IS 733 Lesson 6

Supervised Learning

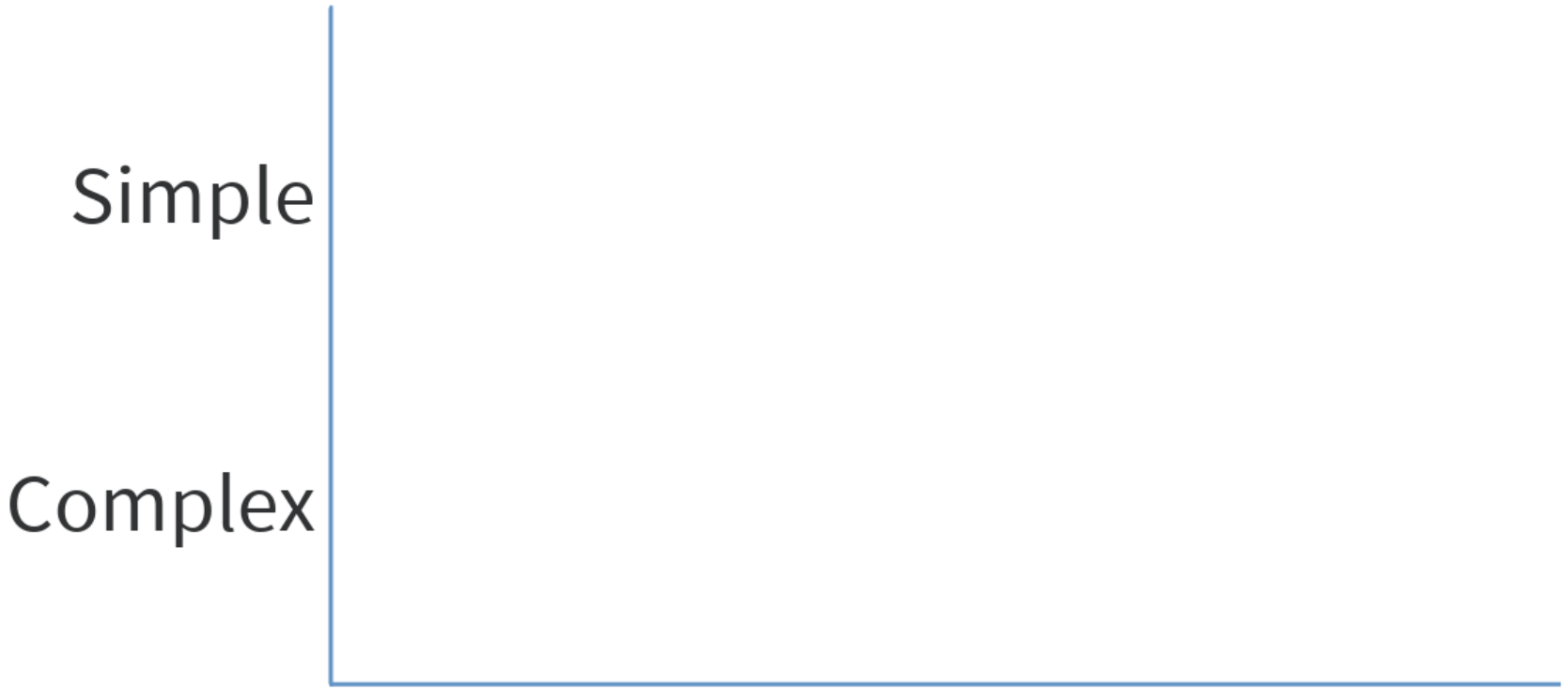
Slides based on those from Data Mining by I. H. Witten, E. Frank, M. A. Hall and C. J. Pal, Data Mining: Concepts and Techniques by Han et al., and Vandana Janeja, James Foulds

Announcements

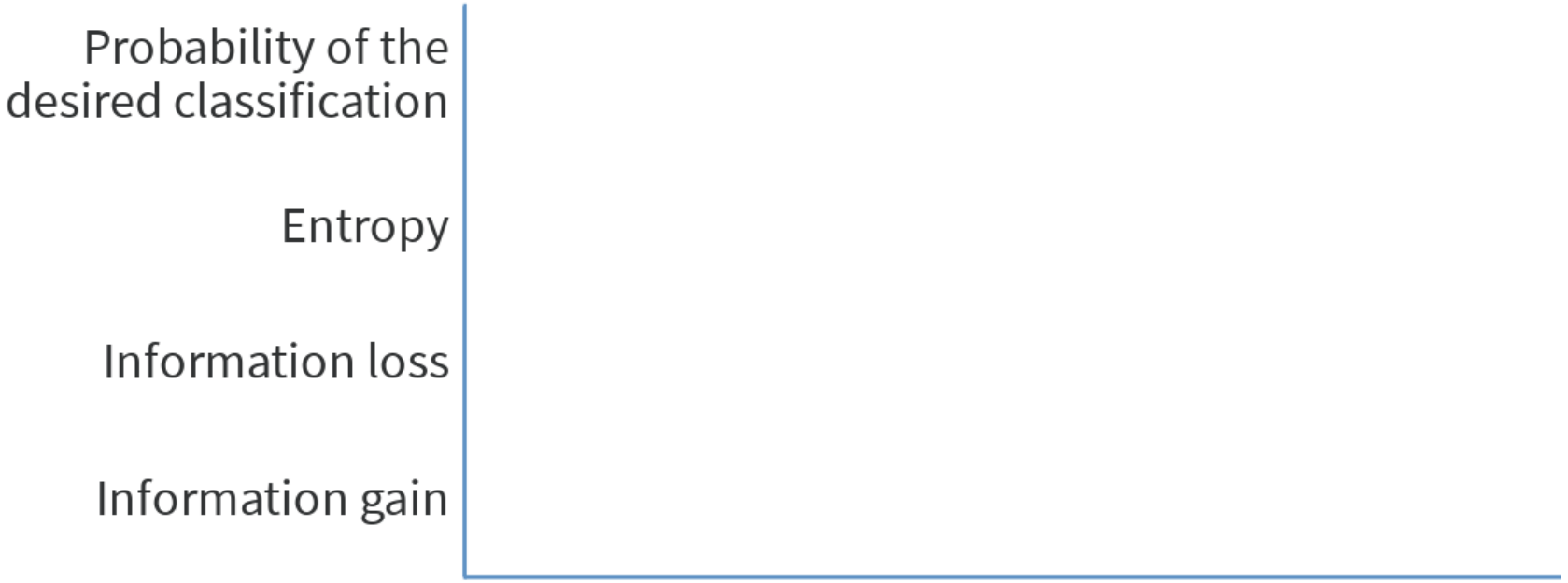
- **Homework 2** was due last Friday (3/5) on, Blackboard

- **Homework 3** is posted at the course webpage and due on, Blackboard by 3/30

According to Witten et al., when analyzing practical data sets it is best to try ____ methods first.



In the decision tree learning algorithm discussed in the book chapter (ID3), the next attribute to split on is always chosen to be the one that maximizes the _____.

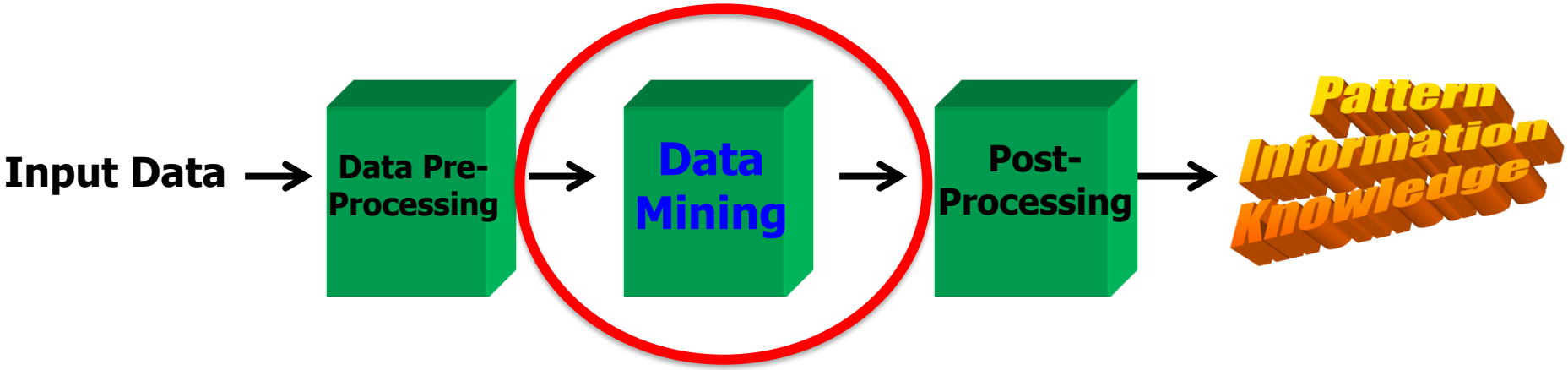


Learning outcomes

By the end of the lesson, you should be able to:

- **Discuss** the differences between **decision tree** and **decision rule** learning algorithms
- **Calculate** the **information gain** and **gain ratio** splitting criteria used by decision tree induction algorithms
- **Perform** the steps of the training algorithms for the **PRISM** rule learner, given a small dataset

Data Mining


Input Data
Data Pre-Processing
Data Mining
Post-Processing
Pattern
Information
Knowledge

