UNIVERSITY OF MANNHEIM

Data Mining

Classification

- Part 1 -

Outline

1. What is Classification?
2. K-Nearest-Neighbors
3. Decision Trees
4. Model Evaluation
5. Rule Learning
6. Naïve Bayes
7. Support Vector Machines
8. Artificial Neural Networks
9. Hyperparameter Selection

1. What is Classification?

- Goal: Previously unseen records should be assigned a class from a given set of classes as accurately as possible

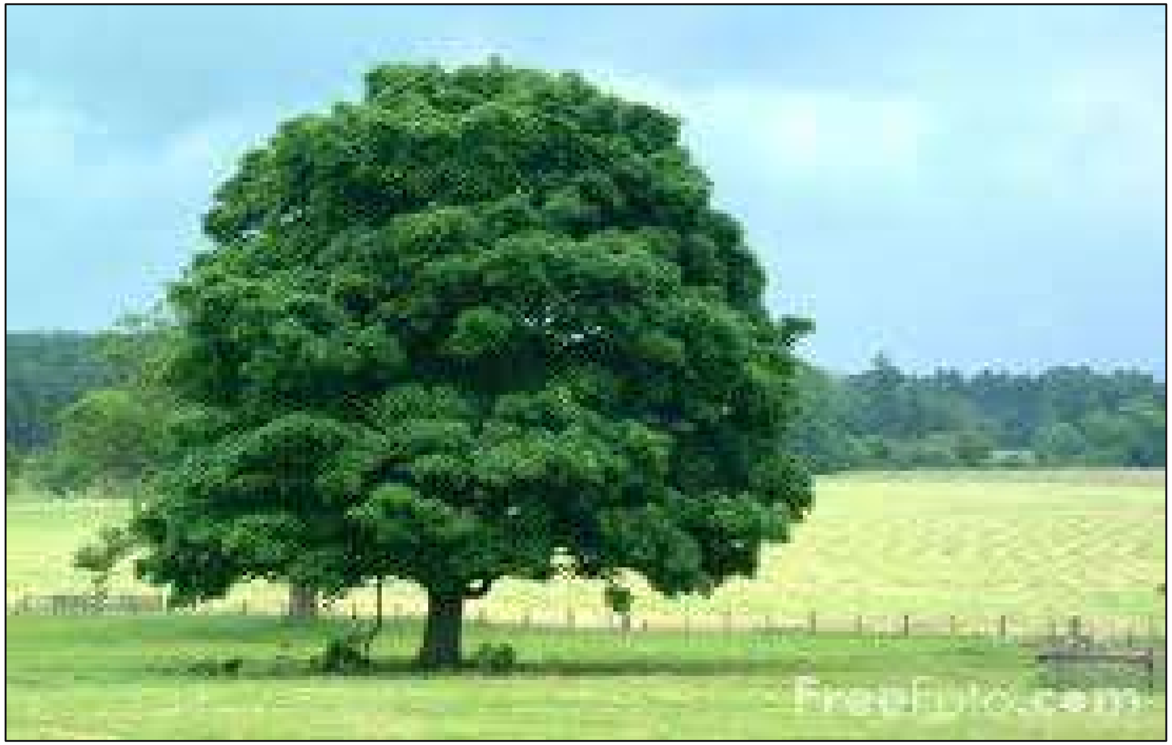
- Approach:
  - Given a collection of records (*training set*)
    - each record contains a set of *attributes*
    - one of the attributes is the *class attribute (label)* that should be predicted
  - Learn a *model* for the class attribute as a function of the values of other attributes

- Variants:
  - Binary classification (e.g. fraud/no fraud or true/false)
  - Multi-class classification (e.g. low, medium, high)
  - Multi-label classification (more than one class per record, e.g. user interests)

Introduction to Classification

A Couple of Questions:

- What is this?
- Why do you know?
- How have you come to that knowledge?



Introduction to Classification

- Goal: Learn a model for recognizing a concept, e.g. trees
- Training data:

"tree"

"tree"

"tree"

"not a tree"

"not a tree"

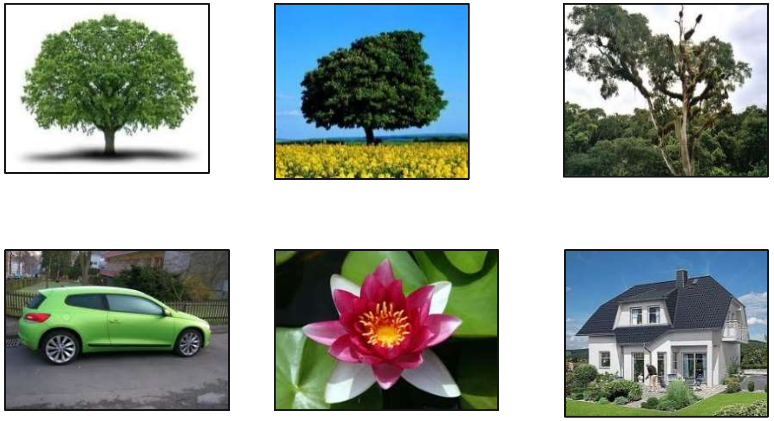
"not a tree"

Introduction to Classification

- We (or the learning algorithm) look at positive and negative examples (training data)
- … and derive a model

  e.g., "Trees are big, green plants that have a trunk and no wheels."

- Goal: Classification of unseen instances

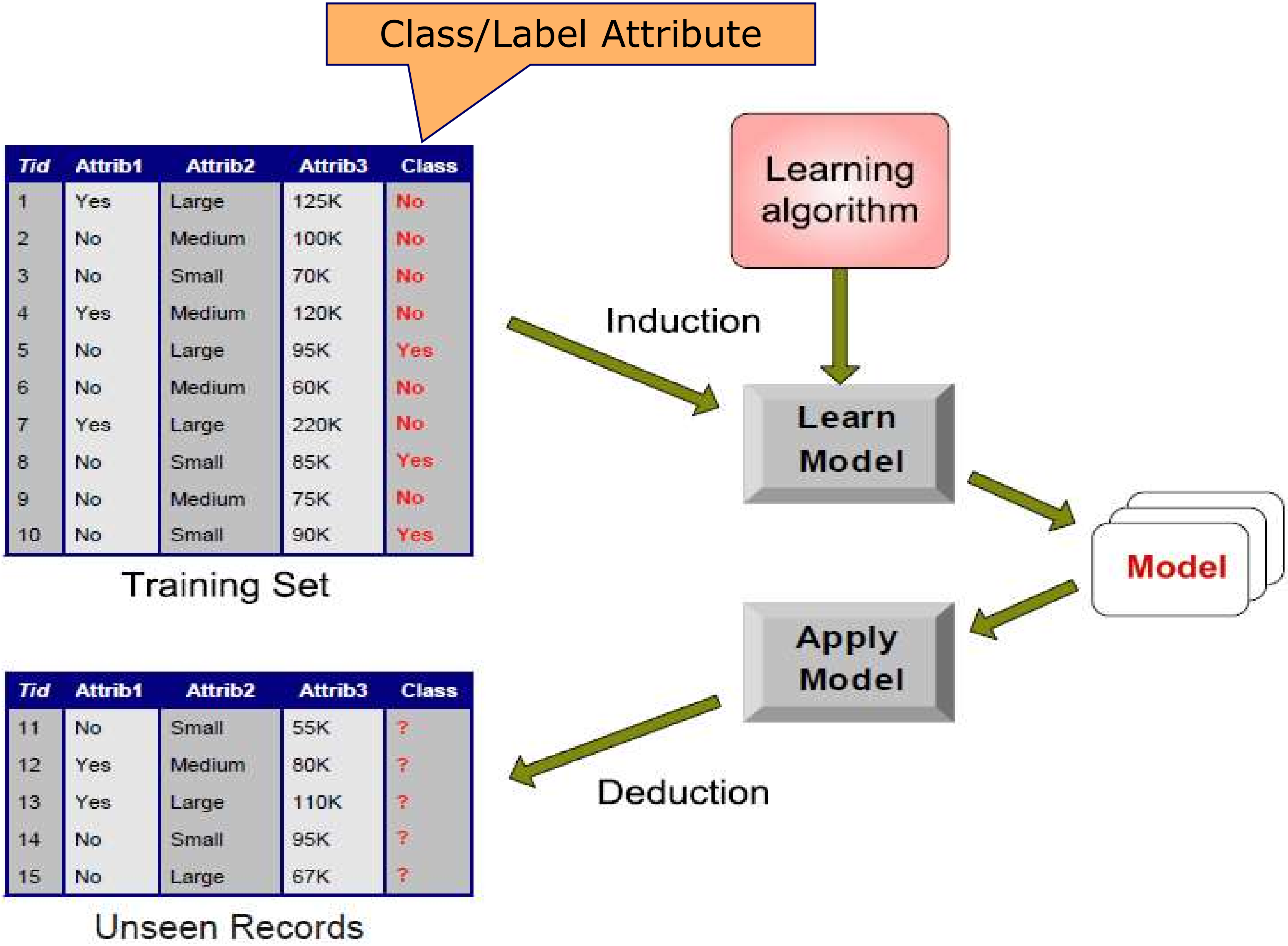


Tree?

Tree?

Warning:
Models are only approximating examples!
Not guaranteed to be correct or complete!

Model Learning and Model Application Process



Tid	Attrib1	Attrib2	Attrib3	Class
1	Yes	Large	125K	No
2	No	Medium	100K	No
3	No	Small	70K	No
4	Yes	Medium	120K	No
5	No	Large	95K	Yes
6	No	Medium	60K	No
7	Yes	Large	220K	No
8	No	Small	85K	Yes
9	No	Medium	75K	No
10	No	Small	90K	Yes

Training Set

Tid	Attrib1	Attrib2	Attrib3	Class
11	No	Small	55K	?
12	Yes	Medium	80K	?
13	Yes	Large	110K	?
14	No	Small	95K	?
15	No	Large	67K	?

