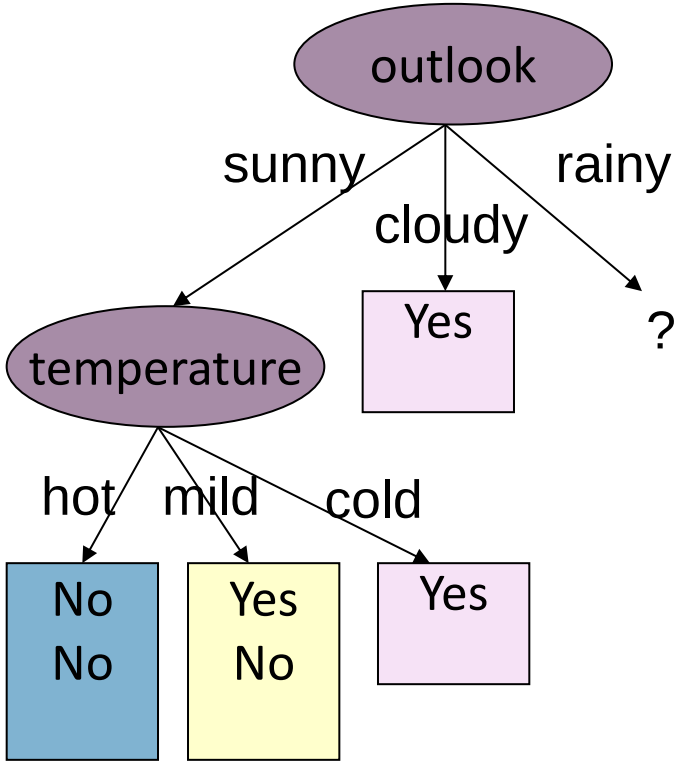
# CMM510: Data mining

# Decision tree algorithms

## - Improving performance

# Decision Tree (revision?)

- Turns a dataset into if-then-else statements so that predictions on new data can be made.
- Which attribute comes first (or next)?
- Which attribute gives most “information gain”?
- Or which gives a purer node (Gini index)?
- Lab: Caret and first classification model



# A word on last week's lab

- If you haven't tried it, try it today.
- Look for the "confusion matrix" for each model.
- Consider why models might be better or worse. "Getting more right" is usually considered "better" but not always.
- Is it better to get an unnecessary test for diabetes or get no test at all when you have it?

# Contents

- Lowering priority of high-branching nodes – gain ratio
- Coping with real data
  - numeric values
  - missing values
- Pruning – removing tree branches
  - noisy data
- Winnowing – feature selection
- Summary

# Decision Tree Family of Algorithms

**ID3 (public domain)**

Basic method for constructing decision trees

Nominal attributes, Gain Ratio

**C4.5 (public domain)**

Improved method for real applications

Numeric attributes, missing values, noisy data, rule generation from trees

**C5.0 (was commercial)**

Improved pruning

# Other tree-based algorithms

CART - classification and regression tree.

Uses *Gini Impurity* instead of *Information Gain*.

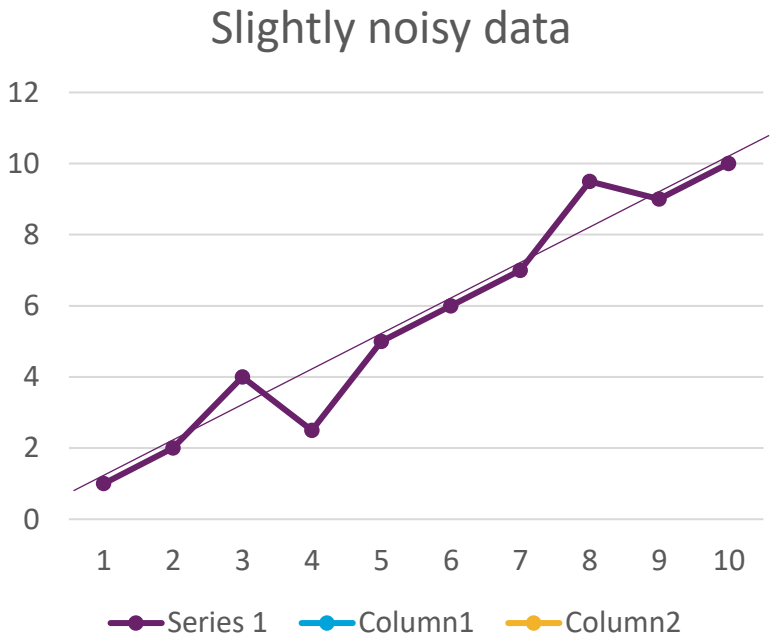
CHAID – CHi-squared automatic interaction detector.

Random forest – ensemble classification method based on decision trees (will see ensembles in another lecture).

- **Gini Impurity:** measure of how often a randomly chosen instance would be incorrectly classified if it was randomly assigned a class according to the distribution of the training set.

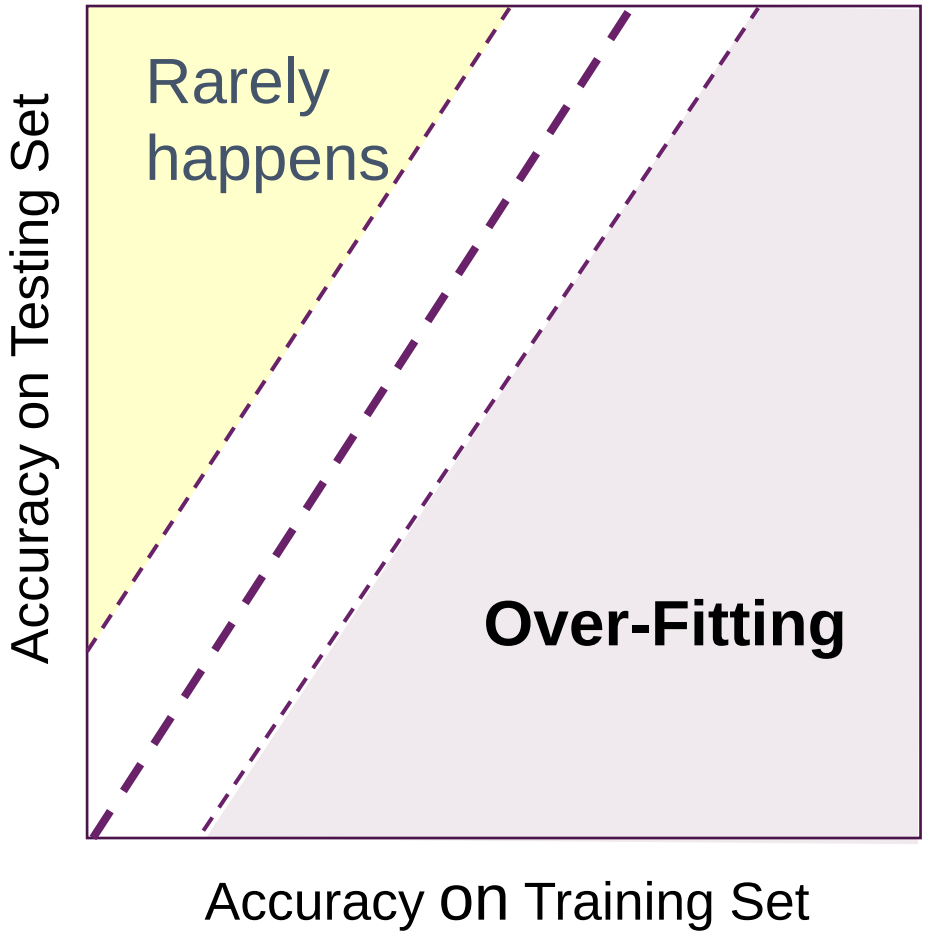
# Noise

- Any data which cannot be interpreted correctly
  - Corrupt data
  - Meaningless data
- Causes are varied
  - Sensor malfunction
  - External event causing data misreading
  - Deliberate input of erroneous data
  - … etc.
- Present in a large number of datasets
- Leads to overfitting