Supervised learning

- Typical applications:
  - Credit/loan approval

  - Medical diagnosis: if a tumor is cancerous or benign

  - Fraud detection: if a transaction is fraudulent

  - Web page categorization: which category it is

Supervised vs. Unsupervised Learning

- Supervised learning (e.g. classification)
  - Supervision: The training data are accompanied by **labels,** e.g. the **class** of the observations
  - The goal is to predict the labels for new, unseen data

Outlook	Temperature	Humidity	Windy	Play
Sunny	85	85	False	No
Sunny	80	90	True	No
Overcast	83	86	False	Yes
Rainy	75	80	False	Yes
…	…	…	…	

Supervised vs. Unsupervised Learning

- Unsupervised learning (e.g. clustering)
  - The class labels of training data are **unknown**
  - We aim to find patterns in the data without class labels, e.g.
    - clustering the data into groups
    - finding associations between the attributes

Outlook	Temperature	Humidity	Windy	Play
Sunny	85	85	False	No
Sunny	80	90	True	No
Overcast	83	86	False	Yes
Rainy	75	80	False	Yes
…	…	…	…	…

Prediction Problems: Classification vs. Numeric Prediction

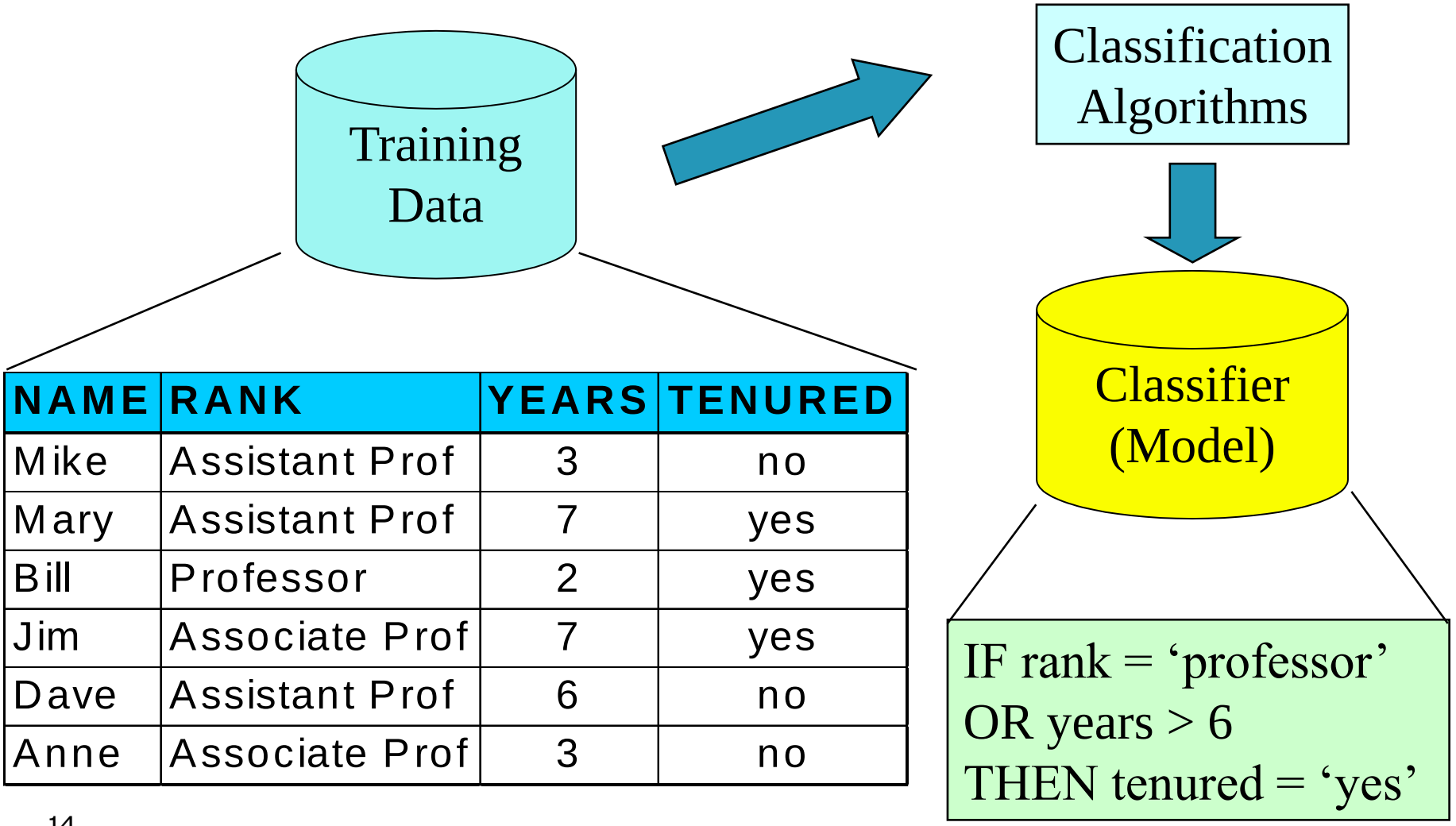
- Classification
  - predicts categorical class labels (discrete or nominal)
  - classifies data (constructs a model) based on the training set and the values (class labels) in a classifying attribute and uses it in classifying new data

- Numeric Prediction
  - models continuous-valued functions, i.e., predicts unknown or missing values

Classification—A Two-Step Process

- Step 1, Model construction:
  - **Given the data, build a model of it!**
  - Each tuple/sample is assumed to belong to a predefined class, as determined by the class label attribute
  - The set of tuples used for model construction is training set
  - The model is represented as either:
    - classification rules, decision trees, linear model, etc.

Process (1): Model Construction

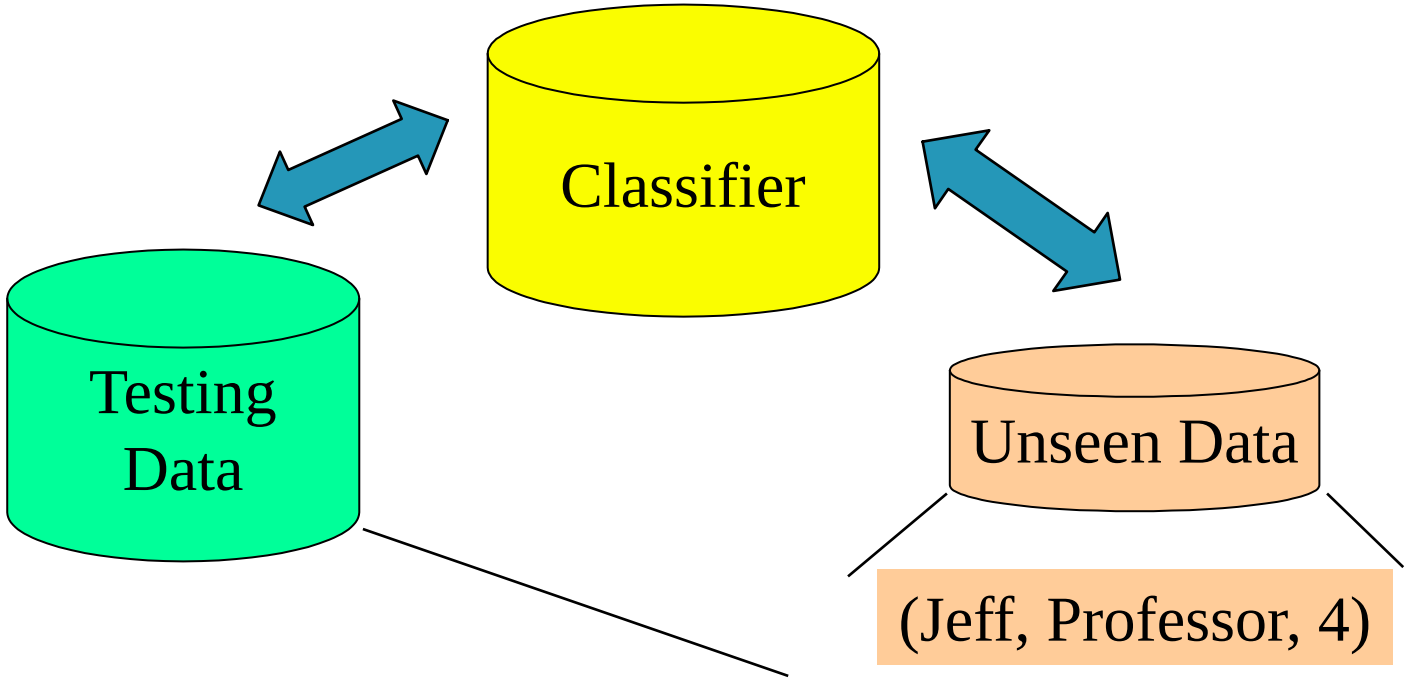


NAME	RANK	YEARS	TENURED
Mike	Assistant Prof	3	no
Mary	Assistant Prof	7	yes
Bill	Professor	2	yes
Jim	Associate Prof	7	yes
Dave	Assistant Prof	6	no
Anne	Associate Prof	3	no

Classification—A Two-Step Process

- Step 2, Model usage: for classifying future or unknown objects
  - Estimate accuracy of the model
    - The known label of test sample is compared with the classified result from the model
    - Accuracy rate is the percentage of test set samples that are correctly classified by the model
    - Test set is independent of training set (otherwise overfitting)
  - If the accuracy is acceptable, use the model to classify new data
- Note: If *the test set* is used to select models, it is called validation set

Process (2): Using the Model in Prediction



NAME	RANK	YEARS	TENURED
Tom	Assistant Prof	2	no
Merlisa	Associate Prof	7	no
George	Professor	5	yes
Joseph	Assistant Prof	7	yes

Elements of a Classification Algorithm

- **Representation**
- **Objective function for training**
- **Optimization algorithm**

Elements of a Classification Algorithm

- **Representation**
  - Rules, trees, linear model, instance-based, probabilistic model,…
    - Representation is important for interpretation. Flexibility of representation trades overfitting versus underfitting (bias-variance trade-off).
- **Objective function for training**
- **Optimization algorithm**

Elements of a Classification Algorithm

- **Representation**
  - Rules, trees, linear model, instance-based, probabilistic model,…
    - Representation is important for interpretation. Flexibility of representation trades overfitting versus underfitting (bias-variance trade-off).

