Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Business Analytics using Data Mining

BU7143

Dr. Nicholas P. Danks
Business Analytics

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

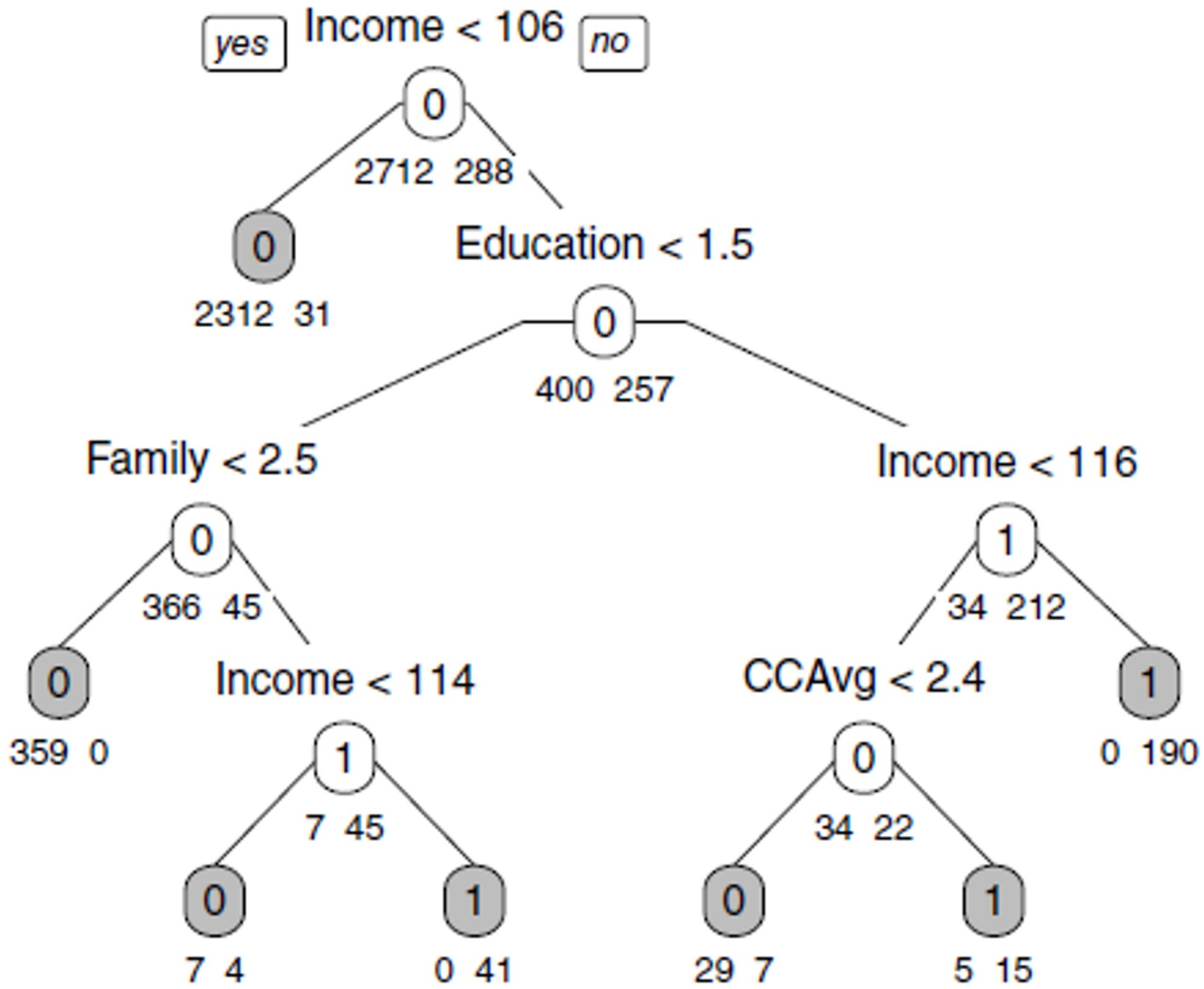


FIGURE 9.1 BEST-PRUNED TREE OBTAINED BY FITTING A FULL TREE TO THE TRAINING DATA

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