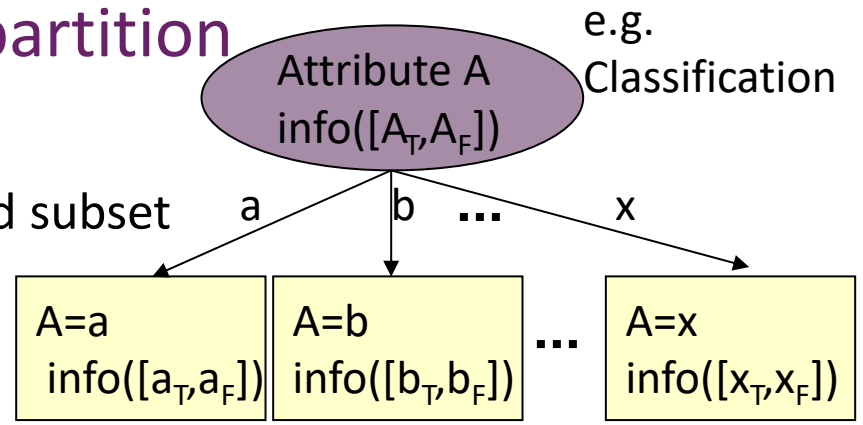
# Overfitting

- Model fits the training data too well.
  - Including any noise
- Model does not generalise well to other data (e.g. testing set).
- Model performance (e.g. accuracy) on training set is superior to model performance on testing set.



# Decision trees purity criteria: Information gain

- Information Gain - difference between
  - Information (entropy) contained in dataset (at node where attribute is considered for splitting)
  - Expected information from attribute split
- Expected Information from attribute splitting
  - weighted average of information from each partition
  - Is number of branches important?
    - Lots of branches – more chance of pure node, i.e. classified subset
    - Very few branches – less chance of pure node.



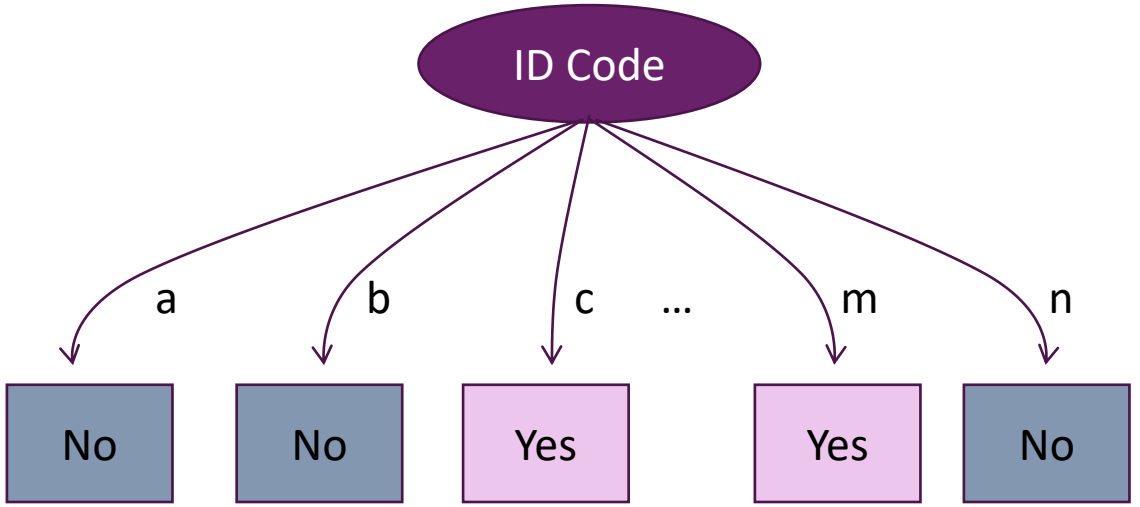
# Highly-branching attributes

- Subsets are more likely to be pure if there is a large number of values
  - More branches, so fewer values per branch
- Attributes with a large number of values likely to have higher information gain.
- Information gain is biased towards choosing attributes with a large number of values
  - This may result in *overfitting*
- Gain ratio compensates (a little!) for this bias

# Overfitting

- Happens when we split our dataset into too many tiny decisions.
- Also likely to happen as the number of attributes approaches the number of samples (i.e. the dataset is as wide as it is long)
- We have enough “names” that each of you probably has a unique name.

# Root Node: ID code



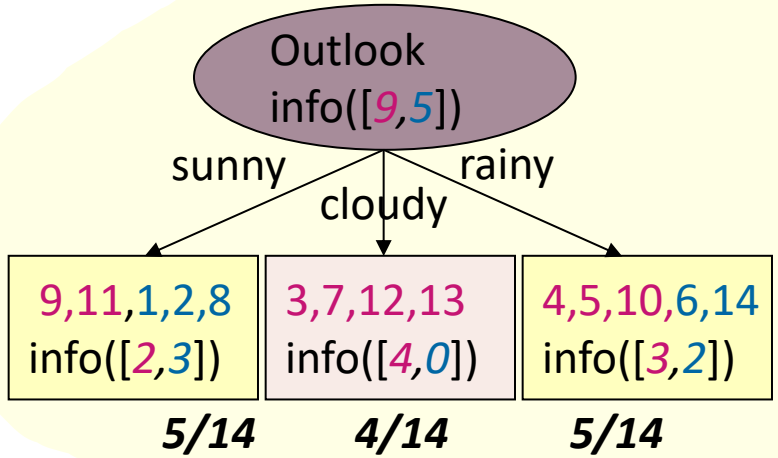

- Assume weather data has ID code
  - Each instance has unique ID code
  - ID codes a, b, c, …, n
  - Splitting on ID code leads to classified tree (max. information gain)
- info(Weather Data) = 0.940
- info(ID split) = info([0,1],[0,1],[1,0],…,[1,0],[0,1]) = 1/14*0+1/14*0+1/14*0+…+1/14*0+1/14*0
- Gain(ID) = 0.940 – 0 = 0.940 bits
  - Gain is maximal for ID code
- Not a useful decision node!

# Gain Ratio

- Takes number and size of branches into account when choosing an attribute
- Information gain
  - entropy with respect to classification
- Gain ratio normalises information gain with entropy in attribute partition
  - entropy with respect to attribute partition
    - ignores classification

# Example: Gain Ratio Outlook

- *Information as before (entropy wrt classification)*
  - *info(data) = info([9,5]) = 0.940*
  - *info(outlook) = 0.693*
- *Gain as before*
  - *gain(outlook)*
    *= 0.940 – 0.693 = 0.247*
- Information wrt Outlook partitions
  - 5 sunny, 4 cloudy, 5 rainy
  - info(data wrt outlook) = info([5,4,5])
    = -5/14*log2(5/14) – 4/14 log2(4/14) – 5/14*log2(5/14) = 1.563
- GainRatio(outlook) =
  gain(outlook)/info(data wrt outlook)= 0.247/1.563 = 0.158