- **Objective function for training**
  - E.g. Error rate, accuracy, loss, on training data
  - May include **regularization**, encouraging simpler models to prevent overfitting (e.g. via some penalty in the objective function)

- **Optimization algorithm**

Elements of a Classification Algorithm

- **Representation**
  - Rules, trees, linear model, instance-based, probabilistic model,…
    - Representation is important for interpretation. Flexibility of representation trades overfitting versus underfitting (bias-variance trade-off).

- **Objective function for training**
  - E.g. Error rate, accuracy, loss, on training data
  - May include **regularization**, encouraging simpler models to prevent overfitting (e.g. via some penalty in the objective function)

- **Optimization algorithm**
  - Aims to find “best” classifier according to objective function
  - Search algorithm can also avoid overfitting
    - Only consider simple classifiers, or search these first and stop early

Elements of a Classification Algorithm

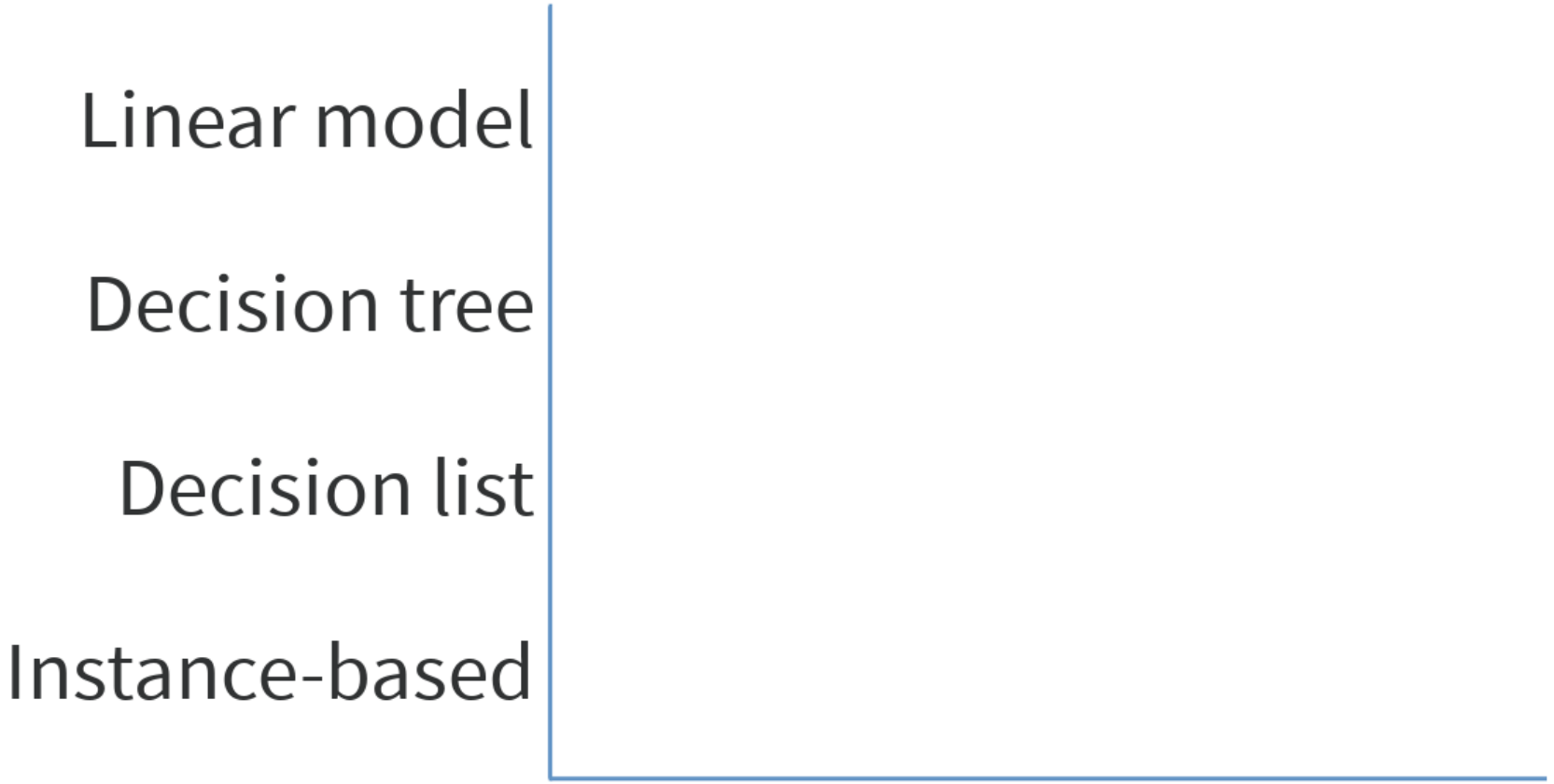
- **Representation**
  - Rules, trees, linear model, instance-based, probabilistic model,…
    - Representation is important for interpretation. Flexibility of representation trades overfitting versus underfitting (bias-variance trade-off).

- **Objective function for training**
  - E.g. Error rate, accuracy, loss, on training data
  - May include **regularization**, encouraging simpler models to prevent overfitting (e.g. via some penalty in the objective function)

- **Optimization algorithm**
  - Aims to find "best" classifier according to objective function
  - Search algorithm can also avoid overfitting
    - Only consider simple classifiers, or search these first and stop early

- **The whole game: select the above so that the method generalizes well, is computationally efficient, and is interpretable (if desired)**

Which representation does not require that a model be constructed based on the data?



Simplicity first

- Simple algorithms often work very well!
- There are many kinds of simple structure, e.g.:
  - One attribute does all the work
  - All attributes contribute equally & independently
  - Logical structure with a few attributes suitable for tree
  - A weighted linear combination of the attributes
  - Strong neighborhood relationships based on distance
  - …
- Success of method depends on the domain

Simplest possible classifier: ZeroR

- Ignore the input! Predict the **majority class**, according to the training data

- **Representation**
  - One rule with no antecedent: *If (true), predict Class=C*

- **Objective function for training**
  - Error on training data

- **Search algorithm**
  - Calculate class frequencies.
    Select the one with the highest frequency

Inferring rudimentary rules: 1R

- 1R rule learner:
  - A set of rules that all test **one particular attribute** - the one that yields the lowest classification error
  - Equivalent to a 1-level decision tree

Outlook

Sunny → No

Overcast → Yes

Rainy → Yes

Outlook	Temperature	Humidity	Windy	Play
Sunny	85	85	False	No
Sunny	80	90	True	No
Overcast	83	86	False	Yes
Rainy	75	80	False	Yes
...	...	...	...	...

Inferring rudimentary rules: 1R

- Basic version (assumes nominal attributes):
  - For each attribute
    - Make one branch for each value of the attribute
    - **To each branch, assign the most frequent class value** of the instances pertaining to that branch
  - Calculate the error rate of the rules
    - The **Error rate** equals the proportion of instances that do not belong to the majority class of their corresponding branch
  - Choose attribute with lowest **error rate**

1R Rule Learner

- **Representation**
  - Rules (tests on one attribute)
    Equivalently – a 1-level decision trees
- **Objective function for training**
  - Error rate on training data
- **Search algorithm**
  - Consider each attribute, create rules predicting class for each attribute value.
  - Pick the attribute that performs best on training data.

