



Business Analytics (BA)

Course Code: IM-618

Credit: 3

7th March to 20th March 2020

**Post Graduate Diploma in Management
(PGDM)**

Term III (Jan – Mar 2020)

Naïve Bayes

It is well for the heart to be naive and for the mind not to be.
Anatole France

Characteristics

- Data-driven, not model-driven
- Makes no assumptions about the data
- Named after mid-16th century English statistician and Presbyterian minister Thomas Bayes



Naïve Bayes: The Basic Idea

For a given new record to be classified, find other records like it (i.e., same values for the predictors)

What is the prevalent class among those records?

Assign that class to your new record

Naïve Bayes Algorithm

- Naïve Bayes Algorithm learns the probability of an object with certain features belonging to a particular group/class.
- All the properties individually contribute to the probability
- Since it is simple so we call it Naïve and this was presented by Thomas Bayes so it is called Naïve Bayes

Example: Selection of Fruit based on color, shape, size

Where it is used?

If you want to build a spam detector, analyze customer sentiment, or automatically categorize products, customers, or competitors, you can do that using a Naïve Bayes classifier

The Mathematics :

$$P(A/B) = P(B/A) P(A) / P(B)$$

$P(A/B)$: Probability (Conditional probability) of occurrence of event A given the event B is True

$P(A)$ and $P(B)$: Probabilities of the occurrence of event A and B respectively

$P(B/A)$: Probability of the occurrence of event B given the event A is true

TERMINOLOGY

A is called the proposition and B is called the evidence

$P(A)$ is called the prior probability of proposition and $P(B)$ is called the prior probability of evidence

$P(A/B)$ is called the posterior

$P(B/A)$ is the likelihood

Posterior = (Likelihood)*(Proposition prior Probability)/Evidence prior probability

Example:

Suppose you have to draw a single card from a standard deck of 52 cards. Now the probability that the card is a Queen is $P(\text{Queen}) = \frac{4}{52} = \frac{1}{13}$. If you are given evidence that the card that you have picked is a face card, the posterior probability $P(\text{Queen}|\text{Face})$ can be calculated using Bayes' Theorem as follows:

$$P(\text{Queen}|\text{Face}) = \frac{P(\text{Face}|\text{Queen})}{P(\text{Face})} \cdot P(\text{Queen})$$

Now $P(\text{Face}|\text{Queen}) = 1$ because given the card is Queen, it is definitely a face card. We have already calculated $P(\text{Queen})$. The only value left to calculate is $P(\text{Face})$, which is equal to $\frac{3}{13}$ as there are three face cards for every suit in a deck. Therefore,

$$P(\text{Queen}|\text{Face}) = \frac{1}{13} \cdot \frac{13}{3} = \frac{1}{3}$$

Usage

- Requires categorical variables
- Numerical variable must be binned and converted to categorical
- Can be used with very large data sets
- Example: Spell check programs assign your misspelled word to an established “class” (i.e., correctly spelled word)

Exact Bayes Classifier

- Relies on finding other records that share same predictor values as record-to-be-classified.
- Want to find “probability of belonging to class C , given specified values of predictors.”
- Even with large data sets, may be hard to find other records that **exactly match** your record, in terms of predictor values.

Calculations

- Assume independence of predictor variables (within each class)
- Use multiplication rule
- Find same probability that record belongs to class C, given predictor values, without limiting calculation to records that share all those same values

Solution – Naïve Bayes

1. Take a record, and note its predictor values
2. Find the probabilities those predictor values occur across all records in C1
3. Multiply them together, then by proportion of records belonging to C1
4. Same for C2, C3, etc.
5. Prob. of belonging to C1 is value from step (3) divide by sum of all such values C1 ... Cn
6. Establish & adjust a “cutoff” prob. for class of interest

Example: Financial Fraud

Target variable:

Audit finds fraud, no fraud

Predictors:

Prior pending legal charges
(yes/no)

Size of firm (small/large)

Charges?	Size	Outcome
y	small	truthful
n	small	truthful
n	large	truthful
n	large	truthful
n	small	truthful
n	small	truthful
y	small	fraud
y	large	fraud
n	large	fraud
y	large	fraud

Exact Bayes Calculations

Goal: classify (as “fraudulent” or as “truthful”) a small firm with charges filed

There are 2 firms like that, one fraudulent and the other truthful

$P(\text{fraud} \mid \text{charges}=y, \text{size}=small) = \frac{1}{2} = 0.50$

Note: calculation is limited to the two firms matching those characteristics

Naïve Bayes Calculations

Same goal as before

Compute 2 quantities:

Proportion of “charges = y” among frauds, times proportion of “small” among frauds, times proportion frauds
 $= 3/4 * 1/4 * 4/10 = 0.075$

Prop “charges = y” among frauds, times prop. “small” among truthfuls, times prop. truthfuls $= 1/6 * 4/6 * 6/10$
 $= 0.067$

$P(\text{fraud} \mid \text{charges, small}) = 0.075 / (0.075 + 0.067) = 0.53$

- Note that probability **estimate** does not differ greatly from **exact**
- All records are used in calculations, not just those matching predictor values
- This makes calculations practical in most circumstances
- Relies on assumption of independence between predictor variables within each class

Independence Assumption

- Not strictly justified (variables often correlated with one another)
- Often “good enough” – ranking of probabilities is more important than unbiased estimate of actual probabilities

Advantages & Disadvantages

Advantages

- It is easy and fast to predict the class of the test data set. It also performs well in multi-class prediction
- When assumption of independence holds, a Naive Bayes classifier performs better compare to other models like logistic regression and you need less training data.
- It perform well in case of categorical input variables compared to numerical variable(s). For numerical variable, normal distribution is assumed (bell curve, which is a strong assumption).
- Handles purely categorical data well
- Works well with very large data sets
- Simple & computationally efficient

Disadvantages

- Requires large number of records
- If categorical variable has a category (in test data set), which was not observed in training data set, then model will assign a 0 (zero) probability and will be unable to make a prediction. This is often known as Zero Frequency.
- On the other side naive Bayes is also known as a bad estimator, so the probability outputs are not to be taken too seriously.
- Another limitation of Naive Bayes is the assumption of independent predictors. In real life, it is almost impossible that we get a set of predictors which are completely independent.

Naïve Bayes in R

- Use package `e1071`
- Function `naiveBayes`
- See Table 8.4 for code for running Naïve Bayes
 - Includes code for binning numeric variables into categories, which is required for NB

Summary

- No statistical models involved
- Naïve Bayes (like KNN) pays attention to complex interactions and local structure
- Computational challenges remain
- Probability rankings are more accurate than the actual probability estimates
 - Good for applications using lift (e.g. response to mailing), less so for applications requiring probabilities (e.g. credit scoring)

K-Nearest-Neighbor

**If you want to annoy your neighbors, tell the truth about them.
-Pietro Aretino**

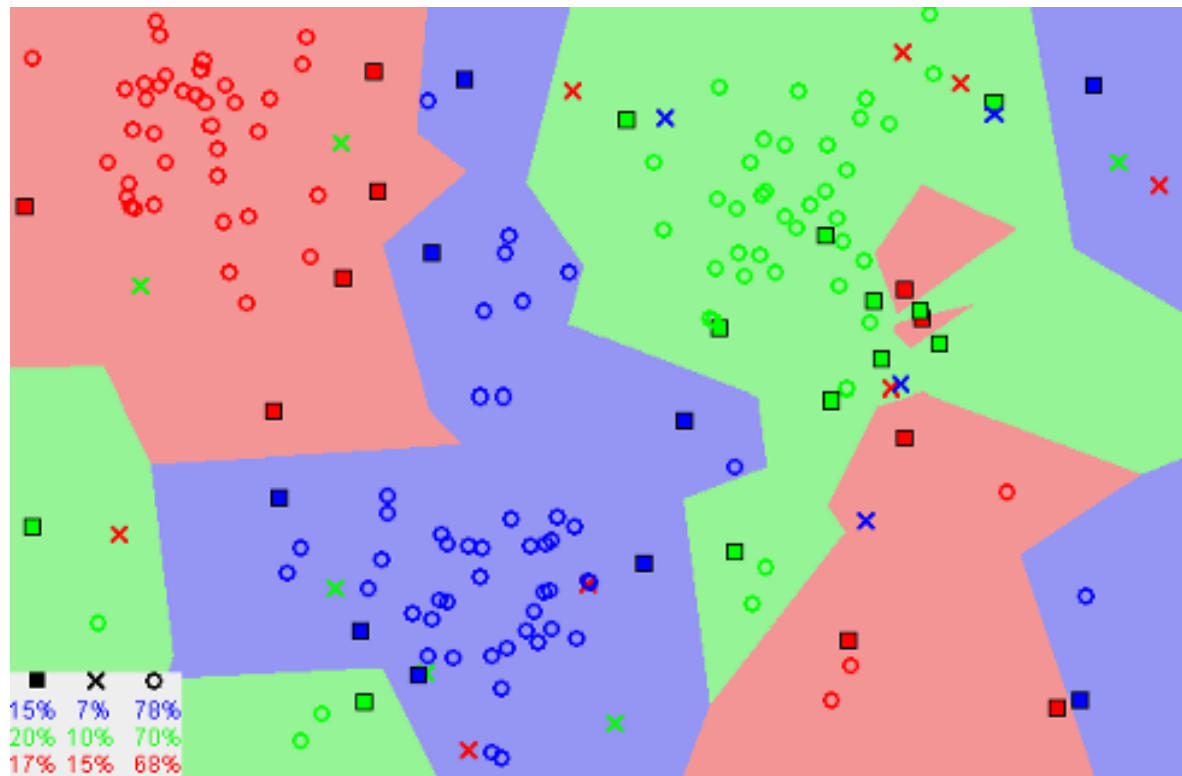
KNN Explained:

A k-nearest-neighbor algorithm, often abbreviated k-nn, is an approach to data classification that estimates how likely a data point is to be a member of one group or the other depending on what group the data points nearest to it are in.

The k-nearest-neighbor is an example of a "lazy learner" algorithm, meaning that it does not build a model using the training set until a query of the data set is performed.

- k-nearest neighbor is a type of instance-based learning.
- You can use it for text document classification, financial distress prediction modeling, and competitor analysis and classification.

KNN Example:



Similar data points are close to each other

Characteristics of K-NN

- Data-driven, not model-driven
- Makes no assumptions about the data

Basic Idea

- For a given record to be classified, identify nearby records
- “Near” means records with similar predictor values X_1, X_2, \dots, X_p
- Classify the record as whatever the predominant class is among the nearby records (the “neighbors”)

The KNN Algorithm

- Load the data
- Initialize K to your chosen number of neighbors
- For each example in the data
 - Calculate the distance between the query example and the current example from the data.
 - Add the distance and the index of the example to an ordered collection
- Sort the ordered collection of distances and indices from smallest to largest (in ascending order) by the distances
- Pick the first K entries from the sorted collection
- Get the labels of the selected K entries
- If regression, return the mean of the K labels
- If classification, return the mode of the K labels

How to measure “nearby”?