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

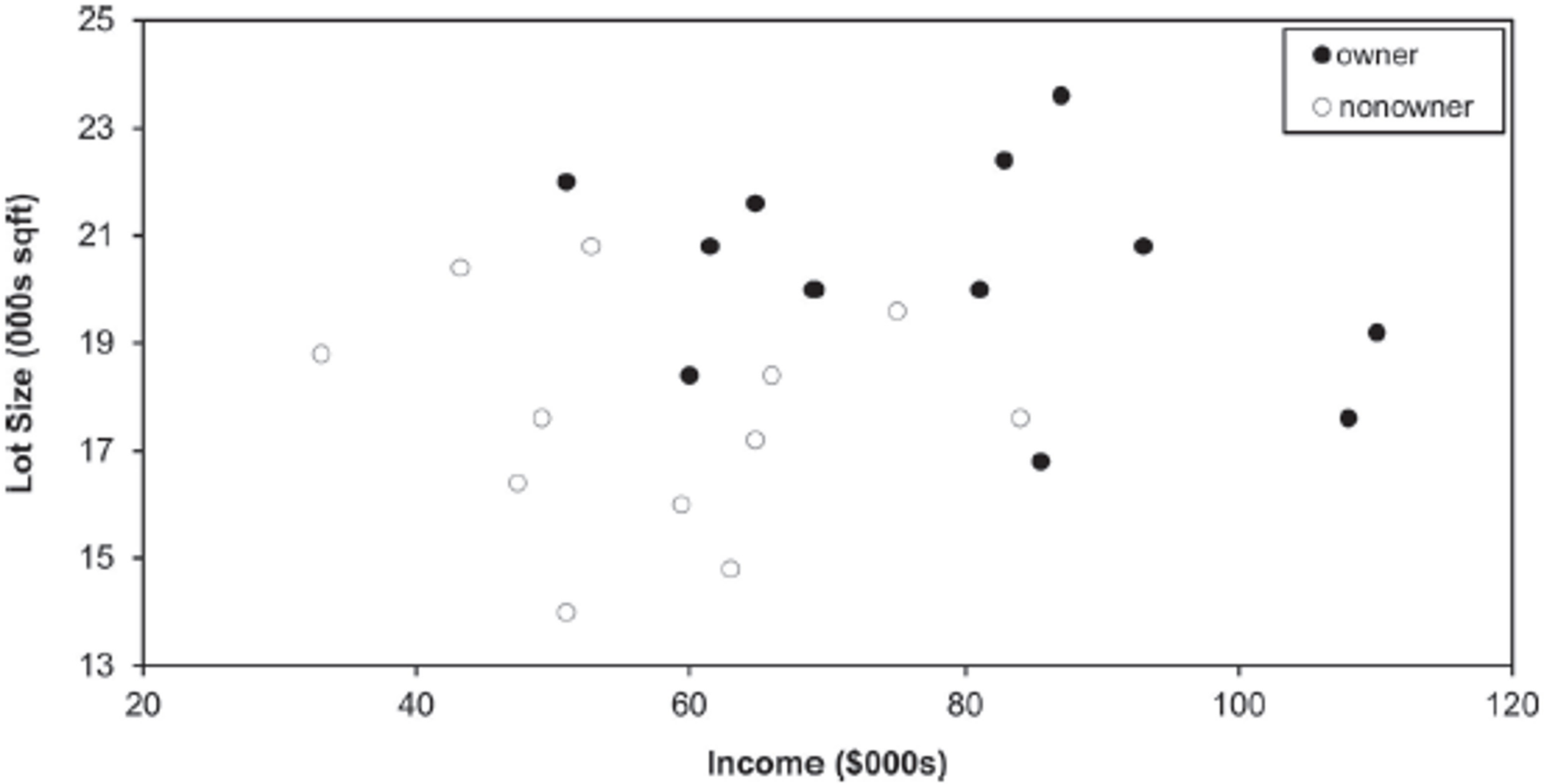


FIGURE 9.2 SCATTER PLOT OF LOT SIZE VS. INCOME FOR 24 OWNERS AND NONOWNERS OF RIDING MOWERS

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

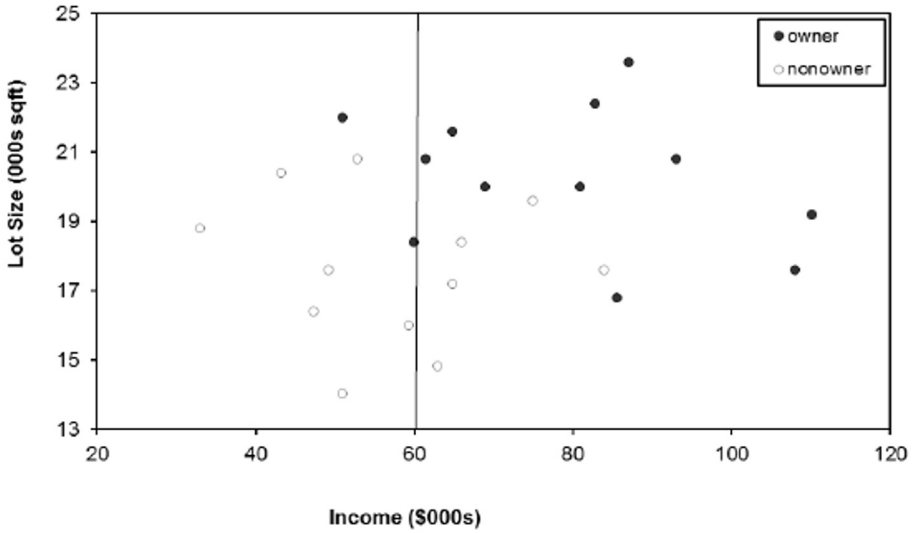


FIGURE 9.3 SPLITTING THE 24 RECORDS BY INCOME VALUE OF 60