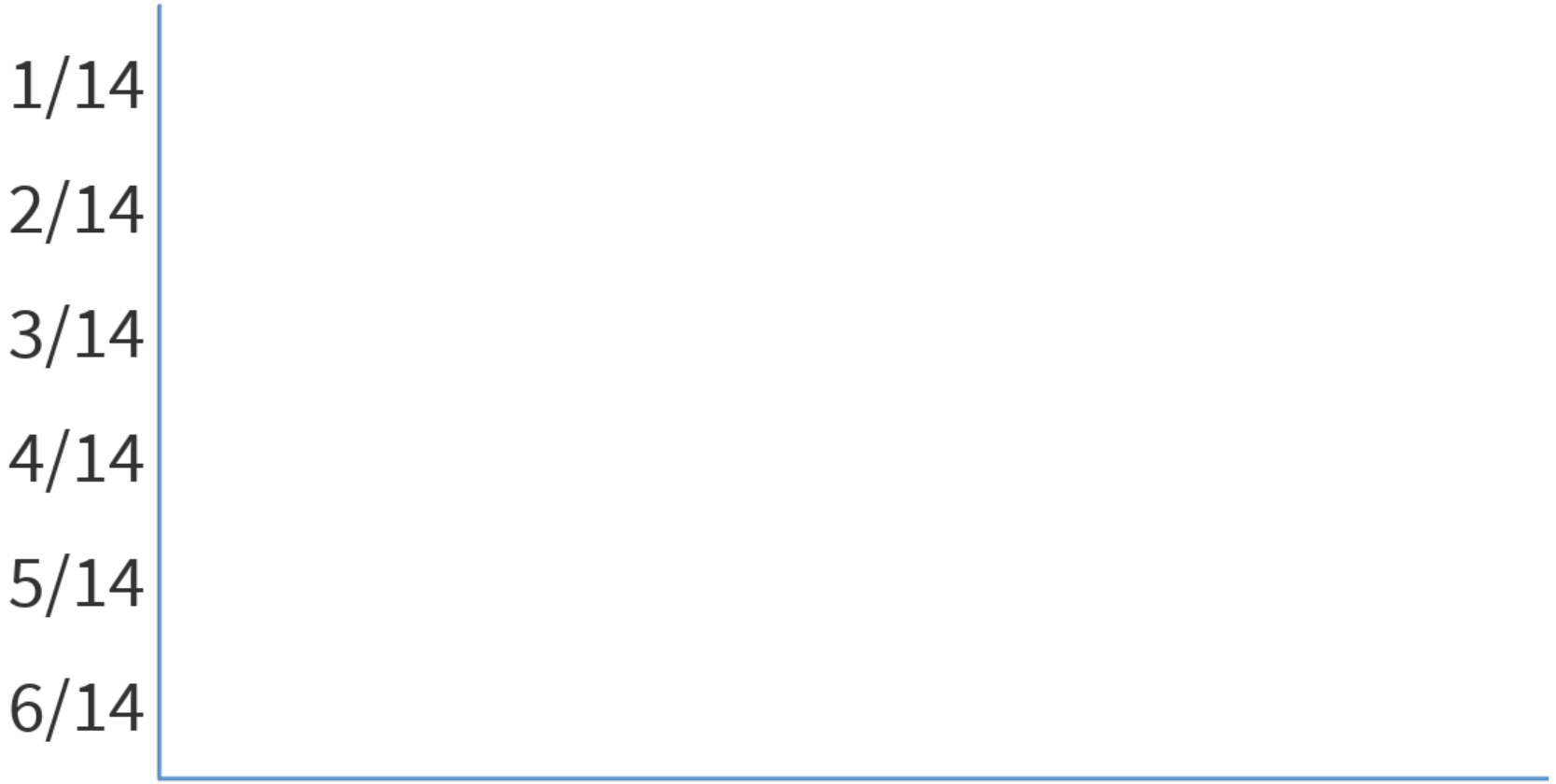
Evaluating the weather attributes

Outlook	Temp	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No

Attribute	Rules	Errors	Total errors
Outlook	Sunny → No	2/5	4/14
	Overcast → Yes	0/4	
	Rainy → Yes	2/5	
Temp	Hot → No*	2/4	5/14
	Mild → Yes	2/6	
	Cool → Yes	1/4	
Humidity	High → No	3/7	4/14
	Normal → Yes	1/7	
Windy	?		

* indicates a tie

What is the error rate for the windy attribute?



Evaluating the weather attributes

Outlook	Temp	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No

Attribute	Rules	Errors	Total errors
Outlook	Sunny → No	2/5	4/14
	Overcast → Yes	0/4	
	Rainy → Yes	2/5	
Temp	Hot → No*	2/4	5/14
	Mild → Yes	2/6	
	Cool → Yes	1/4	
Humidity	High → No	3/7	4/14
	Normal → Yes	1/7	
Windy	False → Yes	2/8	5/14
	True → No*	3/6	

* indicates a tie

Dealing with numeric attributes

- Idea: discretize numeric attributes into sub ranges (intervals)
- How to divide each attribute's overall range into intervals?
  - Sort instances according to attribute's values
  - Place breakpoints where (majority) class changes
  - This minimizes the total classification error
- Example: *temperature* from weather data

64 65 68 69 70 71 72 72 75 75 80 81 83 85
Yes | No | Yes Yes Yes | No No Yes | Yes Yes | No | Yes Yes | No

Outlook	Temperature	Humidity	Windy	Play
Sunny	85	85	False	No
Sunny	80	90	True	No
Overcast	83	86	False	Yes
Rainy	75	80	False	Yes
…	…	…	…	…

The problem of overfitting

- Discretization procedure is very **sensitive to noise**
  - A single instance with an incorrect class label will probably produce a separate interval

- Simple solution:
  enforce minimum number of instances in majority class per interval
- Example: *temperature* attribute with required minimum number of instances in majority class set to three:

64	65	68	69	70	71	72	72	75	75	80	81	83	85
Yes	No	Yes	Yes	Yes	No	No	Yes	Yes	Yes	No	Yes	Yes	No

64	65	68	69	70	71	72	72	75	75	80	81	83	85
Yes	No	Yes	Yes	Yes	No	No	Yes	Yes	Yes	No	Yes	Yes	No

Results with overfitting avoidance

- Resulting rule sets for the four attributes in the weather data, with only two rules for the temperature attribute:

Attribute	Rules	Errors	Total errors
Outlook	Sunny → No	2/5	4/14
	Overcast → Yes	0/4	
	Rainy → Yes	2/5	
Temperature	≤ 77.5 → Yes	3/10	5/14
	> 77.5 → No*	2/4	
Humidity	≤ 82.5 → Yes	1/7	3/14
	> 82.5 and ≤ 95.5 → No	2/6	
	> 95.5 → Yes	0/1	
Windy	False → Yes	2/8	5/14
	True → No*	3/6	

Discussion of 1R

- 1R was described in a paper by Holte (1993):

  Very Simple Classification Rules Perform Well on Most Commonly Used Datasets

  Robert C. Holte, Computer Science Department, University of Ottawa

  - Contains an experimental evaluation on 16 datasets (using *cross-validation* to estimate classification accuracy on fresh data)
  - 1R's simple rules performed not much worse than much more complex decision trees

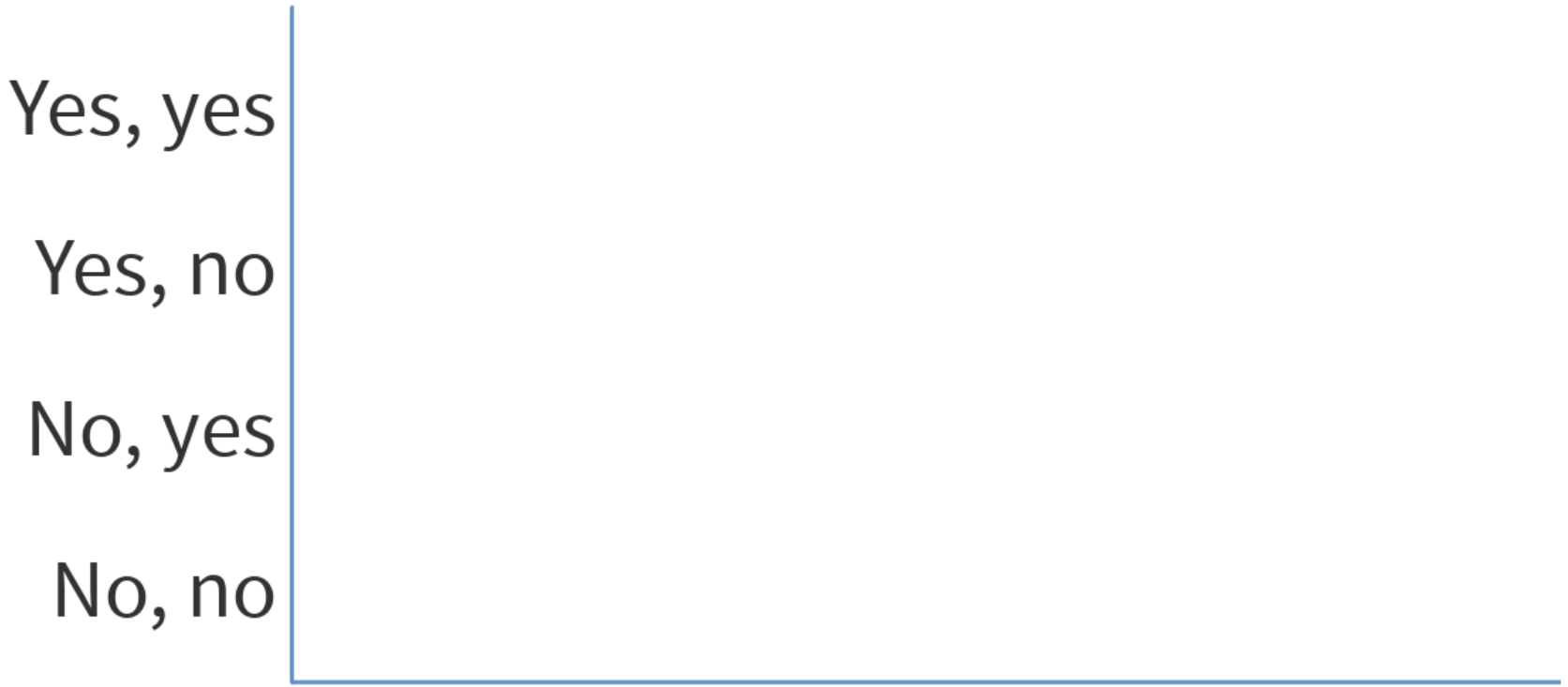
- **Lesson: simplicity first can pay off on practical datasets**
- Note that 1R does not perform as well on more recent, more sophisticated benchmark datasets