The most popular distance measure is **Euclidean distance**

$$\sqrt{(x_1 - u_1)^2 + (x_2 - u_2)^2 + \dots + (x_p - u_p)^2}$$

- Typically, predictor variables are first normalized (= standardized) to put them on comparable scales
- Use `preProcess()` from `caret` package to normalize
- Otherwise, metrics with large scales dominate

Choosing k

K is the number of nearby neighbors to be used to classify the new record

$K=1$ means use the single nearest record

$K=5$ means use the 5 nearest records

Typically choose that value of k which has lowest error rate in validation data

Low k vs. High k

Low values of k (1, 3, ...) capture local structure in data (but also noise)

High values of k provide more smoothing, less noise, but may miss local structure

Note: the extreme case of $k = n$ (i.e., the entire data set) is the same as the “naïve rule” (classify all records according to majority class)

Example: Riding Mowers

Data: 24 households classified
as owning or not owning
riding mowers

Predictors: Income, Lot Size

Income	Lot_Size	Ownership
60.0	18.4	owner
85.5	16.8	owner
64.8	21.6	owner
61.5	20.8	owner
87.0	23.6	owner
110.1	19.2	owner
108.0	17.6	owner
82.8	22.4	owner
69.0	20.0	owner
93.0	20.8	owner
51.0	22.0	owner
81.0	20.0	owner
75.0	19.6	non-owner
52.8	20.8	non-owner
64.8	17.2	non-owner
43.2	20.4	non-owner
84.0	17.6	non-owner
49.2	17.6	non-owner
59.4	16.0	non-owner
66.0	18.4	non-owner
47.4	16.4	non-owner
33.0	18.8	non-owner
51.0	14.0	non-owner
63.0	14.8	non-owner

Finding nearest neighbors in R

Library FNN provides a list of neighbors

Library class allows numerical output

See Table 7.2 for code using knn from FNN library; compares each record from validation* set to k nearest records in training

*termed the test set in R

Use library `caret` to get accuracy of different values of k, applied to validation data (see next slide for code)

Output

```
> accuracy.df
  k accuracy
1  1    0.7
2  2    0.7
3  3    0.8
4  4    0.9
5  5    0.8
6  6    0.9
7  7    0.9
8  8    1.0
9  9    0.9
10 10    0.9
11 11    0.9
12 12    0.8
13 13    0.4
14 14    0.4
```

When the most-accurate k is an even number (here it's 8), it is possible for ties to occur in classifying new records. R breaks ties randomly.

KNN Code For Riding Mower Example

```
library(caret)

# initialize a data frame with two columns: k, and accuracy.
accuracy.df <- data.frame(k = seq(1, 14, 1), accuracy = rep(0, 14))

# compute knn for different k on validation.
for(i in 1:14) {
  knn.pred <- knn(train.norm.df[, 1:2], valid.norm.df[, 1:2],
                  cl = train.norm.df[, 3], k = i)
  accuracy.df[i, 2] <- confusionMatrix(knn.pred, valid.norm.df[,
    3])$overall[1]
}
```

Using K-NN for Prediction (for Numerical Outcome)

- Instead of “majority vote determines class” use average of response values
- May be a weighted average, weight decreasing with distance

Summary

Advantages

- Simple
- No assumptions required about Normal distribution, etc.
- Effective at capturing complex interactions among variables without having to define a statistical model

Disadvantages

- Required size of training set increases exponentially with # of predictors, p
- This is because expected distance to nearest neighbor increases with p (with large vector of predictors, all records end up “far away” from each other)
- In a large training set, it takes a long time to find distances to all the neighbors and then identify the nearest one(s)
- These constitute “curse of dimensionality”

Summary

- Find distance between record-to-be-classified and all other records
- Select k-nearest records
 - Classify it according to majority vote of nearest neighbors
 - Or, for prediction, take the as average of the nearest neighbors
- “Curse of dimensionality” – need to limit # of predictors

Cluster Analysis

Clustering: The Main Idea

Goal: Form groups (clusters) of similar records

Used for **segmenting markets** into groups of similar customers

Example: Claritas segmented US neighborhoods based on demographics & income: “Furs & station wagons,” “Money & Brains”

Other Applications

- Periodic table of the elements
- Classification of species
- Grouping securities in portfolios
- Grouping firms for structural analysis of economy
- Army uniform sizes

Example: Public Utilities

Goal: find clusters of similar utilities

Data: 22 firms, 8 variables

Fixed-charge covering ratio

Rate of return on capital

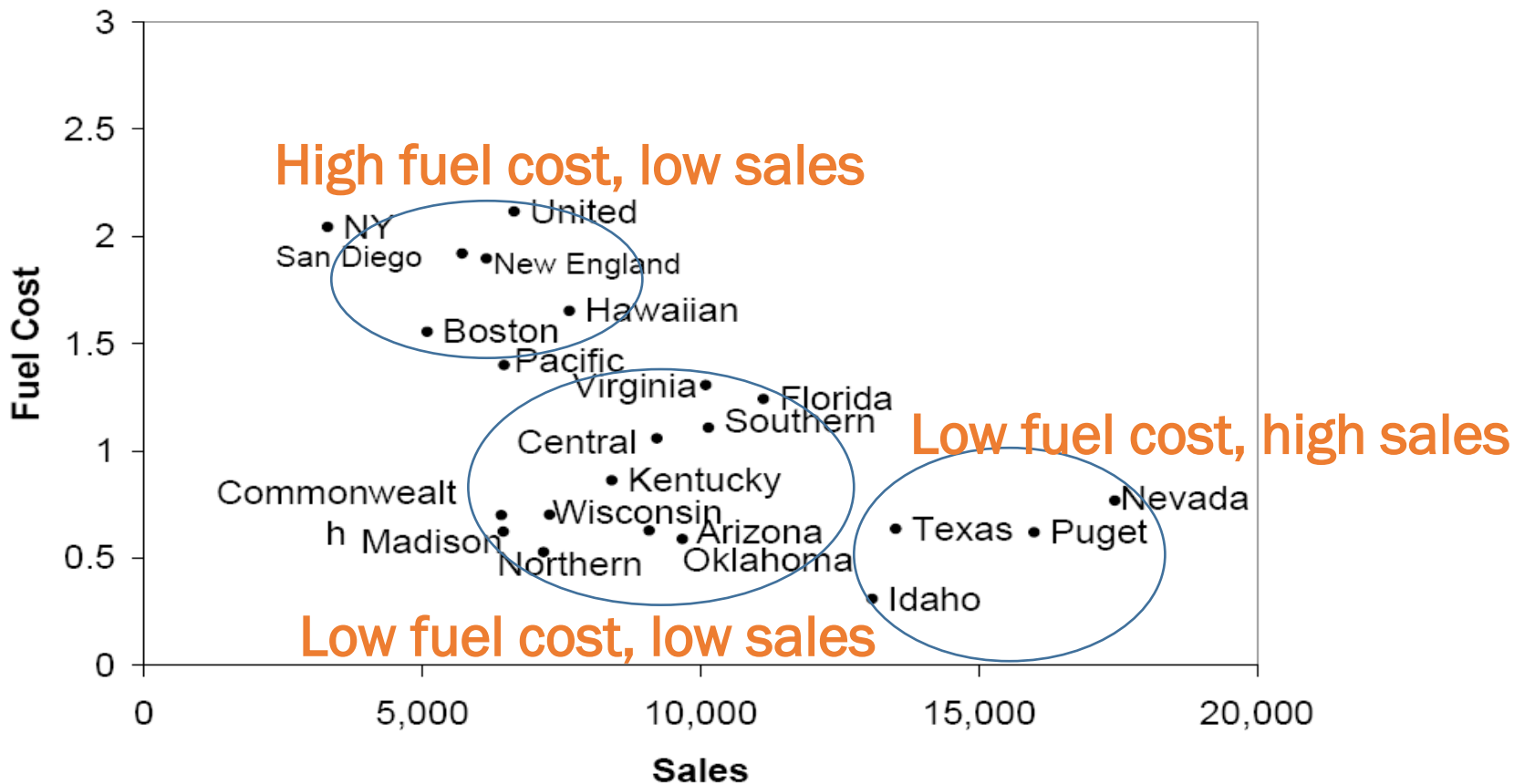
Cost per kilowatt capacity

Annual load factor

Growth in peak demand

Company	Fixed_charge	RoR	Cost	Load	Δ Demand	Sales	Nuclear	Fuel_Cost
Arizona	1.06	9.2	151	54.4	1.6	9077	0	0.628
Boston	0.89	10.3	202	57.9	2.2	5088	25.3	1.555
Central	1.43	15.4	113	53	3.4	9212	0	1.058
Commonwealth	1.02	11.2	168	56	0.3	6423	34.3	0.7
Con Ed NY	1.49	8.8	192	51.2	1	3300	15.6	2.044
Florida	1.32	13.5	111	60	-2.2	11127	22.5	1.241
Hawaiian	1.22	12.2	175	67.6	2.2	7642	0	1.652
Idaho	1.1	9.2	245	57	3.3	13082	0	0.309
Kentucky	1.34	13	168	60.4	7.2	8406	0	0.862
Madison	1.12	12.4	197	53	2.7	6455	39.2	0.623
Nevada	0.75	7.5	173	51.5	6.5	17441	0	0.768
New England	1.13	10.9	178	62	3.7	6154	0	1.897
Northern	1.15	12.7	199	53.7	6.4	7179	50.2	0.527
Oklahoma	1.09	12	96	49.8	1.4	9673	0	0.588
Pacific	0.96	7.6	164	62.2	-0.1	6468	0.9	1.4
Puget	1.16	9.9	252	56	9.2	15991	0	0.62
San Diego	0.76	6.4	136	61.9	9	5714	8.3	1.92
Southern	1.05	12.6	150	56.7	2.7	10140	0	1.108
Texas	1.16	11.7	104	54	-2.1	13507	0	0.636
Wisconsin	1.2	11.8	148	59.9	3.5	7287	41.1	0.702
United	1.04	8.6	204	61	3.5	6650	0	2.116
Virginia	1.07	9.3	174	54.3	5.9	10093	26.6	1.306

Sales & Fuel Cost: 3 rough clusters can be seen



Extension to More Than 2 Dimensions

In prior example, clustering was done by eye

Multiple dimensions require formal algorithm with

- A **distance measure**

- A way to use the distance measure in forming clusters

We will consider two algorithms: **hierarchical** and **non-hierarchical**

Hierarchical Clustering

Agglomerative Methods

- Begin with n -clusters (each record its own cluster)

- Keep joining records into clusters until one cluster is left (the entire data set)

