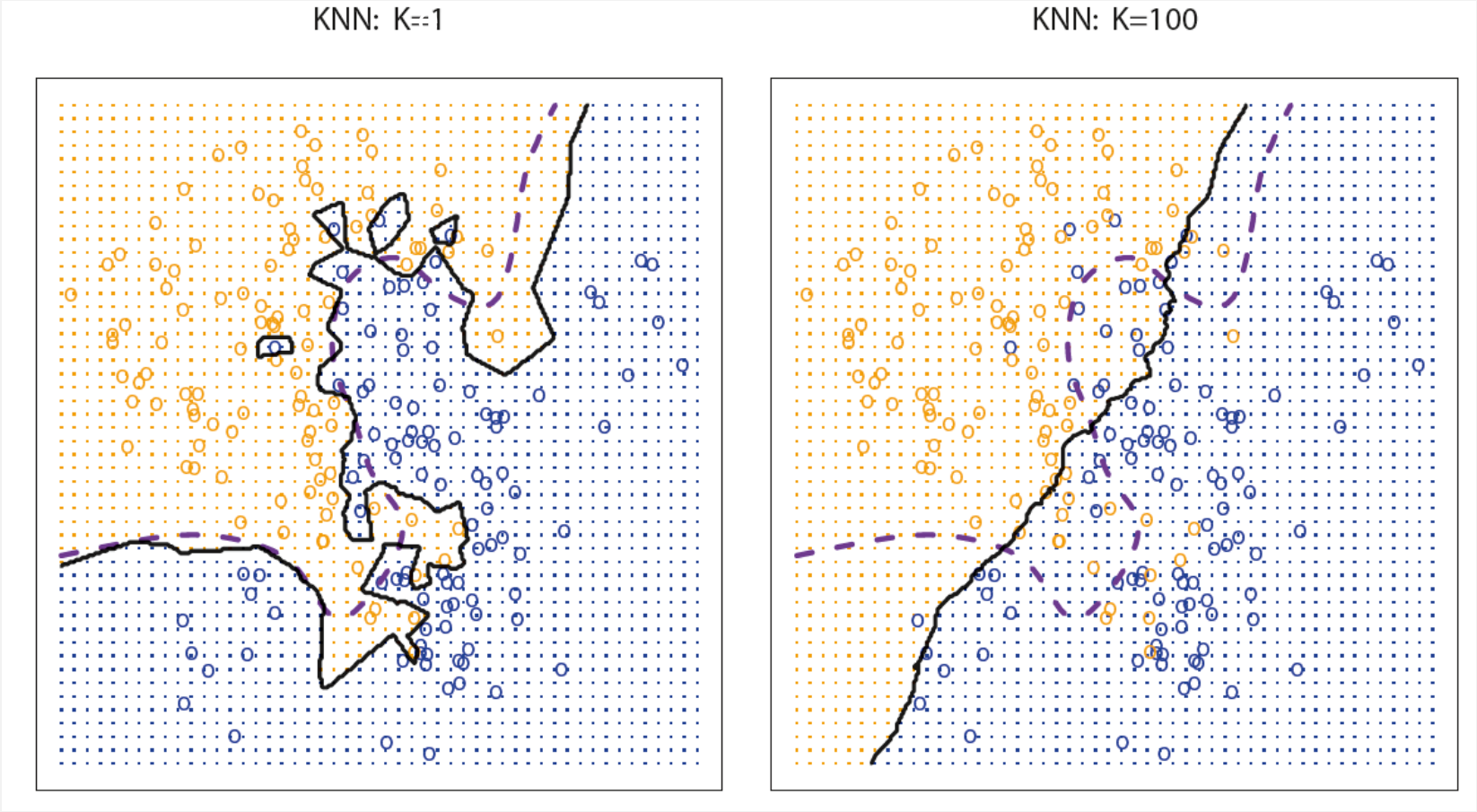


Figure2.15, *An Introduction to Statistical Learning with Applications in R* by Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani

The bigger *K*, the better?



low-bias but very high-variance

low-variance but high-bias

Figure2.16, *An Introduction to Statistical Learning with Applications in R* by Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani

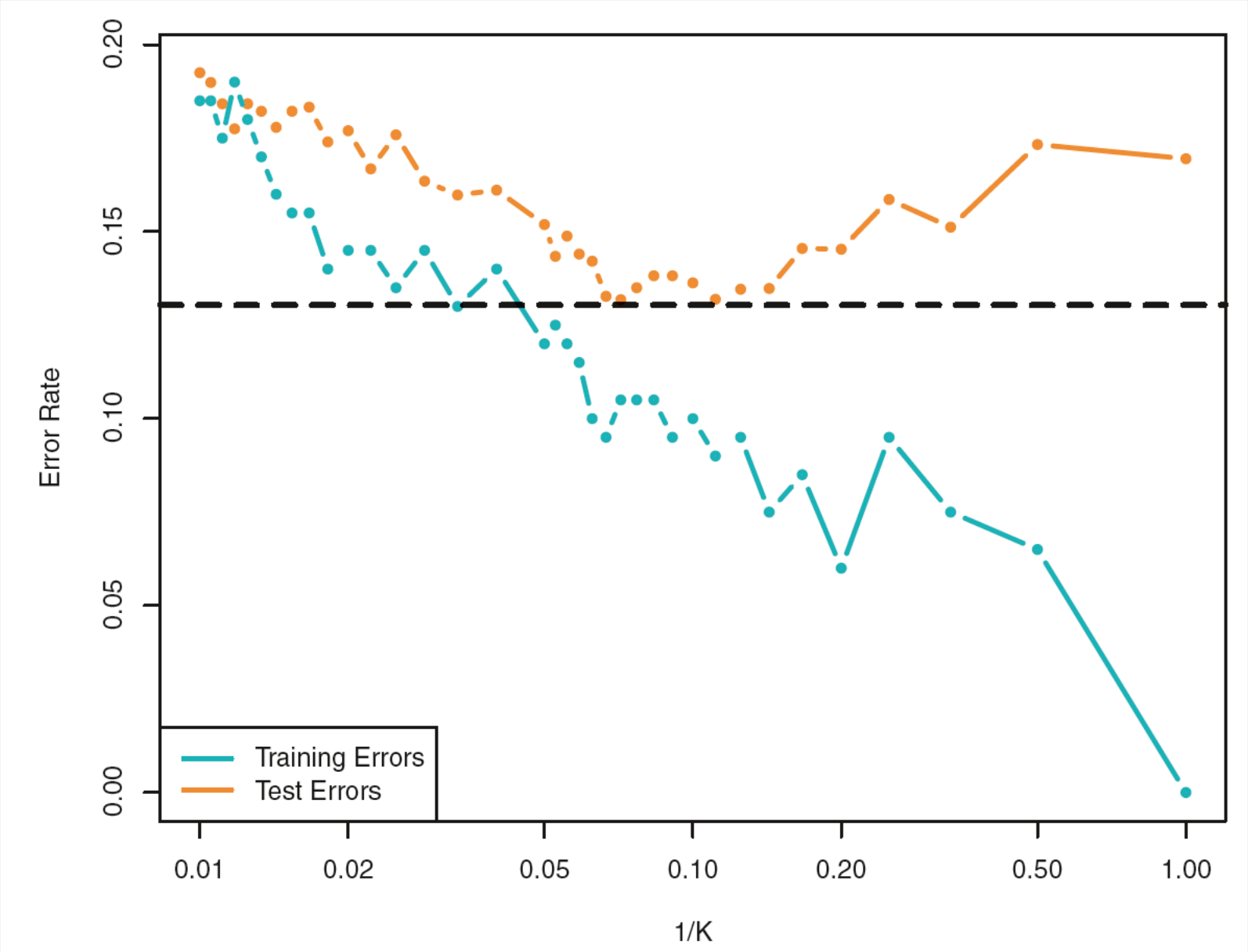


Figure2.17, *An Introduction to Statistical Learning with Applications in R* by Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani

Using nearest neighbor methods

- One problem with *K*-NN is the nature of its concept space?
- 3-nearest neighbor
  - => zero, one, two, or three examples of churn
  - => the estimated probability of churn = 0%, 33%, 66%, or 100%.