Constructing decision trees

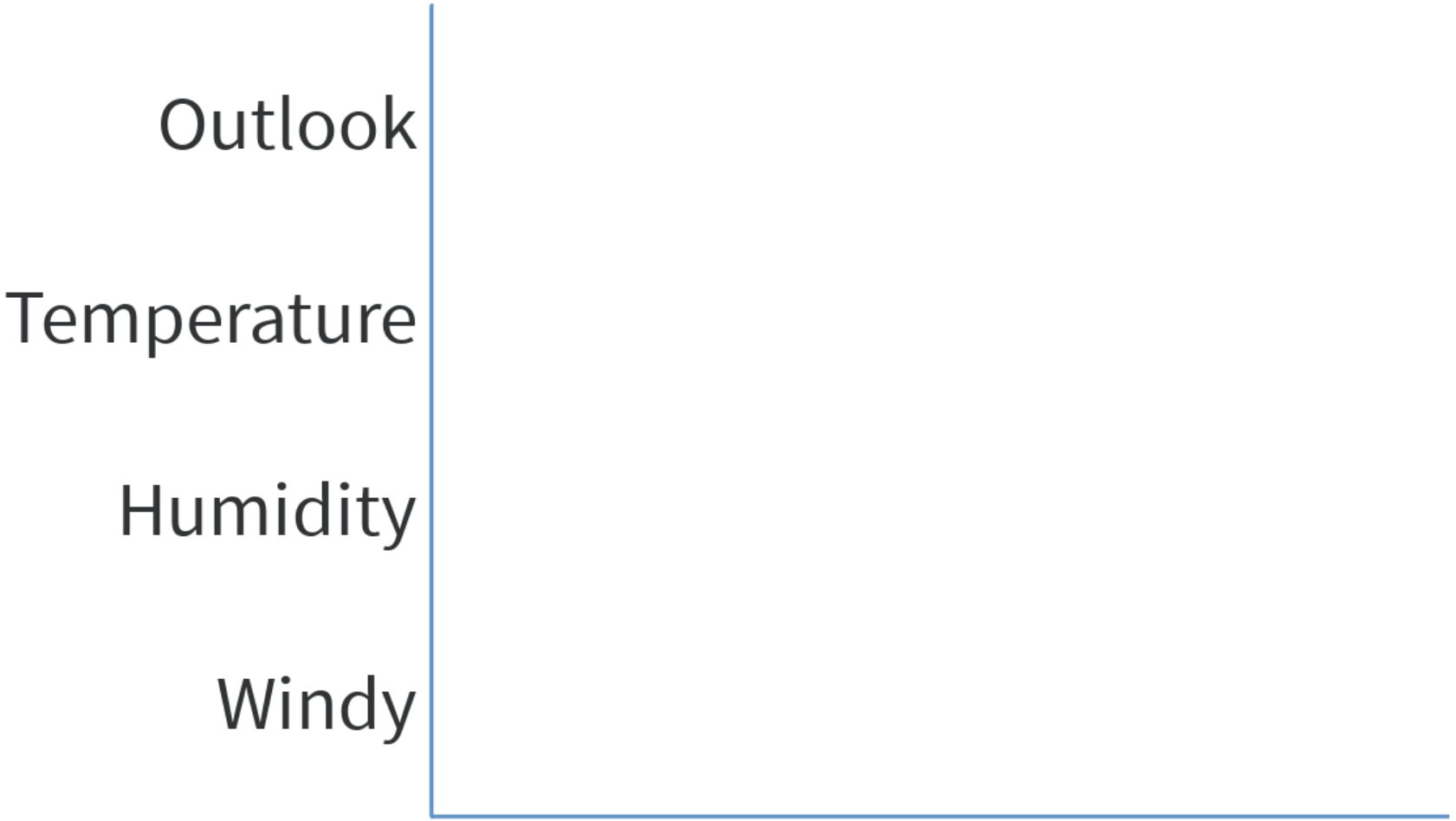
- Strategy: top down learning using recursive *divide-and-conquer* process
  - First: select attribute for root node
    Create branch for each possible attribute value
  - Then: split instances into subsets
    One for each branch extending from the node
  - Finally: repeat recursively for each branch, using only instances that reach the branch
- Stop if all instances have the same class

(For numeric attributes, compare to a constant value, e.g. $x_i > c$ *?*)

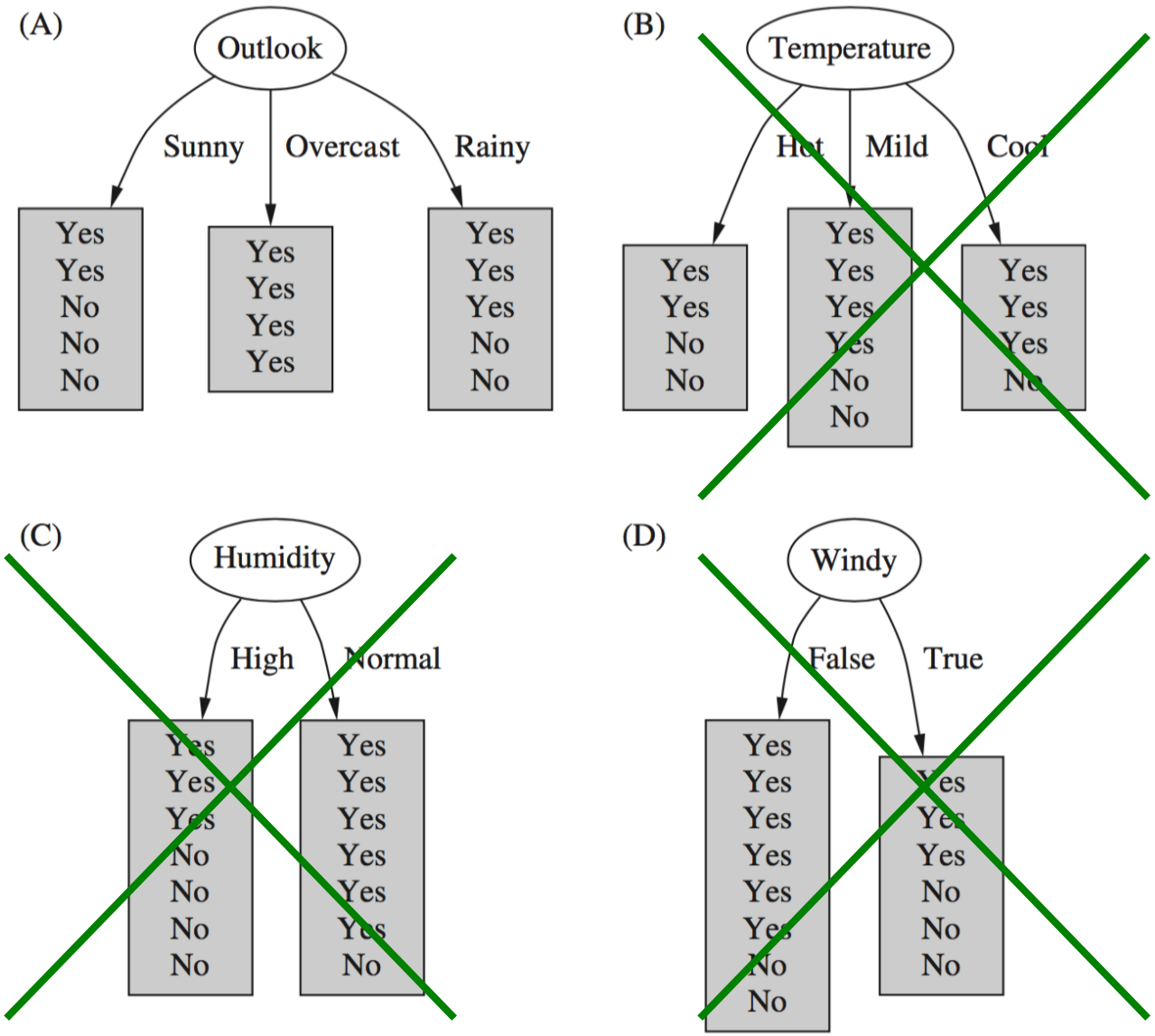
Does it make sense to split on an attribute more than once, for nominal attributes? What about for numeric attributes?



Which attribute to select?



Which attribute to select?



Criterion for attribute selection

- Which is the best attribute?
  - Want to get the smallest tree
  - Heuristic: choose the attribute that produces the “purest” nodes
- Popular selection criterion: ***information gain***
  - Information gain increases with the average purity of the subsets
- Strategy: amongst attributes available for splitting, choose attribute that gives greatest information gain
- Information gain requires measure of *impurity*
- Impurity measure that it uses is the ***entropy*** of the class distribution, which is a measure from information theory

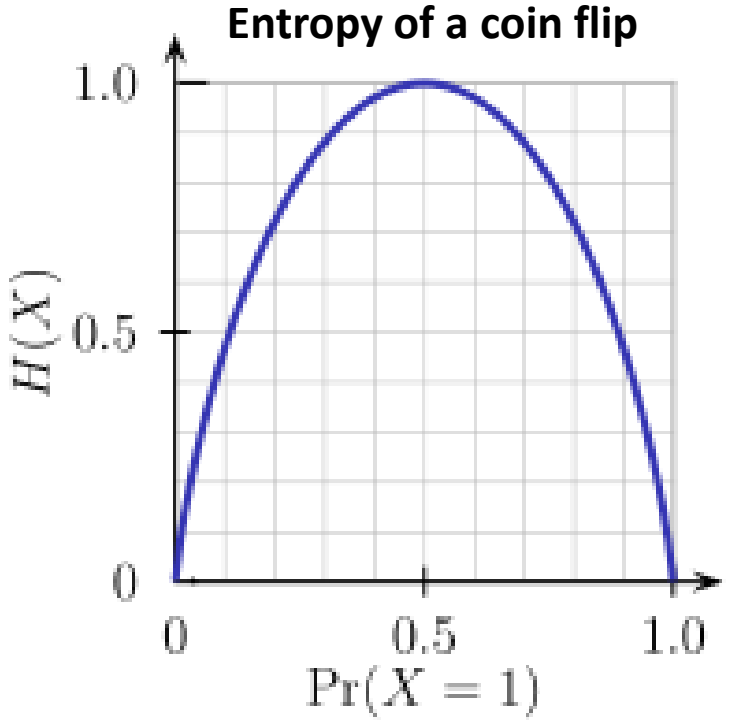
Wishlist for an impurity measure

- Properties we would like to see in an impurity measure:
  - When node is pure, measure should be zero
  - When impurity is maximal (i.e., all classes equally likely), measure should be maximal
  - Measure should ideally obey *multistage property* (i.e., decisions can be made in several stages, without affecting the overall impurity score):