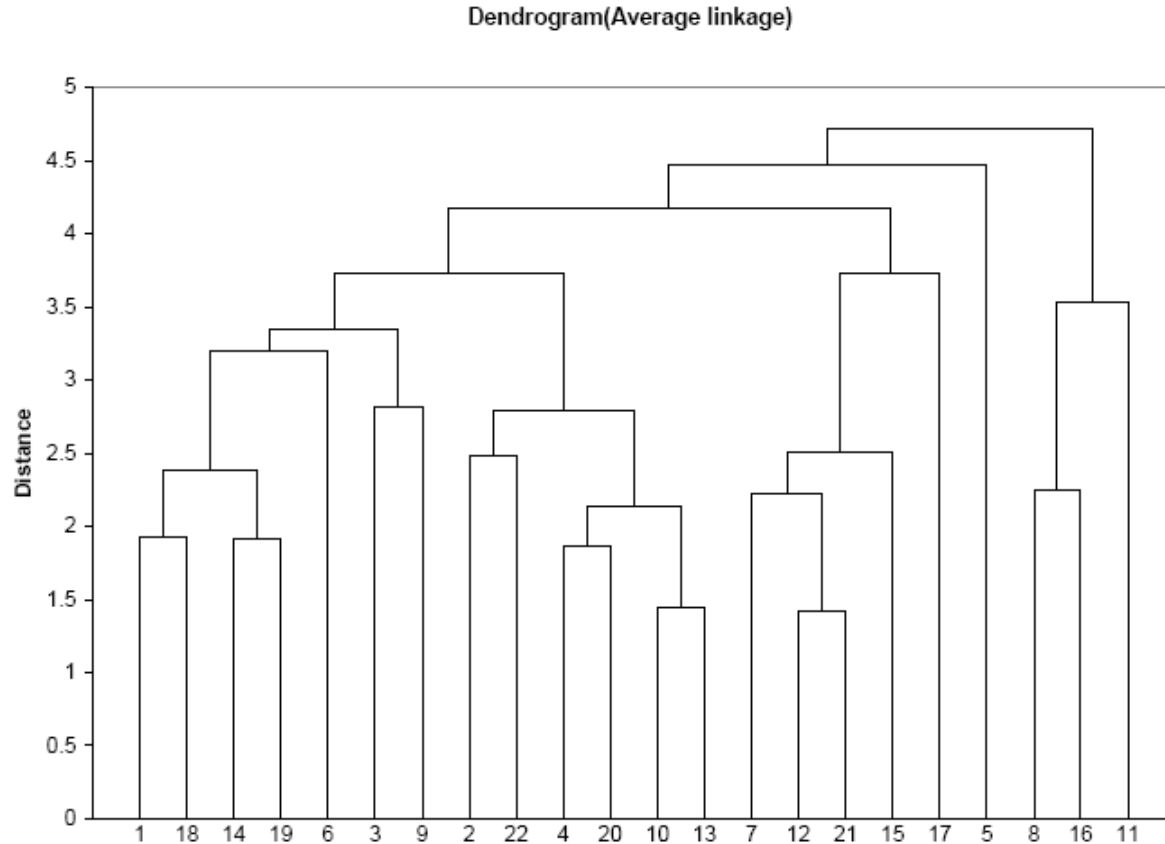
- Most popular

Divisive Methods

- Start with one all-inclusive cluster

- Repeatedly divide into smaller clusters

A Dendrogram shows the cluster hierarchy



Measuring Distance
Between records
Between clusters

Measuring Distance Between Records

Distance Between Two Records

Euclidean Distance is most popular:

$$d_{ij} = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \cdots + (x_{ip} - x_{jp})^2}$$

Normalizing with Examples

Problem: Raw distance measures are highly influenced by scale of measurements

Solution: normalize (standardize) the data first

- Subtract mean, divide by std. deviation
- Also called **z-scores**

For 22 utilities:

Avg. sales = 8,914


Std. dev. = 3,550

Normalized score for Arizona sales:

$$(9,077 - 8,914) / 3,550 = 0.046$$

Compute Distance Matrix for Utility Pairs

```
utilities.df <- read.csv("Utilities.csv")  
# set row names to the utilities column  
row.names(utilities.df) <- utilities.df[,1]  
# remove the utility column  
utilities.df <- utilities.df[,-1]  
# compute Euclidean distance  
d <- dist(utilities.df, method = "euclidean")
```



Could use a variety of metrics here, see R
documentation for `dist`

Output of Distance Matrix (showing first 5 pairs)

	Arizona	Boston	Central	Commonwealth	NY
Boston	3989.40808				
Central	140.40286	4125.04413			
Commonwealth	2654.27763	1335.46650	2789.75967		
NY	5777.16767	1788.06803	5912.55291	3123.15322	
Florida	2050.52944	6039.68908	1915.15515	4704.36310	7827.42921

Code for Normalizing then Computing Distance

scale, without further arguments,
normalizes each column

```
# normalize input variables
utilities.df.norm <- sapply(utilities.df, scale)
# add row names: utilities
row.names(utilities.df.norm) <- row.names(utilities.df)
# compute normalized distance based on Sales (column
  6) and Fuel Cost (column 8)
d.norm <- dist(utilities.df.norm[,c(6,8)], method =
  "euclidean")
```

For Categorical Data: Similarity

To measure the distance between records in terms of two 0/1 variables, create table with counts:

	0	1
0	a	b
1	c	d

Similarity metrics based on this table:

- Matching coef. = $(a+d)/p$
- Jaquard's coef. = $d/(b+c+d)$
 - Use in cases where a matching "1" is much greater evidence of similarity than matching "0" (e.g. "owns Corvette")

Other Distance Measures

- Correlation-based similarity
- Statistical distance (Mahalanobis)
- Manhattan distance (absolute differences)
- Maximum coordinate distance
- Gower's similarity (for mixed variable types: continuous & categorical)

Measuring Distance Between Clusters

Minimum Distance (Cluster A to Cluster B)

- Also called **single linkage**
- Distance between two clusters is the distance between the pair of records A_i and B_j that are closest

Maximum Distance (Cluster A to Cluster B)

- Also called **complete linkage**
- Distance between two clusters is the distance between the pair of records A_i and B_j that are farthest from each other

Average Distance (Cluster A to Cluster B)

- Also called **average linkage**
- Distance between two clusters is the average of all possible pair-wise distances

Centroid Distance

- Distance between two clusters is the distance between the two cluster centroids.
- Centroid is the vector of variable averages for all records in a cluster

The Hierarchical Clustering Steps (Using Agglomerative Method)

1. Start with n clusters (each record is its own cluster)
2. Merge two closest records into one cluster
3. At each successive step, the two clusters closest to each other are merged

Dendrogram, from bottom up, illustrates the process

Code for Hierarchical Clustering and Plotting the Dendrogram

```
# in hclust() set argument method =  
# to "ward.D", "single", "complete", "average",  
# "median", or "centroid"
```

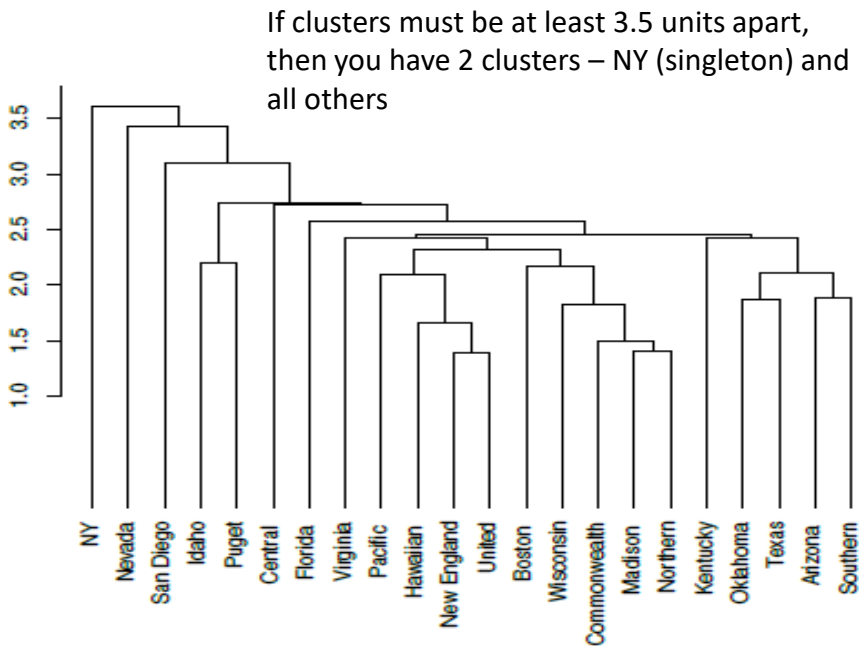
```
hc1 <- hclust(d.norm, method = "single")
```

```
plot(hc1, hang = -1, ann = FALSE)
```

causes x-axis labels to hang
below the 0 line

turns off annotation (plot
and axis titles)

Dendrogram for Single Linkage



Reading the Dendrogram

See process of clustering: Lines connected lower down are merged earlier

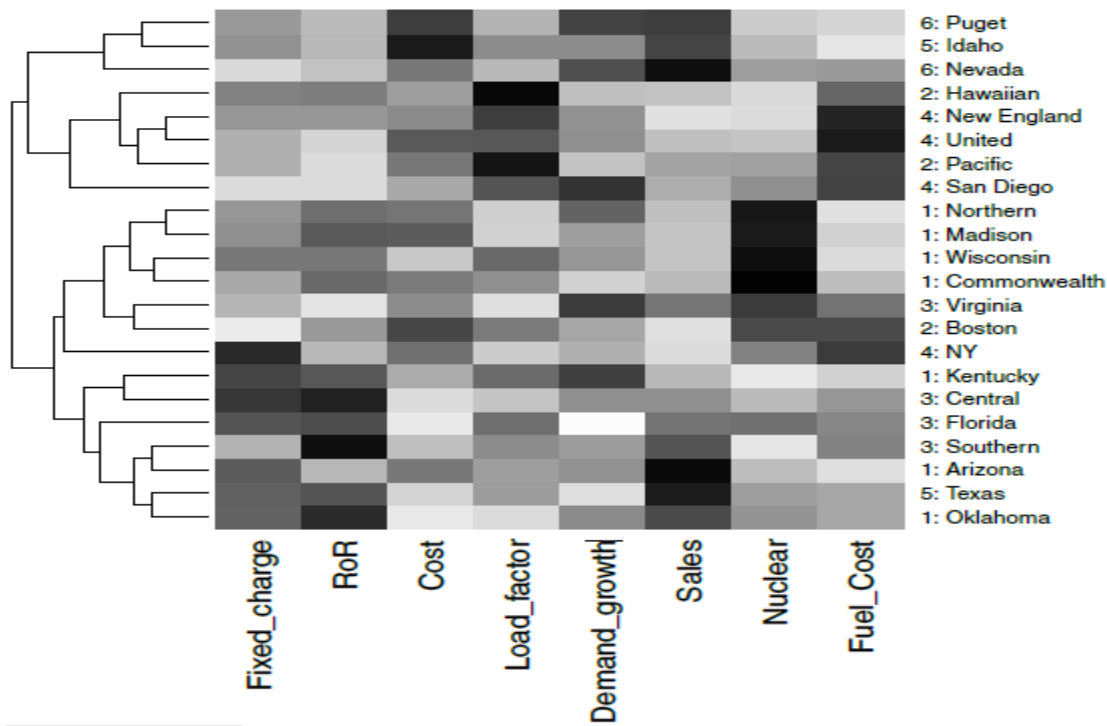
- New England and United will be merged first, then Madison and Northern

Determining number of clusters: For a given “distance between clusters”, a horizontal line intersects the clusters that are that far apart, to create clusters

- E.g., at distance of 3.5, data can be reduced to 2 clusters – NY (singleton cluster) and all others
- At distance of data can be reduced to 6 clusters – 4 singletons, 1 doubleton, and all others

Visualize Cluster Features with Heatmap -(darker = higher values)

Considering 3 clusters,
bottom cluster has higher
Fixed_charge and RoR than
the other two



Interpretation

Goal: obtain meaningful and useful clusters

Caveats:

- (1) Random chance can often produce apparent clusters
- (2) Different cluster methods produce different results

Solutions:

- Obtain summary statistics
- Also review clusters in terms of variables **not** used in clustering
- Label the cluster (e.g. clustering of financial firms in 2008 might yield label like “midsize, sub-prime loser”)

Desirable Cluster Features

Stability – are clusters and cluster assignments sensitive to slight changes in inputs? Are cluster assignments in partition B similar to partition A?

