

Data Mining – Chapter 8

Kyuseok Shim
Seoul National University

<http://kdd.snu.ac.kr/~shim>

Extended from the slides of the book "Data Mining:
Concepts and Techniques (3rd ed.)" provided by Jiawei Han,
Micheline Kamber, and Jian Pei

Chapter 8. Classification: Basic Concepts

- Classification: Basic Concepts
- Decision Tree Induction
- Bayes Classification Methods
- Rule-Based Classification
- Model Evaluation and Selection
- Techniques to Improve Classification Accuracy:
Ensemble Methods
- Summary



Supervised vs. Unsupervised Learning

- Supervised learning (classification)
 - Supervision: The training data (observations, measurements, etc.) are accompanied by **labels** indicating the class of the observations
 - New data is classified based on the training set
- Unsupervised learning (clustering)
 - The class labels of training data is unknown
 - Given a set of measurements, observations, etc. with the aim of establishing the existence of classes or clusters in the data

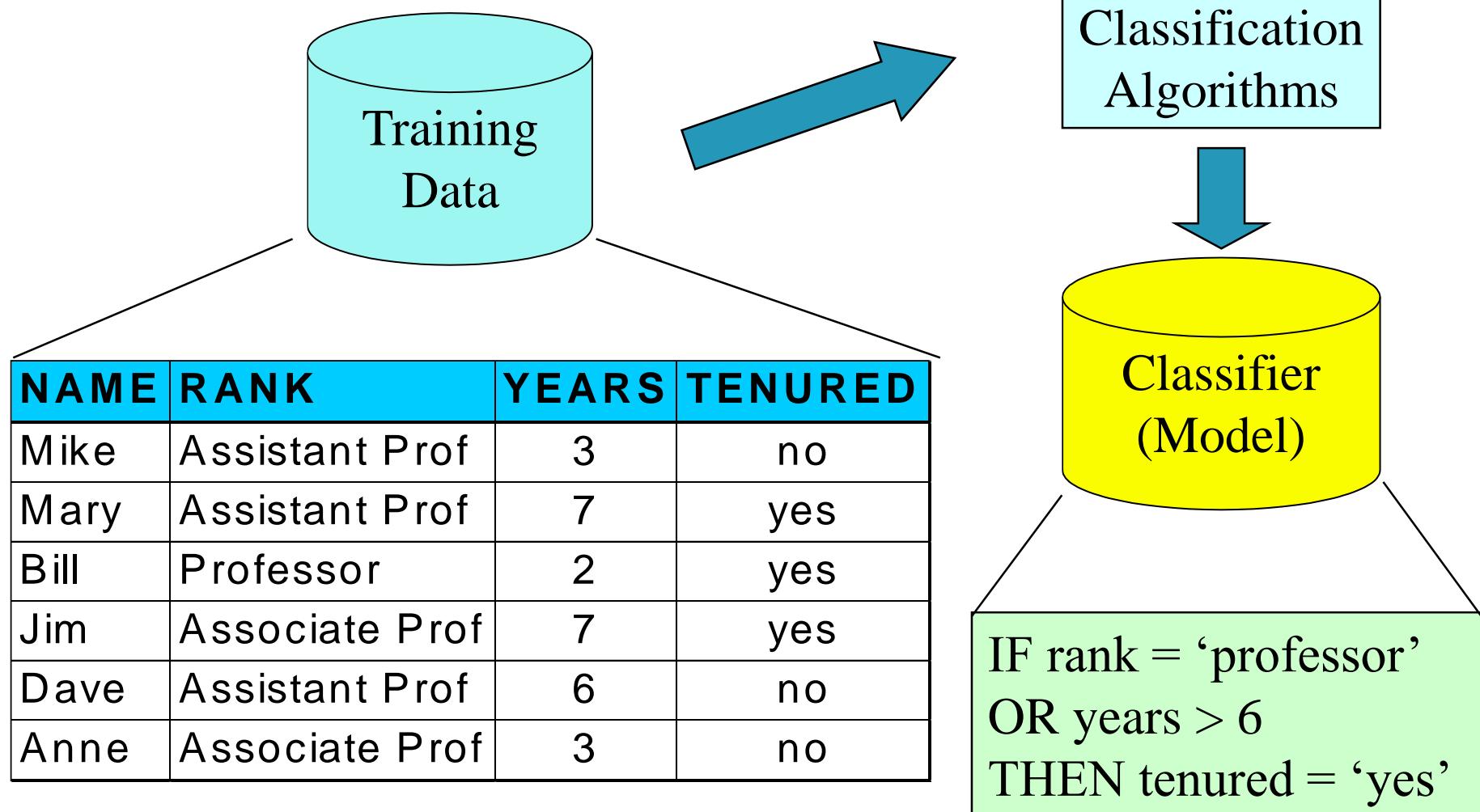
Prediction Problems: Classification vs. Numeric Prediction

- 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
- 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

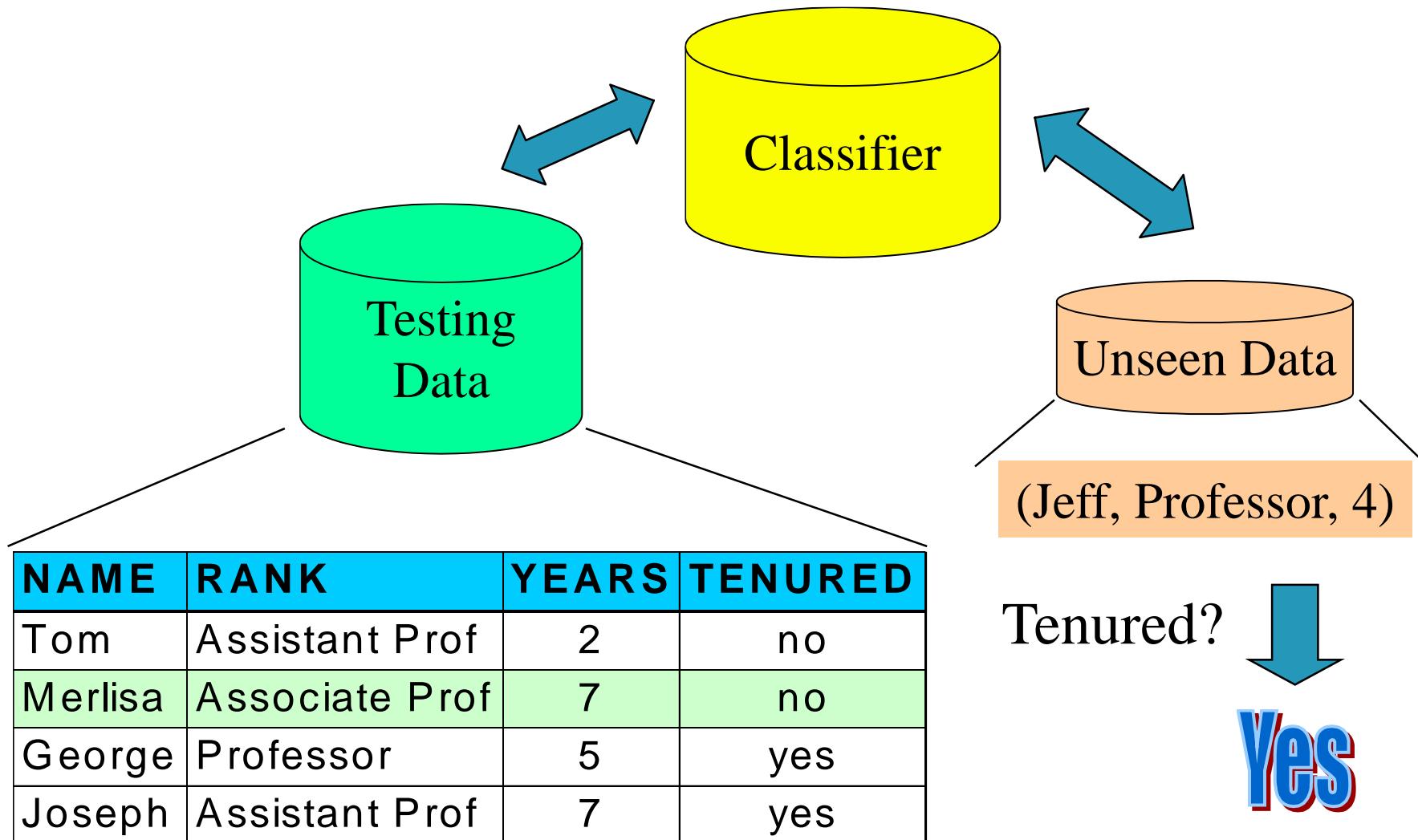
Classification—A Two-Step Process

- Model construction: describing a set of predetermined classes
 - 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 classification rules, decision trees, or mathematical formulae
- 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 (test) set**

Process (1): Model Construction

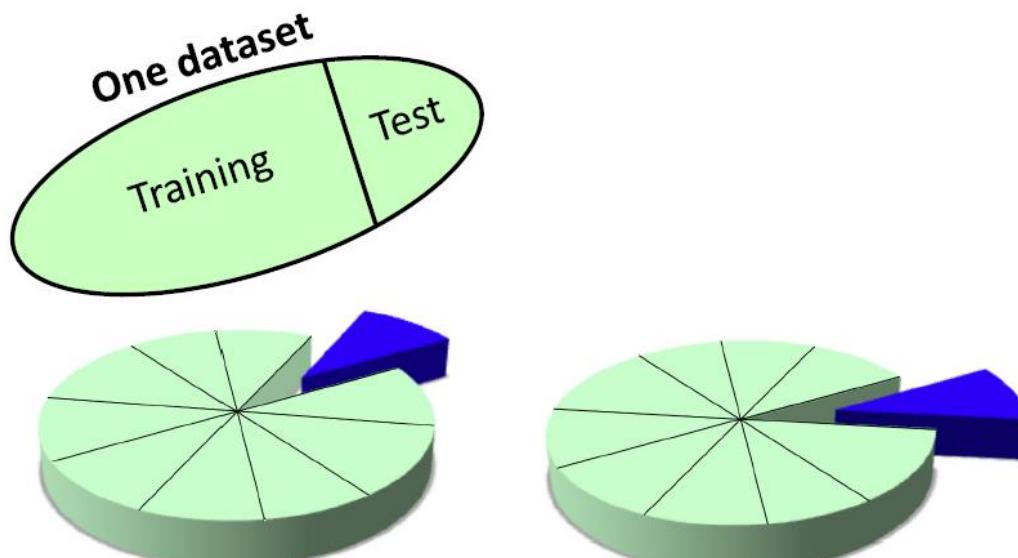


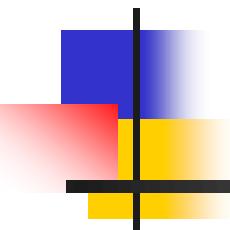
Process (2): Using the Model in Prediction



Cross-validation

- 10-fold cross-validation
 - Divide dataset into 10 parts (folds)
 - Hold out each part in turn
 - Average the results
 - Each data point used once for testing, 9 times for training



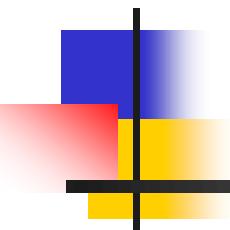


K-nearest Neighbor Classifier

K-nearest Neighbor Classifier

- Nearest-neighbor
- k-nearest-neighbors
 - Choose majority class among several neighbors (k of them)
- In Weka,
 - **lazy>IBk** (instance-based k-nearest neighbours)

Weka – K-nearest Neighbor Classifier



glass.arff

- Classify the type of glass
 - Motivated by criminological investigation
 - At the scene of the crime, the glass left can be used as evidence...if it is correctly identified!

Features:

RI: refractive index

Na: Sodium

Mg: Magnesium

...

Types of glass:

building_windows_float_processed

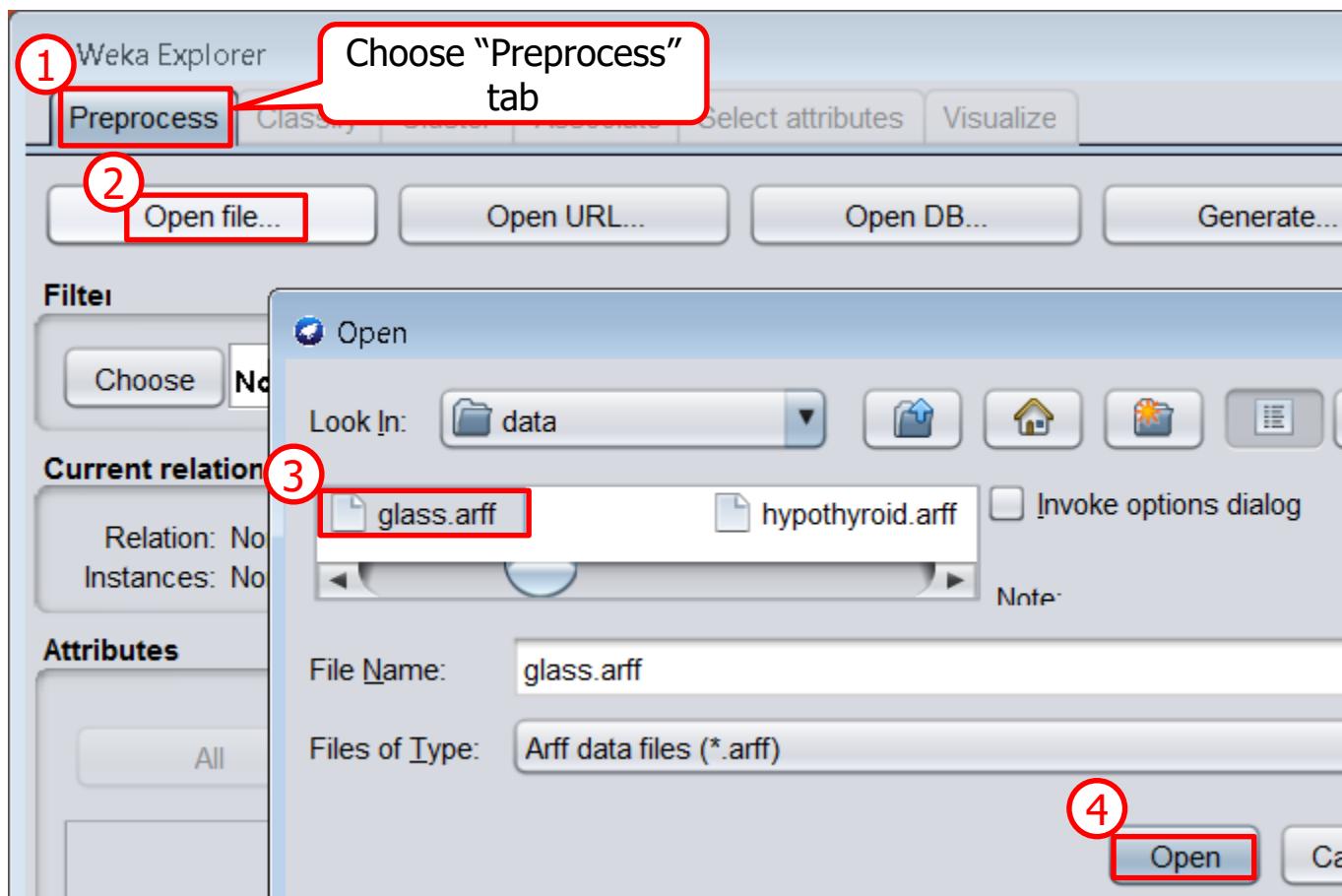
building_windows_non_float_processed

vehicle_windows_float_processed

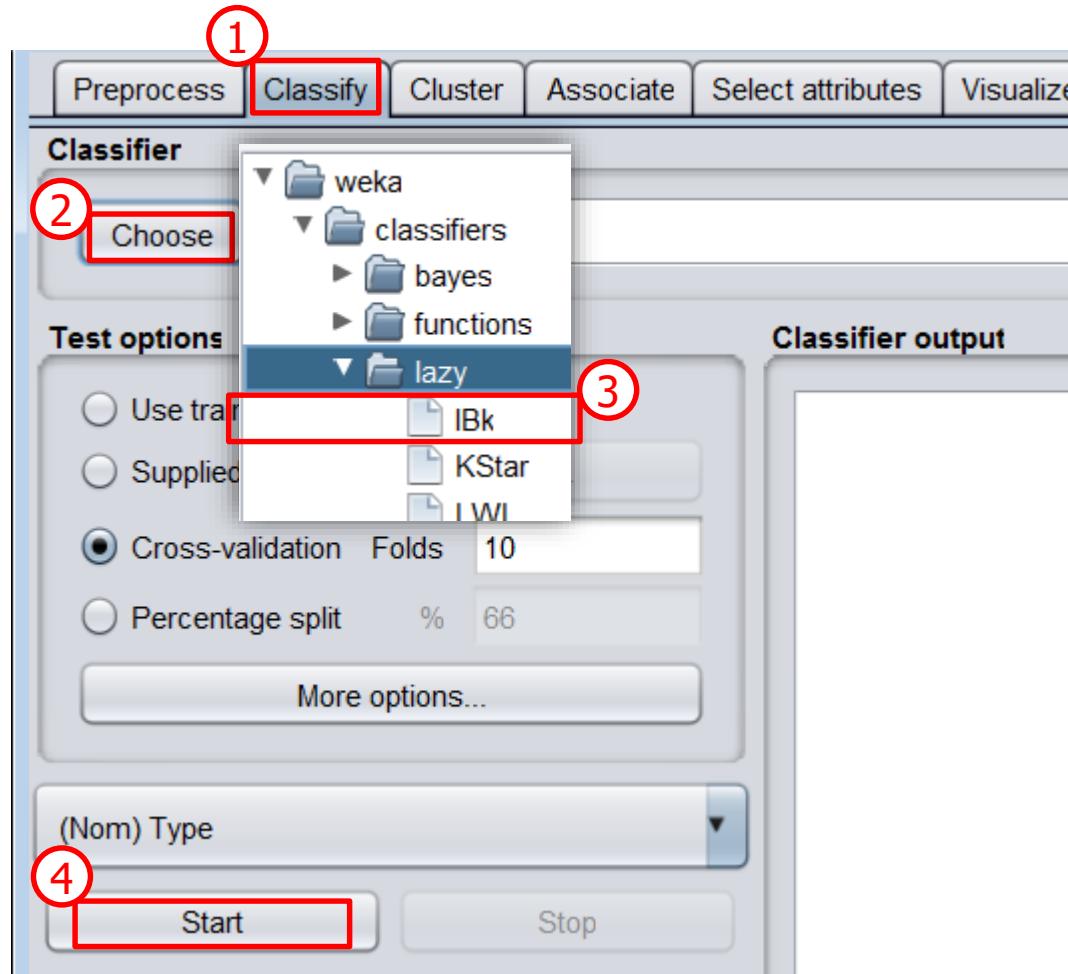
...

Open the Dataset

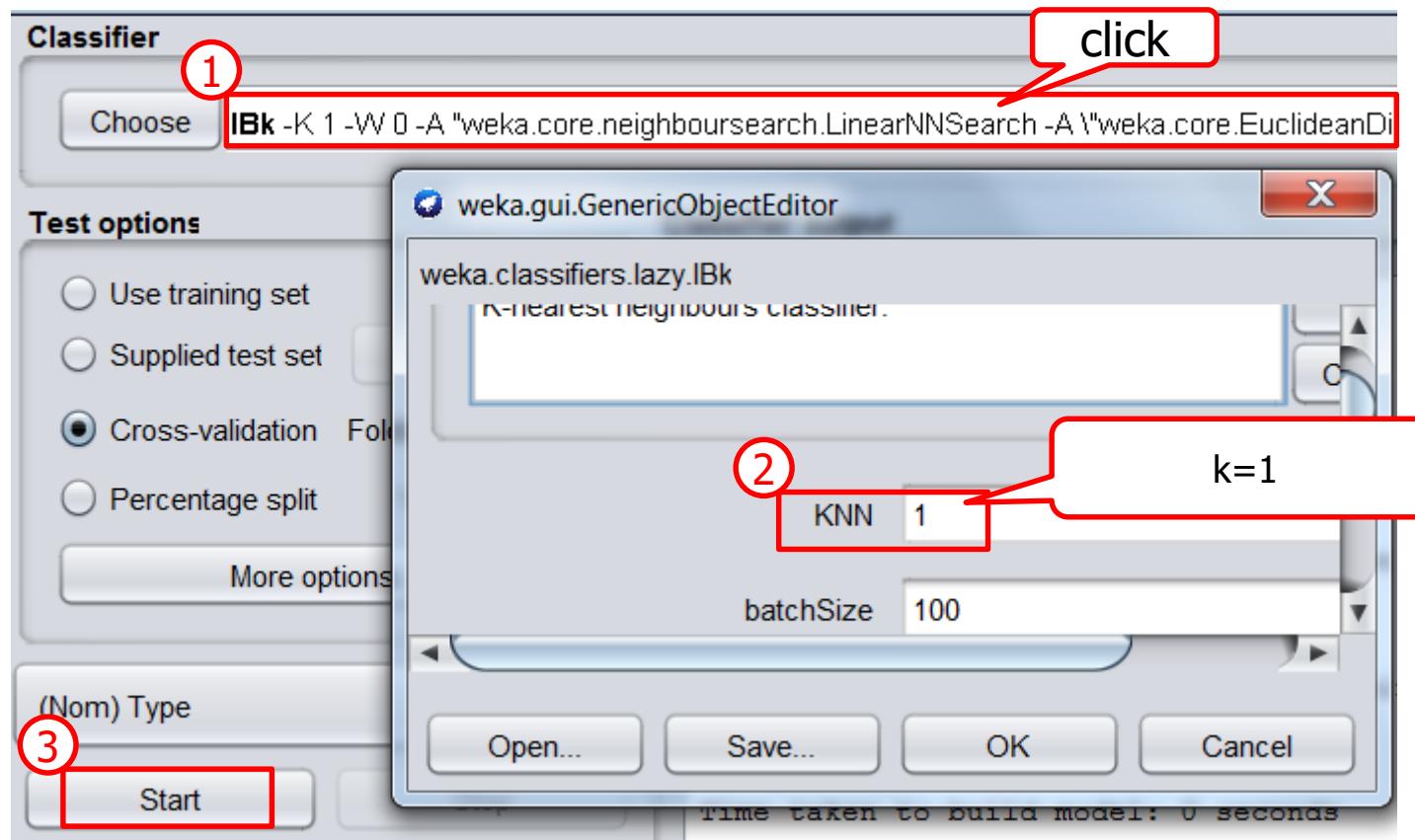
- C:\Program Files\Weka-3-8\data\glass.arff



Select the Classifier



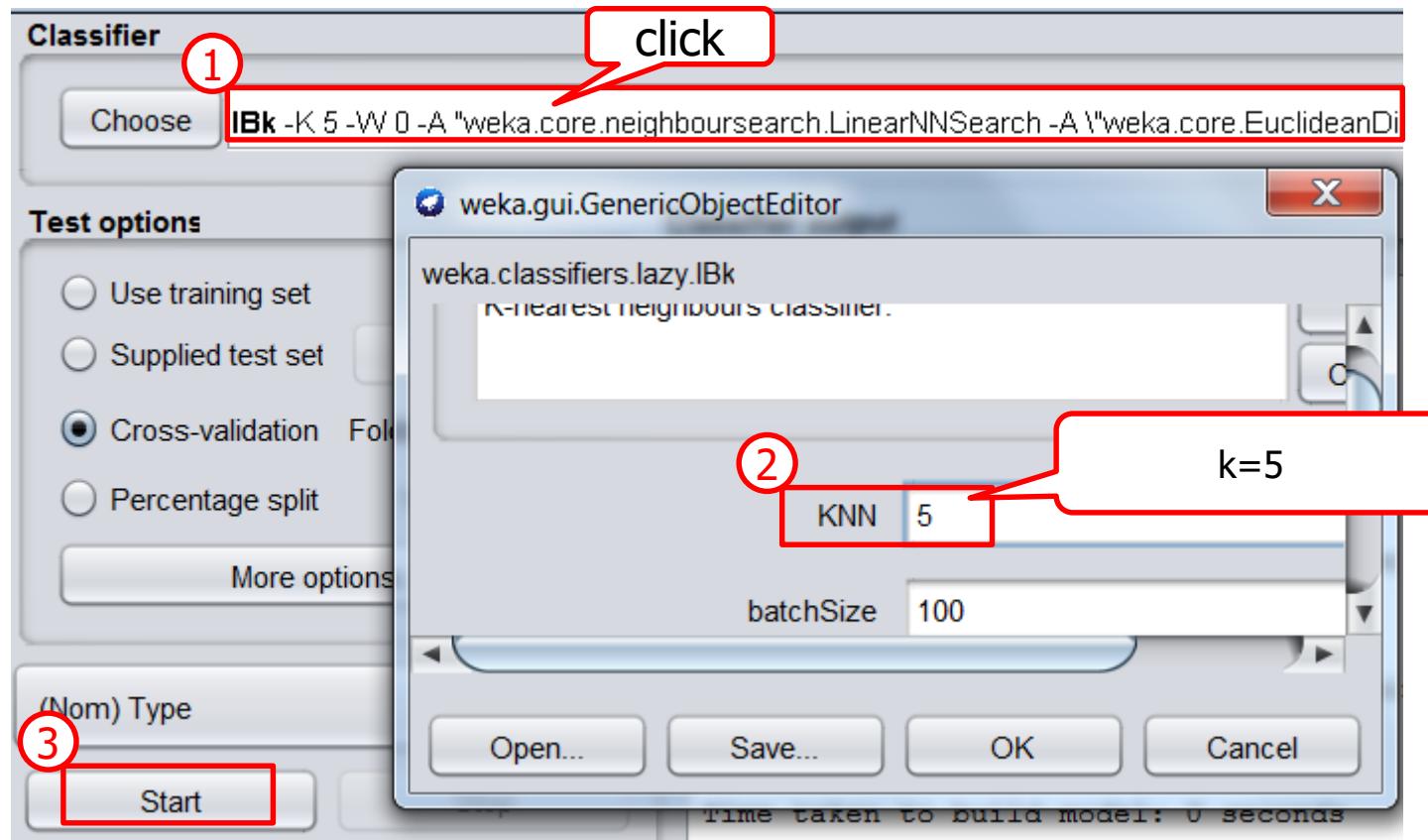
k=1



k=1

Classifier output		
==== Stratified cross-validation ====		
==== Summary ====		
Correctly Classified Instances	151	70.5607 %
Incorrectly Classified Instances	63	29.4393 %
Kappa statistic	0.6005	
Mean absolute error	0.0897	
Root mean squared error	0.2852	
Relative absolute error	42.3747 %	
Root relative squared error	87.8627 %	
Total Number of Instances	214	

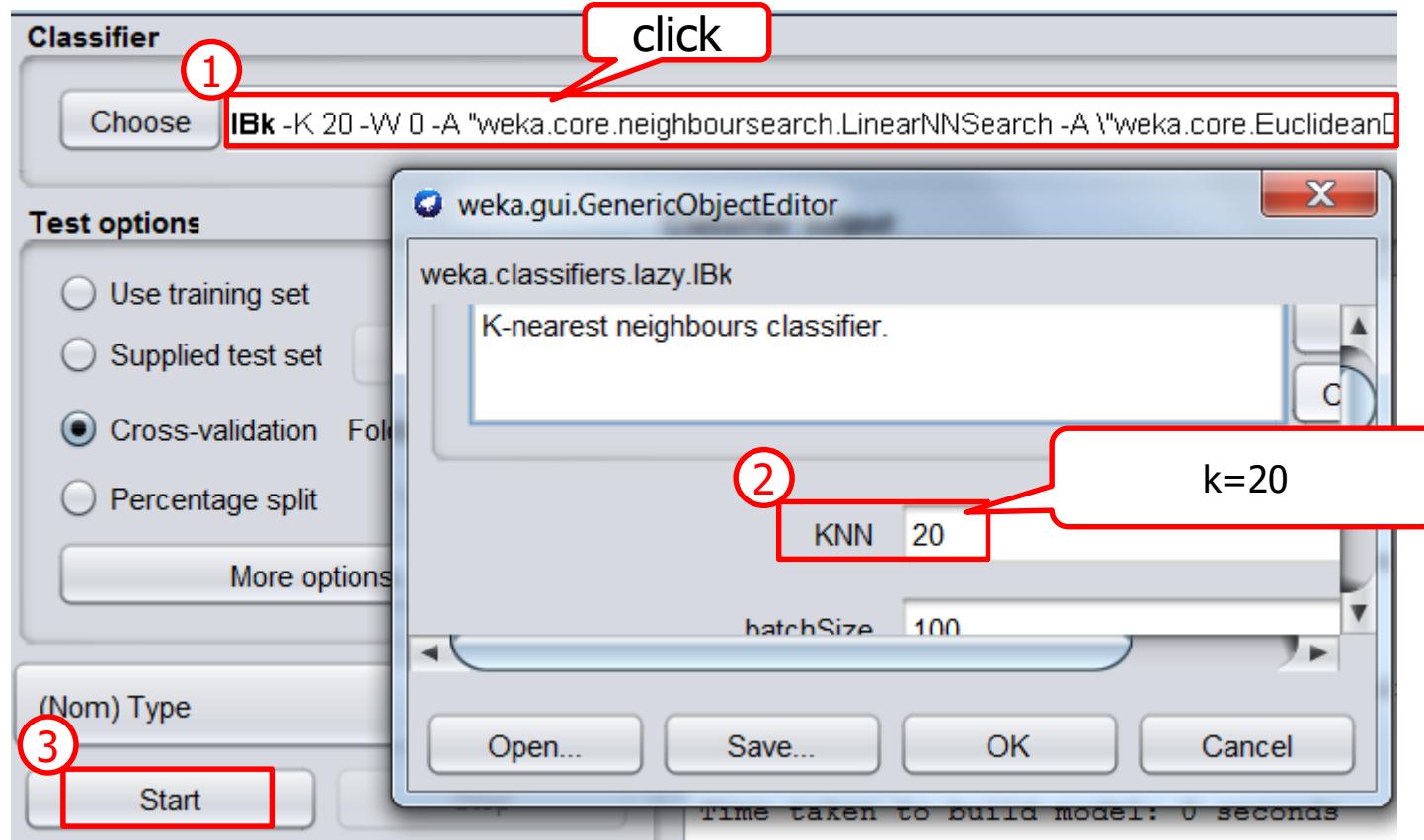
k=5



k=5

Classifier output			
==== Stratified cross-validation ====			
==== Summary ====			
Correctly Classified Instances	145	67.757	%
Incorrectly Classified Instances	69	32.243	%
Kappa statistic	0.5469		
Mean absolute error	0.1085		
Root mean squared error	0.2563		
Relative absolute error	51.243	%	
Root relative squared error	78.9576	%	
Total Number of Instances	214		

k=20



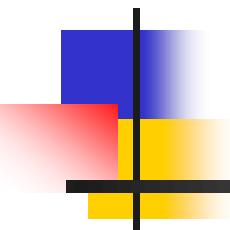
k=20

Classifier output		
<pre>==== Stratified cross-validation ==== ==== Summary ==== </pre>		
<pre>Correctly Classified Instances 140 65.4206 % Incorrectly Classified Instances 74 34.5794 % </pre>		
Correctly Classified Instances	140	65.4206 %
Incorrectly Classified Instances	74	34.5794 %
Kappa statistic	0.5049	
Mean absolute error	0.1336	
Root mean squared error	0.2588	
Relative absolute error	63.0737 %	
Root relative squared error	79.7334 %	
Total Number of Instances	214	

K-nearest Neighbor Classifier

- Investigate effect of changing k
- Glass dataset
 - lazy > IBk, k = 1, 5, 20
- 10-fold cross-validation
 - k = 1 k = 5 k = 20
 - 70.6% 67.8% 65.4%

Python – K-nearest Neighbor Classifier



Download the Dataset

- Download glass.csv from <http://kdd.snu.ac.kr/python/>
- Save the csv file in the same directory as the source file (.ipynb)

glass.csv

- Classify the type of glass
 - Motivated by criminological investigation
 - At the scene of the crime, the glass left can be used as evidence...if it is correctly identified!

Features:

RI: refractive index

Na: Sodium

Mg: Magnesium

...

Types of glass:

building_windows_float_processed

building_windows_non_float_processed

vehicle_windows_float_processed

...

Import Libraries

```
import pandas as pd
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.neighbors import KNeighborsClassifier
```

- `pandas`: a library for data analysis
- `cross_val_score`: a function for K-fold cross validation
- `KNeighborsClassifier`: a class for K-nearest neighbor classifier
- `KFold` : K-fold cross validation model

Open the Dataset

```
In [90]: df = pd.read_csv('glass.csv')  
df
```

```
Out [90]:
```

	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type
0	1.51793	12.79	3.50	1.12	73.03	0.64	8.77	0.00	0.00	'build wind float'
1	1.51643	12.16	3.52	1.35	72.89	0.57	8.53	0.00	0.00	'vehic wind float'
2	1.51793	13.21	3.48	1.41	72.64	0.59	8.43	0.00	0.00	'build wind float'
3	1.51299	14.40	1.74	1.54	74.55	0.00	7.59	0.00	0.00	tableware
4	1.53393	12.30	0.00	1.00	70.16	0.12	16.19	0.00	0.24	'build wind non-float'
5	1.51655	12.75	2.85	1.44	73.27	0.57	8.79	0.11	0.22	'build wind non-float'
6	1.51779	13.64	3.65	0.65	73.00	0.06	8.93	0.00	0.00	'vehic wind float'
7	1.51837	13.14	2.84	1.28	72.85	0.55	9.07	0.00	0.00	'build wind float'
8	1.51545	14.14	0.00	2.68	73.39	0.08	9.07	0.61	0.05	headlamps
9	1.51789	13.19	3.90	1.30	72.33	0.55	8.44	0.00	0.28	'build wind non-float'
10	1.51625	13.36	3.58	1.49	72.72	0.45	8.21	0.00	0.00	'build wind non-float'

Data Preprocessing

```
X = df.values[:, :-1]
y = df.values[:, -1]
```

```
print(X)
```

```
[[1.51793 12.79 3.5 ... 8.77 0.0 0.0]
 [1.51643 12.16 3.52 ... 8.53 0.0 0.0]
 [1.51793 13.21 3.48 ... 8.43 0.0 0.0]
 ...
 [1.51613 13.92 3.52 ... 7.94 0.0 0.14]
 [1.51689 12.67 2.88 ... 8.54 0.0 0.0]
 [1.51852 14.09 2.19 ... 9.32 0.0 0.0]]
```

```
print(y)
```

```
["'build wind float'" "'vehic wind float'" "'build wind float'"
 'tableware' "'build wind non-float'" "'build wind non-float'"
 "'vehic wind float'" "'build wind float'" 'headlamps'
 "'build wind non-float'" "'build wind non-float'"
 "'build wind non-float'" "'build wind float'" "'vehic wind float'"
 "'vehic wind float'" "'build wind non-float'" 'headlamps'
 "'build wind non-float'" 'containers' "'build wind non-float'"
 "'build wind float'" "'build wind non-float'" "'build wind non-float'"
 "'build wind float'" 'containers' "'build wind non-float'"
 "'build wind non-float'" 'headlamps' "'build wind non-float'"
 "'vehic wind float'" "'build wind non-float'" "'vehic wind float'"
 'tableware' "'build wind non-float'" "'build wind float'"
 "'build wind float'" "'build wind float'" "'build wind non-float'"
 "'build wind non-float'" "'build wind non-float'" "'build wind float'"
```

Changing Model Parameters

```
clf = KNeighborsClassifier(n_neighbors=10,  
                           weights='uniform',  
                           metric='euclidean')
```

n_neighbors : number of neighbors (k)

weights : weight function used in prediction.

- 'uniform' : all neighbors have same weight

- 'distance' : weights are given according to the distance

- * Note : user defined function can also be called

metric : the distance metric to use

K-fold Cross-validation

```
cv = KFold(  
    n_splits=10,  
    shuffle=True,  
    random_state=0)  
cv_results = cross_val_score(clf, X, y, cv=cv)  
  
print(cv_results.mean())
```

The number of folds

Whether to shuffle the data
before splitting into batches

The random seed

0.6155844155844156

K-fold Cross-validation

```
cv = KFold(  
    n_splits=10,  
    shuffle=True,  
    random_state=0)  
cv_results = cross_val_score(clf, X, y, cv=cv)  
  
print(cv_results.mean())
```

Features

The classifier

Labels

The diagram illustrates the components of a K-fold cross-validation process. It features three red-bordered boxes with labels: 'Features' at the top right, 'The classifier' in the middle right, and 'Labels' at the bottom right. Three red arrows point from these labels to specific elements in the Python code: 'Features' points to 'X', 'The classifier' points to 'clf', and 'Labels' points to 'y'.

0.6155844155844156

K-fold Cross-validation

```
cv = KFold(  
    n_splits=10,  
    shuffle=True,  
    random_state=0)  
cv_results = cross_val_score(clf, X, y, cv=cv)  
  
print(cv_results.mean())  
0.6155844155844156
```

Scores of 10-fold cross-validations

Print the average of scores

Prediction with KNN

```
clf.fit(X,y)
pred_y = clf.predict(
    [[1.5, 13, 1.5, 1.5, 70, 0.5, 8.9, 0.1, 0.2]])
print(pred_y)           Test data should be a 2-D array
```

```
[''build wind float'']
```

The prediction result is printed

Comparison with Varying k

```
clf = KNeighborsClassifier(n_neighbors=20, weights='uniform')
clf2 = KNeighborsClassifier(n_neighbors=5, weights='uniform')
clf3 = KNeighborsClassifier(n_neighbors=1, weights='uniform')

results = cross_val_score(clf, X, y, cv=cv)
results2 = cross_val_score(clf2, X, y, cv=cv)
results3 = cross_val_score(clf3, X, y, cv=cv)

print("20 neighbors: {}".format(
    results.mean()))
print("5 neighbors: {}".format(
    results2.mean()))
print("1 neighbors: {}".format(
    results3.mean()))
```

20 neighbors: 0.6155844155844156
5 neighbors: 0.648051948051948
1 neighbors: 0.7370129870129871

Varying the number
of neighbors

Note: It is not always a good idea to
increase k

Practice

- Iris 데이터를 가지고 glass 데이터를 이용하여 k-nearest neighbor classifier를 실습한 것 처럼 하시오.

K-nearest Neighbor Classifier

- Often very accurate ... but slow:
 - Scan entire training data to make each prediction?
 - Sophisticated data structures can make this faster
- Assumes all attributes equally important
 - Remedy: attribute selection or weights
- Remedies against noisy instances:
 - Majority vote over the k nearest neighbors
 - Weight instances according to prediction accuracy
 - Identify reliable “prototypes” for each class

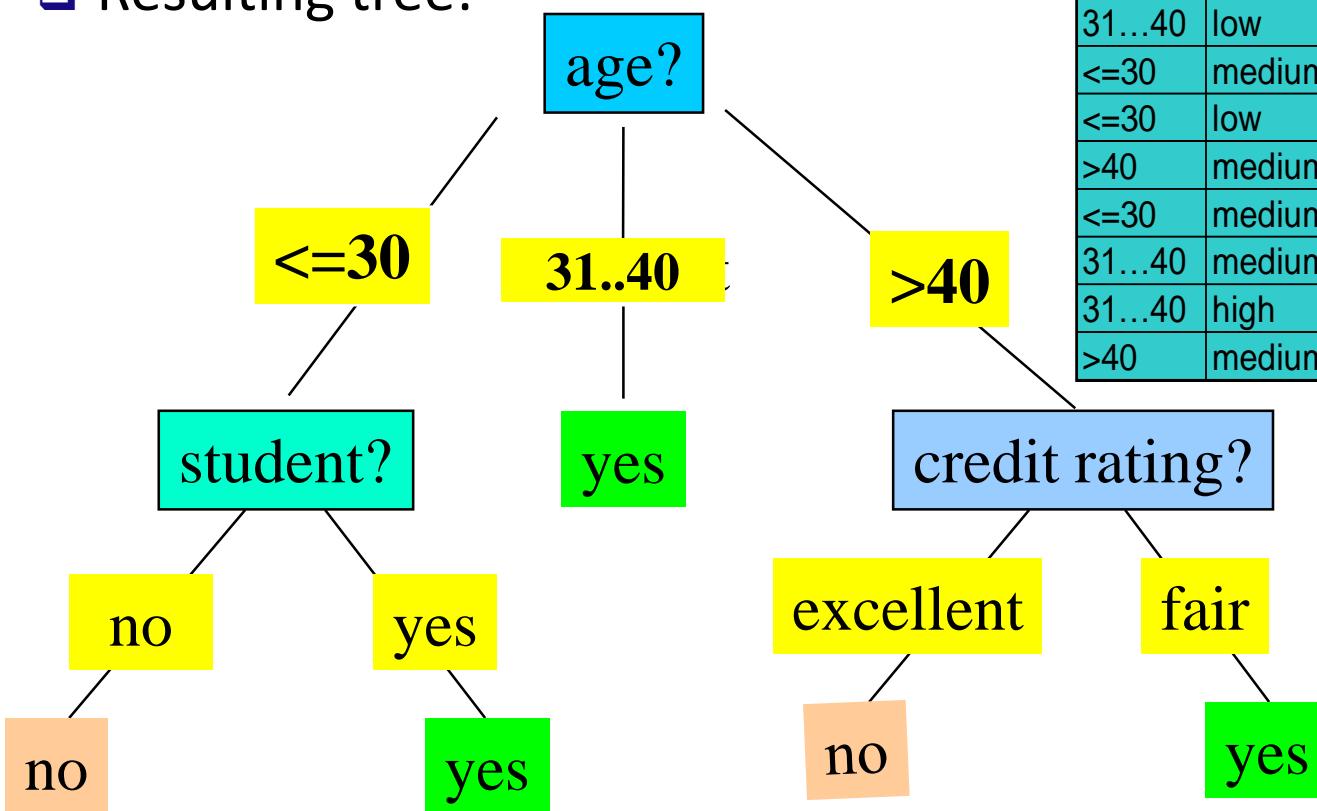
Chapter 8. Classification: Basic Concepts

- Classification: Basic Concepts
- Decision Tree Induction
- Bayes Classification Methods
- Rule-Based Classification
- Model Evaluation and Selection
- Techniques to Improve Classification Accuracy:
Ensemble Methods
- Summary



Decision Tree Induction: An Example

- Training data set: Buys_computer
- The data set follows an example of Quinlan's ID3 (Playing Tennis)
- Resulting tree:



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

Decision Tree Algorithm

- A decision tree is created in two phases:
 - Building Phase
 - Recursively split nodes using best splitting attribute for node until all the examples in each node belong to one class
 - Pruning Phase
 - Prune leaf nodes recursively to prevent over-fitting
 - Smaller imperfect decision tree generally achieves better accuracy

Building Phase

- General tree-growth algorithm (binary tree)

Partition(Data S)

If (all points in S are of the same class) then
return;

for each attribute A do

 evaluate splits on attribute A;

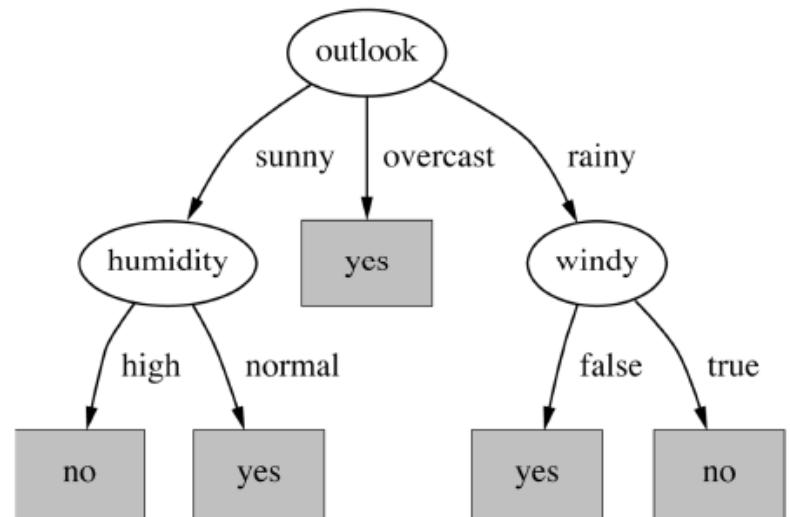
Use best split to partition S into S₁ and S₂;

Partition(S₁);

Partition(S₂);

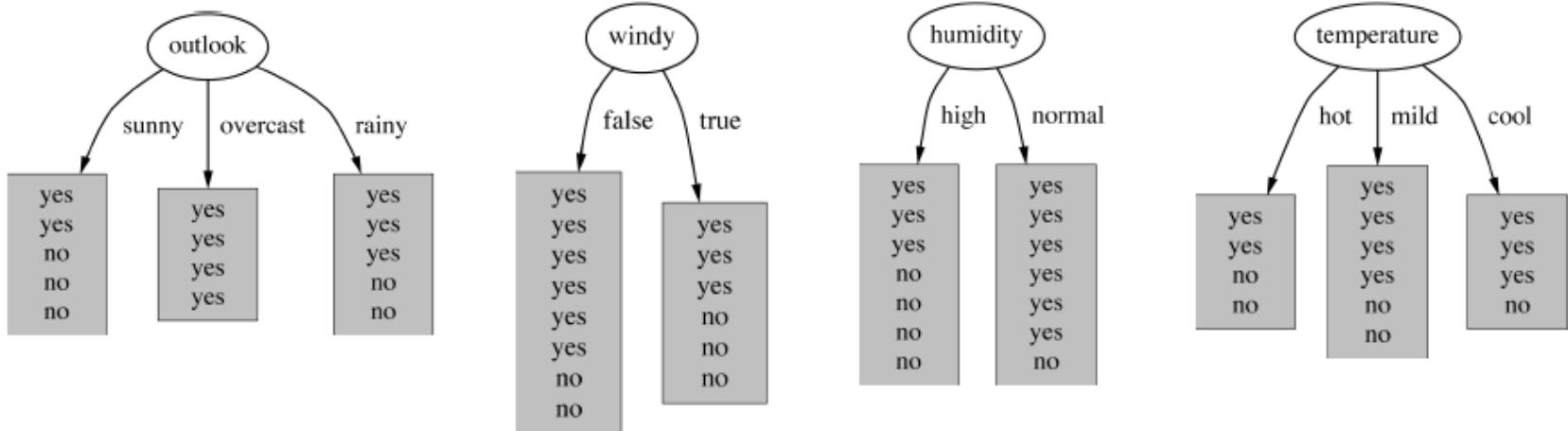
Decision trees

- Top-down: recursive divide-and-conquer
 - **Select** attribute for root node
 - Create branch for each possible attribute value
 - **Split** instances into subsets
 - One for each branch extending from the node
 - **Repeat** recursively for each branch
 - using only instances that reach the branch
 - **Stop**
 - if all instances have the same class



Decision Trees

Which attribute to select?



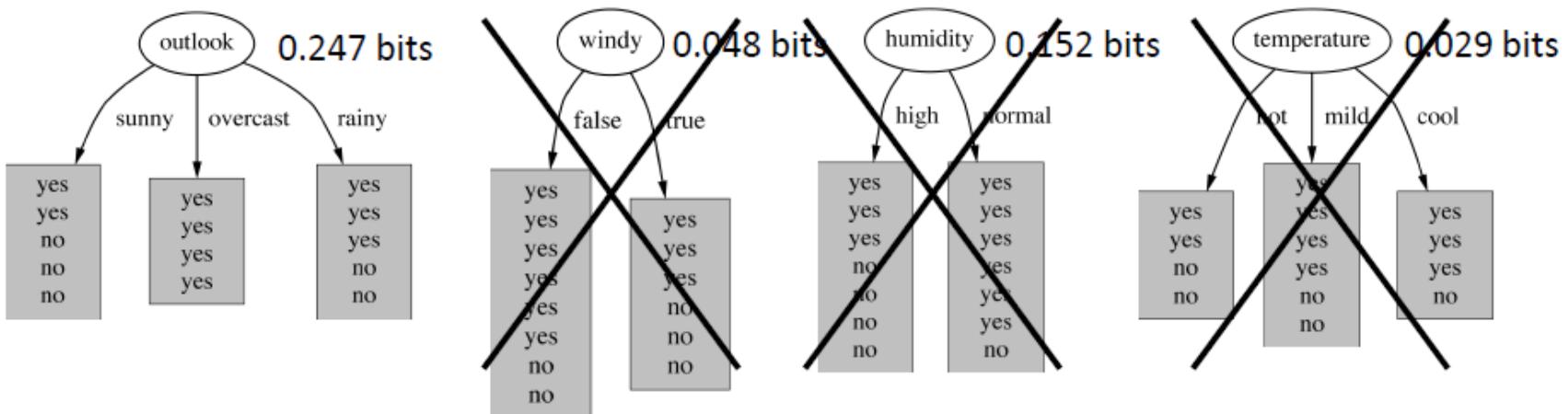
Ian H. Witten's slide

Decision Trees

- Which is the best attribute?
 - Aim: to get the smallest tree
 - Heuristic
 - choose the attribute that produces the “purest” nodes
 - i.e., the greatest information gain
 - Information theory: measure information in bits
 - $\text{entropy}(p_1, p_2, \dots, p_n) = -p_1 \log p_1 - p_2 \log p_2 - \dots - p_n \log p_n$
- Information gain
 - Amount of information gained by knowing the value of the attribute
 - $(\text{Entropy of distribution before the split}) - (\text{entropy of distribution after it})$
 - Claude Shannon, American mathematician and scientist 1916–2001

Decision Trees

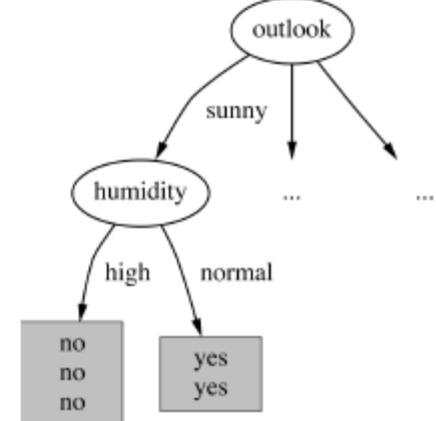
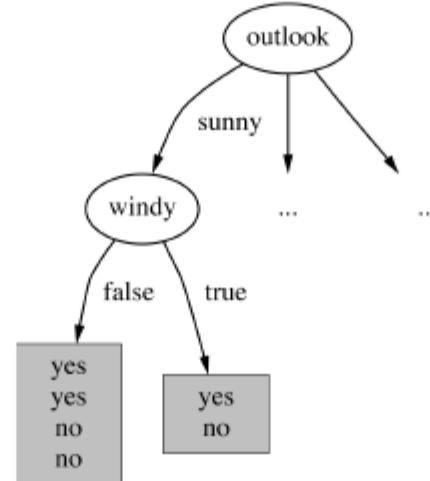
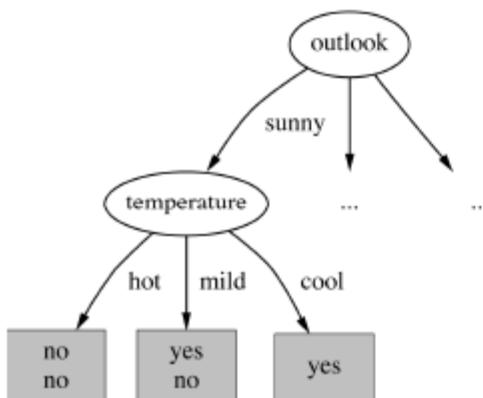
Which attribute to select?



Ian H. Witten's slide

Decision Trees

Continue to split ...



$$\text{gain}(\text{temperature}) = 0.571 \text{ bits}$$

$$\text{gain}(\text{windy}) = 0.020 \text{ bits}$$

$$\text{gain}(\text{humidity}) = 0.971 \text{ bits}$$

Splitting Numeric Attributes

- Split on temperature attribute:

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

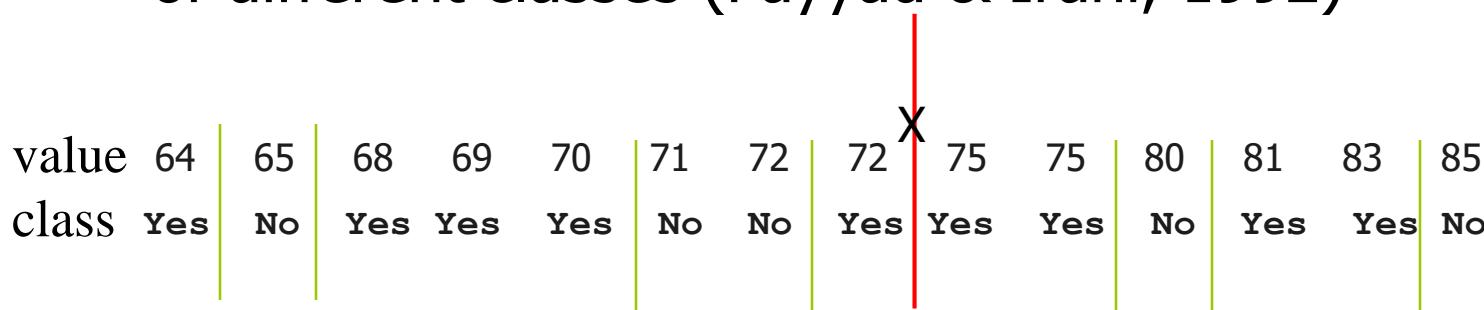
- E.g. $\text{temperature} < 71.5$: yes/4, no/2
 $\text{temperature} \geq 71.5$: yes/5, no/3
- $\text{Info}([4,2],[5,3])$
 $= 6/14 \text{ info}([4,2]) + 8/14 \text{ info}([5,3])$
 $= 0.939 \text{ bits}$
- Place split points halfway between values
- Can evaluate all split points in one pass!

Avoid repeated sorting!

