NUS National University of Singapore

School of Computing

Decision Making Technologies for Business

BT5152
2018/2019 Semester I

Week 3

Associate Professor
HUANG, Ke-Wei

Today's Class

1. Decision Tree Classification
2. Decision Tree Regression
3. Decision Tree Pruning
4. Rule-Based Classification
    1. 1R
    2. RIPPER
    3. Sequential Covering

Part 1: Decision Tree Classification

- Pioneered by ID3 algorithm in Quinlan, J. R. (1986). Induction of decision trees. *Machine learning*, *1*(1), 81-106.
- Decision Tree is one of the most important classification algorithm because

1. Easy to understand the algorithm
2. Easy to understand the output
3. Fast
4. Acceptable classification accuracy

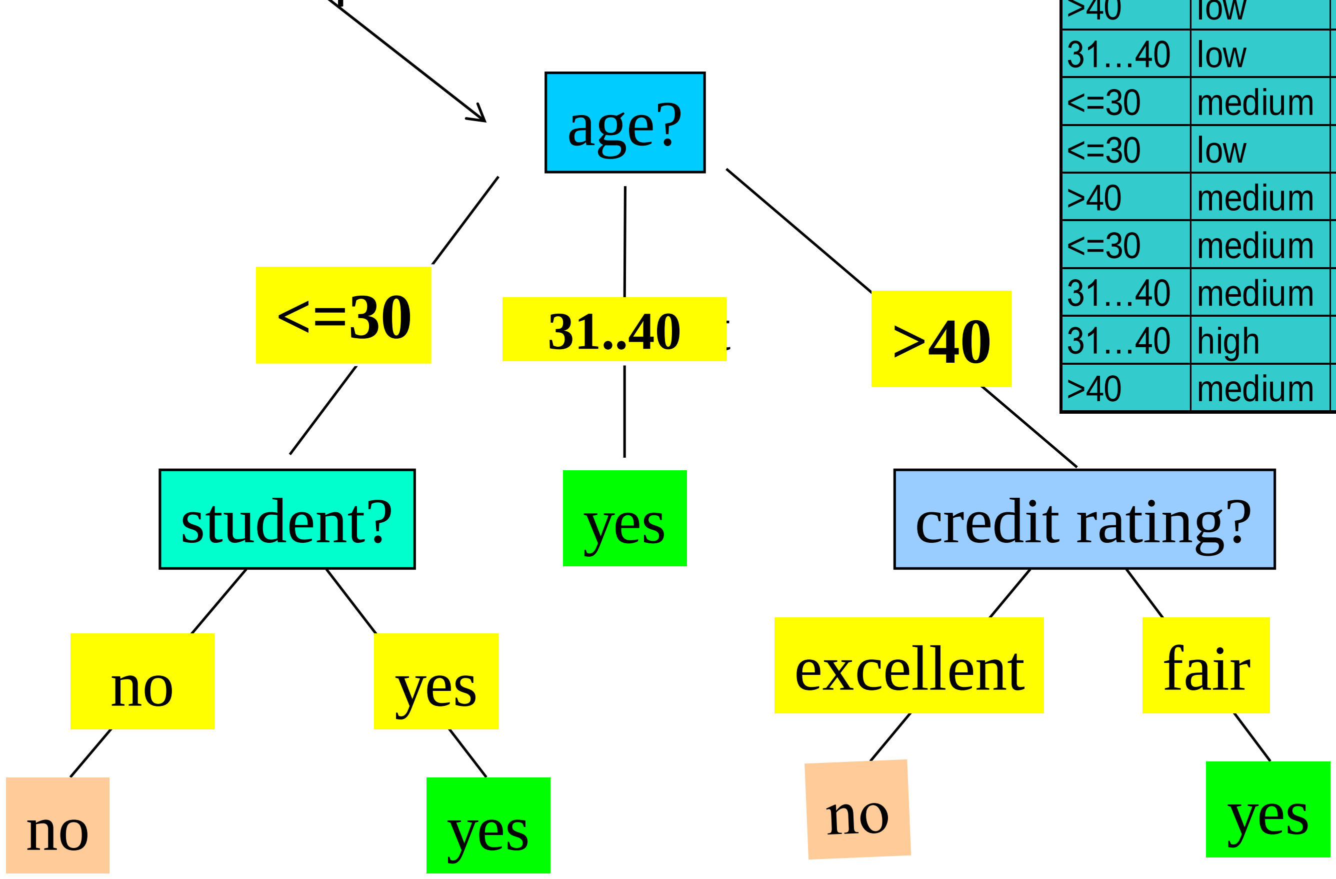
- Decision Tree has been used in loan approval classification in finance, predicting a customer will stay or leave (churn analysis), or diagnosis of medical condition.

What is Decision Tree? An Example

- The data set follows an example of Quinlan's ID3.
- Predict "Buys_computer"
- Final Output

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

Label



Overview of the Ideas of Tree Algorithms

- It uses two common tricks in CS algorithms:
  (1) Divide and Conquer and
  (2) Greedy Approach.
- Step 1: we choose one feature that can BEST split the examples into two or more groups. "Best" is judged by Entropy reduced or by other metrics of "information gain".
- Step 2: for each branch of the trees, we iteratively find the next feature to best split the subsets until the stopping condition is met.
- Step 3: Pruning the tree to avoid overfitting.

Algorithm for ID3 Decision Tree

- Basic algorithm (a greedy algorithm)
  - Tree is constructed in a top-down, recursive, divide-and-conquer manner
  - At start, all the training examples are at the root
  - Attributes are categorical (if continuous-valued, they are discretized in advance)
  - Examples are partitioned recursively based on selected attributes
  - Test attributes are selected on the basis of a "heuristic" or statistical measure (information gain)
- Conditions for stopping partitioning
  - All samples for a given node belong to the same class
  - There are no remaining attributes for further partitioning – majority voting is employed for classifying the leaf
  - There are no samples left

Brief Review of Entropy

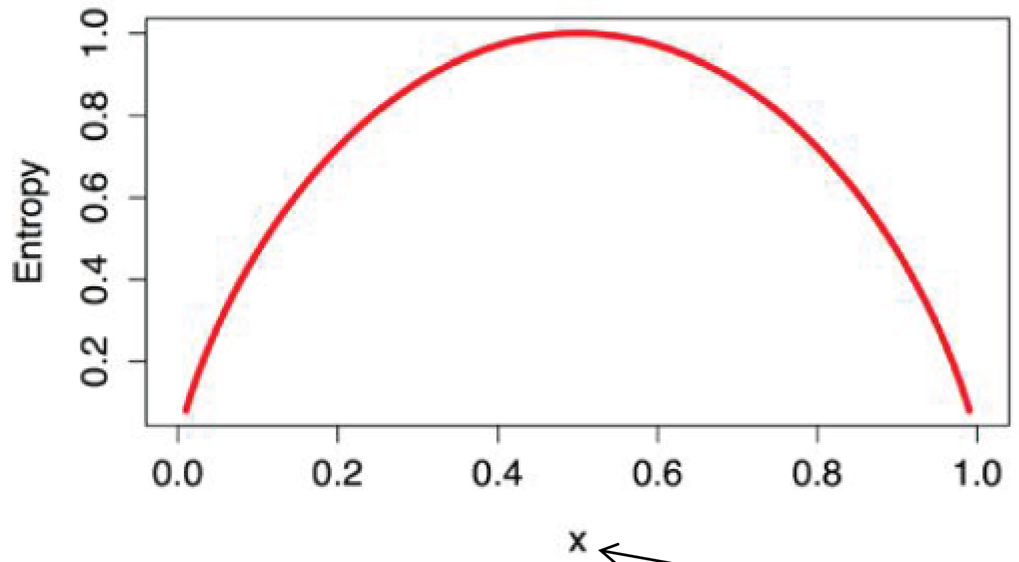
- Entropy (Information Theory)
  - A measure of uncertainty associated with a random variable
  - Calculation: For a discrete random variable *Y* taking *m* distinct values $\{y_1, \ldots, y_m\}$,
    - $H(Y) = -\sum_{i=1}^{m} p_i \log(p_i)$, where $p_i = P(Y = y_i)$
  - Interpretation:
    - Higher entropy => higher uncertainty
    - Lower entropy => lower uncertainty
- Conditional Entropy
  - $H(Y|X) = \sum_x p(x) H(Y|X = x)$