Unseen Records

Classification Examples

- Credit Risk Assessment
  - Attributes: your age, income, debts, …
  - Class: are you getting credit by your bank?
- Marketing
  - Attributes: previously bought products, browsing behavior
  - Class: are you a target customer for a new product?
- SPAM Detection
  - Attributes: words and header fields of an e-mail
  - Class: regular e-mail or spam e-mail?
- Identifying Tumor Cells
  - Attributes: features extracted from x-rays or MRI scans
  - Class: malignant or benign cells

Classification Techniques

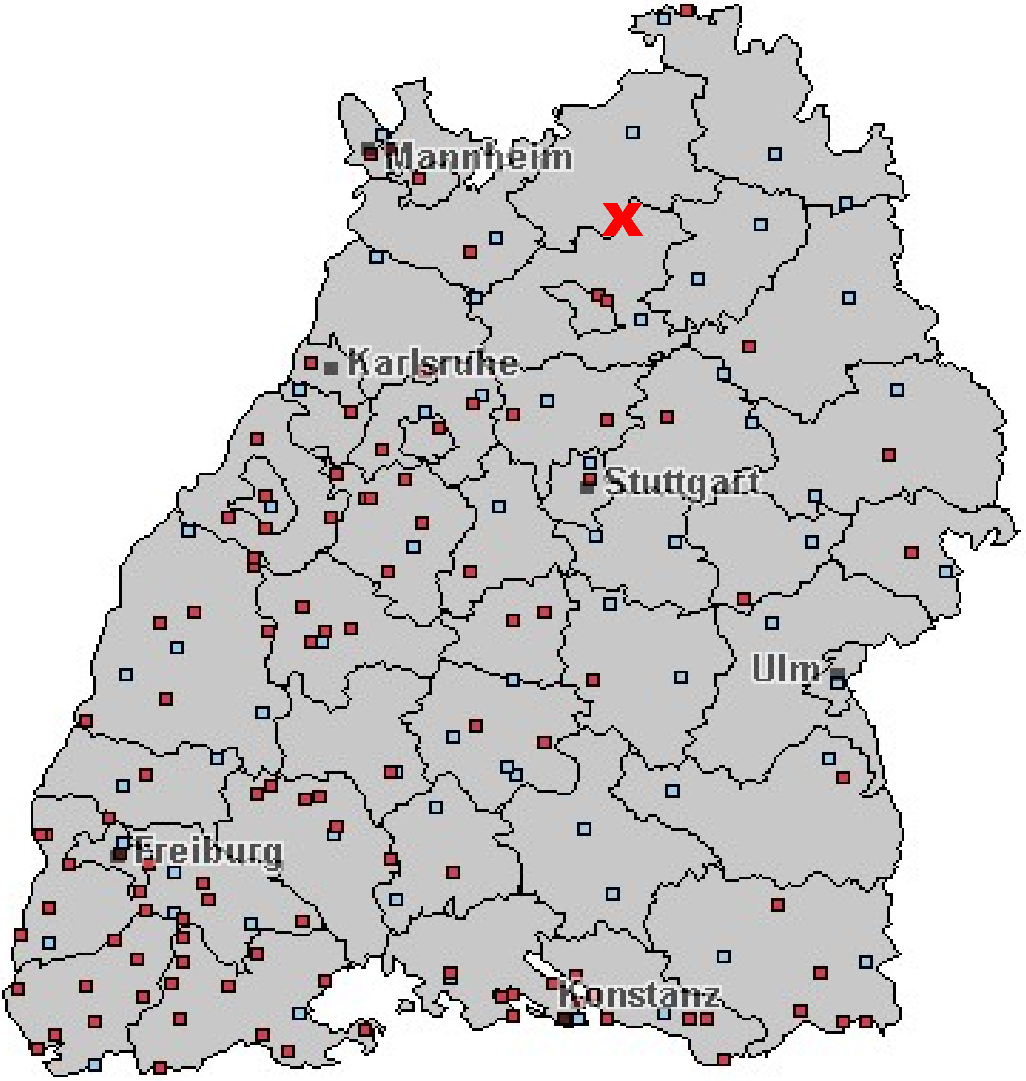
1. K-Nearest-Neighbors
2. Decision Trees / Random Forests
3. Rule Learning
4. Naïve Bayes
5. Support Vector Machines
6. Artificial Neural Networks
7. Deep Neural Networks
8. Many others …

2. K-Nearest-Neighbors

Example Problem

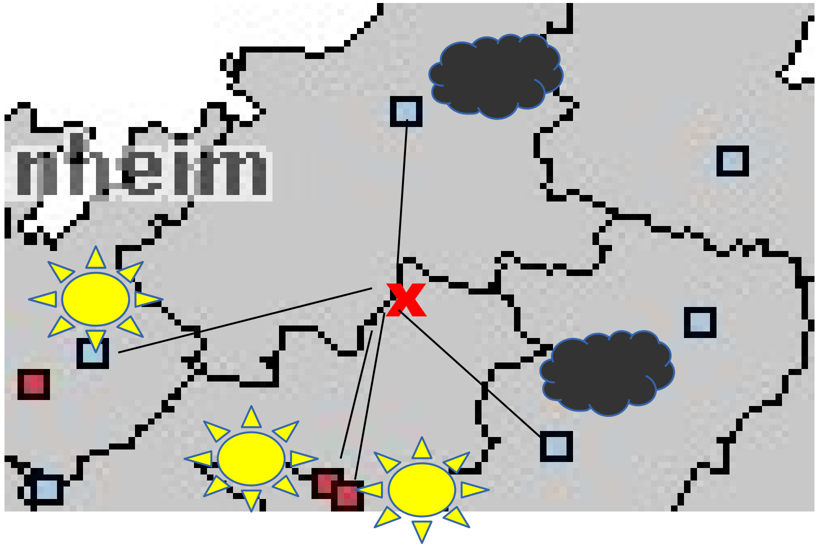
- Predict the current weather in a certain place
- where there is no weather station
- How could you do that?

☀ ? ☁ ?



Basic Idea

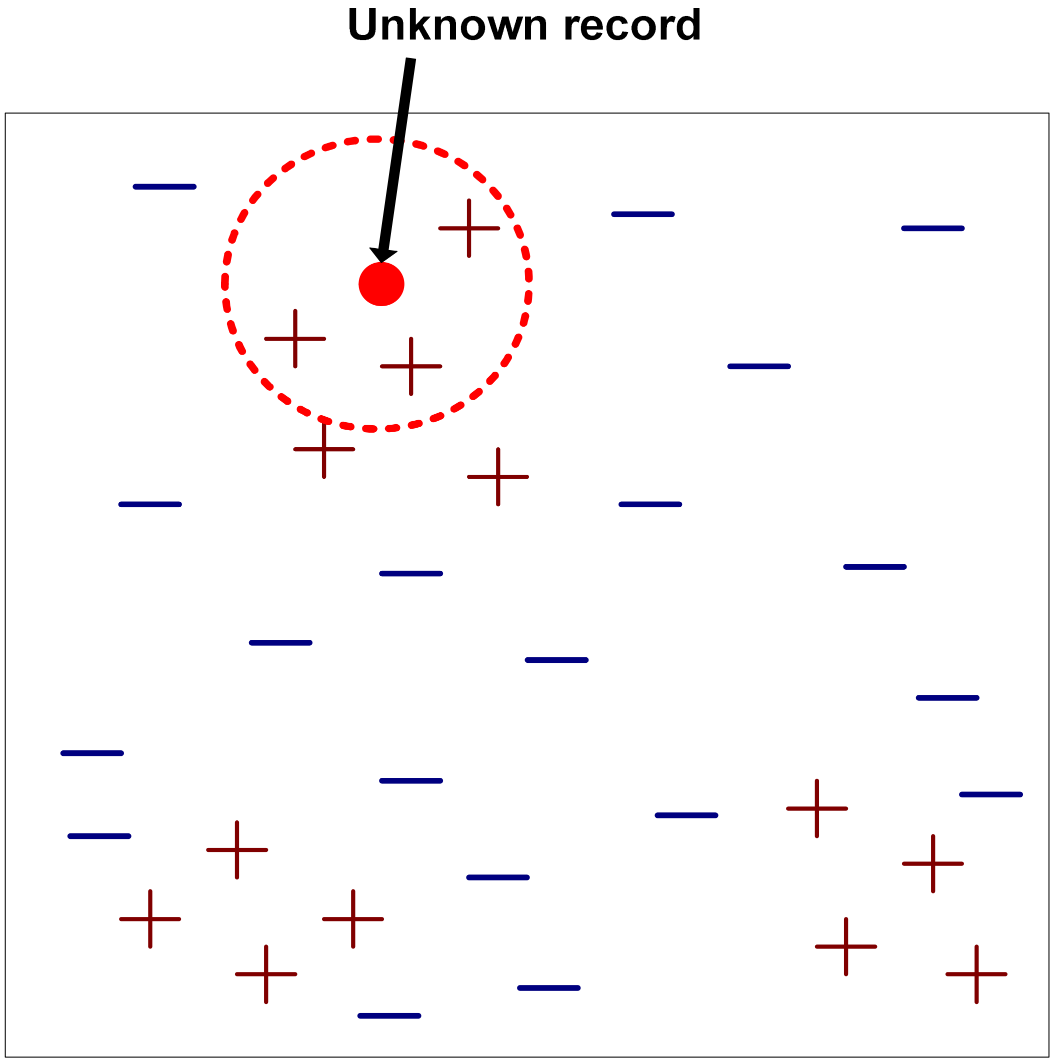
- Use the average forecast of the nearest stations



- Example:
  - 3x sunny
  - 2x cloudy
  - result = sunny

- This approach is called K-Nearest-Neighbors
  - where k is the number of neighbors to consider
  - in the example: k=5
  - in the example: “near” denotes geographical proximity

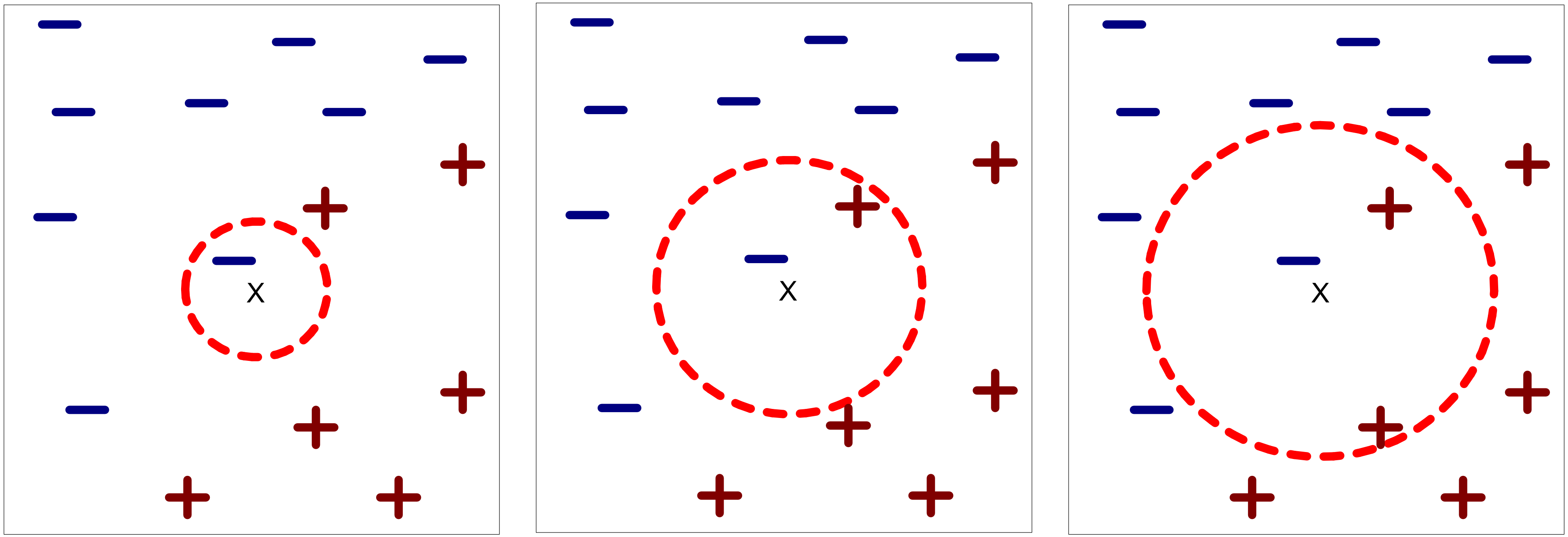
K-Nearest-Neighbors Classifiers



- Require three things
  - A set of stored records
  - A distance measure to compute distance between records
  - The value of k, the number of nearest neighbors to consider
- To classify an unknown record:
  1. Compute distance to each training record
  2. Identify k-nearest neighbors
  3. Use class labels of nearest neighbors to determine the class label of unknown record
     - by taking majority vote or
     - by weighing the vote according to distance