- Sort instances by the values of the numeric attribute
 - Time complexity for sorting: $O(n \log n)$
- Q. Does this have to be repeated at each node of the tree?
- A: No! Sort order for children can be derived from sort order for parent
 - Time complexity of derivation: $O(n)$
 - Drawback: need to create and store an array of sorted indices for each numeric attribute

More speeding up

- Entropy only needs to be evaluated between points of different classes (Fayyad & Irani, 1992)



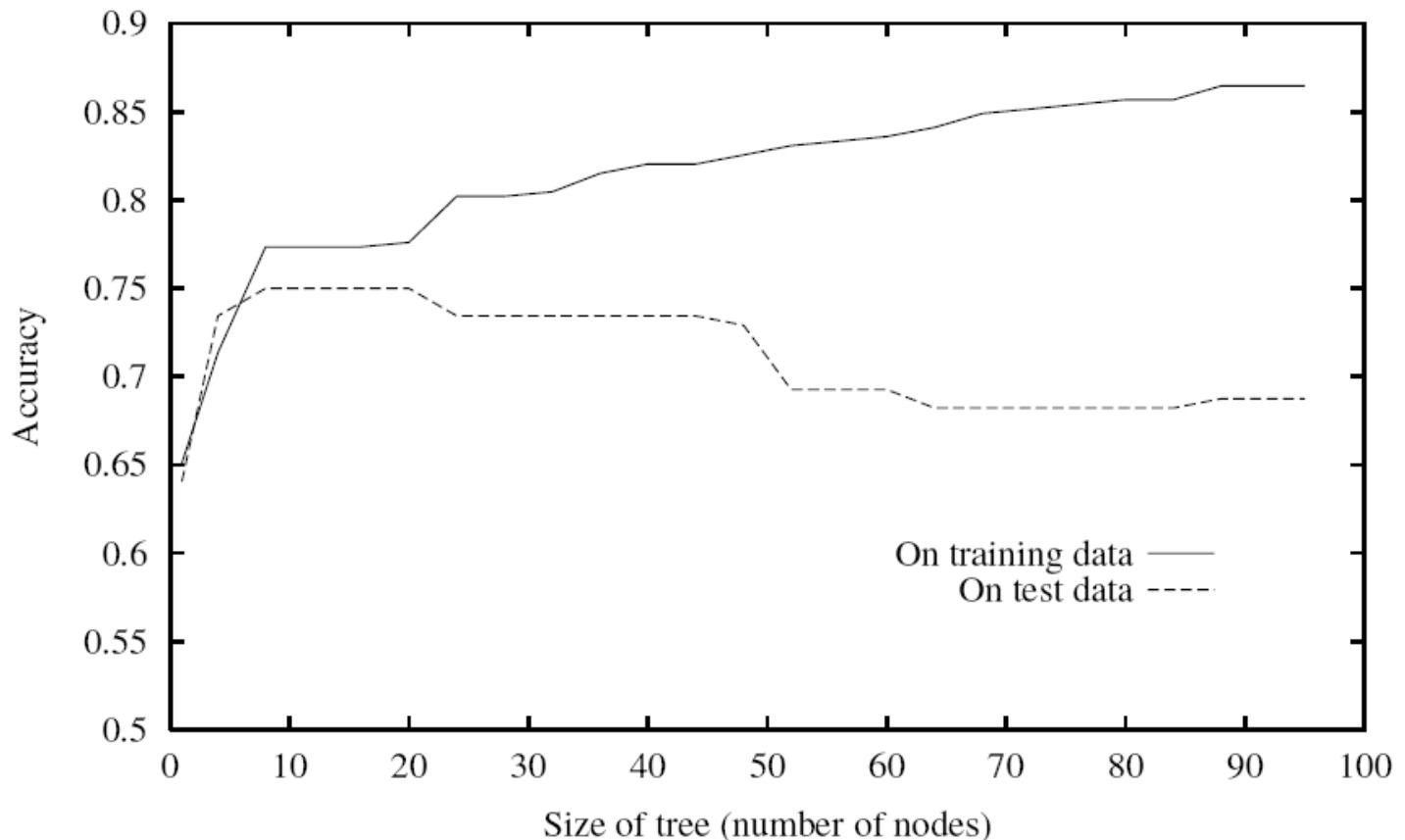
Potential optimal breakpoints

Breakpoints between values of the same class cannot be optimal

Binary vs. multi-way splits

- Splitting (multi-way) on a nominal attribute exhausts all information in that attribute
 - Nominal attribute is tested (at most) once on any path in the tree
- Not so for binary splits on numeric attributes!
 - Numeric attribute may be tested several times along a path in the tree
- Disadvantage: tree is hard to read
- Remedy:
 - Pre-discretize numeric attributes, *or*
 - Use multi-way splits instead of binary ones

Overfitting in Decision Tree Learning



Avoiding Overfitting

- How can we avoid overfitting?
 - Method 1: Stop growing when data split not statistically significant
 - Method 2: Grow full tree, then post-prune
- How to select the “best” tree:
 - Measure performance over training data
 - Measure performance over separate validation data set

Pruning

- Goal: Prevent overfitting to noise in the data
- Two strategies for “pruning” the decision tree:
 - ◆ Postpruning - take a fully-grown decision tree and discard unreliable parts
 - ◆ Prepruning - stop growing a branch when information becomes unreliable
- Postpruning preferred in practice—prepruning can “stop too early”

Prepruning

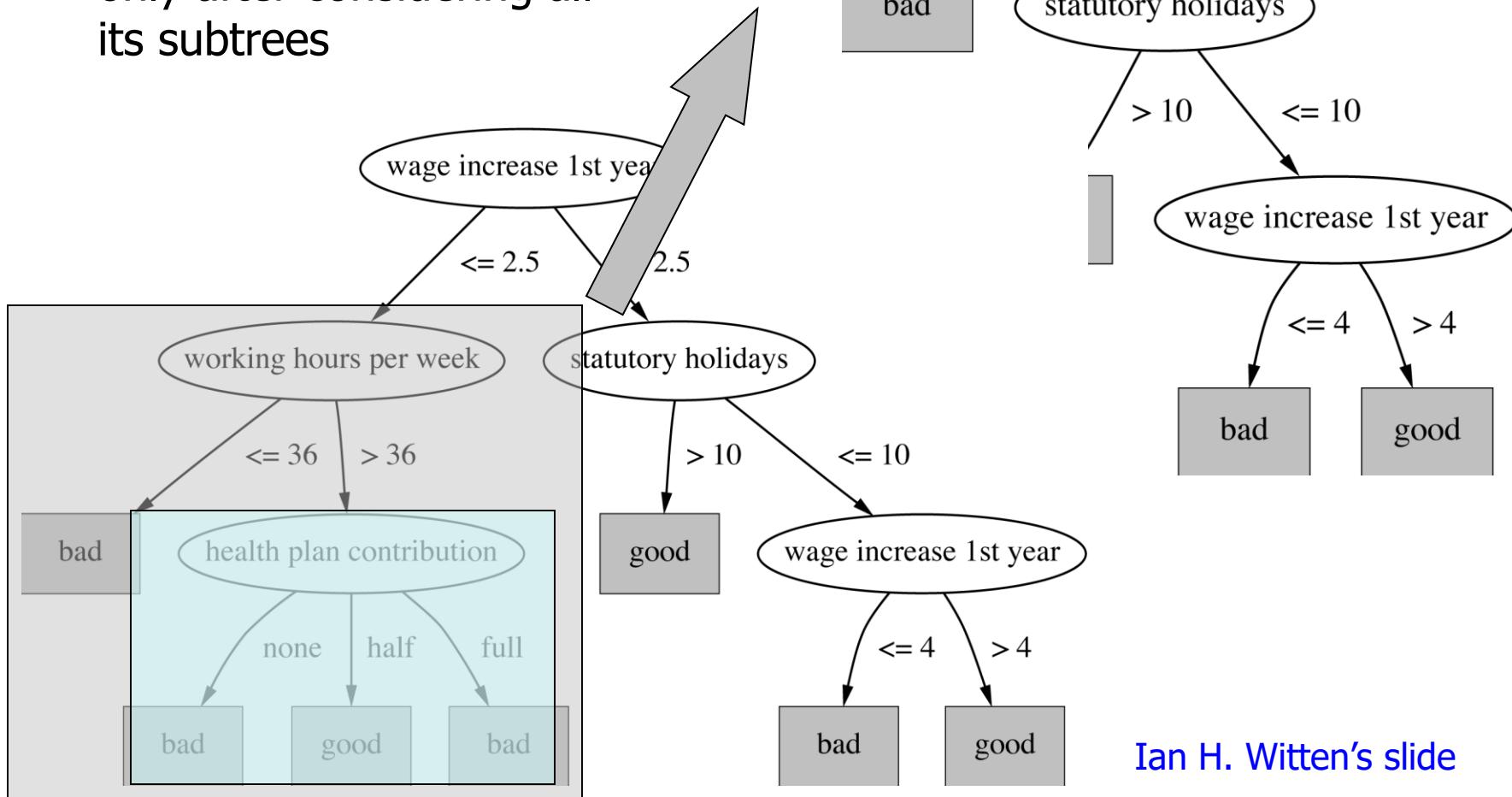
- Based on statistical significance test
 - Stop growing the tree when there is no *statistically significant* association between any attribute and the class at a particular node
- Most popular test: chi-squared test
- ID3 used chi-squared test in addition to information gain
 - Only statistically significant attributes were allowed to be selected by information gain procedure
- Pre-pruning may stop the growth process prematurely: early stopping
- Pre-pruning faster than post-pruning

Post-pruning

- First, build full tree
- Then, prune it
 - Fully-grown tree shows all attribute interactions
- Two pruning operations:
 - Subtree replacement
 - Subtree raising
- Possible strategies:
 - Error estimation
 - Significance testing
 - MDL principle

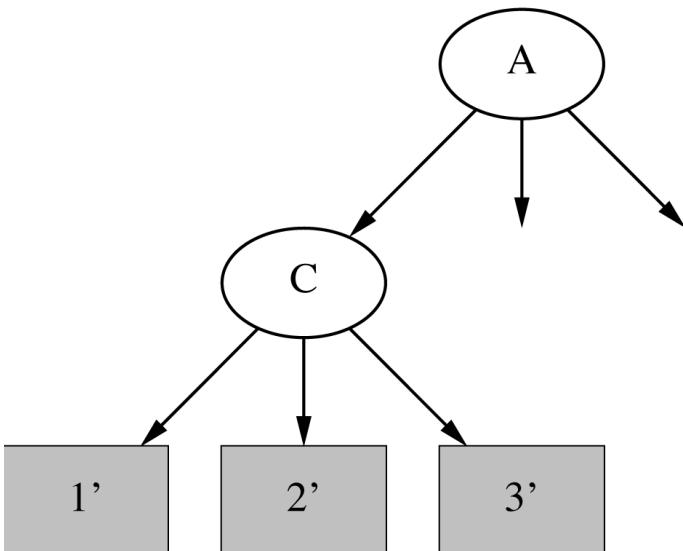
Subtree Replacement

- Bottom-up
- Consider replacing a tree only after considering all its subtrees

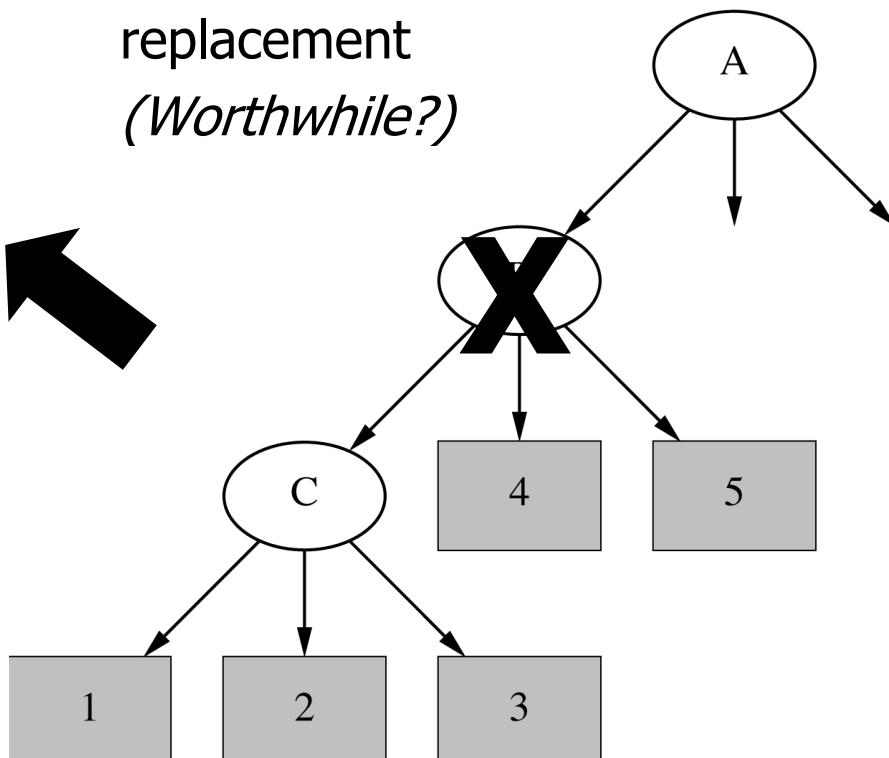


Ian H. Witten's slide

Subtree Raising



- Delete node
- Redistribute instances
- Slower than subtree replacement
(Worthwhile?)

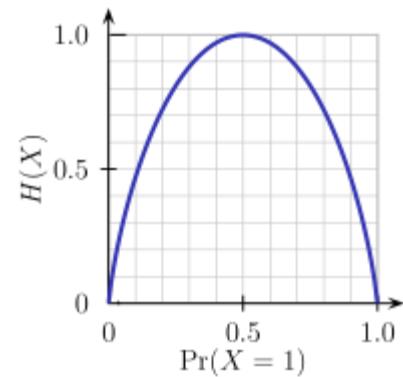


Algorithm for Decision Tree Induction

- 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 (e.g., **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 $\{y_1, \dots, y_m\}$,
 - $H(Y) = -\sum_{i=1}^m p_i \log(p_i)$, where $p_i = P(Y = y_i)$
 - Interpretation:
 - Higher entropy => higher uncertainty
 - Lower entropy => lower uncertainty
- Conditional Entropy
 - $H(Y|X) = \sum_x p(x)H(Y|X = x)$



$$m = 2$$

Attribute Selection Measure: Information Gain (ID3/C4.5)

- Select the attribute with the highest information gain
- Let p_i be the probability that an arbitrary tuple in D belongs to class $C_{i,D}$, 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

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

Attribute Selection: Information Gain

- Class P: `buys_computer = "yes"`
- Class N: `buys_computer = "no"`

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

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

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

$$+ \frac{5}{14} I(3,2) = 0.694$$

$\frac{5}{14} I(2,3)$ means “age ≤ 30 ” has 5 out of 14 samples, with 2 yes’es and 3 no’s. Hence

$$Gain(age) = Info(D) - Info_{age}(D) = 0.246$$

Similarly,

$$Gain(income) = 0.029$$

$$Gain(student) = 0.151$$

$$Gain(credit_rating) = 0.048$$

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

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 value 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* for A is selected as the split-point for A
- Split:
 - D1 is the set of tuples in D satisfying $A \leq \text{split-point}$, and D2 is the set of tuples in D satisfying $A > \text{split-point}$

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 (normalization to 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. $SplitInfo_{income}(D) = -\frac{4}{14} \times \log_2\left(\frac{4}{14}\right) - \frac{6}{14} \times \log_2\left(\frac{6}{14}\right) - \frac{4}{14} \times \log_2\left(\frac{4}{14}\right) = 1.557$
 - $gain_ratio(income) = 0.029/1.557 = 0.019$
- The attribute with the maximum gain ratio is selected as the splitting attribute

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 (*need to enumerate all the possible splitting points for each attribute*)

Computation of Gini Index

- Ex. D has 9 tuples 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
- $$\begin{aligned} gini_{income \in \{low, medium\}}(D) &= \left(\frac{10}{14} \right) Gini(D_1) + \left(\frac{4}{14} \right) Gini(D_2) \\ &= \frac{10}{14} \left(1 - \left(\frac{7}{10} \right)^2 - \left(\frac{3}{10} \right)^2 \right) + \frac{4}{14} \left(1 - \left(\frac{2}{4} \right)^2 - \left(\frac{2}{4} \right)^2 \right) \\ &= 0.443 \\ &= Gini_{income \in \{high\}}(D). \end{aligned}$$

Gini_{low,high} is 0.458; Gini_{medium,high} is 0.450. Thus, split on the {low,medium} (and {high}) since it has the lowest Gini index

- All attributes are assumed continuous-valued
- May need other tools, e.g., clustering, to get the possible split values
- Can be modified for categorical attributes

Comparing Attribute Selection Measures

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

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
- MDL (Minimal Description Length) principle (i.e., the simplest solution is preferred):
 - The best tree as the one that requires the fewest # of bits to both (1) encode the tree, and (2) encode the exceptions to the tree
- 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

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
- Two approaches to avoid overfitting
 - Prepruning: *Halt tree construction early*-do not split a node if this would result in the goodness measure falling below a threshold
 - Difficult to choose an appropriate threshold
 - Postpruning: *Remove branches* from a “fully grown” tree—get a sequence of progressively pruned trees
 - Use a set of data different from the training data to decide which is the “best pruned tree”

Enhancements to Basic Decision Tree Induction

- Allow for **continuous-valued attributes**
 - Dynamically define new discrete-valued attributes that partition the continuous attribute value into a discrete set of intervals
- Handle **missing attribute values**
 - Assign the most common value of the attribute
 - Assign probability to each of the possible values
- **Attribute construction**
 - Create new attributes based on existing ones that are sparsely represented
 - This reduces fragmentation, repetition, and replication

Classification in Large Databases

- Classification—a classical problem extensively studied by statisticians and machine learning researchers
- Scalability: Classifying data sets with millions of examples and hundreds of attributes with reasonable speed
- Why is decision tree induction popular?
 - relatively faster learning speed (than other classification methods)
 - convertible to simple and easy to understand classification rules
 - can use SQL queries for accessing databases
 - comparable classification accuracy with other methods
- RainForest (VLDB'98 — Gehrke, Ramakrishnan & Ganti)
 - Builds an AVC-list (attribute, value, class label)

Scalable Decision Tree Induction Methods

- Classifiers from machine learning community:
 - ID3[Qui86]
 - C4.5[Qui93]
 - CART[BFO84]
- Pruning phase followed by building phase

Scalable Decision Tree Induction Methods

- **SLIQ** (EDBT'96 — Mehta et al.)
 - Builds an index for each attribute and only class list and the current attribute list reside in memory
- **SPRINT** (VLDB'96 — J. Shafer et al.)
 - Constructs an attribute list data structure
- **PUBLIC** (VLDB'98 — Rastogi & Shim)
 - Integrates tree splitting and tree pruning: stop growing the tree earlier
- **RainForest** (VLDB'98 — Gehrke, Ramakrishnan & Ganti)
 - Builds an AVC-list (attribute, value, class label)
- **BOAT** (PODS'99 — Gehrke, Ganti, Ramakrishnan & Loh)
 - Uses bootstrapping to create several small samples

Decision Tree Algorithm

- A decision tree is created in two phases:
 - Building Phase
 - Recursively split nodes using best splitting attribute for node until all the examples in each node belong to one class
 - Pruning Phase
 - Prune leaf nodes recursively to prevent over-fitting
 - Smaller imperfect decision tree generally achieves better accuracy

SPRINT - overview

- [VLDB'96 — J. Shafer et al.]
- A scalable classifier(no memory restriction)
- Use pre-sorting method(for numerical attribute)
- Data Structures for tree making
 - Attribute lists
 - Histograms
- MDL principle for encoding and pruning

SPRINT

- [Shafer, Agrawal, Manish 96]
- Building Phase
 - Initialize root node of tree
 - while** a node N that can be split exists **do**
 - for each** attribute A, evaluate splits on A
 - use best split to split N
- Use gini index to find best split
- Separate attribute lists maintained in each node of tree
- Attribute lists for numeric attributes sorted

How can we get best split?

- Select the attribute that is most useful for classifying training set
- Gini index and entropy
 - Statistical properties
 - Measure how well an attribute separates the training set
 - Entropy ($\text{entropy}(T) = - \sum p_j \times \log_2(p_j)$)
 - Gini Index ($\text{gini}(T) = 1 - \sum p_j^2$)

Performing the Split

- Each attribute list will be partitioned into two lists, one for each child
- Splitting attribute
 - Scan the attribute list, apply the split test, and move records to one of the two new lists
- Non-splitting attribute
 - Cannot apply the split test on non-splitting attributes
 - Use rid to split attribute lists

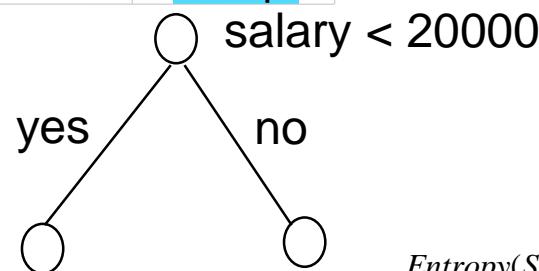
An Entropy Example

salary	company	label
10000	KTF	reject
40000	LGT	accept
15000	LGT	reject
75000	SKT	accept
18000	SKT	accept

$$Pobability(class = "reject") = \frac{2}{5}$$

$$Pobability(class = "accept") = \frac{3}{5}$$

$$\begin{aligned} Entropy(S) &= -\frac{3}{5} \log_2 \frac{3}{5} - \frac{2}{5} \log_2 \frac{2}{5} \\ &= 0.970951 \end{aligned}$$



$$\begin{aligned} Entropy(S_{left}) &= -\frac{1}{3} \log_2 \frac{1}{3} - \frac{2}{3} \log_2 \frac{2}{3} \\ &= 0.918296 \end{aligned}$$

$$Entropy(S_{right}) = 0$$

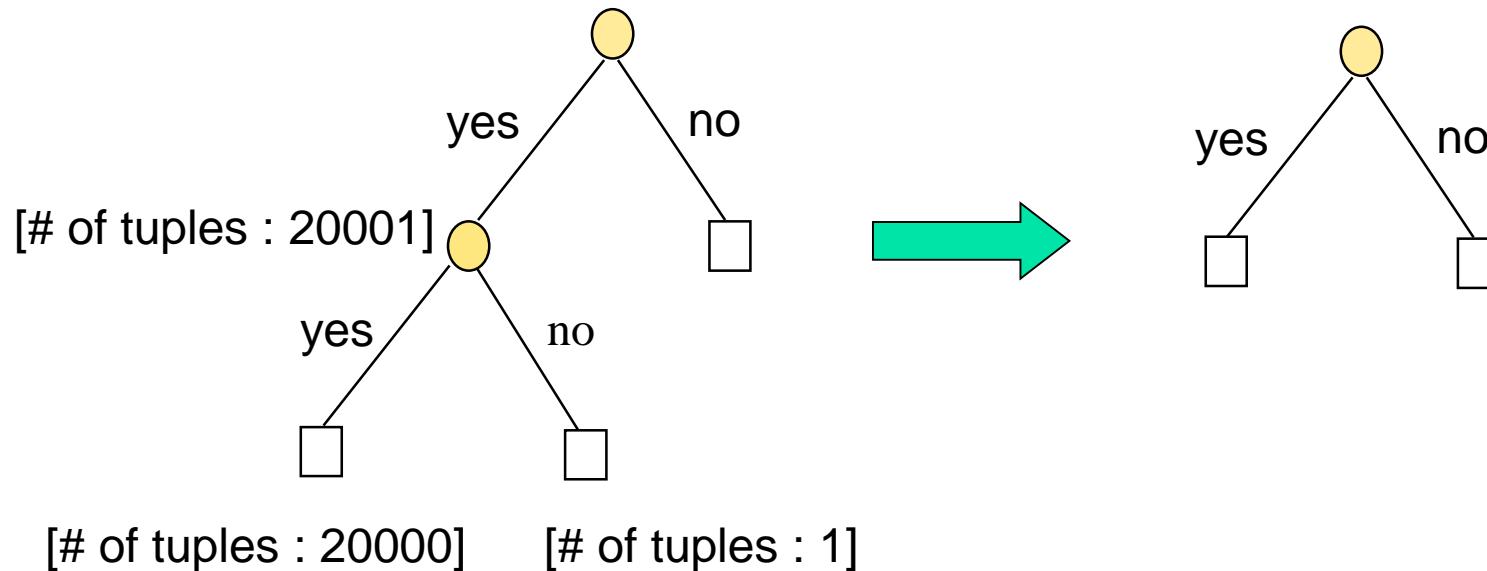
salary	company	label
10000	LGT	reject
15000	LGT	reject
18000	SKT	accept

salary	company	label
40000	LGT	accept
75000	SKT	accept

$$E_{split}(S) = \frac{N_{left}}{N} E(S_{left}) + \frac{N_{right}}{N} E(S_{right}) \longrightarrow Entropy_{split}(S) = \frac{3}{5} \times 0.918296 + \frac{2}{5} \times 0 = 0.550978$$

Pruning Phase

- Smaller imperfect decision tree generally achieves better accuracy
- Prune leaf nodes recursively to prevent over-fitting



Attribute List

- SPRINT creates an attribute list for each attribute
- Numerical attribute list is sorted
- Attribute records contains
 - Attribute value
 - Class label
 - Index of the record

[Training set]

salary	company	label
10000	KTF	reject
40000	LGT	accept
15000	LGT	reject
75000	SKT	accept
18000	SKT	accept

salary	label	rid
10000	reject	0
15000	reject	2
18000	accept	4
40000	accept	1
75000	accept	3

[Attribute list for salary]

company	label	rid
KTF	reject	0
LGT	accept	1
LGT	reject	2
SKT	accept	3
SKT	accept	4

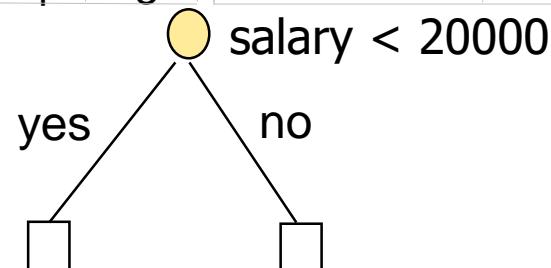
[Attribute list for company]

Attribute list (cont.)

- All attribute lists are made at the root
- As the tree is grown, the attribute lists belonging to each node are partitioned and associated with the children

salary	label	rid
10000	reject	0
15000	reject	2
18000	accept	4
40000	accept	1
75000	accept	3

company	label	rid
KTF	reject	0
LGT	accept	1
LGT	reject	2
SKT	accept	3
SKT	accept	4



salary	label	rid
10000	reject	0
15000	reject	2
18000	accept	4

salary	label	rid
40000	accept	1
75000	accept	3

company	label	rid
KFT	reject	0
LGT	reject	2
SKT	accept	4

company	label	rid
LGT	accept	1
SKT	accept	3

Histogram

[Attribute List]

salary	label	rid
10000	reject	0
15000	reject	2
18000	accept	4
40000	accept	1
75000	accept	3

[position of cursor]

← position 0
← position 2
← position 5

[state of Class Histograms]

	accept	reject
C_{below}	0	0
C_{above}	3	2
C_{below}	0	2
C_{above}	3	0

- For continuous attributes, two histograms are associated with each decision-tree node. These histograms, denoted as C_{above} and C_{below}
 - C_{below} : maintains this distribution for attribute records that already been processed
 - C_{above} : maintains this distribution for attribute records that have not been processed

Finding Split Points

- Numeric attributes
 - C_{below} initials to zeros
 - C_{above} initials with the class distribution at that node
 - Scan the attribute list to find the best split
- Categorical attributes
 - Scan the attribute list to build the count matrix
 - Use the subsetting algorithm to find the best split

Evaluate numeric attributes

[Histogram for salary]

accept reject

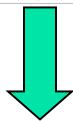
[Position 1]	C _{below}	<table border="1"><tr><td>0</td><td>1</td></tr></table>	0	1	$\rightarrow Entropy_{split}(S) = \frac{1}{5} \times \frac{1}{1} \log 1 + \frac{4}{5} \times \left(-\frac{3}{4} \log \frac{3}{4} - \frac{1}{4} \log \frac{1}{4} \right)$
0	1				
C _{above}	<table border="1"><tr><td>3</td><td>1</td></tr></table>	3	1		
3	1				
[Position 2]	C _{below}	<table border="1"><tr><td>0</td><td>2</td></tr></table>	0	2	$\rightarrow Entropy_{split}(S) = 0.950978$
0	2				
C _{above}	<table border="1"><tr><td>3</td><td>0</td></tr></table>	3	0		
3	0				
[Position 3]	C _{below}	<table border="1"><tr><td>1</td><td>2</td></tr></table>	1	2	$\rightarrow Entropy_{split}(S) = 0.550978$
1	2				
C _{above}	<table border="1"><tr><td>2</td><td>0</td></tr></table>	2	0		
2	0				
[Position 4]	C _{below}	<table border="1"><tr><td>2</td><td>2</td></tr></table>	2	2	$\rightarrow Entropy_{split}(S) = 0.8$
2	2				
C _{above}	<table border="1"><tr><td>1</td><td>0</td></tr></table>	1	0		
1	0				

Choose Position 3 has lowest Entropy!

Evaluate categorical attributes

[Attribute List]

company	label	rid
KTF	reject	0
LGT	accept	1
LGT	reject	2
SKT	accept	3
SKT	accept	4



[Histogram for education]

	accept	reject
KTF	0	1
LGT	1	1
SKT	2	0



→ 3 distinct value → $2^3 - 2$ split condition exists!

{KTF}

$$\begin{aligned} Entropy_{split}(S) &= \frac{1}{5} \times \frac{1}{1} \log 1 + \frac{4}{5} \times \left(-\frac{3}{4} \log \frac{3}{4} - \frac{1}{4} \log \frac{1}{4} \right) \\ &= 0.811278 \end{aligned}$$

{LGT}

$$Entropy_{split}(S) = 0.950978$$

{SKT}

$$Entropy_{split}(S) = \frac{3}{5} \times 0.918296 + \frac{2}{5} \times 0 = 0.550978$$

{KTF, LGT}

$$Entropy_{split}(S) = \frac{3}{5} \times 0.918296 + \frac{2}{5} \times 0 = 0.550978$$

{LGT, SKT}

$$Entropy_{split}(S) = 0.550978$$

{KTF, SKT}

$$Entropy_{split}(S) = 0.811278$$

Choose {SKT} has lowest Entropy!

Overfitting

- Consider error of hypothesis h over
 - training data: $\text{error}_{\text{train}}(h)$
 - entire distribution D of data: $\text{error}_D(h)$
- Hypothesis $h \in H$ **overfits** training data if there is an alternative hypothesis $h' \in H$ such that

$$\text{error}_{\text{train}}(h) < \text{error}_{\text{train}}(h')$$

and

$$\text{error}_D(h) > \text{error}_D(h')$$

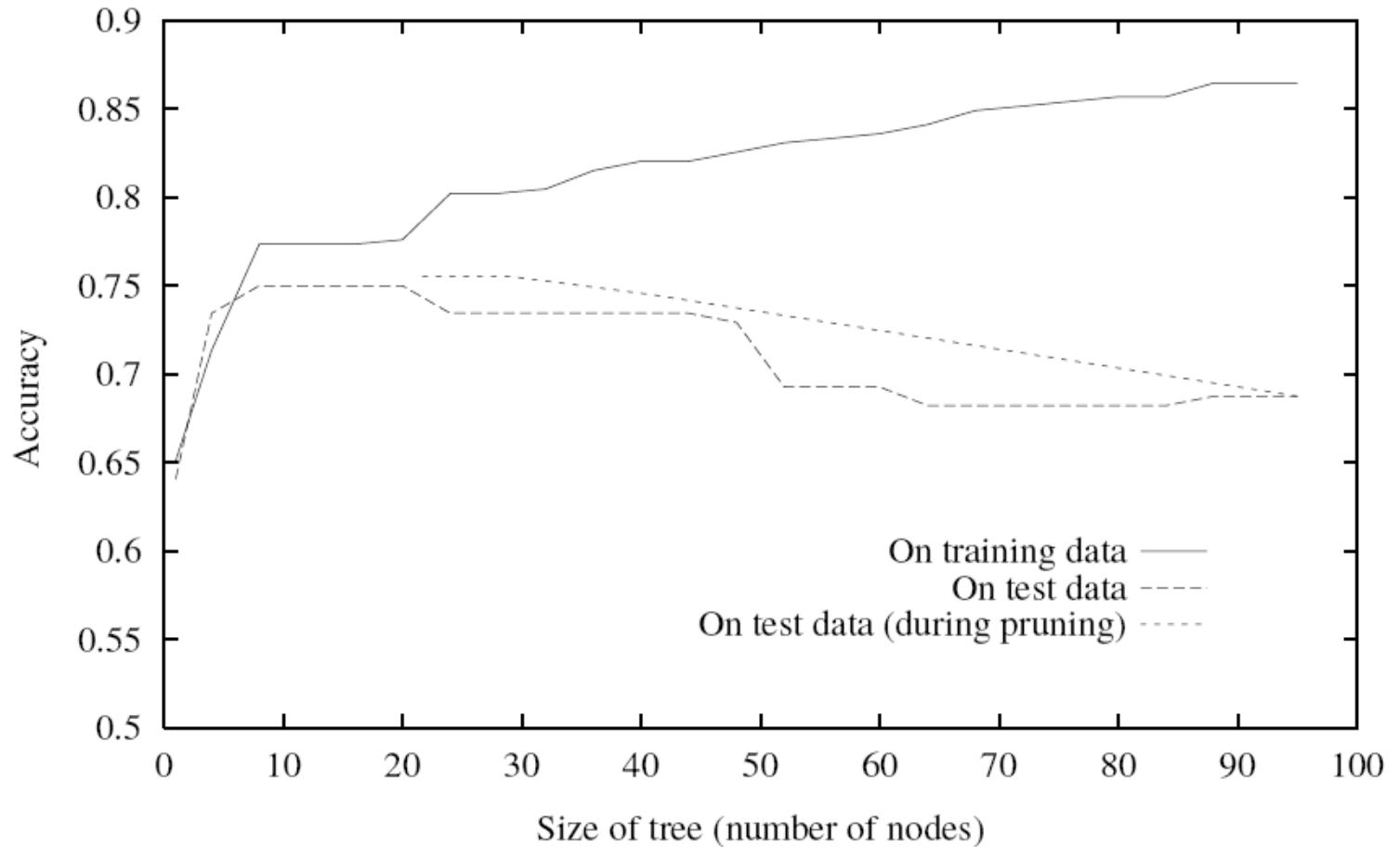
Avoiding Overfitting

- How can we avoid overfitting?
 - Method 1: Stop growing when data split not statistically significant
 - Method 2: Grow full tree, then post-prune
- How to select the “best” tree:
 - Measure performance over training data
 - Measure performance over separate validation data set
- MDL: minimize
 $\text{size(tree)} + \text{size(misclassifications(tree))}$

Reduced-Error Pruning

- Split data into training and validation set
- Do until further pruning is harmful:
 1. Evaluate impact on validation set of pruning each possible node (plus those below it)
 2. Greedily remove the one whose removal most increases validation set accuracy
- Produces smallest version of the most accurate subtrees
- What if data is limited? – use train data!

Effect of Reduced-Error Pruning



MDL Principle

- An information-theoretic measure for quantifying and thereby resolving the tradeoff between the conciseness and preciseness
- MDL principle has been successfully applied in a variety of situations
 - e.g. decision tree classifiers
- Roughly speaking, the best theory to infer from a set of data is the one that minimizes the sum of
 - the length of the theory, in bits (**conciseness**)
 - the length of the data, in bits, when encoded with the help of the theory (**preciseness**)

MDL Module

- MDL Principle: minimize the sum of
 - Theory description length, plus
 - Data description length given the theory
- In order to use MDL, need to:
 - Define theory description length
 - Define data description length given the theory
 - Solve the resulting minimization problem

An Example of MDL: DTD

- Description Length of a DTD
 - Number of bits required to encode the DTD
 - $|\text{Size of DTD}| \times \log (\# \text{ of alphabets})$
- Description Length of a sequence given a candidate DTD
 - Number of bits required to specify the sequence given DTD
 - Use a sequence of encoding indices
 - Encoding of a given a is the empty string
 - Encoding of a given $(a|b|c)$ is the index 0
 - Encoding of aaa given a^* is the index 3

An Example of MDL

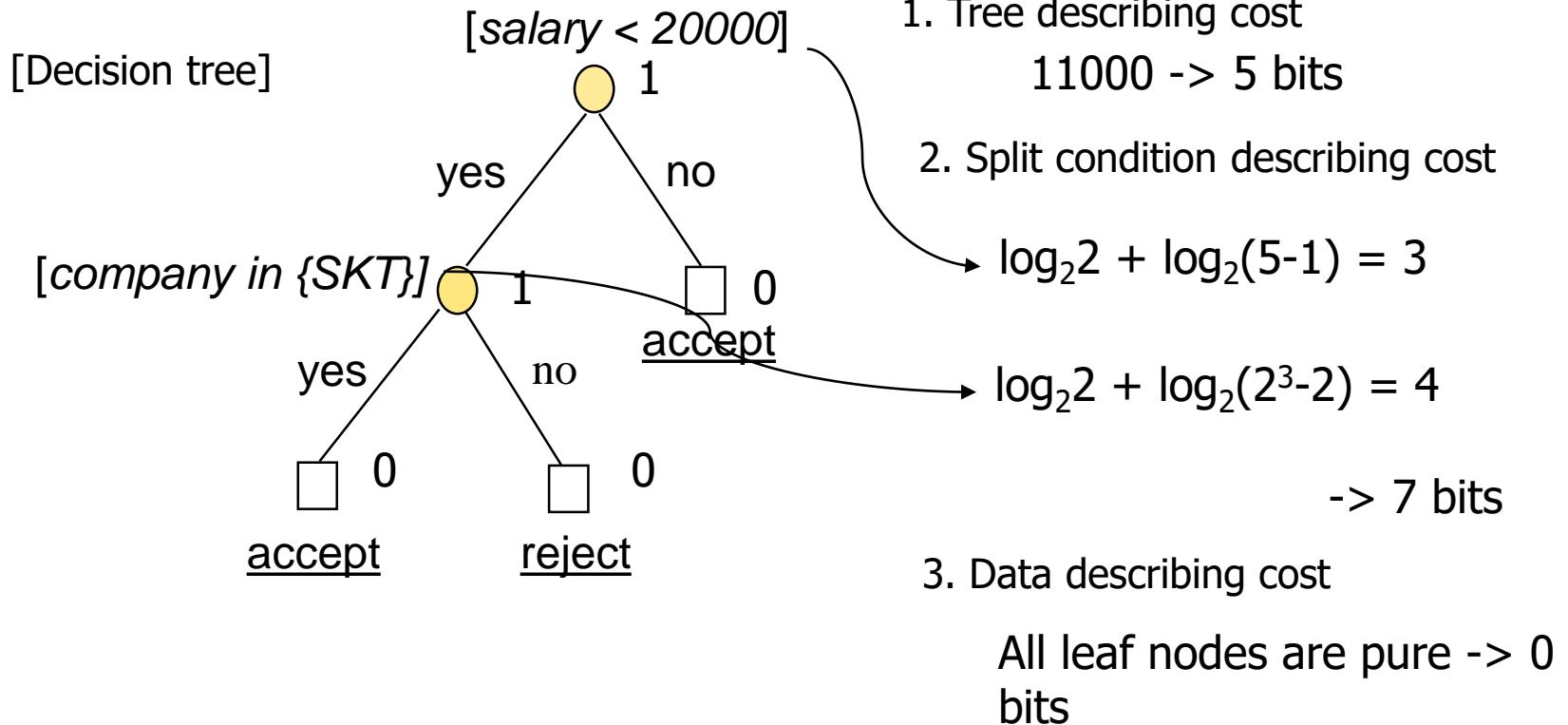
$$I = \{ab, abab, ababab\}$$

- $(a \mid b)^*$
 - abab: cost of 5 (the number of repetitions (4) + 4 characters to represent chosen character)
 - MDL cost = 6 (encoding DTD) + 3 + 5 + 7 = 21
- ab | abab | ababab
 - MDL cost = 14 + 3 = 17
- ab | ab(ab | abab)
 - MDL cost = 14 + 1 + 2 + 2 = 19
- $(ab)^*$
 - MDL cost = 5 + 3 = 8

MDL Principle

- Given a model, M , and the data, D
- MDL principle states that the best model for encoding data is the one that minimizes $\text{Cost}(M,D) = \text{Cost}(D|M) + \text{Cost}(M)$
 - $\text{Cost}(D|M)$ is the cost, in number of bits, of encoding the data given a model M
 - $\text{Cost}(M)$ is the cost of encoding the model M
 - The cost of describing the tree - number of bits used to encode each node
 - The costs of describing the splits

Encoding Example

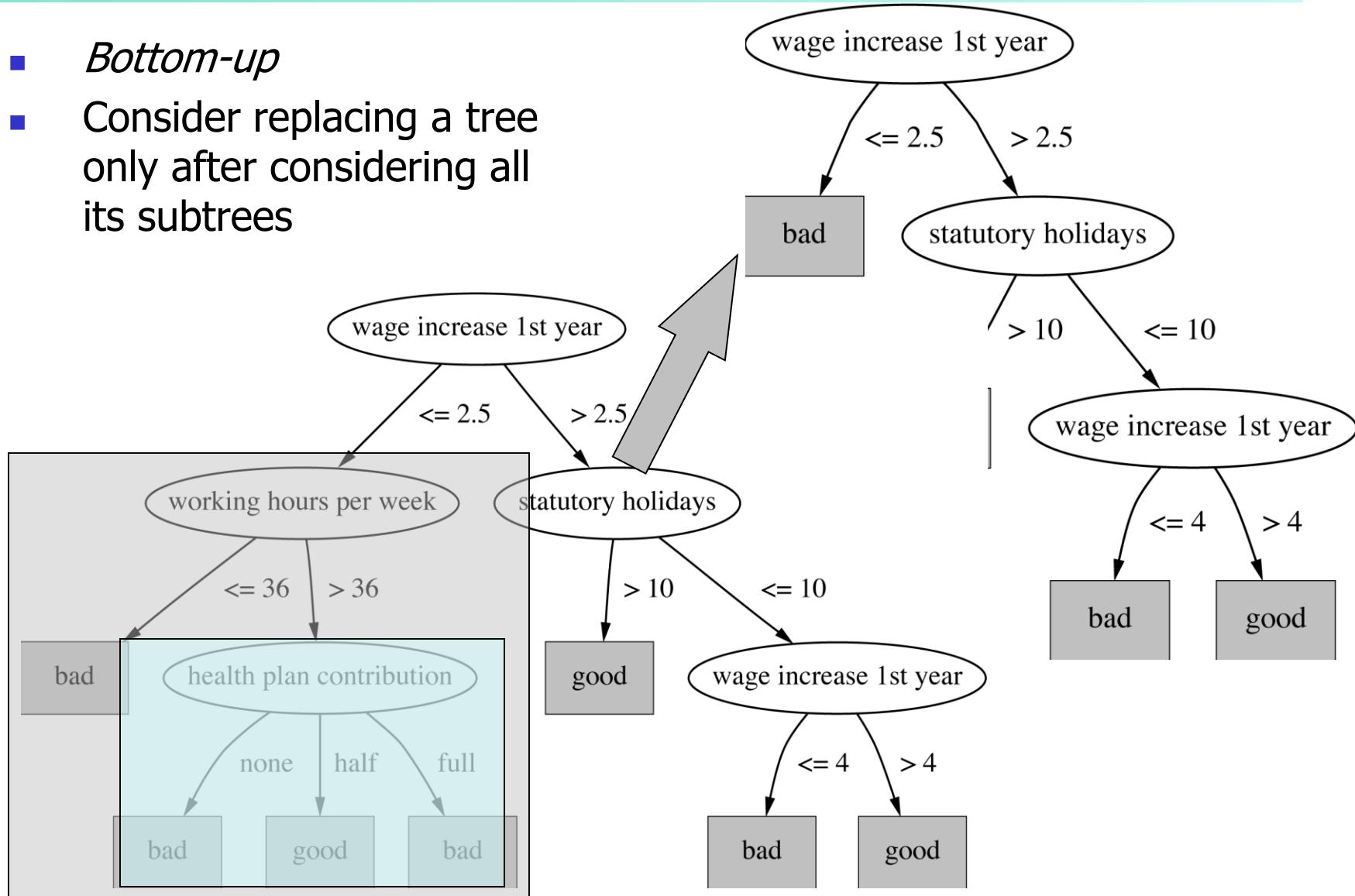


MDL Pruning Algorithm

- The models are the set of trees obtained by pruning the initial decision T
- The data is the training set S
- The goal is to find the subtree of T that best describes the training set S (i.e. with the minimum cost)
- The algorithm evaluates the cost at each decision tree node to determine whether to convert the node into a leaf, prune the left or the right child, or leave the node intact

MDL Pruning - Subtree Replacement

- *Bottom-up*
- Consider replacing a tree only after considering all its subtrees



Pruning Using MDL Principle

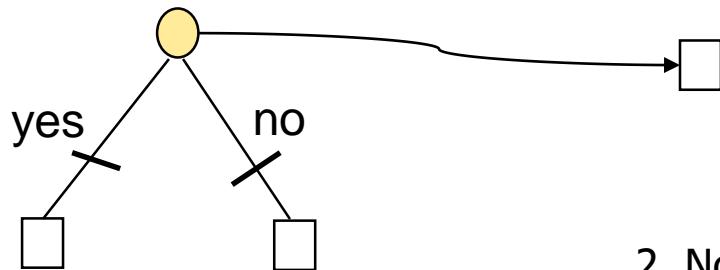
- View decision tree as a means for efficiently encoding classes of records in training set
- **MDL Principle:** best tree is the one that can encode records using the fewest bits
- Cost of encoding tree includes
 - 1 bit for encoding type of each node (e.g. leaf or internal)
 - C_{split} : cost of encoding attribute and value for each split
 - n^*E : cost of encoding the n records in each leaf (E is entropy)

Pruning Using MDL Principle

- Problem: to compute the minimum cost subtree at root of built tree
- Suppose minC_N is the cost of encoding the minimum cost subtree rooted at N
- Prune children of a node N if $\text{minC}_N = n*E+1$
- Compute minC_N as follows:
 - N is leaf: $n*E+1$
 - N has children N_1 and N_2 : $\min\{n*E+1, C_{\text{split}}+1+\text{minC}_{N1}+\text{minC}_{N2}\}$
- Prune tree in a bottom-up fashion

MDL Pruning Example

[Decision tree]



1. Pruned case

(tree describing cost) = 1bit

(split condition describing cost)= 0bit

(data describing cost) = $C(s)$

(total cost) = $1 + C(s)$

2. Not pruned case

(tree describing cost) = $1+1+1 = 3$ bit

(split condition describing cost)= $C_{\text{split}}(S)$

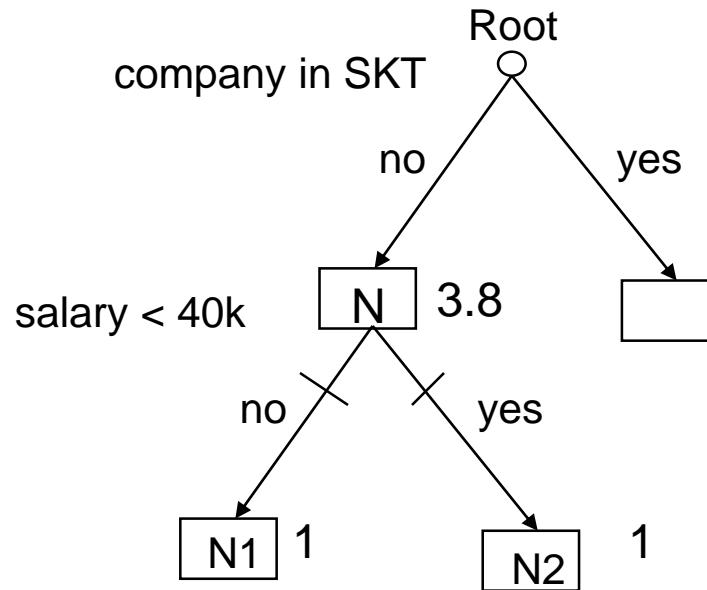
(data describing cost) = $C(S_{\text{left}}) + C(S_{\text{right}})$

(total cost) = $3 + C_{\text{split}}(S) + C(S_{\text{left}}) + C(S_{\text{right}})$

If $[1+C(S)] < [3+C_{\text{split}}(S) + C(S_{\text{left}}) + C(S_{\text{right}})]$ then Prune!

MDL Pruning - Example

10	reject	1
18	reject	5
40	accept	2

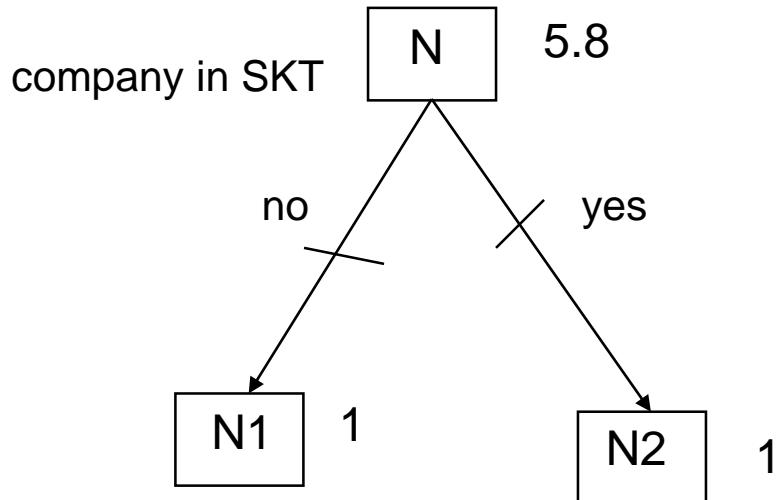


- Cost of encoding records in N ($n \cdot E + 1$) = 3.8
- $C_{\text{split}} = 2.6$
- $\min CN = \min\{3.8, 2.6+1+1+1\} = 3.8$
- Since $\min CN = n \cdot E + 1$, N1 and N2 are pruned

PUBLIC

- [Rastogi, Shim 98]
- Prune tree during (**not after**) building phase
- Execute pruning algorithm (**periodically**) on partial tree
- Problem: how to compute minCN for a “yet to be expanded” leaf N in a partial tree
- Solution: compute lower bound on the subtree cost at N and use this as minCN when pruning
 - minCN is thus a “lower bound” on the cost of subtree rooted at N
 - Prune children of a node N if $\text{minCN} = n * E + 1$
- **Guaranteed** to generate **identical** tree to that generated by SPRINT

PUBLIC(1)



- Simple lower bound for a subtree: 1
- Cost of encoding records in N = $n \cdot E + 1 = 5.8$
- Csplit = 4
- $\text{minCN} = \min\{5.8, 4+1+1+1\} = 5.8$
- Since $\text{minCN} = n \cdot E + 1$, N1 and N2 are pruned

PUBLIC(S)

Theorem: The cost of any subtree with s splits and rooted at node

$$N \text{ is at least } 2*s+1+s*\log a + \sum_{i=s+2}^k n_i$$

- a is the number of attributes
- k is the number of classes
- n_i ($\geq n_i+1$) is the number of records belonging to class i

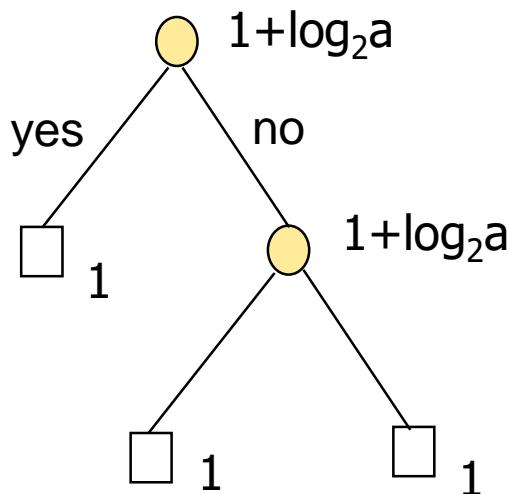
Lower bound on subtree cost at N is thus the minimum of:

- $n*E+1$ (cost with zero split)
- $2*s+1+s*\log a + \sum_{i=s+2}^k n_i$

PUBLIC(S) (Cont.)

Complex lower bound for a subtree: s splits

[Decision tree]



Number of internal nodes : s

Number of leaf nodes : s + 1

Number of minor class: k – (s+1)

(tree describing cost) = $(2*s+1)$ bit

(split condition describing cost) = $\sum C_{\text{split}}(S) = s * \log_2 a$

(data describing cost) = $\sum C(S_{\text{leaf}}) = \sum_{i=s+2}^k n_i$

(lower bound of cost) =

$$(2*s+1) + (s * \log_2 a) + \sum_{i=s+2}^k n_i$$

If $[1+C(S)] < [s\text{-split lower bound}]$

then Prune and Mark pruned

Scalability Framework for RainForest

- Separates the scalability aspects from the criteria that determine the quality of the tree
- Builds an AVC-list: **AVC (Attribute, Value, Class_label)**
- **AVC-set** (of an attribute X)
 - Projection of training dataset onto the attribute X and class label where counts of individual class label are aggregated
- **AVC-group** (of a node n)
 - Set of AVC-sets of all predictor attributes at the node n

Rainforest: Training Set and Its AVC Sets

Training Examples

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

AVC-set on *Age*

Age	Buy_Computer	
	yes	no
<=30	2	3
31..40	4	0
>40	3	2

AVC-set on *income*

income	Buy_Computer	
	yes	no
high	2	2
medium	4	2
low	3	1

AVC-set on *Student*

student	Buy_Computer	
	yes	no
yes	6	1
no	3	4

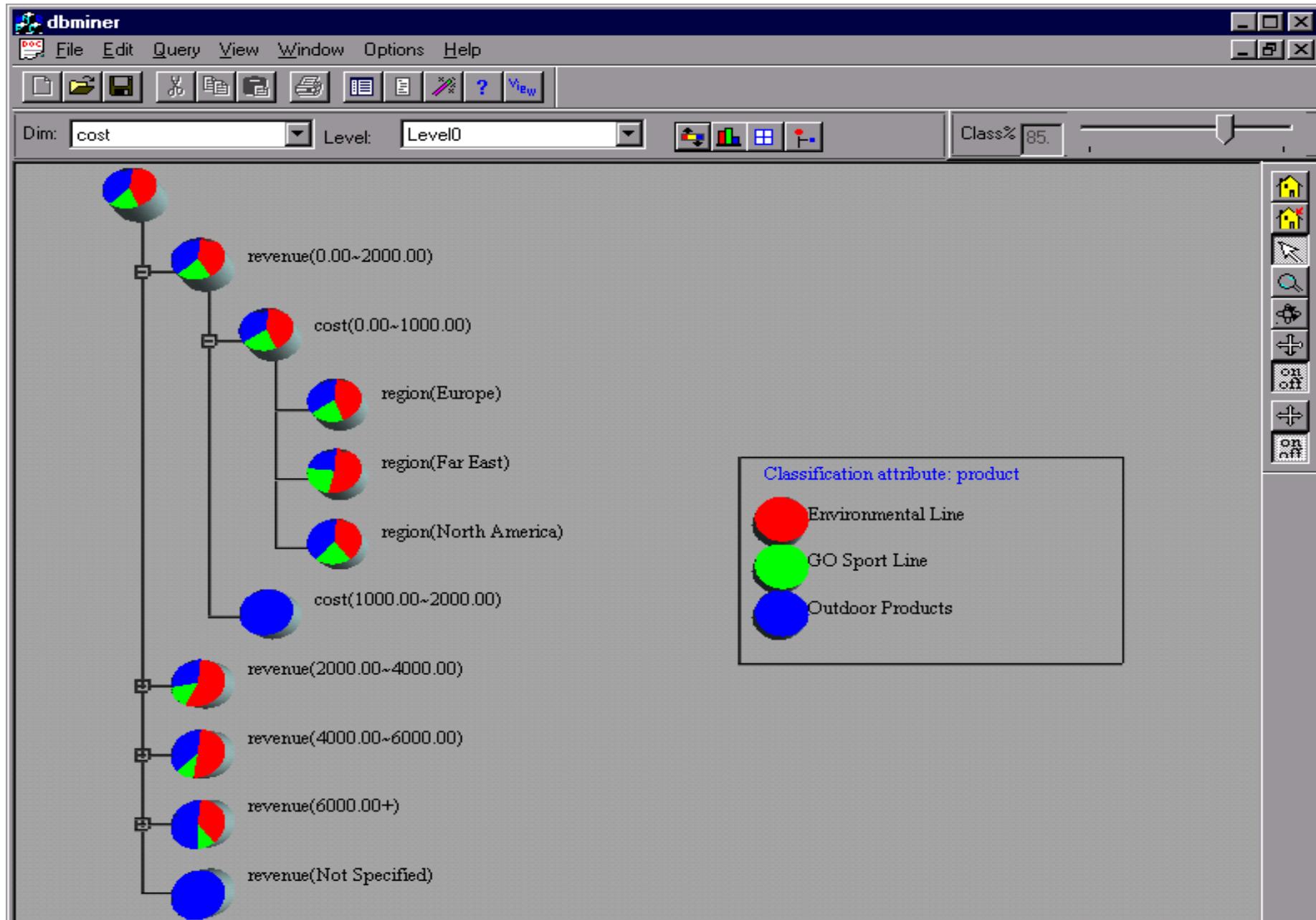
AVC-set on *credit_rating*

Credit rating	Buy_Computer	
	yes	no
fair	6	2
excellent	3	3

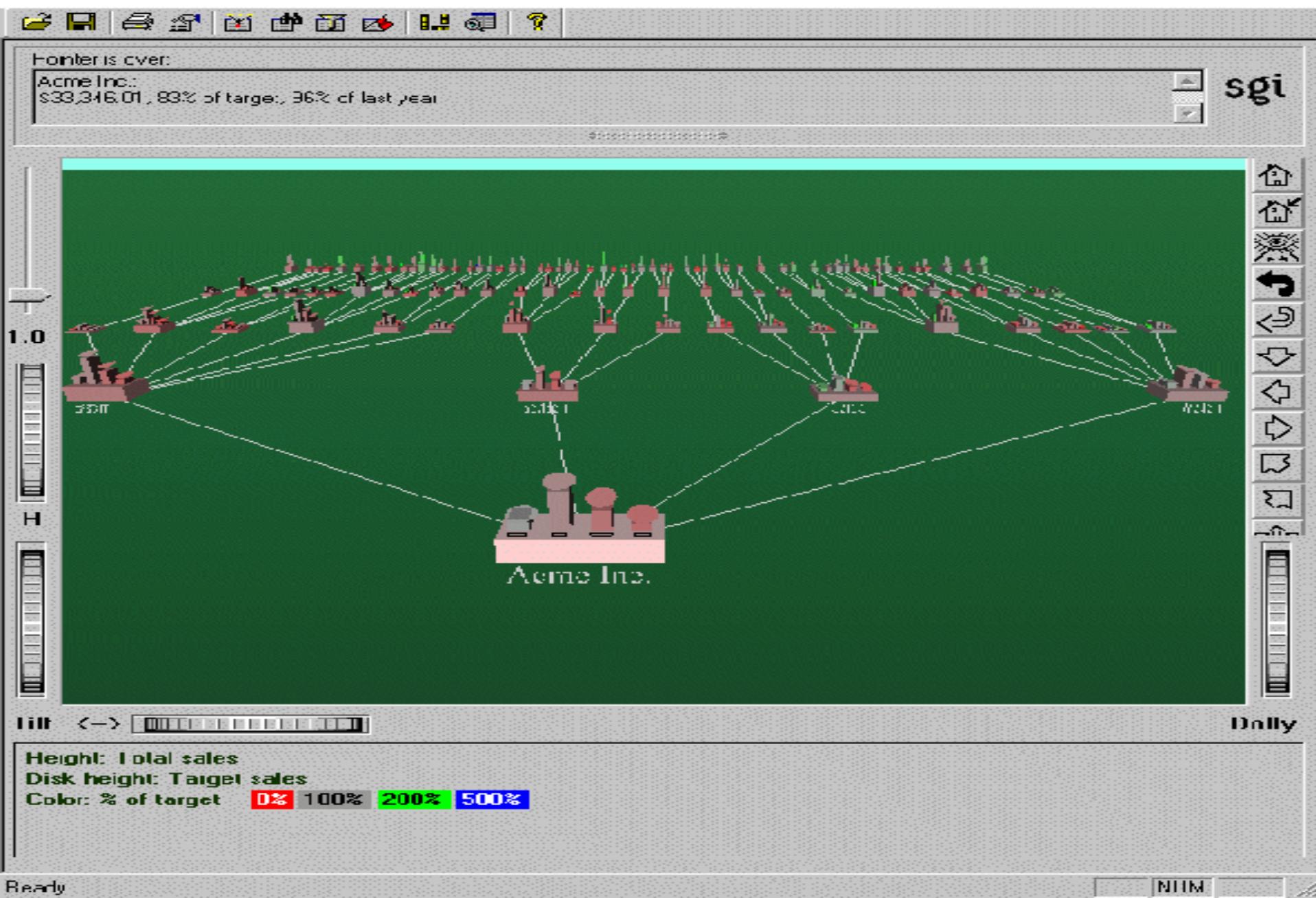
BOAT (Bootstrapped Optimistic Algorithm for Tree Construction)

- Use a statistical technique called *bootstrapping* to create several smaller samples (subsets), each fits in memory
- Each subset is used to create a tree, resulting in several trees
- These trees are examined and used to construct a new tree T'
 - It turns out that T' is very close to the tree that would be generated using the whole data set together
- Adv: requires only two scans of DB, an incremental alg.