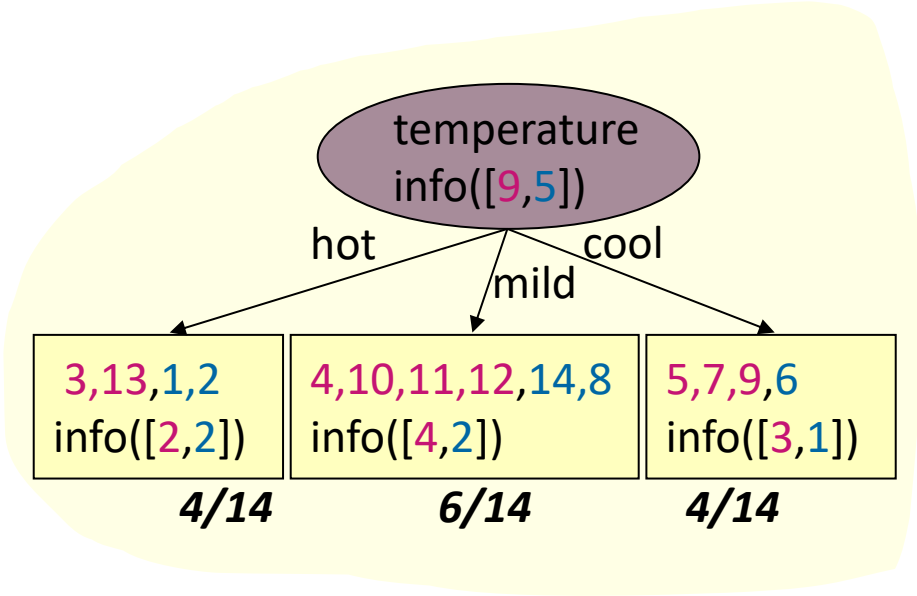



| # | Outlook | Temperature | Humidity | Windy | Play |
|---|---|---|---|---|---|
| 1 | Sunny | Hot | High | False | No |
| 2 | Sunny | Hot | High | True | No |
| 3 | Cloudy | Hot | High | False | Yes |
| 4 | Rainy | Mild | High | False | Yes |
| 5 | Rainy | Cool | Normal | False | Yes |
| 6 | Rainy | Cool | Normal | True | No |
| 7 | Cloudy | Cool | Normal | True | Yes |
| 8 | Sunny | Mild | High | False | No |
| 9 | Sunny | Cool | Normal | False | Yes |
| 10 | Rainy | Mild | Normal | False | Yes |
| 11 | Sunny | Mild | Normal | True | Yes |
| 12 | Cloudy | Mild | High | True | Yes |
| 13 | Cloudy | Hot | Normal | False | Yes |
| 14 | Rainy | Mild | High | True | No |

# Example: Gain Ratio Temp

- *Gain*
  - *info(data) = 0.940*
  - *info(temp) = 0.911 bits*
  - *gain(temp) = 0.029 bits*
- Information wrt temperature partitions
  - 4 hot, 6 mild, 4 cool
  - info(data wrt temp) = info([4,6,4])
    $=-4/14*\log_2(4/14) - 6/14\ \log_2(6/14) - 4/14*\log_2(4/14)$
    $= 1.545$
- GainRatio(temp)

  gain(temp)/info(data wrt temp) = 0.029/1.545 = 0.019

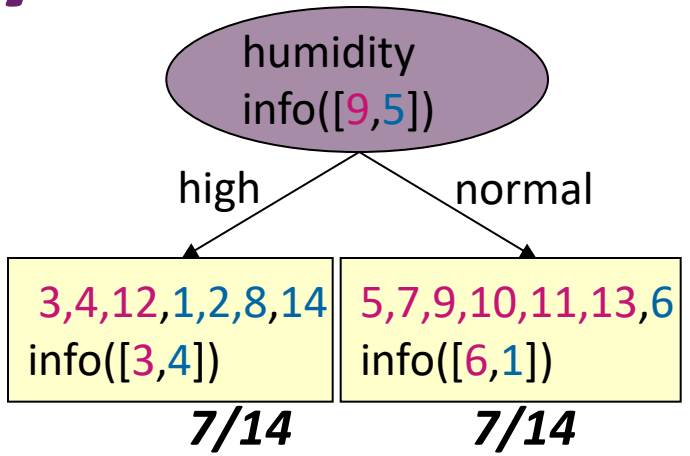



| # | Outlook | Temperature | Humidity | Windy | Play |
|---|---|---|---|---|---|
| 1 | Sunny | Hot | High | False | No |
| 2 | Sunny | Hot | High | True | No |
| 3 | Cloudy | Hot | High | False | Yes |
| 4 | Rainy | Mild | High | False | Yes |
| 5 | Rainy | Cool | Normal | False | Yes |
| 6 | Rainy | Cool | Normal | True | No |
| 7 | Cloudy | Cool | Normal | True | Yes |
| 8 | Sunny | Mild | High | False | No |
| 9 | Sunny | Cool | Normal | False | Yes |
| 10 | Rainy | Mild | Normal | False | Yes |
| 11 | Sunny | Mild | Normal | True | Yes |
| 12 | Cloudy | Mild | High | True | Yes |
| 13 | Cloudy | Hot | Normal | False | Yes |
| 14 | Rainy | Mild | High | True | No |

# Example: Gain Ratio Humidity

- *Gain*
  - *info(data) = 0.940*
  - *info(humidity) = 0.790 bits*
  - *gain(humidity) = 0.151 bits*
- Information wrt humidity partitions
  - 7 high, 7 normal
  - Info(data wrt humidity) = info([7,7])
    = $-7/14*\log_2(7/14) - 7/14*\log_2(7/14) = 1$
- GainRatio(humidity)

  gain(humidity)/info(data wrt humidity) = 0.151/1 = 0.151

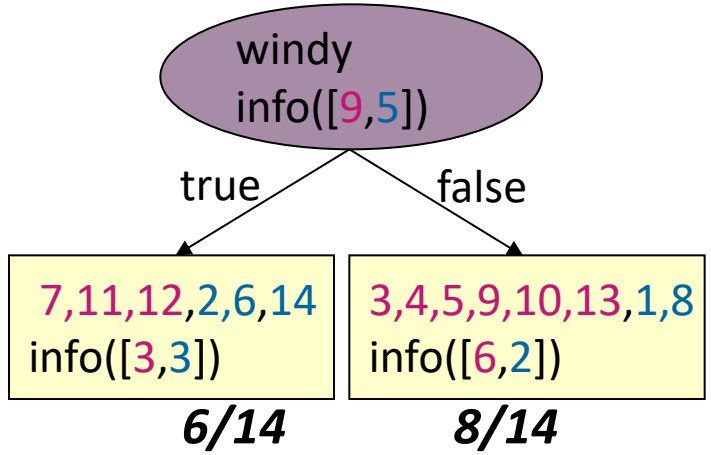



| # | Outlook | Temperature | Humidity | Windy | Play |
|---|---|---|---|---|---|
| 1 | Sunny | Hot | High | False | No |
| 2 | Sunny | Hot | High | True | No |
| 3 | Cloudy | Hot | High | False | Yes |
| 4 | Rainy | Mild | High | False | Yes |
| 5 | Rainy | Cool | Normal | False | Yes |
| 6 | Rainy | Cool | Normal | True | No |
| 7 | Cloudy | Cool | Normal | True | Yes |
| 8 | Sunny | Mild | High | False | No |
| 9 | Sunny | Cool | Normal | False | Yes |
| 10 | Rainy | Mild | Normal | False | Yes |
| 11 | Sunny | Mild | Normal | True | Yes |
| 12 | Cloudy | Mild | High | True | Yes |
| 13 | Cloudy | Hot | Normal | False | Yes |
| 14 | Rainy | Mild | High | True | No |

# Example: Gain Ratio Windy

- *Gain*
  - *info(data) = 0.940*
  - *info(windy) = 0.892 bits*
  - *gain(windy) = 0.048 bits*
- Information wrt windy partitions
  - 6 true, 8 false
  - Info(data wrt windy) = info([6,8])
    $=-6/14*\log_2(6/14) - 8/14\ \log_2(8/14) = 0.986$
- GainRatio(windy)

  gain(windy)/info(data wrt windy)= 0.048/0.986 = 0.049




| # | Outlook | Temperature | Humidity | Windy | Play |
|---|---|---|---|---|---|
| 1 | Sunny | Hot | High | False | No |
| 2 | Sunny | Hot | High | True | No |
| 3 | Cloudy | Hot | High | False | Yes |
| 4 | Rainy | Mild | High | False | Yes |
| 5 | Rainy | Cool | Normal | False | Yes |
| 6 | Rainy | Cool | Normal | True | No |
| 7 | Cloudy | Cool | Normal | True | Yes |
| 8 | Sunny | Mild | High | False | No |
| 9 | Sunny | Cool | Normal | False | Yes |
| 10 | Rainy | Mild | Normal | False | Yes |
| 11 | Sunny | Mild | Normal | True | Yes |
| 12 | Cloudy | Mild | High | True | Yes |
| 13 | Cloudy | Hot | Normal | False | Yes |
| 14 | Rainy | Mild | High | True | No |