Examples of K-Nearest Neighbors

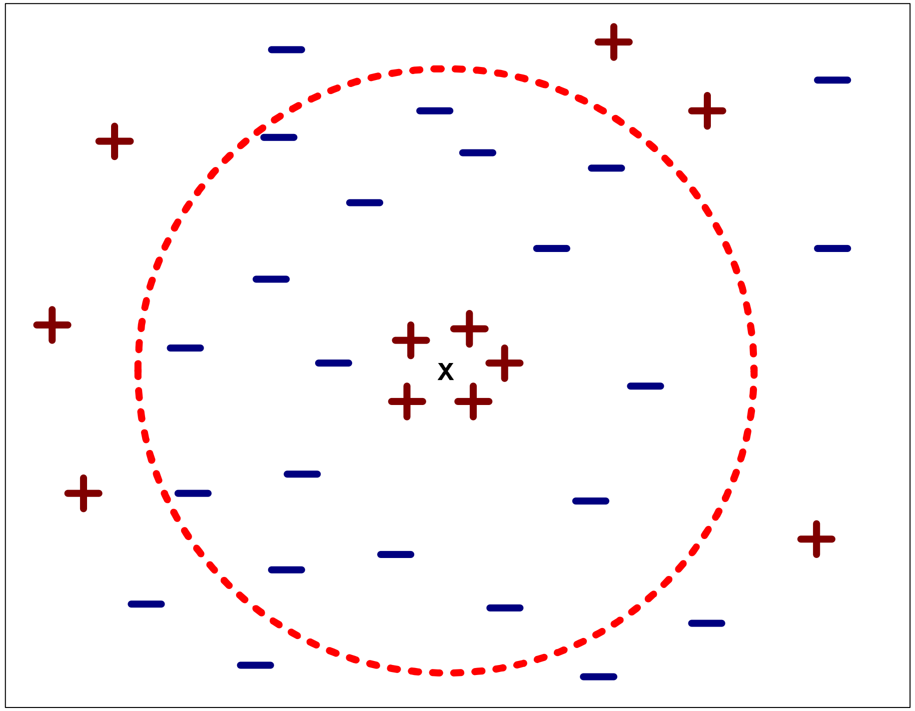
The k-nearest neighbors of a record x are data points that have the k smallest distances to x



(a) 1-nearest neighbor (b) 2-nearest neighbor (c) 3-nearest neighbor

Choosing a Good Value for K

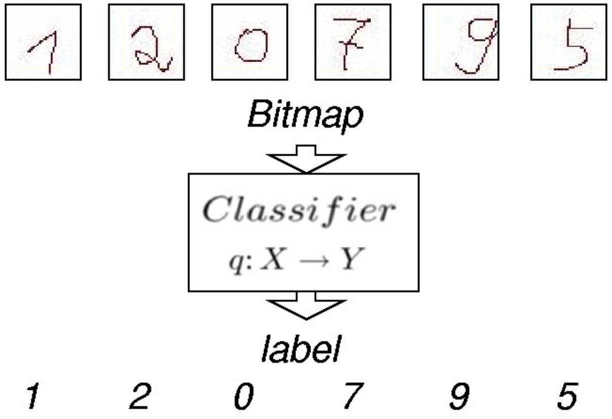
- If k is too small, the result is sensitive to noise points
- If k is too large, the neighborhood may include points from other classes



- Rule of thumb: Test k values between 1 and 20
  - setup: see section on Hyperparameter Selection

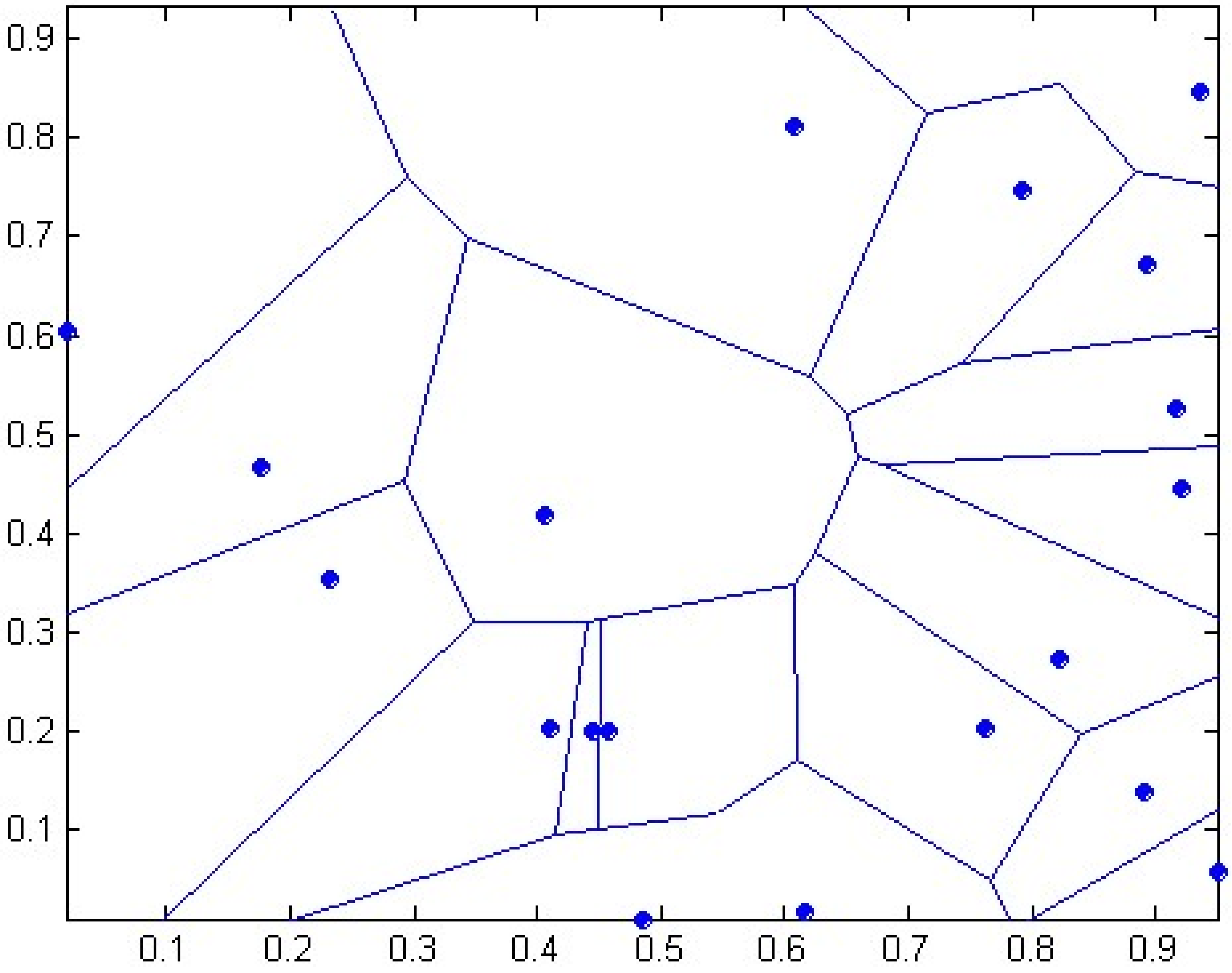
Discussion of K-Nearest-Neighbor Classification

- Often very accurate
  - for instance for optical character recognition (OCR)

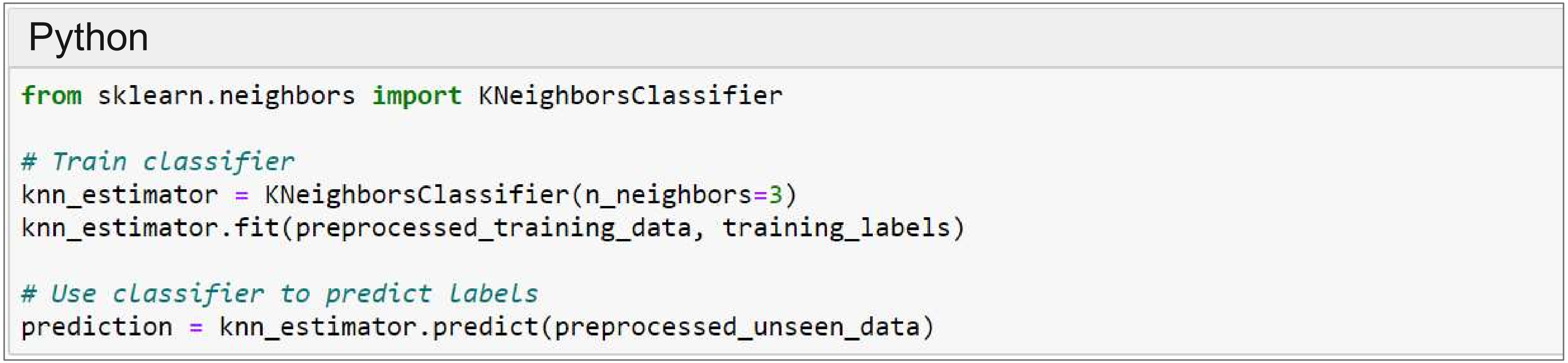


- … but slow
  - as unseen record needs to be compared to all training examples
- Results depend on choosing a good proximity measure
  - attribute weights, asymmetric binary attributes, …
  - see slide set Cluster Analysis
- KNN can handle decision boundaries which are not parallel to the axes (unlike decision trees)