Brief Review of Entropy

- For example, if there are 40% "1" and 60% "0", then the entropy is 0.9709506

```
-0.60 * log2(0.60) - 0.40 * log2(0.40)
```

This results in the following figure:



This is the proportion of "1"

Attribute Selection Measure: Information Gain (ID3 algorithm)

- In each iteration of the tree, we select the attribute with the highest **information gain.**
- Let p_i be the probability that an arbitrary example in D belongs to class C_i, estimated by $|C_{i,D}|/|D|$
- Before we split, Expected entropy is:

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i)$$

- After using a new feature to split D into *v* partitions:

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j)$$

- Information gained by branching on attribute A

$$Gain(A) = Info(D) - Info_A(D)$$

Example:

- We have 14 examples at the root.
- 9 are "YES"
- 5 are "NO"
- So the Prior Entropy (Expected Information) is

$$Info(D) = I(9,5) = -\frac{9}{14}\log_2(\frac{9}{14}) - \frac{5}{14}\log_2(\frac{5}{14}) = 0.940$$

- Notice that the we use log_2, not log_10 or natural log. This is because we like to have the results that when the examples are equally distributed, the entropy is 1.
- Most of the modern decision trees use binary split even the attribute is a multi-class variable.
- Also, can you show I(9,5) = I(5,9)?

Information Gain of AGE

1. Now the algorithm needs to calculate the information gain of each one of all features and pick the best one. E.g. “Age”, Results are given below

age	p_i	n_i	$I(p_i, n_i)$
<=30	2	3	0.971
31…40	4	0	0
>40	3	2	0.971

I(2,3) = I(3,2) = -(2/5)*log(2/5) – (3/5)*log(3/5)=0.970951

2. So after splitting by Age, the total entropy is given by

$$Info_{age}(D) = \frac{5}{14} I(2,3) + \frac{4}{14} I(4,0) + \frac{5}{14} I(3,2) = 0.694$$

(The first term means “age <=30” has 5 out of 14 samples, with 2 yes and 3 no.)

3. The information gain is given by 0.94 - 0.694 =0.246.

Information Gain of Level 1

- Similarly, we can calculate the information gain of the other 3 variables. The results are Gain(income) = 0.029, Gain(student) = 0.151, and Gain(credit rating) = 0.048.
- So using “Age” is the best split for the first level because it reduces entropy the most.
- Next, we also need to repeat the process for two sub-trees (Age<=30) and (Age>40) whereas the sub-tree (Age between 30-40) is completed because all examples are “yes” already.

Information Gain of Level 2

- For both sub-trees, the prior entropy is I(2,3)=0.971.
- Age<=30. You can observe that "student" gives perfect split.

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
<=30	medium	yes	excellent	yes

- Age>40. Credit_Rating gives the perfect split.

age	income	student	credit_rating	buys_computer
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
>40	medium	yes	fair	yes
>40	medium	no	excellent	no

- So the final output is the tree on slide #9.

Computing Information-Gain for Continuous-Valued Attributes

- Let attribute A be a continuous-valued attribute
- Must determine the *best split point* for A
  - Sort the values of A in increasing order
  - Typically, the midpoint between each pair of adjacent values is considered as a possible *split point*
    - $(a_i+a_{i+1})/2$ is the midpoint between the values of a_i and a_{i+1}
  - The point with the *minimum expected information requirement (min. of entropy)* for A is selected as the split-point for A
- Split:
  - D1 is the set of examples in D satisfying A ≤ split-point, and D2 is the set of examples in D satisfying A > split-point

Gain Ratio for Attribute Selection (C4.5)

- Information gain measure is biased towards attributes with a large number of values
- C4.5 (a successor of ID3) uses gain ratio to overcome the problem.
- The idea is to use the entropy of that feature to normalize information gain

$$SplitInfo_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times \log_2\left(\frac{|D_j|}{|D|}\right)$$

  - GainRatio(A) = Gain(A)/SplitInfo(A)
- Ex.

$$SplitInfo_{income}(D) = -\frac{4}{14} \times \log_2\left(\frac{4}{14}\right) - \frac{6}{14} \times \log_2\left(\frac{6}{14}\right) - \frac{4}{14} \times \log_2\left(\frac{4}{14}\right) = 1.557$$

  - This is the entropy of "Income", not of label.
  - gain_ratio(income) = 0.029/1.557 = 0.019
- The attribute with the maximum gain ratio is selected as the splitting attribute

Evolution of Tree Algorithms

- ID3 is the first tree algorithm and is covered in most textbooks but practically outdated.
- C4.5 and C5.0 are enhanced/improved version of ID3. Both are still available in most packages. Due to its simplicity and speed, these are still used as a "benchmarking" case so that you know your fancy algorithms or tricks works or not.
- CART (rpart package in R) algorithm next is the core tree algorithm that are used in modern algorithms (Random Forest and Gradient Boosting Machines) that will be covered later this semester. GBM algorithms are the state-of-art algorithms for prediction.

Gini Index (CART, IBM IntelligentMiner)

- If a data set *D* contains examples from *n* classes, gini index, *gini*(*D*) is defined as

$$gini(D) = 1 - \sum_{j=1}^{n} p_j^2$$

where p_j is the relative frequency of class *j* in *D*

- If a data set *D* is split on A into two subsets D_1 and D_2, the *gini* index *gini*(*D*) is defined as

$$gini_A(D) = \frac{|D_1|}{|D|} gini(D_1) + \frac{|D_2|}{|D|} gini(D_2)$$

- Reduction in Impurity:

$$\Delta gini(A) = gini(D) - gini_A(D)$$

- The attribute provides the smallest $gini_{split}(D)$ (or the largest reduction in impurity) is chosen to split the node

Computation of Gini Index

- Ex. D has 9 examples in buys_computer = “yes” and 5 in “no”

$$gini(D) = 1 - \left(\frac{9}{14}\right)^2 - \left(\frac{5}{14}\right)^2 = 0.459$$

- Suppose the attribute income partitions D into 10 in D_1: {low, medium} and 4 in D_2 {High}

$$\begin{aligned}
&= \frac{10}{14}\left(1-\left(\frac{7}{10}\right)^2-\left(\frac{3}{10}\right)^2\right)+\frac{4}{14}\left(1-\left(\frac{2}{4}\right)^2-\left(\frac{2}{4}\right)^2\right) \\
&= 0.443 \\
&= Gini_{income\,\in\,\{high\}}(D).
\end{aligned}$$

2 buying and 2 not buying among 4 high income customers

7 buy and 3 not buying for low and medium income.