Separation – check ratio of between-cluster variation to within-cluster variation (higher is better)

Nonhierarchical Clustering: K-Means Clustering

Algorithm

1. Choose # of clusters desired, k
2. Start with a partition into k clusters
Often based on random selection of k centroids
3. At each step, move each record to cluster with closest centroid
4. Recompute centroids, repeat step 3
5. Stop when moving records increases within-cluster dispersion

K-means Algorithm: Choosing k and Initial Partitioning

Choose k based on the how results will be used

e.g., “How many market segments do we want?”

Also experiment with slightly different k 's

Initial partition into clusters can be random, or based on domain knowledge

If random partition, repeat the process with different random partitions

Code for K-means Clustering (data prep as for hierarchical)

ask for 6 clusters

```
km <- kmeans(utilities.df.norm, 6)
```

```
# show cluster membership
```

```
km$cluster
```

Output

```
> km$cluster
  Arizona    Boston    Central Commonwealth    NY
      3         2         3         4         1
Florida    Hawaiian    Idaho    Kentucky    Madison
      3         2         6         3         4
Nevada    New England    Northern    Oklahoma    Pacific
      6         2         4         3         2
Puget    San Diego    Southern    Texas    Wisconsin
      6         5         3         3         4
United    Virginia
      2         4
```

Cluster Characteristics

Centroids = each cluster has a vector of variable means

```
> km$centers
  Fixed_charge      RoR      Cost Load_factor Demand_growth
1  2.03732429 -0.8628882  0.5782326 -1.2950193 -0.7186431
2 -0.35819462 -0.3637904  0.3985832  1.1572643 -0.3017426
3  0.50431607  0.7795509 -0.9858961 -0.3375463 -0.4895769
4 -0.01133215  0.3313815  0.2189339 -0.3580408  0.1664686
5 -1.91907572 -1.9323833 -0.7812761  1.1034665  1.8468982
6 -0.60027572 -0.8331800  1.3389101 -0.4805802  0.9917178

  Sales      Nuclear Fuel_Cost
1 -1.5814284  0.2143888  1.6926380
2 -0.7080723 -0.4025746  1.1171999
3  0.3518600 -0.5232108 -0.4105368
4 -0.4018738  1.5650384 -0.5954476
5 -0.9014253 -0.2203441  1.4696557
6  1.8565214 -0.7146294 -0.9657660

> # within-cluster sum of squares
> km$withinss
[1] 0.000000 11.935249 26.507769 10.177094 0.000000 9.533522

> # cluster size
> km$size
[1] 1 5 7 5 1 3
```

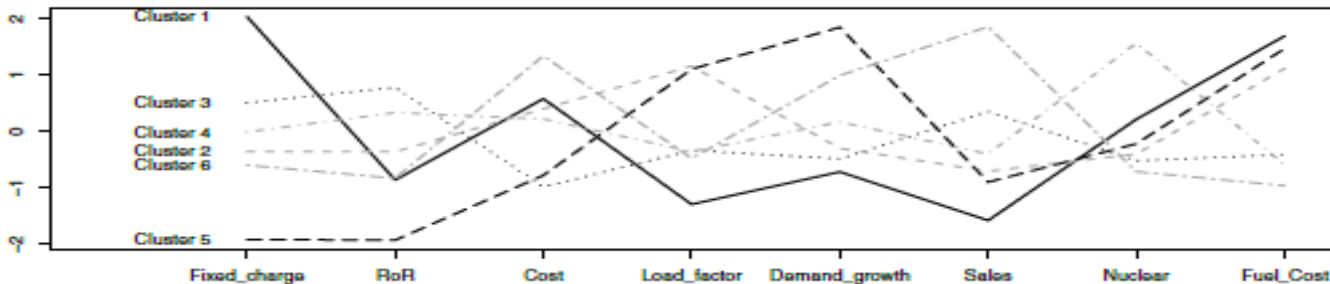
Cluster 1 has normalized average demand growth of -0.7186

Clusters 1 and 5 are each singletons, so within-cluster distance = 0

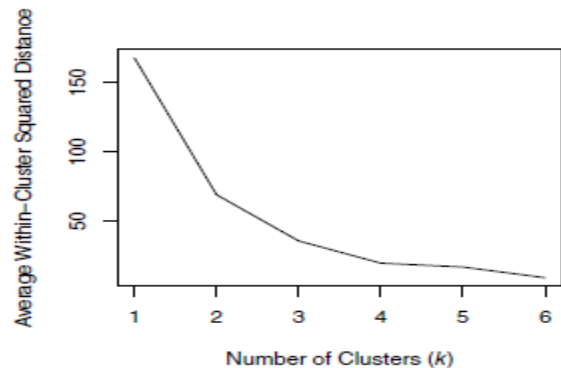
number of utilities in
each cluster

Visualize Cluster Centroids with Profile Plot

```
# plot an empty scatter plot
plot(c(0), xaxt = 'n', ylab = "", type = "l", ylim =
      c(min(km$centers), max(km$centers)), xlim = c(0, 8))
# label x-axes
axis(1, at = c(1:8), labels = names(utilities.df))
# plot centroids
for (i in c(1:6))
  lines(km$centers[i,], lty = i, lwd = 2, col = ifelse(i %in% c(1, 3,
5), "black", "dark grey"))
# nam
text(c(1:6)))
```



Different Choices for K



As the number of clusters increases, the cluster members are closer to one another.

Question: What happens to the distance *between* clusters?

Summary

- Cluster analysis is an exploratory tool. Useful only when it produces **meaningful** clusters
- **Hierarchical** clustering gives visual representation of different levels of clustering
 - On other hand, due to non-iterative nature, it can be unstable, can vary highly depending on settings, and is computationally expensive
- **Non-hierarchical** is computationally cheap and more stable; requires user to set k
- Can use both methods
- Be wary of chance results; data may not have definitive “real” clusters

Association Rules

What are Association Rules?

- Study of “what goes with what”
 - “Customers who bought X also bought Y”
 - What symptoms go with what diagnosis
- Transaction-based or event-based
- Also called “market basket analysis” and “affinity analysis”
- Originated with study of customer transactions databases to determine associations among items purchased

Generating Rules : Terminology used

“IF” part = **antecedent**

“THEN” part = **consequent**

“Item set” = the items (e.g., products) comprising the antecedent or consequent

- Antecedent and consequent are *disjoint* (i.e., have no items in common)

Example: Used in many recommender systems

Bound Away
[Last Train Home](#)



List Price: \$16.98

Price: **\$16.98** and eligible for **FREE Super Saver Shipping** on orders over \$25. [See details.](#)

Availability: Usually ships within 24 hours

Want it delivered Tomorrow? Order it in the next 4 hours and 9 minutes, and choose **One-Day S** checkout. [See details.](#)

41 used & new from \$6.99

► [See more product details](#)

[Share your own customer images](#)

 Based on customer purchases, this is the #82 [Early Adopter Product in Alternative Rock.](#)

801x612

Buy this title for only \$.01 when you get a new Amazon Visa® Card

Apply now and if you're approved instantly, **save \$30** off your first purchase, earn **3% rewards**, get a **0% APR,*** and pay no



Amazon Visa discount: \$30.00

Applied to this item: - \$16.97

Discount remaining: \$13.03

 [Find out how](#)

[\(Don't show again\)](#)

Customers who bought this title also bought:

- [Time and Water](#) ~ Last Train Home (👁 [why?](#))
- [Cold Roses](#) ~ Ryan Adams & the Cardinals (👁 [why?](#))
- [Tambourine](#) ~ Tift Merritt (👁 [why?](#))
- [Last Train Home](#) ~ Last Train Home (👁 [why?](#))
- [True North](#) ~ Last Train Home (👁 [why?](#))
- [Universal United House of Prayer](#) ~ Buddy Miller (👁 [why?](#))
- [Wicked Twisted Road \[ENHANCED\]](#) ~ Reckless Kelly (👁 [why?](#))
- [Hacienda Brothers](#) ~ Hacienda Brothers (👁 [why?](#))

Example: Many Rules are Possible

Transaction	Faceplate Colors Purchased				
1	red	white	green		
2	white	orange			
3	white	blue			
4	red	white	orange		
5	red	blue			
6	white	blue			
7	white	orange			
8	red	white	blue	green	
9	red	white	blue		
10	yellow				



For example: Transaction 1 supports several rules, such as

- “If red, then white” (“If a red faceplate is purchased, then so is a white one”)
- “If white, then red”
- “If red and white, then green”
- + several more

Frequent Item Sets & Support

- Ideally, we want to create all possible combinations of items
- **Problem:** computation time grows exponentially as # items increases
- **Solution:** consider only “frequent item sets
- Criterion for frequent: *support*

Support for an itemset = # (or percent) of transactions that include an itemset

- Example: support for the item set {red, white} is 4 out of 10 transactions, or 40%

Support for a rule = # (or percent) of transactions that include both the antecedent and the consequent

Apriori Algorithm

Generating Frequent Item Sets

For k products...