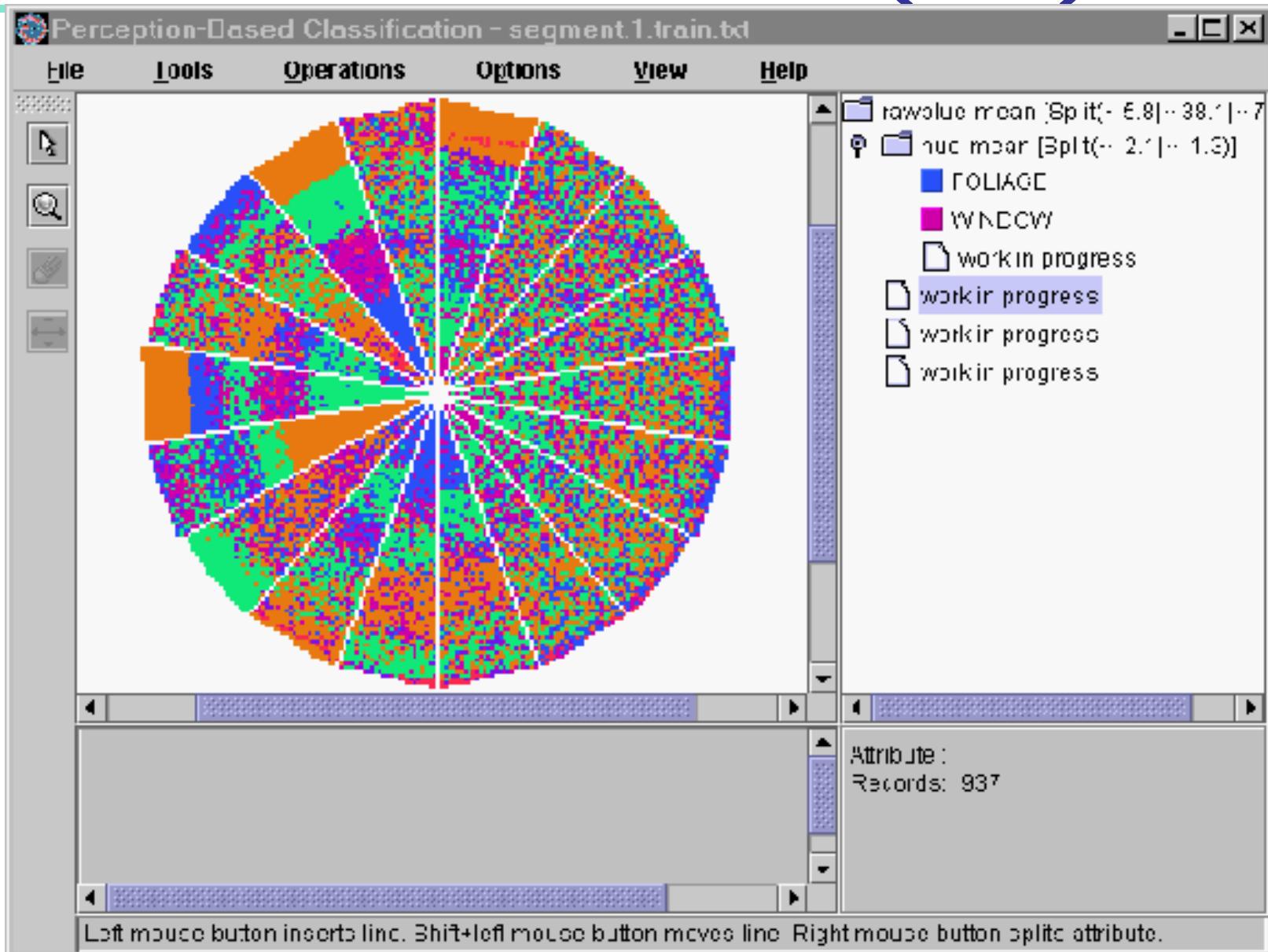
Presentation of Classification Results

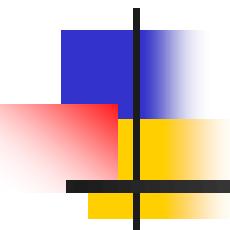


Visualization of a Decision Tree in SGI/MineSet 3.0



Interactive Visual Mining by Perception-Based Classification (PBC)





Weka – Decision Classifier

C4.5rules: choices and options

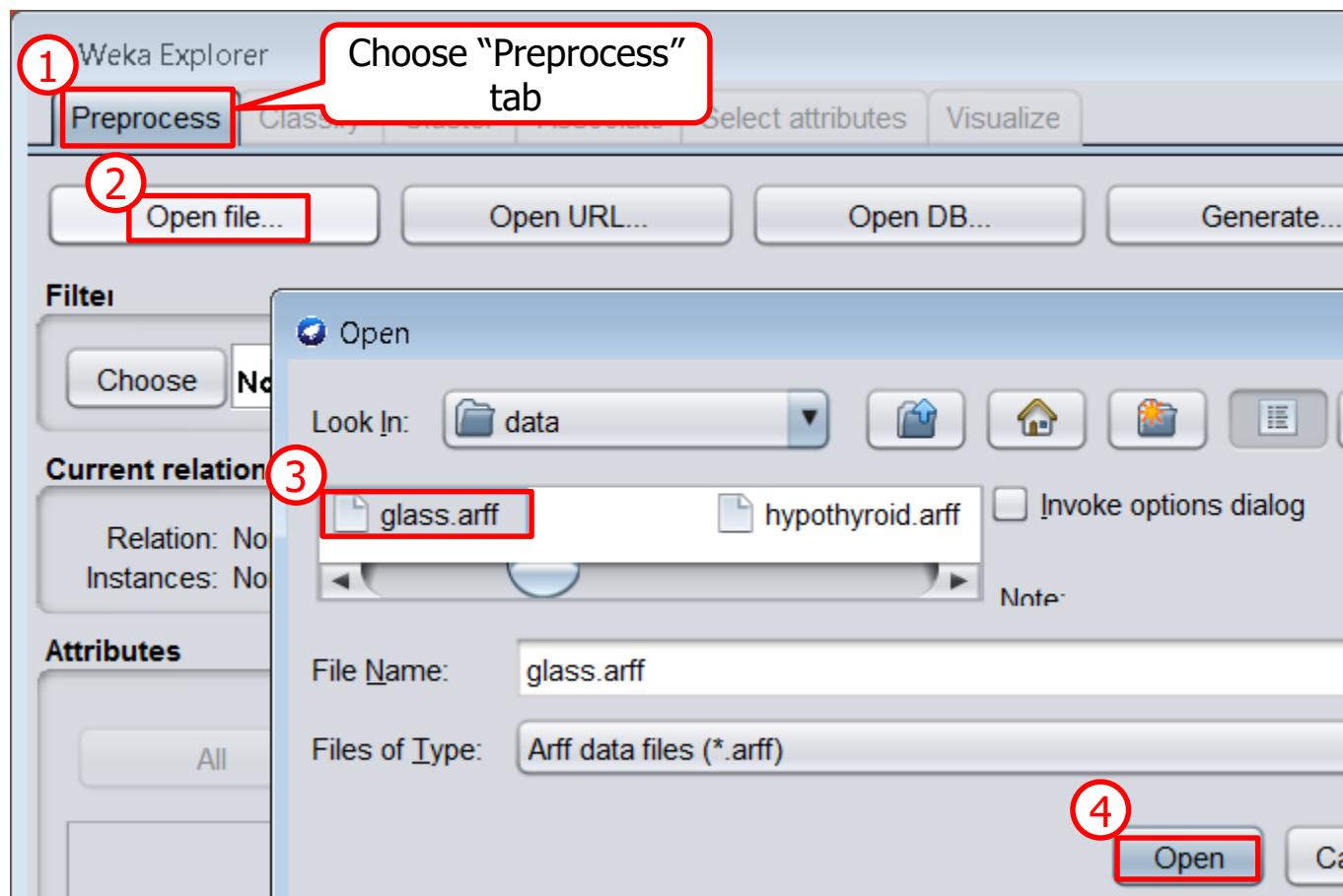
- C4.5rules slow for large and noisy datasets
- Commercial version C5.0rules uses a different technique
 - Much faster and a bit more accurate
- C4.5 has two parameters
 - Confidence value (default 25%): lower values incur heavier pruning
 - Minimum number of instances in the two most popular branches (default 2)

Building a classifier

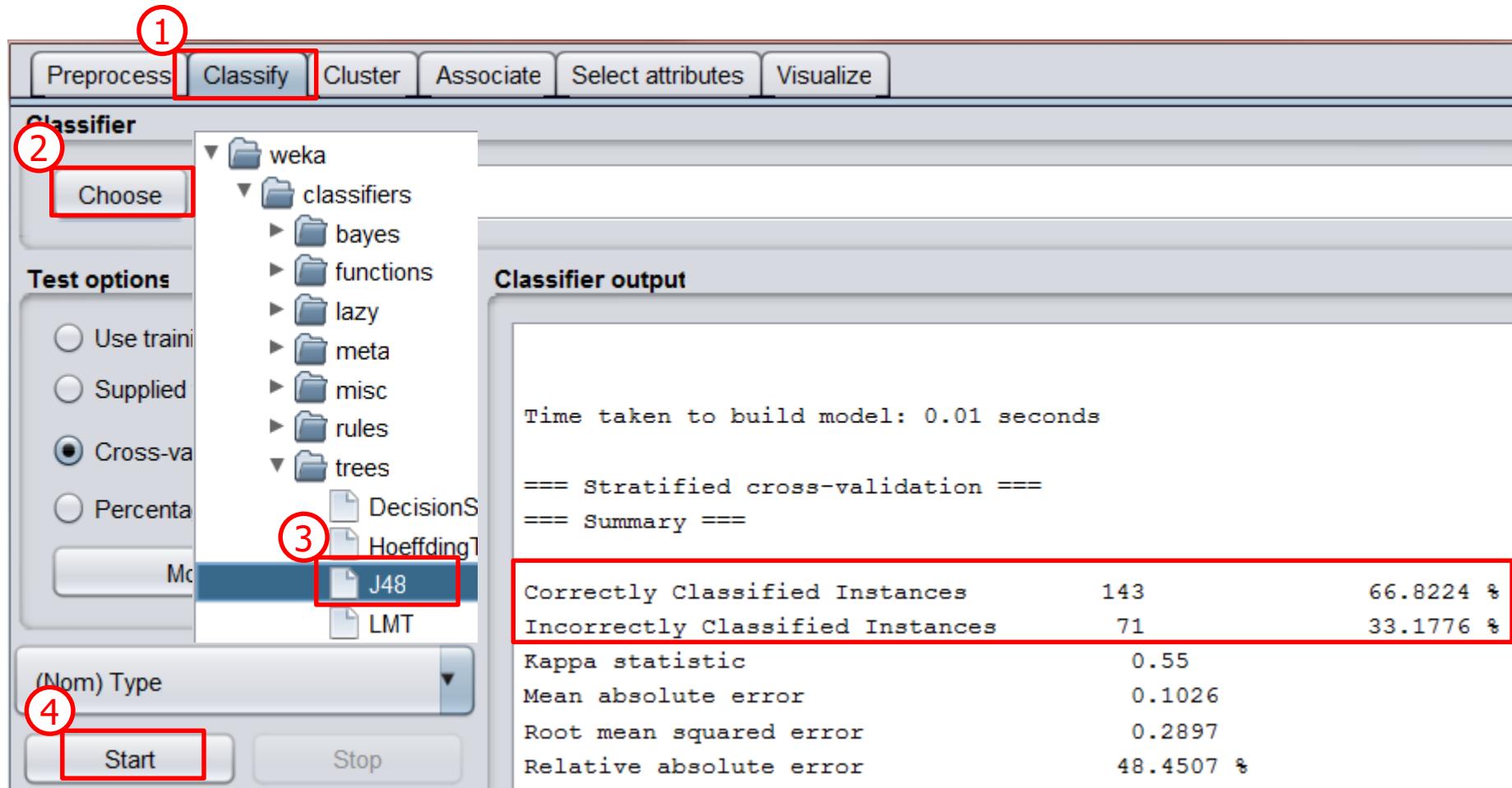
- Use J48 to analyze the glass dataset
 - J48: an open source Java implementation of the C4.5 algorithm
 - Open file glass.arff
 - Check the available classifiers
 - Choose the J48 decision tree learner (trees>J48)
 - Run it
 - Examine the output
 - Look at the correctly classified instances and the confusion matrix

Open the Dataset

- C:\Program Files\Weka-3-8\data\glass.arff



Run the J48



Classifier output

```
==== Classifier model (full training set) ====
```

```
J48 pruned tree
```

```
Ba <= 0.27
```

A leaf node

```
| Mg <= 2.41
```

```
| | K <= 0.03
```

```
| | | Na <= 13.75: build wind non-float (3.0)
```

```
| | | Na > 13.75: tableware (9.0)
```

```
| | K > 0.03
```

The number of instances
classified incorrectly

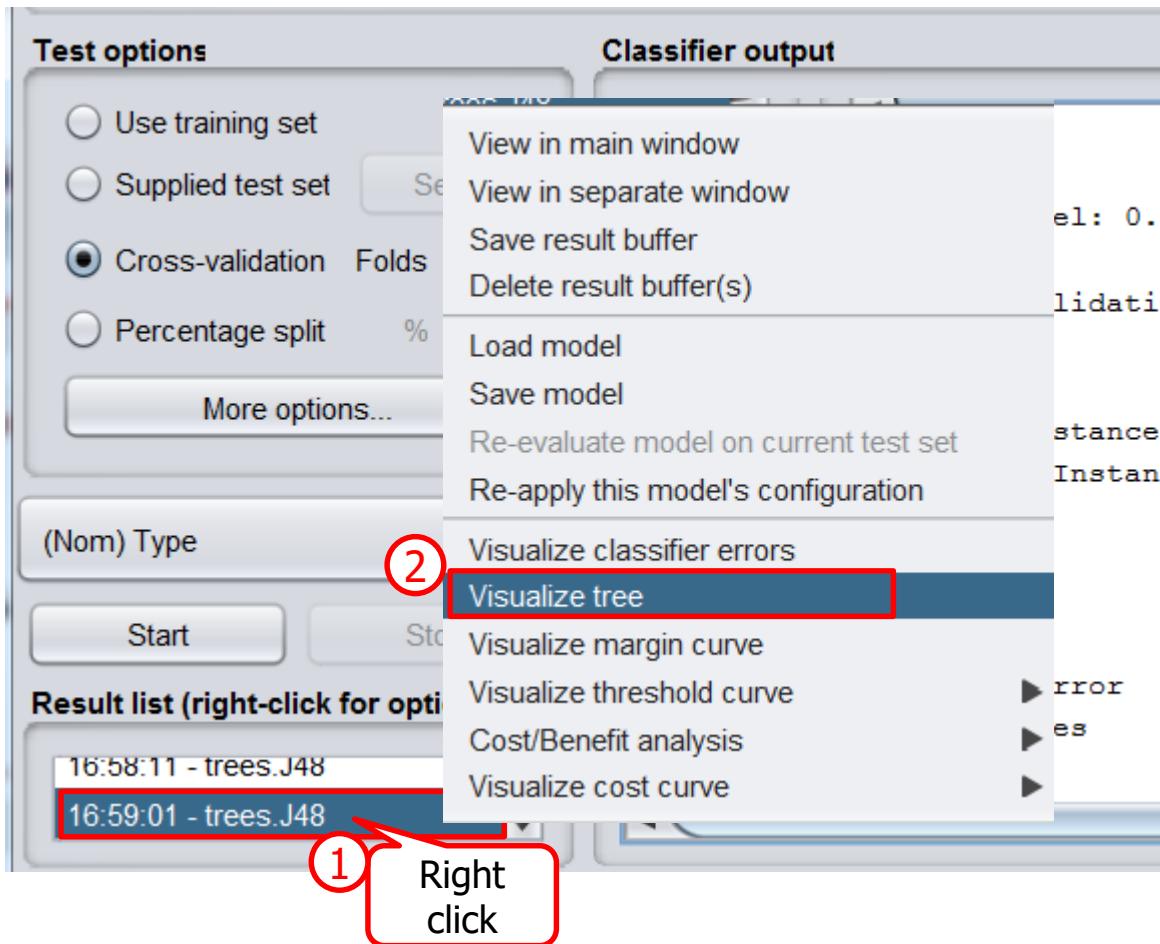
```
| | | Na <= 13.4
```

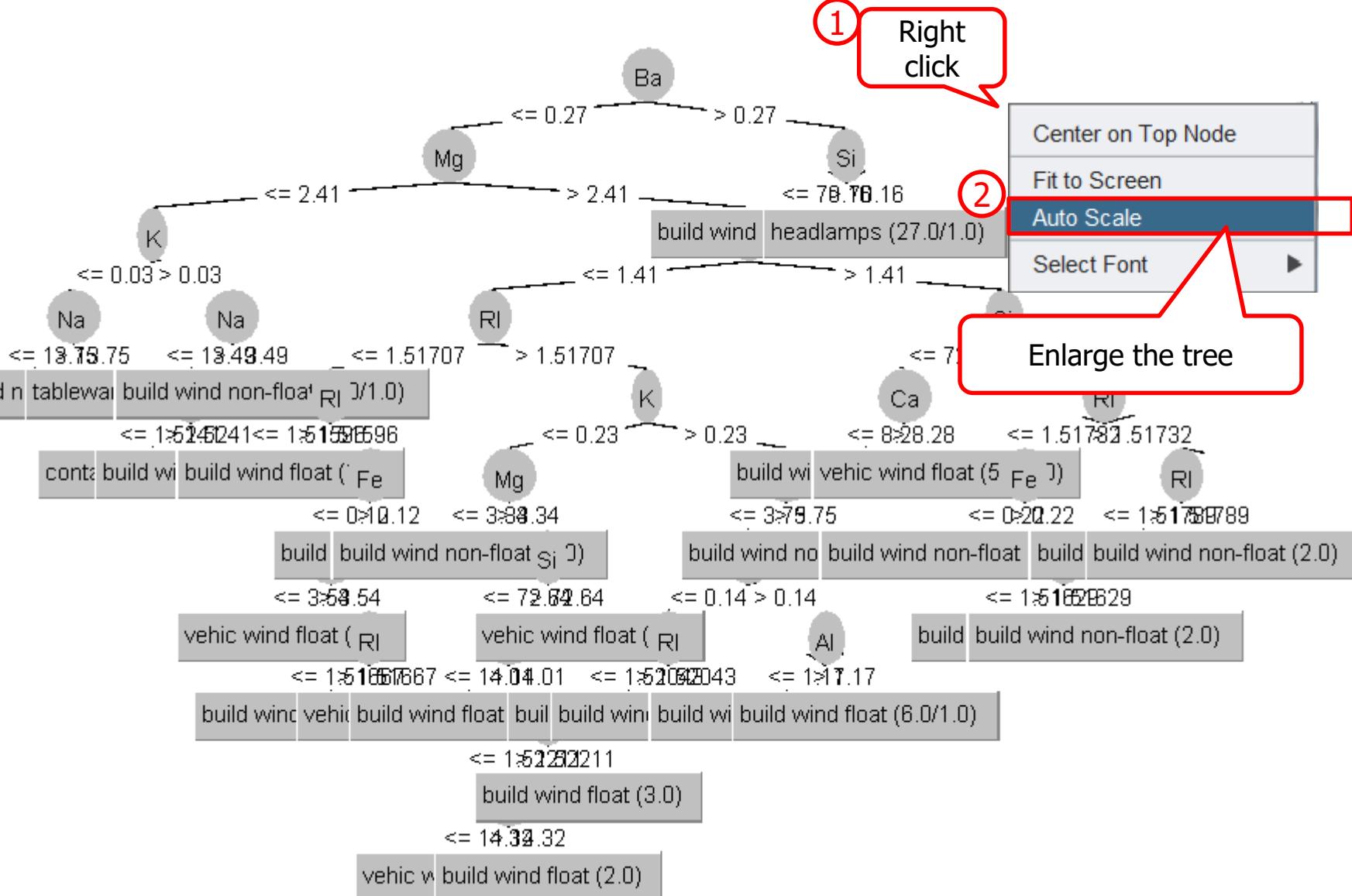
```
| | | | RI <= 1.5241: containers (13.0/1.0)
```

```
| | | | RI > 1.5241: build wind non-float (3.0)
```

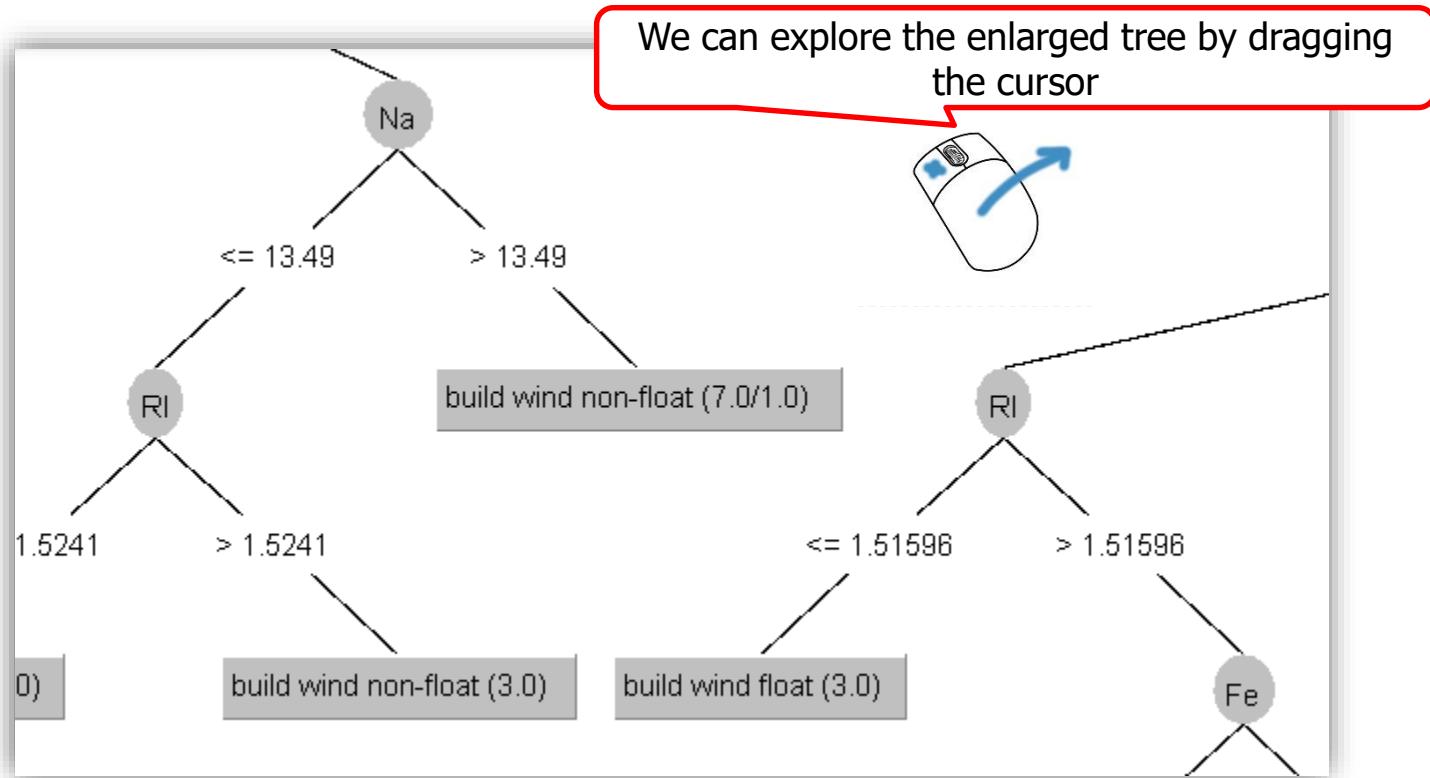
```
| | | Na > 13.49: build wind non-float (7.0/1.0)
```

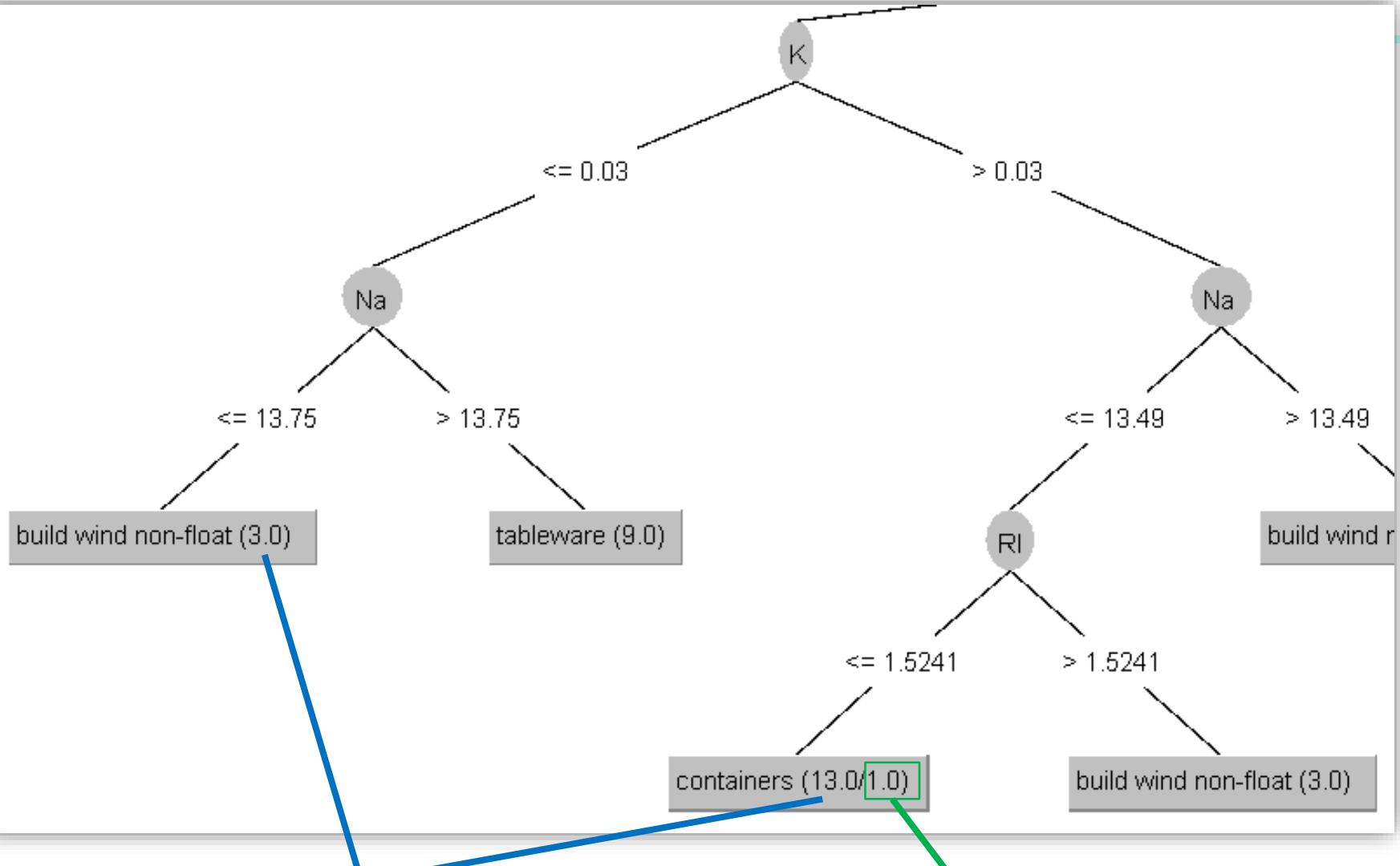
Visualize Tree





Visualize Tree





The number of instances
belonging to the leaf node

The number of instances
classified incorrectly

Using a filter

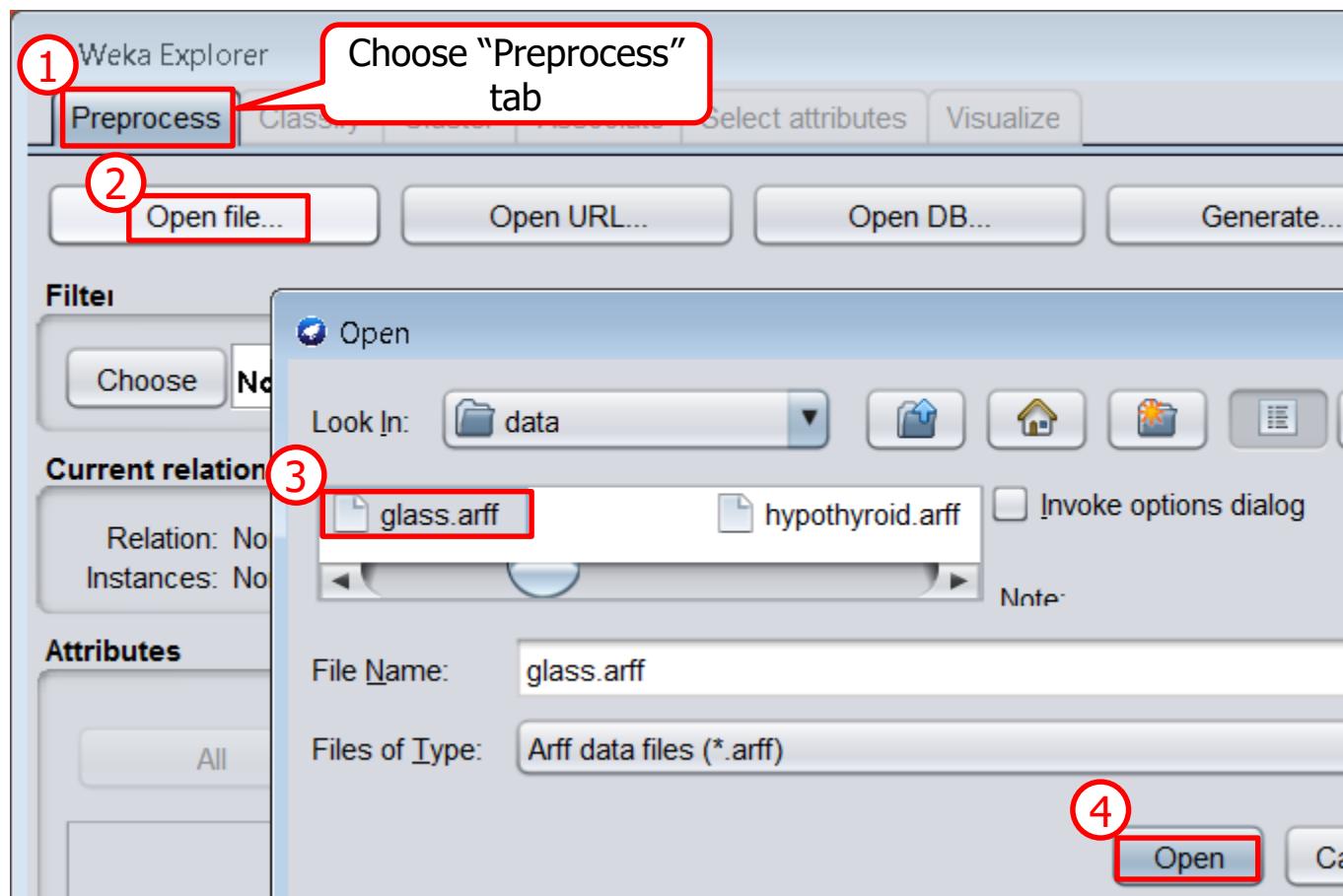
- Filters in Weka
 - Supervised vs unsupervised
 - Supervised: taking advantage of the class information
 - Unsupervised: does not utilize the class information
 - Attribute vs instance
 - Modify or add a new attribute (column) or instance (row)
- To find the right filter, you need to observe data
- Filters can be very powerful
- Judiciously removing attributes can
 - improve performance
 - increase comprehensibility

Using a filter

- Fewer attributes, better classification!
 - Open glass.arff
 - Run J48 (trees>J48)
 - Remove all attributes except RI and Mg
 - Look at the decision trees
 - Use right-click menu to visualize decision trees

Open the Dataset

- C:\Program Files\Weka-3-8\data\glass.arff



Preprocess Classify Cluster Associate Select attributes Visual

Open file... Open URL... Open DB... Gener...

Filter:

Choose **None**

Current relation

Relation: Glass Attributes: 10
Instances: 214 Sum of weights: 214

Attributes

All None Invert Pattern

1

No.	Name
1	RI
2	Na
3	Mg
4	Al
5	Si
6	K
7	Ca
8	Ba
9	Fe
10	Type

2

Remove

1

Preprocess Classify Cluster Associate Select attributes Visualize

2

Classifier

Choose

Test options

Use training data
 Supplied
 Cross-validation
 Percentage

Model

(Nom) Type

Start Stop

Result list (right-click for options)

13:07:03 - trees.J48

weka

classifiers

bayes

functions

lazy

meta

misc

rules

trees

DecisionStump

HoeffdingTree

J48

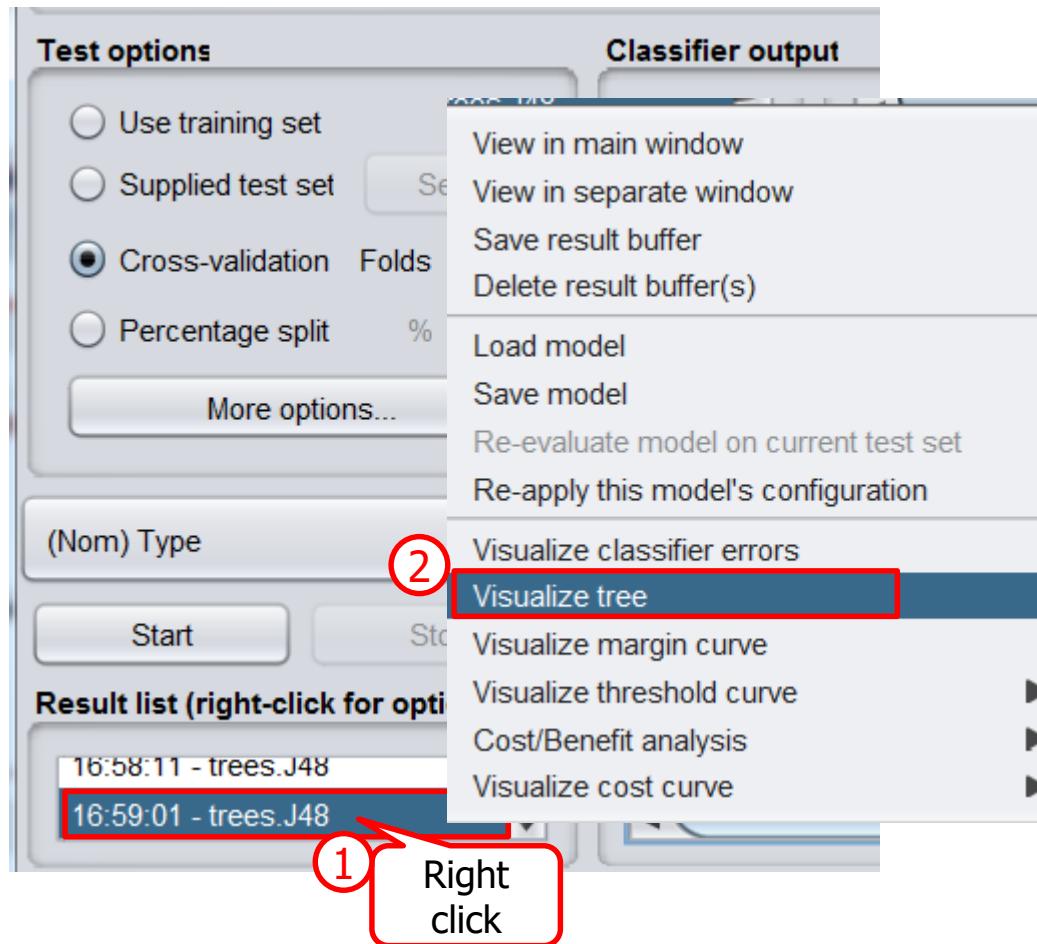
LMT

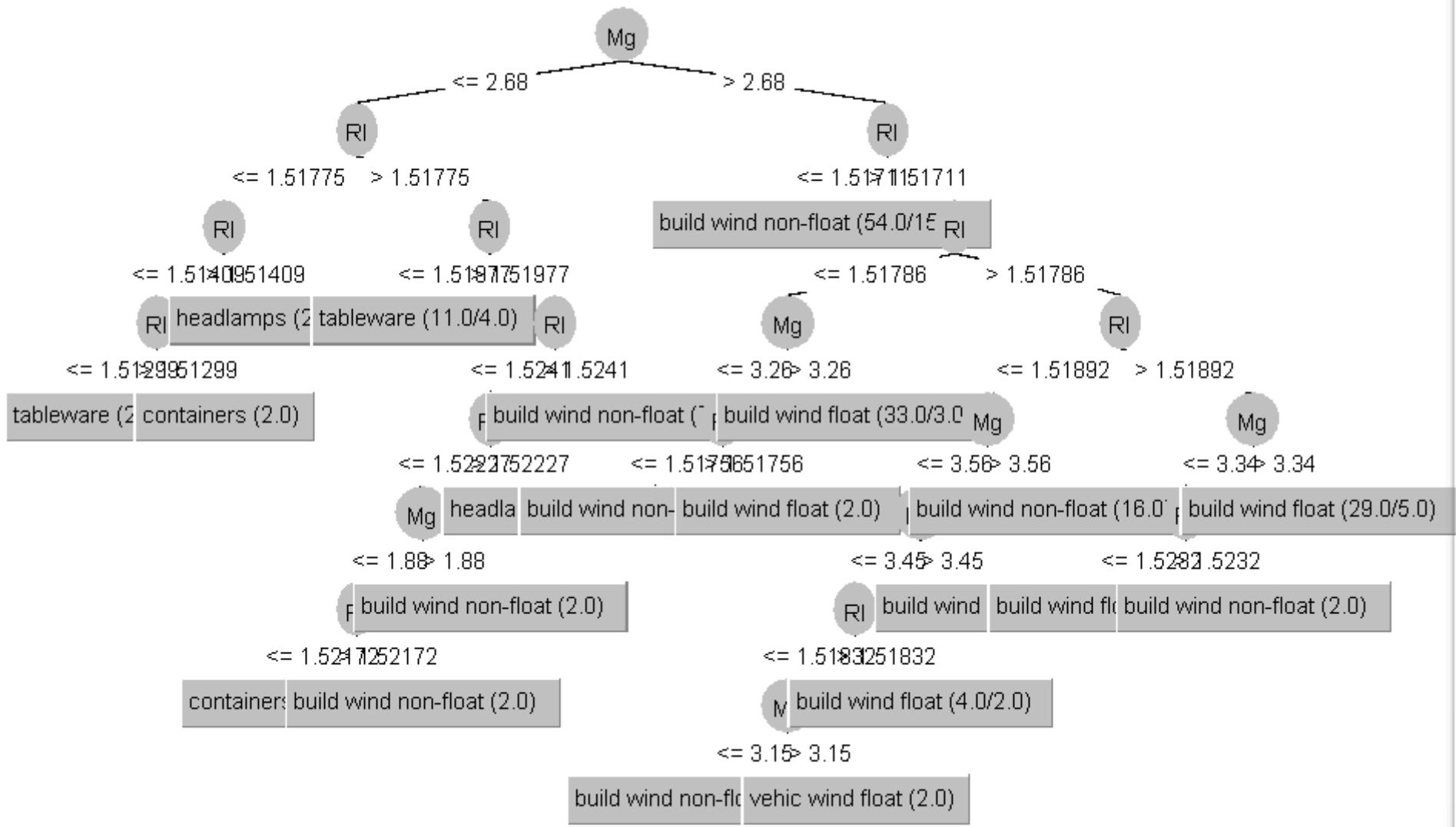
Classifier output

```
Time taken to build model: 0 seconds
==== Stratified cross-validation ====
==== Summary ====
Correctly Classified Instances           147           68.6916 %
Incorrectly Classified Instances        67            31.3084 %
Kappa statistic                         0.5628
Mean absolute error                     0.1124
Root mean squared error                 0.267
Relative absolute error                  53.082 %
Root relative squared error             82.2535 %
Total Number of Instances                214
```

Accuracy was **66.82%** before removing attributes

Visualize Tree



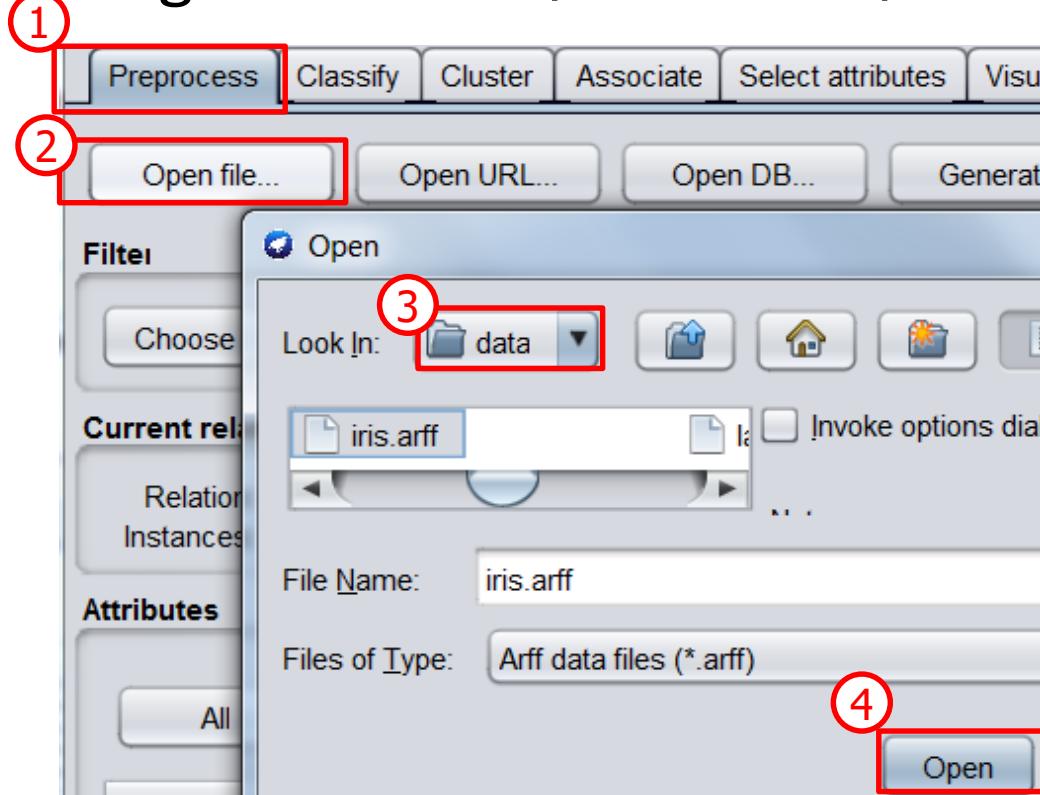


Visualizing your data

- Using the Visualize panel
 - Open iris.arff
 - Bring up Visualize panel
 - Click one of the plots; examine some instances
 - Set x axis to petalwidth and y axis to petallength
 - Click on Class colour to change the colour
 - Bars on the right change correspond to attributes: click for x axis;
■ right-click for y axis
 - Jitter slider - just adds artificial random noise to the coordinates of the plotted points so that you can see points that might have been obscured by others
 - Show Select Instance: Rectangle option
 - Submit, Reset, Clear and Save

Open the Dataset

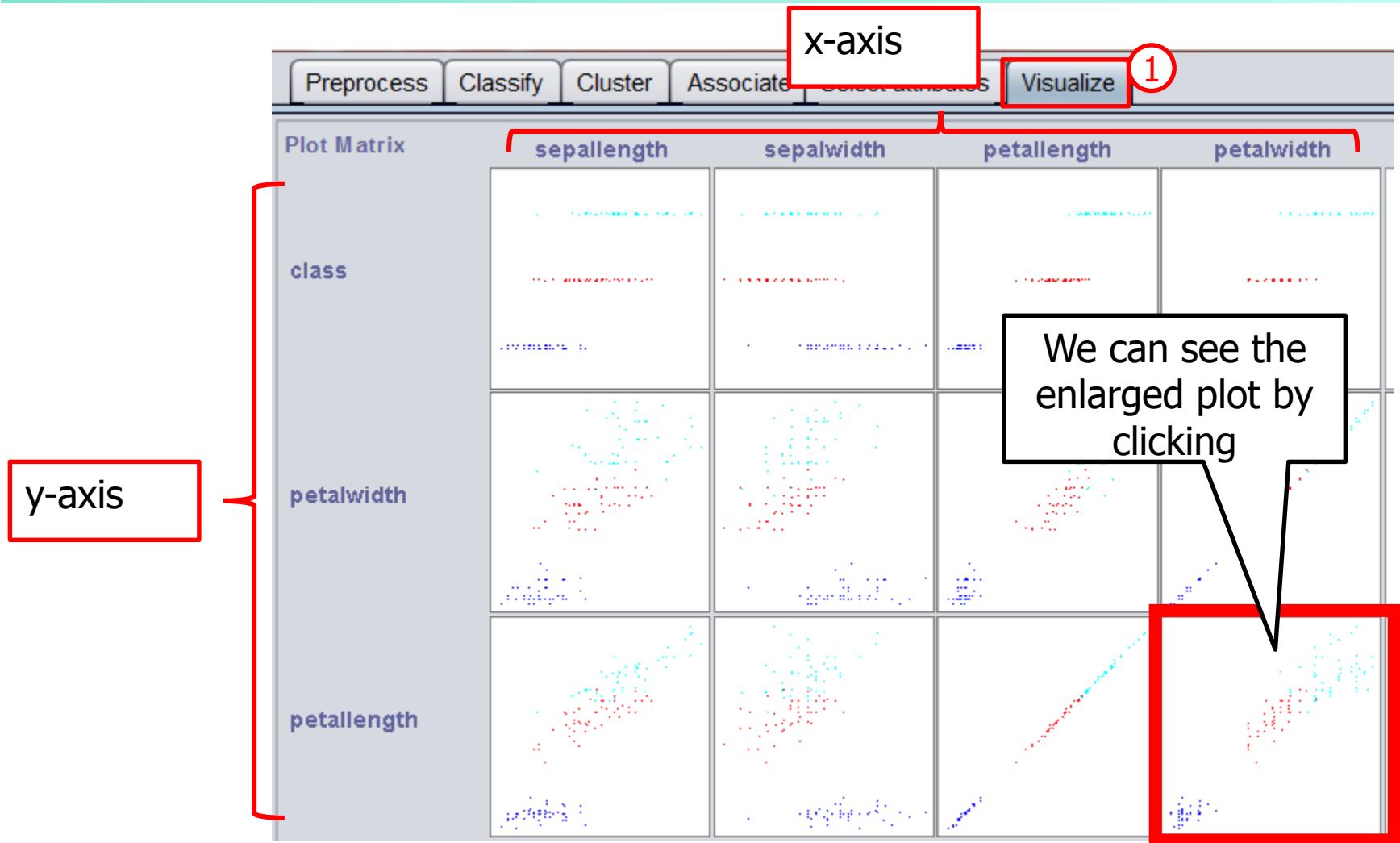
- C:\Program Files\Weka-3-8\data\iris.arff



iris.arff

- Classify the type of iris plants
- Attributes
 - Petal length, petal width, etc

Visualize the Data



Weka Explorer: Visualizing iris

X: petalwidth (Num)

Y: petallength (Num)

Colour: class (Nom)

Select Instance

Re...

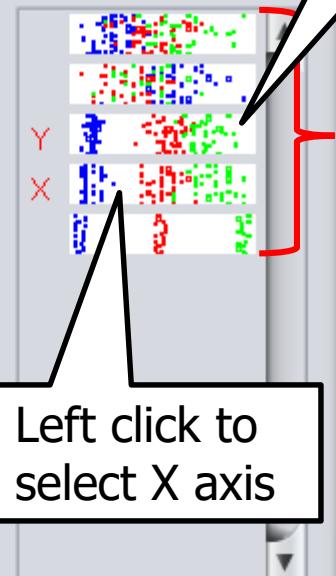
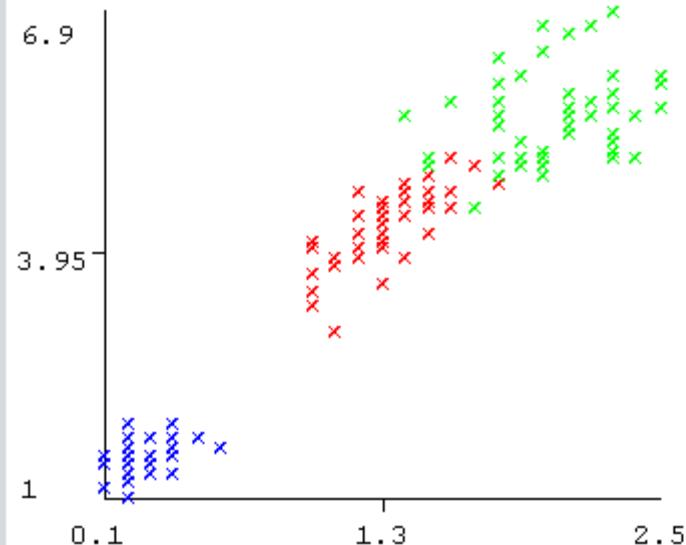
Cle...

Op...

Save

Jitter

Plot: iris



We can change
the axes

Right click to
select X axis

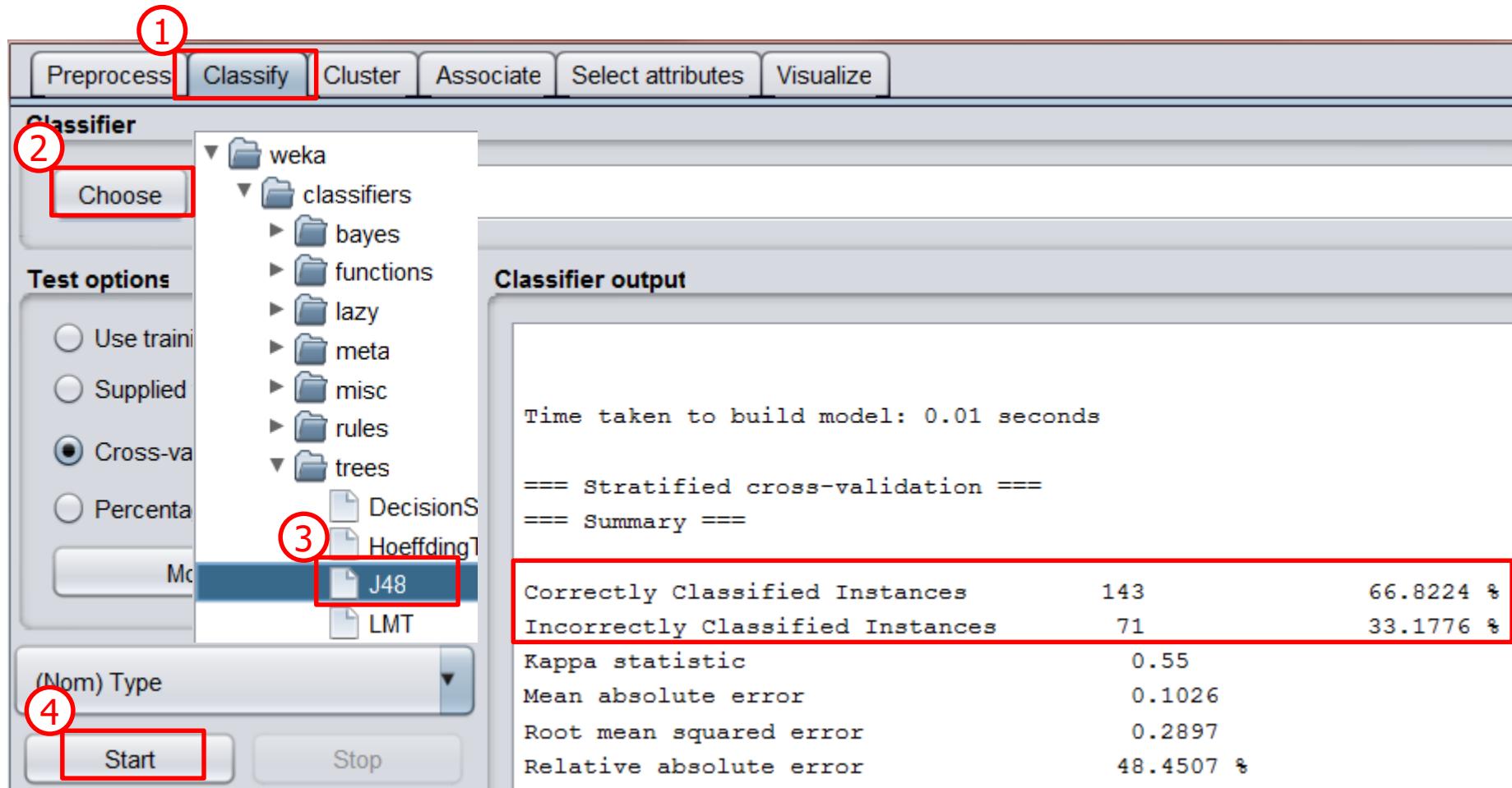
Represents each
attribute (point
distribution colored by
class for the attribute)

Left click to
select X axis

Visualizing your data

- Visualizing classification errors
 - Run J48
 - Visualize classifier errors (from Results list)
 - Plot predictedclass against class
 - Identify errors shown by confusion matrix

Run the J48



The screenshot shows the Weka interface with the following steps highlighted:

1. The "Classify" tab is selected.
2. The "Choose" button in the "Test options" panel is highlighted.
3. The "J48" classifier is selected in the "trees" folder.
4. The "Start" button is highlighted.