- That binary split is better than the other two possible binary splits: $\text{Gini}_{\{\text{low,high}\}}$ is 0.458; $\text{Gini}_{\{\text{medium,high}\}}$ is 0.450. Thus, split on the {low,medium} {high} since it has the lowest Gini index.
- The gain after split is 0.459 - 0.443

Comparing Attribute Selection Measures

- The three measures, in general, return good results but
  - **Information gain**:
    - biased towards multi-valued attributes
  - **Gain ratio**:
    - tends to prefer unbalanced splits in which one partition is much smaller than the others
  - **Gini index**:
    - Biased to multi-valued attributes
    - Performance is worse when the number of classes is large
    - Tends to favor tests that result in equal-sized partitions and purity in both partitions

Computation of Gini Index

- Ex. D has 9 examples in buys_computer = "yes" and 5 in "no"

$$gini(D) = 1 - \left(\frac{9}{14}\right)^2 - \left(\frac{5}{14}\right)^2 = 0.459$$

- Suppose the attribute income partitions D into 10 in D_1: {low, medium} and 4 in D_2 {High}

$$\begin{aligned} &= \frac{10}{14}\left(1-\left(\frac{7}{10}\right)^2-\left(\frac{3}{10}\right)^2\right)+\frac{4}{14}\left(1-\left(\frac{2}{4}\right)^2-\left(\frac{2}{4}\right)^2\right) \\ &= 0.443 \\ &= Gini_{income \in \{high\}}(D). \end{aligned}$$

2 buying and 2 not buying among 4 high income customers

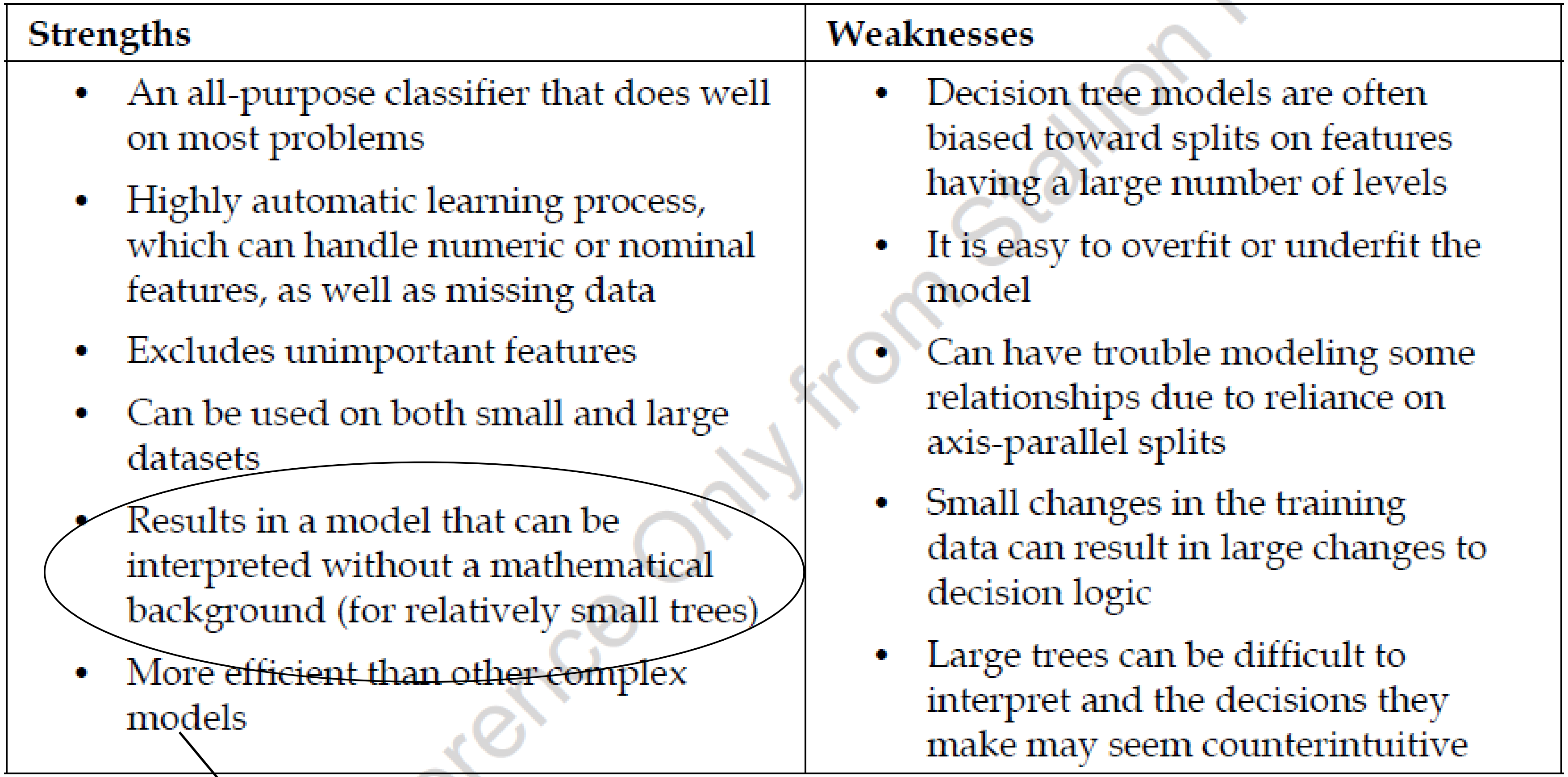
7 buy and 3 not buying for low and medium income.

- That binary split is better than the other two possible binary splits: $Gini_{\{low,high\}}$ is 0.458; $Gini_{\{medium,high\}}$ is 0.450. Thus, split on the {low,medium} {high} since it has the lowest Gini index.
- The gain after split is 0.459 - 0.443

Other Attribute Selection Measures

- CHAID: a popular decision tree algorithm, measure based on χ^2 test for independence
- C-SEP: performs better than info. gain and Gini index in certain cases
- G-statistic: has a close approximation to χ^2 distribution
- Multivariate splits (partition based on multiple variable combinations)
  - CART: finds multivariate splits based on a linear comb. of attrs.
- **Which attribute selection measure is the best?**
  - Most give good results, none is significantly superior than others in all cases!
  - MDL (Minimal Description Length) principle (i.e., the simplest solution is preferred), C50 and RPART are two popular packages.

Decision Tree: Strengths and Weakness



Strengths	Weaknesses
• An all-purpose classifier that does well on most problems • Highly automatic learning process, which can handle numeric or nominal features, as well as missing data • Excludes unimportant features • Can be used on both small and large datasets • Results in a model that can be interpreted without a mathematical background (for relatively small trees) • More ~~efficient than other complex~~ models	• Decision tree models are often biased toward splits on features having a large number of levels • It is easy to overfit or underfit the model • Can have trouble modeling some relationships due to reliance on axis-parallel splits • Small changes in the training data can result in large changes to decision logic • Large trees can be difficult to interpret and the decisions they make may seem counterintuitive

Decision Tree is much faster than Neural Network and SVM, and thus is more suitable for ensemble methods.

More on advantages of trees

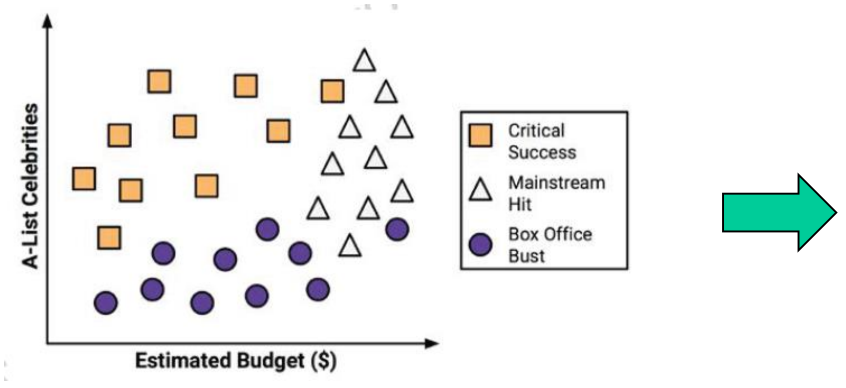
1. Output can be understood by managers: both transparency and interpretability are high. This is critical for some organizations and is a critical factor that tree is important in practice.
2. Fast. Speed has many advantages once you’ve learned how slow Neural Network and SVM could be.
3. Variable selection & reduction is automatic, scalability in terms of number of features is high.
   – Both 2 and 3 are important when you have limited computational resources and also you need to explore a number of models and features, such as when you compete in a data competition task.
4. Convertible to simple and easy to understand classification rules. Those rules are the same as SQL commands in database for implementation.