# Example: Gain Ratios

- Gain Ratios
  - GainRatio(outlook) = 0.247/1.563 = 0.158
  - GainRatio(temp) = 0.029/1.545 = 0.019
  - GainRatio(humidity) = 0.151/1 = 0.151
  - GainRatio(windy) = 0.048/0.986 = 0.049
- Outlook still selected as root node
  - But humidity is much closer choice

# Problems with Gain Ratio

- Extreme scenario with very high number of values
  - reject attribute even if high gain or gain ratio
- Gain ratio may overcompensate
  - may choose an attribute because its intrinsic information is very low
  - Standard solution
    - only consider attributes with greater than average information gain
      - calculate information gain for all attributes
      - calculate gain ratio for those with > average gain
      - choose attribute with highest gain ratio

# Contents (2)

# Numeric Attributes

- Standard method
  - Binary splits (e.g. temp < 45).
- Different from nominal attributes
  - Every attribute has many possible split points.
- Solution is straightforward extension
  - Evaluate information gain (or gain ratio) for every possible split point of attribute.
  - Choose “best” split point.
  - Gain for best split point is gain for attribute.
- Computationally more demanding
  - Many possible split points.

# Example – numeric values

| Outlook | Temperature | Humidity | Windy | Play |
|---|---|---|---|---|
| Sunny | 85 | 85 | False | No |
| Sunny | 80 | 90 | True | No |
| Cloudy | 83 | 86 | False | Yes |
| Rainy | 70 | 96 | False | Yes |
| Rainy | 68 | 80 | False | Yes |
| Rainy | 65 | 70 | True | No |
| Cloudy | 64 | 65 | True | Yes |
| Sunny | 72 | 95 | False | No |
| Sunny | 69 | 70 | False | Yes |
| Rainy | 75 | 80 | False | Yes |
| Sunny | 75 | 70 | True | Yes |
| Cloudy | 72 | 90 | True | Yes |
| Cloudy | 81 | 75 | False | Yes |
| Rainy | 71 | 91 | True | no |

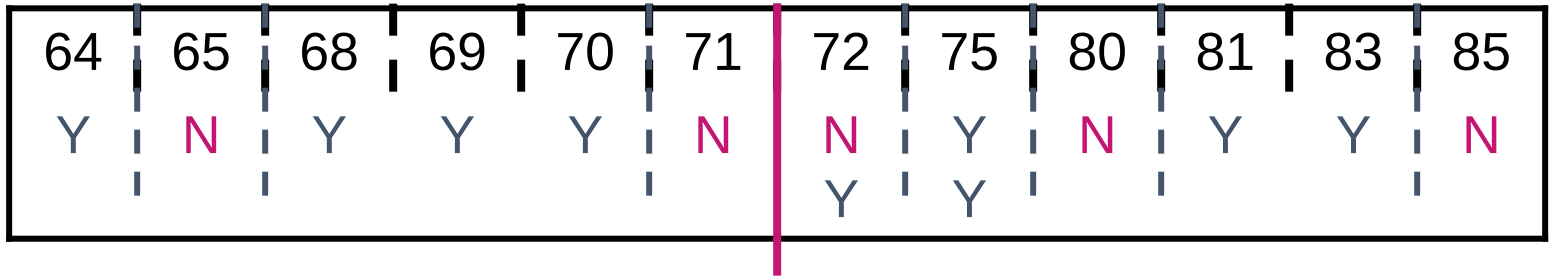
May want to standardise or normalise numeric values BEFORE tree-building.

Maximum value

Minimum value

# Weather Example

- Split on temperature attribute according to class value
  - Order values (keep class)
  - Split points
    - Mid point between
      2 values
      where class changes

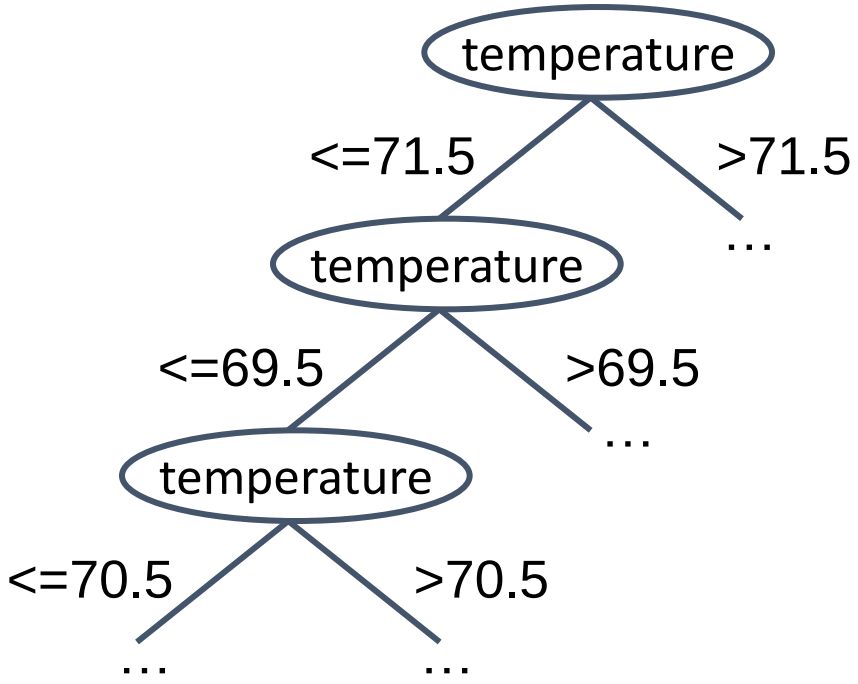



- Normal info gain calculation
  - temperature < 71.5: 4 yes and 2 no
  - temperature >= 71.5: 5 yes and 3 no
  - I([4,2],[5,3])=6/14 * I([4,2])+8/14 * I([5,3]) = 0.939
- But do for each split point
  - 64.5, 66.5, 70.5, 71.5, 73.5, 77.5, 80.5, 84

# Notes on Binary Splits

- Nominal attributes
  - Information is exhausted using one multi-way split
- Not so for binary splits on numeric attributes
  - Same numeric attribute may be tested several times along path
- Disadvantage: tree relatively hard to read
- Possible remedies
  - Pre-discretisation of numeric attributes
  - multi-way splits instead of binary ones

# … Numeric Attributes




- Numeric attributes can be used many times as the root of the subtree

# Contents (3)

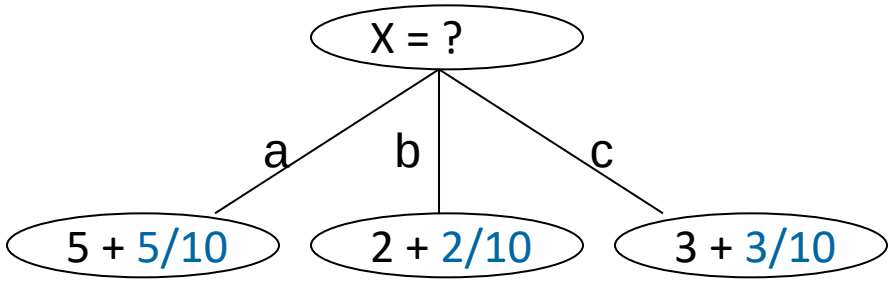
- Lowering priority of high-branching nodes
- Coping with real data
  - numeric values
  - missing values
- Pruning – removing tree branches
  - noisy data
- Summary

# Missing Values

- Represented as NA in R
  - Create a new value if “being missing” is significant.
  - Otherwise, utilise other values in the instance without highlighting missing value.
- C5 splits instances with missing values into pieces when
  - Learning decision tree.
  - Applying decision tree to classify new problems.