The "Classifier output" window displays the following information:

```
==== Detailed Accuracy By Class ====
TP Rate   FP Rate   P:
0.980     0.000     1
0.940     0.030     0
0.960     0.030     0
Weighted Avg.    0.960     0.020     0

==== Confusion Matrix ====
a  b  c  <-- classified as
49  1  0  |  a = Iris-setosa
0  47  3  |  b = Iris-versicolor
0  2  48  |  c = Iris-virginica
```

We can visualize the confusion matrix

[Preprocess](#)[Classify](#)[Cluster](#)[Associate](#)[Select attributes](#)[Visualize](#)

Classifier

[Choose](#)

J48 -C 0.25 -M 2

Test options

 Use training set Supplied test set [Set...](#) Cross-validation Folds [10](#) Percentage split % [90](#)[More options...](#)

Classifier output

==== Detailed Accuracy By Class ====

	TP Rate	FP Rate	P:
	0.980	0.000	1
	0.940	0.030	0
	0.960	0.030	0
Weighted Avg.	0.960	0.020	0

==== Confusion Matrix ====

(Nom) class
View in main window
View in separate window
Save result buffer
Delete result buffer(s)

ssified as
is-setosa
is-versicolor
is-virginica

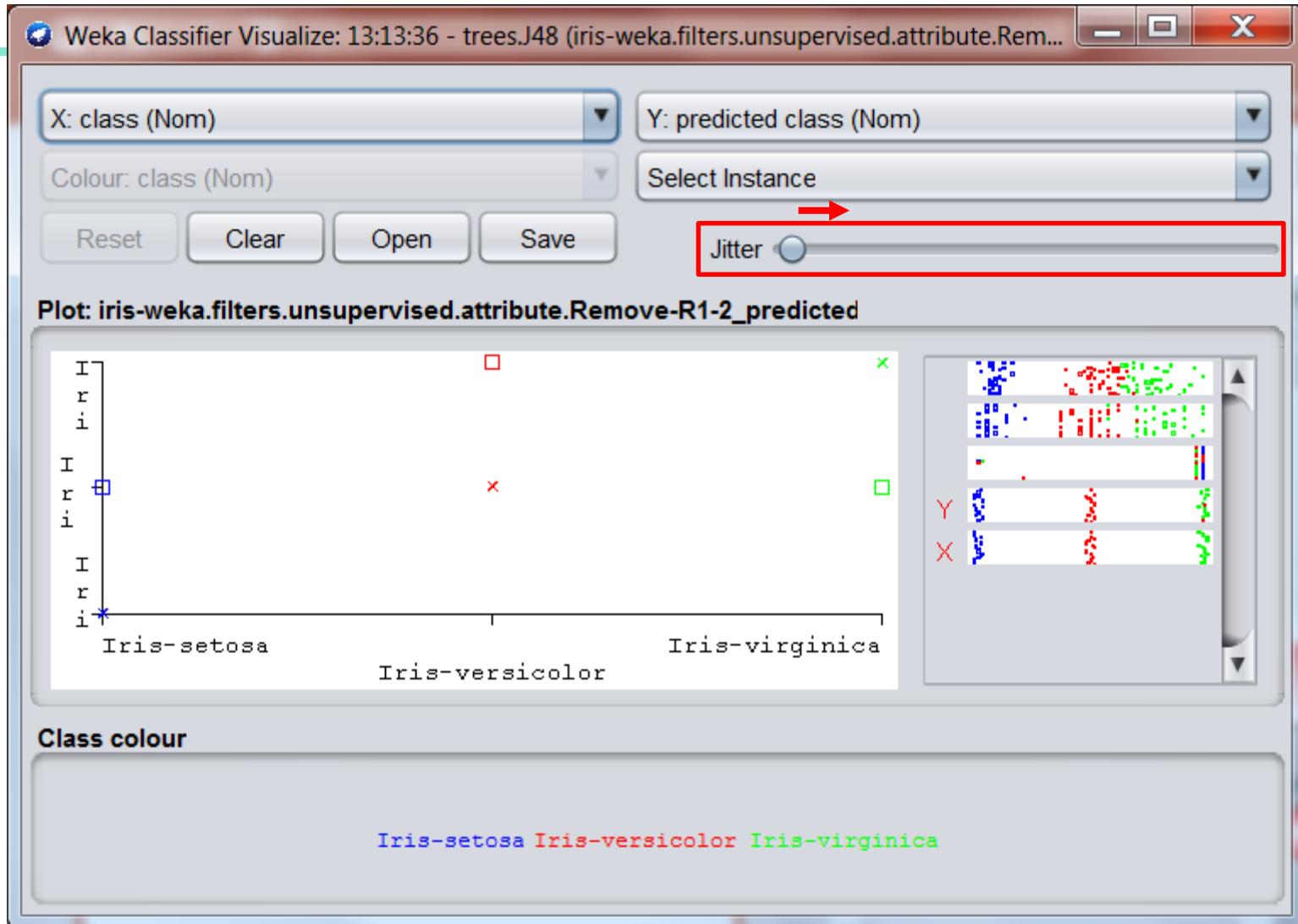
Result list (right-click for more options)

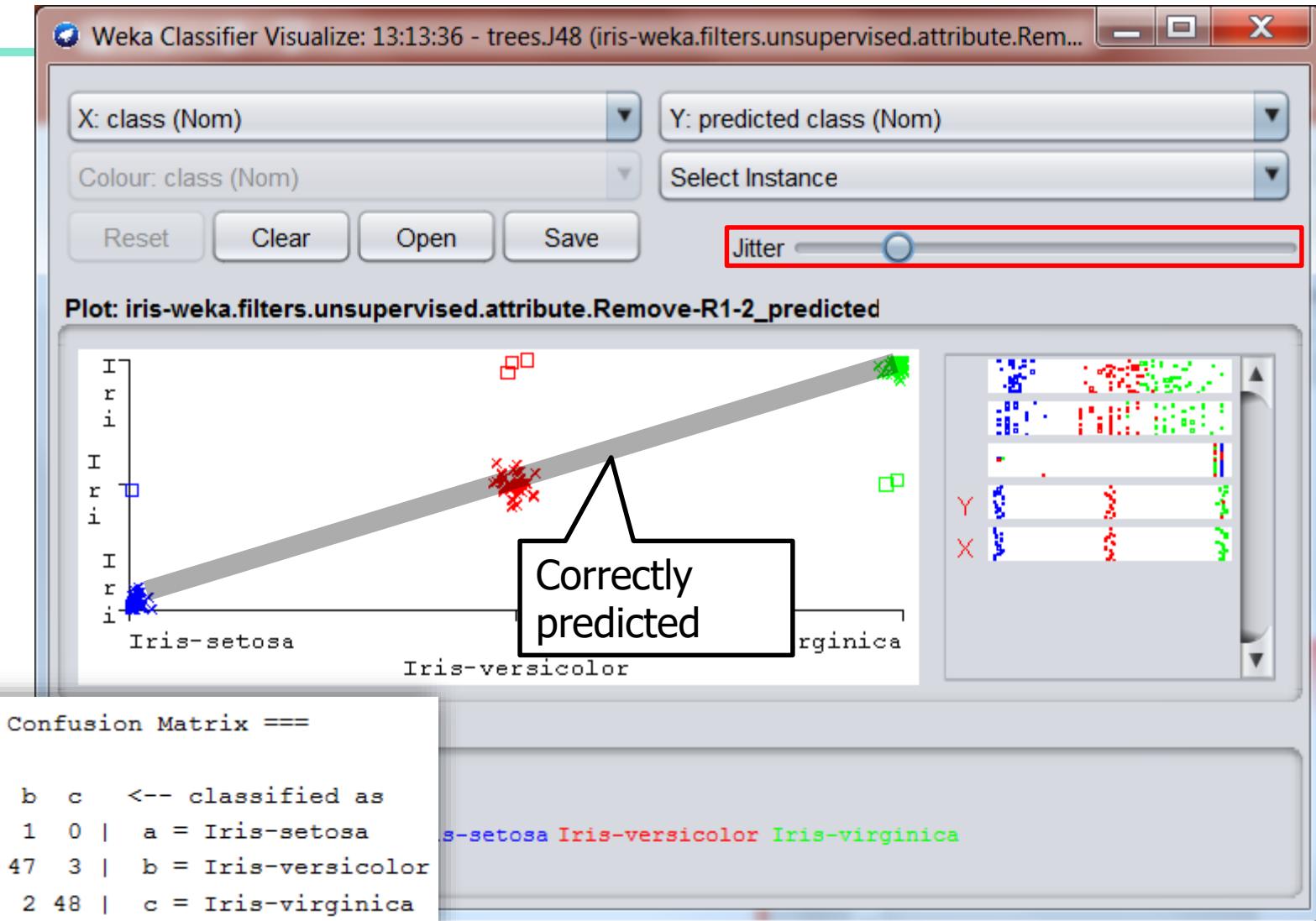
19:34:16 - trees.J48

19:34:21 - trees.J48

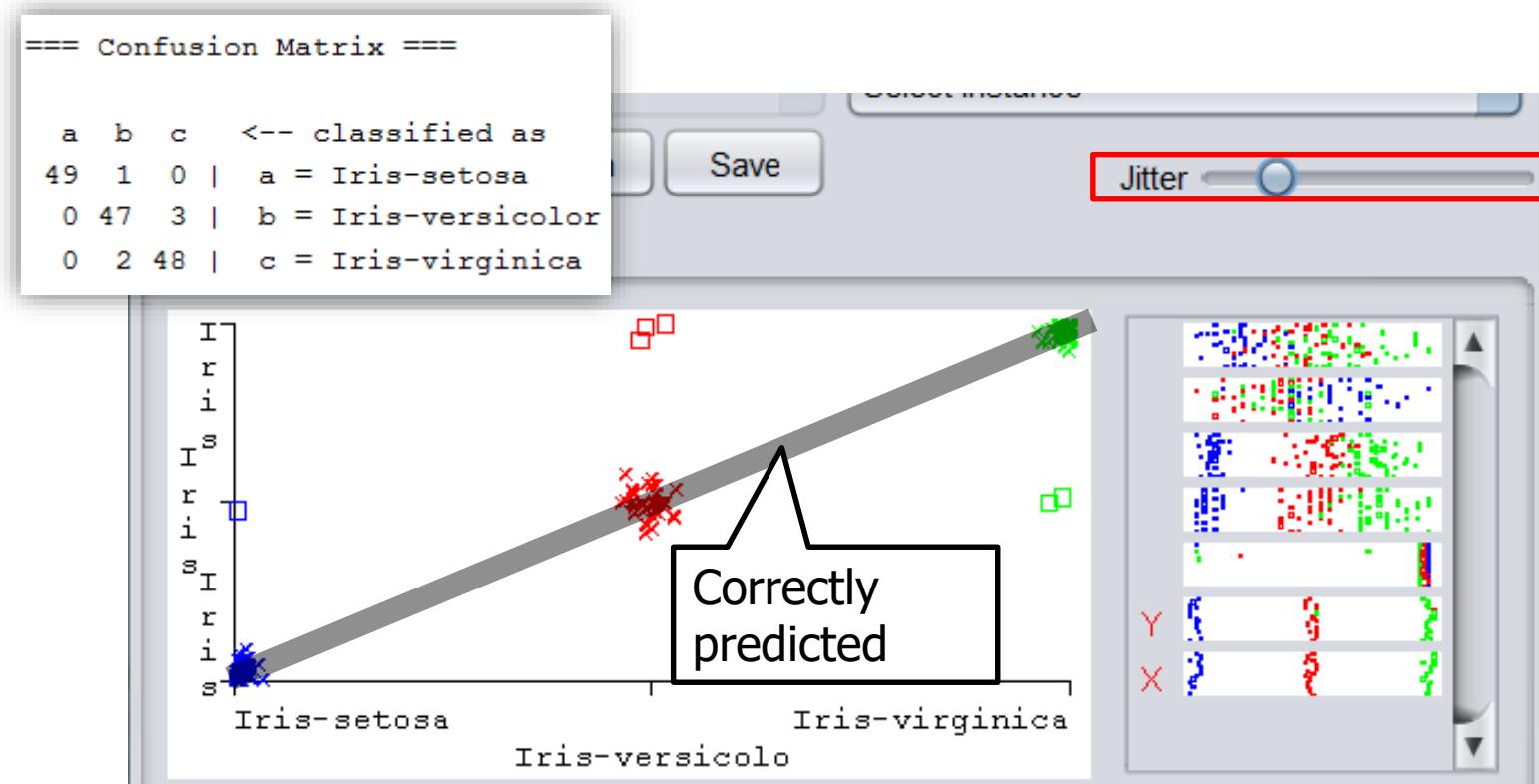
① Right click
②

- [View in main window](#)
- [View in separate window](#)
- [Save result buffer](#)
- [Delete result buffer\(s\)](#)
- [Load model](#)
- [Save model](#)
- [Re-evaluate model on current test set](#)
- [Re-apply this model's configuration](#)
- [Visualize classifier errors](#)
- [Visualize tree](#)
- [Visualize margin curve](#)





Visualizing Classification Errors



Simple algorithms often work very well!

- There are many kinds of simple structure
 - One attribute does all the work
 - Attributes contribute equally and independently
 - A decision tree that tests a few attributes Lessons
 - Calculate distance from training instances
 - Result depends on a linear combination of attributes

ZeroR

- Just answer the majority class of train data all the time

Outlook	Temp	Humidity	Wind	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

- Play=No: 5/14 tuples
- Play=Yes: 9/14 tuples
- Answer Play=Yes for every test data

OneR: One attribute does all the work

- Learn a 1-level “decision tree”
 - i.e., rules that all test one particular attribute
- Basic version
 - One branch for each value
 - Each branch assigns most frequent class
 - Error rate: proportion of instances that don’t belong to the majority class of their corresponding branch
 - Choose attribute with smallest error rate

OneR: One attribute does all the work

- For each attribute,
 - For each value of the attribute,
 - make a rule as follows:
 - count how often each class appears
 - find the most frequent class
 - make the rule assign that class to this attribute-value
 - Calculate the error rate of this attribute's rules
 - Choose the attribute with the smallest error rate

OneR: One attribute does all the work

Outlook	Temp	Humidity	Wind	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	
Wind	False → Yes	2/8	5/14
	True → No*	3/6	

* indicates a tie

OneR: One attribute does all the work

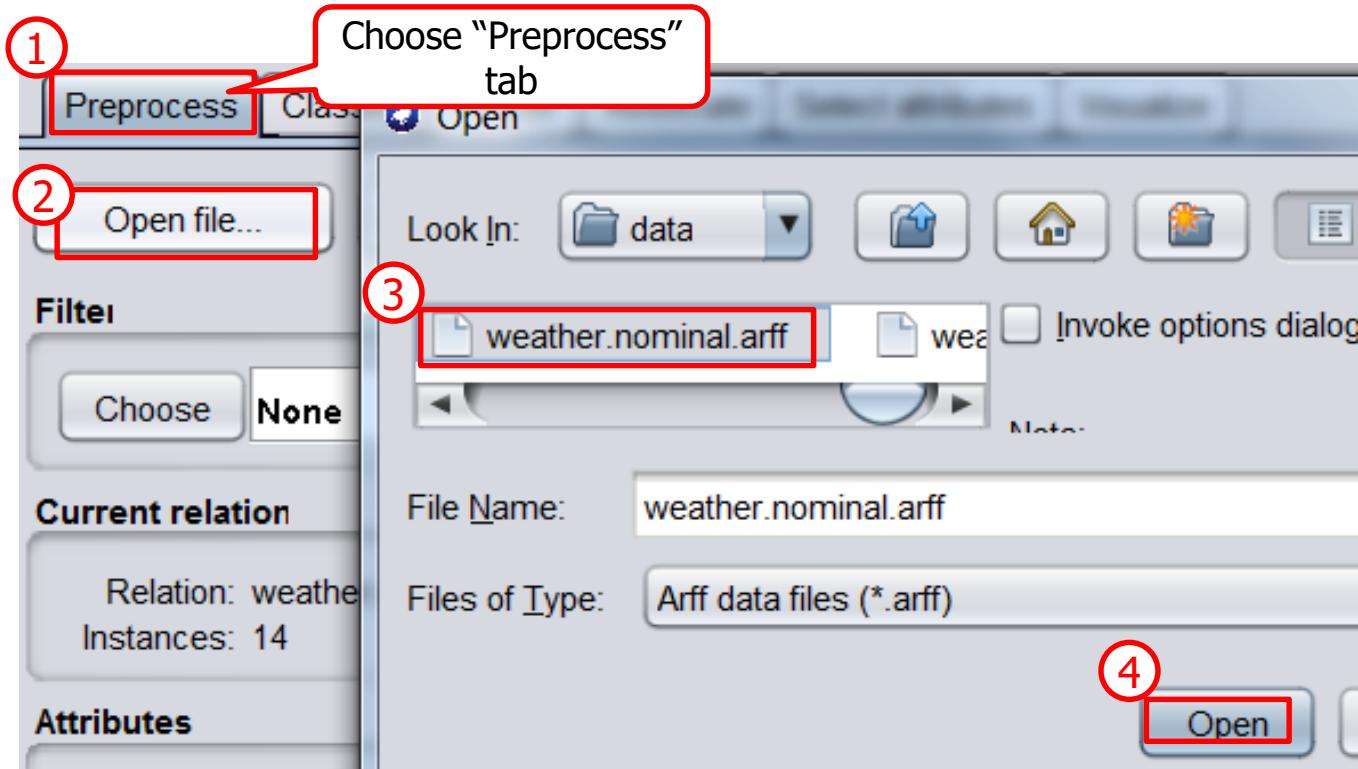
- Use OneR
 - Open file weather.nominal.arff
 - Choose OneR rule learner (rules>OneR)
 - Look at the rule (*note: Weka runs OneR 11 times*)

OneR: One attribute does all the work

- Incredibly simple method, described in 1993
- *"Very Simple Classification Rules Perform Well on Most Commonly Used Datasets"*
 - Experimental evaluation on 16 datasets
 - Used cross-validation
 - Simple rules often outperformed far more complex methods
- How can it work so well?
 - Some datasets really are simple
 - Some are so small/noisy/complex that nothing can be learned from them!

Open the Dataset

- C:\Program Files\Weka-3-8\data\weather.nominal.arff



Overfitting

- Any machine learning method may “overfit” the training data
 - by producing a classifier that fits the training data too tightly
- Works well on training data but not on independent test data
- Remember the “User classifier”? Imagine tediously putting a tiny circle around every single training data point
- Overfitting is a general problem
 - we illustrate it with OneR

Overfitting

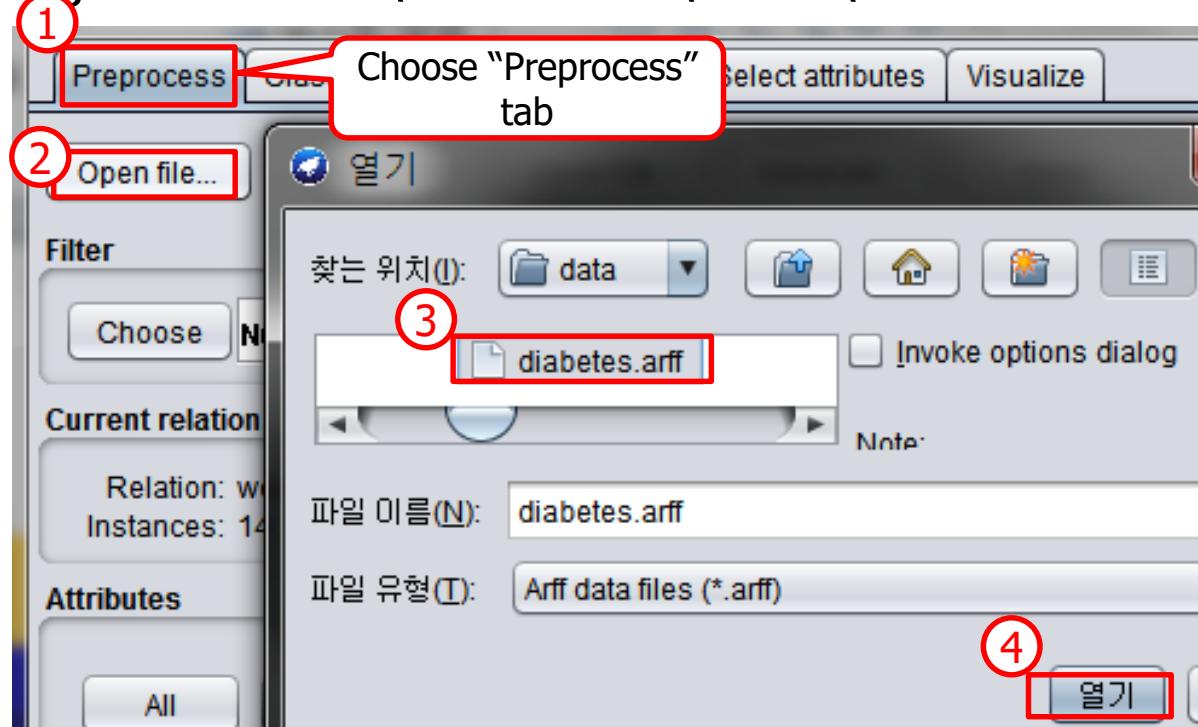
- Overfitting is a general phenomenon that plagues all ML methods
- One reason why you must never evaluate on the training set
 - Overfitting can occur more generally
- e.g) try many ML methods, choose the best for your data
 - you cannot expect to get the same performance on new test data
- Divide data into training, test, validation sets?

Overfitting

- Experiment with diabetes dataset
 - Open file **diabetes.arff**
 - Choose ZeroR rule learner (**rules>ZeroR**)
 - Use cross-validation: 65.1%
 - Choose OneR rule learner (**rules>OneR**)
 - Use cross-validation: 72.1%
 - Look at the rule (plas = plasma glucose concentration)
 - Change minBucketSize parameter to **1**: 54.9%
 - Evaluate on training set: 86.6%
 - Look at rule again

Open the Dataset

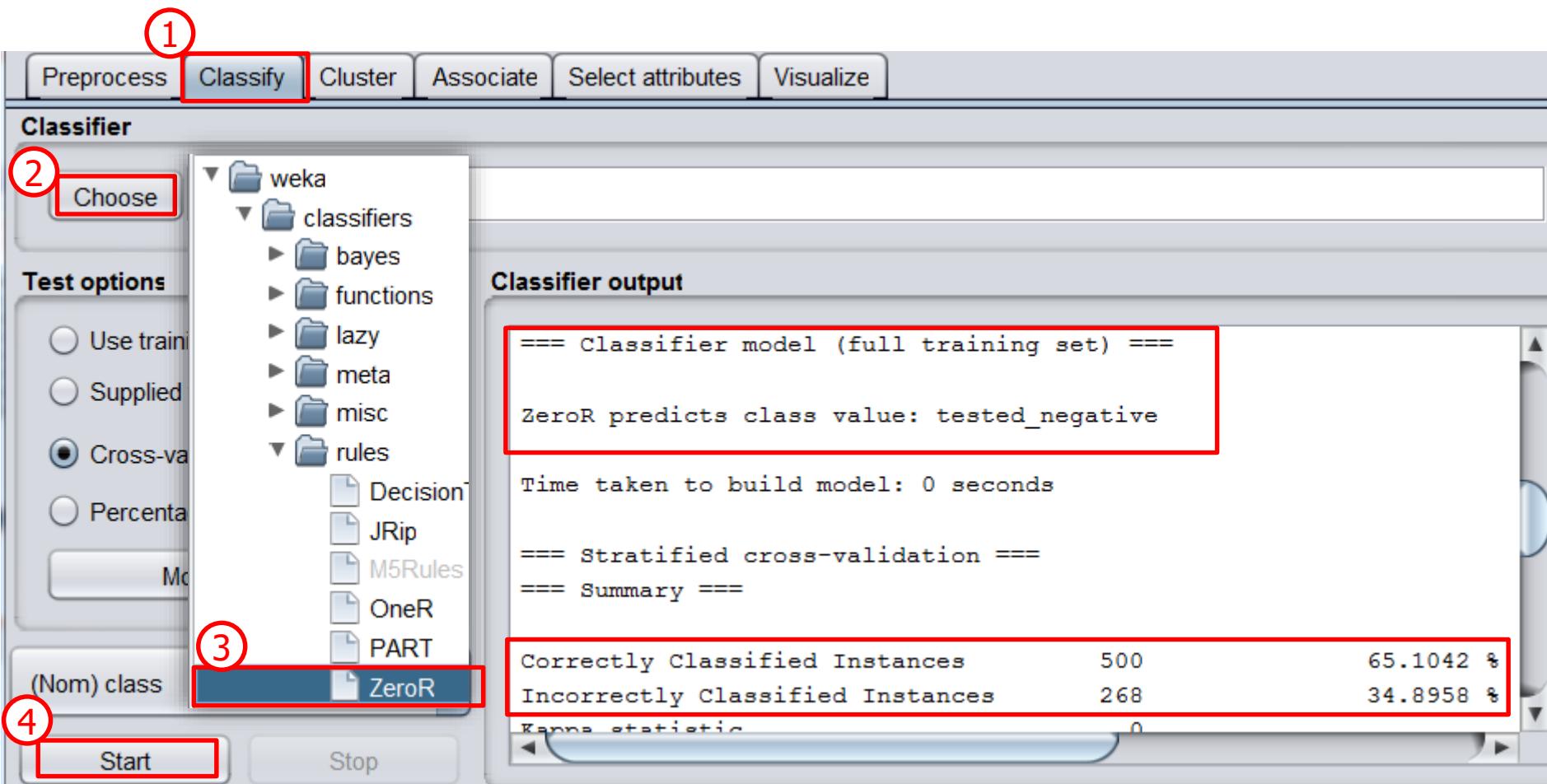
- C:\Program Files\Weka-3-8\data\diabetes.arff



diabetes.arff

- Classify whether the patient shows signs of diabetes
- Attributes are all numeric
 - Age, body mass index, etc

Run ZeroR



Classifier

1 Choose

Test options

- Use training data
- Supplied test set
- Cross-validation
- Percentage

Model

(Nom) class

3 Start Stop

Result list (right-click for options)

```
20:00:10 - rules.ZeroR  
20:16:25 - rules.ZeroR  
20:20:33 - rules.OneR  
20:23:06 - rules.OneR  
20:28:54 - rules.ZeroR  
20:30:25 - rules.OneR  
20:30:36 - rules.ZeroR  
20:31:48 - rules.OneR
```

weka

classifiers

- bayes
- functions
- lazy
- meta
- misc
- rules

2 OneR

Classifier output

Run OneR with default settings
(minBucketSize=6)

minBucketSize: the minimum number instance in a leaf node

The selected attribute

plas:

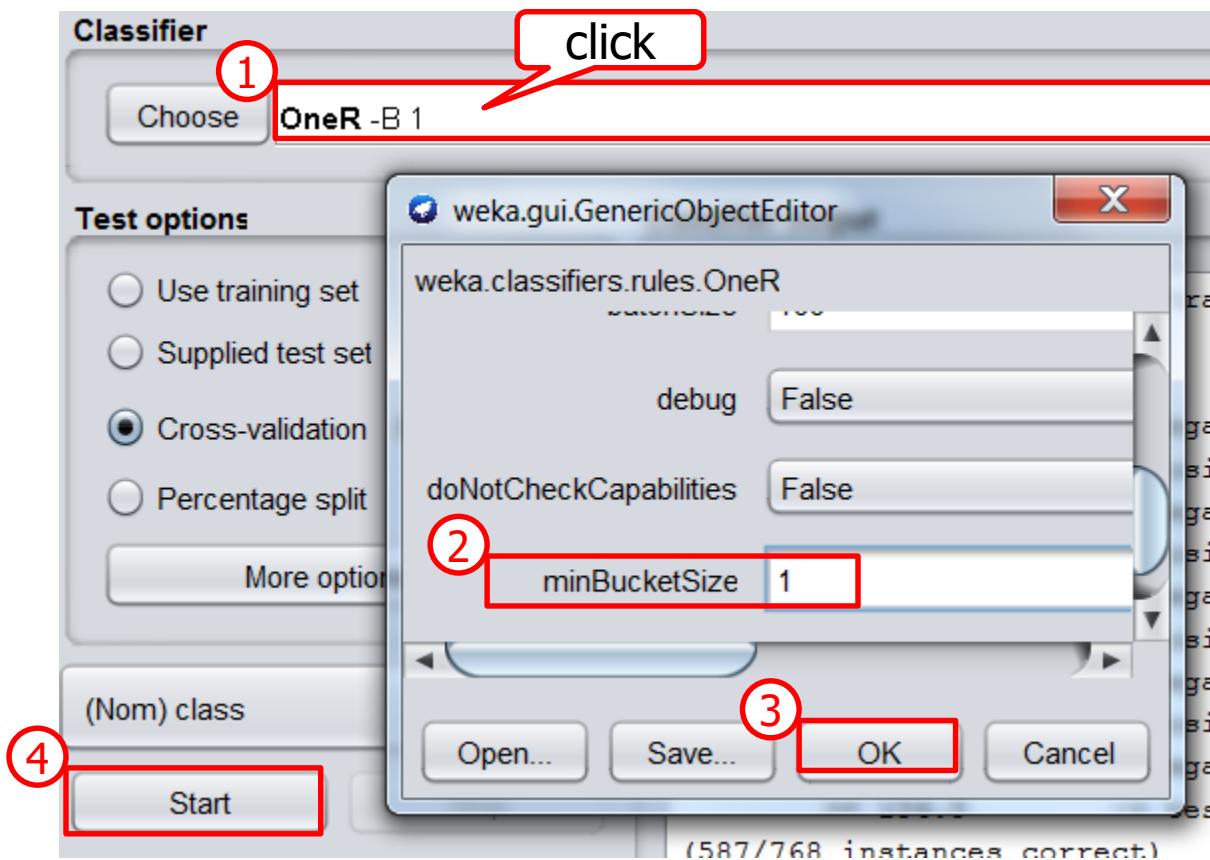
```
==> tested_negative  
< 114.5 -> tested_positive  
< 115.5 -> tested_negative  
< 127.5 -> tested_positive  
< 128.5 -> tested_negative  
< 133.5 -> tested_positive  
< 135.5 -> tested_negative  
< 143.5 -> tested_positive  
< 152.5 -> tested_negative  
< 154.5 -> tested_positive  
>= 154.5 -> tested positive
```

(587/768 instances correct)

More complex model

```
Time taken to build model: 0 seconds  
  
==== Stratified cross-validation ====  
==== Summary ====  
  
Correctly Classified Instances 549 71.4844 %  
Incorrectly Classified Instances 219 28.5156 %
```

Change minBucketSize



Classifier output

```
==== Classifier model (full training set) ====
```

```
pedi:
```

```
< 0.1265      -> tested_negative  
< 0.1275      -> tested_positive  
< 0.1285      -> tested_negative  
< 0.1295      -> tested_positive  
< 0.1345      -> tested_negative  
< 0.1355      -> tested_positive  
< 0.1405      -> tested_negative  
< 0.1415      -> tested_positive  
< 0.1625      -> tested_negative  
< 0.1635      -> tested_positive  
< 0.1645  
< 0.1655  
< 0.1775  
< 0.1785  
< 0.195 -> tested_negative  
< 0.1965  
< 0.1985  
< 0.1995  
< 0.2045  
< 0.2055  
< 0.211 -> tested_positive  
< 0.2135  
< 0.2195  
< 0.2205
```

Too complex model overfit to the training data

Classifier output

```
Time taken to build model: 0.02 seconds
```

```
==== Stratified cross-validation ====
```

```
==== Summary ====
```

Correctly Classified Instances	439	57.1615 %
Incorrectly Classified Instances	329	42.8385 %
Kappa statistic	0.0207	
Mean absolute error	0.4284	
Root mean squared error	0.6545	

Accuracy on the Training Data

Classifier

Choose OneR -B 1

Test options

Use training set (1)

Supplied test set Set...

Cross-validation Folds 10

Percentage split % 90

More options...

(Nom) class

Start (2) Stop

Classifier output

```
==== Evaluation on training set ====
Time taken to test model
==== Summary ====
Correctly Classified Instances 672 87.5 %
Incorrectly Classified Instances 96 12.5 %
Kappa statistic 0.7055
Mean absolute error 0.125
Root mean squared error 0.3536
```

The model is overfit to the training data

Category	Value	Percentage	Unit
Correctly Classified Instances	672	87.5	%
Incorrectly Classified Instances	96	12.5	%

Pruning parameters

- minNumObj (default value 2)
 - Not splitting the node to avoid smaller number of objects in the node than minNumObj
- confidenceFactor (default value 0.25)
 - Build full tree and then work back from the leaves, applying a statistical test at each stage
 - Compares (1) the weighted error of each child node versus (2) the misclassification error if the child nodes were deleted and the decision node were assigned the class label of the majority class
- subtreeRaising (default true)
 - Pruning an interior node and raising the subtree beneath it up one level

Pruning decision trees

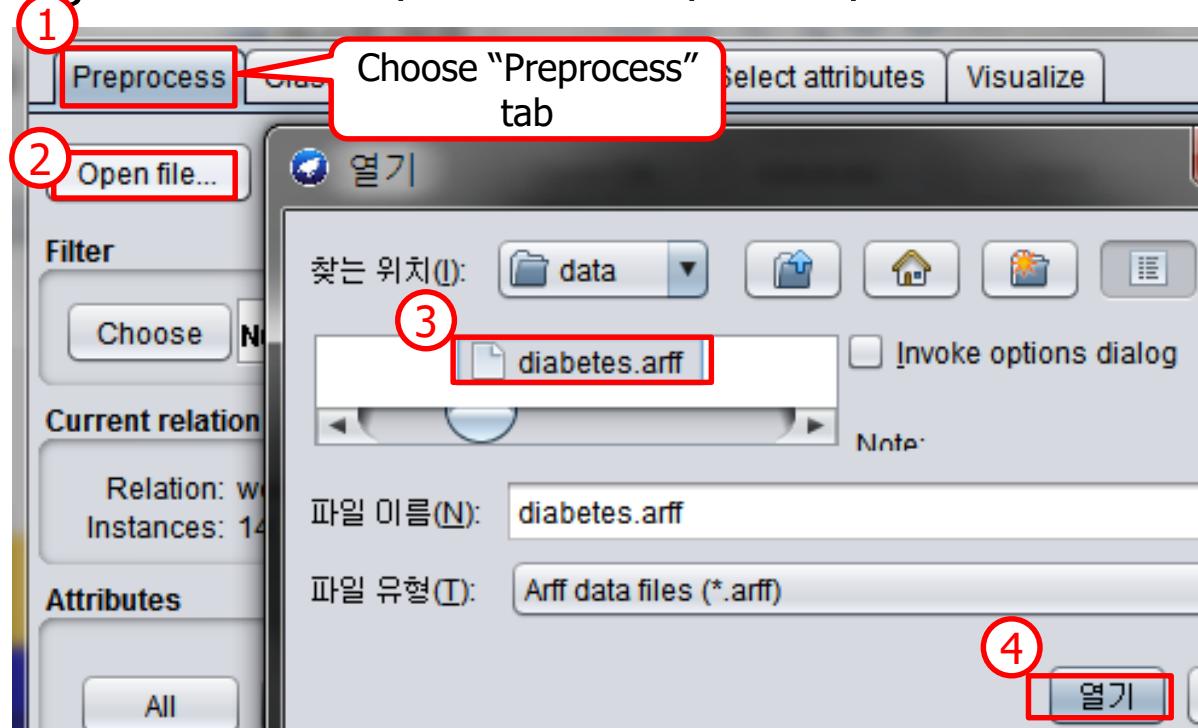
- Over-fitting - Sometimes simplifying a decision tree gives better results
 - Open file **diabetes.arff**
 - Choose J48 decision tree learner (**trees>J48**)
 - Prunes by default: 73.8% accuracy, tree has 20 leaves, 39 nodes
 - Turn off pruning: 72.7% 22 leaves, 43 nodes
 - Extreme example: **breast-cancer.arff**
 - Default (pruned): **75.5%** accuracy, tree has **4 leaves**, 6 nodes
 - Unpruned: **69.6% 152 leaves**, 179 nodes



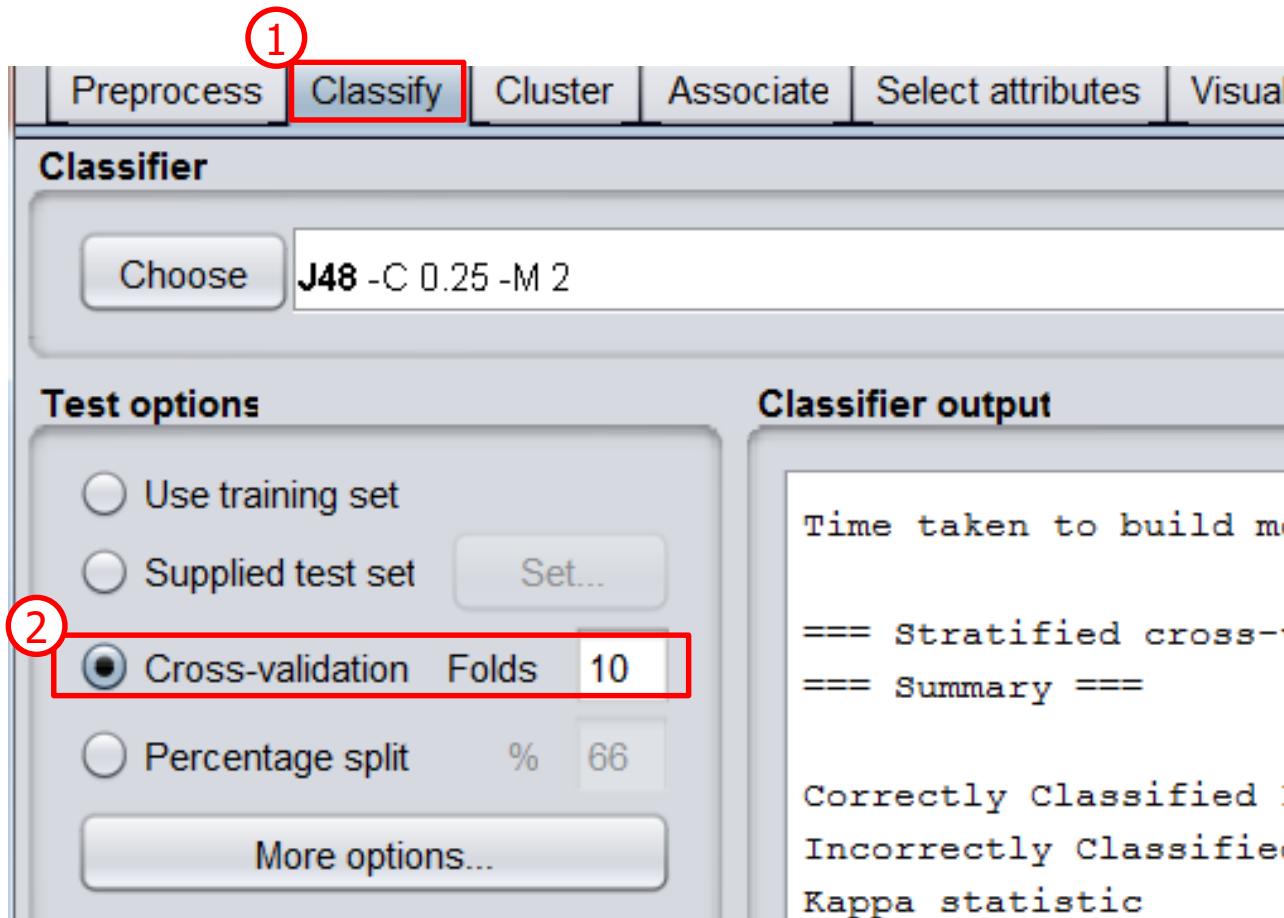
Accurate even if it has less leaves

Open the Dataset

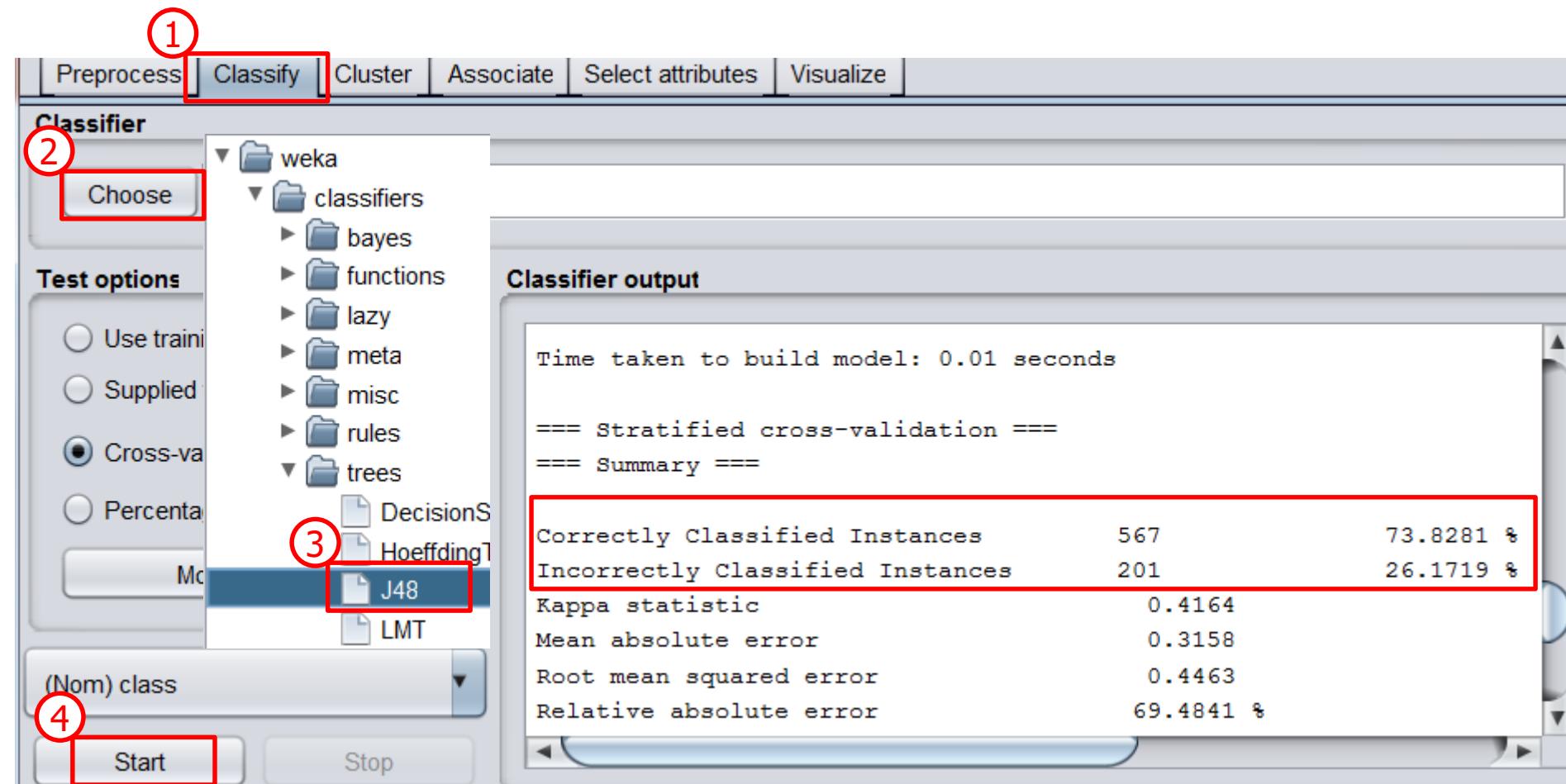
- C:\Program Files\Weka-3-8\data\diabetes.arff



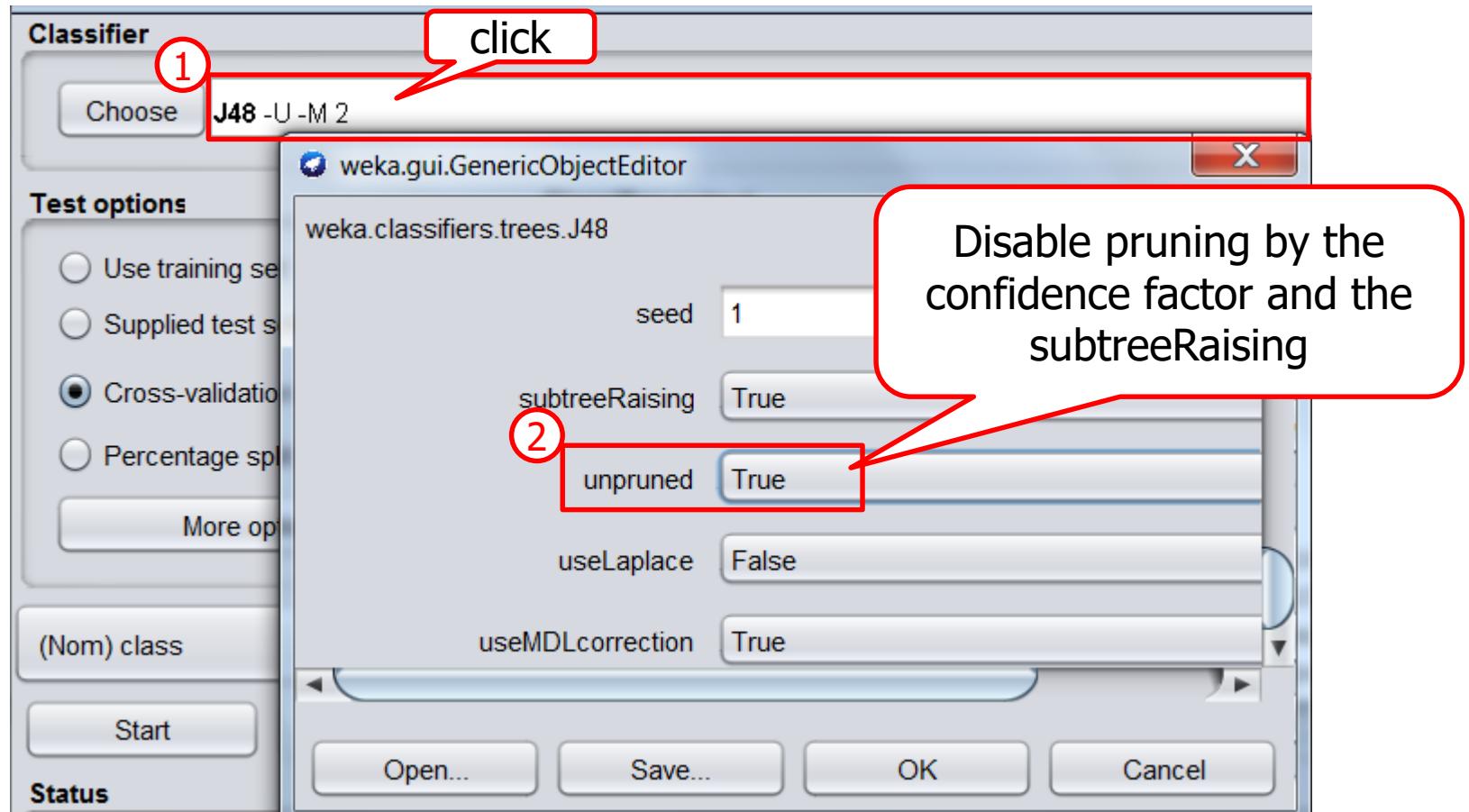
Set the Test Options



Run J48 with Pruning (Default)



Disable the Pruning



Run J48 without Pruning

Classifier

Choose **J48 -U -M 2**

Test options

Use training set

Supplied test set Set...

Cross-validation Folds 10

Percentage split % 66

More options...

(Nom) class 1

Start **Stop**

Classifier output

```
Size of the tree : 10
Time taken to build model: 0.01 seconds
==== Stratified cross-validation ====
==== Summary ====
Correctly Classified Instances      558          72.6563 %
Incorrectly Classified Instances   210          27.3438 %
Kappa statistic                      0.3992
Mean absolute error                  0.3146
```

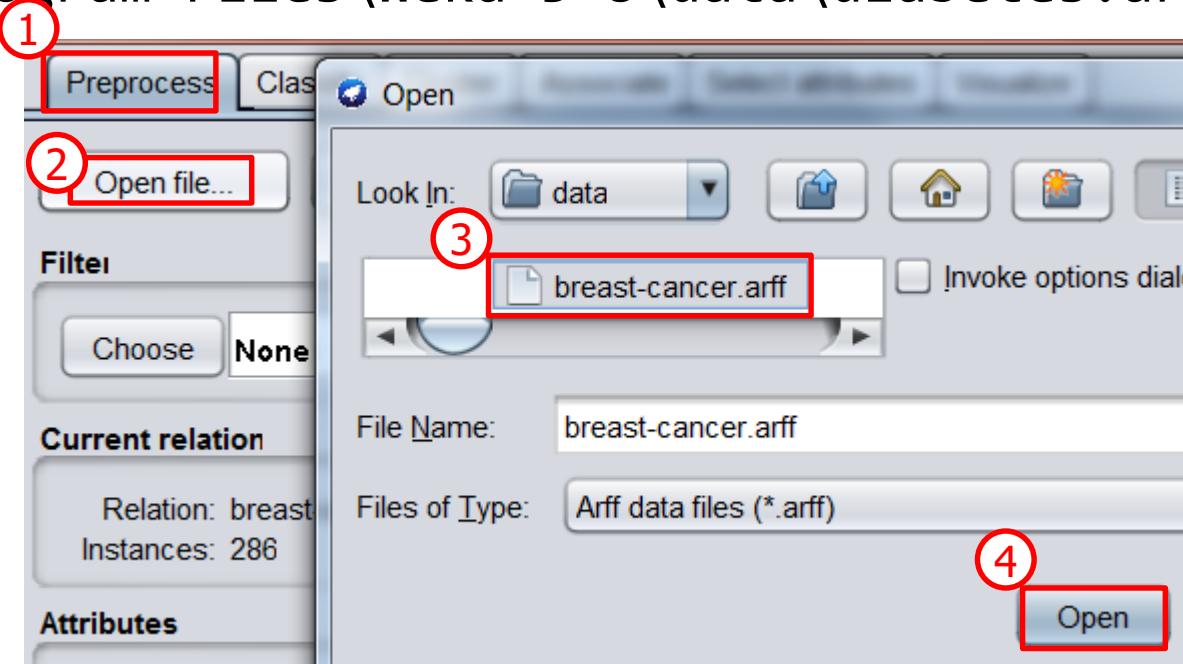
The classifier output window displays the results of a 10-fold cross-validation run. The correctly classified instances are 558 (72.6563%), and the incorrectly classified instances are 210 (27.3438%). The Kappa statistic is 0.3992, and the Mean absolute error is 0.3146.

Extreme Example: breast-cancer.arff

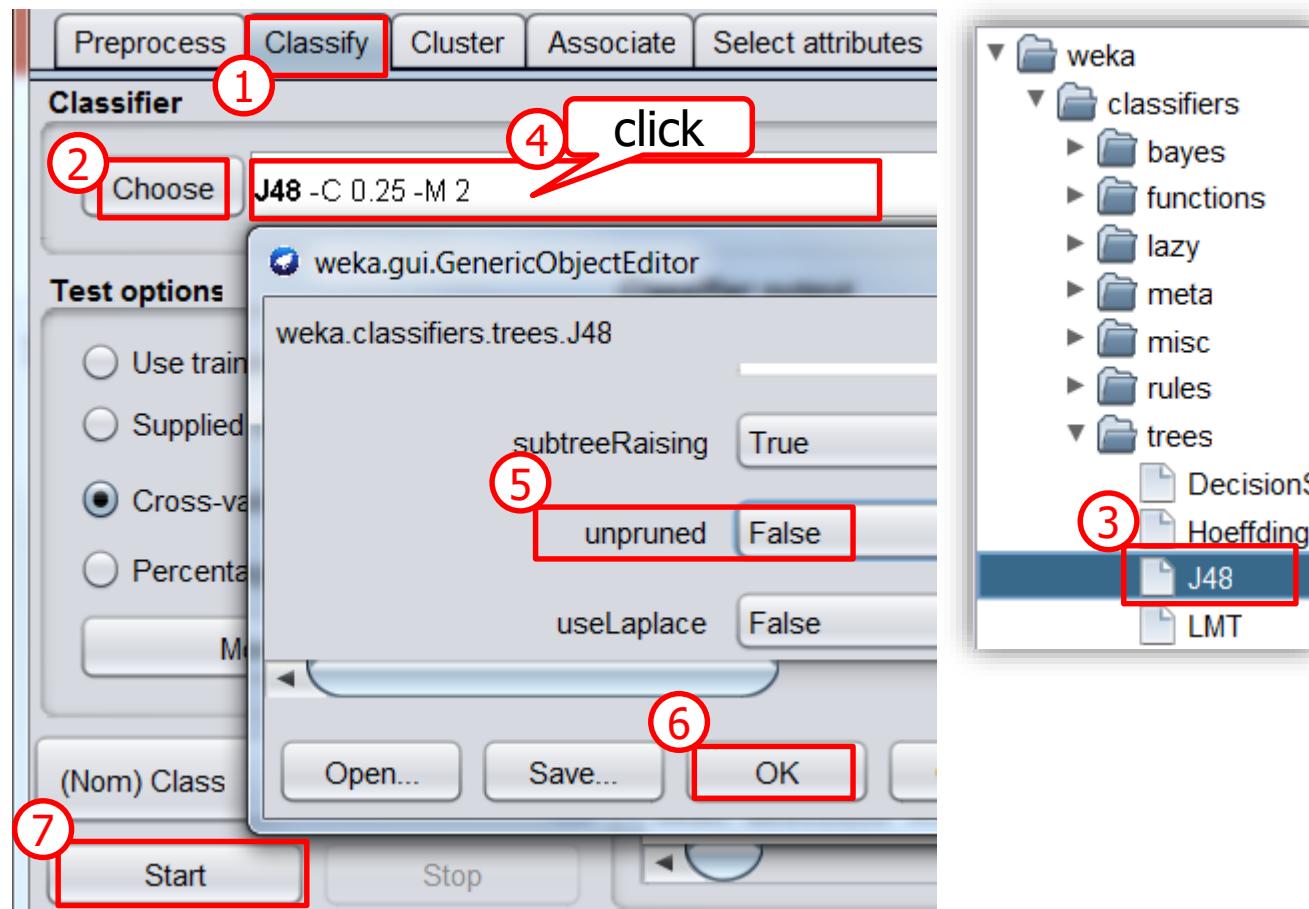
- Predicts breast cancer recurrence
 - Class: no-recurrence-events, recurrence-events
- Attributes: age, tumor-size, etc

Open the Dataset

- C:\Program Files\Weka-3-8\data\diabetes.arff



Run J48 with Pruning



Run J48 with Pruning

```
Classifier output
TIME TAKEN TO BUILD MODEL: 0.01 SECONDS

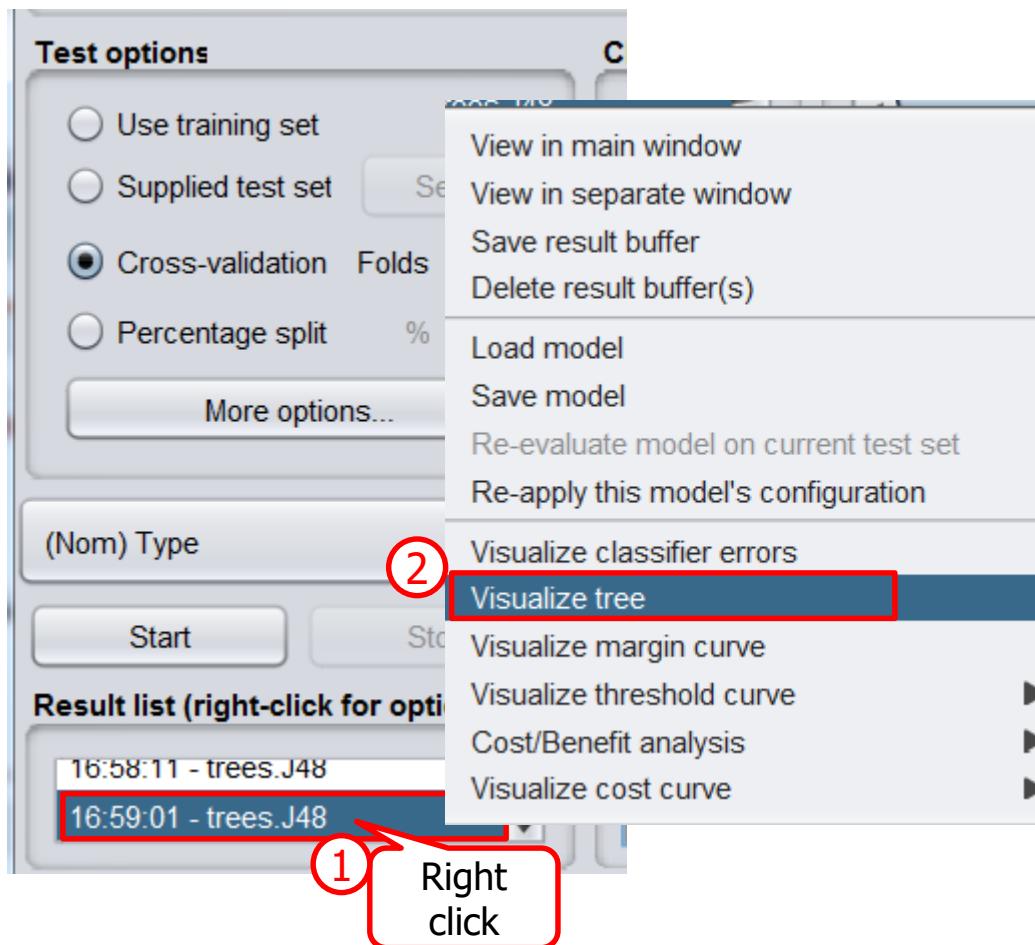
==== Stratified cross-validation ====
==== Summary ====



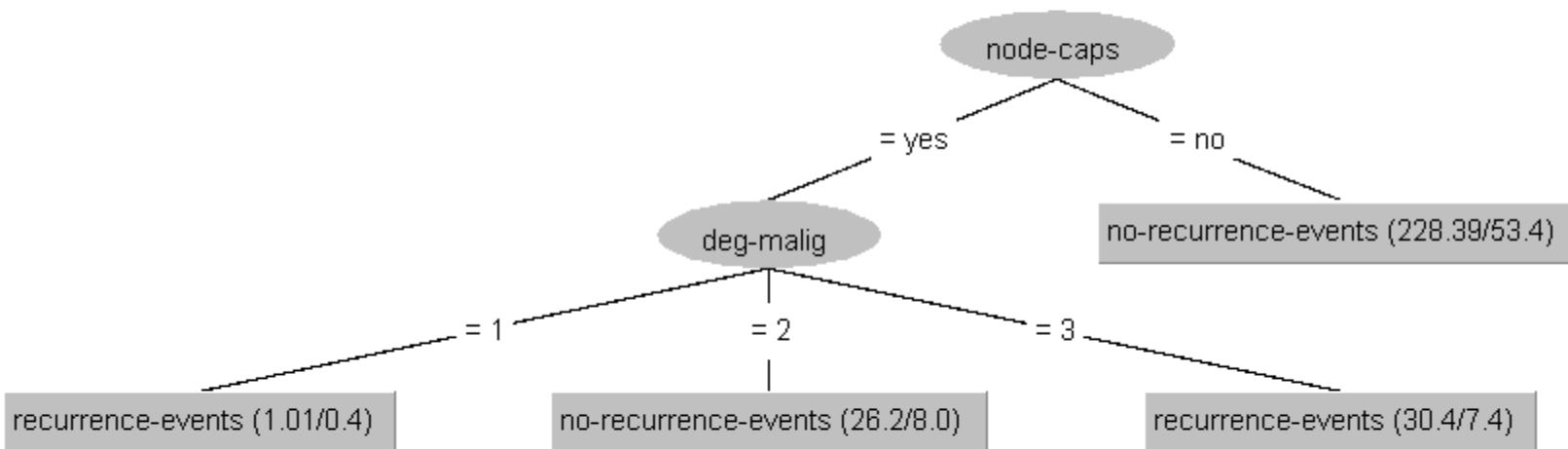
|                                  |           |           |
|----------------------------------|-----------|-----------|
| Correctly Classified Instances   | 216       | 75.5245 % |
| Incorrectly Classified Instances | 70        | 24.4755 % |
| Kappa statistic                  | 0.2826    |           |
| Mean absolute error              | 0.3676    |           |
| Root mean squared error          | 0.4324    |           |
| Relative absolute error          | 87.8635 % |           |
| Root relative squared error      | 94.6093 % |           |
| Total Number of Instances        | 286       |           |