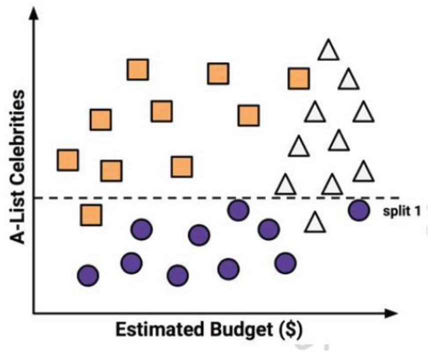
More on the Disadvantages

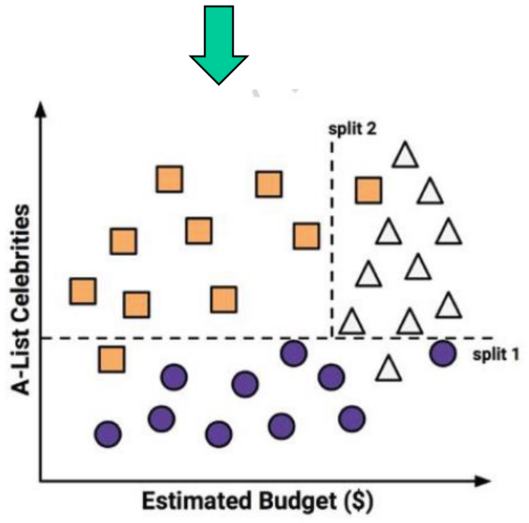
- Traditional linear regression, Neural Network, and SVM all “optimize” a performance metric globally whereas Decision Tree only uses a local greedy approach and no guarantee on the overall optimal solution in terms of accuracy. From this perspective, decision tree performance is worse in terms of accuracy.
  - DT does not minimize sum of squared errors like in OLS/Neural Net/SVM which minimize sum of errors all at once by selecting beta coefficients.
  - However, the decision tree performance is acceptably good.
- (3rd bullet in the table) Since the process deals with one variable at a time, no easy way to capture interactions between variables and more complicated relationship among features.

More on the Disadvantages

- Decision Tree always use a straight line to classify examples!







This way to visualize decision tree classification will show that decision tree is not flexible enough to have shapes like a circle or else…

Part 2. Regression Trees for Prediction

- The decision tree we’ve learned is for predicting categorical labels (categorical dependent variables).
- We can easily modified the algorithm so it can be used to predict a continuous outcome variable
- Procedure similar to classification tree
  - Assume there are K attributes, we use “greedy-approach + divide-and-conquer”, both of which are widely used heuristics in CS algorithms.
  - Step 1: At the root node, we try to pick one attribute out of K attributes that can best split the examples in the training set.
  - The problem is how to define “BEST SPLITS”? Reducing entropy/Gini is replaced by?

Procedure of Regression Tree

- Step 1 (Conti.): The most common one is "reducing sum of squared error w.r.t. the mean of sub-tree (same one as that used in linear regression)". Some algorithms reduce the standard deviation or variance of DV in the sub-trees.
- Step 2: repeat Step 1 to grow the full tree.
- Step 3: In regression trees the predicted value of the leaf node is determined by the average of the training data in that leaf.
- Step 4: Over-fitting is a more serious problem. Prune the tree and evaluating the final performance.
  - using summary measures such as root-mean-square error (RMSE, same in OLS or other statistical methods)
  - Mean absolute error, relative squared error, correlation coefficient.

Part 3: Overfitting and Tree Pruning

- Overfitting: An induced tree may overfit the training data
  - Too many branches, some may reflect anomalies due to noise or outliers
  - Poor accuracy for unseen samples
  - More severe when you have less data and more attributes
- As an extreme example, suppose we want to use decision tree to automate the master program admission and we have 1000 examples in the training set and 20 binary features. A fully grown tree may have only 1 or few examples at the leaf; we are clearly overfitting.

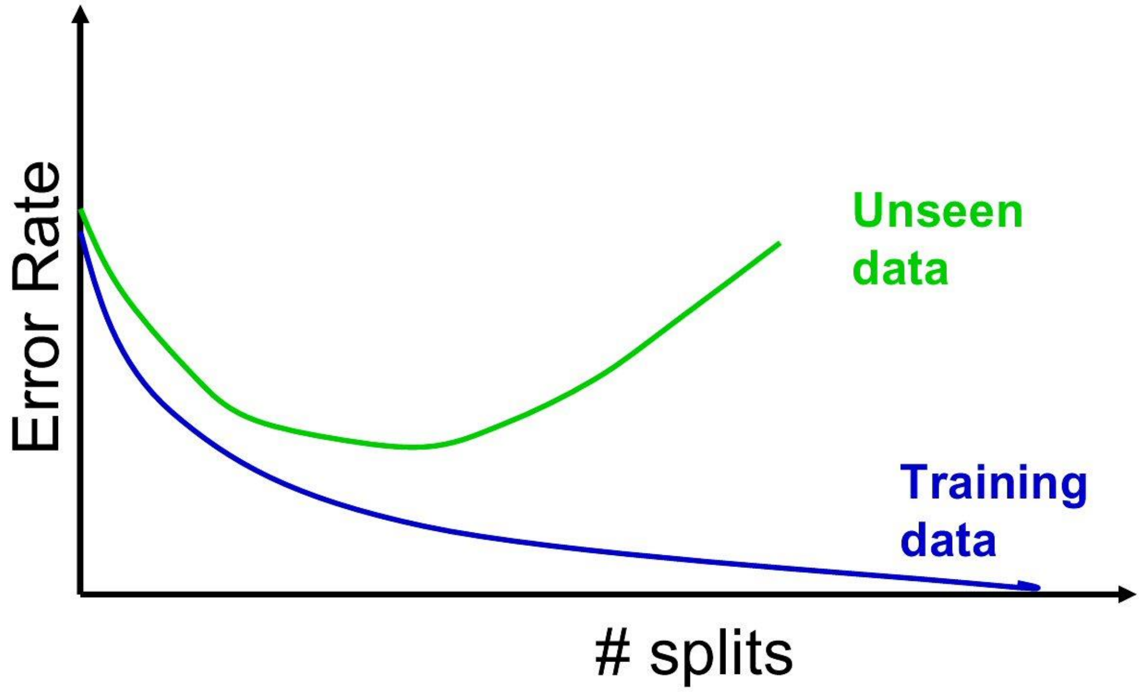
Overfitting and Tree Pruning

Two approaches to avoid overfitting

- Early-Stopping = Prepruning: *Halt tree construction early*
  - Only build a tree with a maximum number of levels or maximum number of nodes.
  - Or Each leaf must contain a minimum number of examples.
  - Other "complexity" measures as stopping criterion. Different algorithms use a different heuristics of complexity metric.
  - Difficult to choose an appropriate threshold
- Postpruning: *Remove branches* from a "fully grown" tree. Post-pruning algorithms are more technical and complicated.

Full Tree Error Rate

One common method of pruning is to have a subset of examples as the "validation set" (green line below). For decision tree classification/regression, if we have more levels and leafs, the error rate always reduce on the training set but not necessarily on the validation set. So we will simply experiment on these two datasets to prune the tree to minimize the error rate on the validation set.



More on Pruning of Trees

- For those of you are more interested in technical subjects, see
  - Esposito, F., Malerba, D., Semeraro, G., & Kay, J. (1997). A comparative analysis of methods for pruning decision trees. *IEEE transactions on pattern analysis and machine intelligence*, *19*(5), 476-491.
- 6 Methods discussed in this paper include Reduced Error Pruning (REP) (similar to the idea on the previous slide.). Roughly speaking, the other methods involves creating a scoring function that represents the degree of overfitting.
- The author finds that REP creates the smallest subtree with the lowest error rate with respect to the pruning set in quite a number of datasets.
- We will practice similar ideas when we learn rpart in R.