- It can be shown that **entropy is the only function that satisfies all three properties!**

Brief Review of Entropy

- Entropy (Information Theory)
  - A measure of uncertainty associated with a random variable

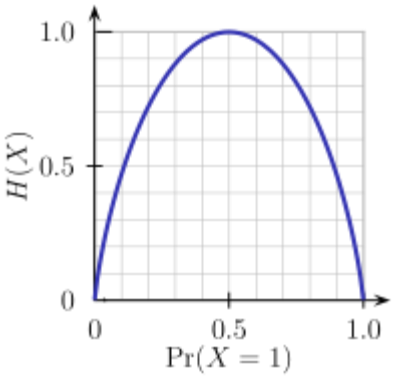


Brief Review of Entropy

- 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 $p_i = P(Y = y_i)$

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



Information Gain

- Select the attribute with the **highest information gain**
- Let p_i be the probability that an arbitrary tuple in D belongs to class C_i, estimated by $|C_{i,D}|/|D|$
- Expected information (entropy) needed to classify a tuple in D:

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

Information Gain

- Select the attribute with the **highest information gain**
- Let p_i be the probability that an arbitrary tuple in D belongs to class C_i, estimated by $|C_{i,D}|/|D|$
- Expected information (entropy) needed to classify a tuple in D:

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

- Information needed (after using A to split D into v partitions) to classify D:

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

Information Gain

- Select the attribute with the **highest information gain**
- Let p_i be the probability that an arbitrary tuple in D belongs to class C_i, estimated by $|C_{i,D}|/|D|$
- Expected information (entropy) needed to classify a tuple in D:

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

- Information needed (after using A to split D into v partitions) to classify D:

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

- Information gained by branching on attribute A

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

Example: attribute *Outlook*

Outlook	Temp	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No

- *Outlook* = *Sunny* :

$$\text{Info}([2, 3]) = 0.971 \text{ bits}$$

- *Outlook* = *Overcast* :

$$\text{Info}([4, 0]) = 0.0 \text{ bits}$$

- *Outlook* = *Rainy* :

$$\text{Info}([3, 2]) = 0.971 \text{ bits}$$

- Expected information for attribute:

$$\text{Info}([2, 3], [4, 0], [3, 2]) = (5/14) \times 0.971 + (4/14) \times 0 + (5/14) \times 0.971$$

$$= 0.693 \text{ bits}$$

Computing information gain

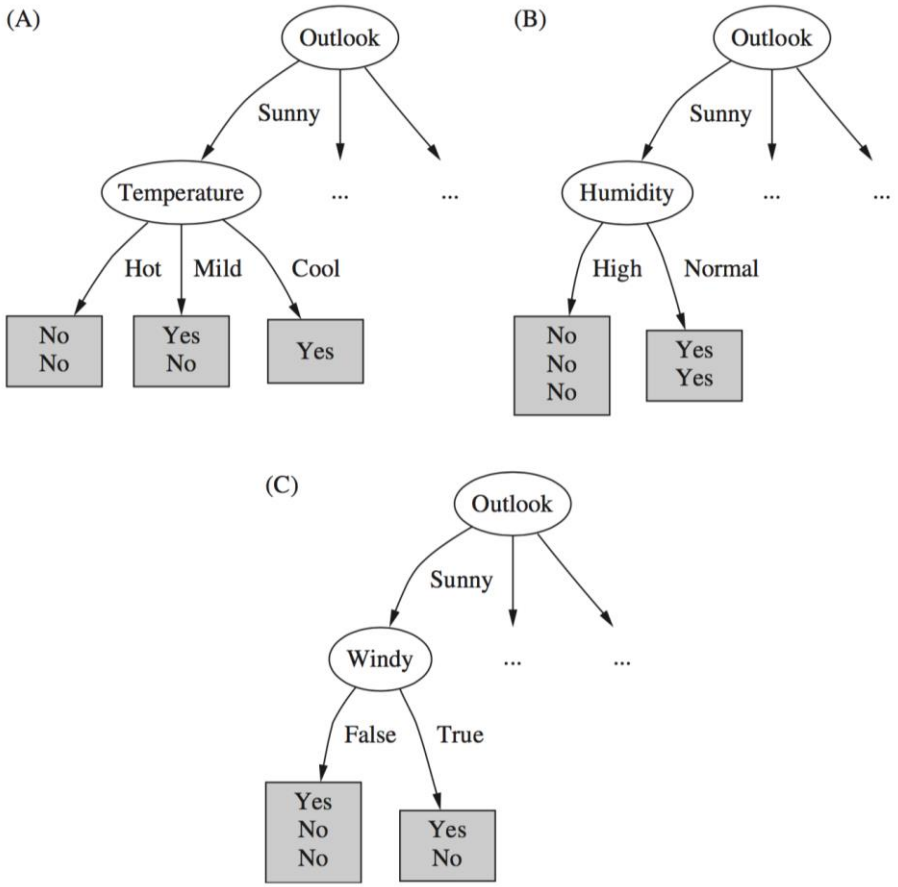
- Information gain: information before splitting – information after splitting

  Gain(*Outlook*) = Info([9,5]) – info([2,3],[4,0],[3,2])
  = 0.940 – 0.693
  = 0.247 bits

- Information gain for attributes from weather data:

  Gain(*Outlook*) = 0.247 bits
  Gain(*Temperature*) = 0.029 bits
  Gain(*Humidity*) = 0.152 bits
  Gain(*Windy*) = 0.048 bits

Continuing to split

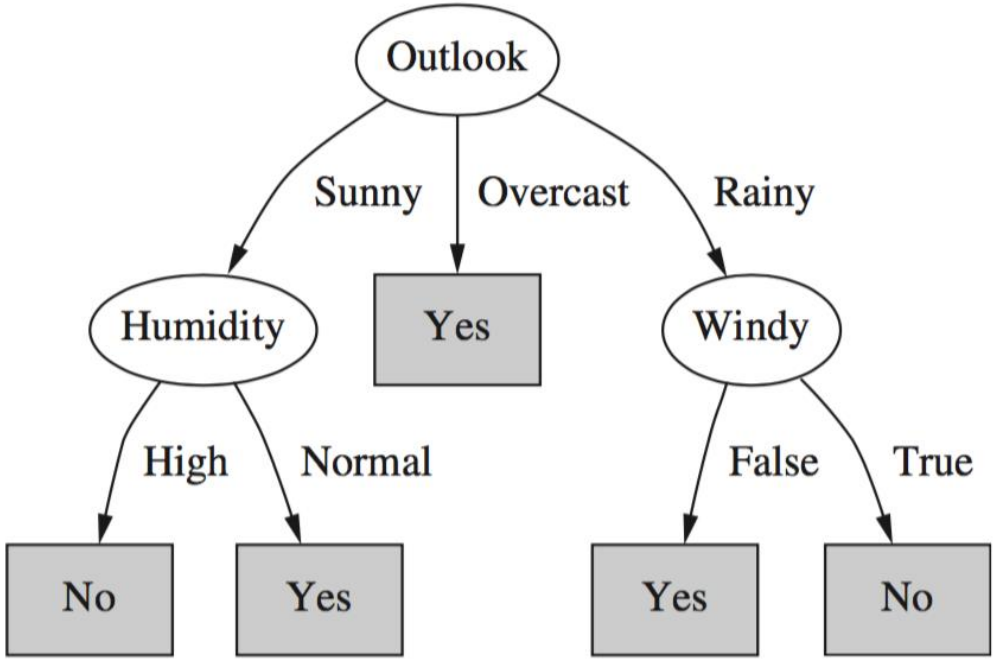


Gain(*Temperature*) = 0.571 bits

Gain(*Humidity*) = 0.971 bits

Gain(*Windy*) = 0.020 bits

Final decision tree



- Note: not all leaves need to be pure; sometimes identical instances have different classes
  - Splitting stops when data cannot be split any further