```

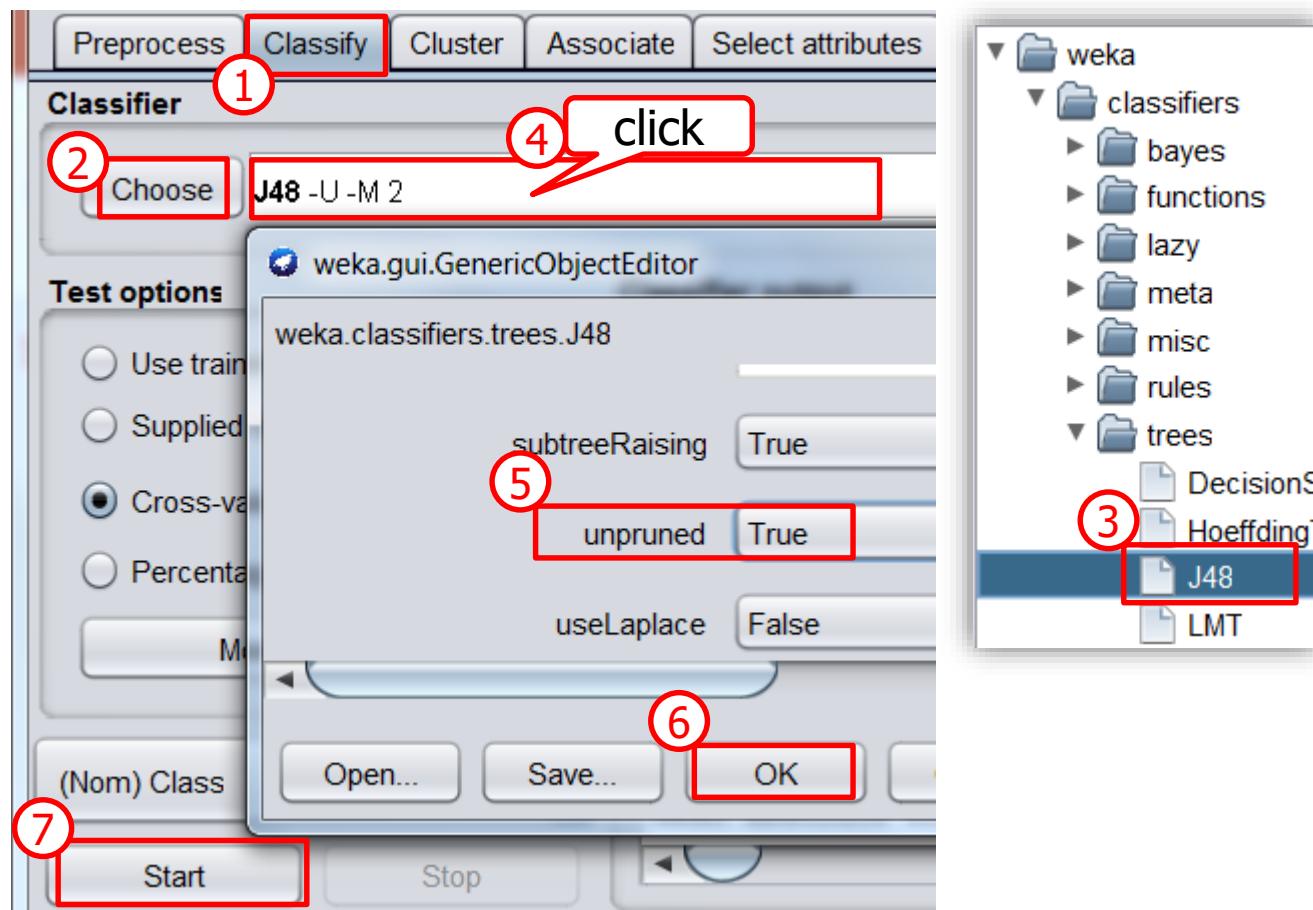
Run J48 with Pruning



Tree View



Run J48 without Pruning



Run J48 without Pruning

```
Classifier output

TIME TAKEN TO BUILD MODEL: 0 SECONDS

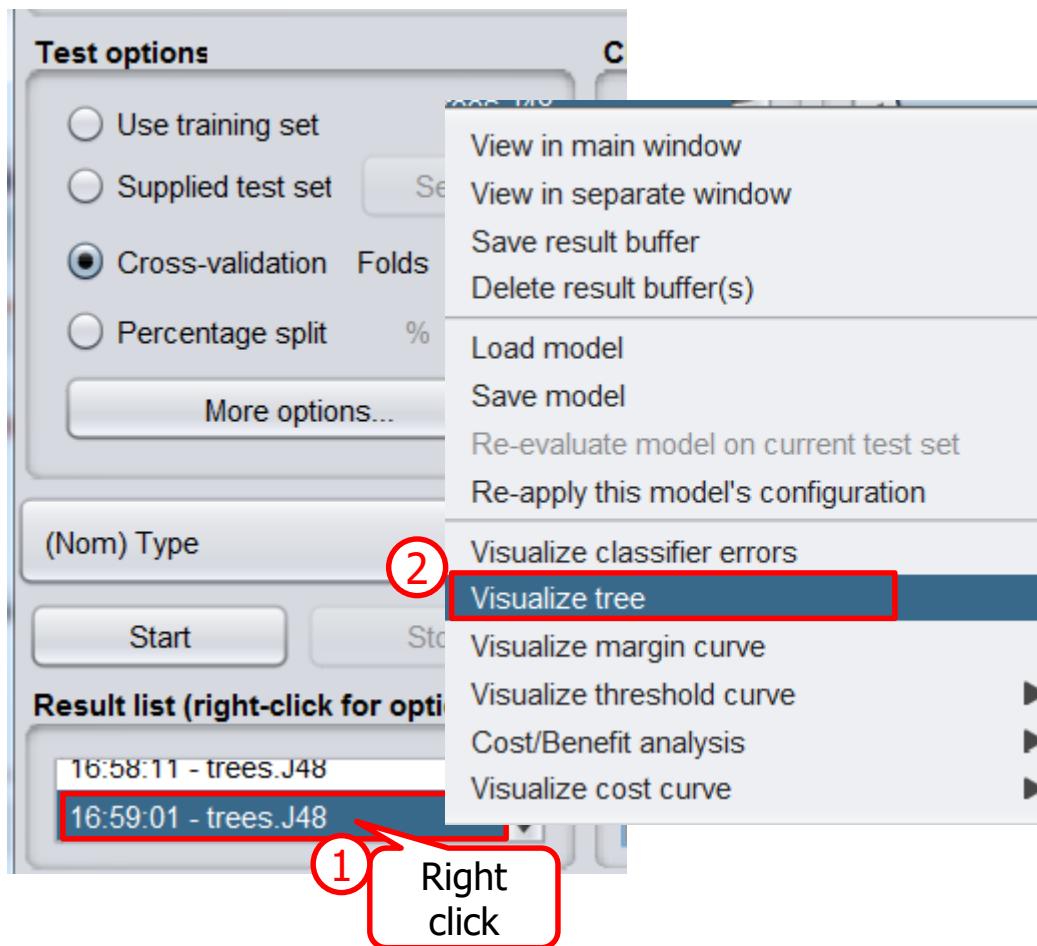
==== Stratified cross-validation ====
==== Summary ====



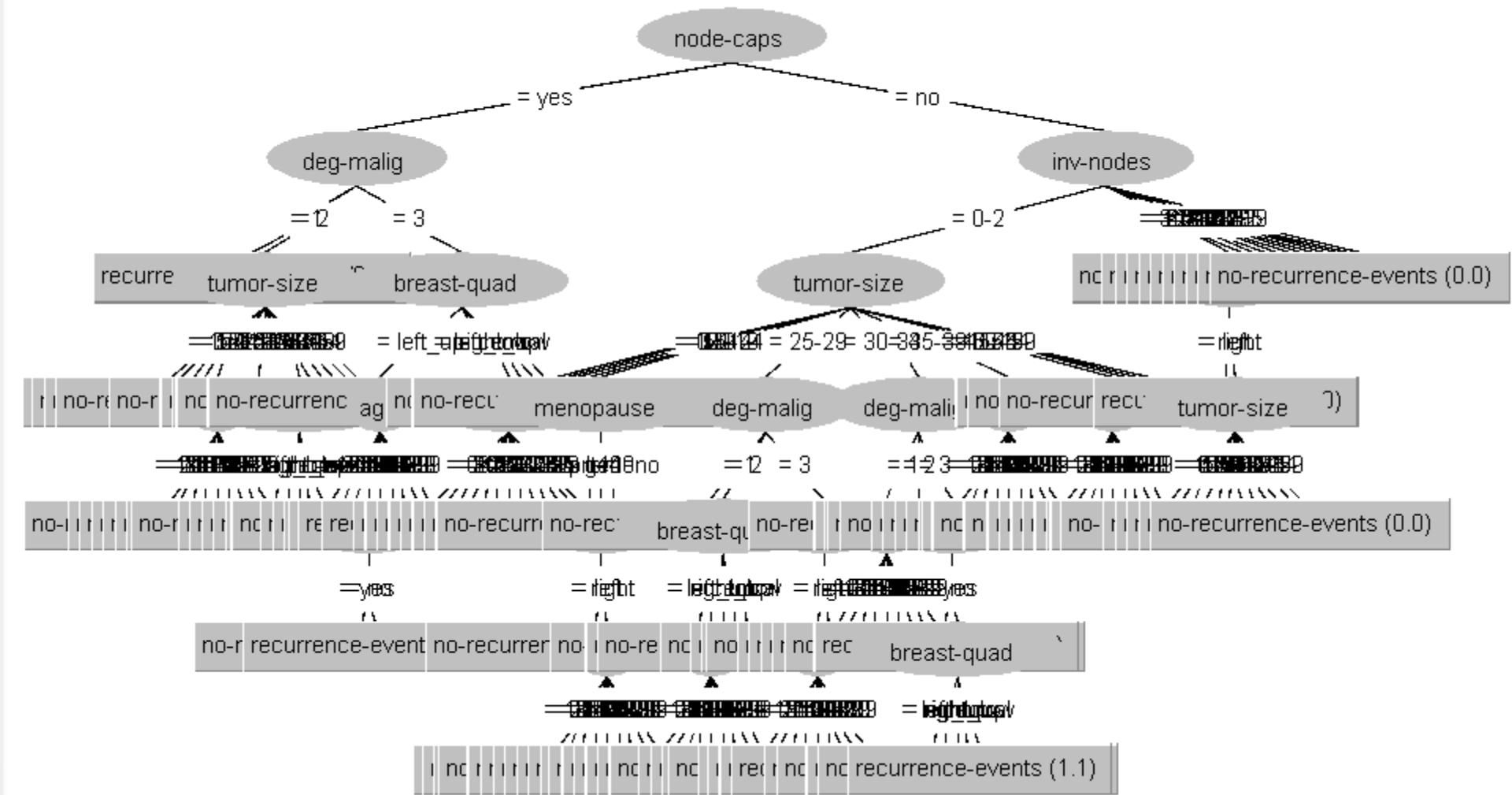
|                                  |            |           |
|----------------------------------|------------|-----------|
| Correctly Classified Instances   | 199        | 69.5804 % |
| Incorrectly Classified Instances | 87         | 30.4196 % |
| Kappa statistic                  | 0.2043     |           |
| Mean absolute error              | 0.3478     |           |
| Root mean squared error          | 0.5143     |           |
| Relative absolute error          | 83.1224 %  |           |
| Root relative squared error      | 112.5118 % |           |
| Total Number of Instances        | 286        |           |

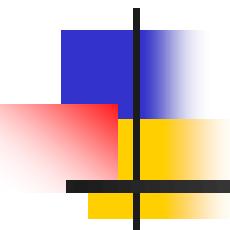

```

Run J48 without Pruning



Tree View





Python – Decision Classifier

Import Libraries

```
from sklearn import tree
import graphviz
from sklearn.preprocessing import OneHotEncoder
import numpy as np
import pandas as pd
```

Open the Dataset

```
df = pd.read_csv('weather.nominal.csv')  
df
```

	outlook	temperature	humidity	windy	play
0	sunny	hot	high	False	no
1	sunny	hot	high	True	no
2	overcast	hot	high	False	yes
3	rainy	mild	high	False	yes
4	rainy	cool	normal	False	yes
5	rainy	cool	normal	True	no
6	overcast	cool	normal	True	yes
7	sunny	mild	high	False	no
8	sunny	cool	normal	False	yes
9	rainy	mild	normal	False	yes
10	sunny	mild	normal	True	yes

Data Preprocessing

- Sklearn does not provide the decision tree with categorical data
 - Should convert the categorical data to numerical data
 - We will use one hot encoding in sklearn

	Weather		Sunny	Overcast	Rainy
1	Sunny	1	1	0	0
2	Overcast	2	0	1	0
3	Rainy	3	0	0	1

Data Preprocessing

Features

Class label

	outlook	temperature	humidity	windy	play
0	sunny	hot	high	False	no
1	sunny	hot	high	True	no
2	overcast	hot	high	False	yes

```
print(X)
```

```
[['sunny' 'hot' 'high' False]
['sunny' 'hot' 'high' True]
['overcast' 'hot' 'high' False]
['rainy' 'mild' 'high' False]
['rainy' 'cool' 'normal' False]
['rainy' 'cool' 'normal' True]
['overcast' 'cool' 'normal' True]
['sunny' 'mild' 'high' False]
['sunny' 'cool' 'normal' False]
['rainy' 'mild' 'normal' False]
['sunny' 'mild' 'normal' True]
['overcast' 'mild' 'high' True]
['overcast' 'hot' 'normal' False]
['rainy' 'mild' 'high' True]]
```

```
print(y)
```

```
['no' 'no' 'yes' 'yes' 'yes' 'no' 'yes' 'yes' 'yes' 'yes' 'yes'
 'no']
```

Ignore the last element

```
X = df.values[:, :-1]
y = df.values[:, -1]
```

Only the last element

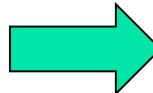
Data Preprocessing

- X is transformed by one hot encoding

```
enc = OneHotEncoder(handle_unknown='ignore')
enc.fit(X)
en_X = enc.transform(X).toarray()
en_X
```

print(X)

```
[['sunny' 'hot' 'high' False]
 ['sunny' 'hot' 'high' True]
 ['overcast' 'hot' 'high' False]
 ['rainy' 'mild' 'high' False]
 ['rainy' 'cool' 'normal' False]
 ['rainy' 'cool' 'normal' True]
 ['overcast' 'cool' 'normal' True]
 ['sunny' 'mild' 'high' False]
 ['sunny' 'cool' 'normal' False]
 ['rainy' 'mild' 'normal' False]
 ['sunny' 'mild' 'normal' True]
 ['overcast' 'mild' 'high' True]
 ['overcast' 'hot' 'normal' False]
 ['rainy' 'mild' 'high' True]]
```



print(en_X)

```
[[0. 0. 1. 0. 1. 0. 1. 0. 1. 0.]
 [0. 0. 1. 0. 1. 0. 1. 0. 0. 1.]
 [1. 0. 0. 0. 1. 0. 1. 0. 1. 0.]
 [0. 1. 0. 0. 0. 1. 1. 0. 1. 0.]
 [0. 1. 0. 1. 0. 0. 0. 1. 1. 0.]
 [0. 1. 0. 1. 0. 0. 0. 1. 0. 1.]
 [1. 0. 0. 1. 0. 0. 0. 1. 0. 1.]
 [0. 0. 1. 0. 0. 1. 1. 0. 1. 0.]
 [0. 0. 1. 1. 0. 0. 0. 1. 1. 0.]
 [0. 1. 0. 0. 0. 1. 0. 1. 1. 0.]
 [0. 0. 1. 0. 0. 1. 0. 1. 0. 1.]
 [1. 0. 0. 0. 0. 1. 1. 0. 0. 1.]
 [1. 0. 0. 0. 1. 0. 0. 1. 1. 0.]
 [0. 1. 0. 0. 1. 0. 0. 1. 1. 0.]
 [0. 1. 0. 0. 0. 1. 1. 0. 0. 1.]]
```

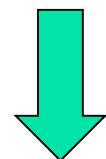
Data Preprocessing

- Check the encoded result

```
categories = []
for x in enc.categories_:
    categories += list(x)
print(categories)
```

```
['overcast', 'rainy', 'sunny', 'cool', 'hot', 'mild', 'high', 'normal', False, True]
```

0 1 0 0 0 1 1 0 0 1



['rainy', 'mild', 'high', True]

Building a Tree and Testing

Train a decision tree

1

```
clf = tree.DecisionTreeClassifier(criterion = "entropy")
clf = clf.fit(en_X,y)
print(clf.classes_)
```

['no' 'yes'] Labels of the model

2

```
test = [['overcast','mild','high', True]
en_test = enc.transform(test).toarray()
print(en_test)
pred_y = clf.predict(en_test)
print(pred_y)
```

```
[[1. 0. 0. 0. 1. 1. 0. 0. 1.]]
['yes']
```

Encode the test data

Note: test data should be
a 2-D array

Output represents the
predicted label of each data

The probability of each label
can also be shown

3

```
pred_prob = clf.predict_proba(en_test)
print(pred_prob)

[[0. 1.]]
```

Cross-validation

```
cv = KFold(n_splits = 10,  
            shuffle=True,  
            random_state=0)  
cv_results = cross_val_score(clf, en_X, y, cv=cv)  
  
print(cv_results.mean())          X → en_X
```

0.55

You can reuse the code used for KNN classifier
since the cross-validation code is almost the same

Visualize Tree

- Tree class includes the function *export_graphviz*

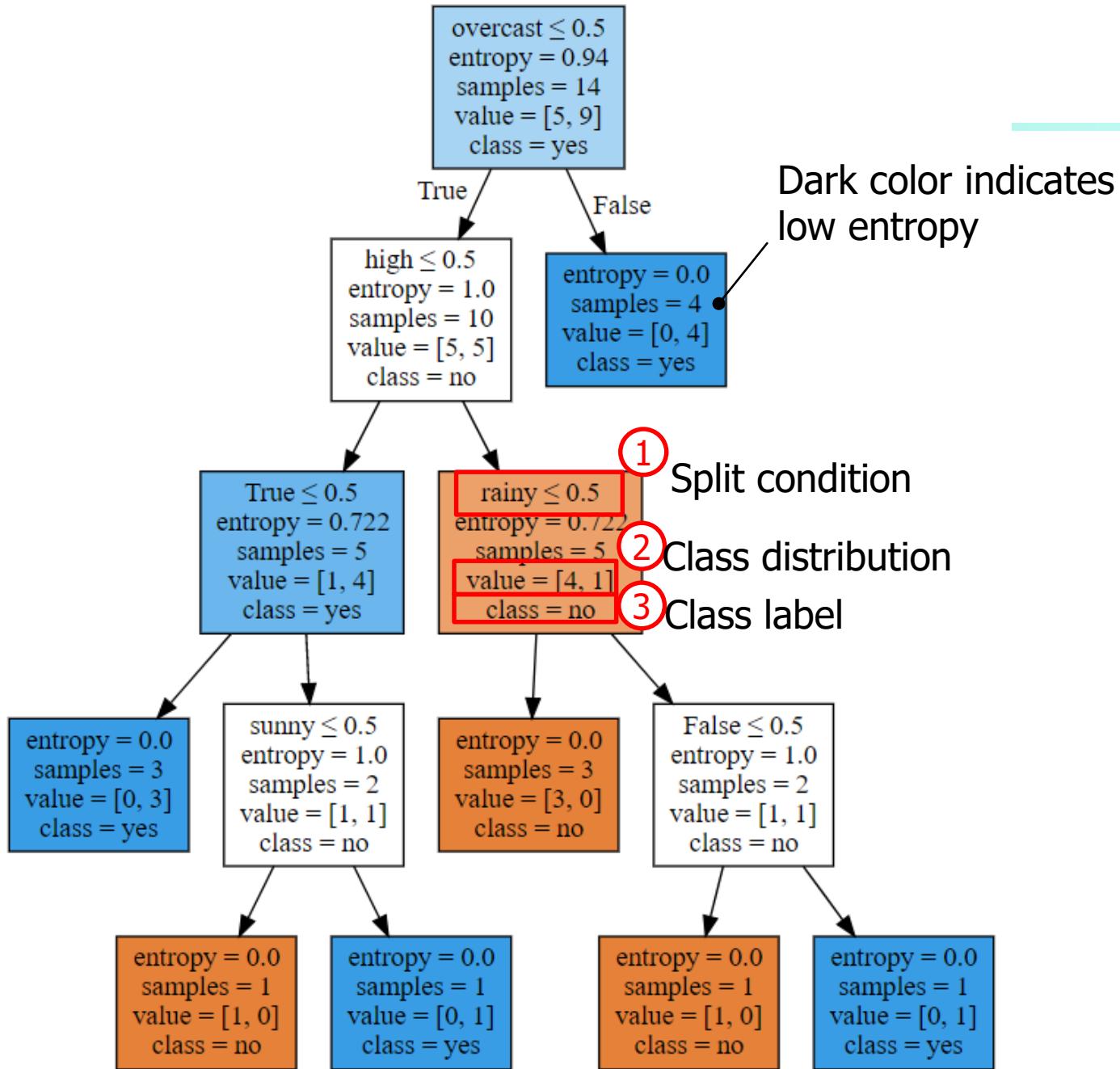
```
dot_data = tree.export_graphviz(clf, out_file=None,
                                feature_names=categories,
                                class_names=clf.classes_,
                                filled=True,
                                special_characters = True)
graph = graphviz.Source(dot_data)
graph
```

feature_names : the list of attributes of data

class_names : the list of labels of data

filled : graphviz colors the node if set True

special_characters : shows the special characters (i.e., \leq or \leq)



Changing Parameters

```
clf = tree.DecisionTreeClassifier(criterion = "entropy",
                                  max_depth = 3,
                                  min_samples_split = 3,
                                  min_samples_leaf = 2)
```

criterion : measures used for decision tree ("entropy" or "gini")

max_depth : maximum depth of the decision tree

min_samples_split : the minimum number of samples required
to split an internal node

min_samples_leaf : the minimum number of samples in a leaf

ZeroR in Python

X is not needed since zeroR retrieves the major label

```
def zeroR(y, test):  
    output_values = [label for label in y]  
    prediction = max(set(output_values), key=output_values.count)  
    predicted = [prediction for i in range(len(test))]  
    return predicted
```

```
zeroR(y, [['overcast', 'cool', 'high', False]])
```

```
['yes']
```

ZeroR in Python

Generates the list which has
the same element with y

```
def zeroR(y, test):  
    output_values = [label for label in y]  
    prediction = max(set(output_values), key=output_values.count)  
    predicted = [prediction for i in range(len(test))]  
    return predicted
```

```
zeroR(y, [['overcast', 'cool', 'high', False]])
```

```
['yes']
```

ZeroR in Python

The set of distinct labels
(i.e., {yes, no})

```
def zeroR(y, test):
    output_values = [label for label in y]
    prediction = max(set(output_values), key=output_values.count)
    predicted = [prediction for i in range(len(test))]
    return predicted
```

```
zeroR(y, [['overcast', 'cool', 'high', False]])
```

```
['yes']
```

ZeroR in Python

```
def zeroR(y, test):  
    output_values = [label for label in y]  
    prediction = max(set(output_values), key=output_values.count)  
    predicted = [prediction for i in range(len(test))]  
    return predicted
```

The label with the maximum freq.
will be retrieved

```
zeroR(y, [['overcast', 'cool', 'high', False]])
```

```
['yes']
```

ZeroR in Python

```
def zeroR(y, test):
    output_values = [label for label in y]
    prediction = max(set(output_values), key=output_values.count)
    predicted = [prediction for i in range(len(test))]
    return predicted
```

Iteration for the number of instances
in test set

```
zeroR(y, [['overcast', 'cool', 'high', False]])
```

['yes'] Test data should be a 2-D array

ZeroR in Python

```
def zeroR(y, test):
    output_values = [label for label in y]
    prediction = max(set(output_values), key=output_values.count)
    predicted = [prediction for i in range(len(test))]
    return predicted
```

Retrieve the major label
for all instances of the test set

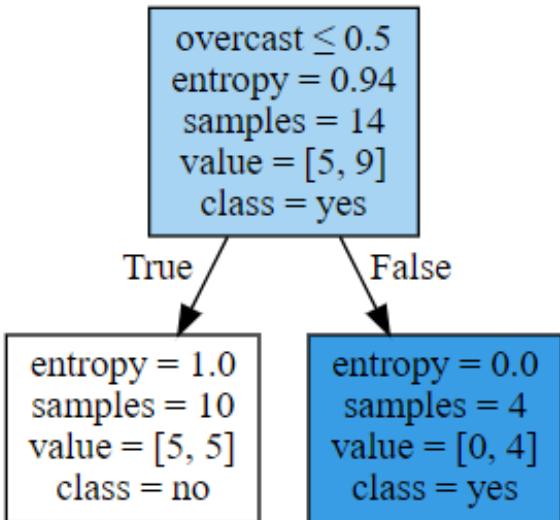
```
zeroR(y, [['overcast', 'cool', 'high', False]])
```

['yes'] Test data should be a 2-D array

Decision Tree with Depth 1

- Instead of OneR, we generate decision tree with depth only 1

```
clf = tree.DecisionTreeClassifier(criterion='entropy',
                                  max_depth=1)
```



```
cv = KFold(n_splits = 10,
            shuffle=True,
            random_state=0)
cv_results = cross_val_score(clf, en_X, y, cv=cv)
print(cv_results.mean())
```

0.3

Attribute Selection

As we did with Weka, we will filter the attributes of "glass.csv" and generate a decision tree

Preprocess Classify Cluster Associate Select attributes Visual

Open file... Open URL... Open DB... Gener

Filter

Choose None

Current relation

Relation: Glass Attributes: 10 Instances: 214 Sum of weights: 214

Attributes

All None Invert Pattern

No.	Name
1	RI
2	Na
3	Mg
4	Al
5	Si
6	K
7	Ca
8	Ba
9	Fe
10	Type

Remove

```
df = pd.read_csv('glass.csv')  
df
```

	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type
0	1.51793	12.79	3.50	1.12	73.03	0.64	8.77	0.00	0.00	'build wind float'
1	1.51643	12.16	3.52	1.35	72.89	0.57	8.53	0.00	0.00	'vehic wind float'
2	1.51793	13.21	3.48	1.41	72.64	0.59	8.43	0.00	0.00	'build wind float'
3	1.51299	14.40	1.74	1.54	74.55	0.00	7.59	0.00	0.00	tableware
4	1.53393	12.30	0.00	1.00	70.16	0.12	16.19	0.00	0.24	build wind non-float'
5	1.51655	12.75	2.85	1.44	73.27	0.57	8.79	0.11	0.22	build wind non-float'
6	1.51779	13.64	3.65	0.65	73.00	0.06	8.93	0.00	0.00	'vehic wind float'
7	1.51837	13.14	2.84	1.28	72.85	0.55	9.07	0.00	0.00	'build wind float'
8	1.51545	14.14	0.00	2.68	73.39	0.08	9.07	0.61	0.05	headlamps
9	1.51789	13.19	3.90	1.30	72.33	0.55	8.44	0.00	0.28	build wind non-float'
10	1.51625	13.36	3.58	1.49	72.72	0.45	8.21	0.00	0.00	build wind non-float'

Attribute Selection

```
X = []
for row in df.values:
    new_row = list(row[1:2]) + list(row[3:-1])
    X.append(new_row)
```

X

```
[[12.79, 1.12, 73.03, 0.64, 8.77, 0.0, 0.0],
 [12.16, 1.35, 72.89, 0.57, 8.53, 0.0, 0.0],
 [13.21, 1.41, 72.64, 0.59, 8.43, 0.0, 0.0],
 [14.4, 1.54, 74.55, 0.0, 7.59, 0.0, 0.0],
 [12.3, 1.0, 70.16, 0.12, 16.19, 0.0, 0.24],
 [12.75, 1.44, 73.27, 0.57, 8.79, 0.11, 0.22],
 [13.64, 0.65, 73.0, 0.06, 8.93, 0.0, 0.0],
 [13.14, 1.28, 72.85, 0.55, 9.07, 0.0, 0.0],
 [14.14, 2.68, 73.39, 0.08, 9.07, 0.61, 0.05],
 [13.19, 1.3, 72.33, 0.55, 8.44, 0.0, 0.28],
 [13.36, 1.49, 72.72, 0.45, 8.21, 0.0, 0.0],
 [12.2, 1.16, 73.55, 0.62, 8.9, 0.0, 0.24],
 [13.21, 0.79, 71.99, 0.13, 10.02, 0.0, 0.0],
 [14.03, 0.58, 71.79, 0.11, 9.65, 0.0, 0.0],
 [13.14, 1.76, 72.48, 0.6, 8.38, 0.0, 0.17],
 [13.48, 1.71, 72.52, 0.62, 7.99, 0.0, 0.0],
 [14.75, 2.0, 73.02, 0.0, 8.53, 1.59, 0.08],
```

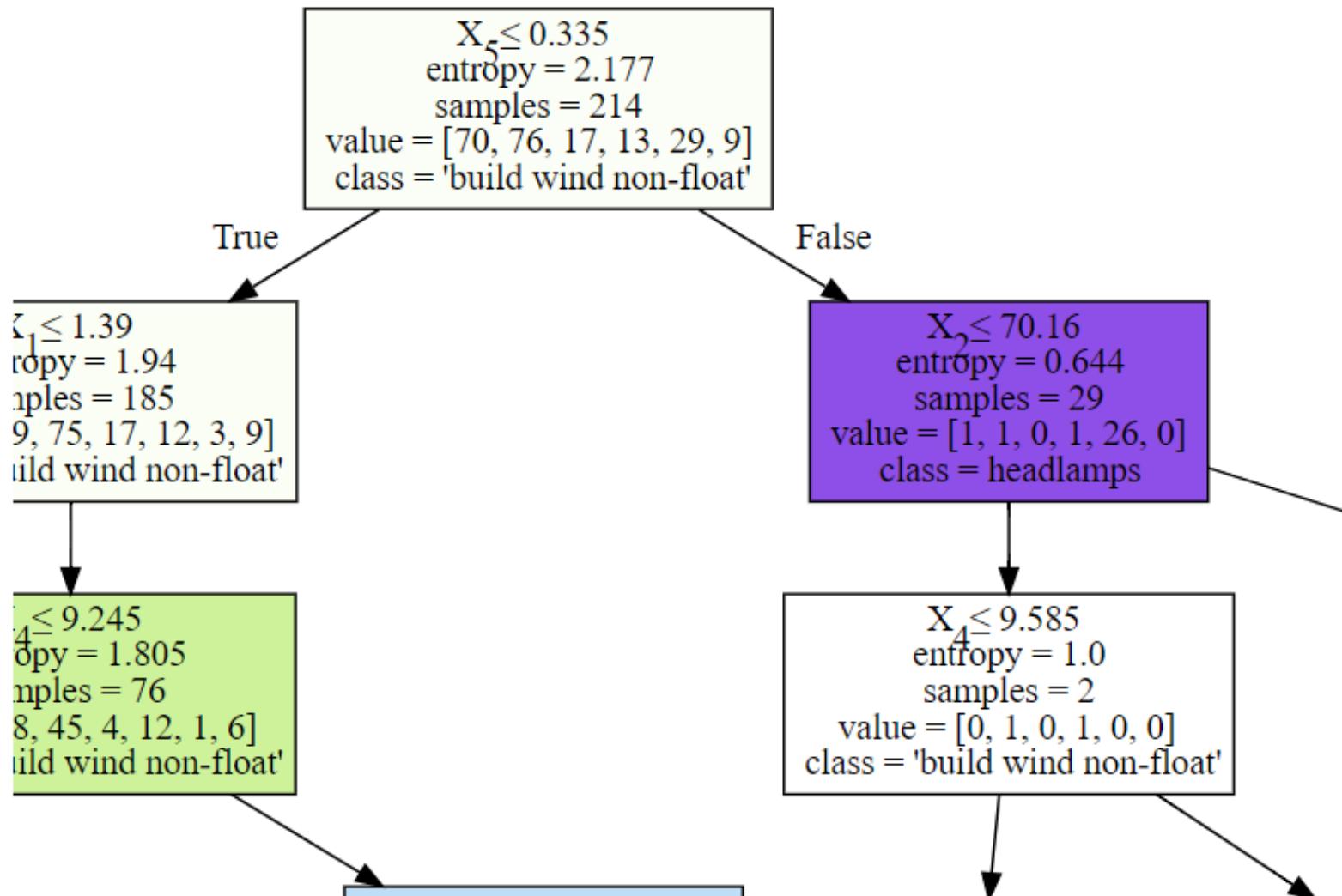
```
df = pd.read_csv('glass.csv')
df
```

	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type
0	1.51793	12.79	3.50	1.12	73.03	0.64	8.77	0.00	0.00	'build wind float'
1	1.51643	12.16	3.52	1.35	72.89	0.57	8.53	0.00	0.00	'vehic wind float'
2	1.51793	13.21	3.48	1.41	72.64	0.59	8.43	0.00	0.00	'build wind float'
3	1.51299	14.40	1.74	1.54	74.55	0.00	7.59	0.00	0.00	tableware
4	1.53393	12.30	0.00	1.00	70.16	0.12	16.19	0.00	0.24	build wind non-float'
5	1.51655	12.75	2.85	1.44	73.27	0.57	8.79	0.11	0.22	build wind non-float'
6	1.51779	13.64	3.65	0.65	73.00	0.06	8.93	0.00	0.00	'vehic wind float'
7	1.51837	13.14	2.84	1.28	72.85	0.55	9.07	0.00	0.00	'build wind float'
8	1.51545	14.14	0.00	2.68	73.39	0.08	9.07	0.61	0.05	headlamps
9	1.51789	13.19	3.90	1.30	72.33	0.55	8.44	0.00	0.28	build wind non-float'
10	1.51625	13.36	3.58	1.49	72.72	0.45	8.21	0.00	0.00	build wind non-float'

```
clf.fit(X, y)
```

No encoding needed since "glass.csv" is numeric

```
dot_data = tree.export_graphviz(clf, out_file = None,  
                                class_names=clf.classes_,  
                                filled=True,  
                                special_characters = True)  
graph = graphviz.Source(dot_data)  
graph
```



ZeroR with diabetes.csv

```
from sklearn.model_selection import train_test_split
import pandas as pd          function for splitting data

df = pd.read_csv('diabetes.csv')
X = df.values[:, :-1]
y = df.values[:, -1]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1)

hit = 0
for i in range(len(y_test)):
    predicted = zeroR(y_train, X_test[i:i+1])
    if(predicted == y_test[i:i+1]):
        hit += 1

print(hit / len(y_test))
```

0.6493506493506493

ZeroR with diabetes.csv

```
from sklearn.model_selection import train_test_split
import pandas as pd

df = pd.read_csv('diabetes.csv')
X = df.values[:, :-1]
y = df.values[:, -1]

X_train, X_test, y_train, y_test # 
    = train_test_split(X, y, test_size = 0.1)

hit = 0
for i in range(len(y_test)):
    predicted = zeroR(y_train, X_test[i:i+1])
    if(predicted == y_test[i:i+1]):
        hit += 1

print(hit / len(y_test))
```

Train : Test
= 9 : 1

0.6493506493506493

ZeroR with diabetes.csv

```
from sklearn.model_selection import train_test_split
import pandas as pd

df = pd.read_csv('diabetes.csv')
X = df.values[:, :-1]
y = df.values[:, -1]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1)

hit = 0
for i in range(len(y_test)):
    predicted = zeroR(y_train, X_test[i:i+1])
    if(predicted == y_test[i:i+1]): 2-D array
        hit += 1 1-D array

print(hit / len(y_test))
```

Count the number of correct predictions

0.6493506493506493

Breast-cancer.csv

```
df = pd.read_csv('breast-cancer.csv')
df
```

	age	menopause	tumor-size	inv-nodes	node-caps	deg-malig	breast	breast-quad	irradiat	Class
0	'40-49'	'premeno'	'15-19'	'0-2'	'yes'	'3'	'right'	'left_up'	'no'	'recurrence-events'
1	'50-59'	'ge40'	'15-19'	'0-2'	'no'	'1'	'right'	'central'	'no'	'no-recurrence-events'
2	'50-59'	'ge40'	'35-39'	'0-2'	'no'	'2'	'left'	'left_low'	'no'	'recurrence-events'
3	'40-49'	'premeno'	'35-39'	'0-2'	'yes'	'3'	'right'	'left_low'	'yes'	'no-recurrence-events'
4	'40-49'	'premeno'	'30-34'	'3-5'	'yes'	'2'	'left'	'right_up'	'no'	'recurrence-events'
5	'50-59'	'premeno'	'25-29'	'3-5'	'no'	'2'	'right'	'left_up'	'yes'	'no-recurrence-events'

Effect of Tree Size

```
clf = tree.DecisionTreeClassifier(criterion='entropy')
cv = KFold(n_splits = 10,
            shuffle=True,
            random_state=0)
cv_results = cross_val_score(clf, en_X, y, cv=cv)

print(cv_results.mean())
```

0.6294334975369458

```
clf = tree.DecisionTreeClassifier(criterion='entropy',
                                  max_depth=3)

cv = KFold(n_splits = 10,
            shuffle=True,
            random_state=0)
cv_results = cross_val_score(clf, en_X, y, cv=cv)

print(cv_results.mean())
```

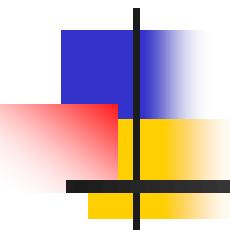
0.7201970443349753

Practice

- Diabetes.arff 를 가지고 decision tree를 만들어서 visualization 하는 코드도 작성하시오.

Practice

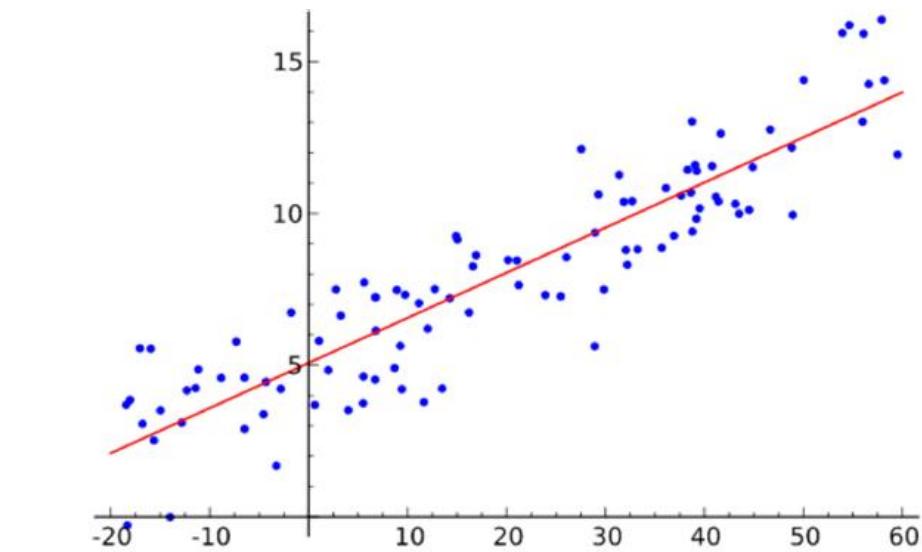
- Breast-cancer.arff 를 가지고 one-hot encoding을 해서 decision tree를 만들고 만들어진 decision tree를 visualization 하는 코드도 작성하시오.



Linear Regression

Linear Regression

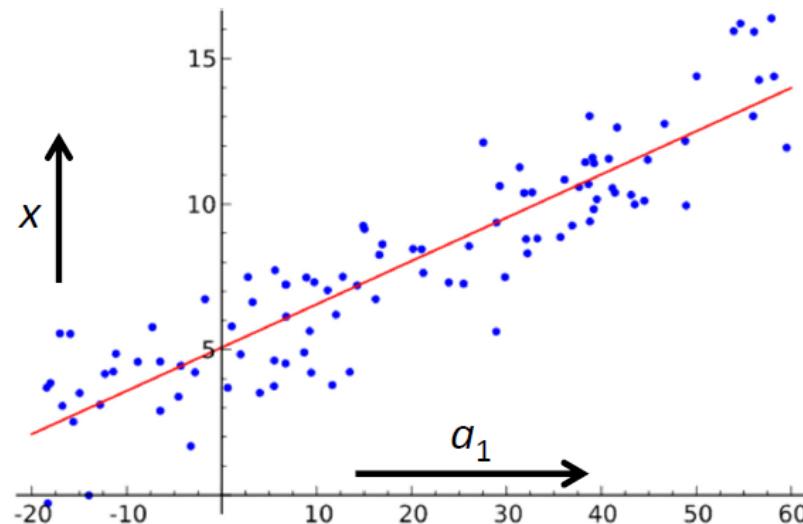
- Numeric prediction (called “regression”)
 - Data sets so far: nominal and numeric attributes, but only nominal classes
 - Now: numeric classes
 - Classical statistical method (from 1805!)



Linear Regression

$$x = w_0 + w_1 a_1 + w_2 a_2 + \dots + w_k a_k$$

(Works most naturally
with numeric attributes)



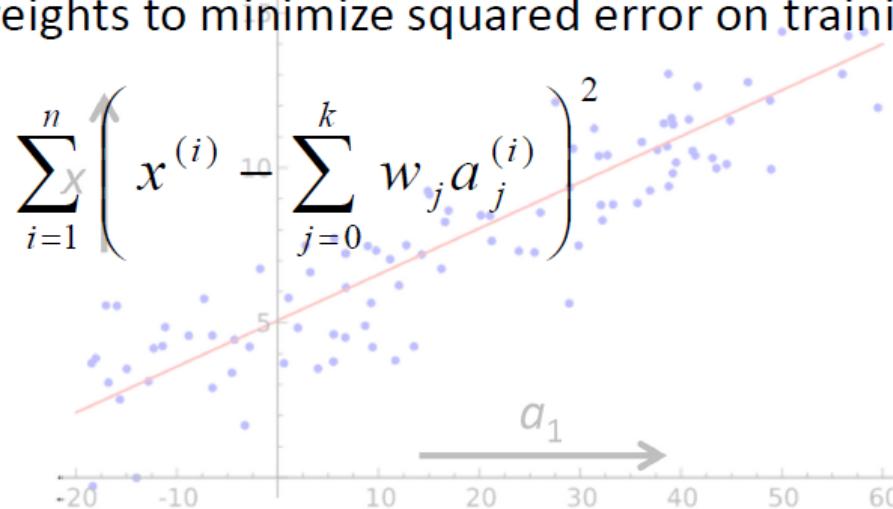
Linear Regression

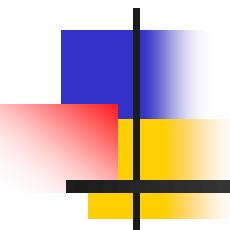
$$x = w_0 + w_1 a_1 + w_2 a_2 + \dots + w_k a_k$$

- ❖ Calculate weights from training data
- ❖ Predicted value for first training instance $a^{(1)}$

$$w_0 a_0^{(1)} + w_1 a_1^{(1)} + w_2 a_2^{(1)} + \dots + w_k a_k^{(1)} = \sum_{j=0}^k w_j a_j^{(1)}$$

- ❖ Choose weights to minimize squared error on training data





Weka – Linear Regression

Linear Regression

- ❖ Open file **cpu.arff**: all numeric attributes and classes
- ❖ Choose **functions>LinearRegression**
- ❖ Run it
- ❖ Output:

– *Correlation coefficient*

– *Mean absolute error*

– *Root mean squared error*

– *Relative absolute error*

– *Root relative squared error*

$$\frac{|p_1 - a_1| + \dots + |p_n - a_n|}{n}$$

$$\sqrt{\frac{(p_1 - a_1)^2 + \dots + (p_n - a_n)^2}{n}}$$

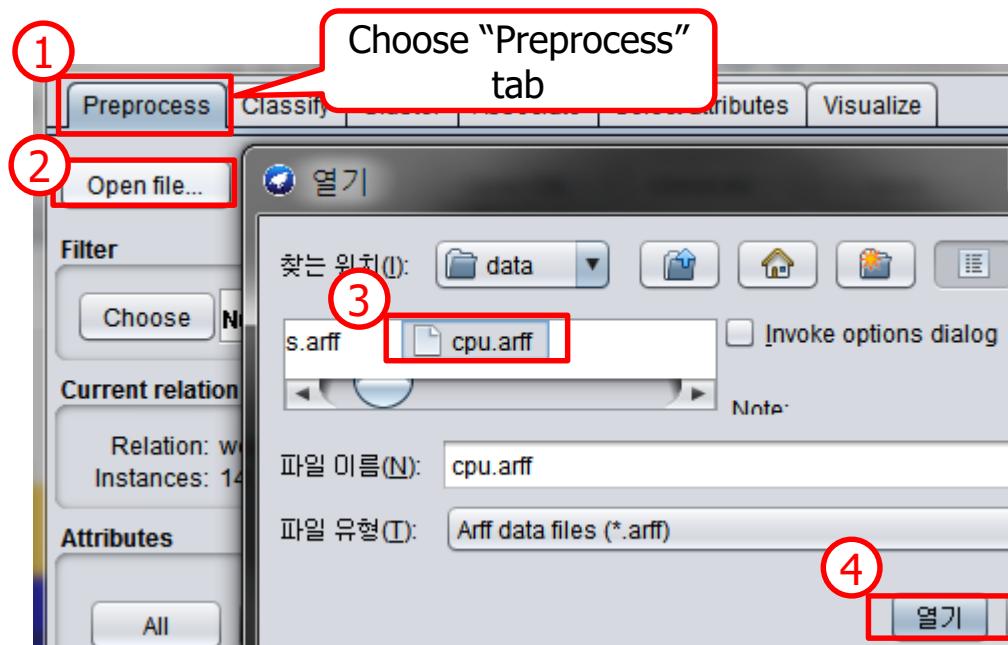
$$\frac{|p_1 - a_1| + \dots + |p_n - a_n|}{|a_1 - \bar{a}| + \dots + |a_n - \bar{a}|}$$

$$\sqrt{\frac{(p_1 - a_1)^2 + \dots + (p_n - a_n)^2}{(a_1 - \bar{a})^2 + \dots + (a_n - \bar{a})^2}}$$

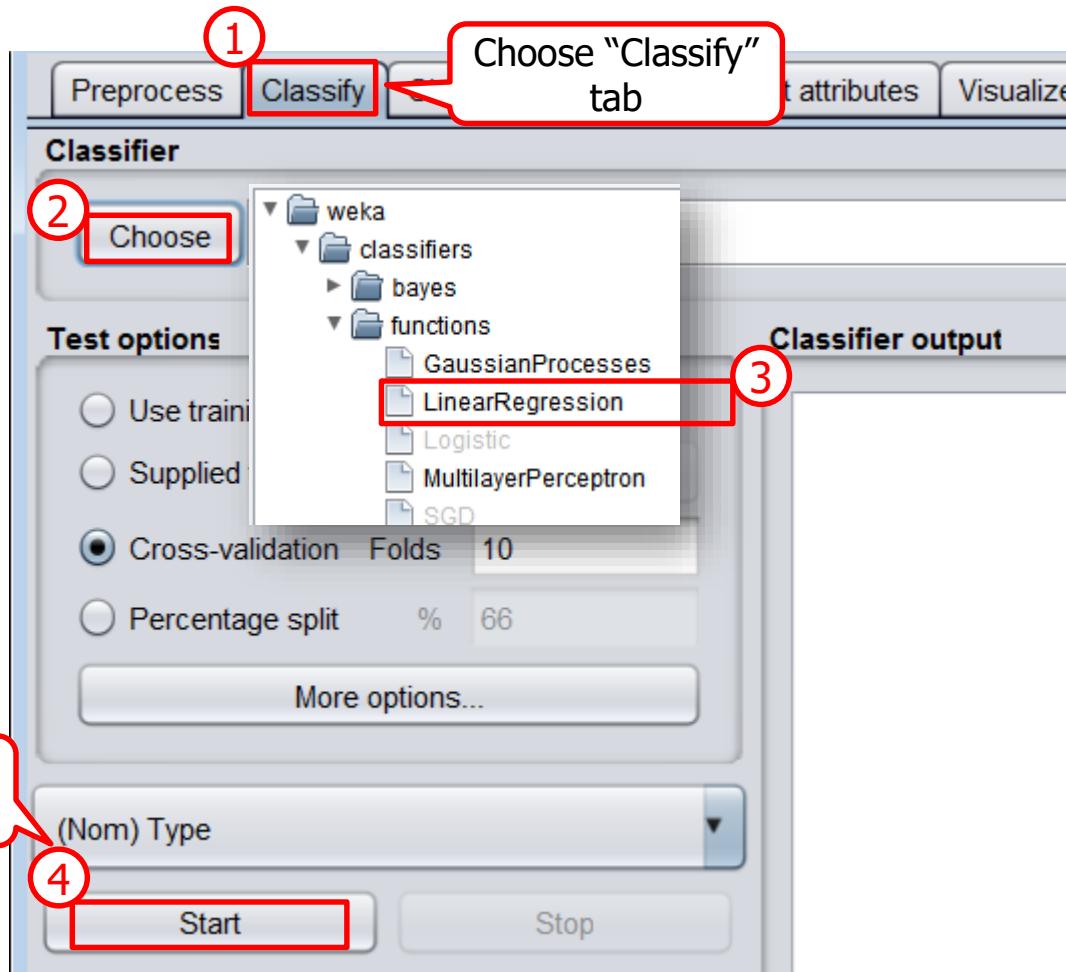
- ❖ Examine model

Open the Dataset

- C:\Program Files\Weka-3-8\data\cpu.arff

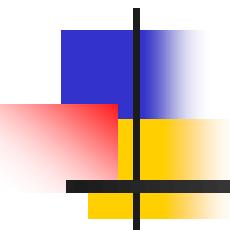


Select the Classifier



Linear Regression

Classifier output	
==== Cross-validation ===	
==== Summary ===	
Correlation coefficient	0.9012
Mean absolute error	41.0886
Root mean squared error	69.556
Relative absolute error	42.6943 %
Root relative squared error	43.2421 %
Total Number of Instances	209



Python – Linear Regression

Download the Dataset

- Download cpu.csv from <http://kdd.snu.ac.kr/python/>
- Save the csv file in the same directory as the source file (.ipynb)

Import Libraries

```
import numpy as np
import pandas as pd
from sklearn import linear_model, metrics
from sklearn.model_selection import cross_val_score
```

Import Libraries

```
import numpy as np  
import pandas as pd  
from sklearn import linear_model, metrics  
from sklearn.model_selection import cross_val_score
```

- numpy : A library that supports various array operations
- import numpy as **np** : We will use name ‘np’ instead of ‘numpy’
 - ex) arr1 = numpy.array([1,2,3,4]) X
arr1 = np.array(([1,2,3,4])) O

Import Libraries

```
import numpy as np
import pandas as pd
from sklearn import linear_model, metrics
from sklearn.model_selection import cross_val_score
```

- pandas : A library that supports data analysis
- We will use it to load csv data and visualize it

Import Libraries

```
import numpy as np
import pandas as pd
from sklearn import linear_model, metrics
from sklearn.model_selection import cross_val_score
```

- From sklearn library, we will import linear_model and metrics modules

Import Libraries

```
import numpy as np
import pandas as pd
from sklearn import linear_model, metrics
from sklearn.model_selection import cross_val_score
```

- From sklearn.model_selection module, we will import cross_val_score function

Load Data

```
In [2] : df = pd.read_csv("cpu.csv")
df[:5]
```

Out [2] :

	MYCT	MMIN	MMAX	CACH	CHMIN	CHMAX	class
0	125	256	6000	256	16	128	198
1	29	8000	32000	32	8	32	269
2	29	8000	32000	32	8	32	220
3	29	8000	32000	32	8	32	172
4	29	8000	16000	32	8	16	132

Load Data

Read the csv file and store it in a pandas dataframe object

In [2] : df = pd.read_csv("cpu.csv")

df [:5] Jupyter notebook can visualize the dataframe object

Out [2] : [:5] means to select the rows up to 5th row

	MYCT	MMIN	MMAX	CACH	CHMIN	CHMAX	class
0	125	256	6000	256	16	128	198
1	29	8000	32000	32	8	32	269
2	29	8000	32000	32	8	32	220
3	29	8000	32000	32	8	32	172
4	29	8000	16000	32	8	16	132

Load Data

```
In [3] : X = df.values[:, :-1]
          y = df.values[:, -1]
          print(X[:3])
          print(y[:3])
```

```
X[:3]   [[ 125    256   6000    256     16    128]
           [   29    8000  32000     32      8     32]
           [   29    8000  32000     32      8     32]]
Y[:3]   [198  269  220]
```

- `df.values` returns an array form of the data
- `df.values[:, :-1]` means that select all rows(:) and all columns except the last column(:-1)

Load Data

```
In [3] : X = df.values[:, :-1]
          y = df.values[:, -1]
          print(X[:3])
          print(y[:3])
```

```
X[:3]   [[ 125    256   6000    256     16    128]
          [   29    8000  32000     32      8     32]
          [   29    8000  32000     32      8     32]]
Y[:3]   [198  269  220]
```

- `df.values` returns an array form of the data
- `df.values[:, :-1]` means that select all rows(:) and all columns except the last column(:-1)

Load Data

```
In [3] : X = df.values[:, :-1]
          y = df.values[:, -1]
          print(X[:3])
          print(y[:3])
```

X[:3]	[[125 256 6000 256 16 128] [29 8000 32000 32 8 32] [29 8000 32000 32 8 32]]
Y[:3]	[198 269 220] MYCT MMIN MMAX CACH CHMIN CHMAX class 0 125 256 6000 256 16 128 198 1 29 8000 32000 32 8 32 269 2 29 8000 32000 32 8 32 220 3 29 8000 32000 32 8 32 172

Regression Model

```
In [4]: regr = linear_model.LinearRegression(fit_intercept = True,
                                             normalize = True)
scores = cross_val_score(regr,
                         X,
                         y,
                         cv = 10,
                         scoring =
                             metrics.make_scorer(metrics.mean_squared_error))
print(np.sqrt(scores).mean())
```

69.99293084239739

Regression Model

Linear regression model

In [4] :

```
regr = linear_model.LinearRegression(fit_intercept = True,  
                                     normalize = True)  
  
scores = cross_val_score(regr,  
                         X,  
                         y,  
                         cv = 10,  
                         scoring =  
                             metrics.make_scorer(metrics.mean_squared_error))  
print(np.sqrt(scores).mean())
```

69.99293084239739

- Create a linear regression model object
- Parameters:
 - fit_intercept : Whether to use constant term
 - normalize : Whether to normalize the input data

Regression Model

```
In [4]: regr = linear_model.LinearRegression(fit_intercept = True,  
                                         normalize = True)  
scores = cross_val_score(regr, _____ Model object to evaluate  
                         X, _____ input data  
                         y, _____ label of the data  
                         cv = 10,  
                         scoring =  
                           metrics.make_scorer(metrics.mean_squared_error))  
print(np.sqrt(scores).mean())
```

69.99293084239739

- Perform a 10-fold validation on the model
- Parameters:
 - cv : Number of folds
 - scoring : Which measure to evaluate the model

Regression Model

```
In [4]: regr = linear_model.LinearRegression(fit_intercept = True,  
                                         normalize = True)  
scores = cross_val_score(regr,  
                        X,  
                        y,  
                        cv = 10,  
                        scoring =  
                           metrics.make_scorer(metrics.mean_squared_error))  
print(np.sqrt(scores).mean())
```

69.99293084239739

Get the mean squared error

- We will evaluate the regression model with root-mean-square error (RMSE)

Regression Model

```
In [4] : regr = linear_model.LinearRegression(fit_intercept = True,  
                                         normalize = True)  
scores = cross_val_score(regr,  
                        X,  
                        y,  
                        cv = 10,  
                        scoring =  
                           metrics.make_scorer(metrics.mean_squared_error))  
print(np.sqrt(scores).mean())
```

69.99293084239739

Get the average RMSE

- score : 10-sized array, where each entry is the mean squared error of each fold
- np.sqrt(scores) : Get the square root of each entry
- .mean() : Get the average of 10 entries

Prediction

Train the model with full data

```
In [5]: regr.fit(X, y)
y_pred = regr.predict([[25, 8000, 64000, 64, 8, 32]])
print(y_pred)
```

[510.56289692]

- Now train the model with full data to get the final model

Prediction

Predict the label of a data instance

```
In [5]: regr.fit(X, y)
y_pred = regr.predict([[25, 8000, 64000, 64, 8, 32]])
print(y_pred)
```

[510.56289692]

- Make a data instance and get the predicted label

MYCT	MMIN	MMAX	CACH	CHMIN	CHMAX	class
25	8000	64000	64	8	32	510.56(predicted)

Prediction

```
In [5]: regr.fit(X, y)
y_pred = regr.predict([[25, 8000, 64000, 64, 8, 32]])
print(y_pred)
```

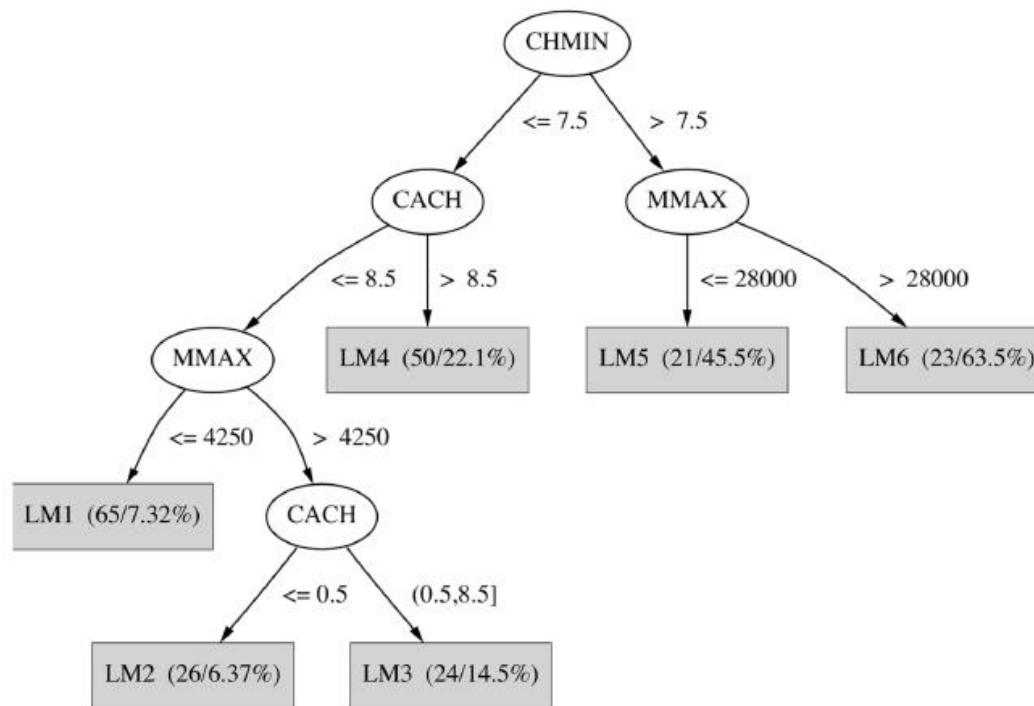
[510.56289692]

- You can also input an array of data instances to get an array of predicted labels

Decision Tree Regression

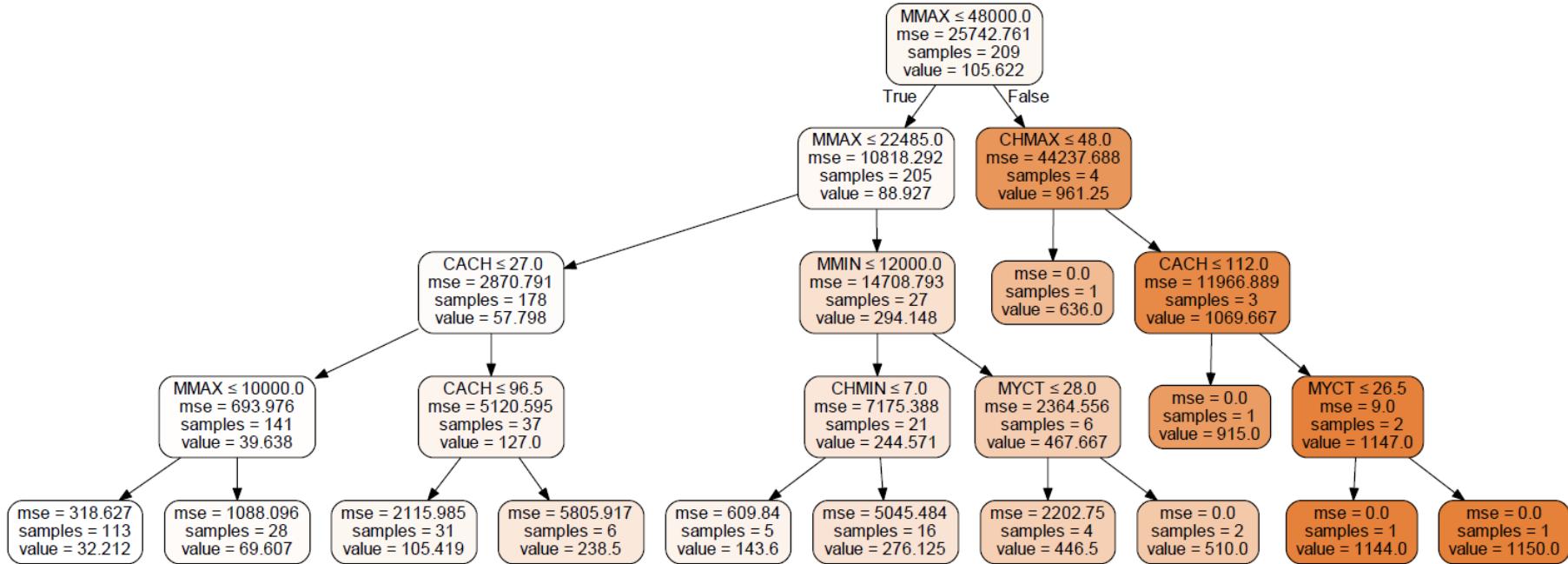
- M5 Model tree

- Each leaf has a linear regression model
- Linear patches approximate continuous function

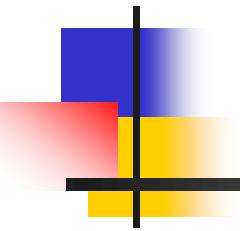


Decision Tree Regression

- The data is split with certain value of an attribute
- The regression result is determined by a constant assigned to each leaf



Weka – Decision Tree Regression



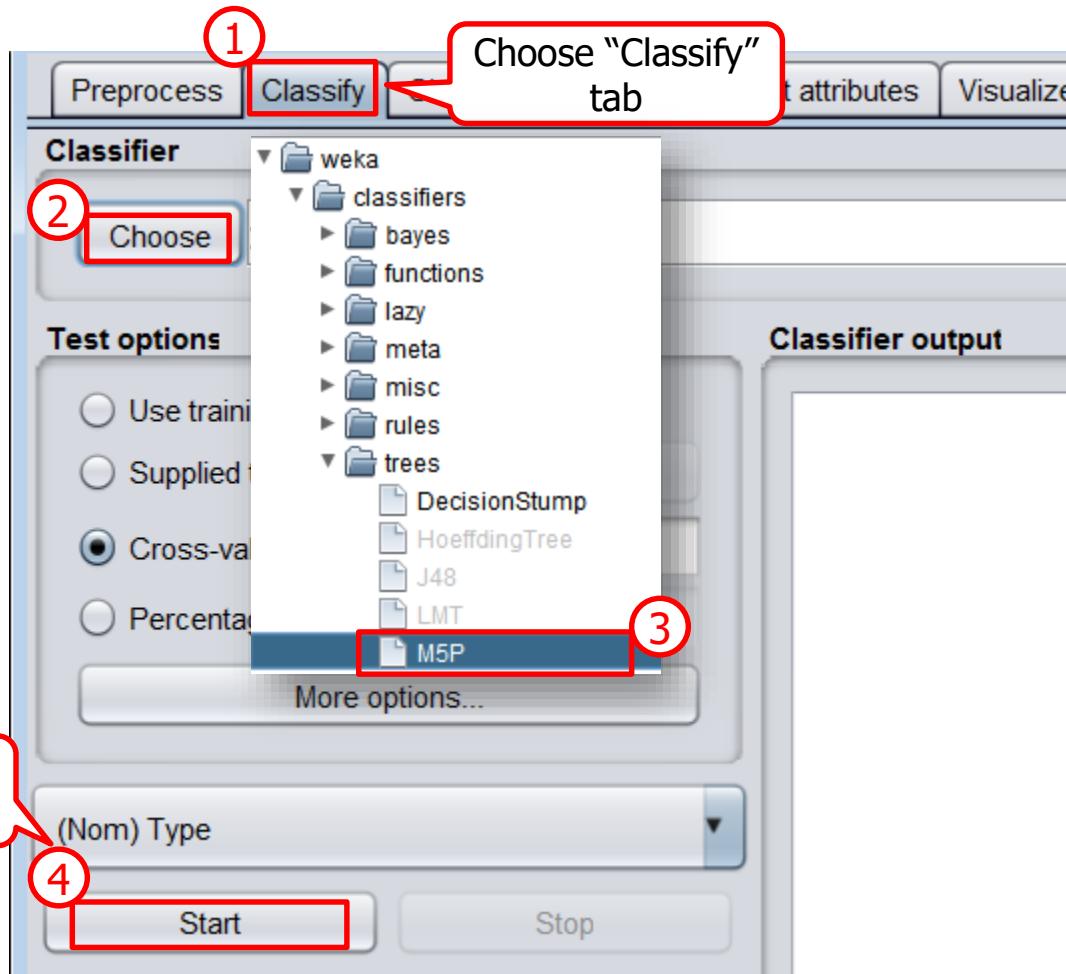
Decision Tree Regression

- Choose **trees>M5P**
- Run it
- Output:
 - *Examine the linear models*
 - *Visualize the tree*
- Compare performance with the **LinearRegression** result

Decision Tree Regression

- Well-founded, venerable mathematical technique:
 - **functions>LinearRegression**
- Practical problems often require non-linear solutions
- **trees>M5P** builds trees of regression models

Select the Classifier



Decision Tree Regression

Classifier output

```
+ 3.3671 * CRIMRA  
- 51.8474

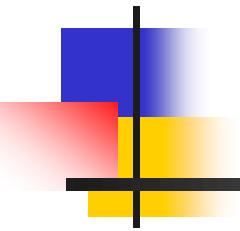
Number of Rules : 5

Time taken to build model: 0.15 seconds

==== Cross-validation ====
==== Summary ===

Correlation coefficient          0.9274
Mean absolute error            29.8309
Root mean squared error        60.7112
Relative absolute error        30.9967 %
Root relative squared error   37.7434 %
Total Number of Instances      209
```

Python – Decision Tree Regression



Download the Dataset

- Download cpu.csv from <http://kdd.snu.ac.kr/python/>
- Save the csv file in the same directory as the source file (.ipynb)

Import Libraries

```
In [1]: from sklearn import tree, metrics  
       from sklearn.model_selection import cross_val_score  
       import graphviz  
       import numpy as np  
       import pandas as pd
```

- **graphviz** : A library that supports tree visualization

Load Data

```
In [2] : df = pd.read_csv("cpu.csv")
X = df.values[:, :-1]
y = df.values[:, -1]
df[:5]
```

Out [2] :

	MYCT	MMIN	MMAX	CACH	CHMIN	CHMAX	class
0	125	256	6000	256	16	128	198
1	29	8000	32000	32	8	32	269
2	29	8000	32000	32	8	32	220
3	29	8000	32000	32	8	32	172
4	29	8000	16000	32	8	16	132

Decision Tree Regression

```
In [3]: regr = tree.DecisionTreeRegressor(max_depth = 4, random_state = 1)
scores = cross_val_score(regr,
                        X,
                        y,
                        cv = 10,
                        scoring =
                            metrics.make_scorer(metrics.mean_squared_error))
print(np.sqrt(scores).mean())
```

66.98221513495744

Decision Tree Regression

```
In [3]: regr = tree.DecisionTreeRegressor(max_depth = 4, random_state = 1)
scores = cross_val_score(regr,
                        X,
                        y,
                        cv = 10,
                        scoring =
                            metrics.make_scorer(metrics.mean_squared_error))
print(np.sqrt(scores).mean())
```

Decision tree
regression model

66.98221513495744

- Create a decision tree regression model object
- Parameters:
 - max_depth : Maximum depth of the tree
 - min_samples_split : Minimum number of samples to split a node
 - min_samples_leaf : Minimum number of samples of a leaf node
 - random_state : Seed of the random number generator

Decision Tree Regression

In [3] :

```
regr = tree.DecisionTreeRegressor(max_depth = 4, random_state = 1)
scores = cross_val_score(regr,
                         X,
                         y,
                         cv = 10,
                         scoring =
                             metrics.make_scorer(metrics.mean_squared_error))
print(np.sqrt(scores).mean())
```

Decision tree
regression model

66.98221513495744

■ Why random number?