Part 4: Rules Classification

- The problem of decision tree is when you have a tree with 10 levels, it becomes so complicated that it is still difficult for human beings to interpret.

The solution is:

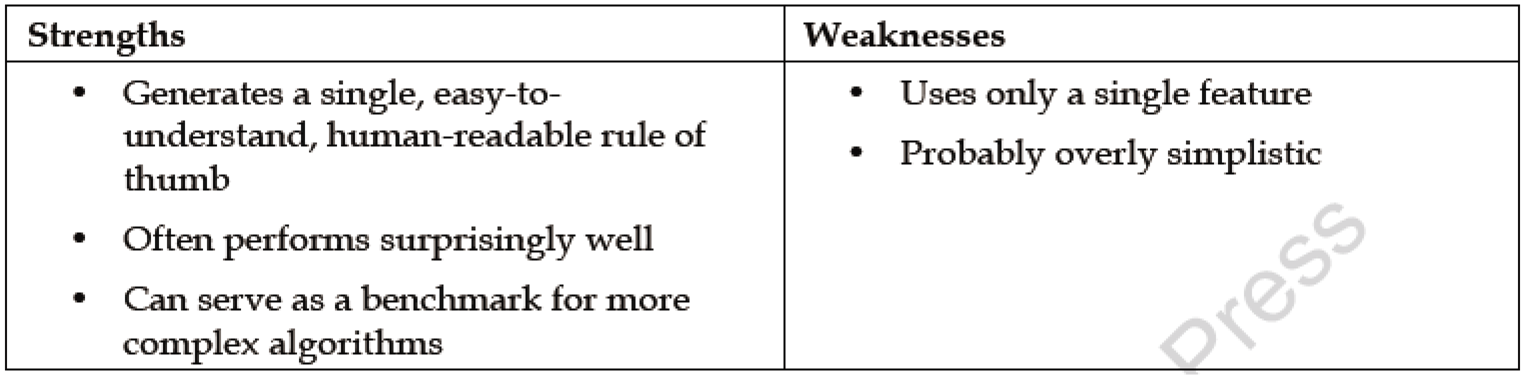
- Represent the knowledge in the form of IF-THEN rules

  Rule #1. IF *age* = youth AND *student* = yes THEN *buys_computer* = yes

  - Rule antecedent/precondition on attributes
  - vs. rule consequent on labels.

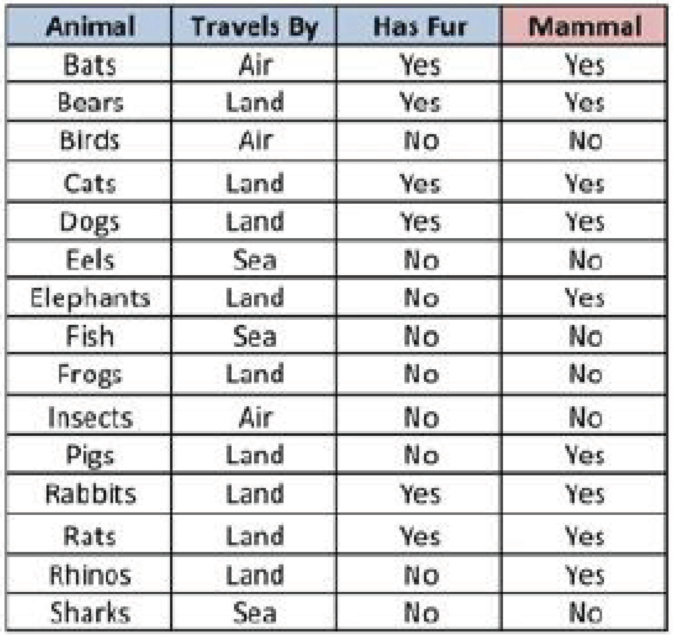
Part 4-1: The 1R Algorithm

- It is also called One Rule or OneR algorithm.
- 1R algorithm selects only ONE BEST RULE!
- Although this may seem over-simplistic, its performance is better than expected.
- Because of its simplicity, it has very high interpretability and is also very fast!



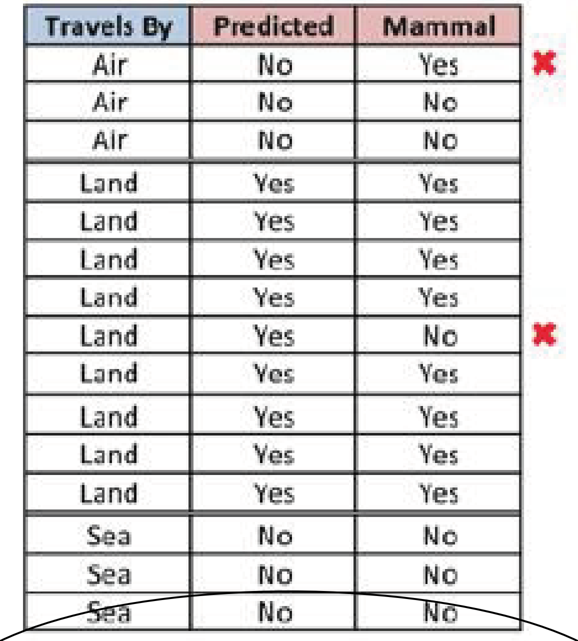
Strengths	Weaknesses
• Generates a single, easy-to-understand, human-readable rule of thumb • Often performs surprisingly well • Can serve as a benchmark for more complex algorithms	• Uses only a single feature • Probably overly simplistic

The 1R Algorithm Example



Animal	Travels By	Has Fur	Mammal
Bats	Air	Yes	Yes
Bears	Land	Yes	Yes
Birds	Air	No	No
Cats	Land	Yes	Yes
Dogs	Land	Yes	Yes
Eels	Sea	No	No
Elephants	Land	No	Yes
Fish	Sea	No	No
Frogs	Land	No	No
Insects	Air	No	No
Pigs	Land	No	Yes
Rabbits	Land	Yes	Yes
Rats	Land	Yes	Yes
Rhinos	Land	No	Yes
Sharks	Sea	No	No

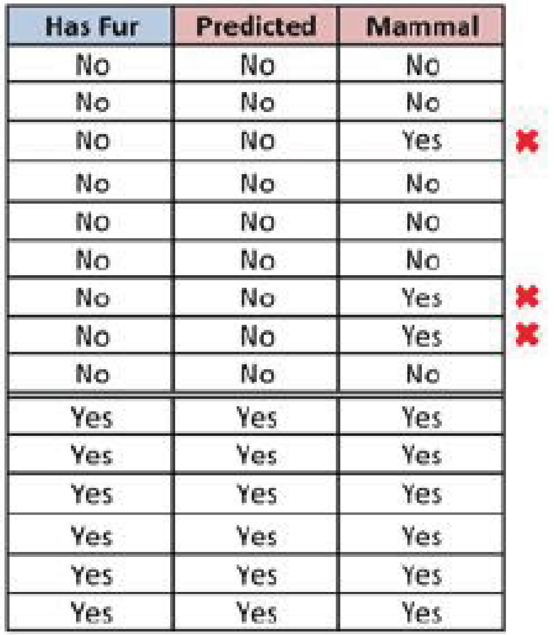
Full Dataset



Travels By	Predicted	Mammal	
Air	No	Yes	✖
Air	No	No	
Air	No	No	
Land	Yes	Yes	
Land	Yes	Yes	
Land	Yes	Yes	
Land	Yes	Yes	
Land	Yes	No	✖
Land	Yes	Yes	
Land	Yes	Yes	
Land	Yes	Yes	
Land	Yes	Yes	
Sea	No	No	
Sea	No	No	
Sea	No	No	