Decision Boundaries of a 1-NN Classifier



KNN Classification in RapidMiner and Python

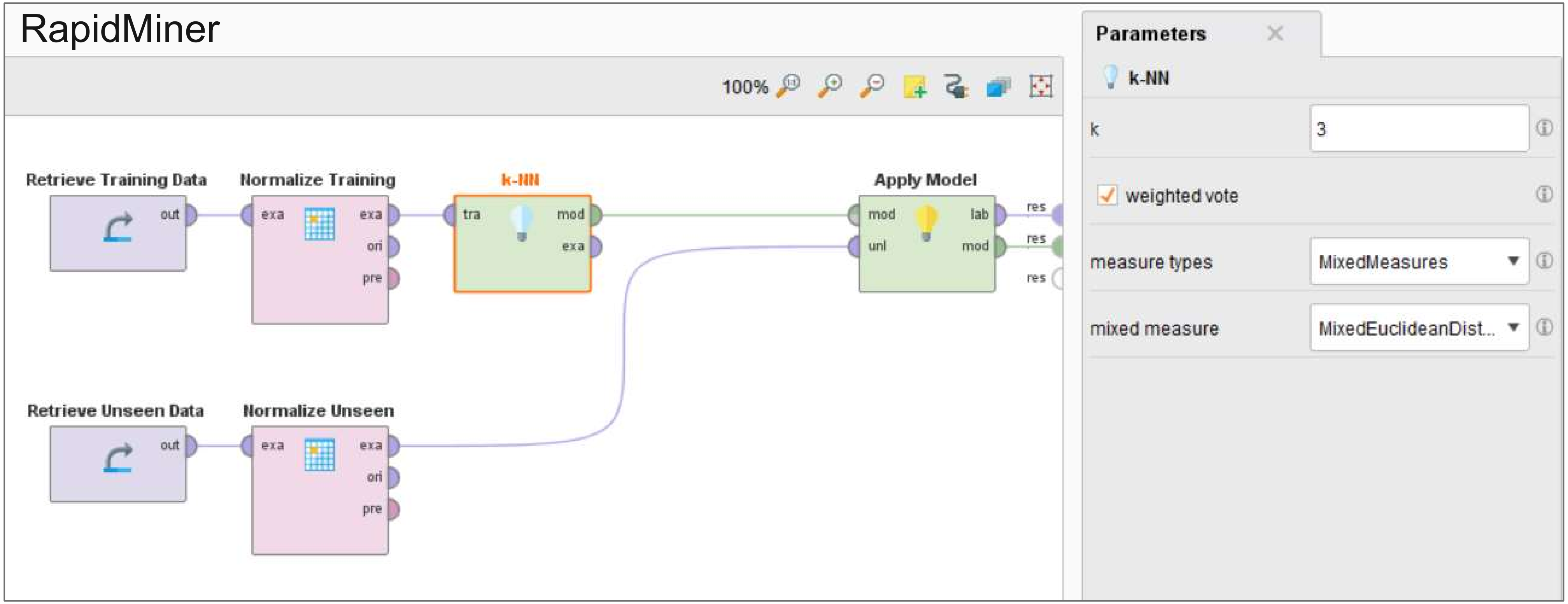


Python

```python
from sklearn.neighbors import KNeighborsClassifier

# Train classifier
knn_estimator = KNeighborsClassifier(n_neighbors=3)
knn_estimator.fit(preprocessed_training_data, training_labels)

# Use classifier to predict labels
prediction = knn_estimator.predict(preprocessed_unseen_data)
```



RapidMiner

Parameters

k-NN

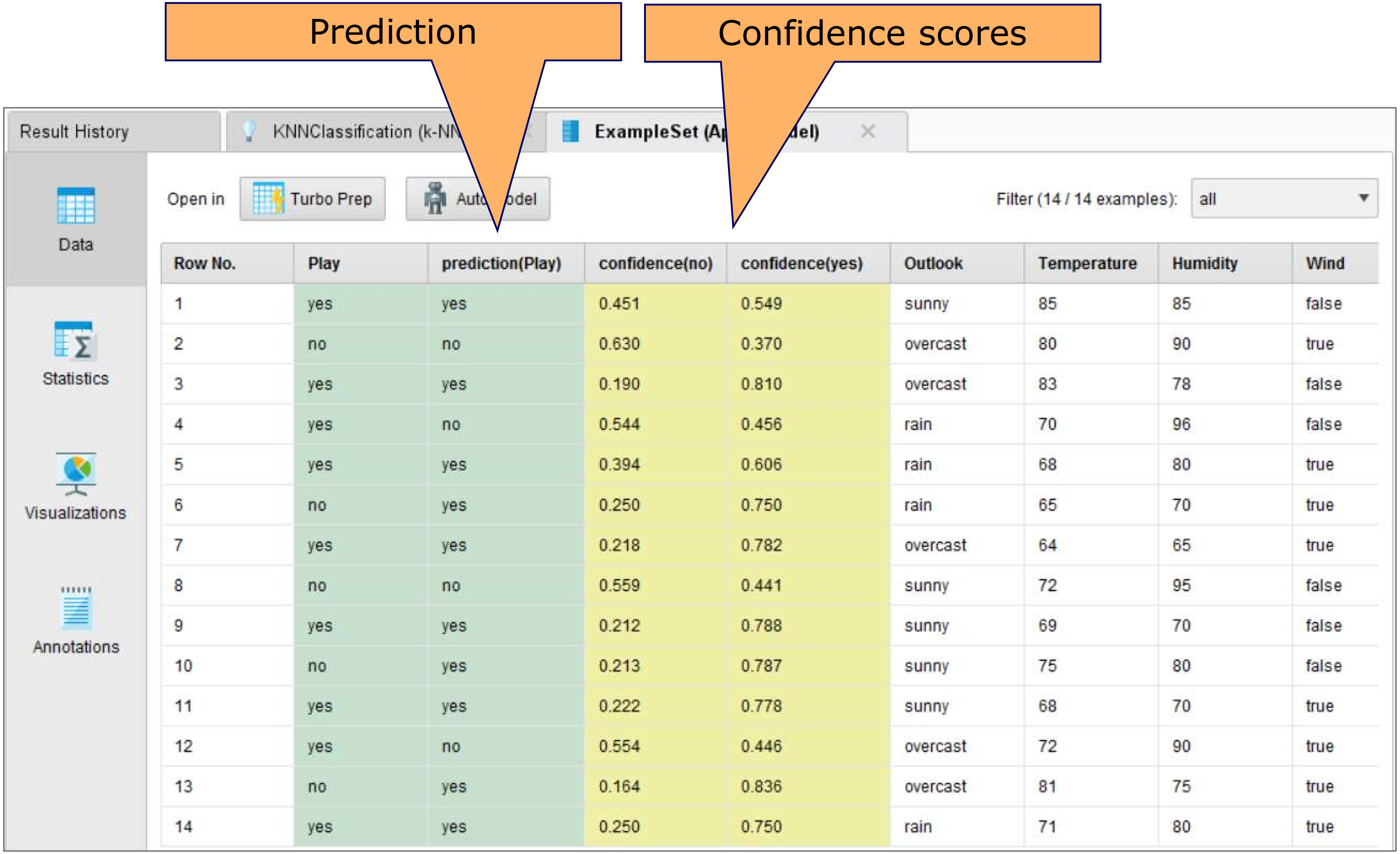
k: 3

☑ weighted vote

measure types: MixedMeasures

mixed measure: MixedEuclideanDist...

Resulting Dataset



Prediction

Confidence scores

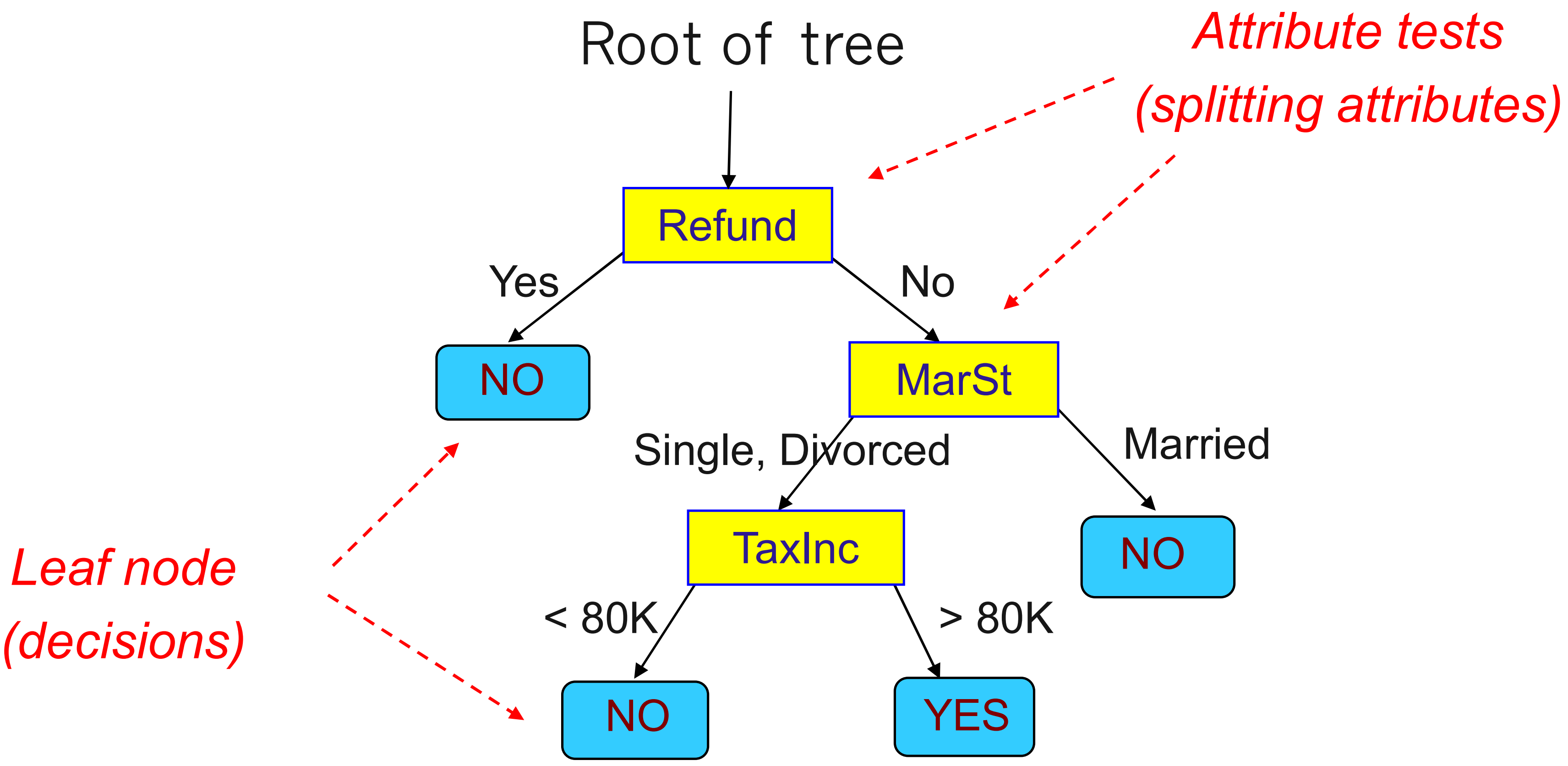
Row No.	Play	prediction(Play)	confidence(no)	confidence(yes)	Outlook	Temperature	Humidity	Wind
1	yes	yes	0.451	0.549	sunny	85	85	false
2	no	no	0.630	0.370	overcast	80	90	true
3	yes	yes	0.190	0.810	overcast	83	78	false
4	yes	no	0.544	0.456	rain	70	96	false
5	yes	yes	0.394	0.606	rain	68	80	true
6	no	yes	0.250	0.750	rain	65	70	true
7	yes	yes	0.218	0.782	overcast	64	65	true
8	no	no	0.559	0.441	sunny	72	95	false
9	yes	yes	0.212	0.788	sunny	69	70	false
10	no	yes	0.213	0.787	sunny	75	80	false
11	yes	yes	0.222	0.778	sunny	68	70	true
12	yes	no	0.554	0.446	overcast	72	90	true
13	no	yes	0.164	0.836	overcast	81	75	true
14	yes	yes	0.250	0.750	rain	71	80	true