1. User sets a minimum support criterion
2. Next, generate list of one-item sets that meet the support criterion
3. Use the list of one-item sets to generate list of two-item sets that meet the support criterion
4. Use list of two-item sets to generate list of three-item sets
5. Continue up through k -item sets

Measures of Rule Performance

Confidence: the % of antecedent transactions that also have the consequent item set

Lift = confidence/(benchmark confidence)

Benchmark confidence = transactions with consequent as % of all transactions

Lift > 1 indicates a rule that is useful in finding consequent items sets (i.e., more useful than just selecting transactions randomly)

Alternate Data Format: Binary Matrix

Transaction	Faceplate Colors Purchased				
1	red	white	green		
2	white	orange			
3	white	blue			
4	red	white	orange		
5	red	blue			
6	white	blue			
7	white	orange			
8	red	white	blue	green	
9	red	white	blue		
10	yellow				

← **Original Data**

Binary Martix →

Transaction	Red	White	Blue	Orange	Green	Yellow
1	1	1	0	0	1	0
2	0	1	0	1	0	0
3	0	1	1	0	0	0
4	1	1	0	1	0	0
5	1	0	1	0	0	0
6	0	1	1	0	0	0
7	1	0	1	0	0	0
8	1	1	1	0	1	0
9	1	1	1	0	0	0
10	0	0	0	0	0	1

Support for Various Itemsets

Transaction	Faceplate Colors Purchased				
1	red	white	green		
2	white	orange			
3	white	blue			
4	red	white	orange		
5	red	blue			
6	white	blue			
7	white	orange			
8	red	white	blue	green	
9	red	white	blue		
10	yellow				

TABLE 14.3

**ITEMSETS WITH SUPPORT
COUNT OF AT LEAST TWO**

Itemset	Support (Count)
{red}	6
{white}	7
{blue}	6
{orange}	2
{green}	2
{red, white}	4
{red, blue}	4
{red, green}	2
{white, blue}	4
{white, orange}	2
{white, green}	2
{red, white, blue}	2
{red, white, green}	2

Process of Rule Selection

Generate all rules that meet specified support & confidence

- Find frequent item sets (those with sufficient support – see above)
- From these item sets, generate rules with sufficient confidence

Example: Rules from {red, white, green}

$\{\text{red}, \text{white}\} > \{\text{green}\}$ with confidence = $2/4 = 50\%$

- $[(\text{support } \{\text{red}, \text{white}, \text{green}\})/(\text{support } \{\text{red}, \text{white}\})]$

$\{\text{red}, \text{green}\} > \{\text{white}\}$ with confidence = $2/2 = 100\%$

- $[(\text{support } \{\text{red}, \text{white}, \text{green}\})/(\text{support } \{\text{red}, \text{green}\})]$

Plus 4 more with confidence of 100%, 33%, 29% & 100%

If confidence criterion is 70%, report only rules 2, 3 and 6

Generating Rules in R

```
fp.df <- read.csv("Faceplate.csv")
# remove first column and convert to matrix
fp.mat <- as.matrix(fp.df[, -1])
# convert the binary incidence matrix into a transactions database
fp.trans <- as(fp.mat, "transactions")
inspect(fp.trans)
## get rules
# when running apriori(), include minimum support & confidence, & target as arguments.
rules <- apriori(fp.trans, parameter = list(supp = 0.2, conf = 0.5, target = "rules"))
```

output (sorted by lift)

	lhs	rhs	support	confidence	lift
15	{Red,White}	=> {Green}	0.2	0.5	2.500000
5	{Green}	=> {Red}	0.2	1.0	1.666667
14	{White,Green}	=> {Red}	0.2	1.0	1.666667
4	{Orange}	=> {White}	0.2	1.0	1.428571
6	{Green}	=> {White}	0.2	1.0	1.428571
13	{Red,Green}	=> {White}	0.2	1.0	1.428571

Interpretation

- **Lift ratio** shows how effective the rule is in finding consequents (useful if finding particular consequents is important)
- **Confidence** shows the rate at which consequents will be found (useful in learning costs of promotion)
- **Support** measures overall impact

Caution: The Role of Chance

- Random data can generate apparently interesting association rules.
- The more rules you produce, the greater this danger.
- Rules based on large numbers of records are less subject to this danger.

Rules from some randomly-generated transactions:

	lhs	rhs	support	confidence	lift
[18]	{item.2}	=> {item.9}	0.08	0.8	1.481481
[89]	{item.2,item.7}	=> {item.9}	0.04	1.0	1.851852
[104]	{item.3,item.4}	=> {item.8}	0.04	1.0	1.851852
[105]	{item.3,item.8}	=> {item.4}	0.04	1.0	5.000000
[113]	{item.3,item.7}	=> {item.9}	0.04	1.0	1.851852
[119]	{item.1,item.5}	=> {item.8}	0.04	1.0	1.851852
[149]	{item.4,item.5}	=> {item.9}	0.04	1.0	1.851852
[155]	{item.5,item.7}	=> {item.9}	0.06	1.0	1.851852
[176]	{item.6,item.7}	=> {item.8}	0.06	1.0	1.851852

Even chance data can produce high lift

Example: Charles Book Club

TABLE 14.7 SUBSET OF BOOK PURCHASE TRANSACTIONS IN BINARY MATRIX FORMAT

ChildBks	YouthBks	CookBks	DoItYBks	cefBks	ArtBks	GeogBks	ItalCook	ItalAtlas	ItalArt	Florence
0	1	0	1	0	0	1	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	1	0	1	0	0	0	0
0	0	1	0	0	0	1	0	0	0	0
1	0	0	0	0	1	0	0	0	0	1
0	1	0	0	0	0	0	0	0	0	0
0	1	0	0	1	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0
1	1	1	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

Row 1, e.g., is a transaction in which books were bought in the following categories:
Youth, Do it Yourself, Geography

Rules Produced by apriori

Output

```
> inspect(sort(rules, by = "lift"))
  lhs                rhs      support confidence lift
16 {DoItYBks,GeogBks} => {YouthBks} 0.05450 0.5396040 2.264864
18 {CookBks,GeogBks}  => {YouthBks} 0.08025 0.5136000 2.155719
13 {CookBks,RefBks}  => {DoItYBks} 0.07450 0.5330948 2.092619
14 {YouthBks,GeogBks} => {DoItYBks} 0.05450 0.5215311 2.047227
20 {YouthBks,CookBks} => {DoItYBks} 0.08375 0.5201863 2.041948
10 {YouthBks,RefBks}  => {CookBks}  0.06825 0.8400000 2.021661
15 {YouthBks,DoItYBks} => {GeogBks} 0.05450 0.5278450 1.978801
19 {YouthBks,DoItYBks} => {CookBks} 0.08375 0.8111380 1.952197
12 {DoItYBks,RefBks}  => {CookBks} 0.07450 0.8054054 1.938400
11 {RefBks,GeogBks}   => {CookBks} 0.06450 0.7889908 1.898895
17 {YouthBks,GeogBks} => {CookBks} 0.08025 0.7679426 1.848237
21 {DoItYBks,GeogBks} => {CookBks} 0.07750 0.7673267 1.846755
 7 {YouthBks,ArtBks}  => {CookBks} 0.05150 0.7410072 1.783411
 9 {DoItYBks,ArtBks}  => {CookBks} 0.05300 0.7114094 1.712177
 3 {RefBks}           => {CookBks} 0.13975 0.6825397 1.642695
 8 {ArtBks,GeogBks}   => {CookBks} 0.05525 0.6800000 1.636582
 4 {YouthBks}         => {CookBks} 0.16100 0.6757608 1.626380
 6 {DoItYBks}         => {CookBks} 0.16875 0.6624141 1.594258
 1 {ItalCook}         => {CookBks} 0.06875 0.6395349 1.539193
 5 {GeogBks}          => {CookBks} 0.15625 0.5857545 1.409758
 2 {ArtBks}           => {CookBks} 0.11300 0.5067265 1.219558
```

Cautions:

Duplication (same trio of books)

No useful info!

```
rules <- apriori(books.trans,
parameter = list(supp= 200/4000, conf = 0.5, target = "rules"))
```


Summary – Association Rules

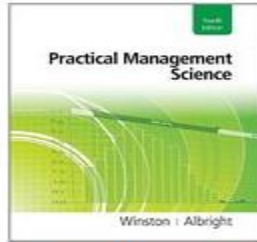
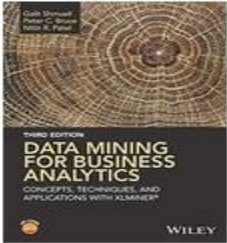
- Association rules (or *affinity analysis*, or *market basket analysis*) produce rules on associations between items from a database of transactions
- Widely used in **recommender systems**
- Most popular method is **Apriori algorithm**
- To reduce computation, we consider only “frequent” item sets (=support)
- Performance of rules is measured by *confidence* and *lift*
- Can produce a profusion of rules; review is required to identify useful rules and to reduce redundancy

Collaborative Filtering

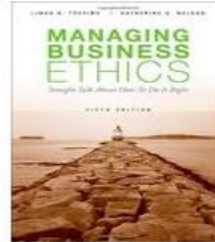
Collaborative Filtering

- User based methods
- Item based methods

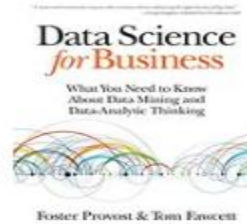
Customers Who Bought This Item Also Bought



Practical Management Science (with Essential Textbook Resources...
› Wayne L. Winston



Managing Business Ethics: Straight Talk about How to Do It Right
Linda K. Trevino



Data Science for Business: What You Need to Know about Data Mining and...
› Foster Provost

Item-user matrix

- Cells are user preferences, r_{ij} , for items
- Preferences can be ratings, or binary (buy, click like)

User ID	Item ID			
	I_1	I_2	\dots	I_p
U_1	$r_{1,1}$	$r_{1,2}$	\dots	$r_{1,p}$
U_2	$r_{2,1}$	$r_{2,2}$	\dots	$r_{2,p}$
\vdots				
U_n	$r_{n,1}$	$r_{n,2}$	\dots	$r_{n,p}$

More efficient to store as rows of triplets

Each row has the user ID, the item ID, and the user's rating of that item

(U_u, I_i, r_{ui})

User-based Collaborative Filtering

- Start with a single user who will be the target of the recommendations
- Find other users who are most similar, based on comparing preference vectors

Measuring Proximity

- Like nearest-neighbor algorithm
- But Euclidean distance does not do well
- Correlation proximity does better (Pearson)
- For each user pair, find the co-rated items, calculate correlation between the vectors of their ratings for those items
 - Note that the average ratings for each user are across all products, not just the co-rated ones

$$\text{Corr}(U_1, U_2) = \frac{\sum (r_{1,i} - \bar{r}_1)(r_{2,i} - \bar{r}_2)}{\sqrt{\sum (r_{1,i} - \bar{r}_1)^2} \sqrt{\sum (r_{2,i} - \bar{r}_2)^2}}$$

Cosine Similarity

- Like correlation coefficient, except do not subtract the means
- “Cold start” problem: For users with just one item, or items with just one neighbor, neither cosine similarity nor correlation produces useful metric
- Binary matrix? Must use all the data, not just the co-rated items.
 - This can add useful info – in the Netflix contest, information about which movies users chose to rate was informative.