Rule for "Travels By"

Error Rate = 2 / 15



Has Fur	Predicted	Mammal	
No	No	No	
No	No	No	
No	No	Yes	✖
No	No	No	
No	No	No	
No	No	No	
No	No	Yes	✖
No	No	Yes	✖
No	No	No	
Yes	Yes	Yes	
Yes	Yes	Yes	
Yes	Yes	Yes	
Yes	Yes	Yes	
Yes	Yes	Yes	
Yes	Yes	Yes	

Rule for "Has Fur"

Error Rate = 3 / 15

只用一种规则来判断是还是否

- If the animal travels by air, it is not a mammal
- If the animal travels by land, it is a mammal
- If the animal travels by sea, it is not a mammal

Experiment: 1R vs C4.5

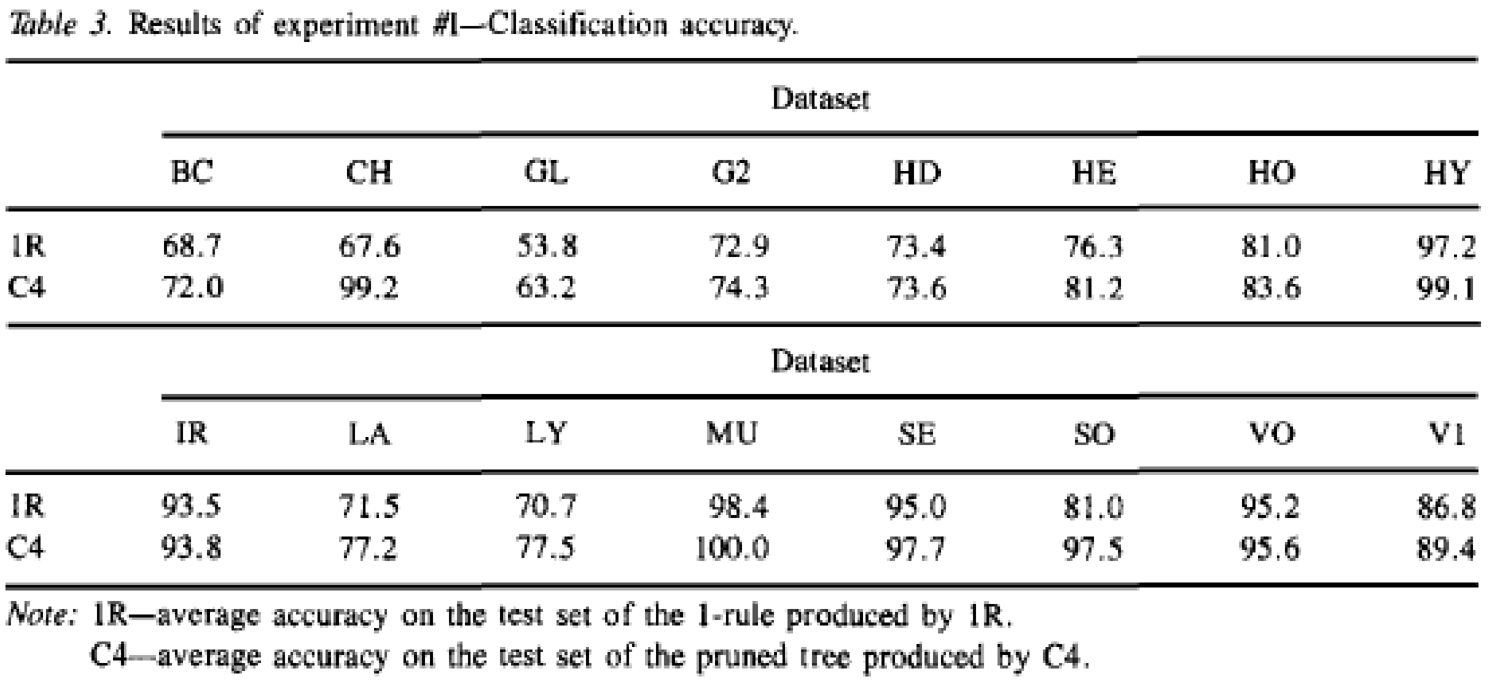


Table 3. Results of experiment #1—Classification accuracy.

	Dataset							
	BC	CH	GL	G2	HD	HE	HO	HY
1R	68.7	67.6	53.8	72.9	73.4	76.3	81.0	97.2
C4	72.0	99.2	63.2	74.3	73.6	81.2	83.6	99.1

	Dataset							
	IR	LA	LY	MU	SE	SO	VO	V1
1R	93.5	71.5	70.7	98.4	95.0	81.0	95.2	86.8
C4	93.8	77.2	77.5	100.0	97.7	97.5	95.6	89.4

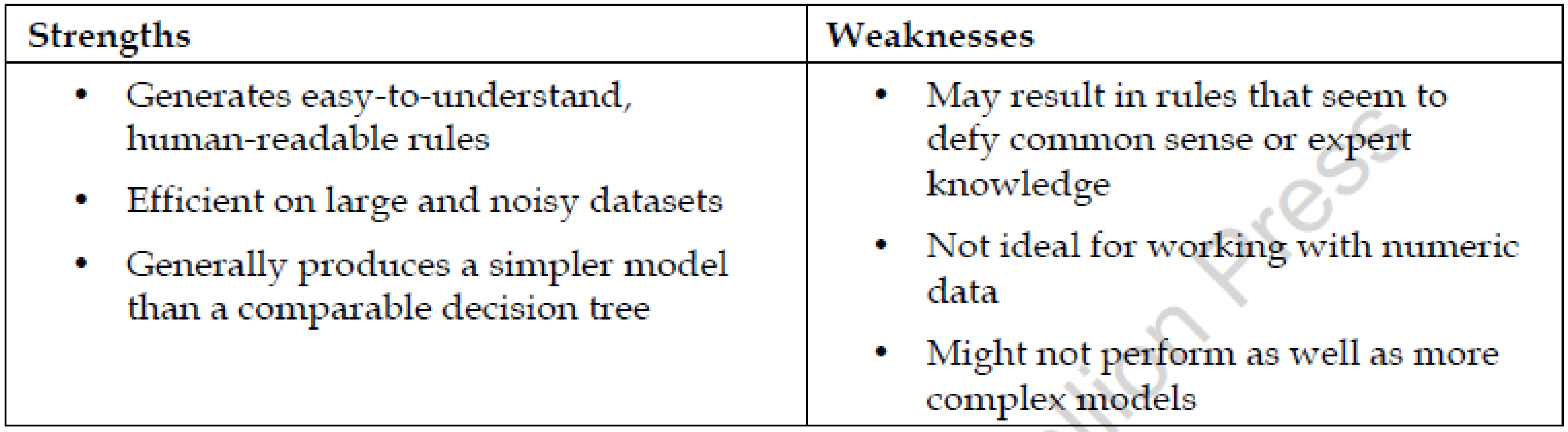
Note: 1R—average accuracy on the test set of the 1-rule produced by 1R.
C4—average accuracy on the test set of the pruned tree produced by C4.

Holte, R. C. (1993). Very simple classification rules perform well on most commonly used datasets. *Machine learning*, *11*(1), 63-90.

Part 4-2: RIPPER Algorithm

- Rule learners took another step forward in 1995 when William W. Cohen introduced the **Repeated Incremental Pruning to Produce Error Reduction (RIPPER) algorithm, which generate rules that match or exceed the** performance of decision trees.
- Cohen, W. W. (1995, July). Fast effective rule induction. In *Proceedings of the twelfth international conference on machine learning* (pp. 115-123).
- The chief benefit is that RIPPER may result in a slightly more parsimonious model than decision tree.