Lazy versus Eager Learning

- Lazy Learning
  - Instance-based learning approaches, like KNN, are also called lazy learning as no explicit knowledge (model) is learned
  - Single goal: Classify unseen records as accurately as possible

- Eager Learning
  - but actually, we might have two goals
    1. classify unseen records
    2. understand the application domain as a human
  - Eager learning approaches generate models that are (might be) interpretable by humans
  - Examples of eager techniques: decision tree learning, rule learning

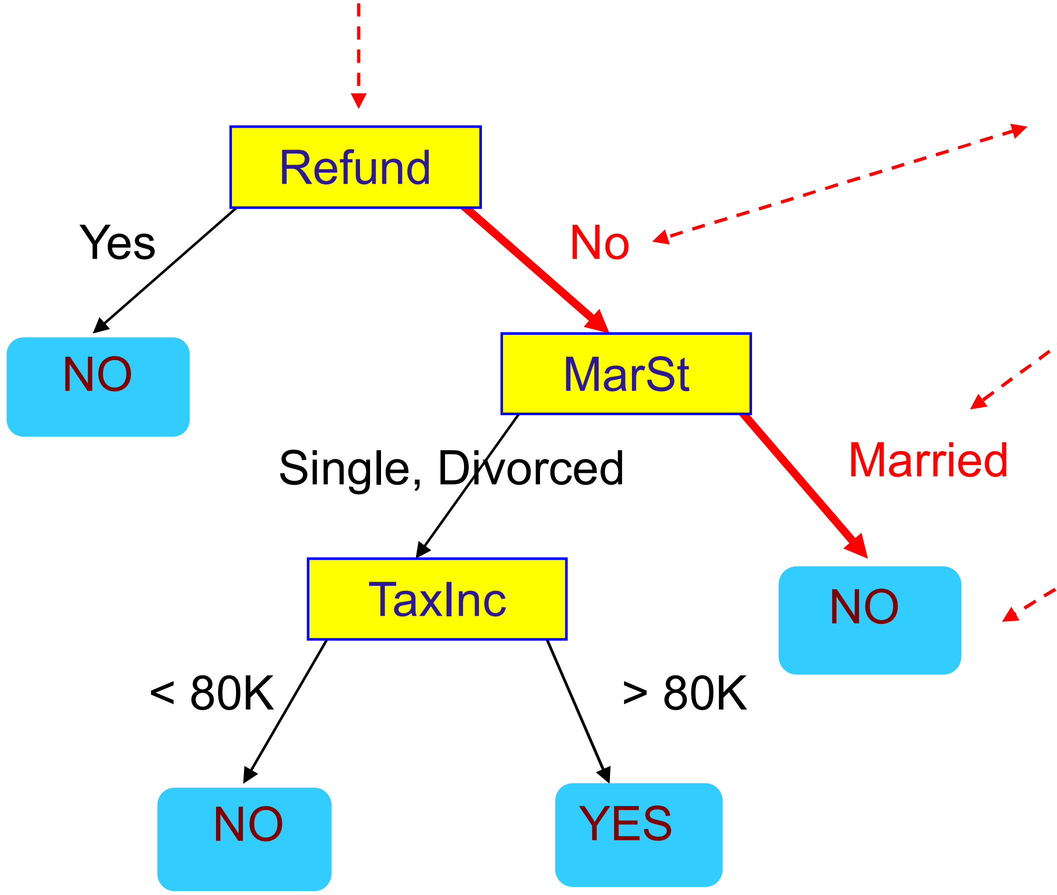
3. Decision Tree Classifiers



Decision trees encode a procedure for taking a classification decision

Applying a Decision Tree to Unseen Data

Start from the root of tree

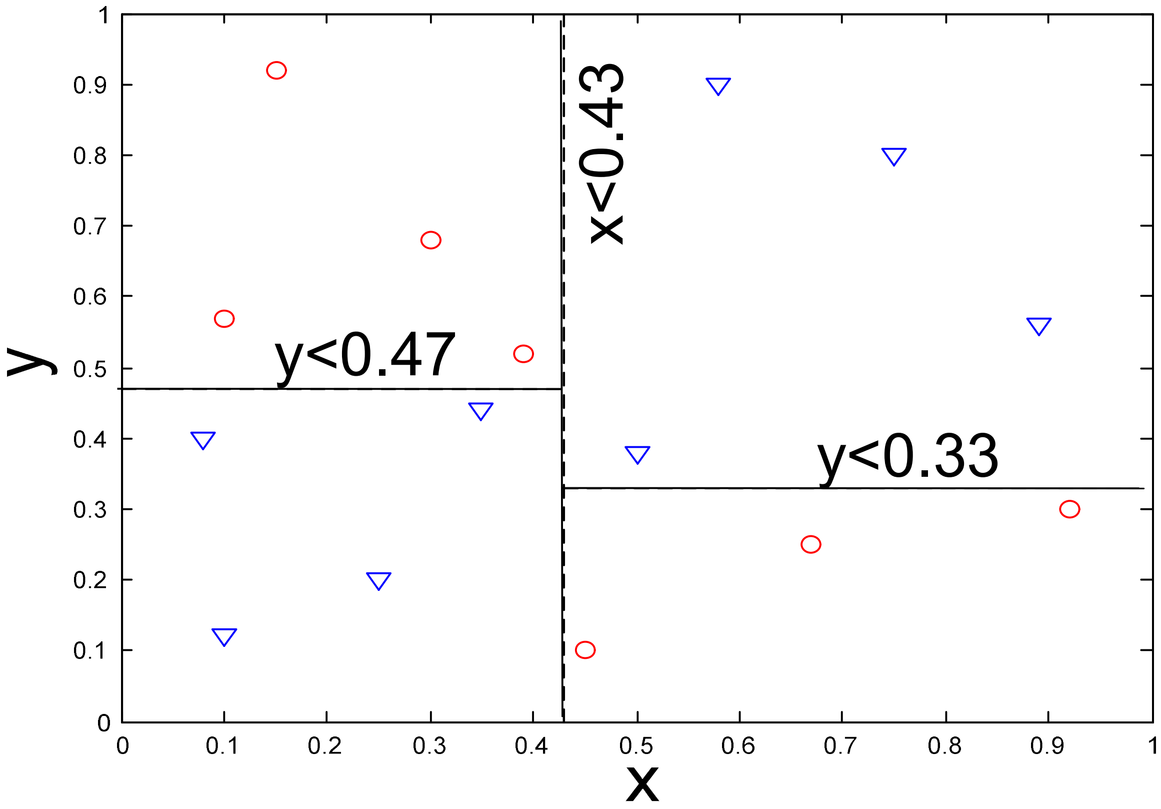


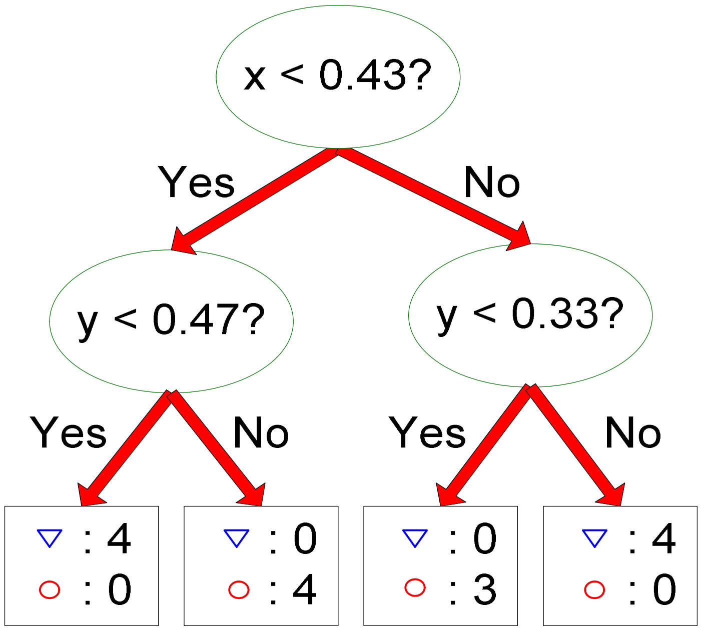
Unseen Record

Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?

Assign Cheat to “No”

Decision Boundary





x < 0.43?

Yes

No

y < 0.47?

y < 0.33?

Yes

No

Yes

No

▽ : 4
○ : 0

▽ : 0
○ : 4

▽ : 0
○ : 3

▽ : 4
○ : 0

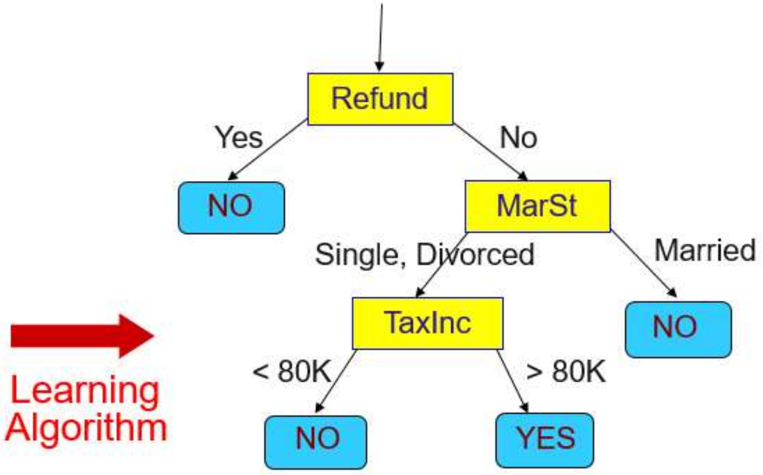
The decision boundaries are parallel to the axes because the test condition involves a single attribute at-a-time

Learning a Decision Tree

- <span>How to learn a decision tree from training data?</span>
  - finding an optimal decision tree is NP-hard
  - tree building algorithms thus use a greedy, top-down, recursive partitioning strategy to induce a reasonable solution
- Many different algorithms have been proposed:
  - Hunt's Algorithm
  - ID3
  - C4.5
  - CHAID

	categorical	categorical	continuous	class
Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