Using nearest neighbor methods

- events with unbalanced outcomes (that probabilities not near 50%)
  - using a large *k* so KNN can express a useful range of probabilities
  - have a good chance of seeing 10 positive examples in each neighborhood
  - around 7% in our training data => 10/0.07 = 142

Variable selection

- KDD2009/mulVal_knn.R

```
logLikelyhood <-
function(outCol,predCol) {
    sum(ifelse(outCol==pos,log(predCol),log(1-predCol)))
}
```

Variable selection

- KDD2009/mulVal_knn.R

```
selVars <- c()
minStep <- 5
baseRateCheck <- logLikelyhood(dCal[,outcome],
sum(dCal[,outcome]==pos)/length(dCal[,outcome]))
```

Variable selection for categorical variables

```
for(v in catVars) {
    pi <- paste('pred',v,sep='')
    liCheck <- 2*((logLikelyhood(dCal[,outcome],dCal[,pi]) -
    baseRateCheck))
    if(liCheck>minStep) {
        print(sprintf("%s, calibrationScore: %g",
        pi,liCheck))
        selVars <- c(selVars,pi)
    }
}
```

Variable selection for numerical variables

```
for(v in numericVars) {
    pi <- paste('pred',v,sep='')
    liCheck <- 2*((logLikelyhood(dCal[,outcome],dCal[,pi]) -
    baseRateCheck) - 1)
    if(liCheck>=minStep) {
        print(sprintf("%s, calibrationScore: %g",
        pi,liCheck))
        selVars <- c(selVars,pi)
    }
}
```

Running *k*-nearest neighbors

```
library('class')
nK <- 200
knnTrain <- dTrain[,selVars]
knnCl <- dTrain[,outcome]==pos
knnPred <- function(df) {
    knnDecision <- knn(knnTrain,df,knnCl,k=nK,prob=T)
    ifelse(knnDecision==TRUE,
    attributes(knnDecision)$prob,
    1-(attributes(knnDecision)$prob))
}
```

Running *k*-nearest neighbors

```
print(calcAUC(knnPred(dTrain[,selVars]),dTrain[,outcome]))
print(calcAUC(knnPred(dCal[,selVars]),dCal[,outcome]))
print(calcAUC(knnPred(dTest[,selVars]),dTest[,outcome]))
```

Check the performance

- Platting 200-nearest neighbor performance

```
dCal$kpred <- knnPred(dCal[,selVars])
ggplot(data=dCal) +
geom_density(aes(x=kpred,
color=as.factor(churn),linetype=as.factor(churn
)))
```

Check the performance

- What do you observe?
  - Distributions
    - multimodal are often evidence that there are significant effects we haven't yet explained.
    - unimodal or even look normal are consistent with the unexplained effects being simple noise.
  - What we're looking for are the two distributions to be unimodal
  - if not separated, at least not completely on top of each other

Check the performance

- Plotting the receiver operating characteristic (ROC) curve

```
plotROC <- function(predcol,outcol) {
    perf <-
    performance(prediction(predcol,outcol==pos),'tpr','fpr')
    pf <-
    data.frame( FPR=perf@x.values[[1]],TPR=perf@y.values[[1]])
    ggplot() +
    geom_line(data=pf,aes(x=FPR,y=TPR)) +
    geom_line(aes(x=c(0,1),y=c(0,1)))
}
print(plotROC(knnPred(dTest[,selVars]),dTest[,outcome]))
```

Summary

AUC

methods	train	calibration	test
Single-variable			
categorical variable	0.830	0.565	0.551
numerical variable	0.635	0.629	0.637
Multivariable			
k-NN	0.744	**0.711**	**0.718**
Naïve Bayes	**0.975**	0.599	0.595
Naïve Bayes - e1071	0.464	0.554	0.567

Single-variable model

- Single-variable models can be thought of as being simple summaries of the training data (categorical variables)

  => the model is essentially a contingency table or pivot table

  => essentially organize the training data into a number of subsets indexed by the predictive variable

  => store a summary of the distribution of outcome as their future prediction.

Naive Bayes

- form their decision by building a large collection of independent single-variable models
- Prediction for a given example is just the product of all the applicable single variable model adjustments
- Predictions are just sums of appropriate summaries of the original training data.

K-nearest neighbor

- summaries of the *k* pieces of training data that are closest to the example to be scored.
- KNN models usually store all of their original training data instead of an efficient summary
- => they truly do memorize the training data

Summary

- Always try single-variable models before trying more complicated techniques.
- Single-variable modeling techniques give you a useful start on variable selection.
- Always compare your model performance to the performance of your best single-variable model.
- Consider decision trees, nearest neighbor, and naive Bayes models as basic data memorization techniques and, if appropriate, try them early in your projects.

AITC
教育部人工智慧技術及應用人才培育計畫
Artificial Intelligence Talent Cultivation Program