RIPPER Algorithm



Strengths	Weaknesses
• Generates easy-to-understand, human-readable rules • Efficient on large and noisy datasets • Generally produces a simpler model than a comparable decision tree	• May result in rules that seem to defy common sense or expert knowledge • Not ideal for working with numeric data • Might not perform as well as more complex models

RIPPER Algorithm has three steps

1. Grow Rules
2. Prune Rules
3. Optimize

This algorithm is just one example of sequential covering algorithms for rule classification.

Sequential Covering

- Find rules that cover many records with high classification accuracy. Then gradually find the set of useful rules, an idea similar to but different from decision tree classification.

- Assessment of a rule: *coverage* and *accuracy*
  - n_{covers} = # of tuples covered by Rule #1
  - $n_{correct}$ = # of tuples correctly classified by Rule #1
  - $|D|$: number of tuples in the training data set

  coverage(R) = $n_{covers} / |D|$

  accuracy(R) = $n_{correct} / n_{covers}$

Rule Induction: Sequential Covering Method

- Typical sequential covering algorithms: FOIL, AQ, CN2, RIPPER. The main idea is explained in details here.
- Steps:
  1. Rules are learned one at a time
     a) We try to find the attribute's rule that can cover a label class as accurately as possible (if tie, then choose the rule that covers more correct cases).
     b) If the min. accuracy does not meet the requirement, we add a condition based on other attributes. Again, choose the rule that leads to highest accuracy (if tie, more correct labels).
     c) Repeat until the accuracy is good enough.
  2. Each time a rule is learned, the tuples covered by the rule are removed
  3. Repeat the process Step1&2 on the remaining tuples until *termination condition*, e.g., when no more training examples or when the quality of a rule returned is below a user-specified threshold.

Example

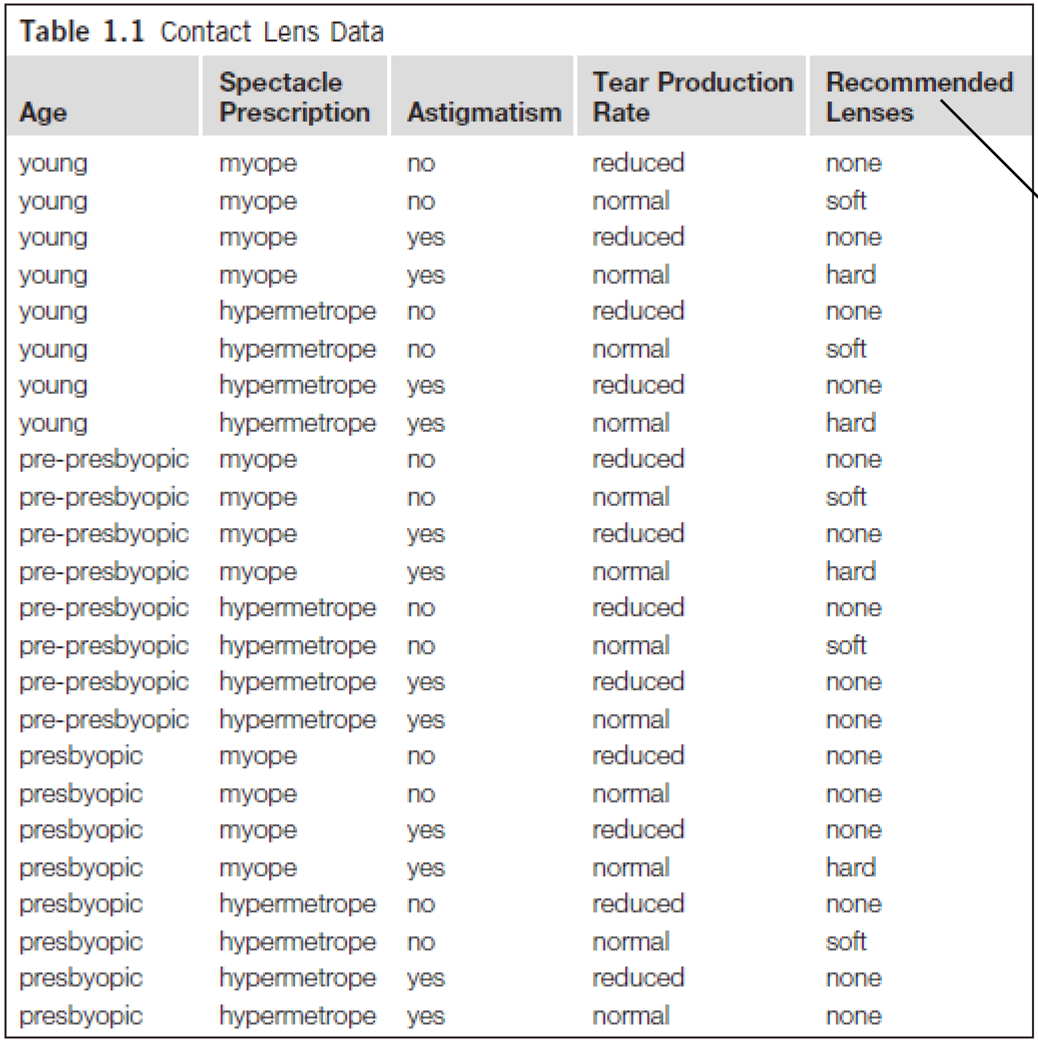


Table 1.1 Contact Lens Data

Age	Spectacle Prescription	Astigmatism	Tear Production Rate	Recommended Lenses
young	myope	no	reduced	none
young	myope	no	normal	soft
young	myope	yes	reduced	none
young	myope	yes	normal	hard
young	hypermetrope	no	reduced	none
young	hypermetrope	no	normal	soft
young	hypermetrope	yes	reduced	none
young	hypermetrope	yes	normal	hard
pre-presbyopic	myope	no	reduced	none
pre-presbyopic	myope	no	normal	soft
pre-presbyopic	myope	yes	reduced	none
pre-presbyopic	myope	yes	normal	hard
pre-presbyopic	hypermetrope	no	reduced	none
pre-presbyopic	hypermetrope	no	normal	soft
pre-presbyopic	hypermetrope	yes	reduced	none
pre-presbyopic	hypermetrope	yes	normal	none
presbyopic	myope	no	reduced	none
presbyopic	myope	no	normal	none
presbyopic	myope	yes	reduced	none
presbyopic	myope	yes	normal	hard
presbyopic	hypermetrope	no	reduced	none
presbyopic	hypermetrope	no	normal	soft
presbyopic	hypermetrope	yes	reduced	none
presbyopic	hypermetrope	yes	normal	none

Label for classification

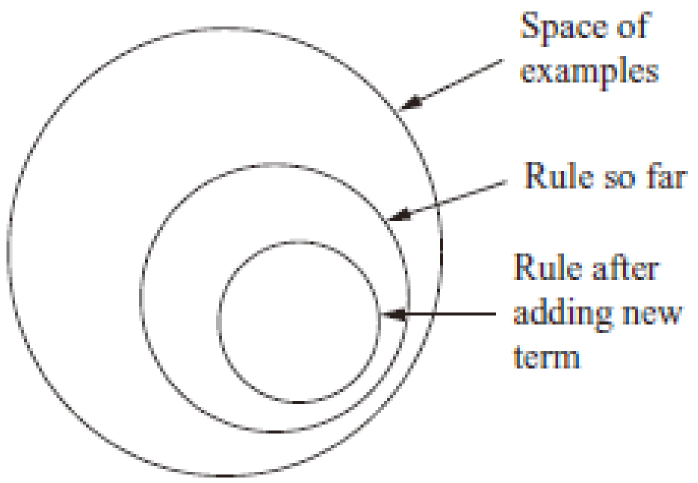
Example

An example will help. For a change, we use the contact lens problem of Table 1.1 (page 6). We will form rules that cover each of the three classes—*hard*, *soft*, and *none*—in turn. To begin, we seek a rule:

```
If ? then recommendation = hard
```