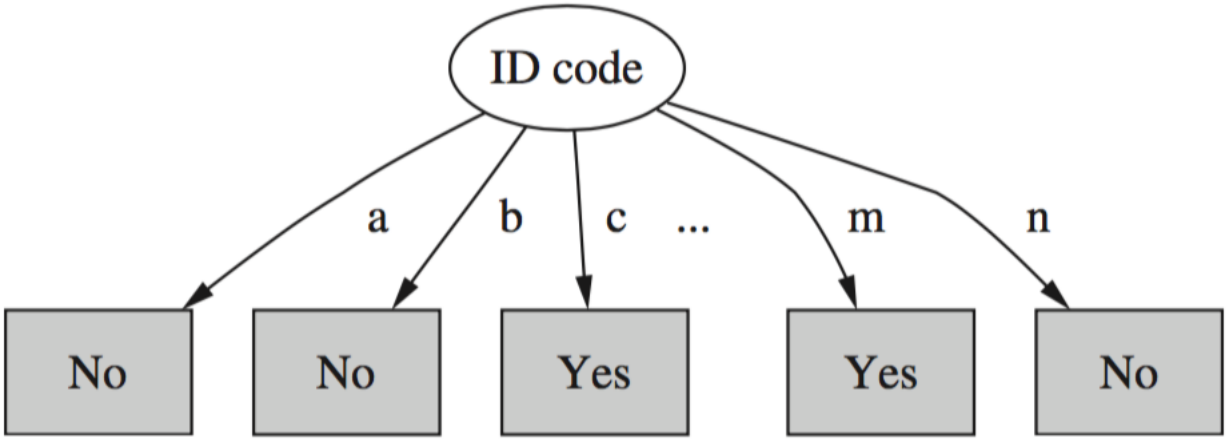
Highly-branching attributes

- Problematic: attributes with a large number of values (extreme case: ID code)
- Subsets are more likely to be pure if there is a large number of values
  - Information gain is biased towards choosing attributes with a large number of values
  - This may result in ***overfitting***
    - – the chosen attribute is very good at splitting the training data, but the model won't generalize well to new data

Weather data with *ID code*

ID code	Outlook	Temp.	Humidity	Windy	Play
A	Sunny	Hot	High	False	No
B	Sunny	Hot	High	True	No
C	Overcast	Hot	High	False	Yes
D	Rainy	Mild	High	False	Yes
E	Rainy	Cool	Normal	False	Yes
F	Rainy	Cool	Normal	True	No
G	Overcast	Cool	Normal	True	Yes
H	Sunny	Mild	High	False	No
I	Sunny	Cool	Normal	False	Yes
J	Rainy	Mild	Normal	False	Yes
K	Sunny	Mild	Normal	True	Yes
L	Overcast	Mild	High	True	Yes
M	Overcast	Hot	Normal	False	Yes
N	Rainy	Mild	High	True	No

Tree stump for *ID code* attribute



- All (single-instance) subsets have entropy zero!
- This means the information gain is maximal for this ID code attribute (namely 0.940 bits)

Gain ratio

- ***Gain ratio*** is a modification of the information gain that reduces its bias towards attributes with many values

- Gain ratio takes number and size of branches into account when choosing an attribute
  - It corrects the information gain by taking the *intrinsic information* of a split into account

Gain ratio

- **Intrinsic information (split info):** entropy of the distribution of instances into branches
- Measures how much info do we need to tell which branch a randomly chosen instance belongs to

$$\text{SplitInfo}(A) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \log_2 \frac{|D_j|}{|D|}$$

Computing the gain ratio

- Example: intrinsic information of ID code

$$\text{SplitInfo}(\text{ID}) = -\sum_{j=1}^{v} \frac{1}{14} \log_2 \frac{1}{14}$$

- Value of attribute should decrease as intrinsic information gets larger
- The ***gain ratio*** is defined as the **information gain** of the attribute, *divided by its* ***intrinsic information***
- Example (*outlook* at root node):

Gain: 0.940–0.693	0.247
Split info: info([5,4,5])	1.577
Gain ratio: 0.247/1.577	0.156

All gain ratios for the weather data

Outlook		Temperature	
Info:	0.693	Info:	0.911
Gain: 0.940-0.693	0.247	Gain: 0.940-0.911	0.029
Split info: info([5,4,5])	1.577	Split info: info([4,6,4])	1.557
Gain ratio: 0.247/1.577	0.157	Gain ratio: 0.029/1.557	0.019
Humidity		Windy	
Info:	0.788	Info:	0.892
Gain: 0.940-0.788	0.152	Gain: 0.940-0.892	0.048
Split info: info([7,7])	1.000	Split info: info([8,6])	0.985
Gain ratio: 0.152/1	0.152	Gain ratio: 0.048/0.985	0.049

"Outlook" is still the chosen attribute to split on

Heuristics for the Gain Ratio

- However, “ID code” has a greater gain ratio!
  - Standard fix: *ad hoc* test to prevent splitting on that type of identifier attribute
- Problem with gain ratio: it may **overcompensate**
  - May choose an attribute just because its intrinsic information is very low
  - Standard fix: only consider attributes with greater than average information gain
- Both fixes implemented in **C4.5** decision tree learner (**J48** in Weka)

ID3 Decision Tree Classifier

- **Representation**
  - Decision tree
- **Objective function for training**
  - Information gain of each split
- **Search algorithm**
  - Recursively construct tree, greedily selecting attributes to split on via information gain

Covering algorithms

- We've seen how to learn decision trees.
  What about decision rules?

- As we saw last week, we can always convert a decision tree into a rule set
  - However, the rule set will be overly complex

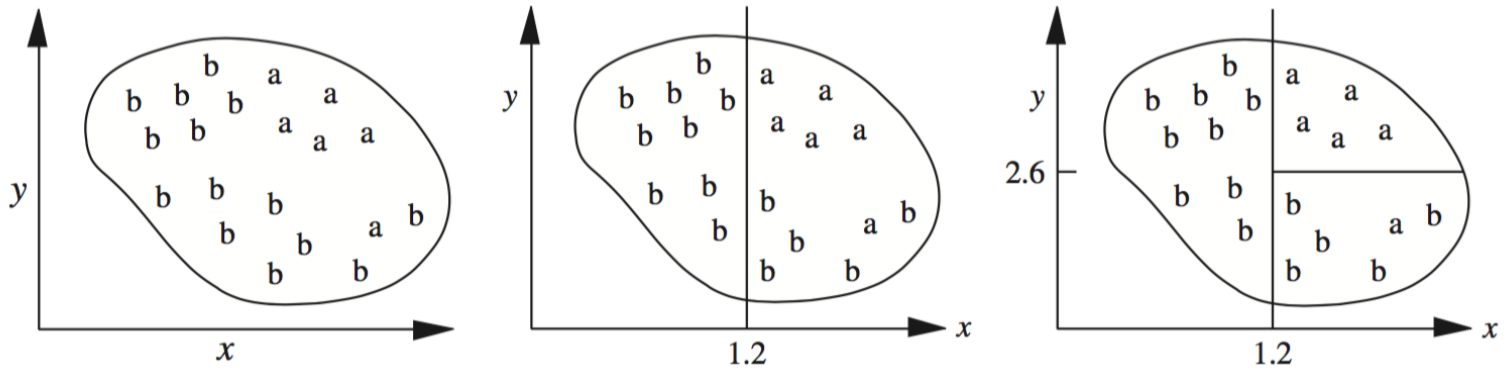
- **How do we learn a smaller, more interpretable set of rules?**

Covering algorithms

- Instead, we can generate rule set directly
  - One approach: for each class in turn, find rule set that covers all instances in it
    (excluding instances not in the class)

- Called a ***covering*** approach:
  - At each stage of the algorithm, a rule is identified that “covers” some of the instances

Example: generating a rule




```
If true
   then class = a
```

```
If x > 1.2
   then class = a
```

```
If x > 1.2 and y > 2.6
   then class = a
```

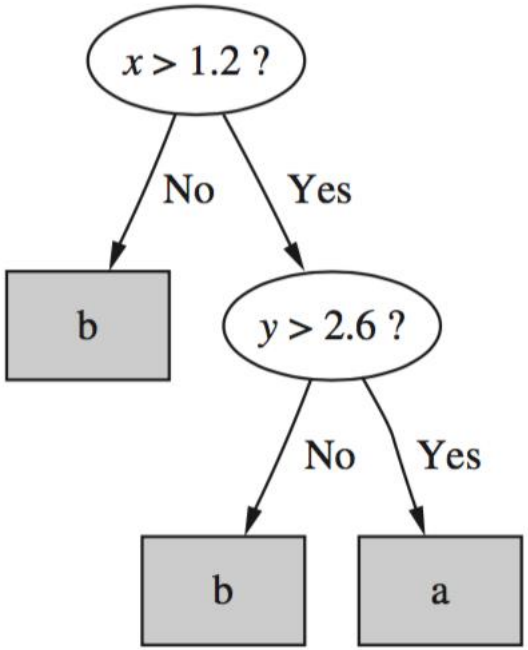
- Possible rule set for class “b”:

```
If x ≤ 1.2 then class = b
If x > 1.2 and y ≤ 2.6 then class = b
```

- Could add more rules, get “perfect” rule set

Rules vs. trees

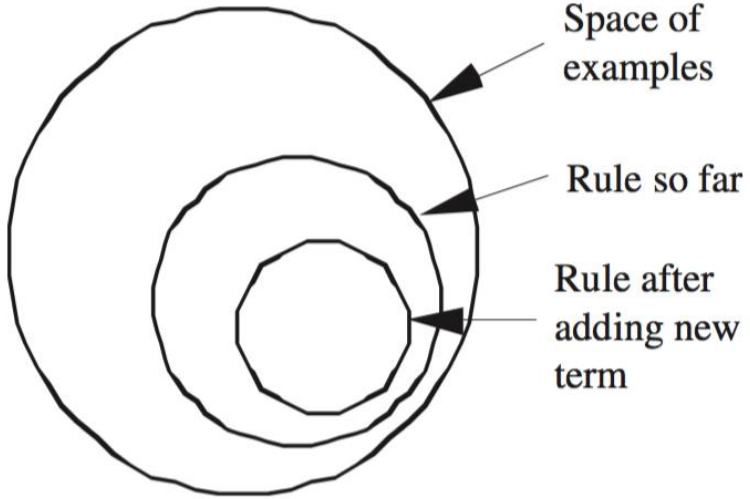
- Corresponding decision tree:
  (produces exactly the same
  predictions)