# Missing Values: Learning

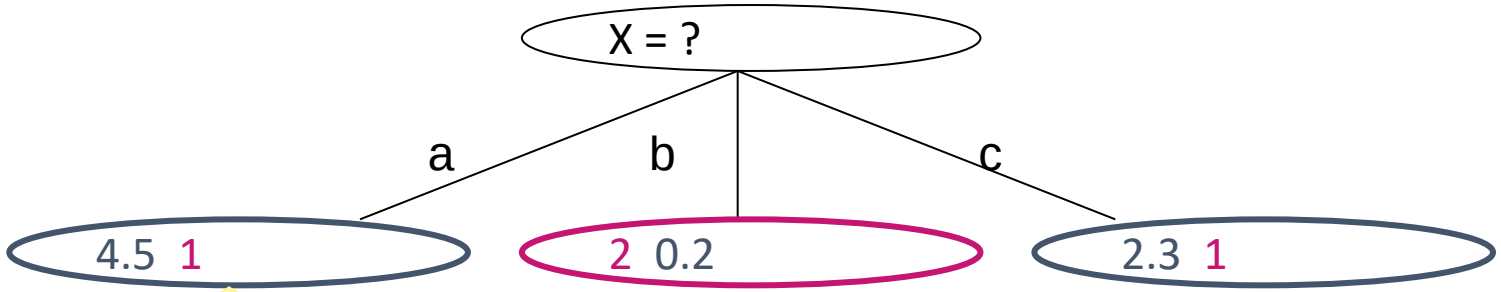
- Sums of weights not counts
  - Split for information gain
    - e.g. 11 instances including 1 with a missing value




  - 5.5+2.2+3.3=11

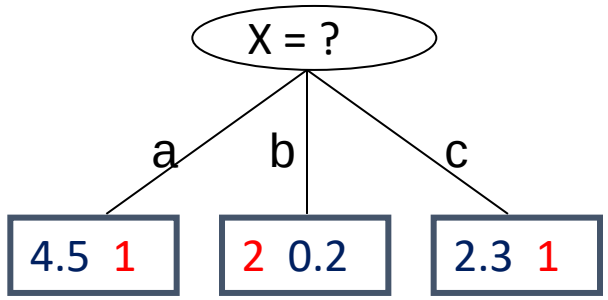
# … Missing Values: Learning

- Split instance for class distribution
  - e.g. binary class -Yes, No
- Assume that instance with missing value is of class Yes




- Info can be calculated as before e.g. Info(7, 4), info(4.5,1), etc.

# Missing Values: Classification




- Same procedure
- Probability distributions merged using weights
  - e.g. X is unknown for new problem
  - Class is Blue?
    - Weight = (5.5*4.5+2.2*0.2+3.3*2.3)/11

      = 0.5*4.5+0.2*0.2+0.3*2.3 = 2.25+0.04+0.69 = **2.98**
  - Class is Red?
    - Weight = (5.5*1+2.2*2+3.3*1)/11

      = 0.5*1+0.2*2+0.3*1 = 0.5+0.4+0.3 = **1.2**

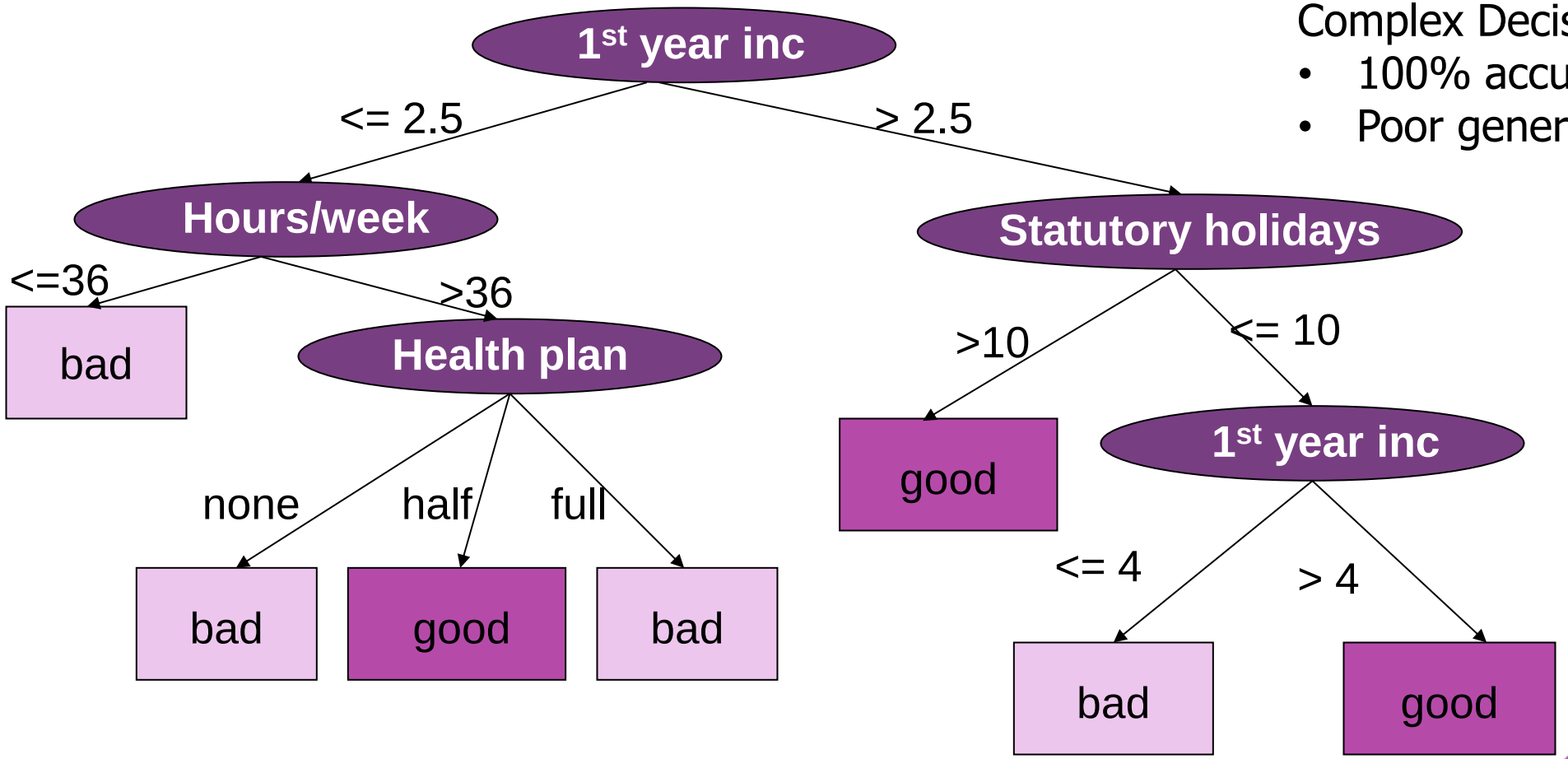
# Contents (4)

- Lowering priority of high-branching nodes
- Coping with real data
  - numeric values
  - missing values
- Pruning – removing tree branches
  - noisy data
- Winnowing – feature selection
- Summary

# Problems with Noisy Data

- Noisy data
  - Wrong attribute values are assigned to instances.
  - Wrong class is assigned to instances.
- Some tree branches may be undesirable
  - Caused by noisy data.
  - Use irrelevant features.
- Pruning simplifies decision tree
  - Prevents over-fitting to noise in data.
  - Prevents over-fitting sparse data.