For the unknown term ?, we have nine choices:

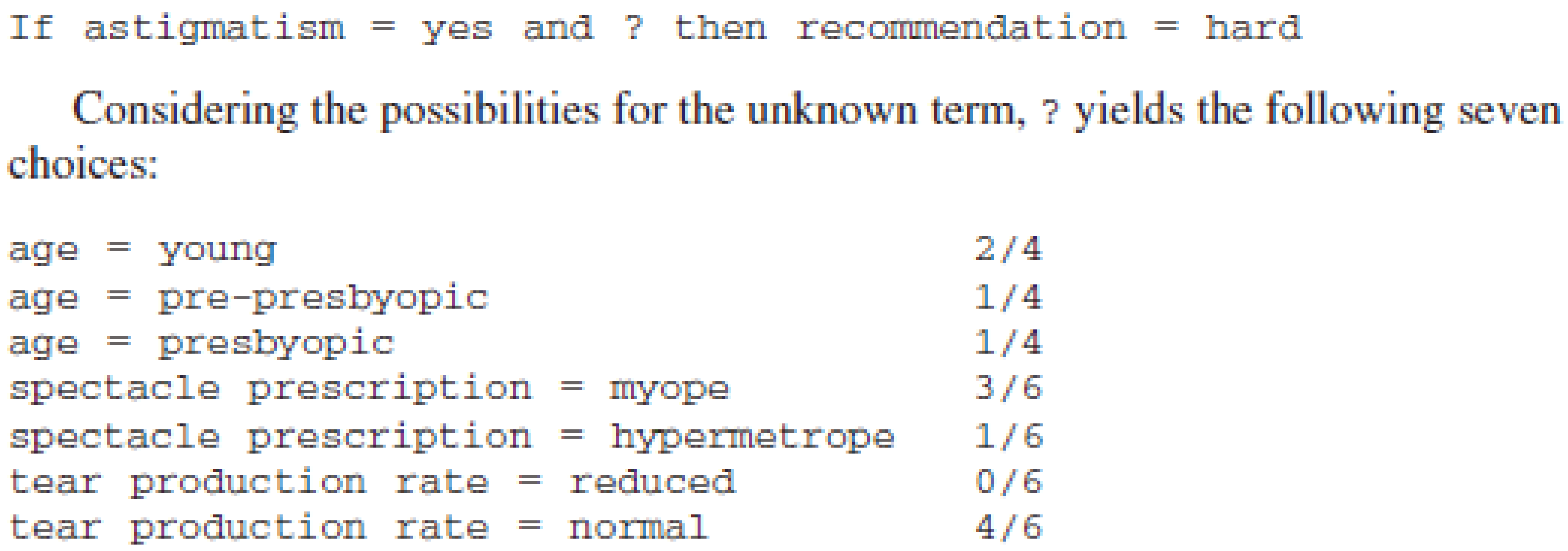
age = young	2/8	spectacle prescription = hypermetrope	1/12
age = pre-presbyopic	1/8	astigmatism = no	0/12
age = presbyopic	1/8	astigmatism = yes	4/12
spectacle prescription = myope	3/12	tear production rate = reduced	0/12
		tear production rate = normal	4/12



Example

- Among the 9 rules, the most **accurate** rules are the 7th and 9th rules.
  - We first compare the number of correct labels, both are still 4 in this example. If there is no tie, then pick the one with larger number of correct labels.
  - Here, we have a completely tie and we randomly pick one to move on => we pick 7th rule.
- Now the accuracy is only 4/12 = 33.33% and is quite low. In most cases, this is still below the stopping criterion. In our example, let's set the stopping criterion at 100%.
- The next step is to add more condition to improve the accuracy.

Example

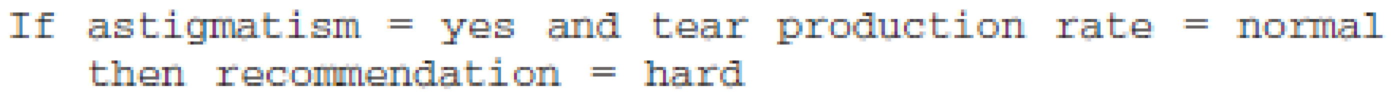


```
If astigmatism = yes and ? then recommendation = hard
```

Considering the possibilities for the unknown term, ? yields the following seven choices:

age = young	2/4
age = pre-presbyopic	1/4
age = presbyopic	1/4
spectacle prescription = myope	3/6
spectacle prescription = hypermetrope	1/6
tear production rate = reduced	0/6
tear production rate = normal	4/6

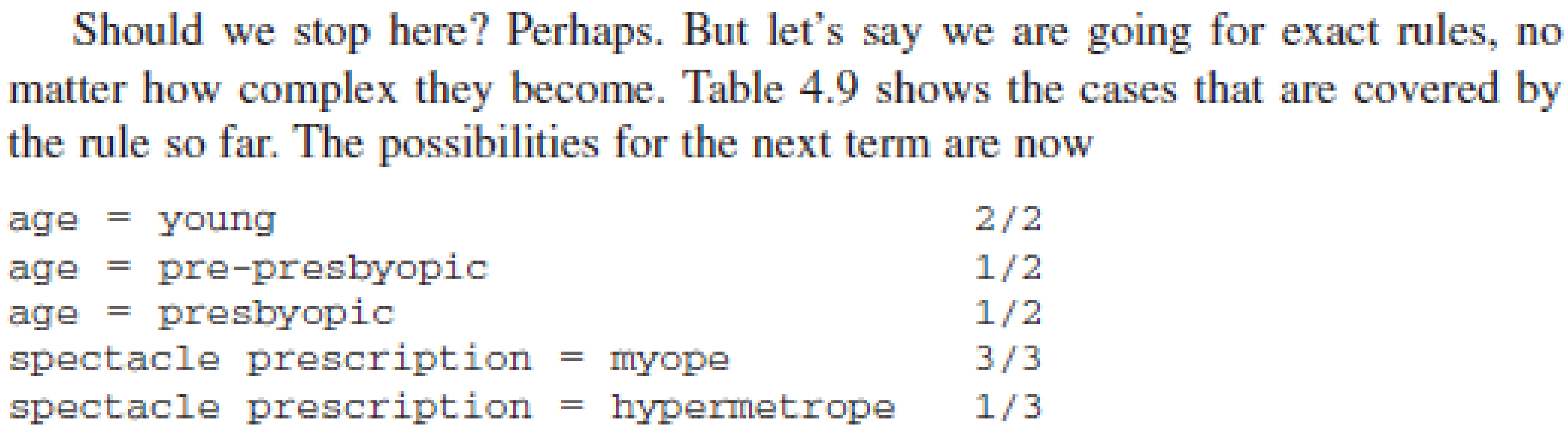
- Now the clear winner is the last rule.



```
If astigmatism = yes and tear production rate = normal
   then recommendation = hard
```

- In some practical scenarios, this may meet the stopping criterion already.

Example



Should we stop here? Perhaps. But let's say we are going for exact rules, no matter how complex they become. Table 4.9 shows the cases that are covered by the rule so far. The possibilities for the next term are now

age = young	2/2
age = pre-presbyopic	1/2
age = presbyopic	1/2
spectacle prescription = myope	3/3
spectacle prescription = hypermetrope	1/3

- Both 1st and 4th are 100% correct. We choose the one with larger coverage. In this case, the 4th rule is the winner in this iteration and we have finally have a perfect covering.
- Now we remove these 3 tuples from the training set and repeat to find more rules.
- We will repeat until all training tuples are covered by rules.