- Learning an optimal decision tree is a NP-complete problem
- Therefore sklearn performs a sub-optimal greedy algorithm
- The random number generator is used in the greedy algorithm

Decision Tree Regression

```
In [3]: regr = tree.DecisionTreeRegressor(max_depth = 4, random_state = 1)
scores = cross_val_score(regr, _____ Model object to evaluate
                         X, _____ input data
                         y, _____ label of the data
                         cv = 10,
                         scoring =
                           metrics.make_scorer(metrics.mean_squared_error))
print(np.sqrt(scores).mean())
```

66.98221513495744

- Perform a 10-fold validation on the model
- Parameters:
 - cv : Number of folds
 - scoring : Which measure to evaluate the model

Tree Visualization

```
In [4]: regr.fit(X, y)
dot_data = tree.export_graphviz(regr,
                                out_file=None,
                                feature_names=df.columns[:-1],
                                class_names=None,
                                filled=True,
                                rounded=True,
                                special_characters=True)
graph = graphviz.Source(dot_data)
graph.render("cpu")
```

Out[4] : 'cpu.pdf'

Tree Visualization

Fit the model using the full data

In [4] :

```
regr.fit(X, y)
dot_data = tree.export_graphviz(regr,
                                out_file=None,
                                feature_names=df.columns[:-1],
                                class_names=None,
                                filled=True,
                                rounded=True,
                                special_characters=True)
graph = graphviz.Source(dot_data)
graph.render("cpu")
```

Out [4] : 'cpu.pdf'

Tree Visualization

Visualizes the graph

In [4] : `regr.fit(X, y)`

```
dot_data = tree.export_graphviz(regr,
                                out_file=None,
                                feature_names=df.columns[:-1],
                                class_names=None,
                                filled=True,
                                rounded=True,
                                special_characters=True)
```

```
graph = graphviz.Source(dot_data)
graph.render("cpu")
```

Out [4] : 'cpu.pdf'

Tree Visualization

In [4] : `regr.fit(X, y)`

Get the column names from the pandas dataframe

```
dot_data = tree.export_graphviz(regr,
                                 out_file=None,
                                 feature_names=df.columns[:-1],
                                 class_names=None,
                                 filled=True,
                                 rounded=True,
                                 special_characters=True)
```

```
graph = graphviz.Source(dot_data)
graph.render("cpu")
```

Out [4] : 'cpu.pdf'

Tree Visualization

Train the model using the full data

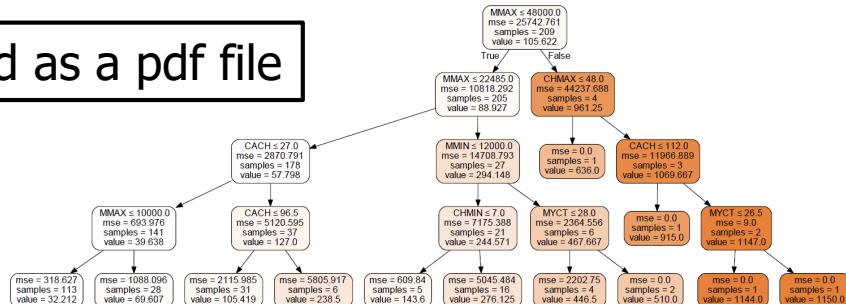
In [4] :

```
regr.fit(X, y)
dot_data = tree.export_graphviz(regr,
                                out_file=None,
                                feature_names=df.columns[:-1],
                                class_names=None,
                                filled=True,
                                rounded=True,
                                special_characters=True)

graph = graphviz.Source(dot_data)
graph.render("cpu")
```

Out [4] : 'cpu.pdf'

The image is saved as a pdf file

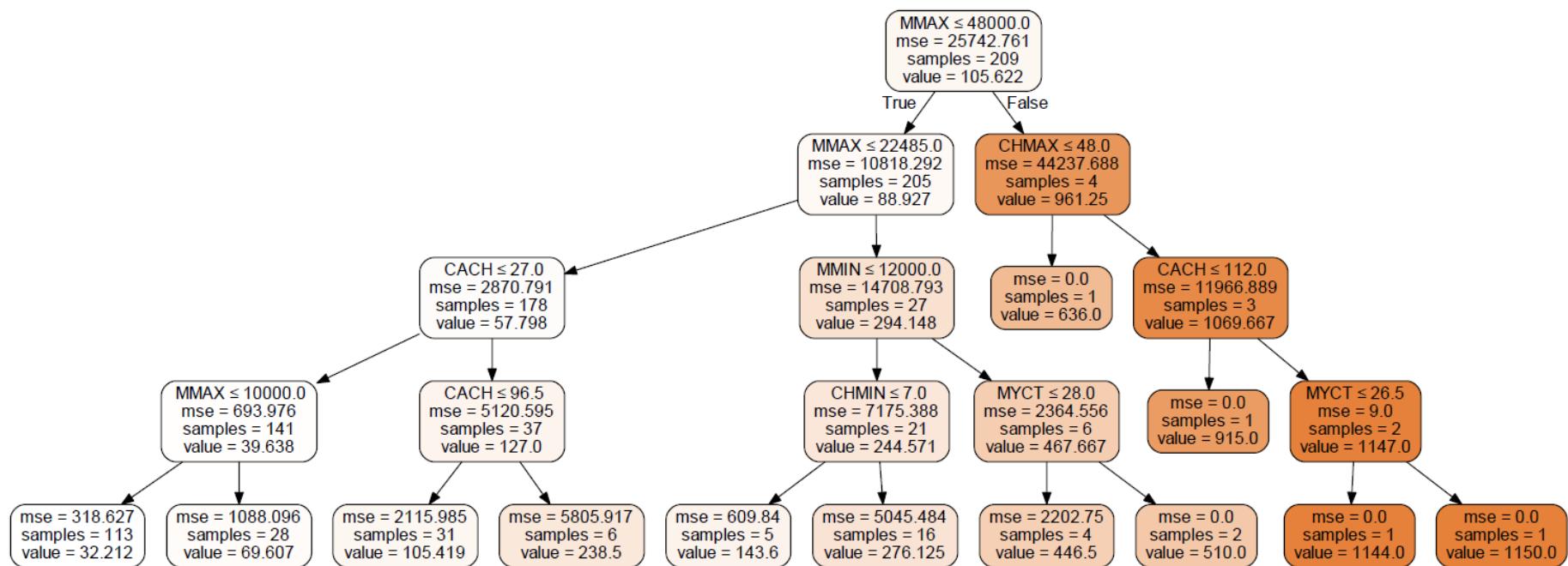


Tree Visualization

In [4] :

```
regr.fit(X, y)
dot_data = tree.export_graphviz(regr,
                                out_file=None,
                                feature_names=df.columns[:-1],
```

Train the model using the full data



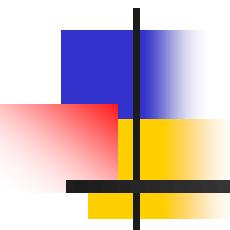
Prediction

```
In [5]: y_pred = regr.predict([[25, 8000, 64000, 64, 8, 32]])
print(y_pred)
```

```
[636.]
```

- Make a data instance and get the predicted label

MYCT	MMIN	MMAX	CACH	CHMIN	CHMAX	class
25	8000	64000	64	8	32	636(predicted)



Logistic Regression

Logistic Regression

- In linear regression, we predicted **numeric** values

$$x = w_0 + w_1 a_1 + w_2 a_2 + \dots + w_k a_k$$

- Then how to predict **categorical** values (classes)?
- Assume that the target class is binary
 - $y = +1$ or -1

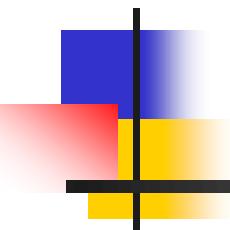
Logistic Regression

- Let a_i be the i -th instance and y_i be the target label
- We will predict the **probability** of each class
 - We will convert probability to some value that have range $[-\infty, \infty]$

$$p(y_i|a_i) = \frac{1}{1 + e^{-x_i y_i}} = \frac{1}{1 + e^{-y_i(w_0 + w_1 a_{i,1} + \dots + w_k a_{i,k})}}$$

- Find the weights w_0, \dots, w_k which maximize the log-likelihood function

$$L = \sum_{i=1}^N \log p(y_i|a_i)$$



Python – Logistic Regression

Download the Dataset

- Download vote.csv from <http://kdd.snu.ac.kr/python/>
- Save the csv file in the same directory as the source file (.ipynb)

Import Libraries

In [1]:

```
import numpy as np
import pandas as pd
from sklearn import preprocessing
from sklearn import linear_model, metrics
from sklearn.model_selection import cross_val_score
```

Import Libraries

```
In [1]: import numpy as np  
import pandas as pd  
from sklearn import preprocessing  
from sklearn import Linear_model, metrics  
from sklearn.model_selection import cross_val_score
```

- In this example, we have to convert categorical features to numeric form
- preprocessing module offers such conversions

Load Data

```
In [2]: df = pd.read_csv("vote.csv")
X = df.values[:, :-1]
y = df.values[:, -1]
df[:5]
```

Out [2]:

	handicapped-infants	water-project-cost-sharing	adoption-of-the-budget-resolution	physician-fee-freeze	el-salvador-aid	religious-groups-in-schools	anti-satellite-test-ban	aid-to-nicaraguan-contras
0	n	y	n	y	y	y	n	n
1	n	y	n	y	y	y	n	n
2	?	y	y	?	y	y	n	n
3	n	y	y	n	?	y	n	n
4	y	y	y	n	y	y	n	n

Load Data

mx-missile	immigration	synfuels-corporation-cutback	education-spending	superfund-right-to-sue	crime	duty-free-exports	export-administration-act-south-africa	Class
n	y	?	y	y	y	n	y	republican
n	n	n	y	y	y	n	?	republican
n	n	y	n	y	y	n	n	democrat
n	n	y	n	y	n	n	y	democrat
n	n	y	?	y	y	y	y	democrat

- The data has 16 features and 1 class label (17 columns)

Preprocessing

```
In [3]: enc = preprocessing.OneHotEncoder()
enc.fit(X)
enc_X = enc.transform(X).toarray()
print(X[0])
print(enc_X[0])
```

```
X[0]  ['n' 'y' 'n' 'y' 'y' 'y' 'n' 'n' 'n' 'y' '?' 'y' 'y' 'y' 'n' 'y']
enc_X[0] [0. 1. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 1. 0.
          0. 1. 0. 0. 0. 1. 1. 0. 0. 0. 1. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 0. 1.]
```

- Because the input features are categorical, perform one-hot encoding

Preprocessing

```
In [3]: enc = preprocessing.OneHotEncoder()
enc.fit(X)
enc_X = enc.transform(X).toarray()
print(X[0])
print(enc_X[0])
```

```
X[0]  ['n' 'y' 'n' 'y' 'y' 'y' 'n' 'n' 'n' 'y' '?' 'y' 'y' 'y' 'n' 'y']
enc_X[0] [0. 1. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 1. 0.
          0. 1. 0. 0. 0. 1. 1. 0. 0. 0. 1. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 0. 1.]
```

- Because the input features are categorical, perform one-hot encoding
- Each feature has 3 possible values ('y', 'n', '?')

Preprocessing

```
In [3]: enc = preprocessing.OneHotEncoder()
enc.fit(X)
enc_X = enc.transform(X).toarray()
print(X[0])
print(enc_X[0])
```

```
X[0]      ['n' 'y' 'n' 'y' 'y' 'y' 'n' 'n' 'n' 'y' '?' 'y' 'y' 'y' 'n' 'y']
enc_X[0]  [0. 1. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 1. 0.
           0. 1. 0. 0. 0. 1. 1. 0. 0. 0. 1. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 0. 1.]
```

- Because the input features are categorical, perform one-hot encoding
- Each feature has 3 possible values ('y', 'n', '?')
- Convert each value to 3 digit binary vector
 - 'y' => [0 0 1]
 - 'n' => [0 1 0]
 - '?' => [1 0 0]

Preprocessing

```
In [3]: enc = preprocessing.OneHotEncoder()  
enc.fit(X)  
enc_X = enc.transform(X).toarray()  
print(X[0])  
print(enc_X[0])
```

```
X[0]  ['n' 'y' 'n' 'y' 'y' 'y' 'n' 'n' 'n' 'y' '?' 'y' 'y' 'y' 'n' 'y']  
enc_X[0] [0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 1. 0. 0. 1. 0.  
          0. 1. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 1.]
```

- Because the input features are categorical, perform one-hot encoding
- Each feature has 3 possible values ('y', 'n', '?')
- Convert each value to 3 digit binary vector
 - 'y' => [0 0 1]
 - 'n' => [0 1 0]
 - '?' => [1 0 0]

Logistic Regression Classification Model

```
In [4]: clf = Linear_model.LogisticRegression(fit_intercept = True)
scores = cross_val_score(clf,
                        enc_X,
                        y,
                        cv = 10)
print(np.sqrt(scores).mean())
```

0.9823981597874688

- Create a Logistic Regression classifier model object
- Parameters:
 - fit_intercept : Whether to use constant term
 - solver : Which optimization algorithm to use
('newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga')
default : 'liblinear'

Logistic Regression Classification Model

```
In [4]: clf = LinearModel.LogisticRegression(fit_intercept = True)
scores = cross_val_score(clf,
                        enc_X,
                        y,
                        cv = 10)
print(np.sqrt(scores).mean())
```

0.9823981597874688

Not specifying the scoring function calls the default scoring function of the classifier

Logistic Regression Classification Model

```
In [4]: clf = LinearModel.LogisticRegression(fit_intercept = True)
scores = cross_val_score(clf,
                        enc_X,
                        y,
                        cv = 10)
print(np.sqrt(scores).mean())
```

0.9823981597874688

In this case, the score is the classification accuracy

Prediction

```
In [5]: clf.fit(enc_X, y)
sample = [['n', 'y', 'y', 'n', 'y', '?', 'n', 'n',
           'y', 'n', 'y', 'n', 'y', 'n', 'n', '?']]
y_pred = clf.predict(enc.transform(sample).toarray())
print(y_pred)
```

['democrat']

Don't forget to preprocess the test data

- Train the model and test with a data instance

handicapped-infants	water-project-cost-sharing	adoption-of-the-budget-resolution	physician-fee-freeze	el-salvador-aid	religious-groups-in-schools	anti-satellite-test-ban	aid-to-nicaraguan-contras
n	y	y	n	y	?	n	n
mx-missile	immigration	synfuels-corporation-cutback	education-spending	superfund-right-to-sue	crime	duty-free-exports	export-administration-act-south-africa
y	n	y	n	y	n	n	?

democrat(predicted)

Classification by Regression

- Can a regression scheme be used for classification? Yes!
 - Two-class problem
 - Training: call the classes 0 and 1
 - Prediction: set a threshold for predicting class 0 or 1
 - Multi-class problem: “multi-response linear regression”
 - Training: perform a regression for each class
 - *Set output to 1 for training instances that belong to the class, 0 for instances that don't*
 - Prediction: choose the class with the largest output
 - ... or use “pairwise linear regression”, which performs a regression for every pair of classes

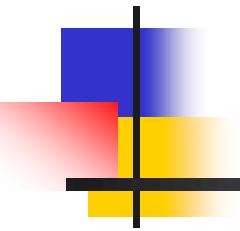
Classification by Regression

- Investigate two-class classification by regression
 - Open file **diabetes.arff**
 - Use attribute filter to convert to numeric
filters>unsupervised>attribute>NominalToBinary
 - but first set Class: class (Nom) to No class, because attribute filters do not operate on the class value
- Choose **functions>LinearRegression**
- Run
- Set **Output predictions** option

Classification by Regression

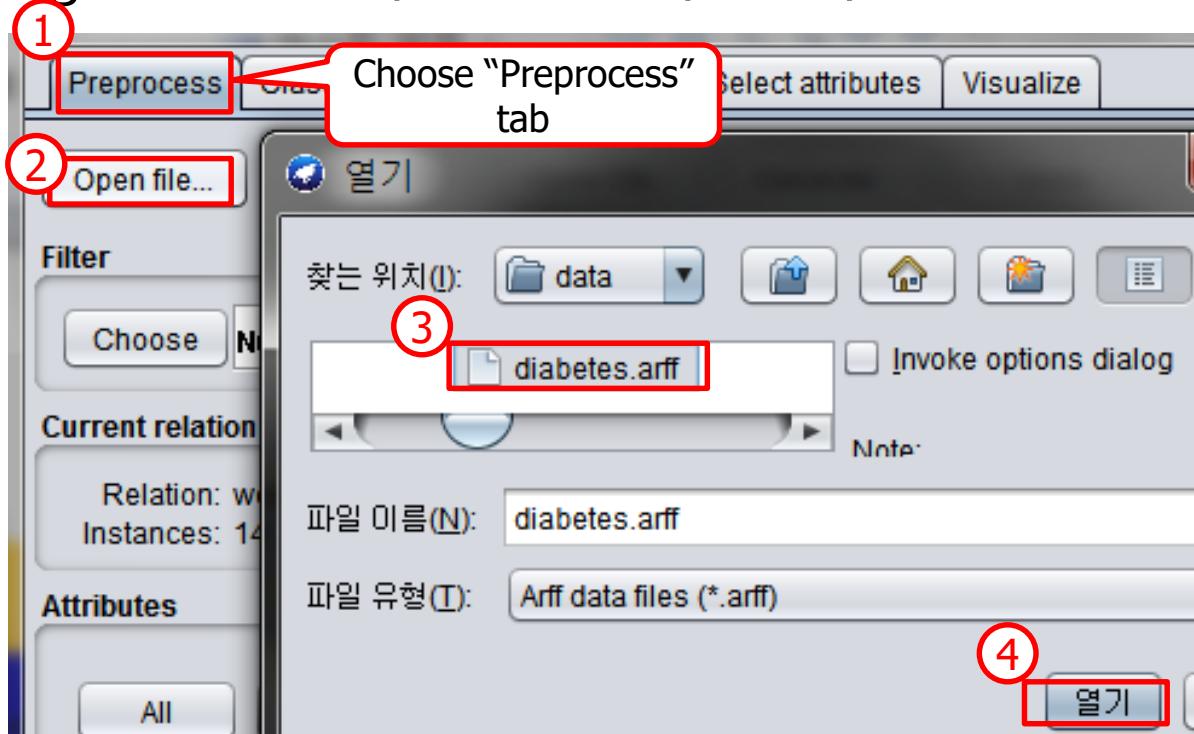
- Supervised attribute filter **AddClassification**
 - choose **functions>LinearRegression** as classifier
 - set **outputClassification** to true
 - Apply; adds new attribute called “**classification**”
- Convert **class** attribute back to nominal
 - unsupervised attribute filter **NumericToNominal**
 - set **attributeIndices** to 9
 - delete all the other attributes
- Classify panel
 - unset **Output predictions** option
 - change prediction from **(Num) classification** to **(Nom) class**
- Select **rules>OneR**; run it
 - rule is based on classification attribute, but it's complex
- Change **minBucketSize** parameter from 6 to 100
 - simpler rule (threshold 0.47) that performs quite well: 76.8%

Python – Classification by Regression

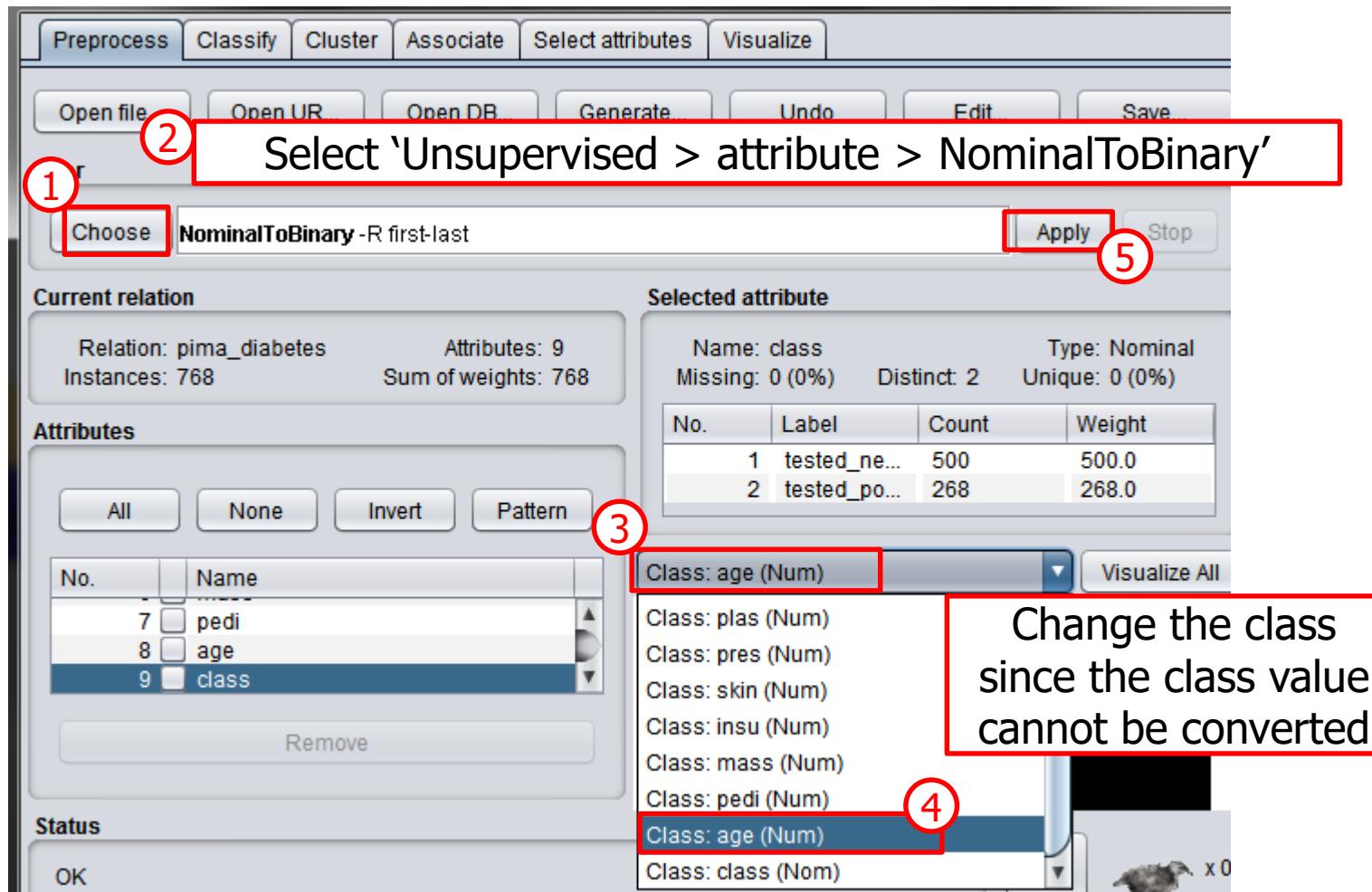


Open the Dataset

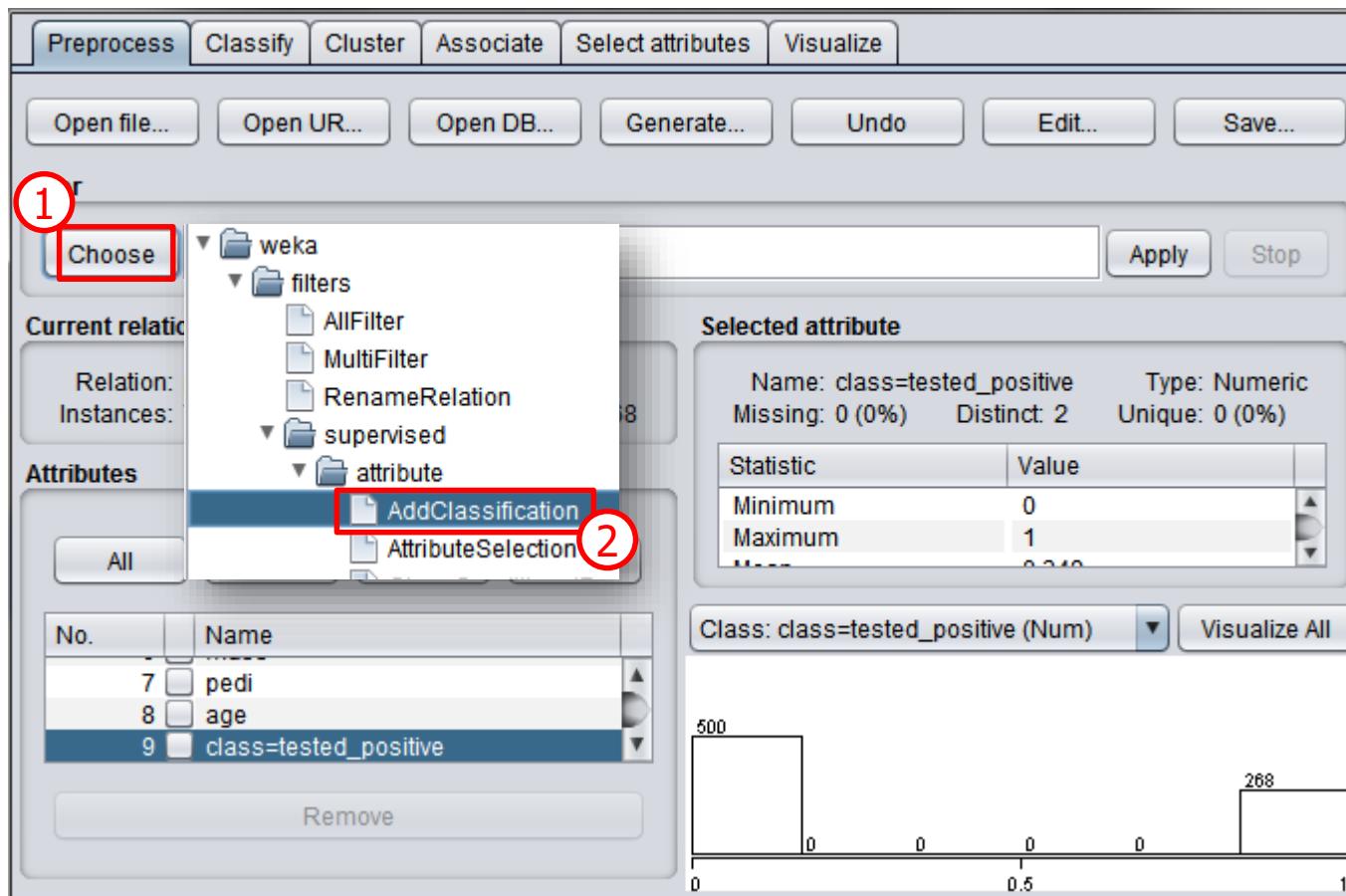
- C:\Program Files\Weka-3-8\data\diabetes.arff



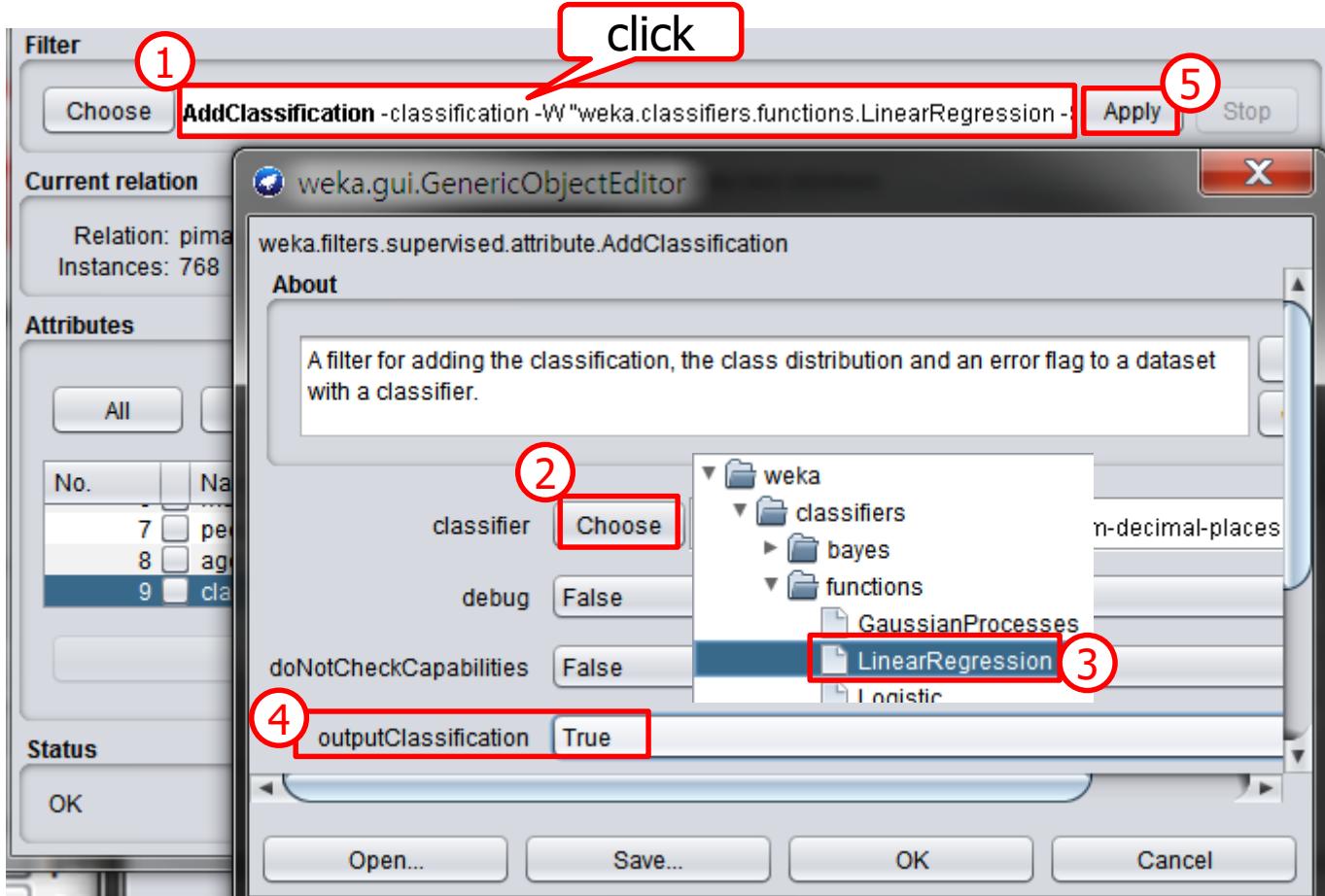
Convert to Numeric Value



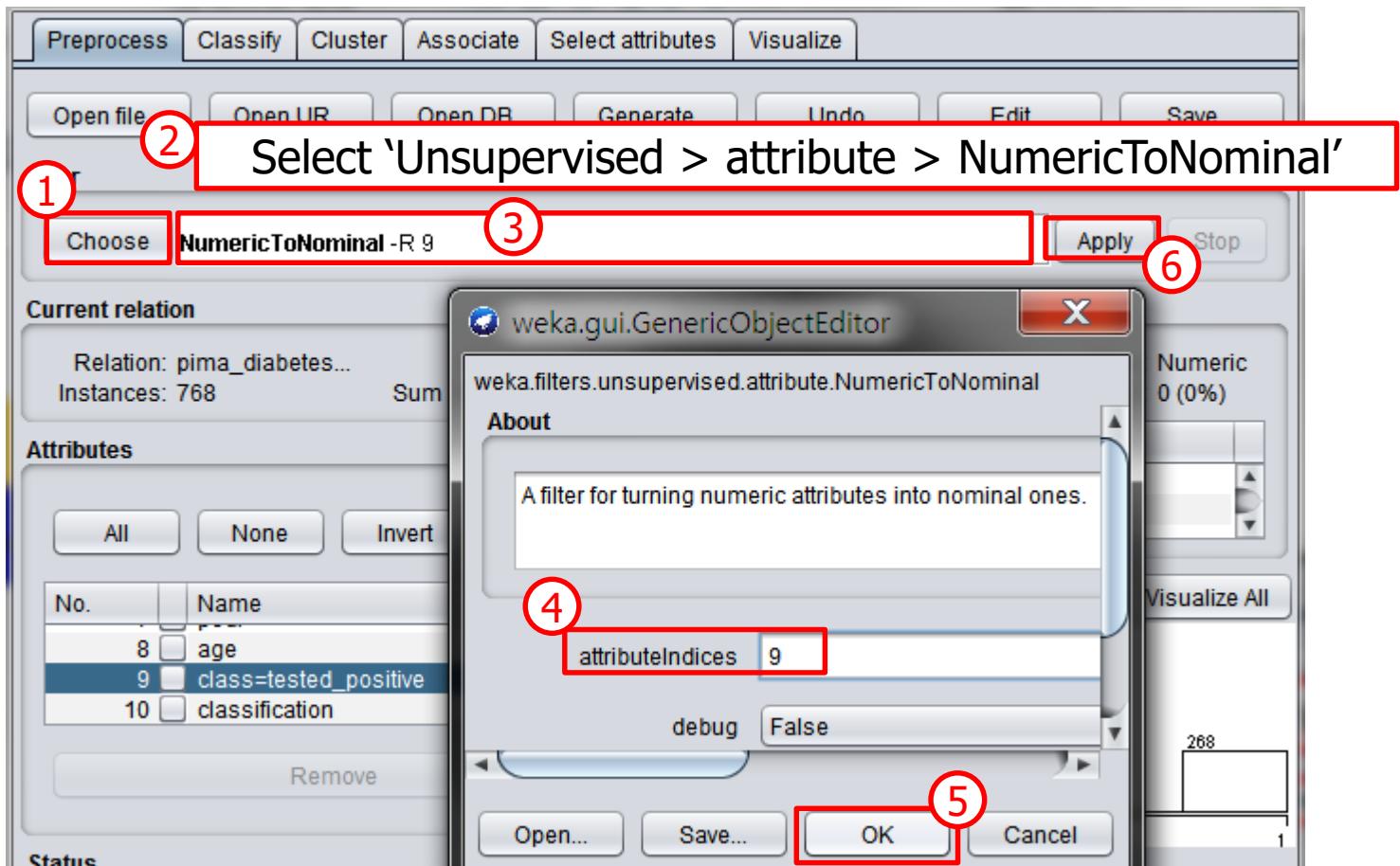
Linear Regression



Linear Regression



Numeric Value to Class



Apply Classifier

Current relation

Relation: pima_diabetes... Attributes: 10
Instances: 768 Sum of weights: 768

Attributes

Check all but 9th and 10th attributes

No. 1 preg
2 plas
3 pres
4 skin
5 insu
6 mass
7 pedi
8 age
9 class=tested_positive
10 classification

Selected attribute

Name: class=tested_positive Type: Nominal
Missing: 0 (0%) Distinct: 2 Unique: 0 (0%)

No.	Label	Count	Weight
1	0	500	500.0
2	1	268	268.0

Class: classification (Num) Visualize All

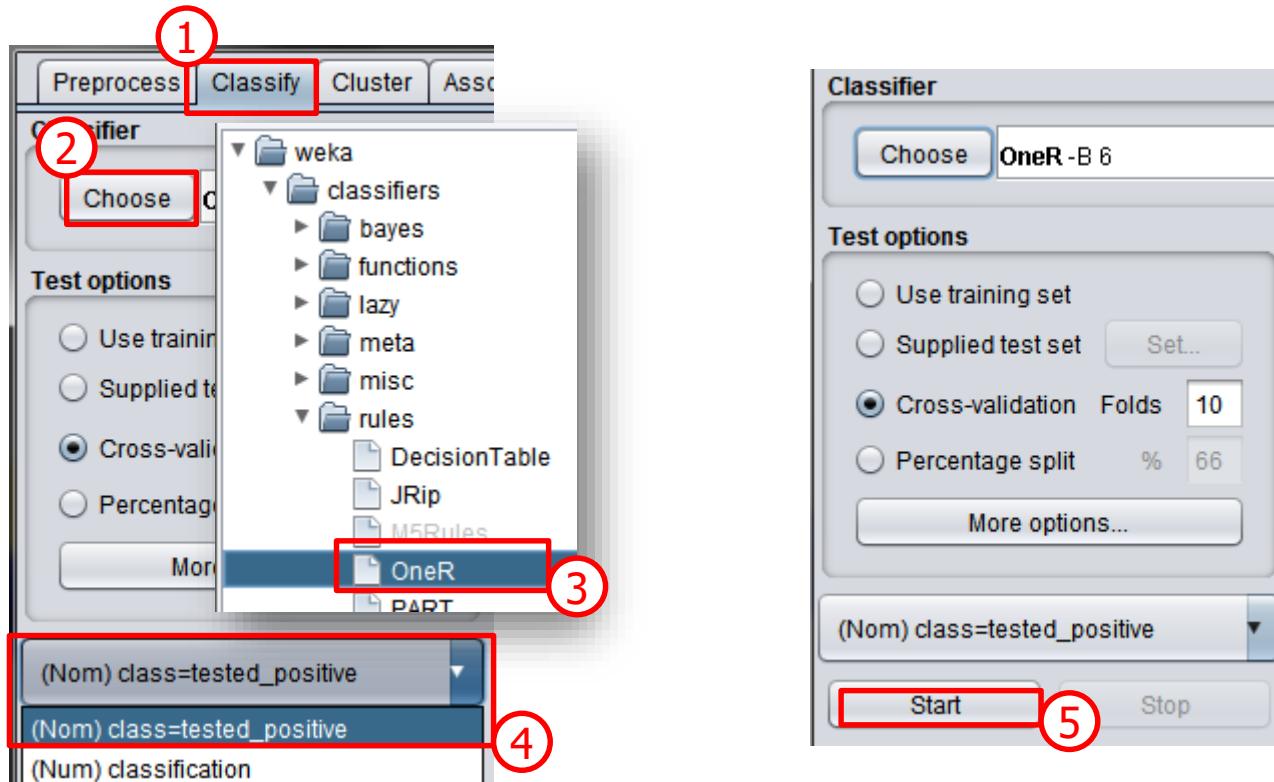
500

268

1 Remove

2 Remove

Apply Classifier



Apply Classifier

```
Classifier output

==== Classifier model (full training set) ====

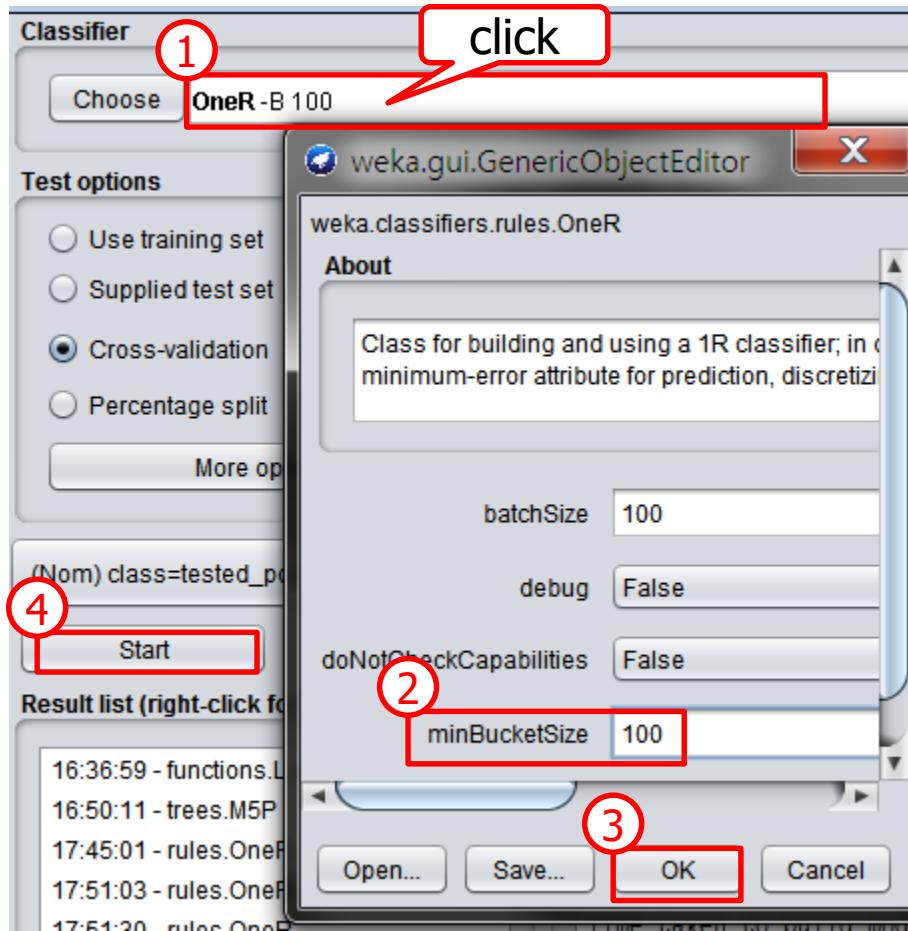
classification:
    < 0.2824512839034967    -> 0
    < 0.29382502646301367    -> 1
    < 0.37940870285866113    -> 0
    < 0.39140476756441367    -> 1
    < 0.39991515917441567    -> 0
    < 0.408357088899183      -> 1
    < 0.4196517257956418      -> 0
    < 0.45395701669502597    -> 1
    < 0.4701023336400427    -> 0
    < 0.48523272153598906    -> 1
    < 0.5078342313936239    -> 0
    < 0.6366653148084418    -> 1
    < 0.6591758248627334    -> 0
    >= 0.6591758248627334    -> 1
(615/768 instances correct)

Time taken to build model: 0 seconds

==== Stratified cross-validation ====
==== Summary ===

Correctly Classified Instances      560          72.9167 %
Incorrectly Classified Instances   208          27.0833 %
```

Change minBucketSize



minBucketSize=100

```
Classifier output

==== Classifier model (full training set) ====

classification:
    < 0.4701023336400427    -> 0
    >= 0.4701023336400427   -> 1
(595/768 instances correct)

Time taken to build model: 0 seconds

==== Stratified cross-validation ====
==== Summary ===

Correctly Classified Instances      590          76.8229 %
Incorrectly Classified Instances   178          23.1771 %
Kappa statistic                      0.459
```

Classification by Regression

■ Results

minBucketSize=6

```
classification:
  < 0.28245128390349705  -> 0
  < 0.29382502646301356  -> 1
  < 0.37940870285866113  -> 0
  < 0.3914047675644139   -> 1
  < 0.39991515917441545  -> 0
  < 0.4083570888991831   -> 1
  < 0.4196517257956417   -> 0
  < 0.45395701669502597  -> 1
  < 0.47010233364004295  -> 0
  < 0.48523272153598895  -> 1
  < 0.507834231393624   -> 0
  < 0.6366653148084416   -> 1
  < 0.6591758248627335  -> 0
  >= 0.6591758248627335 -> 1
(615/768 instances correct)
```

minBucketSize=100

```
classification:
  < 0.47010233364004295  -> 0
  >= 0.47010233364004295 -> 1
(595/768 instances correct)
```

Classification by Regression

■ Results

minBucketSize=6

```
== Stratified cross-validation ==
== Summary ==

Correctly Classified Instances      560          72.9167 %
Incorrectly Classified Instances   208          27.0833 %
Kappa statistic                      0.3945
Mean absolute error                  0.2708
Root mean squared error              0.5204
Relative absolute error              59.5885 %
Root relative squared error         109.1835 %
Total Number of Instances            768
```

Simpler model gives
better accuracy

minBucketSize=100

```
== Stratified cross-validation ==
== Summary ==

Correctly Classified Instances      590          76.8229 %
Incorrectly Classified Instances   178          23.1771 %
Kappa statistic                      0.459
Mean absolute error                  0.2318
Root mean squared error              0.4814
Relative absolute error              50.994 %
Root relative squared error         101.0033 %
Total Number of Instances            768
```

Classification by Regression

- Extend linear regression to classification
 - Easy with two classes
 - Else use multi-response linear regression, or pairwise linear regression
- Also learned about
 - Unsupervised attribute filter `NominalToBinary`, `NumericToNominal`
 - Supervised attribute filter `AddClassification`
 - Setting/unsetting the class
 - OneR's `minBucketSize` parameter
- But we can do better: Logistic regression

Chapter 8. Classification: Basic Concepts

- Classification: Basic Concepts
- Decision Tree Induction
- Bayes Classification Methods
- Rule-Based Classification
- Model Evaluation and Selection
- Techniques to Improve Classification Accuracy:
Ensemble Methods
- Summary



Bayesian Classification: Why?