- But: rule sets *can* be more perspicuous when decision trees suffer from replicated subtrees
- Also: in multiclass situations, covering algorithm concentrates on one class at a time whereas decision tree learner takes all classes into account

Simple covering algorithm

- Basic idea: **generate a rule by adding tests that maximize the rule's accuracy**
- Similar to situation in decision trees: problem of selecting an attribute to split on
  - But: decision tree inducer maximizes overall purity
- Each new test reduces rule's coverage:



Example: contact lens data

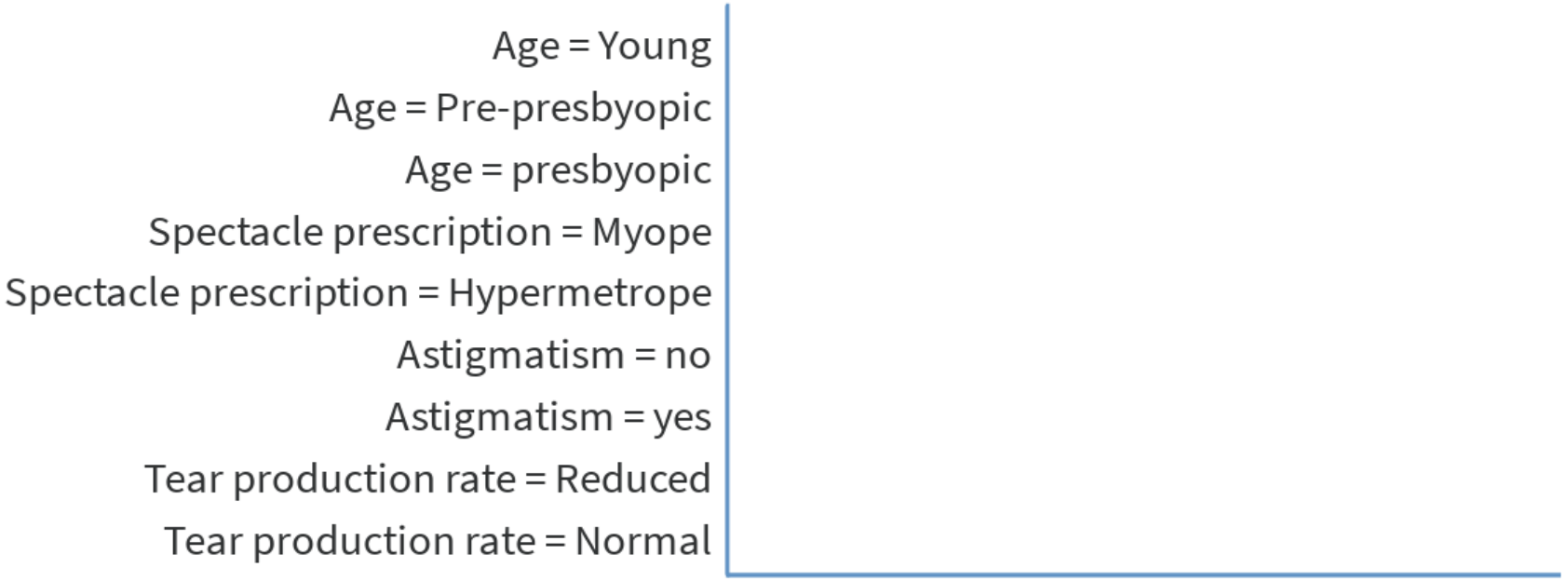
- Rule we seek:

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

- Possible tests:

Age = Young	2/8
Age = Pre-presbyopic	1/8
Age = Presbyopic	1/8
Spectacle prescription = Myope	3/12
Spectacle prescription = Hypermetrope	1/12
Astigmatism = no	0/12
Astigmatism = yes	4/12
Tear production rate = Reduced	0/12
Tear production rate = Normal	4/12

Which test should we add to the rule? (The numbers are p/t, p = number of positive examples of the class covered by rule, t = total number of instances covered by rule.)



Selecting a test

- Goal: maximize accuracy
  - *t* total number of instances covered by rule
  - *p* positive examples of the class covered by rule
  - *t* – *p* number of errors made by rule
  - Select test that maximizes the ratio *p/t*
- We are finished when *p/t* = 1 or the set of instances cannot be split any further

Modified rule and resulting data

- Rule with best test added:

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

- Instances covered by modified rule:

Age	Spectacle prescription	Astigmatism	Tear production rate	Recommended lenses
Young	Myope	Yes	Reduced	None
Young	Myope	Yes	Normal	Hard
Young	Hypermetrope	Yes	Reduced	None
Young	Hypermetrope	Yes	Normal	hard
Pre-presbyopic	Myope	Yes	Reduced	None
Pre-presbyopic	Myope	Yes	Normal	Hard
Pre-presbyopic	Hypermetrope	Yes	Reduced	None
Pre-presbyopic	Hypermetrope	Yes	Normal	None
Presbyopic	Myope	Yes	Reduced	None
Presbyopic	Myope	Yes	Normal	Hard
Presbyopic	Hypermetrope	Yes	Reduced	None
Presbyopic	Hypermetrope	Yes	Normal	None

Further refinement

- Current state:

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

- Possible tests:

```
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
```

Modified rule and resulting data

- Rule with best test added:

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

- Instances covered by modified rule:

Age	Spectacle prescription	Astigmatism	Tear production rate	Recommended lenses
Young	Myope	Yes	Normal	Hard
Young	Hypermetrope	Yes	Normal	hard
Pre-presbyopic	Myope	Yes	Normal	Hard
Pre-presbyopic	Hypermetrope	Yes	Normal	None
Presbyopic	Myope	Yes	Normal	Hard
Presbyopic	Hypermetrope	Yes	Normal	None

Further refinement

- Current state:

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

- Possible tests:

`Age = Young`	`2/2`
`Age = Pre-presbyopic`	`1/2`
`Age = Presbyopic`	`1/2`
`Spectacle prescription = Myope`	`3/3`
`Spectacle prescription = Hypermetrope`	`1/3`

- Tie between the first and the fourth test
  - **We break ties by choosing the one with greater coverage**

The final rule

- Final rule:

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

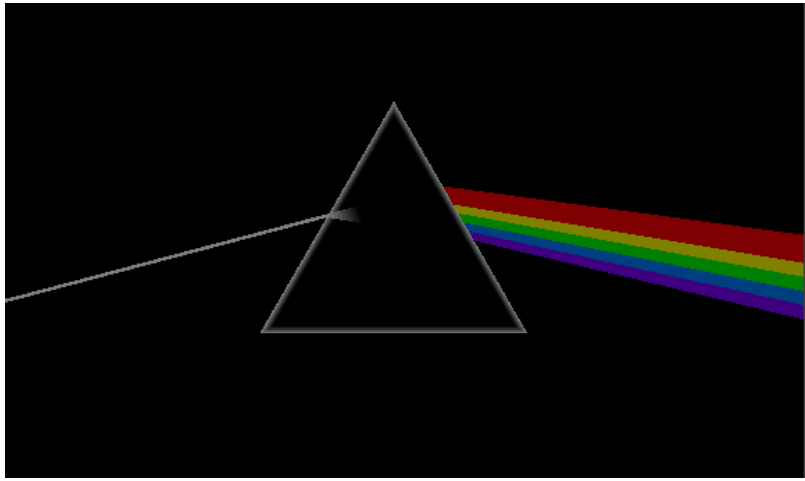
- Second rule for recommending “hard lenses”:
  (built from instances not covered by first rule)

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

- These two rules cover all “hard lenses”:
  - Process is repeated with other two classes

Pseudo-code for PRISM

```
For each class C
  Initialize E to the instance set
  While E contains instances in class C
    Create a rule R with an empty left-hand side that predicts class C
    Until R is perfect (or there are no more attributes to use) do
      For each attribute A not mentioned in R, and each value v,
        Consider adding the condition A = v to the left-hand side of R
        Select A and v to maximize the accuracy p/t
          (break ties by choosing the condition with the largest p)
      Add A = v to R
    Remove the instances covered by R from E
```



Rules vs. decision lists

- PRISM with outer loop removed generates a decision list for one class
  - Subsequent rules are designed for rules that are not covered by previous rules
  - But: order does not matter because all rules predict the same class so outcome does not change if rules are shuffled
- Outer loop considers all classes separately
  - No order dependence implied
- Problems: overlapping rules, default rule required

Separate and conquer rule learning

- Rule learning methods like the one PRISM employs (for each class) are called *separate-and-conquer* algorithms:
  - First, identify a useful rule
  - Then, separate out all the instances it covers
  - Finally, “conquer” the remaining instances
- Difference to divide-and-conquer methods:
  - Subset covered by a rule does not need to be explored any further

PRISM

- **Representation**
  - Rule set (order independent)
- **Objective function for training**
  - Accuracy (or error) on training data
- **Search algorithm**
  - Covering algorithm, applied to each class independently

Think-pair-share:
ID3 vs PRISM

- Please **compare and contrast ID3** and **PRISM** by identifying at least three (3) differences between the methods

Think-pair-share: ID3 vs PRISM

- Please **compare and contrast ID3** and **PRISM** by identifying at least three (3) differences between the methods

  - ID3- divide and conquer, PRISM- separate and conquer
  - decision tree (dependent order) vs rule set (independent)
  - PRISM uses default rules while ID3 does not
  - id3 gain information on each split, PRISM accuracy