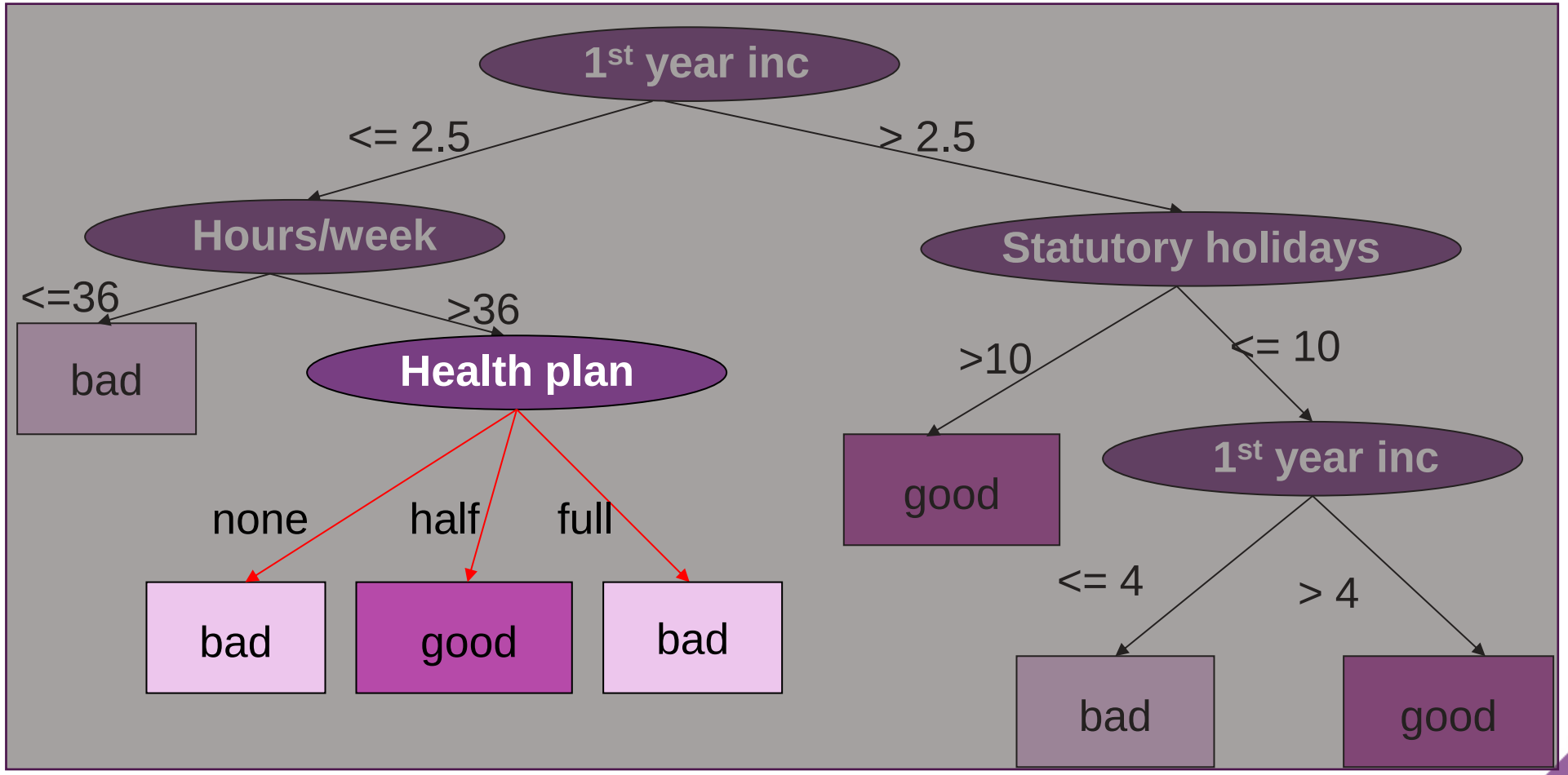
# Decision Tree – example of overfitting



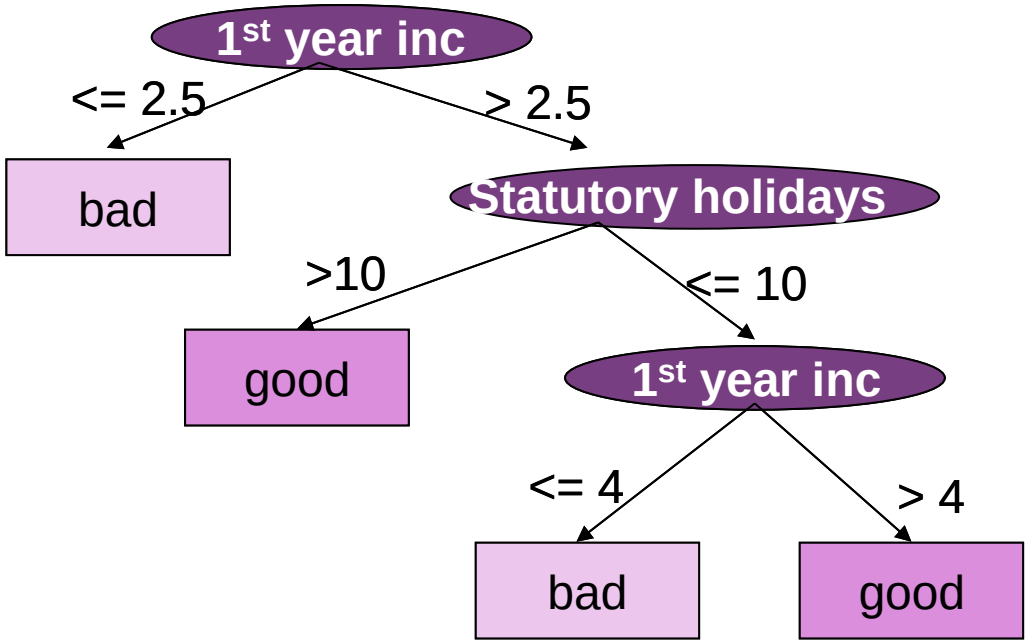

Complex Decision Tree

- 100% accuracy on training set
- Poor generalisation

# Decision Tree – example of overfitting (2)



# Simpler tree – no overfitting




Simple Decision Tree

- <100% accuracy on training set
- good generalisation

# Pruning

- Two strategies
  - Pre-pruning (forward/on-line pruning)
    - stops growing a branch when information becomes unreliable
  - Post-pruning (backward pruning)
    - takes a fully-grown decision tree and discards unreliable parts