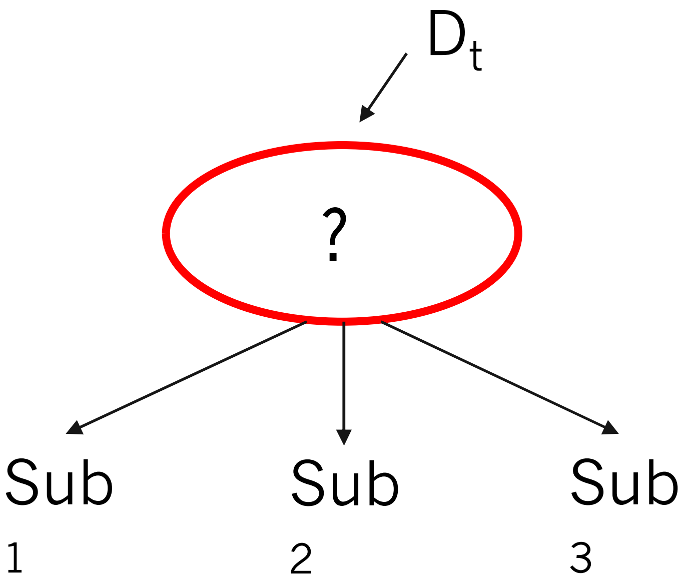
Training Data



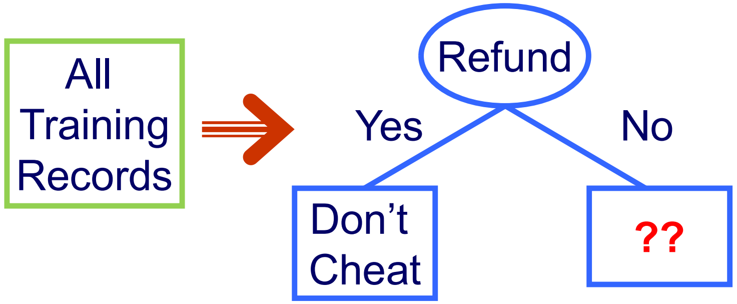
Model: Decision Tree

Hunt's Algorithm

- Let D_t be the set of training records that reach a node t
- Generate leaf node or attribute test:
  - if D_t only contains records that belong to the same class y_t, then t is a leaf node labeled as y_t
  - if D_t contains records that belong to more than one class, use an attribute test to split the data into subsets having a higher purity.
    - for all possible tests: calculate purity of the resulting subsets
    - choose test resulting in highest purity
- Recursively apply this procedure to each subset



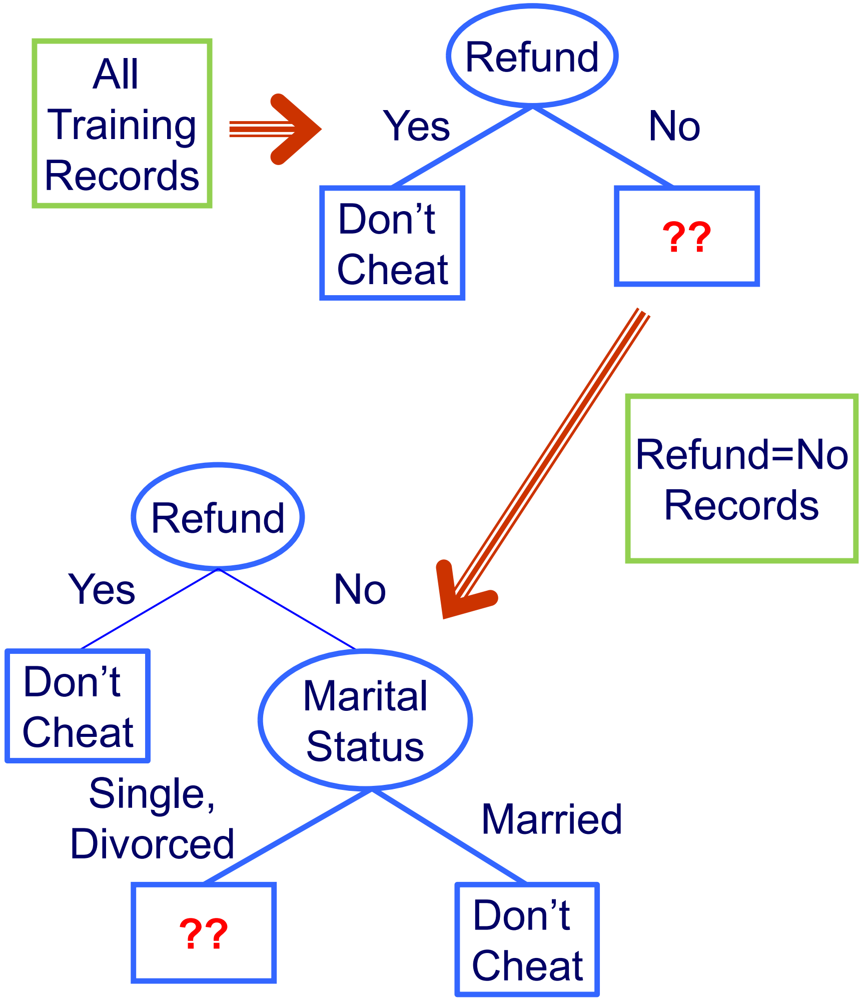
Hunt’s Algorithm – Step 1



1. We calculate the purity of the resulting subsets for all possible splits
   - Purity of split on Refund
   - Purity of split on Marital Status
   - Purity of split on Taxable Income
2. We find the split on Refund to produce the purest subsets

Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	**No**
2	No	Married	100K	**No**
3	No	Single	70K	**No**
4	Yes	Married	120K	**No**
5	No	Divorced	95K	**Yes**
6	No	Married	60K	**No**
7	Yes	Divorced	220K	**No**
8	No	Single	85K	**Yes**
9	No	Married	75K	**No**
10	No	Single	90K	**Yes**

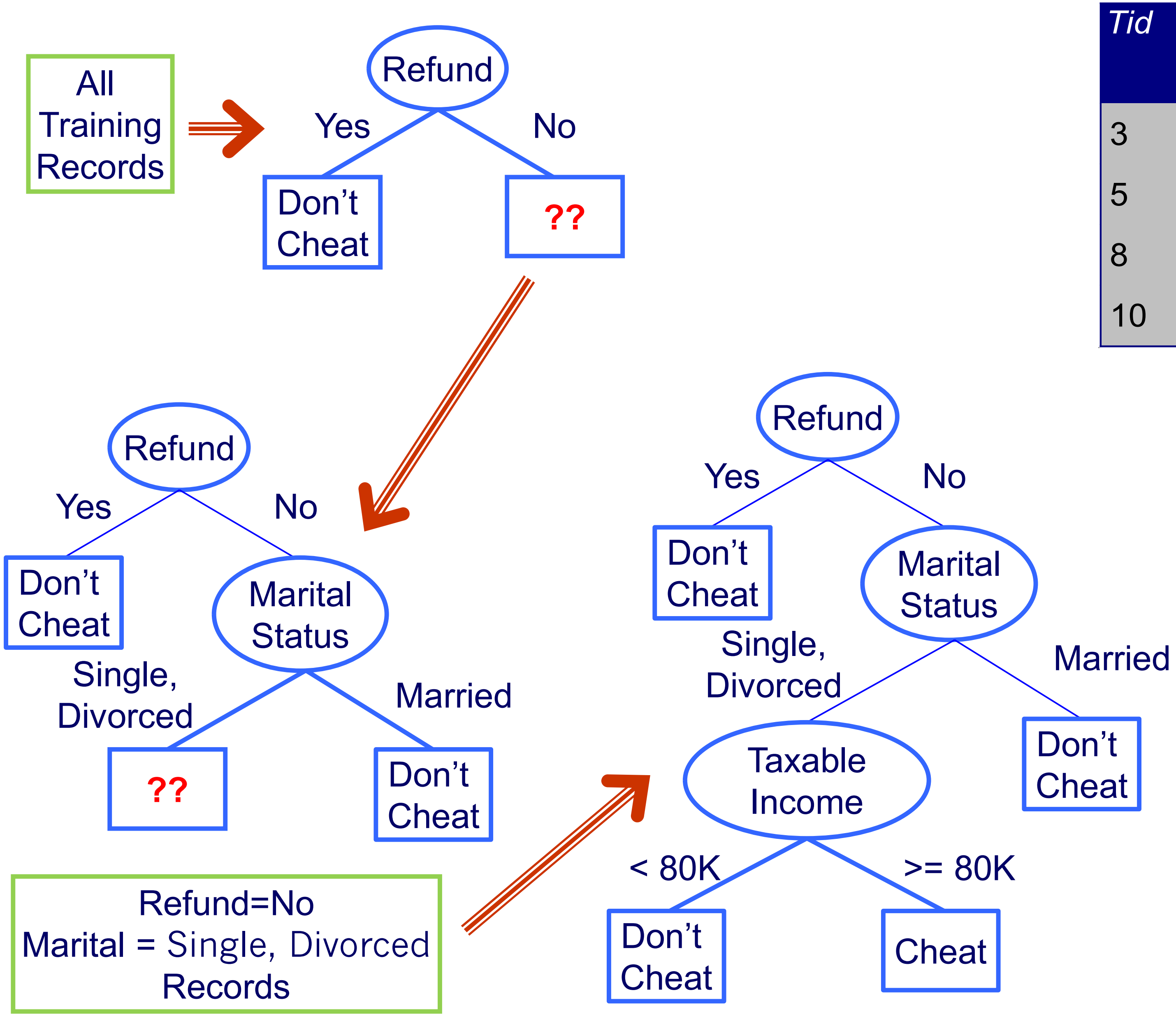
Hunt's Algorithm – Step 2



Tid	Refund	Marital Status	Taxable Income	Cheat
2	No	Married	100K	**No**
3	No	Single	70K	**No**
5	No	Divorced	95K	**Yes**
6	No	Married	60K	**No**
8	No	Single	85K	**Yes**
9	No	Married	75K	**No**
10	No	Single	90K	**Yes**

1. We further examine the Refund=No records
2. Again, we test all possible splits
3. We find the split on Marital Status to produce the purest subsets

Hunt’s Algorithm – Step 3



Tid	Refund	Marital Status	Taxable Income	Cheat
3	No	Single	70K	**No**
5	No	Divorced	95K	**Yes**
8	No	Single	85K	**Yes**
10	No	Single	90K	**Yes**

1. We further examine the Marital Status=Single or =Divorced records
2. We find a split on Taxable Income to produce pure subsets
3. We stop splitting as no sets containing different classes are left

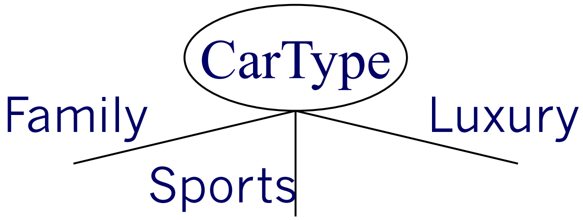
Design Issues for Learning Decision Trees

1. How should training records be split?
   - How to specify the attribute test condition?
     - Depends on number of ways to split: 2-way split, multi-way split
     - Depends on attribute data type: nominal, ordinal, continuous
   - How to determine the best split?
     - Different purity measures can be used