Example – Tiny Netflix subset

Customer ID	Movie ID								
	1	5	8	17	18	28	30	44	48
30878	4	1			3	3	4	5	
124105	4								
822109	5								
823519	3		1	4		4	5		
885013	4	5							
893988	3						4	4	
1248029	3					2	4		3
1503895	4								
1842128	4						3		
2238063	3								

TABLE 14.8

**SAMPLE OF RECORDS FROM THE NETFLIX PRIZE CONTEST, FOR A
SUBSET OF 10 CUSTOMERS AND 9 MOVIES**

Consider users 30878 and 823519

Correlation between users 30878 and 823519

First find average ratings for each user:

$$\bar{r}_{30878} = (4 + 1 + 3 + 3 + 4 + 5)/6 = 3.333$$

$$\bar{r}_{823519} = (3 + 1 + 4 + 4 + 5)/5 = 3.4$$

Find correlation using departure from avg. ratings for the co-rated movies (movies 1, 28 and 30):

$$\begin{aligned}\text{Corr}(U_{30878}, U_{823519}) &= \\ &= \frac{(4 - 3.333)(3 - 3.4) + (3 - 3.333)(4 - 3.4) + (4 - 3.333)(5 - 3.4)}{\sqrt{(4 - 3.333)^2 + (3 - 3.333)^2 + (4 - 3.333)^2} \sqrt{(3 - 3.4)^2 + (4 - 3.4)^2 + (5 - 3.4)^2}} \\ &= 0.6/1.75 = 0.34\end{aligned}$$

Find cosine similarity for same users

Use raw ratings instead of departures from averages:

$$\begin{aligned}\text{Cos Sim}(U_{30878}, U_{823519}) &= \frac{4 \times 3 + 3 \times 4 + 4 \times 5}{\sqrt{4^2 + 3^2 + 4^2} \sqrt{3^2 + 4^2 + 5^2}} \\ &= 44/45.277 = 0.972\end{aligned}$$

Ranges from 0 (no similarity) to 1 (perfect match)

Using the similarity info to make recommendations

- Given a new user, identify k-nearest users
- Consider all the items they rated/purchased, except for the co-rated ones
- Among these other items, what is the best one? “Best” could be
 - Most purchased
 - Highest rated
 - Most rated
- That “best” item is the recommendation for the new user

Item-based collaborative filtering

- When the number of users is huge, user-based calculations pose an obstacle (similarity measures cannot be calculated until user shows up)
- Alternative – when a user purchases an item, focus on similar items
 1. Find co-rated (co-purchased) items (by any user)
 2. Recommend the most popular or most correlated item

R Code with recommenderlab

```
# simulate matrix with 1000 users and 100 movies
m <- matrix(nrow = 1000, ncol = 100)

# simulated ratings (1% of the data)
m[sample.int(100*1000, 1000)] <- ceiling(runif(1000, 0, 5))

## convert into a realRatingMatrix
r <- as(m, "realRatingMatrix")
library(recommenderlab)

# user-based collaborative filtering
UB.Rec <- Recommender(r, "UBCF")
pred <- predict(UB.Rec, r, type="ratings")
as(pred, "matrix")

# item-based collaborative filtering
IB.Rec <- Recommender(r, "IBCF")
pred <- predict(IB.Rec, r, type="ratings")
as(pred, "matrix")
```

Summary – Collaborative Filtering

- User-based – for a new user, find other users who share his/her preferences, recommend the highest-rated item that new user does not have.
 - User-user correlations cannot be calculated until new user appears on the scene... so it is slow if lots of users
- Item-based – for a new user considering an item, find other item that is most similar in terms of user preferences.
 - Ability to calculate item-item correlations in advance greatly speeds up the algorithm

Association Rules vs. Collaborative Filtering

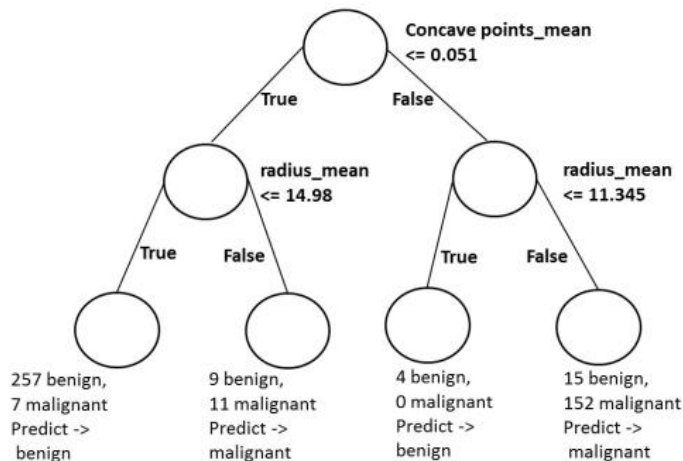
- AR: focus entirely on frequent (popular) item combinations. Data rows are single transactions. Ignores user dimension. Often used in displays (what goes with what).
- CF: focus is on user preferences. Data rows are user purchases or ratings over time. Can capture “long tail” of user preferences – useful for recommendations involving unusual items

Classification and Regression Trees

CART - Explained

There are two major advantages in using tree models,

- They are able to capture the non-linearity in the dataset.
- No need for Standardization of data when using tree models. Because they don't calculate any euclidean distance or other measuring factors between data. Just if-else.



Decision Tree Classifier

CART - Terminologies

Each of a Decision Tree is known as Nodes.

There are three types of nodes,

- Root Node: doesn't have any parent node, and gives two children nodes based on the question
- Internal Node: it will have a parent node, and gives two children nodes
- Leaf Node: it will also have a parent node, but won't have any children nodes

The number of levels of Tree has is known as the **max_depth**.

In the above diagram `max_depth = 3`.

As the `max_depth` increases, the model complexity also will increase

Trees and Rules

Goal: Classify or predict an outcome based on a set of predictors

The output is a set of **rules**

Example:

- Goal: classify a record as “will accept credit card offer” or “will not accept”
- Rule might be “IF (Income ≥ 106) AND (Education < 1.5) AND (Family ≤ 2.5) THEN Class = 0 (nonacceptor)”
- Also called CART, Decision Trees, or just Trees
- Rules are represented by tree diagrams

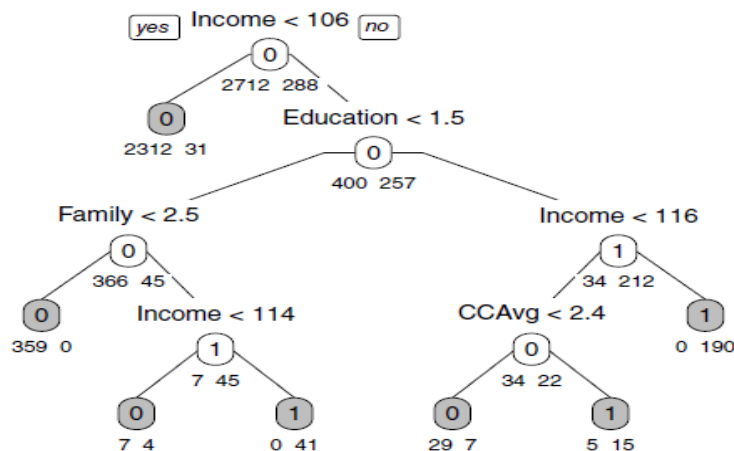


FIGURE 9.1 BEST-PRUNED TREE OBTAINED BY FITTING A FULL TREE TO THE TRAINING DATA

Key Ideas

Recursive partitioning: Repeatedly split the records into two parts so as to achieve maximum homogeneity of outcome within each new part

Pruning the tree: Simplify the tree by pruning peripheral branches to avoid overfitting

Recursive Partitioning - Steps

- Pick one of the predictor variables, x_i
- Pick a value of x_i , say s_i , that divides the training data into two (not necessarily equal) portions
- Measure how “pure” or homogeneous each of the resulting portions is
 - “Pure” = containing records of mostly one class (or, for prediction, records with similar outcome values)
- Algorithm tries different values of x_i and s_i to maximize purity in initial split
- After you get a “maximum purity” split, repeat the process for a second split (on any variable), and so on

Example: Riding Mowers

- Goal: Classify 24 households as owning or not owning riding mowers
- Predictors = Income, Lot Size

library `rpart` for running trees,
function `prp` in library `rpart.plot`
to plot them

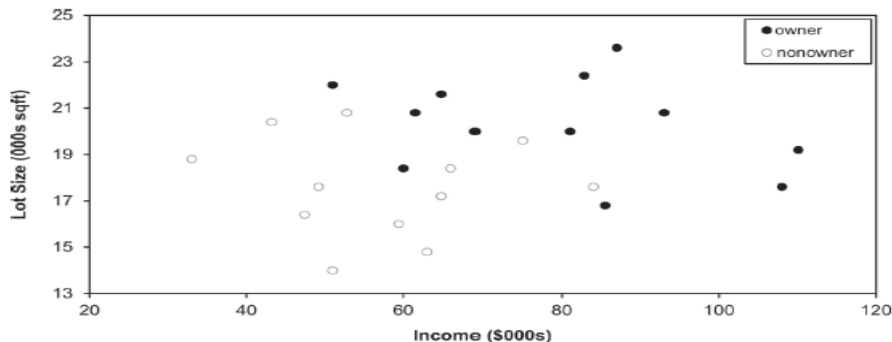


FIGURE 9.2

SCATTER PLOT OF LOT SIZE VS. INCOME FOR 24 OWNERS AND NONOWNERS OF RIDING MOWERS

Income	Lot_Size	Ownership
60.0	18.4	owner
85.5	16.8	owner
64.8	21.6	owner
61.5	20.8	owner
87.0	23.6	owner
110.1	19.2	owner
108.0	17.6	owner
82.8	22.4	owner
69.0	20.0	owner
93.0	20.8	owner
51.0	22.0	owner
81.0	20.0	owner
75.0	19.6	non-owner
52.8	20.8	non-owner
64.8	17.2	non-owner
43.2	20.4	non-owner
84.0	17.6	non-owner
49.2	17.6	non-owner
59.4	16.0	non-owner
66.0	18.4	non-owner
47.4	16.4	non-owner
33.0	18.8	non-owner
51.0	14.0	non-owner
63.0	14.8	non-owner

How to split

- Order records according to one variable, say income
- Take a predictor value, say 60 (the first record) and divide records into those with income ≥ 60 and those < 60
- Measure resulting purity (homogeneity) of class in each resulting portion
- Try all other split values
- Repeat for other variable(s)
- Select the one variable & split that yields the most purity

Note: Categorical Variables

- Examine all possible ways in which the categories can be split.
- E.g., categories A, B, C can be split 3 ways
 - {A} and {B, C}
 - {B} and {A, C}
 - {C} and {A, B}
- With many categories, # of splits becomes huge