- A statistical classifier: performs *probabilistic prediction*, i.e., predicts class membership probabilities
- Foundation: Based on Bayes' Theorem.
- Performance: A simple Bayesian classifier, *naïve Bayesian classifier*, has comparable performance with decision tree and selected neural network classifiers
- Incremental: Each training example can incrementally increase/decrease the probability that a hypothesis is correct — prior knowledge can be combined with observed data
- Standard: Even when Bayesian methods are computationally intractable, they can provide a standard of optimal decision making against which other methods can be measured

Bayes' Theorem: Basics

- Total probability Theorem: $P(B) = \sum_{i=1}^M P(B|A_i)P(A_i)$
- Bayes' Theorem: $P(H|\mathbf{X}) = \frac{P(\mathbf{X}|H)P(H)}{P(\mathbf{X})} = P(\mathbf{X}|H) \times P(H) / P(\mathbf{X})$
 - Let \mathbf{X} be a data sample ("evidence"): class label is unknown
 - Let H be a *hypothesis* that \mathbf{X} belongs to class C
 - Classification is to determine $P(H|\mathbf{X})$, (i.e., *posteriori probability*): the probability that the hypothesis holds given the observed data sample \mathbf{X}
 - $P(H)$ (*prior probability*): the initial probability
 - E.g., \mathbf{X} will buy computer, regardless of age, income, ...
 - $P(\mathbf{X})$: probability that sample data is observed
 - $P(\mathbf{X}|H)$ (*likelihood*): the probability of observing the sample \mathbf{X} , given that the hypothesis holds
 - E.g., Given that \mathbf{X} will buy computer, the prob. that \mathbf{X} is 31..40, medium income

Prediction Based on Bayes' Theorem

- Given training data \mathbf{X} , *posteriori probability of a hypothesis H*, $P(H|\mathbf{X})$, follows the Bayes' theorem

$$P(H|\mathbf{X}) = \frac{P(\mathbf{X}|H)P(H)}{P(\mathbf{X})} = P(\mathbf{X}|H) \times P(H) / P(\mathbf{X})$$

- Informally, this can be viewed as
posteriori = likelihood x prior/evidence
- Predicts \mathbf{X} belongs to C_i iff the probability $P(C_i|\mathbf{X})$ is the highest among all the $P(C_k|\mathbf{X})$ for all the k classes
- Practical difficulty: It requires initial knowledge of many probabilities, involving significant computational cost

Classification Is to Derive the Maximum Posteriori

- Let D be a training set of tuples and their associated class labels, and each tuple is represented by an n -D attribute vector $\mathbf{X} = (x_1, x_2, \dots, x_n)$
- Suppose there are m classes C_1, C_2, \dots, C_m .
- Classification is to derive the maximum posteriori, i.e., the maximal $P(C_i | \mathbf{X})$
- This can be derived from Bayes' theorem

$$P(C_i | \mathbf{X}) = \frac{P(\mathbf{X} | C_i) P(C_i)}{P(\mathbf{X})}$$

- Since $P(\mathbf{X})$ is constant for all classes, only

$$P(C_i | \mathbf{X}) = P(\mathbf{X} | C_i) P(C_i)$$

needs to be maximized

Naïve Bayes Classifier

- A simplified assumption: attributes are conditionally independent (i.e., no dependence relation between attributes):

$$P(\mathbf{X} | C_i) = \prod_{k=1}^n P(x_k | C_i) = P(x_1 | C_i) \times P(x_2 | C_i) \times \dots \times P(x_n | C_i)$$

- This greatly reduces the computation cost: Only counts the class distribution
- If A_k is categorical, $P(x_k | C_i)$ is the # of tuples in C_i having value x_k for A_k divided by $|C_{i,D}|$ (# of tuples of C_i in D)
- If A_k is continuous-valued, $P(x_k | C_i)$ is usually computed based on Gaussian distribution with a mean μ and standard deviation σ

and $P(x_k | C_i)$ is

$$g(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

$$P(\mathbf{X} | C_i) = g(x_k, \mu_{C_i}, \sigma_{C_i})$$

Naïve Bayes Classifier: Training Dataset

Class:

C1:buys_computer = 'yes'

C2:buys_computer = 'no'

Data to be classified:

X = (age <=30,

Income = medium,

Student = yes

Credit_rating = Fair)

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

Naïve Bayes Classifier: An Example

- $P(C_i)$: $P(\text{buys_computer} = \text{"yes"}) = 9/14 = 0.643$
 $P(\text{buys_computer} = \text{"no"}) = 5/14 = 0.357$

- Compute $P(X|C_i)$ for each class

$$P(\text{age} = \text{"<=30"} | \text{buys_computer} = \text{"yes"}) = 2/9 = 0.222$$

$$P(\text{age} = \text{"<= 30"} | \text{buys_computer} = \text{"no"}) = 3/5 = 0.6$$

$$P(\text{income} = \text{"medium"} | \text{buys_computer} = \text{"yes"}) = 4/9 = 0.444$$

$$P(\text{income} = \text{"medium"} | \text{buys_computer} = \text{"no"}) = 2/5 = 0.4$$

$$P(\text{student} = \text{"yes"} | \text{buys_computer} = \text{"yes"}) = 6/9 = 0.667$$

$$P(\text{student} = \text{"yes"} | \text{buys_computer} = \text{"no"}) = 1/5 = 0.2$$

$$P(\text{credit_rating} = \text{"fair"} | \text{buys_computer} = \text{"yes"}) = 6/9 = 0.667$$

$$P(\text{credit_rating} = \text{"fair"} | \text{buys_computer} = \text{"no"}) = 2/5 = 0.4$$

- $X = (\text{age} \leq 30, \text{income} = \text{medium}, \text{student} = \text{yes}, \text{credit_rating} = \text{fair})$

$$P(X|C_i) : P(X|\text{buys_computer} = \text{"yes"}) = 0.222 \times 0.444 \times 0.667 \times 0.667 = 0.044$$

$$P(X|\text{buys_computer} = \text{"no"}) = 0.6 \times 0.4 \times 0.2 \times 0.4 = 0.019$$

$$P(X|C_i) * P(C_i) : P(X|\text{buys_computer} = \text{"yes"}) * P(\text{buys_computer} = \text{"yes"}) = 0.028$$

$$P(X|\text{buys_computer} = \text{"no"}) * P(\text{buys_computer} = \text{"no"}) = 0.007$$

Therefore, X belongs to class ("buys_computer = yes")

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

Avoiding the Zero-Probability Problem

- Naïve Bayesian prediction requires each conditional prob. be **non-zero**. Otherwise, the predicted prob. will be zero

$$P(X | C_i) = \prod_{k=1}^n P(x_k | C_i)$$

- Ex. Suppose a dataset with 1000 tuples, income=low (0), income= medium (990), and income = high (10)
- Use **Laplacian correction** (or Laplacian estimator)

- *Adding 1 to each case*

$$\text{Prob(income = low)} = 1/1003$$

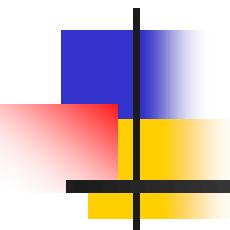
$$\text{Prob(income = medium)} = 991/1003$$

$$\text{Prob(income = high)} = 11/1003$$

- The “corrected” prob. estimates are close to their “uncorrected” counterparts

Naïve Bayes Classifier: Comments

- Advantages
 - Easy to implement
 - Good results obtained in most of the cases
- Disadvantages
 - Assumption: class conditional independence, therefore loss of accuracy
 - Practically, dependencies exist among variables
 - E.g., hospitals: patients: Profile: age, family history, etc.
Symptoms: fever, cough etc., Disease: lung cancer, diabetes, etc.
 - Dependencies among these cannot be modeled by Naïve Bayes Classifier
- How to deal with these dependencies? Bayesian Belief Networks (Chapter 9)



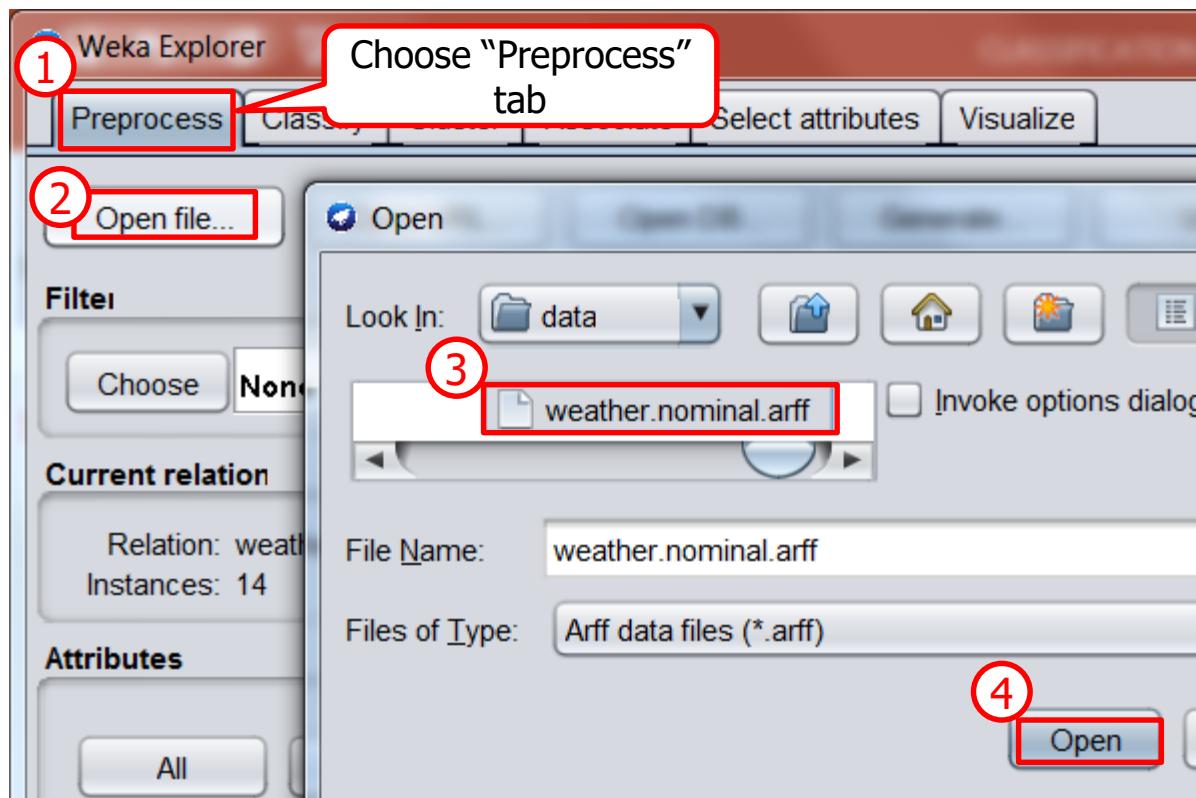
Weka – Use Naïve Bayes

Use Naïve Bayes

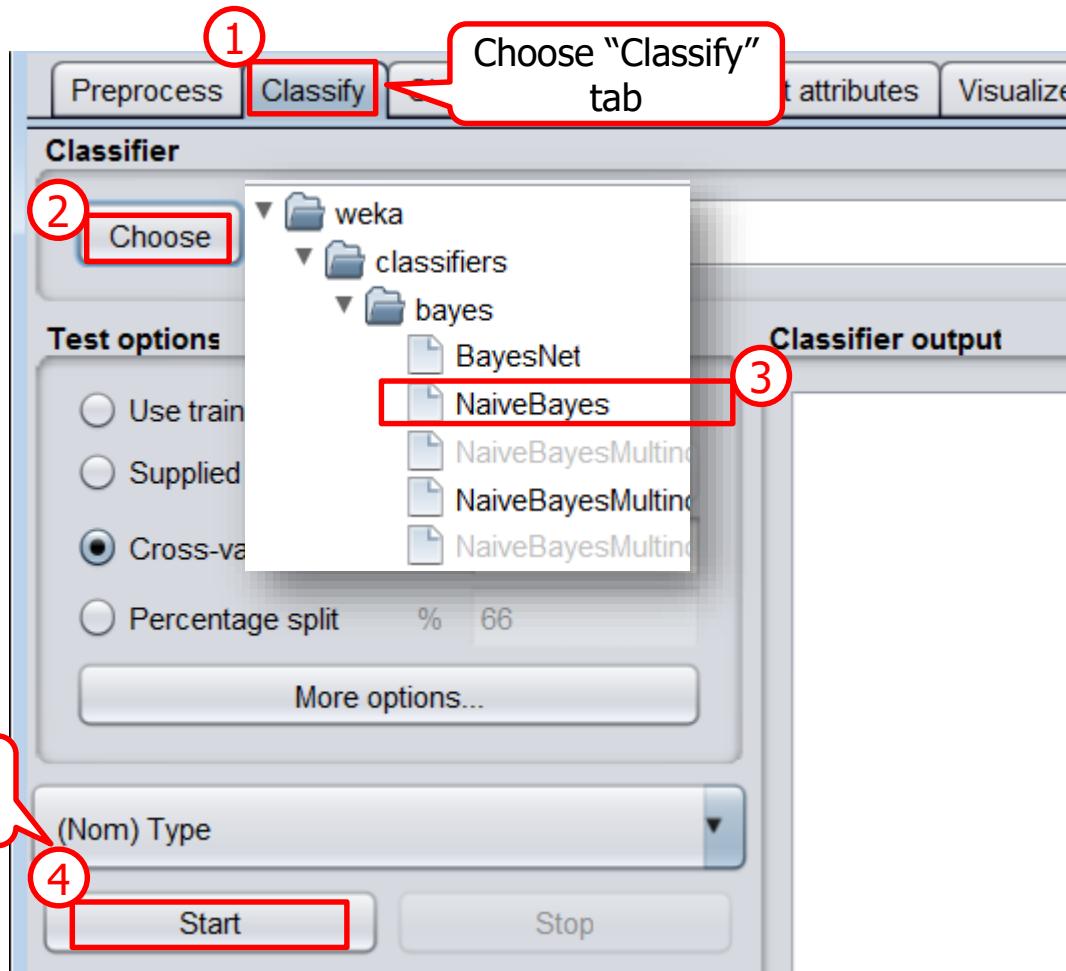
- Open file **weather.nominal.arff**
- Choose Naïve Bayes method (**bayes>NaiveBayes**)
- Look at the classifier
- Avoid zero frequencies: start all counts at 1

Open the Dataset

- C:\Program Files\Weka-3-8\data\weather.nominal.arff



Select the Classifier



Classifier output

TIME taken to build model: 0 seconds

==== Stratified cross-validation ====

==== Summary ====

Correctly Classified Instances	8
Incorrectly Classified Instances	6
Kappa statistic	-0.0244
Mean absolute error	0.4374
Root mean squared error	0.4916
Relative absolute error	91.8631 %
Root relative squared error	99.6492 %
Total Number of Instances	14

57.1429 %

42.8571 %

==== Detailed Accuracy By Class ====

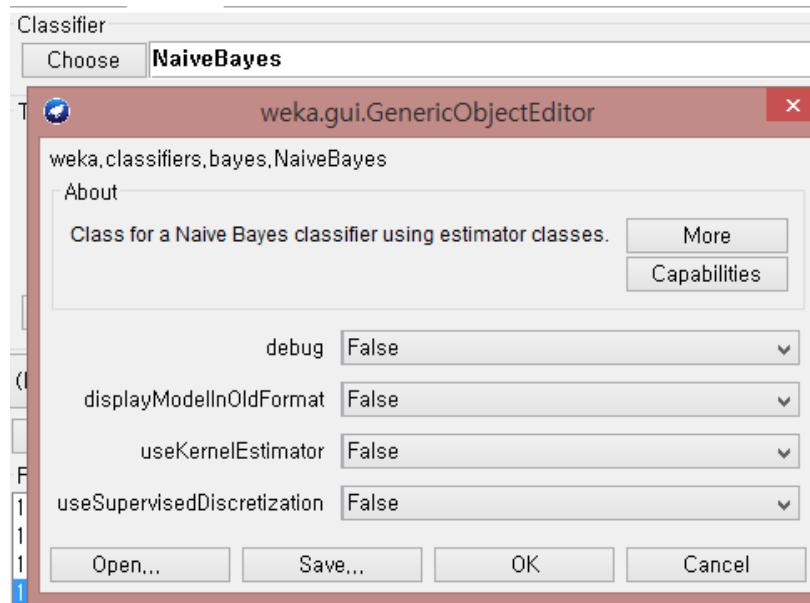
	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC
0	0.778	0.800	0.636	0.778	0.700	-0.0
1	0.200	0.222	0.333	0.200	0.250	-0.0
Weighted Avg.	0.571	0.594	0.528	0.571	0.539	-0.0

==== Confusion Matrix ====

a	b	<-- classified as
7	2	a = yes
4	1	b = no

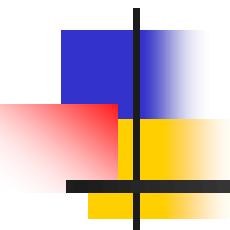
Parameters for NaiveBayesian

- **useKernelEstimator**
 - Use Gaussian kernel estimator rather than a normal distribution for numeric attributes
- **useSupervisedDiscretization**
 - Convert numeric attributes to nominal with MDL principle



Naïve Bayesian Classifier

- “Naïve Bayes”: all attributes contribute equally and independently
- Works surprisingly well
 - Even if independence assumption is clearly violated
- Why?
 - Classification doesn’t need accurate probability estimates *so long as the greatest probability is assigned to the correct class*
- Adding redundant attributes causes problems
 - (e.g. identical attributes) -> *attribute selection*



Python – Use Naïve Bayes

Download the Dataset

- Download weather.nominal.csv from
<http://kdd.snu.ac.kr/python/>
- Save the csv file in the same directory as the source file (.ipynb)

Import Libraries

```
import pandas as pd
from sklearn.model_selection import \
    KFold, cross_val_score
```

Class and function for K-fold corss validation

Import the Dataset

```
df = pd.read_csv('weather.nominal.csv')  
df[:3]
```

	outlook	temperature	humidity	windy	play
0	sunny	hot	high	False	no
1	sunny	hot	high	True	no
2	overcast	hot	high	False	yes

Convert to Arrays

```
X_str = df.values[:, :-1]
y = df.values[:, -1]
print(X_str[:3])
print(y[:3])
```

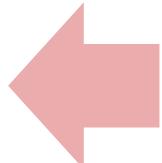
```
[['sunny' 'hot' 'high' False]
 ['sunny' 'hot' 'high' True]
 ['overcast' 'hot' 'high' False]
 ['no' 'no' 'yes']
```

Ordinal Encoding

- Encode categorical string features as an integer array

```
from sklearn.preprocessing import OrdinalEncoder  
enc = OrdinalEncoder() — Create an encoder object  
enc.fit(X_str) — Fit to the data  
X = enc.transform(X_str) — Encode the string features  
print(X[:3])
```

```
[ [2. 1. 0. 0.]  
[2. 1. 0. 1.]  
[0. 1. 0. 0.]]
```



```
print(X_str[:3])
```

```
[['sunny' 'hot' 'high' False]  
['sunny' 'hot' 'high' True]  
['overcast' 'hot' 'high' False]]
```

Naïve Bayesian Classifier

```
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB() —— Create the classifier object
cv = KFold( —— Create an K-Folds cross-validator object
            n_splits=10, —— The number of folds
            shuffle=True, —— Whether to shuffle the data before splitting
            random_state=0) —— The random seed
scores = cross_val_score( —— Compute accuracies for K-folds
                         clf, —— The object of the naïve bayes classifier
                         X, —> The features and labels
                         y,
                         cv=cv)
print(scores.mean()) —— Compute the average accuracy
```

0 . 65

Test the Classifier

Train the classifier for all data

```
clf.fit(X, y)
```

Prepare a test data

```
test_X_str = [  
    ['sunny', 'hot', 'high', False]]
```

```
test_X = enc.transform(test_X_str)
```

```
print(clf.predict(test_X))
```

```
[ 'no' ]
```

Encode the test data

Predict the label for the test data

Parameters

- `sklearn.naive_bayes.MultinomialNB`
 - alpha: smoothing parameter

$$\Pr(x_i|y) = \frac{N_{yi} + \alpha}{N_y + \alpha n}$$

- Accounts for features not present in the learning samples and prevents zero probabilities in further computations
- n : the number of features
- N_{yi} : the number of times that the i -th feature appears in a sample of class y
- $N_y = \sum_{i=1}^n N_{yi}$: the total number of samples in class y

Parameters

- `sklearn.naive_bayes.MultinomialNB`
 - `fit_prior`: Boolean (default=True)
 - If false, a uniform prior will be used

Chapter 8. Classification: Basic Concepts

- Classification: Basic Concepts
- Decision Tree Induction
- Bayes Classification Methods
- Rule-Based Classification
- Model Evaluation and Selection
- Techniques to Improve Classification Accuracy:
Ensemble Methods
- Summary

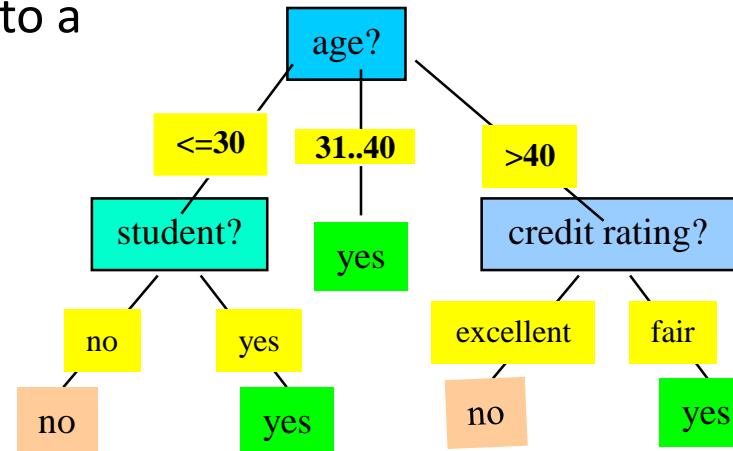


Using IF-THEN Rules for Classification

- Represent the knowledge in the form of **IF-THEN** rules
 - R: IF *age* = youth AND *student* = yes THEN *buys_computer* = yes
 - Rule antecedent/precondition vs. rule consequent
- Assessment of a rule: *coverage* and *accuracy*
 - $n_{\text{covers}} = \# \text{ of tuples covered by } R$
 - $n_{\text{correct}} = \# \text{ of tuples correctly classified by } R$
$$\text{coverage}(R) = n_{\text{covers}} / |D| \quad /* D: training data set */$$
$$\text{accuracy}(R) = n_{\text{correct}} / n_{\text{covers}}$$
- If more than one rule are triggered, need **conflict resolution**
 - Size ordering: assign the highest priority to the triggering rules that has the “toughest” requirement (i.e., with the *most attribute tests*)
 - Class-based ordering: decreasing order of *prevalence or misclassification cost per class*
 - Rule-based ordering (**decision list**): rules are organized into one long priority list, according to some measure of rule quality or by experts

Rule Extraction from a Decision Tree

- Rules are *easier to understand* than large trees
- One rule is created *for each path* from the root to a leaf
- Each attribute-value pair along a path forms a conjunction: the leaf holds the class prediction
- Rules are mutually exclusive and exhaustive



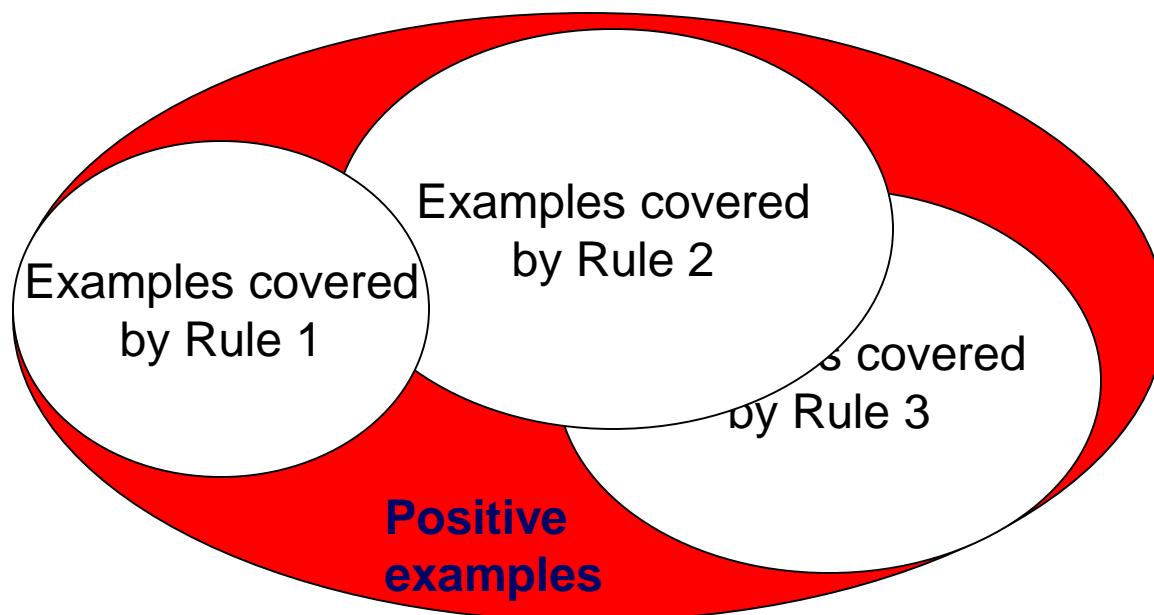
- Example: Rule extraction from our *buys_computer* decision-tree
- | | |
|--|---------------------------------|
| IF <i>age</i> = young AND <i>student</i> = no | THEN <i>buys_computer</i> = no |
| IF <i>age</i> = young AND <i>student</i> = yes | THEN <i>buys_computer</i> = yes |
| IF <i>age</i> = mid-age | THEN <i>buys_computer</i> = yes |
| IF <i>age</i> = old AND <i>credit_rating</i> = excellent | THEN <i>buys_computer</i> = no |
| IF <i>age</i> = old AND <i>credit_rating</i> = fair | THEN <i>buys_computer</i> = yes |

Rule Induction: Sequential Covering Method

- Sequential covering algorithm: Extracts rules directly from training data
- Typical sequential covering algorithms: FOIL, AQ, CN2, RIPPER
- Rules are learned *sequentially*, each for a given class C_i will cover many tuples of C_i but none (or few) of the tuples of other classes
- Steps:
 - Rules are learned one at a time
 - Each time a rule is learned, the tuples covered by the rules are removed
 - Repeat the process 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
- Comp. w. decision-tree induction: learning a set of rules *simultaneously*

Sequential Covering Algorithm

```
while (enough target tuples left)
    generate a rule
    remove positive target tuples satisfying this rule
```



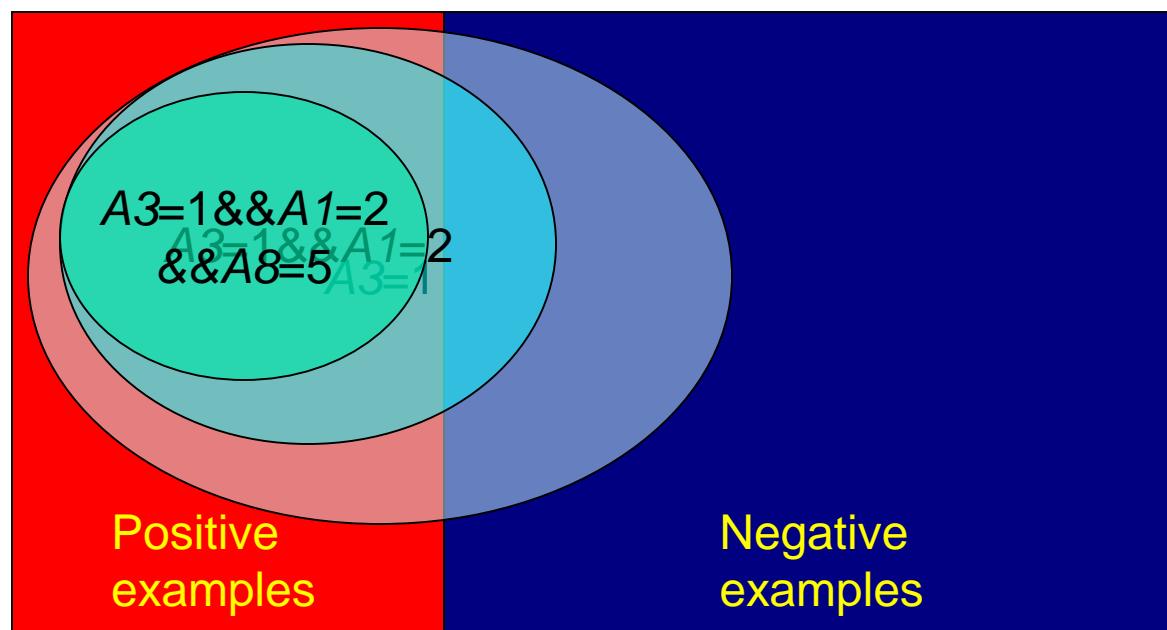
Rule Generation

- To generate a rule

while(true)

 find the best predicate p

if foil-gain(p) > threshold **then** add p to current rule
 else break



How to Learn-One-Rule?

- Start with the *most general rule* possible: condition = empty
- *Adding new attributes* by adopting a greedy depth-first strategy
 - Picks the one that most improves the rule quality
- Rule-Quality measures: consider both coverage and accuracy
 - Foil-gain (in FOIL & RIPPER): assesses `info_gain` by extending condition
 - favors rules that have high accuracy and cover many positive tuples
- Rule pruning based on an independent set of test tuples

$$FOIL_Gain = pos' \times \left(\log_2 \frac{pos'}{pos'+neg'} - \log_2 \frac{pos}{pos+neg} \right)$$

- favors rules that have high accuracy and cover many positive tuples

$$FOIL_Prune(R) = \frac{pos - neg}{pos + neg}$$

Pos/neg are # of positive/negative tuples covered by R.

If $FOIL_Prune$ is higher for the pruned version of R, prune R

Chapter 8. Classification: Basic Concepts

- Classification: Basic Concepts
- Decision Tree Induction
- Bayes Classification Methods
- Rule-Based Classification
- Model Evaluation and Selection
- Techniques to Improve Classification Accuracy:
Ensemble Methods
- Summary



Model Evaluation and Selection

- Evaluation metrics: How can we measure accuracy? Other metrics to consider?
- Use **validation test set** of class-labeled tuples instead of training set when assessing accuracy
- Methods for estimating a classifier's accuracy:
 - Holdout method, random subsampling
 - Cross-validation
 - Bootstrap
- Comparing classifiers:
 - Confidence intervals
 - Cost-benefit analysis and ROC Curves

Classifier Evaluation Metrics: Confusion Matrix

Confusion Matrix:

Actual class\Predicted class	C_1	$\neg C_1$
C_1	True Positives (TP)	False Negatives (FN)
$\neg C_1$	False Positives (FP)	True Negatives (TN)

Example of Confusion Matrix:

Actual class\Predicted class	buy_computer = yes	buy_computer = no	Total
buy_computer = yes	6954	46	7000
buy_computer = no	412	2588	3000
Total	7366	2634	10000

- Given m classes, an entry, $CM_{i,j}$ in a **confusion matrix** indicates # of tuples in class i that were labeled by the classifier as class j
- May have extra rows/columns to provide totals

Classifier Evaluation Metrics: Accuracy, Error Rate, Sensitivity and Specificity

A\P	C	$\neg C$	
C	TP	FN	P
$\neg C$	FP	TN	N
	P'	N'	All

- **Classifier Accuracy**, or recognition rate: percentage of test set tuples that are correctly classified

$$\text{Accuracy} = (\text{TP} + \text{TN})/\text{All}$$

- **Error rate**: $1 - \text{accuracy}$, or
$$\text{Error rate} = (\text{FP} + \text{FN})/\text{All}$$

- **Class Imbalance Problem**:
 - One class may be *rare*, e.g. fraud, or HIV-positive
 - Significant *majority of the negative class* and minority of the positive class
- **Sensitivity**: True Positive recognition rate
 - $\text{Sensitivity} = \text{TP}/\text{P}$
- **Specificity**: True Negative recognition rate
 - $\text{Specificity} = \text{TN}/\text{N}$

Classifier Evaluation Metrics: Precision and Recall, and F-measures

- **Precision:** exactness – what % of tuples that the classifier labeled as positive are actually positive

$$precision = \frac{TP}{TP + FP}$$

- **Recall:** completeness – what % of positive tuples did the classifier label as positive?

$$recall = \frac{TP}{TP + FN}$$

- Perfect score is 1.0
- Inverse relationship between precision & recall
- **F measure (F_1 or F-score):** harmonic mean of precision and recall,

$$F = \frac{2 \times precision \times recall}{precision + recall}$$

- F_β : weighted measure of precision and recall
 - assigns β times as much weight to recall as to precision

$$F_\beta = \frac{(1 + \beta^2) \times precision \times recall}{\beta^2 \times precision + recall}$$

Classifier Evaluation Metrics: Example

Actual Class\Predicted class	cancer = yes	cancer = no	Total	Recognition(%)
cancer = yes	90	210	300	30.00 (<i>sensitivity</i>)
cancer = no	140	9560	9700	98.56 (<i>specificity</i>)
Total	230	9770	10000	96.40 (<i>accuracy</i>)

- $Precision = 90/230 = 39.13\%$ $Recall = 90/300 = 30.00\%$

Evaluating Classifier Accuracy: Holdout & Cross-Validation Methods

- **Holdout method**
 - Given data is randomly partitioned into two independent sets
 - Training set (e.g., 2/3) for model construction
 - Test set (e.g., 1/3) for accuracy estimation
 - Random sampling: a variation of holdout
 - Repeat holdout k times, accuracy = avg. of the accuracies obtained
- **Cross-validation (k -fold, where $k = 10$ is most popular)**
 - Randomly partition the data into k *mutually exclusive* subsets, each approximately equal size
 - At i -th iteration, use D_i as test set and others as training set
 - Leave-one-out: k folds where $k = \#$ of tuples, for small sized data
 - ***Stratified cross-validation***: folds are stratified so that class dist. in each fold is approx. the same as that in the initial data

Evaluating Classifier Accuracy: Bootstrap

- **Bootstrap**
 - Works well with small data sets
 - Samples the given training tuples uniformly *with replacement*
 - i.e., each time a tuple is selected, it is equally likely to be selected again and re-added to the training set
- Several bootstrap methods, and a common one is **.632 bootstrap**
 - A data set with d tuples is sampled d times, with replacement, resulting in a training set of d samples. The data tuples that did not make it into the training set end up forming the test set. About 63.2% of the original data end up in the bootstrap, and the remaining 36.8% form the test set (since $(1 - 1/d)^d \approx e^{-1} = 0.368$)
 - Repeat the sampling procedure k times, overall accuracy of the model:

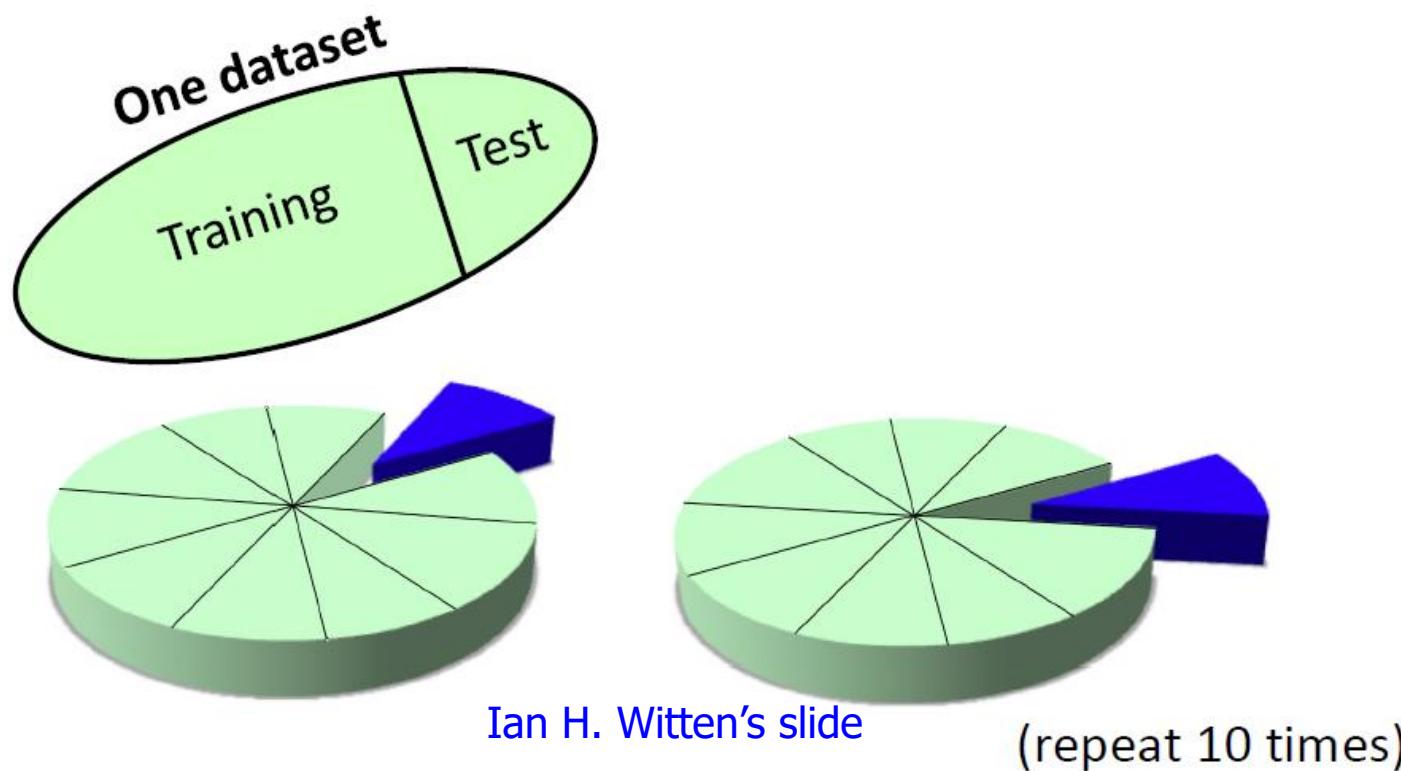
$$Acc(M) = \frac{1}{k} \sum_{i=1}^k (0.632 \times Acc(M_i)_{test_set} + 0.368 \times Acc(M_i)_{train_set})$$

Cross-validation

- Can we improve upon repeated holdout?
 - (i.e. reduce variance)
- Cross-validation
- Stratified cross-validation

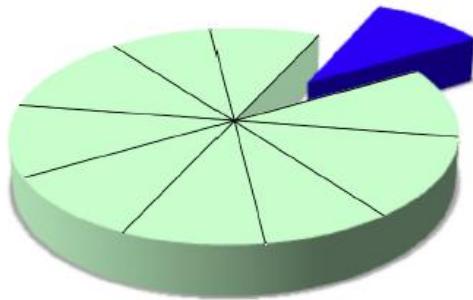
Cross-validation

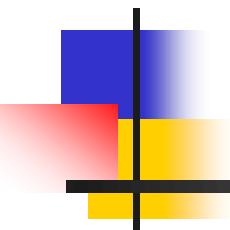
- Repeated holdout
 - (hold out 10% for testing, repeat 10 times)



Cross-validation

- 10-fold cross-validation
 - Divide dataset into 10 parts (folds)
 - Hold out each part in turn
 - Average the results
 - Each data point used once for testing, 9 times for training
- *Stratified* cross-validation
 - Ensure that each fold has the right proportion of each class value

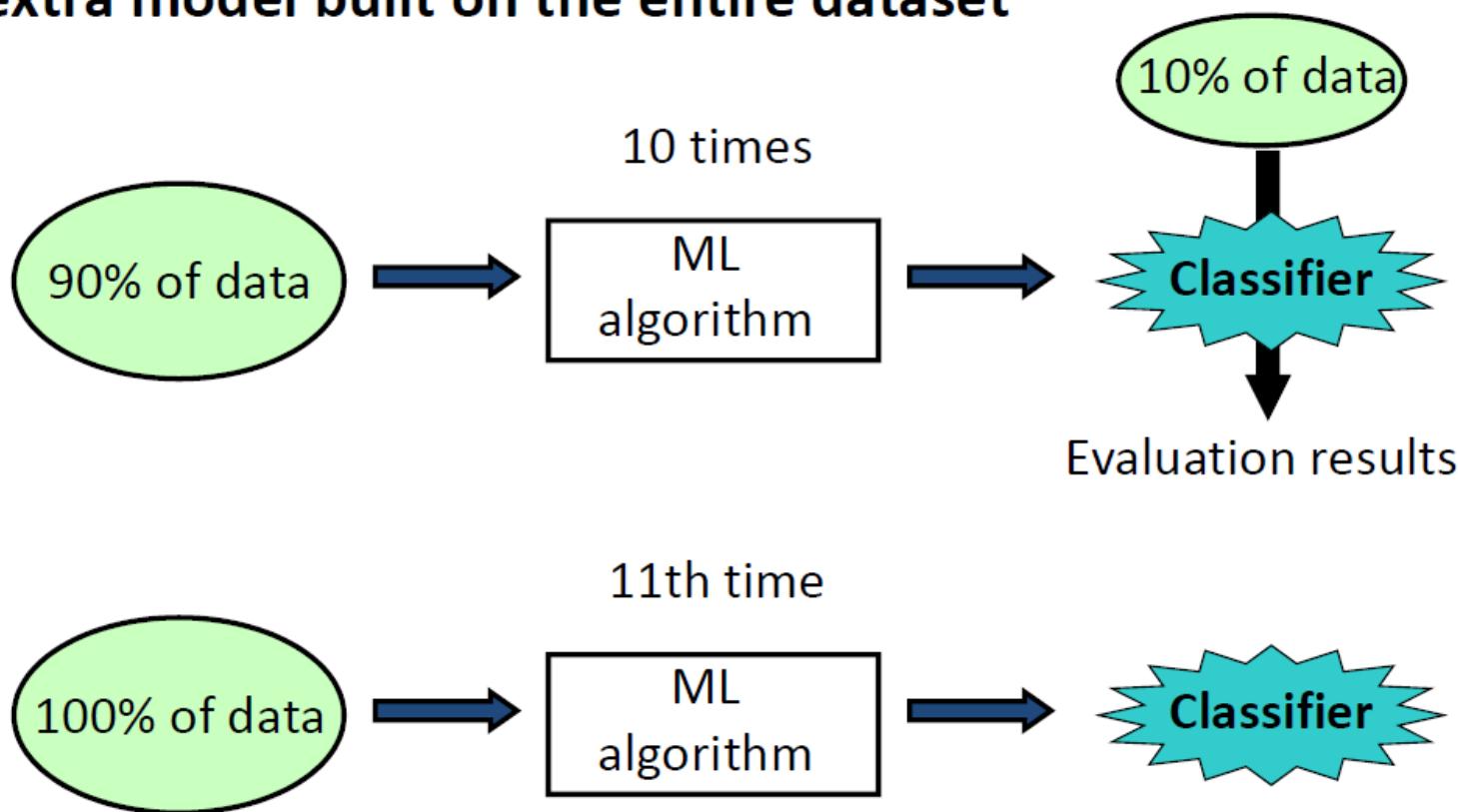




Weka – Cross-validation

Cross-validation

After cross-validation, Weka outputs an extra model built on the entire dataset



Cross-validation

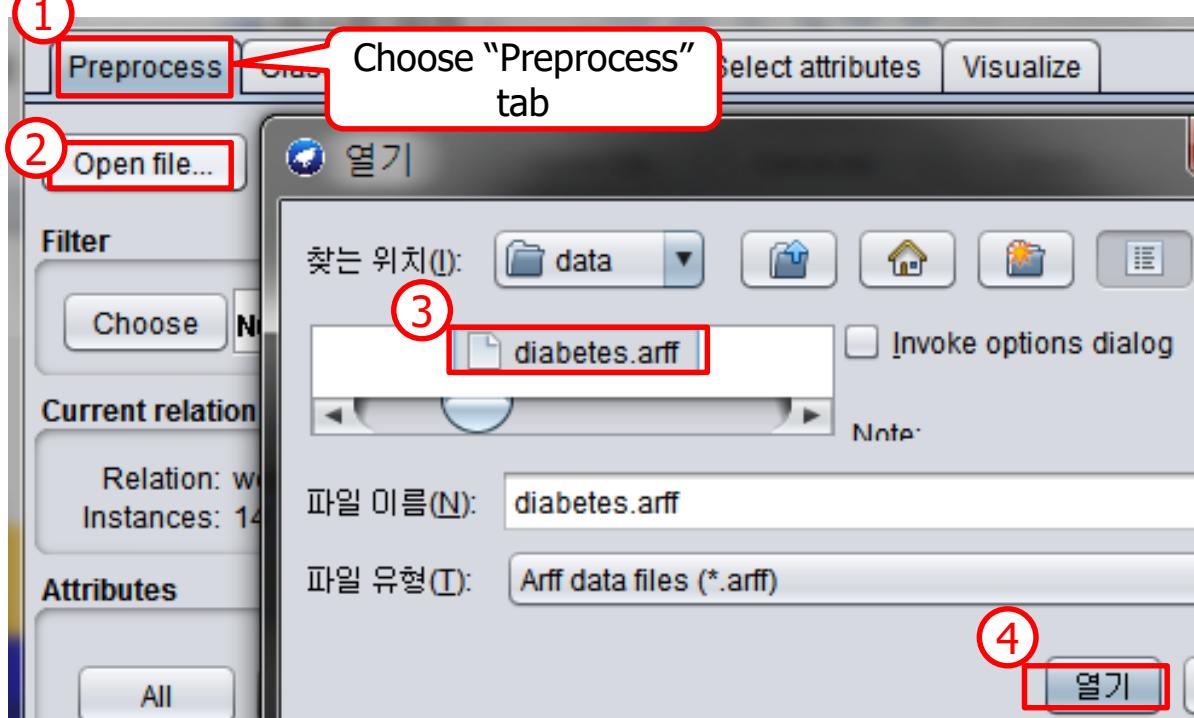
- Cross-validation better than repeated holdout
- Stratified is even better
- With 10-fold cross-validation, Weka invokes the learning algorithm 11 times
- **Practical rule of thumb:**
 - Lots of data? – use percentage split
 - Else stratified 10-fold cross-validation

Cross-validation Results

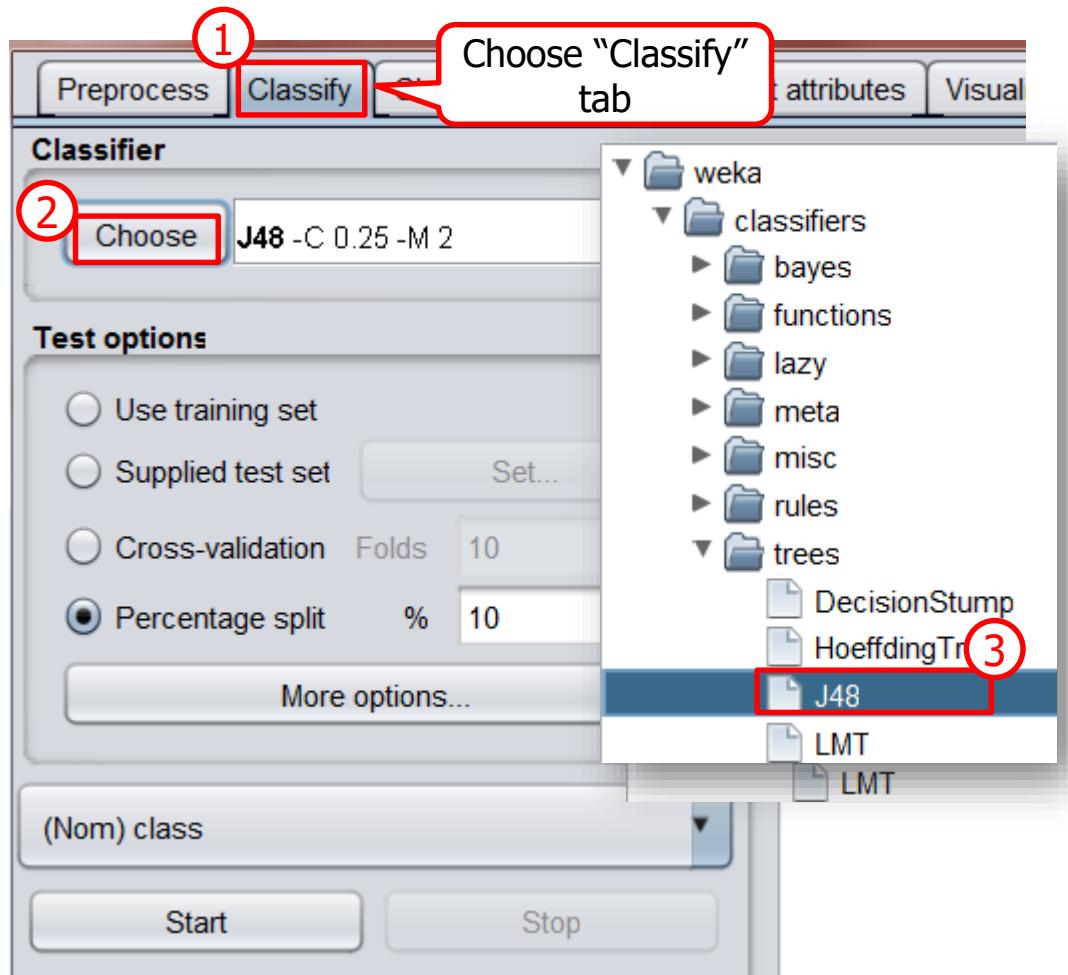
- Is cross-validation really better than repeated holdout?
 - Diabetes dataset
 - Baseline accuracy (rules > ZeroR): 65.1%
 - trees > J48
 - 10-fold cross-validation 73.8%
 - ... with different random number seed
 - 1 2 3 4 5 6 7 8 9 10
 - 73.8 75.0 75.5 75.5 74.4 75.6 73.6 74.0 74.5 73.0

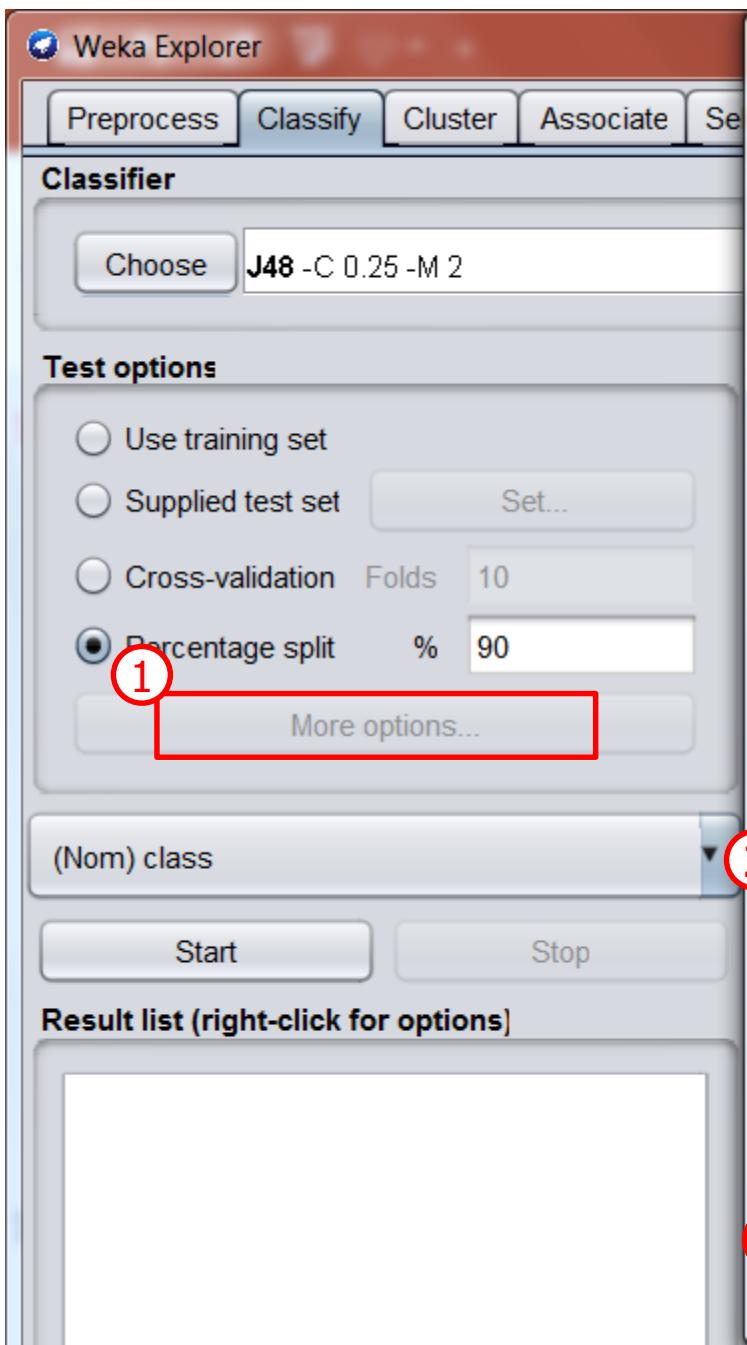
Open the Dataset

- C:\Program Files\Weka-3-8\data\diabetes.arff



Repeated





Holdout (Seed=1)

Choose J48 -C 0.25 -M 2

Test options

- Use training set
- Supplied test set [Set...](#)
- Cross-validation Folds 10
- Percentage split % 90

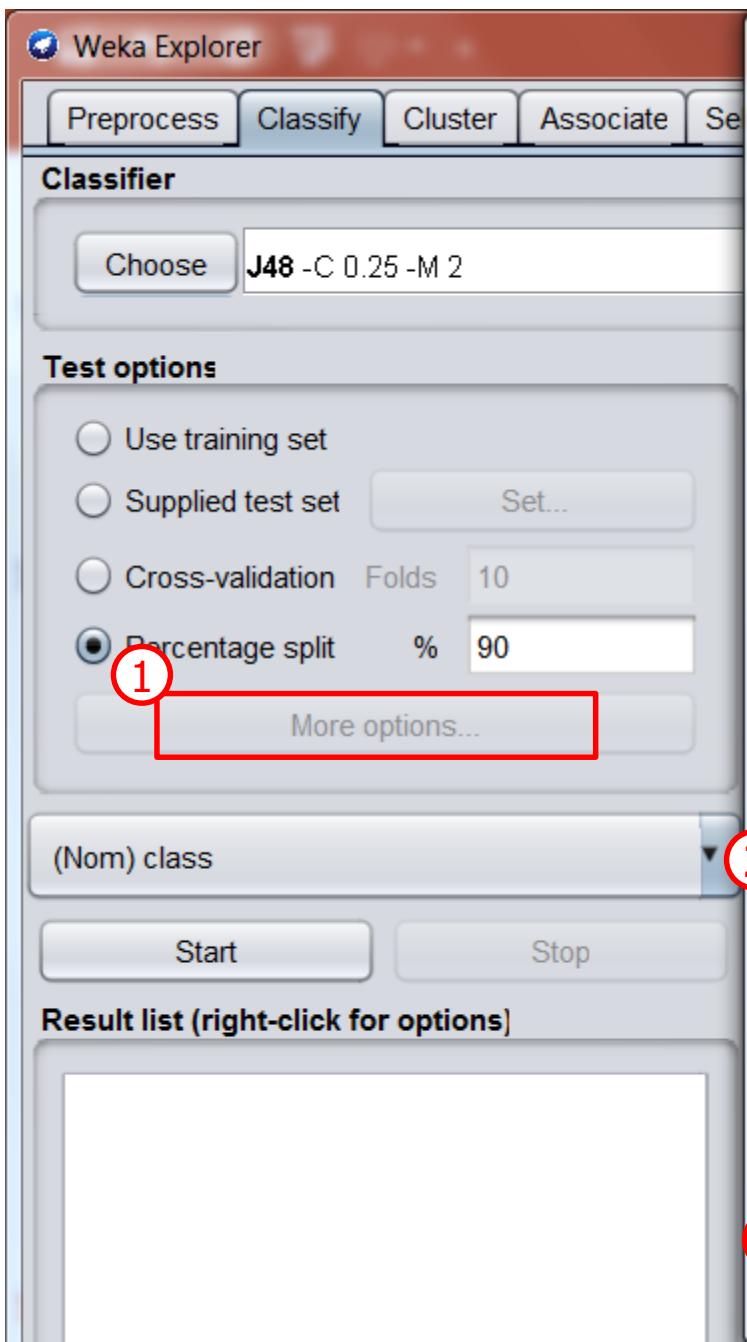
[More options...](#)

(Nom) class

[Start](#) [Stop](#)

Classifier output

```
==== Evaluation on test split ====
Time taken to test model on test split: 0 seconds
==== Summary ====
Correctly Classified Instances      58          75.3247 %
Incorrectly Classified Instances   19          24.6753 %
Kappa statistic                   0.4314
Mean absolute error               0.3036
Root mean squared error           0.4077
Relative absolute error           67.737  %
Root relative squared error      86.9143 %
```



Holdout (Seed=2)

Classifier

Choose **J48 -C 0.25 -M 2**

Test options

Use training set
 Supplied test set
 Cross-validation Folds 10
 Percentage split % 90 More options...