Second Split: Lot size = 21

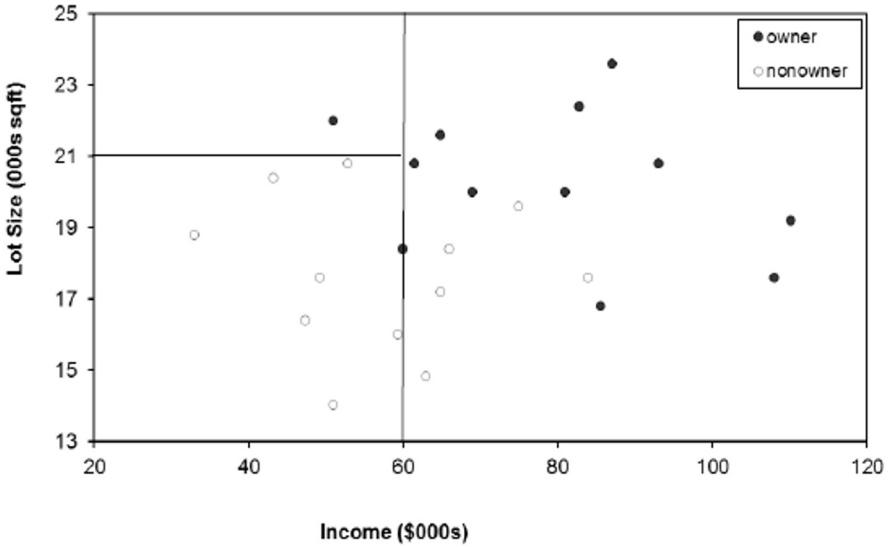


FIGURE 9.5 SPLITTING THE 24 RECORDS FIRST BY INCOME VALUE OF 60 AND THEN LOT SIZE VALUE OF 21

After All Splits

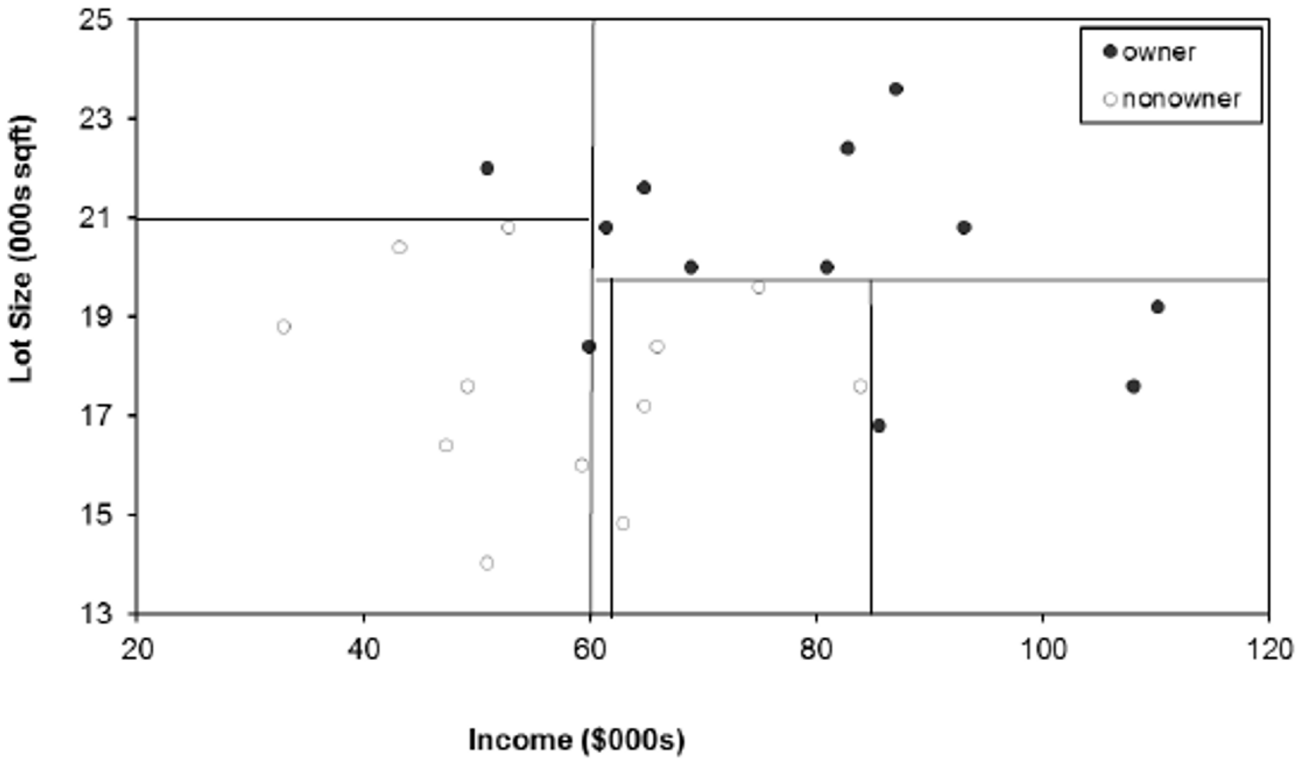


FIGURE 9.6 FINAL STAGE OF RECURSIVE PARTITIONING; EACH RECTANGLE CONSISTING OF A SINGLE CLASS (OWNERS OR NONOWNERS)