The first split: Income = 60

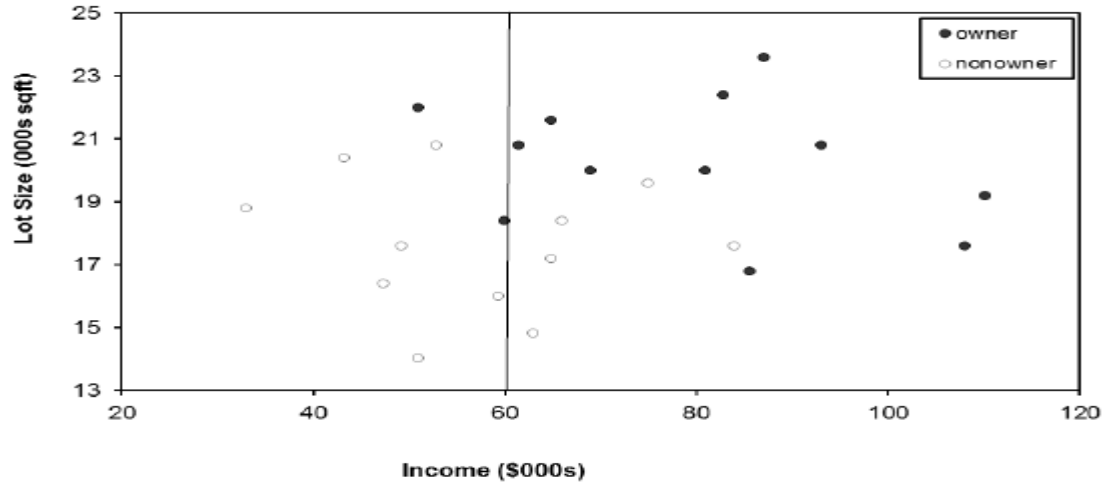


FIGURE 9.3

SPLITTING THE 24 RECORDS BY INCOME VALUE OF 60

Second Split: Lot size = 21

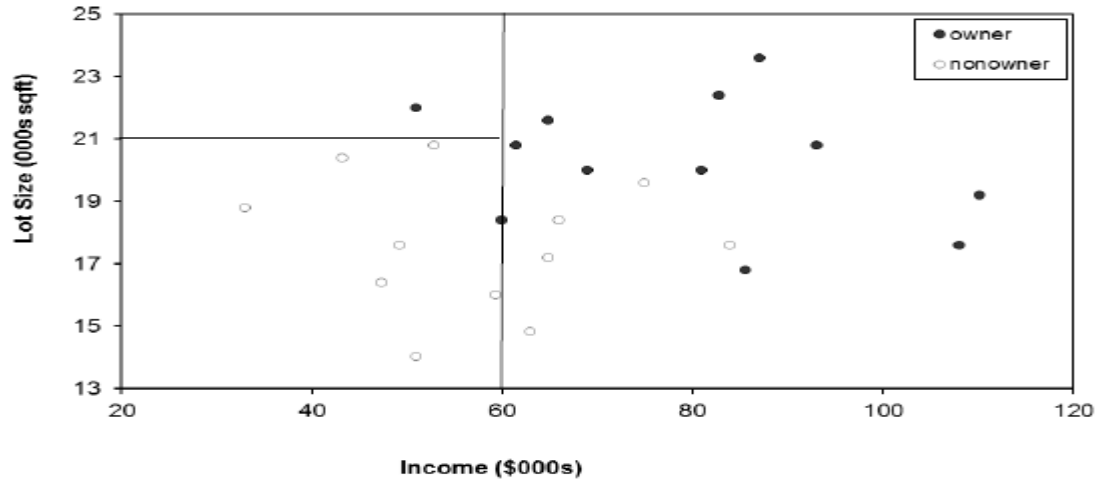


FIGURE 9.5

SPLITTING THE 24 RECORDS FIRST BY INCOME VALUE OF 60 AND THEN LOT SIZE VALUE OF 21

After All Splits

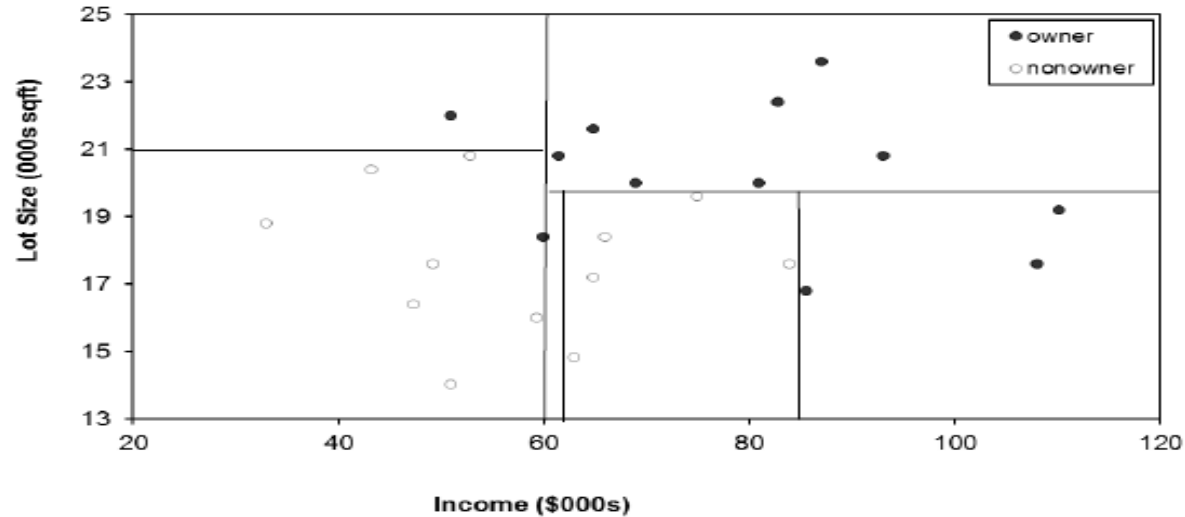


FIGURE 9.6

**FINAL STAGE OF RECURSIVE PARTITIONING; EACH RECTANGLE
CONSISTING OF A SINGLE CLASS (OWNERS OR NONOWNERS)**

Measuring Impurity – Gini Index

Gini Index for rectangle A

$$I(A) = 1 - \sum_{k=1}^m p_k^2$$

p = proportion of cases in rectangle A that belong to class k (out of m classes)

- $I(A) = 0$ when all cases belong to same class
- Max value when all classes are equally represented (= 0.50 in binary case)

Note: XLMiner uses a variant called “delta splitting rule”

Entropy

$$\textit{entropy}(A) = - \sum_{k=1}^m p_k \log_2(p_k)$$

p = proportion of cases in rectangle A that belong to class k (out of m classes)

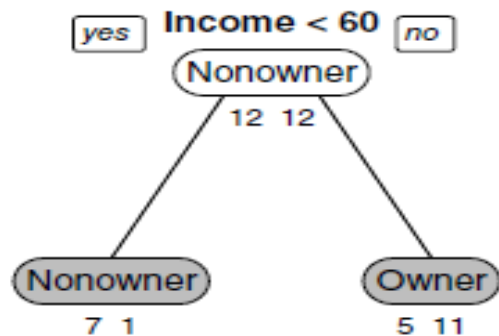
- Entropy ranges between 0 (most pure) and $\log_2(m)$ (equal representation of classes)

Impurity and Recursive Partitioning

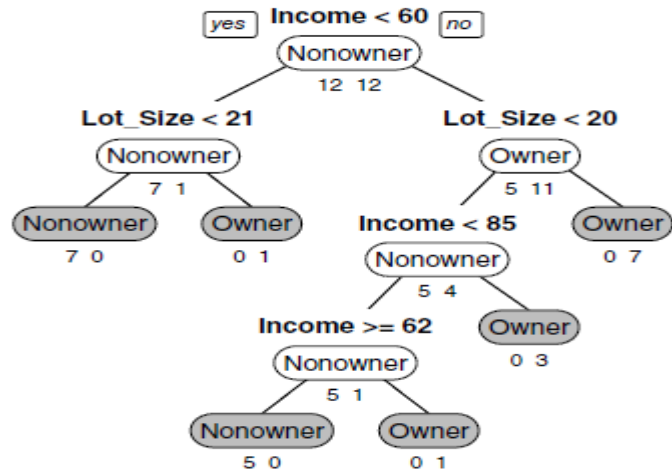
- Obtain overall impurity measure (weighted avg. of individual rectangles)
- At each successive stage, compare this measure across all possible splits in all variables
- Choose the split that reduces impurity the most
- Chosen split points become nodes on the tree

The split

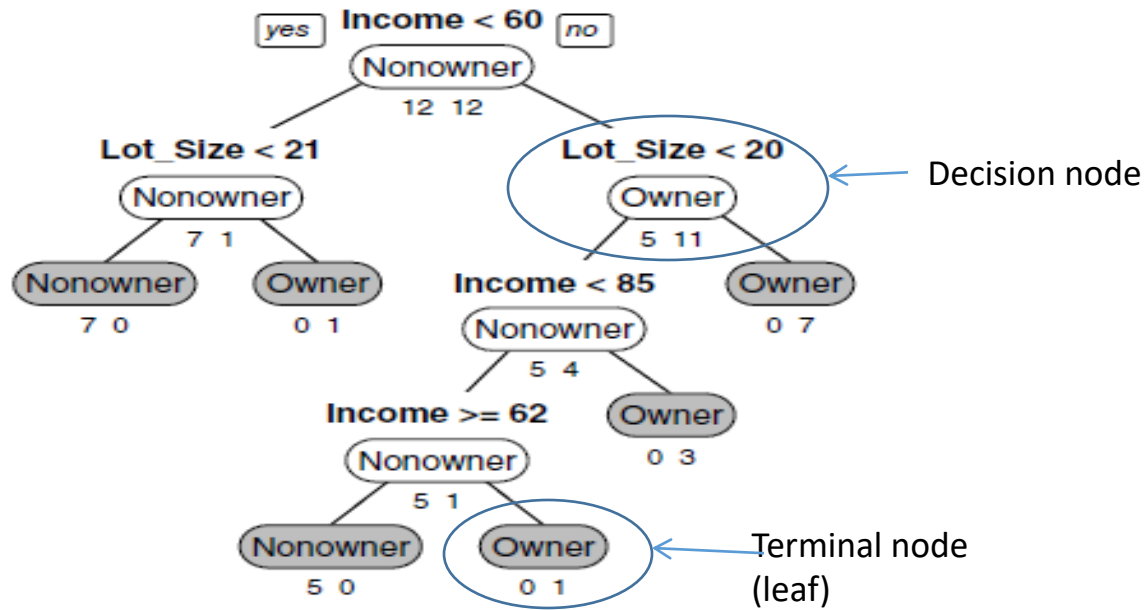
First Split – The Tree

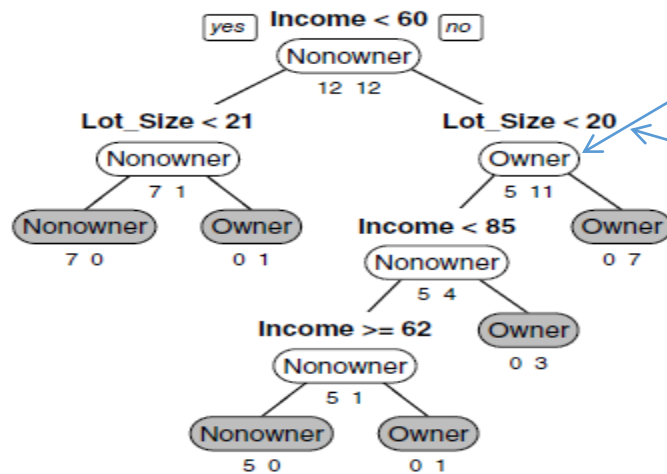


Tree after all splits



The first split is on Income, then the next split is on Lot Size for both the low income group (at lot size 21) and the high income split (at lot size 20)



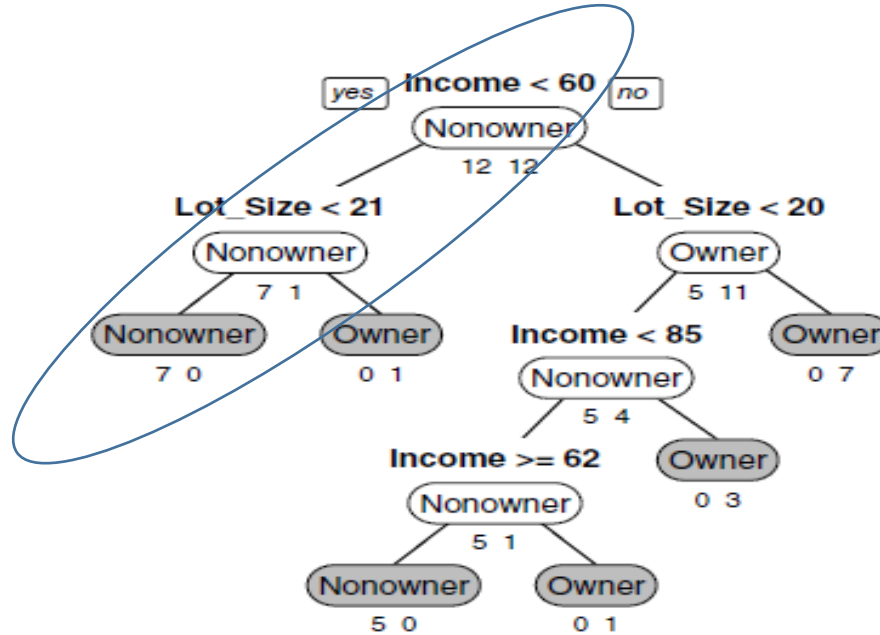


The dominant class in this portion of the first split (those with income ≥ 60) is "owner" – 11 owners and 5 non-owners