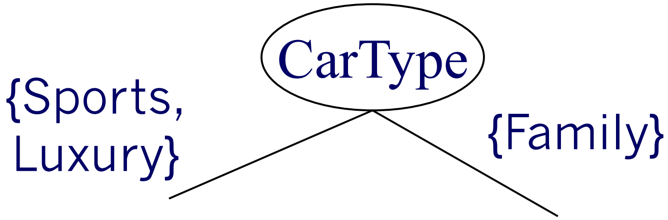
2. When should the splitting procedure stop?
   - Shallow trees might generalize better to unseen records
   - Fully grown trees might overfit training data

Splitting Based on Nominal Attributes

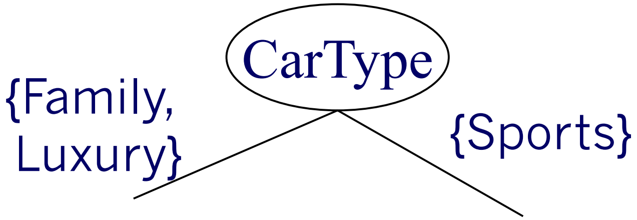
- Multi-way split: Use as many partitions as distinct values



- Binary split: Divides values into two subsets

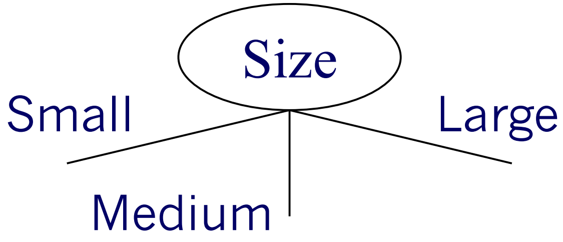


OR

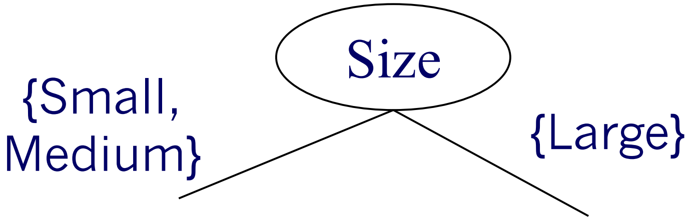


Splitting Based on Ordinal Attributes

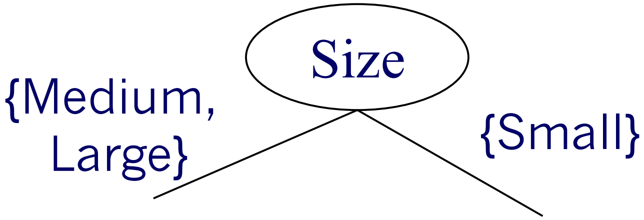
- **Multi-way split:** Use as many partitions as distinct values



- **Binary split:** Divides values into two subsets while keeping the order



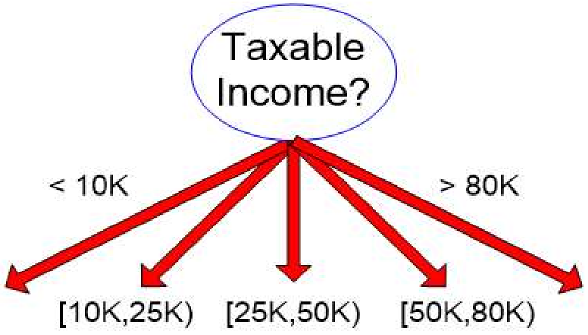
OR



Splitting Based on Continuous Attributes

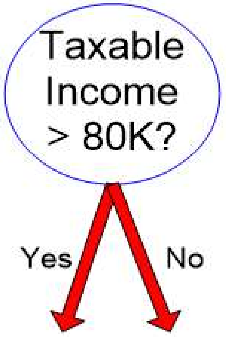
– Different ways of handling continuous attributes

- Discretization to form an ordinal categorical attribute
  - equal-interval binning
  - equal-frequency binning
  - binning based on user-provided boundaries



(ii) Multi-way split

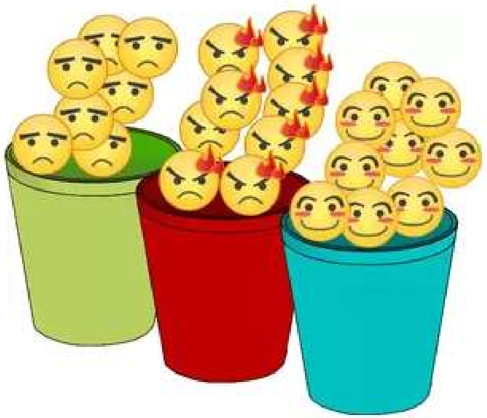
- Binary Decision: (A < v) or (A ≥ v)
  - usually sufficient in practice
  - find the best splitting border v based on a purity measure (discussed later)
  - can be compute intensive



(i) Binary split

Discretization Example

- Values of the attribute, e.g., age of a person:
  - 0, 4, 12, 16, 16, 18, 24, 26, 28



- Equal-interval binning – for bin width of e.g., 10:

• Bin 1: 0, 4	[-,10) bin
• Bin 2: 12, 16, 16, 18	[10,20) bin
• Bin 3: 24, 26, 28	[20,+) bin

– denote negative infinity, + positive infinity

- Equal-frequency binning – for bin density of e.g., 3:

• Bin 1: 0, 4, 12	[-, 14) bin
• Bin 2: 16, 16, 18	[14, 21) bin
• Bin 3: 24, 26, 28	[21,+] bin

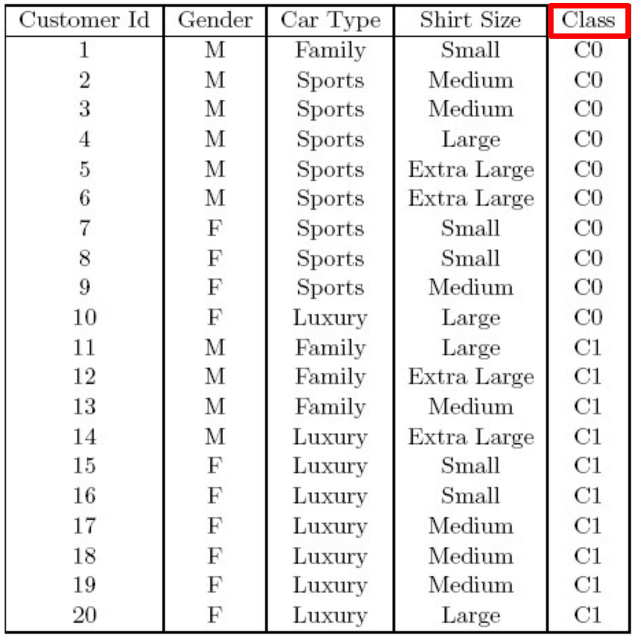
3.2 How to Find the Best Split?

Before splitting the dataset contains:

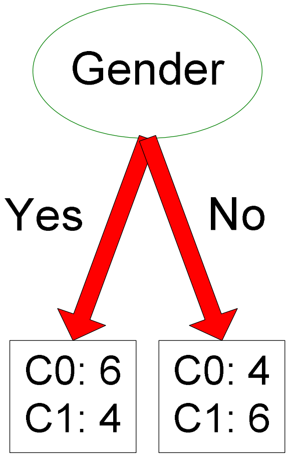
– 10 records of class C0 and

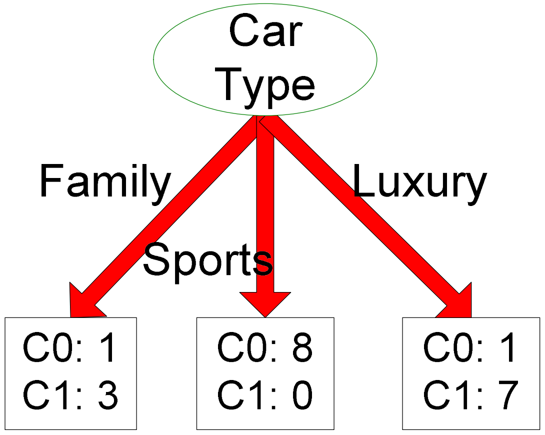
– 10 records of class C1

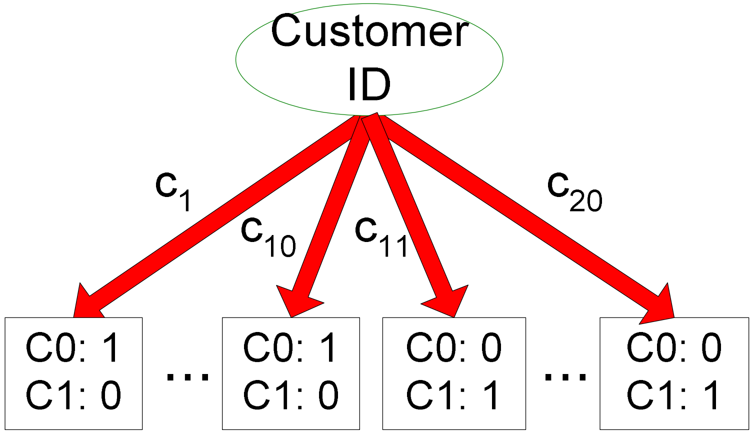
Which attribute test is the best?



Customer Id	Gender	Car Type	Shirt Size	Class
1	M	Family	Small	C0
2	M	Sports	Medium	C0
3	M	Sports	Medium	C0
4	M	Sports	Large	C0
5	M	Sports	Extra Large	C0
6	M	Sports	Extra Large	C0
7	F	Sports	Small	C0
8	F	Sports	Small	C0
9	F	Sports	Medium	C0
10	F	Luxury	Large	C0
11	M	Family	Large	C1
12	M	Family	Extra Large	C1
13	M	Family	Medium	C1
14	M	Luxury	Extra Large	C1
15	F	Luxury	Small	C1
16	F	Luxury	Small	C1
17	F	Luxury	Medium	C1
18	F	Luxury	Medium	C1
19	F	Luxury	Medium	C1
20	F	Luxury	Large	C1







How to Find the Best Split?

- Greedy approach: Test all possible splits and use the one that results in the most homogeneous (= pure) nodes
- Need a measure of node impurity:

C0: 5
C1: 5

Non-homogeneous

High degree of node impurity

C0: 9
C1: 1

Homogeneous

Low degree of node impurity

- Common measures of node impurity:
  1. GINI Index
  2. Entropy

How to Find the Best Split?