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

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

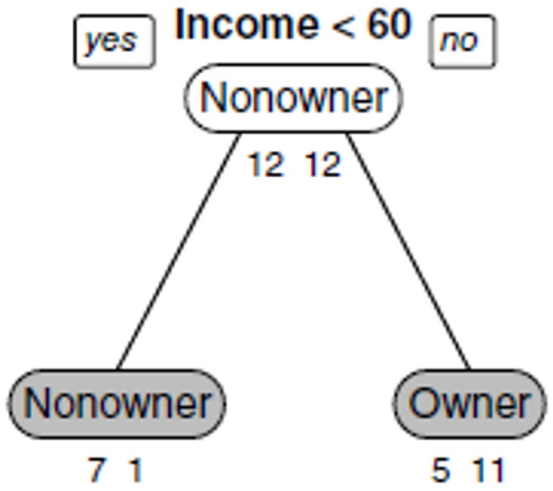
p = proportion of cases in rectangle *A* that belong to class *k* (out of *m* clas*ses*)

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

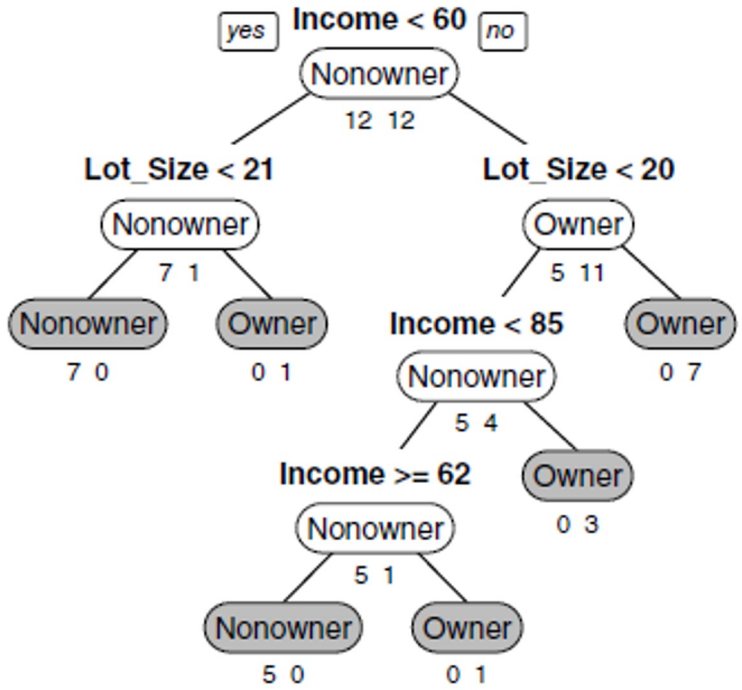
Impurity and Recursive Partitioning

- 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