# Pre-Pruning

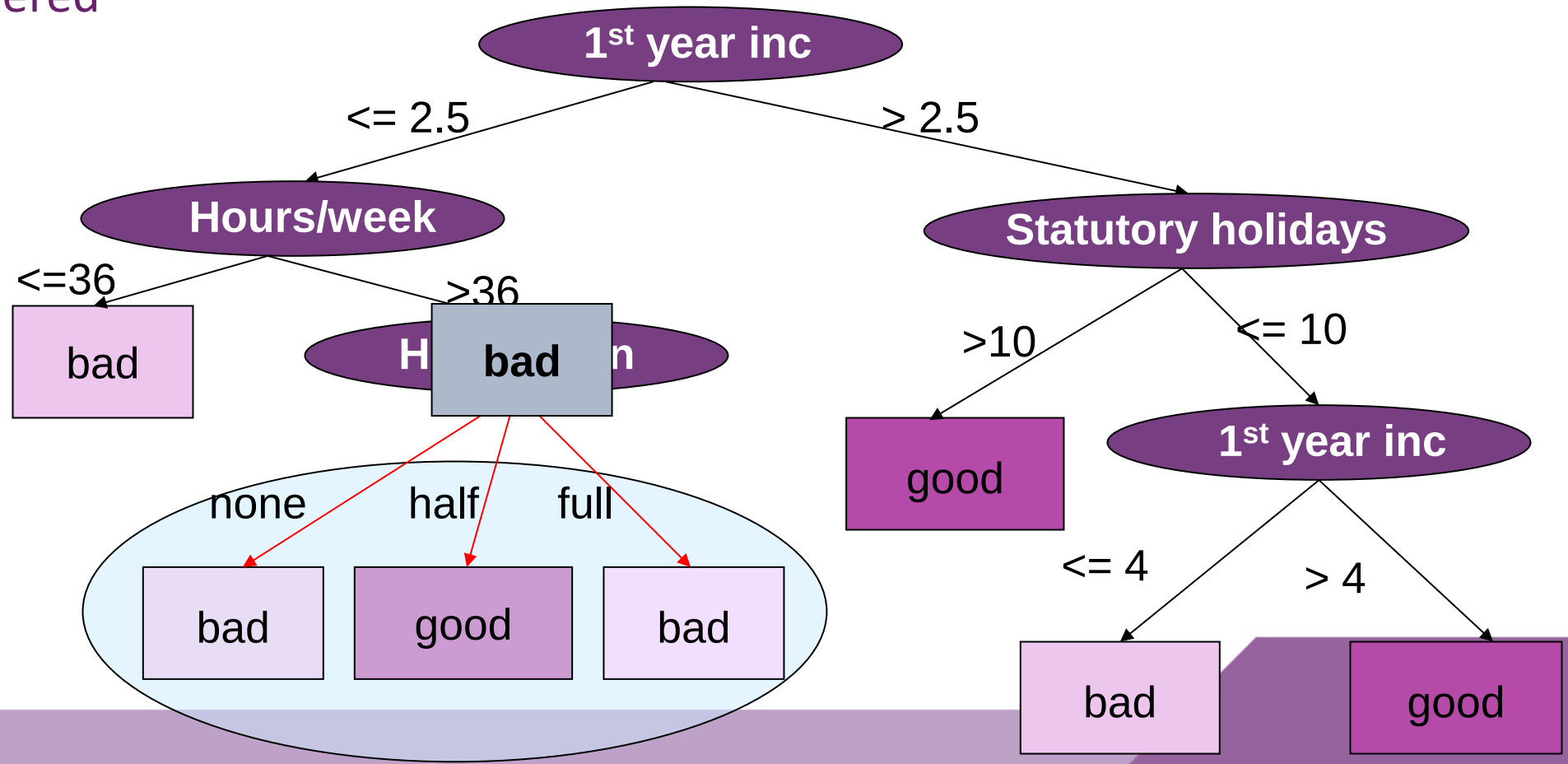
- Stops growing tree
  - Information gain for all attributes is below threshold.
  - Further growing does not improve leaf error threshold.
- Minimum number of instances (cases) in the (2) most popular branches from a node.
  - E.g. instance threshold of 3
    - minimum number of 3 instances in 2 most popular branches
  - Increase –this parameter  if a lot of noisy data
- Pre-pruning may suffer early stopping
  - Growth may stop prematurely
  - May miss out on interactions between attributes i.e. misses out on the combination-lock effect

# Post-Pruning

- Builds full tree first and prunes afterwards
  - Attribute interactions are visible in fully-grown tree.
  - When are subtrees due to chance effects?
  - Error estimation determines whether to prune.
- Two main pruning operations
  - Subtree replacement
    - Bottom-up
    - Tree considered for replacement once all subtrees are done
  - Subtree raising
    - Generally more expensive (complicated!)

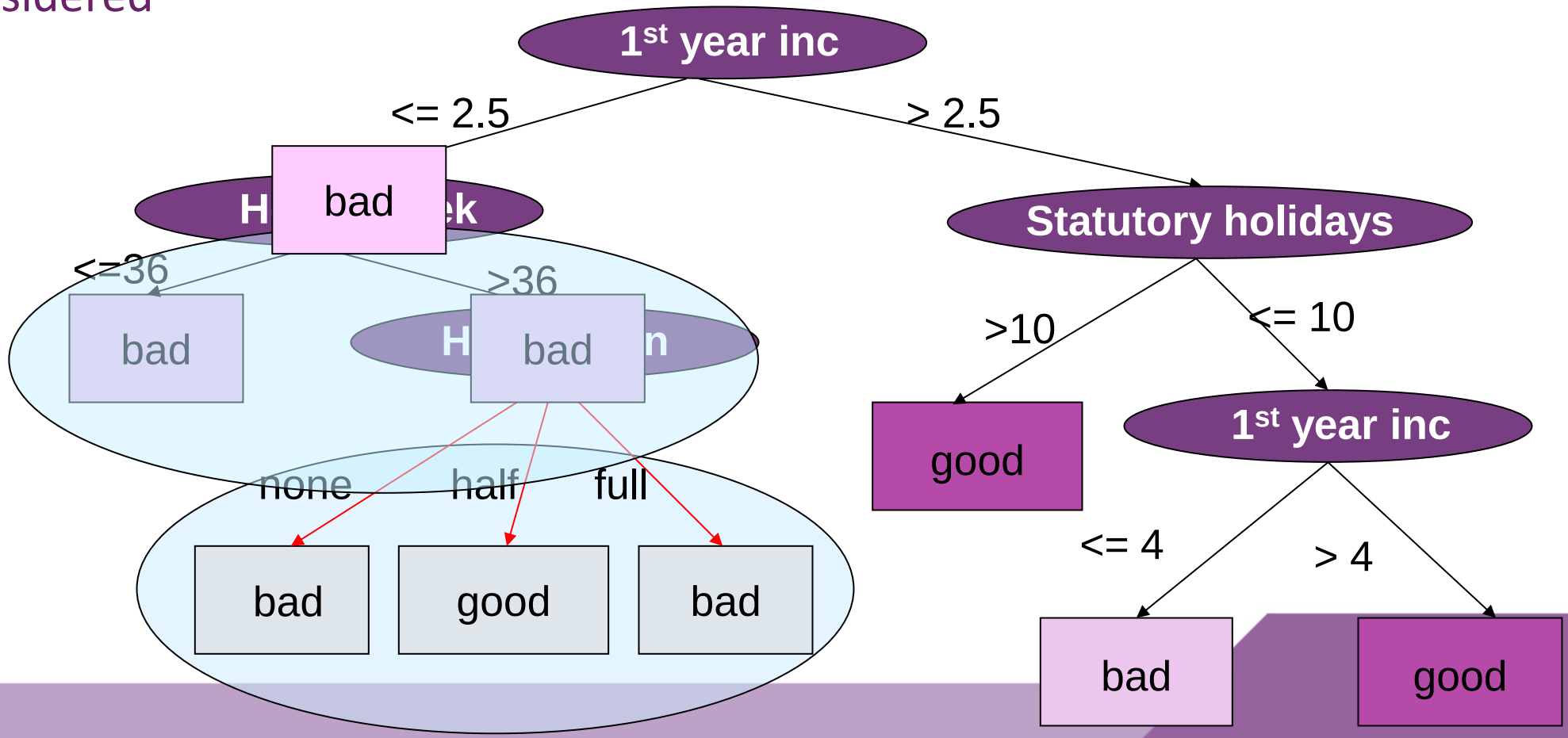
# Post-Pruning: Subtree Replacement

- Bottom-up: Tree considered for replacement once all subtrees have been considered