Classifier output

```
==== Evaluation on test split ====
Time taken to test model on test split: 0 seconds
==== Summary ====
Correctly Classified Instances          60           77.9221 %
Incorrectly Classified Instances        17           22.0779 %
Kappa statistic                         0.482
Mean absolute error                     0.2902
Root mean squared error                 0.4371
Relative absolute error                 64.2632 %
Root relative squared error            92.4021 %
```

1

2

(Nom) class

Cross-validation (Seed=1)

Choose J48 -C 0.25 -M 2

Test options

- Use training set
- Supplied test set
- Cross-validation Folds 10 1
- Percentage split % 90

(Nom) class

Classifier output

```
Time taken to build model: 0.02 seconds
==== Stratified cross-validation ====
==== Summary ====
Correctly Classified Instances      567           73.8281 %
Incorrectly Classified Instances   201           26.1719 %
Kappa statistic                   0.4164
Mean absolute error               0.3158
Root mean squared error          0.4463
Relative absolute error          69.4841 %
Root relative squared error     93.6293 %
```

Cross-validation (Seed=2)

Classifier

Choose **J48 -C 0.25 -M 2**

Test options

Use training set
 Supplied test set
 Cross-validation Folds **10** 1
 Percentage split % **90**

(Nom) class 2

Classifier output

```
Time taken to build model: 0.01 seconds

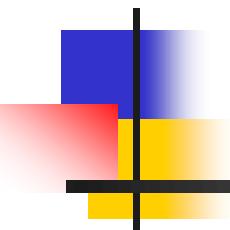
==== Stratified cross-validation ====
==== Summary ===

Correctly Classified Instances      576           75   %
Incorrectly Classified Instances   192            25   %
Kappa statistic                   0.427
Mean absolute error               0.3139
Root mean squared error          0.4408
Relative absolute error           69.0678 %
Root relative squared error      92.4777 %
```

Cross-validation Results

- Cross-validation really is better than repeated holdout
- It reduces the variance of the estimate

		holdout (10%)	cross-validation (10-fold)
Sample mean	$\bar{x} = \frac{\sum x_i}{n}$	75.3	73.8
		77.9	75.0
		80.5	75.5
		74.0	75.5
		71.4	74.4
Variance	$\sigma^2 = \frac{\sum (x_i - \bar{x})^2}{n-1}$	70.1	75.6
		79.2	73.6
		71.4	74.0
		80.5	74.5
		67.5	73.0
Standard deviation	σ	$\bar{x} = 74.8$	$\bar{x} = 74.5$
		$\sigma = 4.6$	$\sigma = 0.9$



Python – Cross-validation

Download the Dataset

- Download diabetes.csv from
<http://kdd.snu.ac.kr/python/>
- Save the csv file in the same directory as the source file (.ipynb)

diabetes.csv

- Classify whether the patient shows signs of diabetes
- Attributes are all numeric
 - Age, body mass index, etc

Import Libraries

```
import pandas as pd
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection \
    import train_test_split, KFold, cross_val_score
```

Import the Dataset

```
df = pd.read_csv('diabetes.csv')  
df[ :3]
```

	preg	plas	pres	skin	insu	mass	pedi	age	class
0	6	148	72	35	0	33.6	0.627	50	tested_positive
1	1	85	66	29	0	26.6	0.351	31	tested_negative
2	8	183	64	0	0	23.3	0.672	32	tested_positive

Import the Dataset

Parse features by excluding the last column

```
X = df.values[:, :-1]
```

```
y = df.values[:, -1]
```

Parse labels from the last column

Import the Dataset

```
print(X.shape)  
print(y.shape)
```

(768, 8)
(768, 1)

768 instances with
8 features

Create a Decision Tree Object

```
decision_tree = DecisionTreeClassifier(  
    min_samples_leaf=2, random_state=0)
```

The minimum number of samples required to
be at a leaf node

Set the random seed

Holdout

```
holdouts = []
for seed in range(10):
    X_train, X_test, y_train, y_test \
        = train_test_split(
            X, y, test_size=0.1,
            random_state=seed)
    decision_tree.fit(X_train, y_train)
    holdouts.append(
        decision_tree.score(X_test, y_test))
```

Holdout

```
holdouts = []          Change the random seed from 0 to 9
for seed in range(10):
    X_train, X_test, y_train, y_test \
        = train_test_split(
            X, y, test_size=0.1,
            random_state=seed)
    decision_tree.fit(X_train, y_train)
    holdouts.append(
        decision_tree.score(X_test, y_test))
```

Split the data to 90 % of train subset and 10 % of test subset

The ratio of test subset

Holdout

```
holdouts = []
for seed in range(10):
    X_train, X_test, y_train, y_test \
        = train_test_split(
            X, y, test_size=0.1,
            random_state=seed)
    decision_tree.fit(X_train, y_train)
    holdouts.append(
        decision_tree.score(X_test, y_test))
```

The score on the test data

Fit the decision tree with the train data

Holdout

- The accuracies of the repeated evaluation with different random seeds

```
print(holdouts)
```

```
[0.7012987012987013, 0.7012987012987013, 0.6363636363636364, 0.6  
623376623376623, 0.6753246753246753, 0.7532467532467533, 0.76623  
37662337663, 0.72727272727273, 0.6493506493506493, 0.623376623  
3766234]
```

Holdout

- The mean and standard deviation of the repeated evaluation

```
print ("== Holdout ==")
print ("mean: {} \nstdev: {} \n".format(
    np.mean(holdouts), np.std(holdouts)))
```

```
== Holdout ==
mean: 0.6896103896103897
stdev: 0.04626367046897891
```

K-fold Cross-validation

```
cross_val = []
for seed in range(10):
    cv = KFold(
        n_splits=10, shuffle=True,
        random_state=seed)
    cross_val.append(
        np.mean(
            cross_val_score(
                decision_tree, X, y, cv=cv) ))
```

K-fold Cross-validation

```
cross_val = []      Change the random from 0 to 9
for seed in range(10):
    cv = KFold(
        n_splits=10, shuffle=True,
        random_state=seed)
    cross_val.append(
        np.mean(
            cross_val_score(
                decision_tree, X, y, cv=cv) )))
```

K-fold Cross-validation

- Create an object for k -fold cross-validation

The number of folds

```
cross_val = []
for seed in range(10):
    cv = KFold(
        n_splits=10, shuffle=True,
        random_state=seed)
    cross_val.append(
        np.mean(
            cross_val_score(
                decision_tree, X, Y, cv=cv)))
```

Whether to shuffle the data
before splitting into batches

K-fold Cross-validation

- Perform the 10-fold cross-validation

```
cross_val = []
for seed in range(10):      The 10 accuracies of the 10-fold cross-
    cv = KFold(           validation
        n_splits=10, shuffle=True,
        random_state=seed)
    cross_val.append(
        np.mean(
            cross_val_score(
                decision_tree, X, y, cv=cv)))
```

The mean of the 10 accuracies

K-fold Cross-validation

- The accuracies of the repeated 10-fold cross-validation with different random seeds

```
print(cross_val)
```

```
[0.6939507860560492, 0.7044429254955571, 0.6965994531784006, 0.7  
136021872863978, 0.701913875598086, 0.7096548188653452, 0.695334  
928229665, 0.701930963773069, 0.7111073137388927, 0.700649350649  
3506]
```

K-fold Cross-validation

- The mean and standard deviation of the repeated cross-validation

```
print("== Cross-validation ==")
print("mean: {} \nstdev: {} \n".format(
    np.mean(cross_val), np.std(cross_val)))
```

```
== Cross-validation ==
mean: 0.7029186602870813
stdev: 0.006433459822272133
```

K-fold Cross-validation

- k -fold Cross-validation reduces the variance of the estimate

```
== Holdout ==
mean: 0.6896103896103897
stdev: 0.04626367046897891
```

```
== Cross-validation ==
mean: 0.7029186602870813
stdev: 0.006433459822272133
```

Score Function - Metrics

- Evaluation tools, such as `model_selection.cross_val_score`, take a `scoring` parameter that controls what metric they apply to the estimators evaluated

Notations

- \hat{y}_i : the predicted value of the i -th sample
- y_i : the corresponding true value
- n : the number of samples

Classification Metrics

- Accuracy score

$$\text{accuracy} = \frac{1}{n} \sum_{i=1}^n 1(\hat{y}_i = y_i)$$

- Balanced accuracy score
 - Avoids inflated performance estimates on imbalanced datasets

$$\text{balanced_accuracy} = \frac{1}{\sum_{i=1}^n w_i} \sum_{i=1}^n w_i 1(\hat{y}_i = y_i)$$

$$w_i = 1 / n_{y_i}$$

- w_i : the sample weight of i -th sample
- n_{y_i} : the number of samples belonging to class y_i

Regression Metrics

- Mean absolute error

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

- Mean squared error

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

- R² score

$$R^2 = 1 - \frac{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}{\frac{1}{n} \sum_{i=1}^n (y_i - \bar{y})^2}$$

Classification Metrics

- Download glass.csv from <http://kdd.snu.ac.kr/python/>
- Save the csv file in the same directory as the source file (.ipynb)

Classification Metrics

■ Import Libraries

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import \
    KFold, cross_val_score
```

Classification Metrics

- Import the dataset

```
df = pd.read_csv('glass.csv')
df[:3]
```

	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type
0	1.51793	12.79	3.50	1.12	73.03	0.64	8.77	0.0	0.0	'build wind float'
1	1.51643	12.16	3.52	1.35	72.89	0.57	8.53	0.0	0.0	'vehic wind float'
2	1.51793	13.21	3.48	1.41	72.64	0.59	8.43	0.0	0.0	'build wind float'

Classification Metrics

- Convert to Arrays

```
X = df.values[:, :-1]
y = df.values[:, -1]
print(X[:3])
print(y[:3])
```

```
[[1.51793 12.79 3.5 1.12 73.03 0.64 8.77 0.0 0.0]
 [1.51643 12.16 3.52 1.35 72.89 0.57 8.53 0.0 0.0]
 [1.51793 13.21 3.48 1.41 72.64 0.59 8.43 0.0 0.0]]
["'build wind float'" "'vehic wind float'" "'build wind float'"]
```

Classification Metrics

- Create a classifier object

```
clf = DecisionTreeClassifier(  
    min_samples_leaf=2,  
    random_state=0)
```

Classification Metrics

```
from sklearn.metrics import \
    make_scorer, accuracy_score, balanced_accuracy_score
cv = KFold(      — Create an K-Folds cross-validator object
    n_splits=5,   — The number of folds
    shuffle=True, — Whether to shuffle the data before splitting
    random_state=0) — The random seed
print("{}: {}".format(
    "default", cross_val_score(
        clf,      — The object of the naïve bayes classifier
        X,       > The features and labels
        Y,
        cv=cv)))
```

The default metric (accuracy) is used since we does not specify it

Scores of 5-fold cross-validation are printed

```
default: [0.81395349 0.81395349 0.65116279 0.69767442 0.52380952]
```

Classification Metrics

```
metrics = (  
    ("acc", accuracy_score),  
    ("balanced-acc", balanced_accuracy_score))  
  
for name, metric in metrics:  
    print("{}:\t{}".format(  
        name,  
        cross_val_score(  
            clf, X, y, cv=cv,  
            scoring=make_scorer(  
                metric))))
```

Create tuples of (name, metric)

Iterate over tuples of (name, metric)

Specify the evaluation metric

```
acc: [0.81395349 0.81395349 0.65116279 0.69767442 0.52380952]  
balanced-acc: [0.84380342 0.79325397 0.50833333 0.71122995 0.45392857]
```

Regression Metrics

- Download cpu.csv from <http://kdd.snu.ac.kr/python/>
- Save the csv file in the same directory as the source file (.ipynb)

Regression Metrics

■ Import Libraries

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import \
    KFold, cross_val_score
```

Regression Metrics

- Import the dataset

```
df = pd.read_csv('cpu.csv')  
df [:3]
```

	MYCT	MMIN	MMAX	CACH	CHMIN	CHMAX	class
0	125	256	6000	256	16	128	198
1	29	8000	32000	32	8	32	269
2	29	8000	32000	32	8	32	220

Regression Metrics

- Convert to Arrays

```
X = df.values[:, :-1]
y = df.values[:, -1]
print(X[:3])
print(y[:3])
```

```
[ [ 125      256     6000      256       16     128]
  [    29     8000   32000       32        8      32]
  [    29     8000   32000       32        8      32] ]
[198  269  220]
```

Regression Metrics

- Create a regression object

```
reg = LinearRegression()
```

Regression Metrics

```
from sklearn.metrics import \
    make_scorer, \
    mean_absolute_error, mean_squared_error, r2_score
cv = KFold(
    n_splits=5,
    shuffle=True,
    random_state=0)
print("{}: {}".format(
    "default",
    cross_val_score(reg, X, y, cv=cv) ))
```

The default metric (R^2 score) is used since we does not specify it

Scores of 5-fold cross-validation are printed

```
default: [0.88405222 0.83569601 0.79535853 0.61908083 0.89487644]
```

Regression Metrics

```
metrics = (  
    ("R2", r2_score),  
    ("MAE", mean_absolute_error),  
    ("MSE", mean_squared_error))  
  
for name, metric in metrics:  
    print("{}:\t {}".format(  
        name,  
        cross_val_score(  
            reg, X, y, cv=cv,  
            scoring=make_scorer(  
                metric))))
```

Create tuples of (name, metric)

Iterate over tuples of (name, metric)

Specify the evaluation metric

```
R2: [0.88405222 0.83569601 0.79535853 0.61908083 0.89487644]  
MAE: [35.13141338 35.95014996 43.35013989 37.5926966 46.20434726]  
MSE: [1790.36708007 3040.07619033 7275.23376115 3894.21956913 5188.64  
145083]
```

Estimating Confidence Intervals: Classifier Models M_1 vs. M_2

- Suppose we have 2 classifiers, M_1 and M_2 , which one is better?
- Use 10-fold cross-validation to obtain $\overline{err}(M_1)$ and $\overline{err}(M_2)$
- These mean error rates are just *estimates* of error on the true population of *future* data cases
- What if the difference between the 2 error rates is just attributed to *chance*?
 - Use a **test of statistical significance**
 - Obtain **confidence limits** for our error estimates

Estimating Confidence Intervals: Null Hypothesis

- Perform 10-fold cross-validation
- Assume samples follow a **t distribution** with **$k-1$ degrees of freedom** (here, $k=10$)
- Use **t-test** (or **Student's t-test**)
- **Null Hypothesis:** M_1 & M_2 are the same
- If we can **reject** null hypothesis, then
 - we conclude that the difference between M_1 & M_2 is **statistically significant**
 - Choose model with lower error rate

Estimating Confidence Intervals: t-test

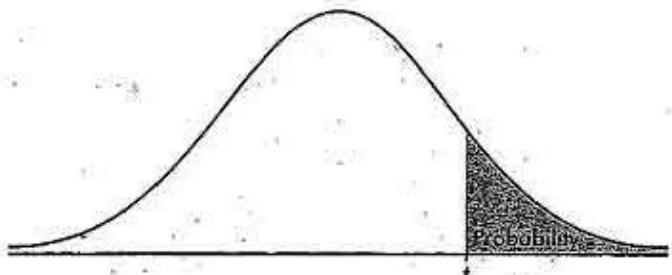
- If only 1 test set available: **pairwise comparison**
 - For i^{th} round of 10-fold cross-validation, the same cross partitioning is used to obtain $\text{err}(M_1)_i$ and $\text{err}(M_2)_i$
 - Average over 10 rounds to get $\overline{\text{err}}(M_1)$ and $\overline{\text{err}}(M_2)$
 - **t-test computes t-statistic with $k-1$ degrees of freedom:**
- $$t = \frac{\overline{\text{err}}(M_1) - \overline{\text{err}}(M_2)}{\sqrt{\text{var}(M_1 - M_2)/k}}$$
 where
$$\text{var}(M_1 - M_2) = \frac{1}{k} \sum_{i=1}^k \left[\text{err}(M_1)_i - \text{err}(M_2)_i - (\overline{\text{err}}(M_1) - \overline{\text{err}}(M_2)) \right]^2$$
- If two test sets available: use **non-paired t-test**

where

$$\text{var}(M_1 - M_2) = \sqrt{\frac{\text{var}(M_1)}{k_1} + \frac{\text{var}(M_2)}{k_2}},$$

where k_1 & k_2 are # of cross-validation samples used for M_1 & M_2 , resp.

Estimating Confidence Intervals: Table for t-distribution



- Symmetric
- Significance level,
e.g., $sig = 0.05$ or
5% means M_1 & M_2
are *significantly*
different for 95% of
population
- Confidence limit, z
 $= sig/2$

TABLE B: *t*-DISTRIBUTION CRITICAL VALUES

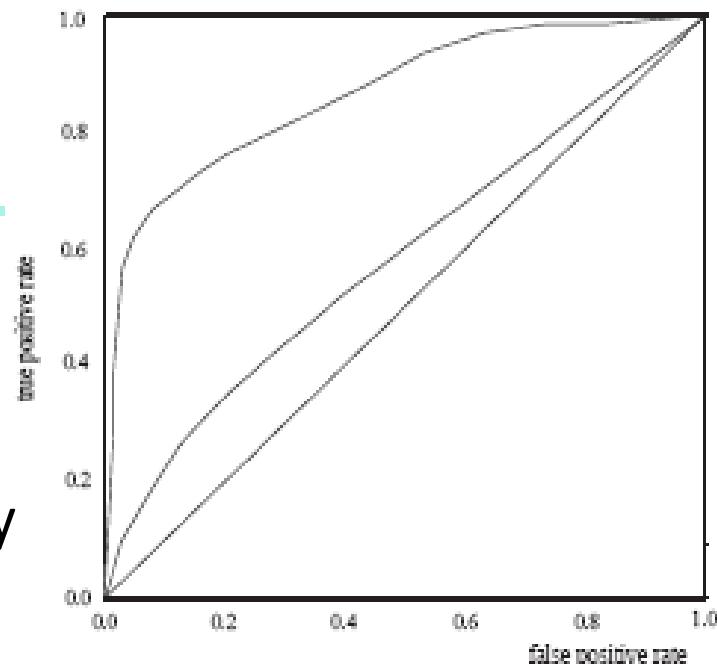
df	Tail probability p											
	.25	.20	.15	.10	.05	.025	.02	.01	.005	.0025	.001	.0005
1	1.000	1.376	1.963	3.078	6.314	12.71	15.89	31.82	63.66	127.3	318.3	636.6
2	.816	1.061	1.386	1.886	2.920	4.303	4.849	6.965	9.925	14.09	22.33	31.60
3	.765	.978	1.250	1.638	2.353	3.182	3.482	4.541	5.841	7.453	10.21	12.92
4	.741	.941	1.190	1.533	2.132	2.776	2.999	3.747	4.604	5.598	7.173	8.610
5	.727	.920	1.156	1.476	2.015	2.571	2.757	3.365	4.032	4.773	5.893	6.869
6	.718	.906	1.134	1.440	1.943	2.447	2.612	3.143	3.707	4.317	5.208	5.959
7	.711	.896	1.119	1.415	1.895	2.365	2.517	2.998	3.499	4.029	4.785	5.408
8	.706	.889	1.108	1.397	1.860	2.306	2.449	2.896	3.355	3.833	4.501	5.041
9	.703	.883	1.100	1.383	1.833	2.262	2.398	2.821	3.250	3.690	4.297	4.781
10	.700	.879	1.093	1.372	1.812	2.228	2.359	2.764	3.169	3.581	4.144	4.587
11	.697	.876	1.088	1.363	1.796	2.201	2.328	2.718	3.106	3.497	4.025	4.437
12	.695	.873	1.083	1.356	1.782	2.179	2.303	2.681	3.055	3.428	3.930	4.318
13	.694	.870	1.079	1.350	1.771	2.160	2.282	2.650	3.012	3.372	3.852	4.221
14	.692	.868	1.076	1.345	1.761	2.145	2.264	2.624	2.977	3.326	3.787	4.140
15	.691	.866	1.074	1.341	1.753	2.131	2.249	2.602	2.947	3.286	3.733	4.073
16	.690	.865	1.071	1.337	1.746	2.120	2.235	2.583	2.921	3.252	3.686	4.015
17	.689	.863	1.069	1.333	1.740	2.110	2.224	2.567	2.898	3.222	3.646	3.965
18	.688	.862	1.067	1.330	1.734	2.101	2.214	2.552	2.878	3.197	3.611	3.922
19	.688	.861	1.066	1.328	1.729	2.093	2.205	2.539	2.861	3.174	3.579	3.883
20	.687	.860	1.064	1.325	1.725	2.086	2.197	2.528	2.845	3.153	3.552	3.850
21	.686	.859	1.063	1.323	1.721	2.080	2.189	2.518	2.831	3.135	3.527	3.819
22	.686	.858	1.061	1.321	1.717	2.074	2.183	2.508	2.819	3.119	3.505	3.792
23	.685	.858	1.060	1.319	1.714	2.069	2.177	2.500	2.807	3.104	3.485	3.768
24	.685	.857	1.059	1.318	1.711	2.064	2.172	2.492	2.797	3.091	3.467	3.745
25	.684	.856	1.058	1.316	1.708	2.060	2.167	2.485	2.787	3.078	3.450	3.725
26	.684	.856	1.058	1.315	1.706	2.056	2.162	2.479	2.779	3.067	3.435	3.707
27	.684	.855	1.057	1.314	1.703	2.052	2.158	2.473	2.771	3.057	3.421	3.690
28	.683	.855	1.056	1.313	1.701	2.048	2.154	2.467	2.763	3.047	3.408	3.674
29	.683	.854	1.055	1.311	1.699	2.045	2.150	2.462	2.756	3.038	3.396	3.659
30	.683	.854	1.055	1.310	1.697	2.042	2.147	2.457	2.750	3.030	3.385	3.646
40	.681	.851	1.050	1.303	1.684	2.021	2.123	2.423	2.704	2.971	3.307	3.551
50	.679	.849	1.047	1.299	1.676	2.009	2.109	2.403	2.678	2.937	3.261	3.496
60	.679	.848	1.045	1.296	1.671	2.000	2.099	2.390	2.660	2.915	3.232	3.460
80	.678	.846	1.043	1.292	1.664	1.990	2.088	2.374	2.639	2.887	3.195	3.416
100	.677	.845	1.042	1.290	1.660	1.984	2.081	2.364	2.626	2.871	3.174	3.390
1000	.675	.842	1.037	1.282	1.646	1.962	2.056	2.330	2.581	2.813	3.098	3.300
∞	.674	.841	1.036	1.282	1.645	1.960	2.054	2.326	2.576	2.807	3.091	3.291
	50%	60%	70%	80%	90%	95%	96%	98%	99%	99.5%	99.8%	99.9%
	Confidence level C											

Estimating Confidence Intervals: Statistical Significance

- Are M_1 & M_2 significantly different?
 - Compute t . Select *significance level* (e.g. $sig = 5\%$)
 - Consult table for t-distribution: Find *t value* corresponding to *k-1 degrees of freedom* (here, 9)
 - t-distribution is symmetric: typically upper % points of distribution shown → look up value for **confidence limit** $z=sig/2$ (here, 0.025)
 - If $t > z$ or $t < -z$, then t value lies in rejection region:
 - **Reject null hypothesis** that mean error rates of M_1 & M_2 are same
 - Conclude: statistically significant difference between M_1 & M_2
 - **Otherwise**, conclude that any difference is **chance**

Model Selection: ROC Curves

- **ROC** (Receiver Operating Characteristics) curves: for visual comparison of classification models
- Originated from signal detection theory
- Shows the trade-off between the true positive rate and the false positive rate
- The area under the ROC curve is a measure of the accuracy of the model
- Rank the test tuples in decreasing order: the one that is most likely to belong to the positive class appears at the top of the list
- The closer to the diagonal line (i.e., the closer the area is to 0.5), the less accurate is the model



- Vertical axis represents the true positive rate
- Horizontal axis rep. the false positive rate
- The plot also shows a diagonal line
- A model with perfect accuracy will have an area of 1.0

Issues Affecting Model Selection

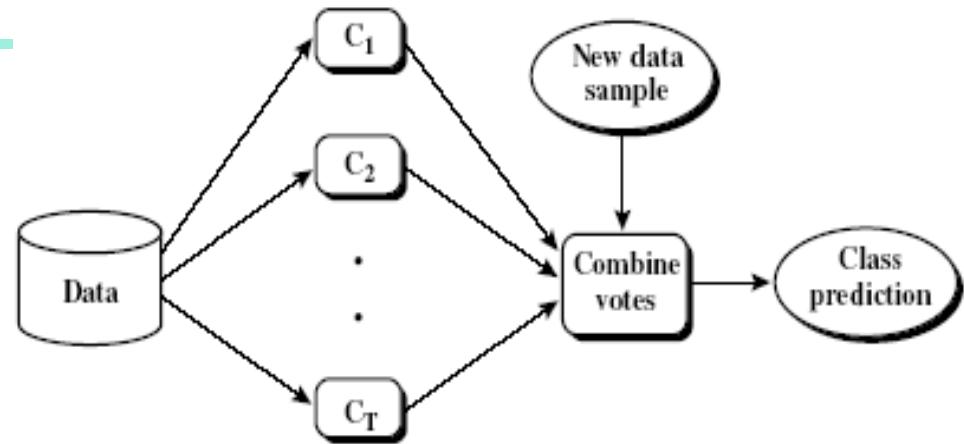
- **Accuracy**
 - classifier accuracy: predicting class label
- **Speed**
 - time to construct the model (training time)
 - time to use the model (classification/prediction time)
- **Robustness:** handling noise and missing values
- **Scalability:** efficiency in disk-resident databases
- **Interpretability**
 - understanding and insight provided by the model
- Other measures, e.g., goodness of rules, such as decision tree size or compactness of classification rules

Chapter 8. Classification: Basic Concepts

- Classification: Basic Concepts
- Decision Tree Induction
- Bayes Classification Methods
- Rule-Based Classification
- Model Evaluation and Selection
- Techniques to Improve Classification Accuracy:
Ensemble Methods
- Summary



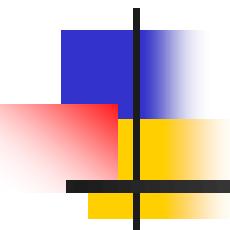
Ensemble Methods: Increasing the Accuracy



- Ensemble methods
 - Use a combination of models to increase accuracy
 - Combine a series of k learned models, M_1, M_2, \dots, M_k , with the aim of creating an improved model M^*
- Popular ensemble methods
 - Bagging: averaging the prediction over a collection of classifiers
 - Boosting: weighted vote with a collection of classifiers
 - Ensemble: combining a set of heterogeneous classifiers

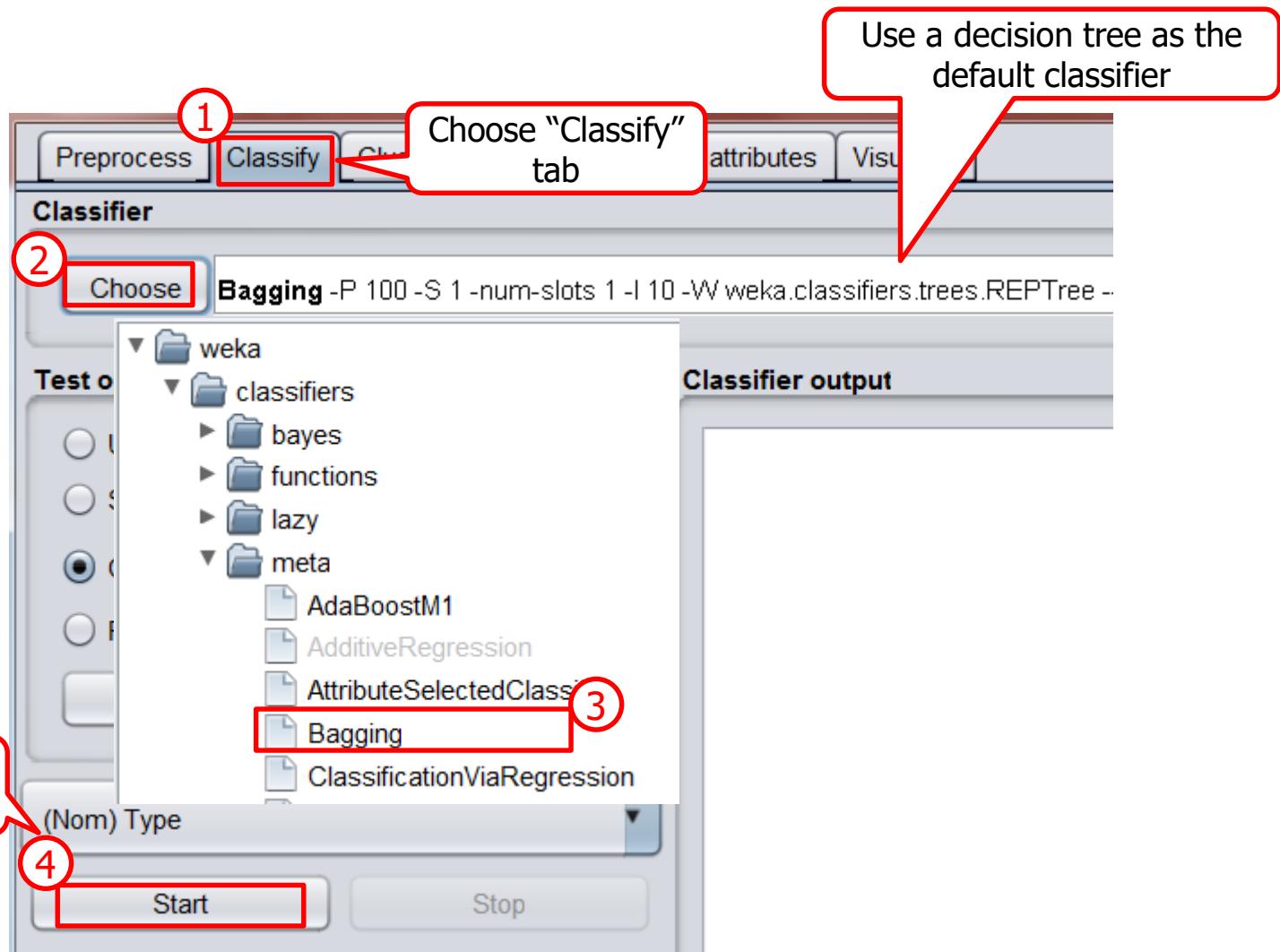
Bagging: Bootstrap Aggregation

- Analogy: Diagnosis based on multiple doctors' majority vote
- Training
 - Given a set D of d tuples, at each iteration i , a training set D_i of d tuples is sampled with replacement from D (i.e., bootstrap)
 - A classifier model M_i is learned for each training set D_i
- Classification: classify an unknown sample X
 - Each classifier M_i returns its class prediction
 - The bagged classifier M^* counts the votes and assigns the class with the most votes to X
- Prediction: can be applied to the prediction of continuous values by taking the average value of each prediction for a given test tuple
- Accuracy
 - Often significantly better than a single classifier derived from D
 - For noise data: not considerably worse, more robust
 - Proved improved accuracy in prediction



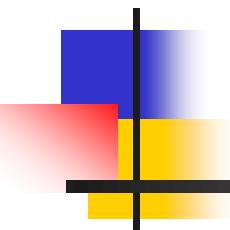
Weka – Bagging

Bagging



Bagging

Classifier output						
==== Summary ====						
Correctly Classified Instances	155				72.4299 %	
Incorrectly Classified Instances	59				27.5701 %	
Kappa statistic	0.6188					
Mean absolute error	0.1206					
Root mean squared error	0.24					
Relative absolute error	56.951 %					
Root relative squared error	73.9649 %					
Total Number of Instances	214					
==== Detailed Accuracy By Class ====						
	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC
	0.829	0.153	0.725	0.829	0.773	0.65
	0.711	0.174	0.692	0.711	0.701	0.53



Python – Bagging

Basic Decision Tree Classifier

```
from sklearn.tree import DecisionTreeClassifier
decision_tree = DecisionTreeClassifier(
    criterion="entropy",
    min_samples_leaf=2,
    random_state=0)
print(cross_val_score(
    decision_tree, X, y, cv=cv).mean())
```

0.6961038961038961

Bagging

```
cv = KFold(n_splits=10,  
           shuffle=True,  
           random_state=0)
```

```
from sklearn.ensemble import BaggingClassifier  
bagging = BaggingClassifier(  
    decision_tree,  
    n_estimators=200,  
    random_state=0)  
print(cross_val_score(bagging, X, y, cv=cv).mean())
```

0.7608225108225108

The base estimator

The number of estimators

Boosting

- Analogy: Consult several doctors, based on a combination of weighted diagnoses—weight assigned based on the previous diagnosis accuracy
- How boosting works?
 - **Weights** are assigned to each training tuple
 - A series of k classifiers is iteratively learned
 - After a classifier M_i is learned, the weights are updated to allow the subsequent classifier, M_{i+1} , to **pay more attention to the training tuples that were misclassified by M_i**
 - The final M^* **combines the votes** of each individual classifier, where the weight of each classifier's vote is a function of its accuracy
- Boosting algorithm can be extended for numeric prediction
- Comparing with bagging: Boosting tends to have greater accuracy, but it also risks overfitting the model to misclassified data

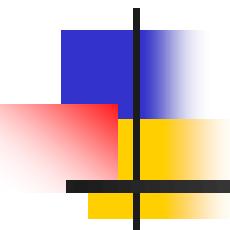
Adaboost (Freund and Schapire, 1997)

- Given a set of d class-labeled tuples, $(\mathbf{X}_1, y_1), \dots, (\mathbf{X}_d, y_d)$
- Initially, all the weights of tuples are set the same ($1/d$)
- Generate k classifiers in k rounds. At round i ,
 - Tuples from D are sampled (with replacement) to form a training set D_i of the same size
 - Each tuple's chance of being selected is based on its weight
 - A classification model M_i is derived from D_i
 - Its error rate is calculated using D_i as a test set
 - If a tuple is misclassified, its weight is increased, o.w. it is decreased
- Error rate: $\text{err}(\mathbf{X}_j)$ is the misclassification error of tuple \mathbf{X}_j . Classifier M_i error rate is the sum of the weights of the misclassified tuples:

$$\text{error}(M_i) = \sum_j^d w_j \times \text{err}(\mathbf{X}_j)$$

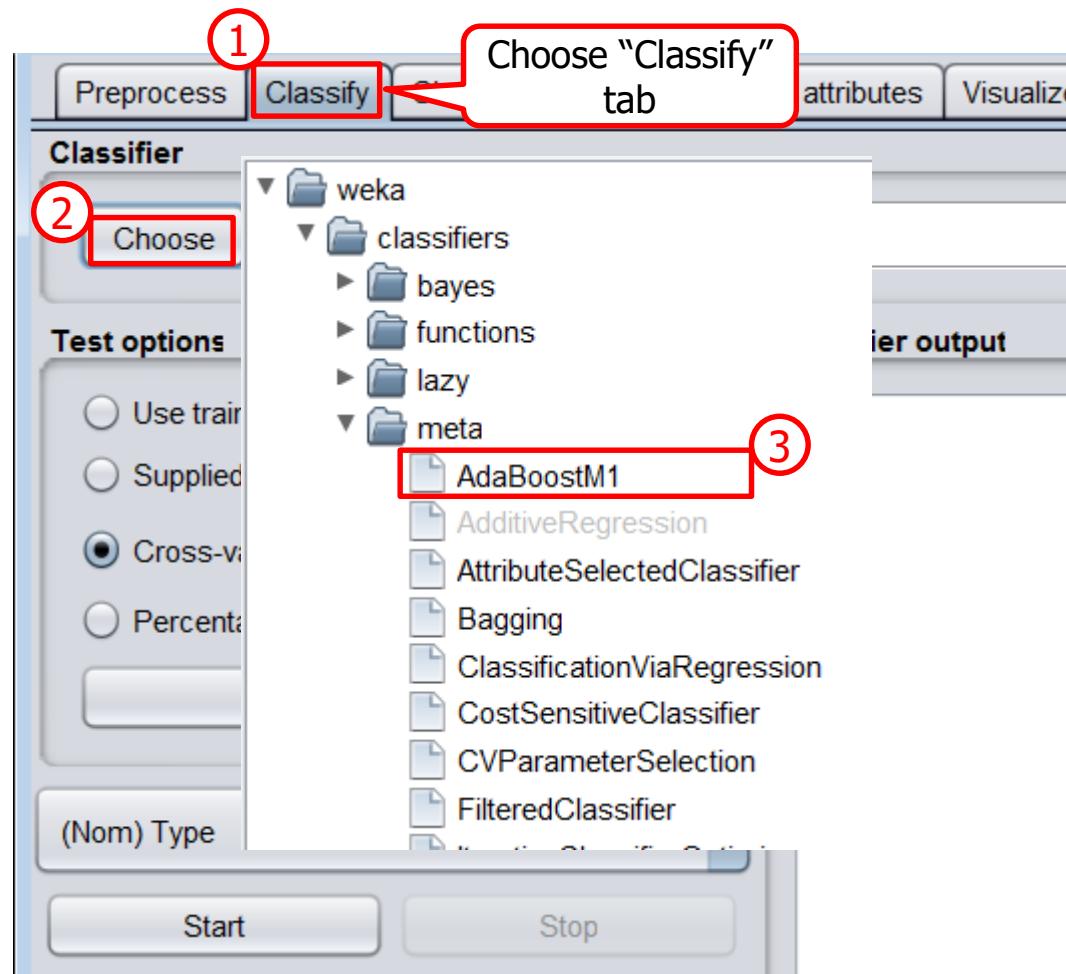
- The weight of classifier M_i 's vote is

$$\log \frac{1 - \text{error}(M_i)}{\text{error}(M_i)}$$

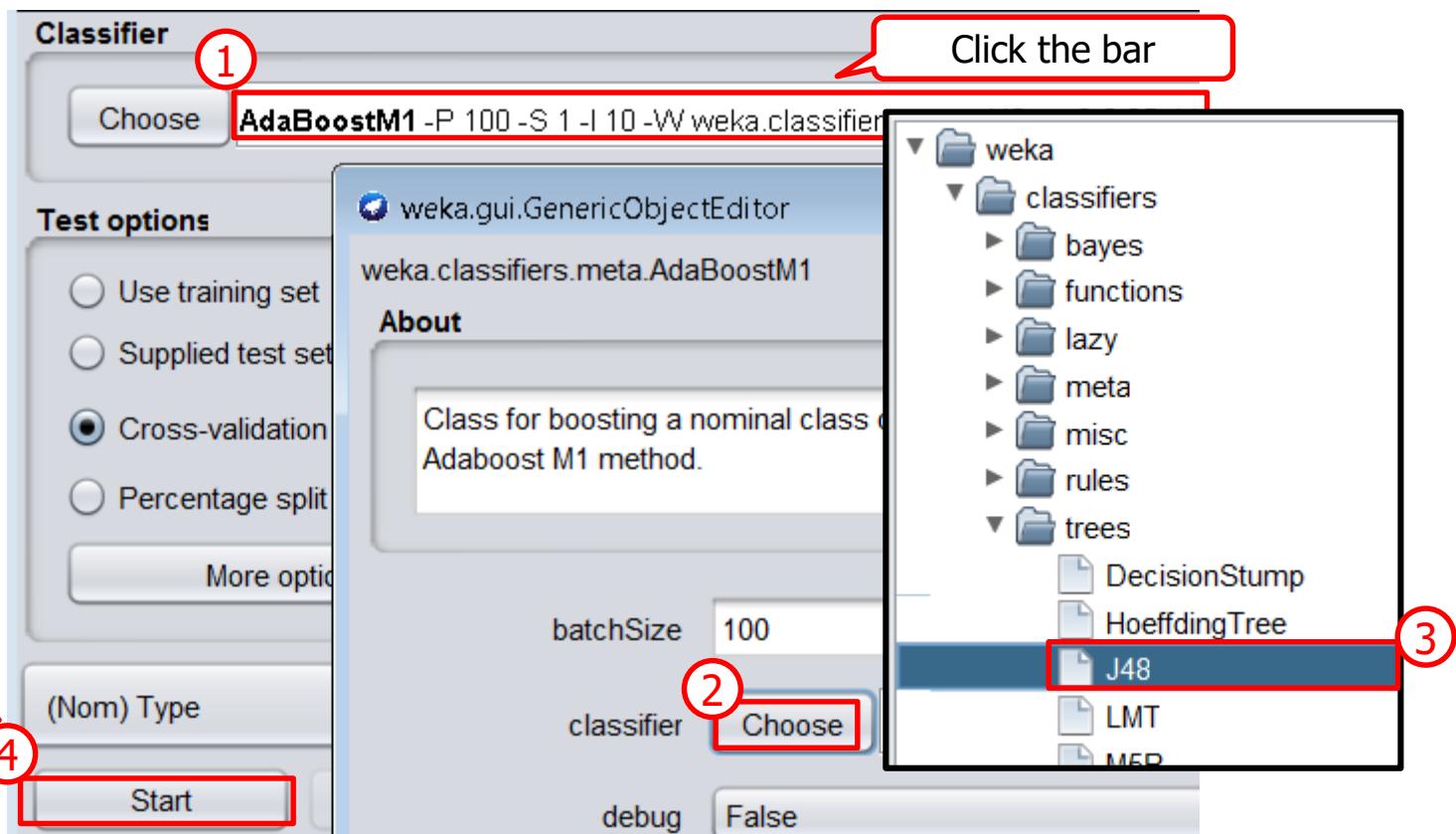


Weka – Boosting

Boosting



Boosting



Boosting

Classifier output

```
==== Stratified cross-validation ====
==== Summary ====

Correctly Classified Instances          159      74.2991 %
Incorrectly Classified Instances        55       25.7009 %
Kappa statistic                         0.6477
Mean absolute error                     0.0711
Root mean squared error                 0.2523
Relative absolute error                  33.5818 %
Root relative squared error            77.7357 %
Total Number of Instances                214

==== Detailed Accuracy By Class ====

           TP Rate   FP Rate   Precision   Recall   F-Measure   MCC

```

Boosting: Intuition

- Train and find a set of weak hypotheses: h_1, \dots, h_T .
- The combined hypothesis H is a **weighted** majority vote of the T weak hypotheses.
 - Each hypothesis h_t has a weight a_t .
- During the training, focus on the examples that are misclassified
 - At round t , example x_i has the weight $D_t(i)$
$$H(x) = \text{sign} \left(\sum_{t=1}^T a_t h_t(x) \right)$$

Basic Setting

- Binary classification problem
- Training data:

$(x_1, y_1), \dots, (x_m, y_m)$, where $x_i \in X, y_i \in Y = \{-1, 1\}$

- $D_t(i)$: the weight of x_i at round t . $D_1(i) = 1/m$
- A learner L that finds a weak hypothesis $h_t: X \rightarrow Y$ given the training set and D_t
- The error of a weak hypothesis h_t :

$$\varepsilon_t = \Pr_{i \sim D_t} [h_t(x_i) \neq y_i] = \sum_{i: h_t(x_i) \neq y_i} D_t(i)$$

The Basic AdaBoost Algorithm

- For $t=1, \dots, T$
 - Train weak learner using training data and D_t
 - Get $h_t: X \rightarrow \{-1,1\}$ with error $\varepsilon_t = \sum_{i:h_t(x_i) \neq y_i} D_t(i)$

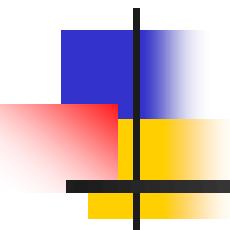
Which hypothesis?  Choose h_t that minimize Z_t

$$Z_t = 2\sqrt{\varepsilon_t(1 - \varepsilon_t)}$$

- Compute $\alpha_t = \frac{1}{2} \ln \frac{1 - \varepsilon_t}{\varepsilon_t}$

- Update $D_{t+1}(i) = \frac{D_t(i)}{Z_t} * \begin{cases} e^{-\alpha_t} & \text{if } h_t(x_i) = y_i \\ e^{\alpha_t} & \text{if } h_t(x_i) \neq y_i \end{cases}$

$$= \frac{D_t(i)e^{-\alpha_t y_i h_t(x_i)}}{Z_t}$$



Python – Boosting

AdaBoost

```
from sklearn.ensemble import AdaBoostClassifier  
adaboost = AdaBoostClassifier(  
    decision_tree,  
    n_estimators=200,  
    random_state=0)  
print(cross_val_score(adaboost, X, y, cv=cv).mean())
```

0.7792207792207793

The base estimator

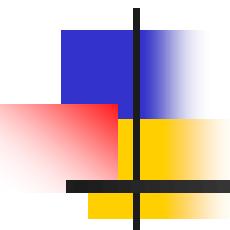
The number of
estimators

Bagging vs. Boosting

- Bagging always uses **resampling** rather than **reweighting**
- Bagging does not modify the distribution over examples or mislabels, but instead always uses the uniform distribution
- In forming the final hypothesis, bagging gives equal weight to each of the weak hypotheses

Random Forest (Breiman 2001)

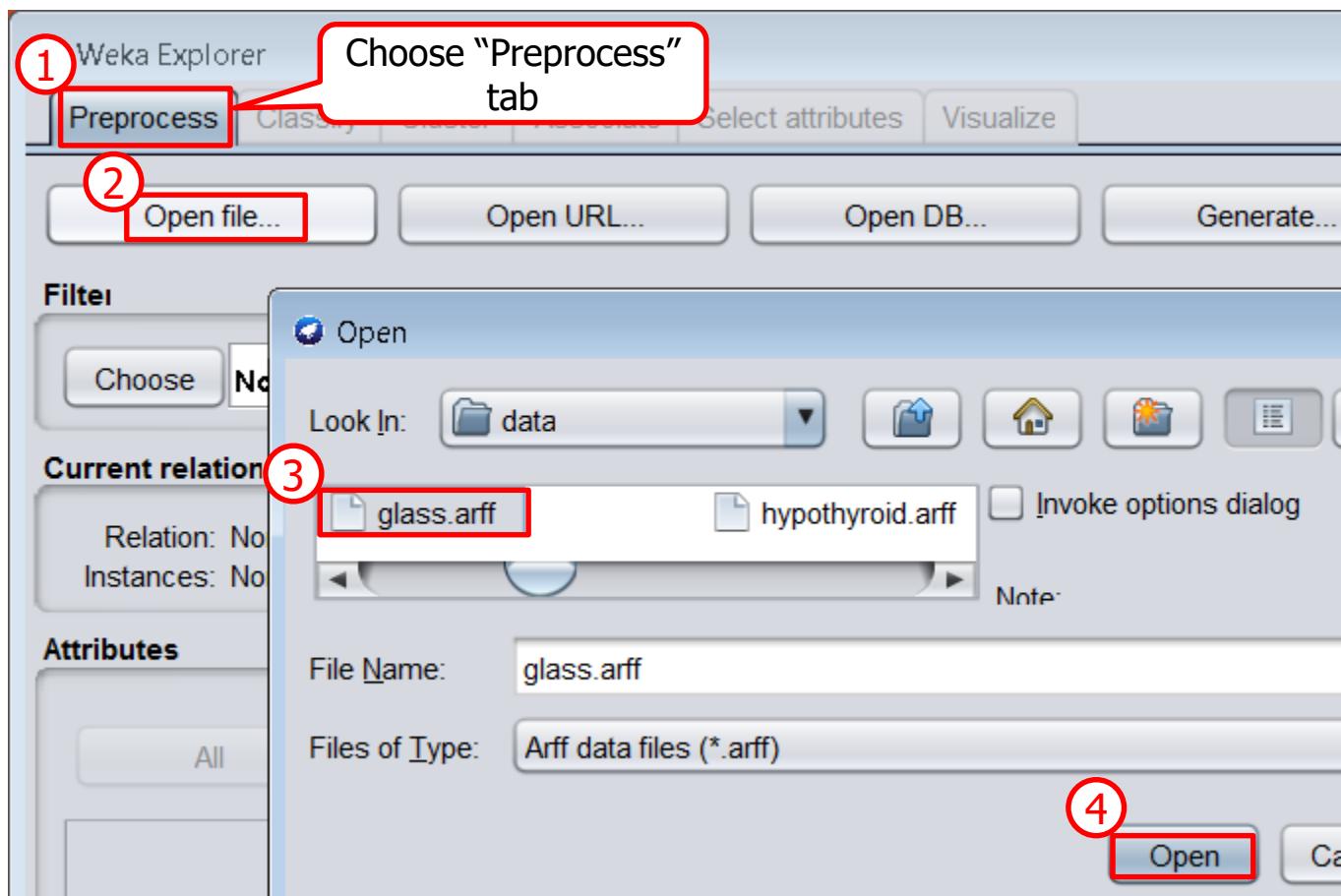
- Random Forest:
 - Each classifier in the ensemble is a *decision tree* classifier and is generated using a random selection of attributes at each node to determine the split
 - During classification, each tree votes and the most popular class is returned
- Two Methods to construct Random Forest:
 - Forest-RI (*random input selection*): Randomly select, at each node, F attributes as candidates for the split at the node. The CART methodology is used to grow the trees to maximum size
 - Forest-RC (*random linear combinations*): Creates new attributes (or features) that are a linear combination of the existing attributes (reduces the correlation between individual classifiers)
- Comparable in accuracy to Adaboost, but more robust to errors and outliers
- Insensitive to the number of attributes selected for consideration at each split, and faster than bagging or boosting



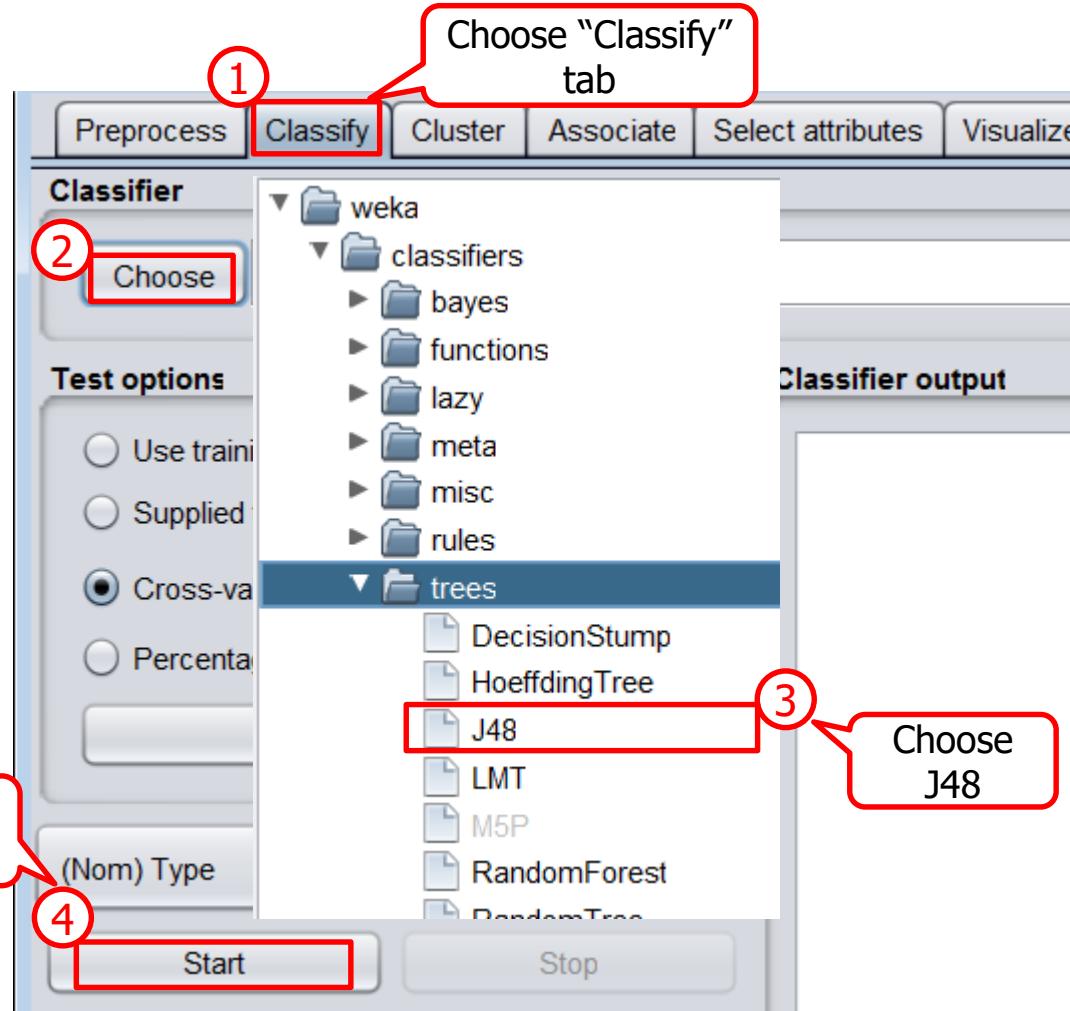
Weka – Random Forests

Open the Dataset

- C:\Program Files\Weka-3-8\data\glass.arff



J48



J48

Classifier output

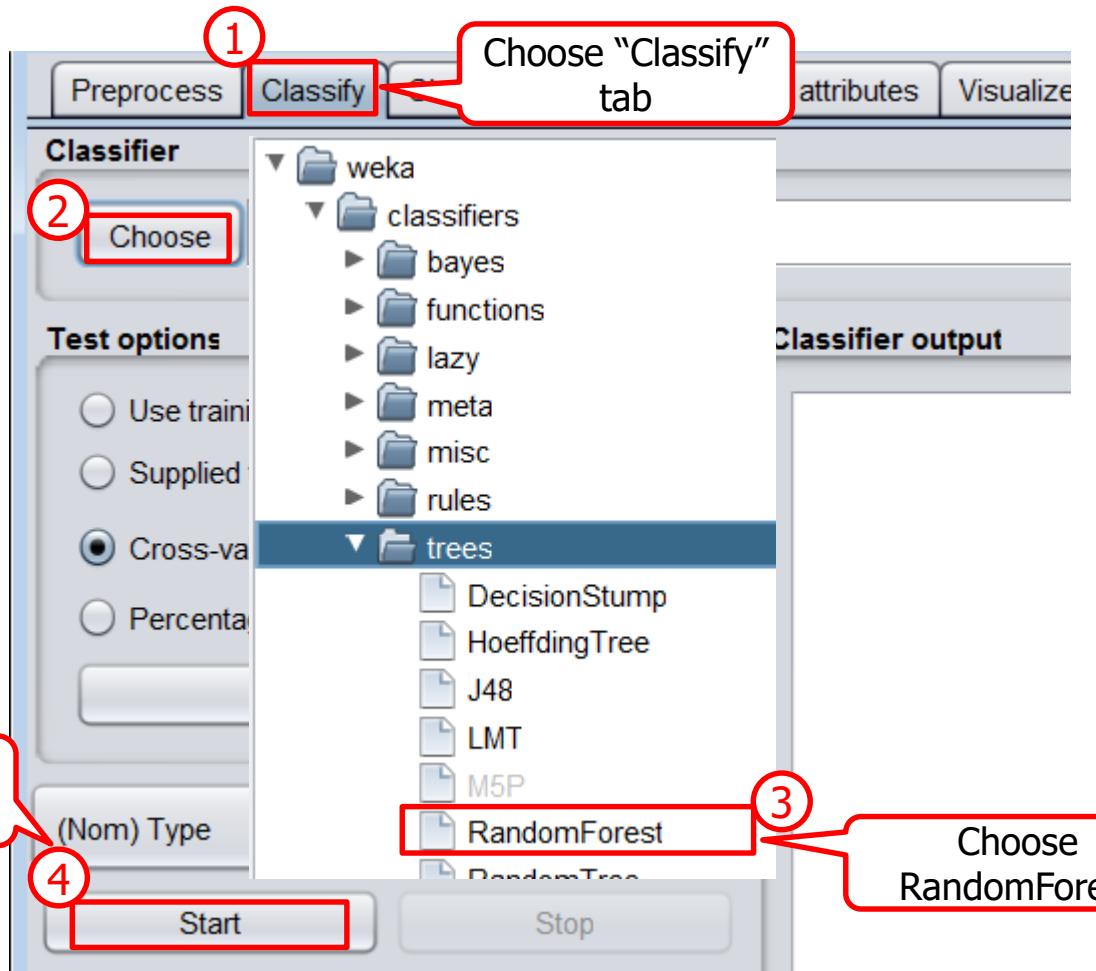
==== Summary ====

Correctly Classified Instances	143	66.8224 %
Incorrectly Classified Instances	71	33.1776 %
Kappa statistic	0.55	
Mean absolute error	0.1026	
Root mean squared error	0.2897	
Relative absolute error	48.4507 %	
Root relative squared error	89.2727 %	
Total Number of Instances	214	

==== Detailed Accuracy By Class ====

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC
0	0.714	0.174	0.667	0.714	0.690	0.53
1	0.618	0.181	0.653	0.618	0.635	0.44

Random Forests



Random Forests

Classifier output

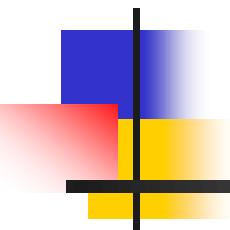
```
==== Stratified cross-validation ====
==== Summary ====
Correctly Classified Instances          171
Incorrectly Classified Instances        43
Kappa statistic                         0.7229
Mean absolute error                     0.1004
Root mean squared error                 0.2123
Relative absolute error                  47.4315 %
Root relative squared error            65.4186 %
Total Number of Instances               214

==== Detailed Accuracy By Class ====

```

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC
0						
1						

79.9065 %
20.0935 %



Python – Random Forests

Download the Dataset

- Download glass.csv from <http://kdd.snu.ac.kr/python/>
- Save the csv file in the same directory as the source file (.ipynb)

glass.csv

- Classify the type of glass
 - Motivated by criminological investigation
 - At the scene of the crime, the glass left can be used as evidence...if it is correctly identified!

Features:

RI: refractive index

Na: Sodium

Mg: Magnesium

...

Types of glass:

building_windows_float_processed

building_windows_non_float_processed

vehicle_windows_float_processed

...

Import the Dataset

```
df = pd.read_csv('glass.csv')  
df[:3]
```

	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type
0	1.51793	12.79	3.50	1.12	73.03	0.64	8.77	0.0	0.0	'build wind float'
1	1.51643	12.16	3.52	1.35	72.89	0.57	8.53	0.0	0.0	'vehic wind float'
2	1.51793	13.21	3.48	1.41	72.64	0.59	8.43	0.0	0.0	'build wind float'

Convert to Arrays

```
X = df.values[:, :-1]
y = df.values[:, -1]
print("X.shape: {}, y.shape: {}".format(
    X.shape, y.shape))
```

X.shape: (214, 9), y.shape: (214,)

214 instances with 9 features

Creating an Object for the Cross-validation

```
cv = KFold(n_splits=10,  
           shuffle=True,  
           random_state=0)
```

- `n_splits`: the number of folds
- `shuffle`: Whether to shuffle the data before splitting into batches
- `random_state`: the random seed

Decision Tree Classifier

```
from sklearn.tree import DecisionTreeClassifier
decision_tree = DecisionTreeClassifier(
    criterion="entropy",
    min_samples_leaf=2,
    random_state=0)
print(cross_val_score(
    decision_tree, X, y, cv=cv).mean())
```

0.6961038961038961

Random Forest

```
from sklearn.ensemble import RandomForestClassifier
random_forest = RandomForestClassifier(
    criterion="entropy",
    min_samples_leaf=2,
    n_estimators=200, n_estimators=200,
    random_state=0)
print(cross_val_score(
    random_forest, X, y, cv=cv).mean())
```

The number of
trees in the forest

0.7930735930735932

Classification of Class-Imbalanced Data Sets

- Class-imbalance problem: Rare positive example but numerous negative ones, e.g., medical diagnosis, fraud, oil-spill, fault, etc.
- Traditional methods assume a balanced distribution of classes and equal error costs: not suitable for class-imbalanced data
- Typical methods for imbalance data in 2-class classification:
 - **Oversampling**: re-sampling of data from positive class
 - **Under-sampling**: randomly eliminate tuples from negative class
 - **Threshold-moving**: moves the decision threshold, t , so that the rare class tuples are easier to classify, and hence, less chance of costly false negative errors
 - Ensemble techniques: Ensemble multiple classifiers introduced above
- Still difficult for class imbalance problem on multiclass tasks

Chapter 8. Classification: Basic Concepts

- Classification: Basic Concepts
 - Decision Tree Induction
 - Bayes Classification Methods
 - Rule-Based Classification
 - Model Evaluation and Selection
 - Techniques to Improve Classification Accuracy:
Ensemble Methods
 - Summary
- 

Summary (I)

- Classification is a form of data analysis that extracts models describing important data classes.
- Effective and scalable methods have been developed for decision tree induction, Naive Bayesian classification, rule-based classification, and many other classification methods.
- Evaluation metrics include: accuracy, sensitivity, specificity, precision, recall, F measure, and F_β measure.
- Stratified k-fold cross-validation is recommended for accuracy estimation. Bagging and boosting can be used to increase overall accuracy by learning and combining a series of individual models.

Summary (II)

- Significance tests and ROC curves are useful for model selection.
- There have been numerous comparisons of the different classification methods; the matter remains a research topic
- No single method has been found to be superior over all others for all data sets
- Issues such as accuracy, training time, robustness, scalability, and interpretability must be considered and can involve trade-offs, further complicating the quest for an overall superior method

References (1)

- C. Apte and S. Weiss. **Data mining with decision trees and decision rules**. Future Generation Computer Systems, 13, 1997
- C. M. Bishop, **Neural Networks for Pattern Recognition**. Oxford University Press, 1995
- L. Breiman, J. Friedman, R. Olshen, and C. Stone. **Classification and Regression Trees**. Wadsworth International Group, 1984
- C. J. C. Burges. **A Tutorial on Support Vector Machines for Pattern Recognition**. *Data Mining and Knowledge Discovery*, 2(2): 121-168, 1998
- P. K. Chan and S. J. Stolfo. **Learning arbiter and combiner trees from partitioned data for scaling machine learning**. KDD'95
- H. Cheng, X. Yan, J. Han, and C.-W. Hsu, **Discriminative Frequent Pattern Analysis for Effective Classification**, ICDE'07
- H. Cheng, X. Yan, J. Han, and P. S. Yu, **Direct Discriminative Pattern Mining for Effective Classification**, ICDE'08
- W. Cohen. **Fast effective rule induction**. ICML'95
- G. Cong, K.-L. Tan, A. K. H. Tung, and X. Xu. **Mining top-k covering rule groups for gene expression data**. SIGMOD'05

References (2)

- A. J. Dobson. **An Introduction to Generalized Linear Models**. Chapman & Hall, 1990.
- G. Dong and J. Li. **Efficient mining of emerging patterns: Discovering trends and differences**. KDD'99.
- R. O. Duda, P. E. Hart, and D. G. Stork. **Pattern Classification**, 2ed. John Wiley, 2001
- U. M. Fayyad. **Branching on attribute values in decision tree generation**. AAAI'94.
- Y. Freund and R. E. Schapire. **A decision-theoretic generalization of on-line learning and an application to boosting**. J. Computer and System Sciences, 1997.
- J. Gehrke, R. Ramakrishnan, and V. Ganti. **Rainforest: A framework for fast decision tree construction of large datasets**. VLDB'98.
- J. Gehrke, V. Gant, R. Ramakrishnan, and W.-Y. Loh, **BOAT -- Optimistic Decision Tree Construction**. SIGMOD'99.
- T. Hastie, R. Tibshirani, and J. Friedman. **The Elements of Statistical Learning: Data Mining, Inference, and Prediction**. Springer-Verlag, 2001.
- D. Heckerman, D. Geiger, and D. M. Chickering. **Learning Bayesian networks: The combination of knowledge and statistical data**. Machine Learning, 1995.
- W. Li, J. Han, and J. Pei, **CMAR: Accurate and Efficient Classification Based on Multiple Class-Association Rules**, ICDM'01.

References (3)

- T.-S. Lim, W.-Y. Loh, and Y.-S. Shih. **A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms.** Machine Learning, 2000.
- J. Magidson. **The Chaid approach to segmentation modeling: Chi-squared automatic interaction detection.** In R. P. Bagozzi, editor, Advanced Methods of Marketing Research, Blackwell Business, 1994.
- M. Mehta, R. Agrawal, and J. Rissanen. **SLIQ : A fast scalable classifier for data mining.** EDBT'96.
- T. M. Mitchell. **Machine Learning.** McGraw Hill, 1997.
- S. K. Murthy, **Automatic Construction of Decision Trees from Data: A Multi-Disciplinary Survey,** Data Mining and Knowledge Discovery 2(4): 345-389, 1998
- J. R. Quinlan. **Induction of decision trees.** *Machine Learning*, 1:81-106, 1986.
- J. R. Quinlan and R. M. Cameron-Jones. **FOIL: A midterm report.** ECML'93.
- J. R. Quinlan. **C4.5: Programs for Machine Learning.** Morgan Kaufmann, 1993.
- J. R. Quinlan. **Bagging, boosting, and c4.5.** AAAI'96.

References (4)

- R. Rastogi and K. Shim. **Public: A decision tree classifier that integrates building and pruning.** VLDB'98.
- J. Shafer, R. Agrawal, and M. Mehta. **SPRINT : A scalable parallel classifier for data mining.** VLDB'96.
- J. W. Shavlik and T. G. Dietterich. **Readings in Machine Learning.** Morgan Kaufmann, 1990.
- P. Tan, M. Steinbach, and V. Kumar. **Introduction to Data Mining.** Addison Wesley, 2005.
- S. M. Weiss and C. A. Kulikowski. **Computer Systems that Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems.** Morgan Kaufman, 1991.
- S. M. Weiss and N. Indurkha. **Predictive Data Mining.** Morgan Kaufmann, 1997.
- I. H. Witten and E. Frank. **Data Mining: Practical Machine Learning Tools and Techniques**, 2ed. Morgan Kaufmann, 2005.
- X. Yin and J. Han. **CPAR: Classification based on predictive association rules.** SDM'03
- H. Yu, J. Yang, and J. Han. **Classifying large data sets using SVM with hierarchical clusters.** KDD'03.