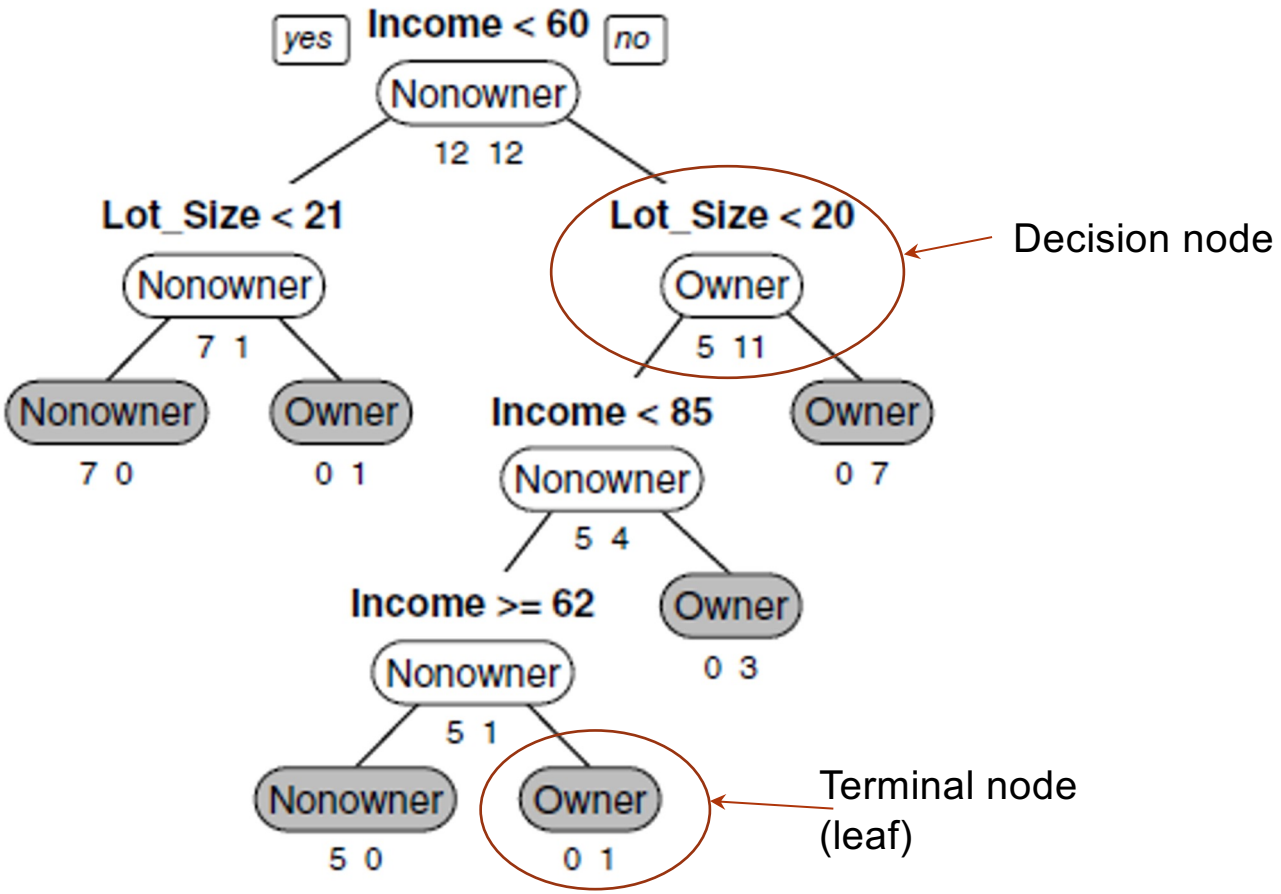
First Split – The Tree

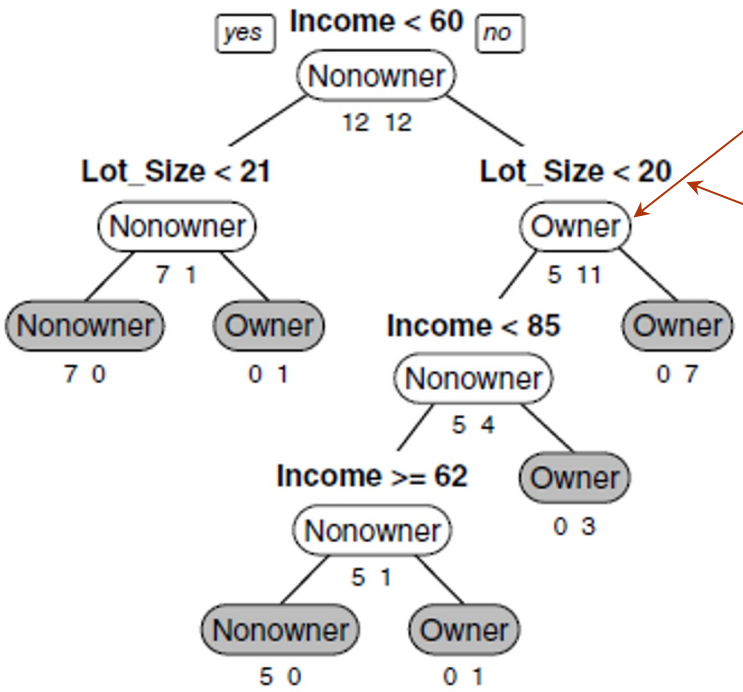


Tree after all splits



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)


yes
Income < 60
no
Nonowner
12 12
Lot_Size < 21
Lot_Size < 20
Nonowner
7 1
Owner
5 11
Nonowner
7 0
Owner
0 1
Income < 85
Owner
0 7
Nonowner
5 4
Income >= 62
Owner
0 3
Nonowner
5 1
Nonowner
5 0
Owner
0 1
Decision node
Terminal node
(leaf)