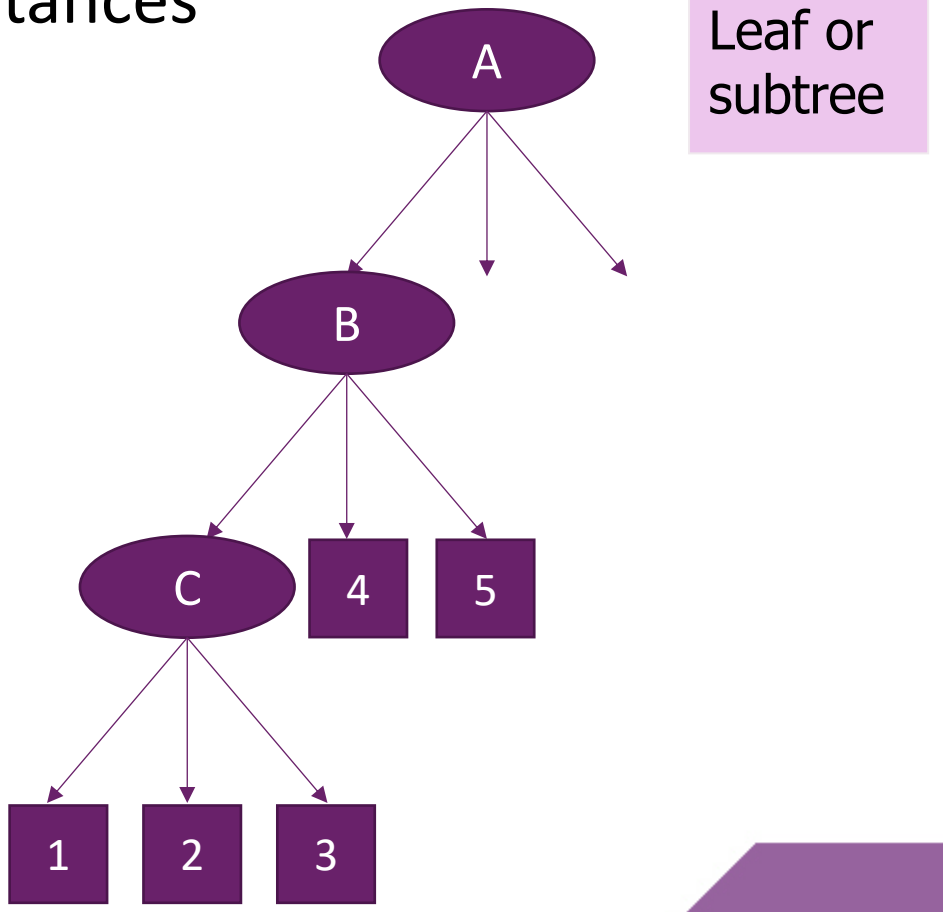
# Post-Pruning: Subtree Replacement (2)

- Bottom-up: Tree considered for replacement once all subtrees have been considered



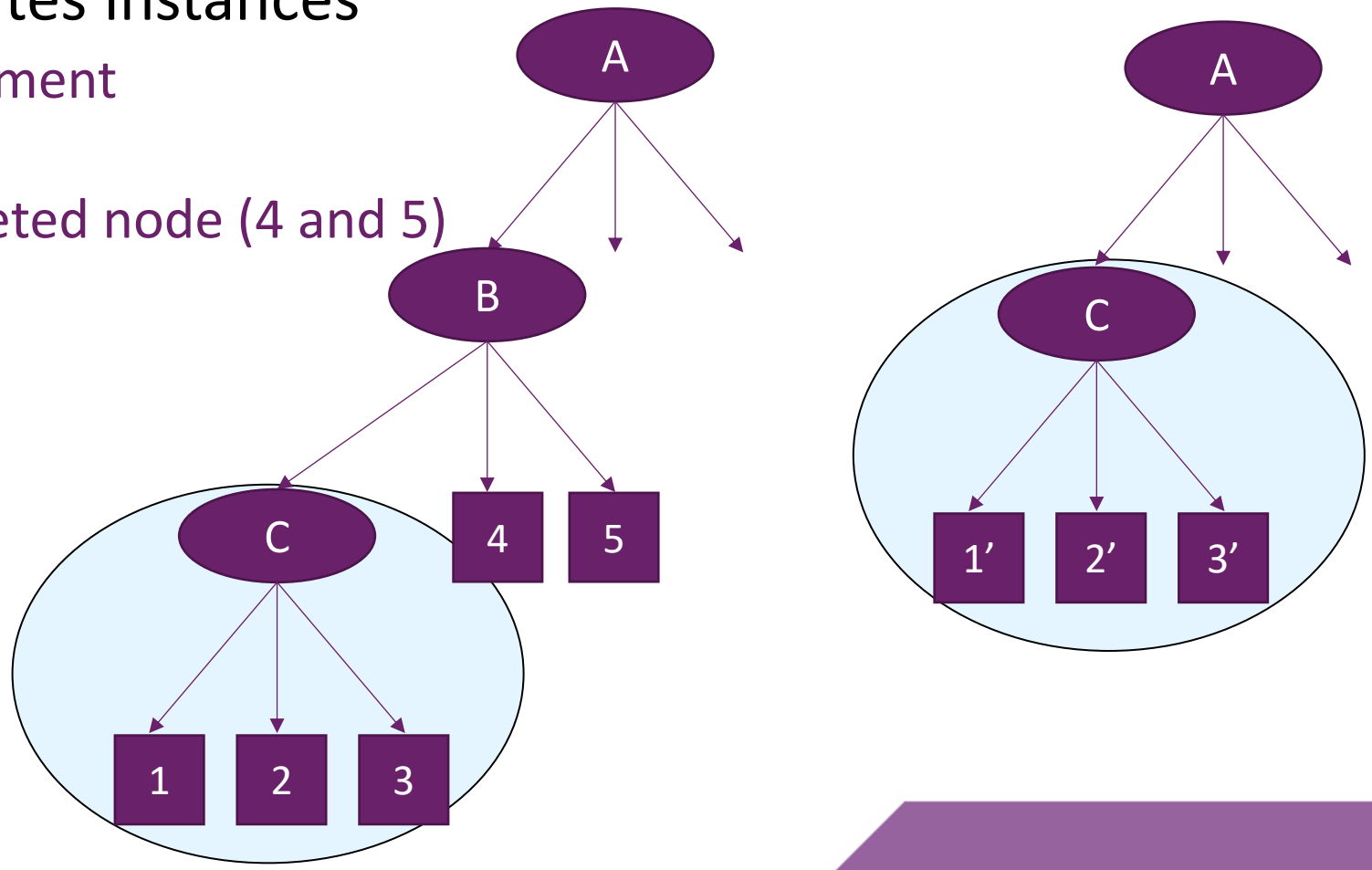
# Post-Pruning: Subtree Raising

- Deletes node and redistributes instances
  - Slower than subtree replacement
    - Worthwhile?
  - Used by C4.5!



# Post-Pruning: Subtree Raising (2)

- Deletes node and redistributes instances
  - Slower than subtree replacement
    - Worthwhile?
  - Instances in branches of deleted node (4 and 5)
    - Re-distributed



# When to Post-Prune?

- Pruning happens if estimated error does not increase
- Error on training data is not useful
  - results in almost no pruning
- Pruning parameter is Confidence Factor
  - Default value is 0.25 (25%).
  - Lower values incur heavier pruning.

# Contents (5)

- Lowering priority of high-branching nodes – gain ratio
- Coping with real data
  - numeric values
  - missing values
- Pruning – removing tree branches
  - noisy data
- Winnowing – feature selection
- Summary

# Winnowing

- Keep only useful attributes (features)
- Data split in half.
- For each attribute
  - Remove attribute
  - Build model with half the data (instances)
  - Test model on other half of data (instances)
  - If removal of attribute does not increase error rate, mark attribute as “useless”.
- Build model using only attributes which have not been deemed “useless”.

# Other pre-processing methods

- Standardisation (centre and scale so mean =0 and SD = 1).
- Normalisation.
- Binarisation.
- Imputation – missing values replaced. E.g. by value of most similar instance (example).
- Transformation: e.g.
  - Principal Component Analysis transforms data to a smaller space of (new) uncorrelated variables.
  - Independent Component Analysis obtains variables which are linear combinations of the original ones. These new variables are independent.
- Attribute selection techniques different to winnowing.

# Summary

- Decision trees can handle
  - Noisy data
  - Numeric data
  - Missing values
- Pruning avoids overfitting
  - Pre-pruning
    - does not grow branches if only few errors/instances, e.g. minimum number of instances
  - Post-pruning
    - removes branches when classification error is small, e.g. confidence factor.
- Winnowing selects attributes to be considered in model building.
- There are lots of pre-processing techniques which can be used with decision trees.