The next split for this group of 16 will be on the basis of lot size, splitting at 20

Read down the tree to derive rules

If Income < 60 AND
Lot Size < 21,
classify as “Nonowner”



The Overfitting Problem

Full trees are complex and overfit the data

- Natural end of process is 100% purity in each leaf
- This **overfits** the data, which end up fitting noise in the data
- Consider Example 2, Loan Acceptance with more records and more variables than the Riding Mower data – the full tree is very complex

The Overfitting Problem

Full trees are too complex – they end up fitting noise, overfitting the data

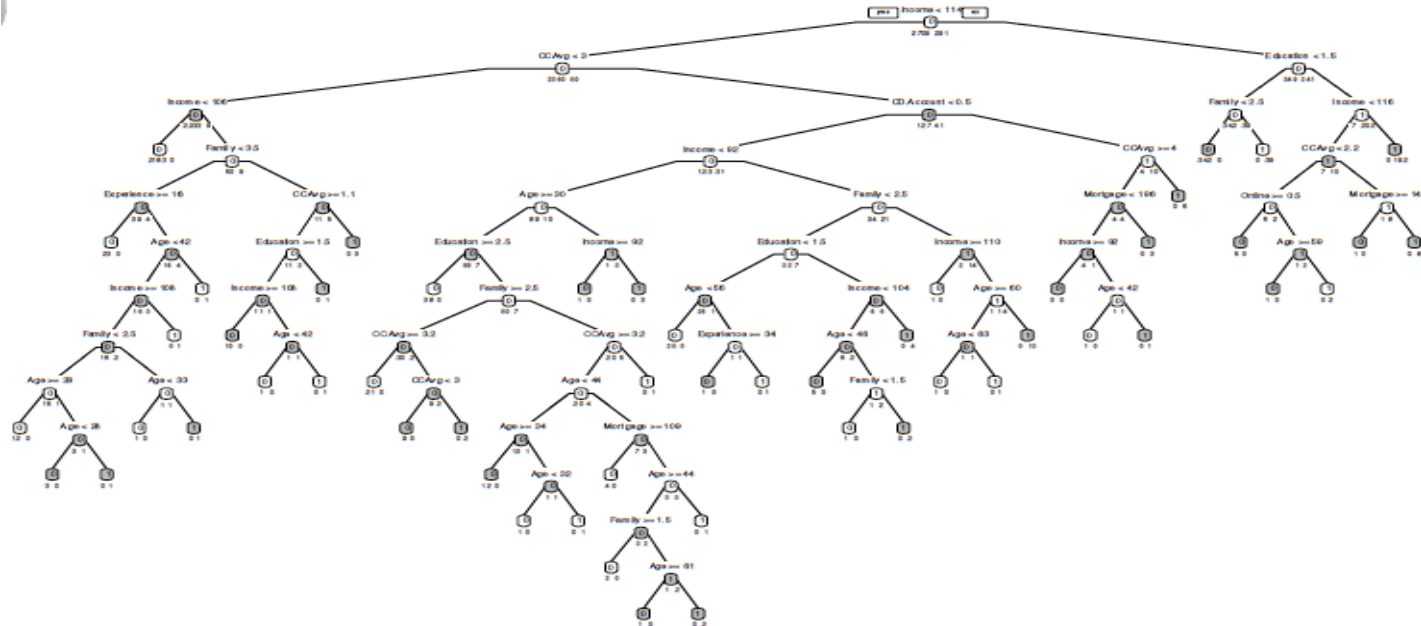
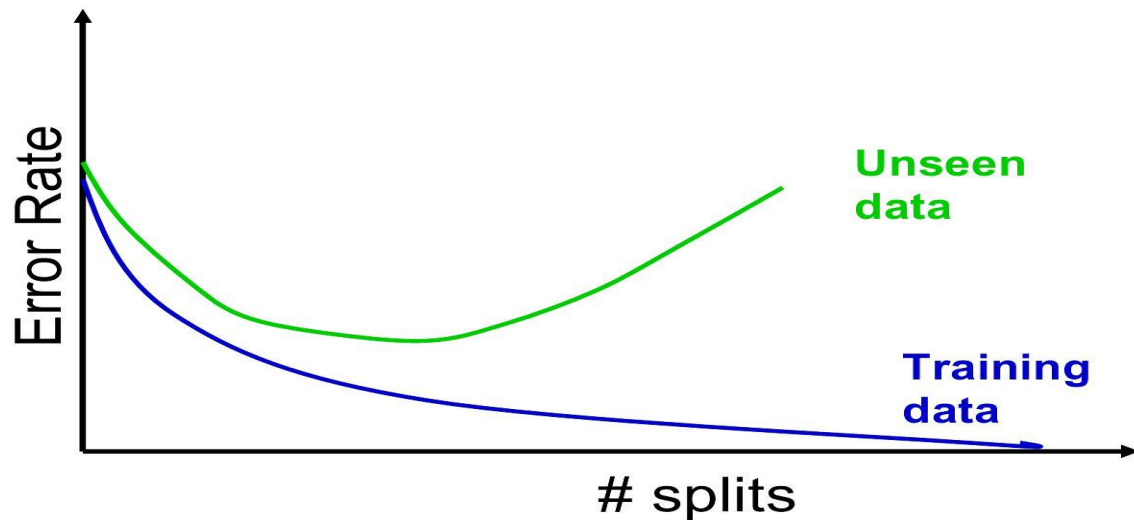


FIGURE 9.10

A FULL TREE FOR THE LOAN ACCEPTANCE DATA USING THE TRAINING SET (3000 RECORDS)

The Overfitting Problem

Overfitting produces poor predictive performance – past a certain point in tree complexity, the error rate on new data starts to increase



Stopping tree growth

CHAID

CHAID, older than CART, uses chi-square statistical test to limit tree growth

Splitting stops when purity improvement is not statistically significant

Pruning

- CART lets tree grow to full extent, then prunes it back
- Idea is to find that point at which the validation error is at a minimum
- Generate successively smaller trees by pruning leaves
- At each pruning stage, multiple trees are possible
- Use *cost complexity* to choose the best tree at that stage



Which branch to cut at each stage of pruning?

$$CC(T) = Err(T) + \alpha L(T)$$

$CC(T)$ = cost complexity of a tree

$Err(T)$ = proportion of misclassified records

α = penalty factor attached to tree size (set by user)

- Among trees of given size, choose the one with lowest CC
- Do this for each size of tree (stage of pruning)

Tree instability



- If 2 or more variables are of roughly equal importance, which one CART chooses for the first split can depend on the initial partition into training and validation
- A different partition into training/validation could lead to a different initial split
- This can cascade down and produce a very different tree from the first training/validation partition
- Solution is to try many different training/validation splits – “cross validation”

Cross validation

- Do many different partitions (“folds*”) into training and validation, grow & pruned tree for each
- Problem: We end up with lots of different pruned trees. Which one to choose?
- Solution: Don’t choose a tree, choose a tree size:
 - For each iteration, record the c_p that corresponds to the minimum validation error
 - Average these c_p ’s
 - With future data, grow tree to that optimum c_p value

*typically folds are non-overlapping, i.e. data used in one validation fold will not be used in others

Cross validation, “best pruned”

- In the above procedure, we select c_p for minimum error tree
- But... simpler is better: slightly smaller tree might do just as well
- Solution: add a cushion to minimum error
 - Calculate standard error of cv estimate – this gives a rough range for chance variation
 - Add standard error to the actual error to allow for chance variation
 - Choose smallest tree within one std. error of minimum error
 - You can then use the corresponding c_p to set c_p for future data

With future data, grow tree to 7 splits:

TABLE 9.4

TABLE OF COMPLEXITY PARAMETER (CP) VALUES AND ASSOCIATED TREE ERRORS



code for tabulating tree error as a function of the complexity parameter (CP)

```
# argument xval refers to the number of folds to use in rpart's built-in
# cross-validation procedure
# argument cp sets the smallest value for the complexity parameter.
cv.ct <- rpart(Personal.Loan ~ ., data = train.df, method = "class",
  cp = 0.00001, minsplit = 5, xval = 5)
# use printcp() to print the table.
printcp(cv.ct)
```

Output

	CP	nsplit	rel error	xerror	xstd
1	0.3350515	0	1.000000	1.00000	0.055705
2	0.1340206	2	0.329897	0.37457	0.035220
3	0.0154639	3	0.195876	0.19931	0.025917
4	0.0068729	7	0.134021	0.17182	0.024096
5	0.0051546	12	0.099656	0.17182	0.024096
6	0.0034364	14	0.089347	0.16838	0.023858
7	0.0022910	19	0.072165	0.17182	0.024096
8	0.0000100	25	0.058419	0.17182	0.024096

estimated
cv error

std. error of
the estimate

smallest tree within 1 xstd of
min. error (it has 7 splits)

minimum
error

Regression Trees

Regression Trees for Prediction

- Used with continuous outcome variable
- Procedure similar to classification tree
- Many splits attempted, choose the one that minimizes impurity

Differences from CT

- Prediction is computed as the **average** of numerical target variable in the rectangle (in CT it is majority vote)
- Impurity measured by **sum of squared deviations** from leaf mean
- Performance measured by RMSE (root mean squared error)

Advantages of trees

- Easy to use, understand
- Produce rules that are easy to interpret & implement
- Variable selection & reduction is automatic
- Do not require the assumptions of statistical models
- Can work without extensive handling of missing data

Disadvantage of single trees: instability and poor predictive performance

Random Forests (library `randomForest`)

1. Draw multiple bootstrap resamples of cases from the data
2. For each resample, use a random subset of predictors and produce a tree
3. Combine the predictions/classifications from all the trees (the “forest”)
 - Voting for classification
 - Averaging for prediction

Boosted Trees (library adabag)

Boosting, like RF, is an ensemble method – but uses an iterative approach in which each successive tree focuses its attention on the misclassified trees from the prior tree.

1. Fit a single tree
2. Draw a bootstrap sample of records with higher selection probability for misclassified records
3. Fit a new tree to the bootstrap sample
4. Repeat steps 2 & 3 multiple times
5. Use weighted voting (classification) or averaging (prediction) with heavier weights for later trees

Summary

- Classification and Regression Trees are an easily understandable and transparent method for predicting or classifying new records
- A single tree is a graphical representation of a set of rules
- Tree growth must be stopped to avoid overfitting of the training data – cross-validation helps you pick the right c_p level to stop tree growth
- Ensembles (random forests, boosting) improve predictive performance, but you lose interpretability and the rules embodied in a single tree

Thank You!