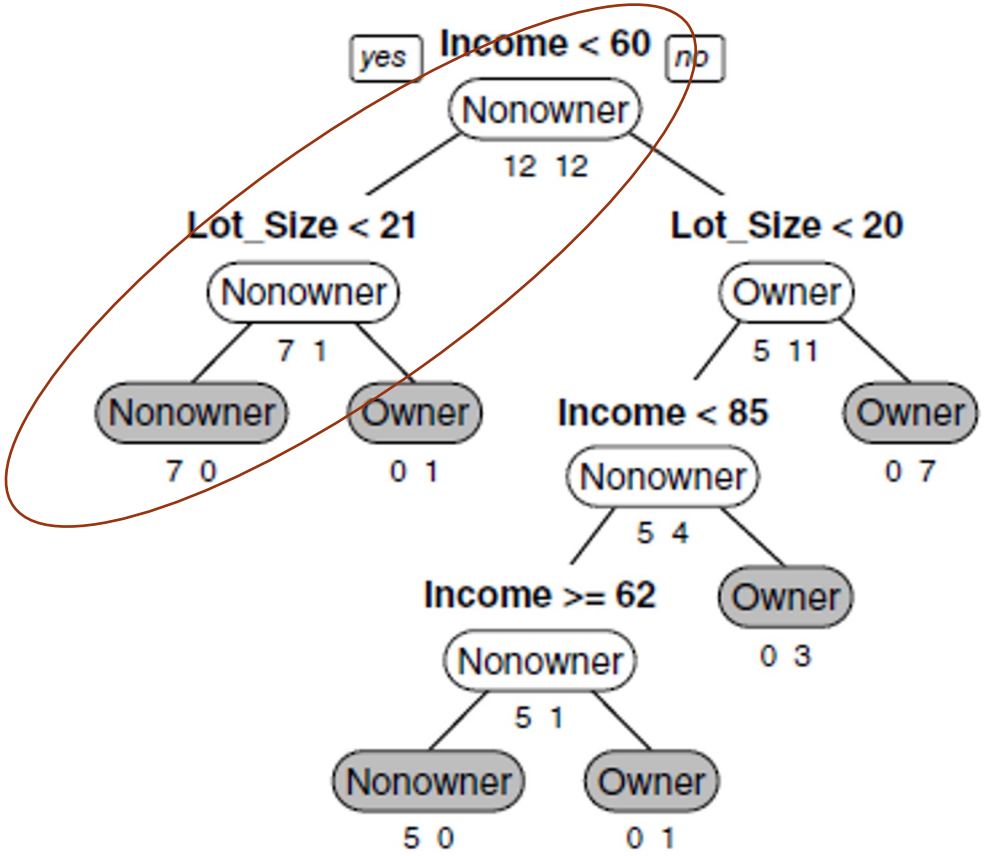


The dominant class in this portion of the first split (those with income >= 60) is “owner” – 11 owners and 5 non-owners

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

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

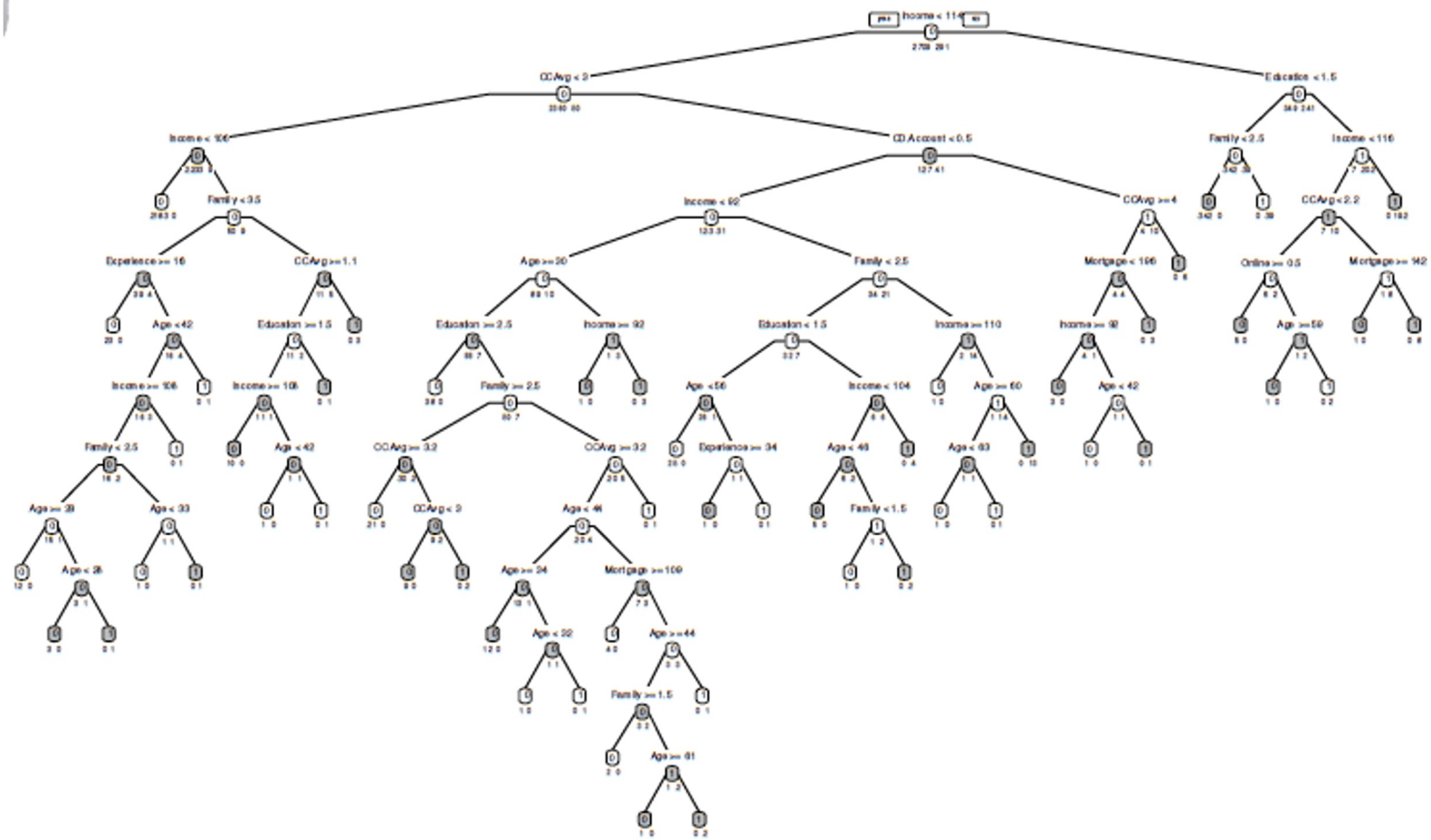
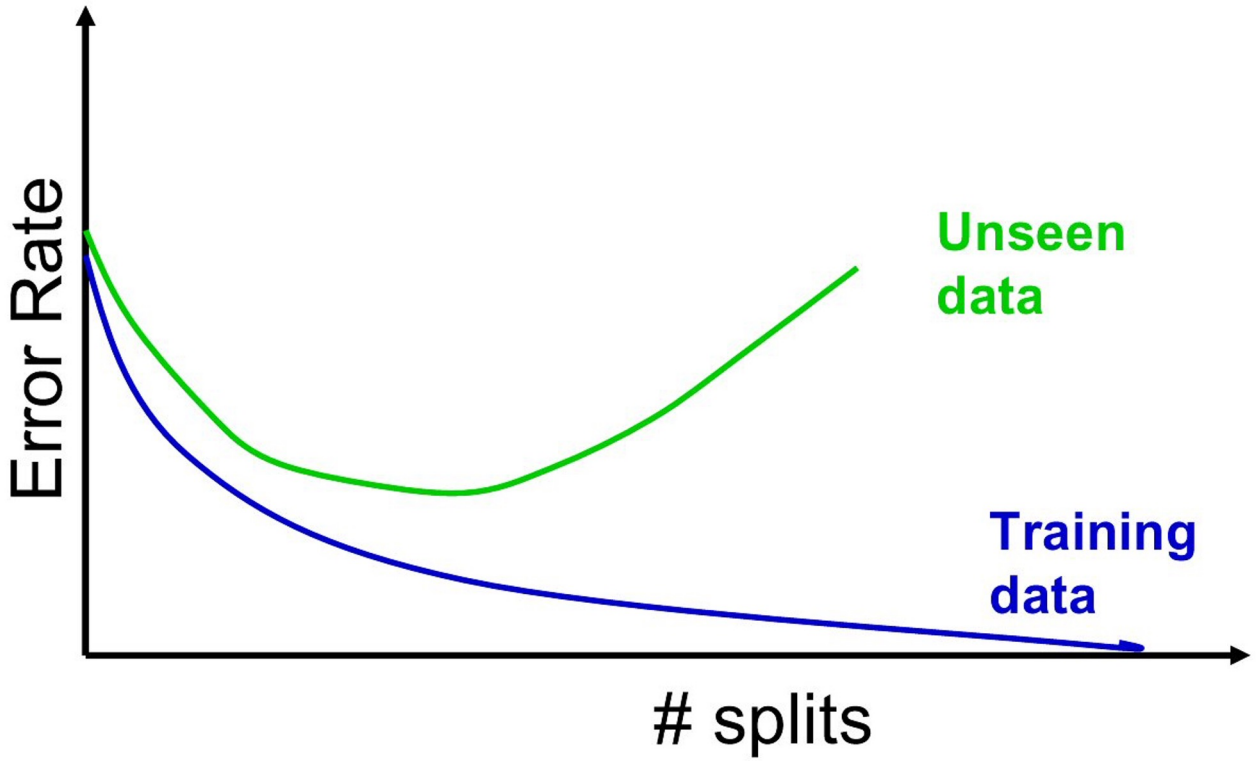
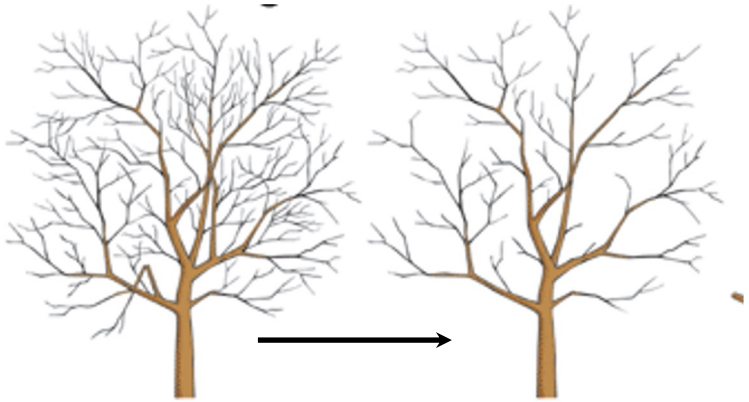


FIGURE 9.10 A FULL TREE FOR THE LOAN ACCEPTANCE DATA USING THE TRAINING SET (3000 RECORDS)

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



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

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 `cp` that corresponds to the minimum validation error
  - Average these `cp`'s
  - With future data, grow tree to that optimum `cp` 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 `cp` 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 `cp` to set `cp` 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 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 and Boosted Trees

- Examples of “ensemble” methods, “Wisdom of the Crowd” (Chap 13)
- Predictions from many trees are combined
- Very good predictive performance, better than single trees (often the top choice for predictive modeling)
- Cost: loss of rules you can explain implement (since you are dealing with many trees, not a single tree)
  - However, RF does produce “variable importance scores,” (using information about how predictors reduce Gini scores over all the trees in the forest)

Random Forests (library randomForest)

- Draw multiple bootstrap resamples of cases from the data
- For each resample, use a random subset of predictors and produce a tree
- 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 `cp` level to stop tree growth
- Ensembles (random forests, boosting) improve predictive performance, but you lose interpretability and the rules embodied in a single tree

Ensembles

Why Combine?

- Ensemble of methods often predicts more accurately
- Business goal may require multiple methods

The Ox Contest

"The Wisdom of Crowds"

- Francis Galton, famous statistician, saw a county fair contest to judge the weight of an ox.
- Individual guesses were all over the map, but the average of them all was within 1% of the ox's true weight
- For more, see *The Wisdom of Crowds*, by James Surowiecki

Ensemble methods for classification and prediction

In an ensemble approach, multiple methods are used initially, and predictions/classifications tabulated

- Predicting a numeric value? Take the average of the values predicted by the various methods
- Predicting a class? Take a majority vote of the classes predicted by the various methods
- Predicting a propensity? Take the average of the propensities

Why does an ensemble make more accurate predictions?

The key is reducing the variance in predictions.

- Individual methods will produce predictions that have errors, some positive and some negative
- If prediction methods are unbiased, on balance, errors tend to cancel each other out
- An average of multiple predictions takes advantage of this canceling out and, most of the time, is more accurate than individual predictions

Popular forms of ensembles

- Bagging
- Boosting
- Most often applied to trees

Bagging
(= bootstrap aggregating)

The “multiplier” effect in bagging comes from multiple bootstrap samples, rather than multiple methods. Bootstrapping is to take resamples, with replacement, from the original data.

1. Generate multiple bootstrap resamples
2. Run algorithm on each and produce scores
3. Average those scores (or take majority vote)

Boosting

Iteratively focus attention on the records that are misclassified, or where error is greatest

1. Fit model to data
2. Resample records with highest weights to misclassified or highest errors
3. Fit model to new sample
4. Repeat steps 2-3

Bagging/Boosting in R

(Universal Bank example)

```
library(adabag)
library(rpart)
library(caret)

[code for data prep and partitioning – see Table 13.1]

# bagging
bag <- bagging(Personal.Loan ~ ., data = train.df)
pred <- predict(bag, valid.df, type = "class")

# boosting
boost <- boosting(Personal.Loan ~ ., data = train.df)
pred <- predict(boost, valid.df, type = "class")
```

Ensembles summary

Ensembles…

1. Generally perform better than individual models
2. Have many variants (averaging, weighted averaging, voting, medians, resampling)
3. Facilitate “parallel processing,” e.g. in contests where multiple teams’ models can be combined
4. Help mitigate overfitting (but do not cure it)
5. Are black-box – transparent methods like trees lose transparency when ensembled

Persuasion (uplift) modeling

Often a business problem cannot be tacked with just one method.

1. Video streaming service wants to offer recommendations, but there are two different users on a single account, same location. Solution – cluster watched videos into 2 clusters, classify new shopping activity into one of the clusters.
2. Political campaign wants to know which of two messages to send to individual voters – i.e. which has most “uplift” in propensity to vote favorably

Uplift modeling (cont.)

1. Uplift modeling starts with an A-B test of two treatments
2. In marketing, might be message A versus message B
3. In political campaigns, might be message versus no message

Uplift modeling (cont.)

1. For each voter you now have two variables – response (0/1), and which message they got (A/B)

Voter #	Message	Response
1	A	0
2	A	1
3	B	0

Uplift modeling (cont.)

1. For each voter you now have two variables – response (0/1), and which message they got (A/B)
2. Now add demographic, marketing, and voting history info for each voter in sample.

Voter #	Newspaper	Voted Primary	Voted General	Age	Message	Response
1	0	0	1	53	A	0
2	1	0	1	61	A	1
3	1	0	0	26	B	0

Uplift modeling (cont.)

1. For each voter you now have two variables – response (0/1), and which message they got (A/B)
2. Now add demographic, marketing, and voting history info for each voter in sample.
3. Fit a classification model (0/1 – respond or not), with all predictor variables, including which message was sent
4. Run the model for all voters twice
    1. With original data
    2. With message predictor reversed

Uplift modeling (cont.)

1. You now have two propensity scores for each voter
    1. One as if they got message A
    2. One as if they got message B
2. Propensity for favorable response with B minus propensity with A is the uplift for B over A
3. Used in marketing to “microtarget” different marketing messages appropriately
4. In political campaigns, often used to determine which is better:
    1. Send a message
    2. Send no message

Uplift code in R

```
library(uplift)
voter.df <- read.csv("Voter-Persuasion.csv")
```

[code for data prep and partitioning – see Table 13.9]

syntax must include the prediction algorithm:
`upliftKNN(), upliftRF(), …`
and specify outcome variable, predictor variables, and treatment variable

For applying a Random Forest to the Voter data:

```
up.fit <- upliftRF(MOVED_AD_NUM ~ AGE + NH_WHITE + COMM_PT + H_F1
     + REG_DAYS+PR_PELIG + E_PELIG + POLITICALC + trt(MESSAGE_A),
     data = train.df, mtry = 3, ntree = 100, split_method = "KL",
     minsplit = 200, verbose = TRUE)
```

Uplift Predictions for Voter Data

```
pred <- predict(up.fit, newdata = valid.df)
# first column: p(y | treatment)
# second column: p(y | control)
head(data.frame(pred, "uplift" = pred[,1] - pred[,2]))
```

Output

	pr.y1_ct1	pr.y1_ct0	uplift
1	0.356284	0.319376	0.036908
2	0.489685	0.437061	0.052624
3	0.380408	0.365436	0.014972
4	0.379597	0.341727	0.037870
5	0.371465	0.293659	0.077806
6	0.405424	0.349785	0.055639