1. Compute impurity measure (P) before splitting
2. Compute impurity measure (M) after splitting for all possible splits
   - compute impurity measure of each child node
   - M is the weighted impurity of children
3. Choose the attribute test condition (split) that produces the highest purity gain

$$\text{Gain} = P - M$$

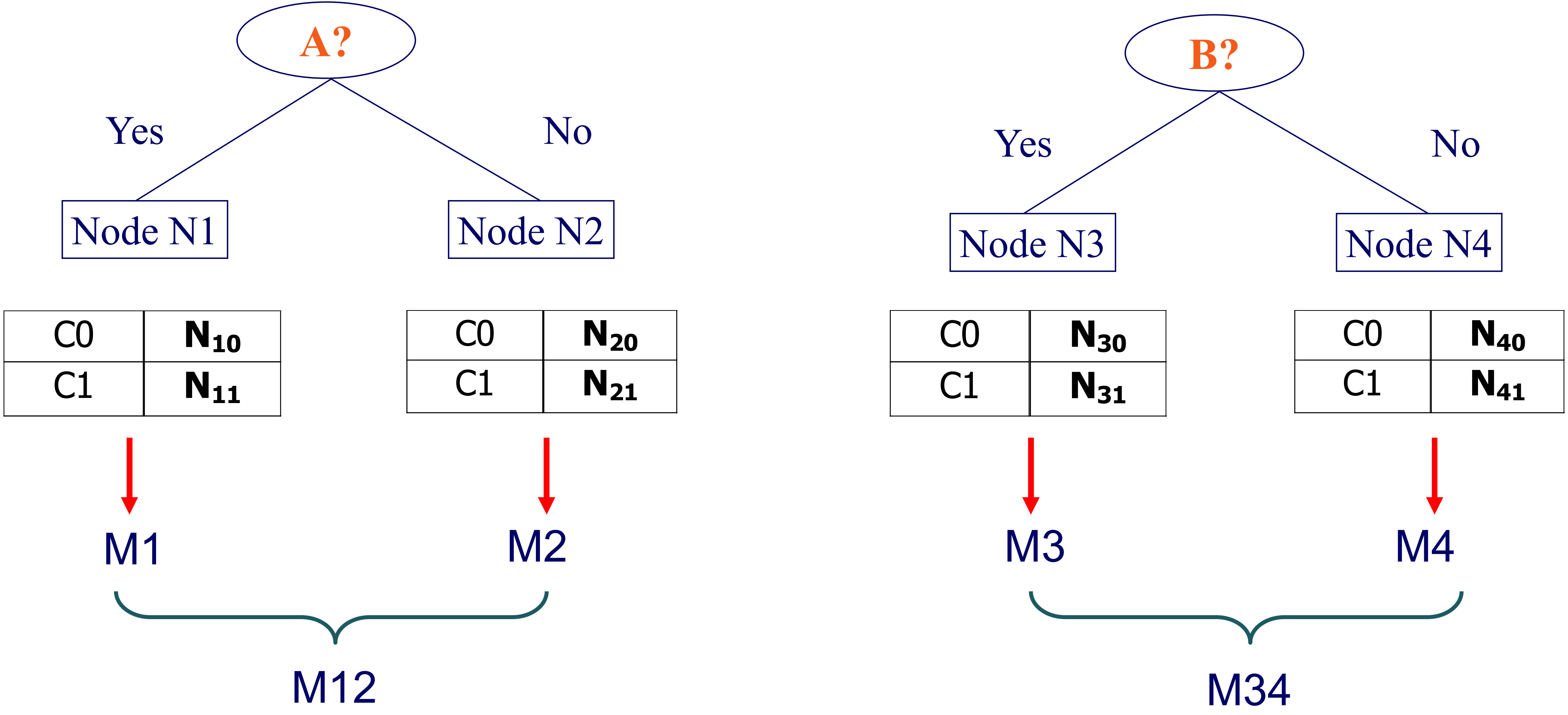
or equivalently, lowest impurity measure after splitting (M)

Comparing Two Splits by Purity Gain

Before Splitting:

C0	$\mathbf{N_{00}}$
C1	$\mathbf{N_{01}}$

→ P



A?

Yes

No

Node N1

Node N2

C0	$\mathbf{N_{10}}$
C1	$\mathbf{N_{11}}$

C0	$\mathbf{N_{20}}$
C1	$\mathbf{N_{21}}$

M1

M2

M12

B?

Yes

No

Node N3

Node N4

C0	$\mathbf{N_{30}}$
C1	$\mathbf{N_{31}}$

C0	$\mathbf{N_{40}}$
C1	$\mathbf{N_{41}}$

M3

M4

M34

Higher purity gain? P – M12 or P – M34

3.2.1 Impurity Measure: GINI Index

- GINI Index for a given node t :

$$GINI(t) = 1 - \sum_{j} [p(j \mid t)]^2$$

$p(j \mid t)$ is the relative frequency of class j at node t

- Minimum (0.0) when all records belong to one class
- Maximum (1 - 1/n_c) when records are equally distributed among all classes. n_c = number of classes

C1	**0**
C2	**6**
Gini=0.000	

C1	**1**
C2	**5**
Gini=0.278	

C1	**2**
C2	**4**
Gini=0.444	

C1	**3**
C2	**3**
Gini=0.500	

Examples for computing GINI

$$GINI(t) = 1 - \sum_{j} [p(j \mid t)]^2$$

C1	**0**
C2	**6**

P(C1) = 0/6 = 0   P(C2) = 6/6 = 1

Gini = 1 – $P(C1)^2$ – $P(C2)^2$ = 1 – 0 – 1 = 0

C1	**1**
C2	**5**

P(C1) = 1/6   P(C2) = 5/6

Gini = 1 – $(1/6)^2$ – $(5/6)^2$ = 0.278

C1	**2**
C2	**4**

P(C1) = 2/6   P(C2) = 4/6

Gini = 1 – $(2/6)^2$ – $(4/6)^2$ = 0.444

Splitting Based on GINI

- When a node p is split into k partitions (subsets), the GINI index of each partition is weighted according to the partition's size
- The quality of the overall split is computed as:

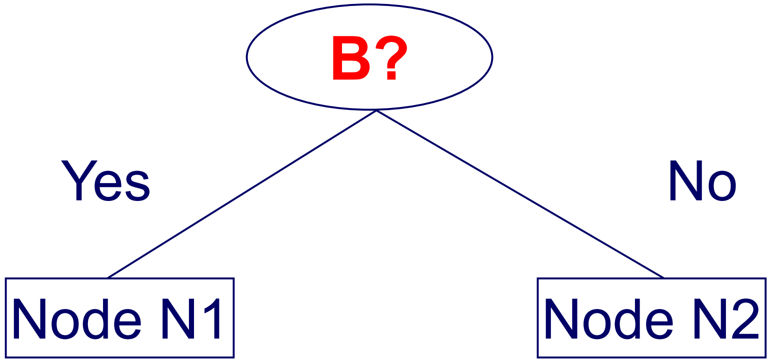
$$GINI_{split} = \sum_{i=1}^{k} \frac{n_i}{n} GINI(i)$$

where: n_i = number of records at child i

n = number of records at node p

Example: Calculating the Purity Gain of a Split

Split into two partitions



	Parent
C1	**6**
C2	**6**
Gini = 0.500	

GINI_{N1}
$= 1 – (5/7)^2 – (2/7)^2$
$= 0.408$

GINI_{N2}
$= 1 – (1/5)^2 – (4/5)^2$
$= 0.32$

	N1	N2
C1	**5**	**1**
C2	**2**	**4**
Gini=0.371		

GINI_{Split}
$= 7/12 * 0.408 +$
$5/12 * 0.32$
$= 0.371$

Weights →

Purity Gain = 0.5 – 0.371 = 0.129

Categorical Attributes: Computing Gini Index

For each distinct attribute value, gather counts for each class

Multi-way split

	CarType		
	Family	Sports	Luxury
C1	1	2	1
C2	4	1	1
Gini	0.393		

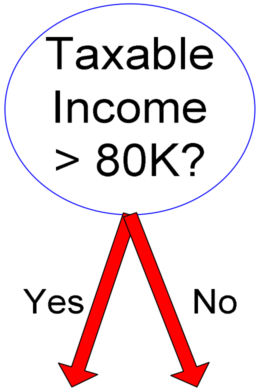
Two-way split
(find best partition of values)

	CarType	
	{Sports, Luxury}	{Family}
C1	3	1
C2	2	4
Gini	0.400	

	CarType	
	{Sports}	{Family, Luxury}
C1	2	2
C2	1	5
Gini	0.419	

Continuous Attributes: Computing Gini Index

- How to find the best binary split for a continuous attribute?
- Efficient computation:
  1. sort the attribute on values
  2. choose split positions at the middle between two values
  3. linearly scan these values, each time updating the count matrix and computing the gini index
  4. choose the split position that has the smalest gini index



	Taxable Income																					
Sorted Values		60		70		75		85		90		95		100		120		125		220		
Split Positions	55		65		72		80		87		92		97		110		122		172		230	
	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>
Yes	0	3	0	3	0	3	0	3	1	2	2	1	3	0	3	0	3	0	3	0	3	0
No	0	7	1	6	2	5	3	4	3	4	3	4	3	4	4	3	5	2	6	1	7	0
Gini	0.420		0.400		0.375		0.343		0.417		0.400		*0.300*		0.343		0.375		0.400		0.420	

3.2.2 Alternative Impurity Measure: Information Gain

- Information gain relies on the entropy of each node
- Entropy of a given node t:

$$Entropy(t) = -\sum_j p(j \mid t) \log_2 p(j \mid t)$$

$p(j \mid t)$ is the relative frequency of class j at node t

- Entropy measures homogeneity of a node
  - Minimum (0.0) when all records belong to one class
  - Maximum ($\log_2 n_c$) when records are equally distributed among all classes

Examples for Computing Entropy

$$Entropy(t) = -\sum_j p(j \mid t) \log_2 p(j \mid t)$$

C1	**0**
C2	**6**

$P(C1) = 0/6 = 0$ $P(C2) = 6/6 = 1$

$\text{Entropy} = -\,0 \log_2 0 - 1 \log_2 1 = -\,0 - 0 = 0$

C1	**1**
C2	**5**

$P(C1) = 1/6$ $P(C2) = 5/6$

$\text{Entropy} = -\,(1/6) \log_2 (1/6) - (5/6) \log_2 (5/6) = 0.65$

C1	**2**
C2	**4**

$P(C1) = 2/6$ $P(C2) = 4/6$

$\text{Entropy} = -\,(2/6) \log_2 (2/6) - (4/6) \log_2 (4/6) = 0.92$

Splitting Based on Information Gain

– Information Gain:

$$GAIN_{split} = Entropy(p) - \left(\sum_{i=1}^{k} \frac{n_i}{n} Entropy(i) \right)$$

Parent Node p is split into k partitions;

n_i is number of records in partition i

– Information gain measures the entropy reduction of a split

– We choose the split with the largest reduction (maximal GAIN)

– Disadvantage: Tends to prefer splits that result in large number of partitions, each being small but pure (split by ID attribute?)

3.2.3 Alternative Splitting Criterion: GainRATIO

- GainRATIO is designed to overcome the tendency to generate a large number of small partitions
- GainRATIO adjusts information gain by the entropy of the partitioning (SplitINFO)
- Higher entropy of the partitioning (large number of small partitions) is penalized!

$$GainRATIO_{split} = \frac{GAIN_{Split}}{SplitINFO}$$

$$SplitINFO = -\sum_{i=1}^{k} \frac{n_i}{n} \log \frac{n_i}{n}$$

Parent Node p is split into k partitions

n_i is the number of records in partition i

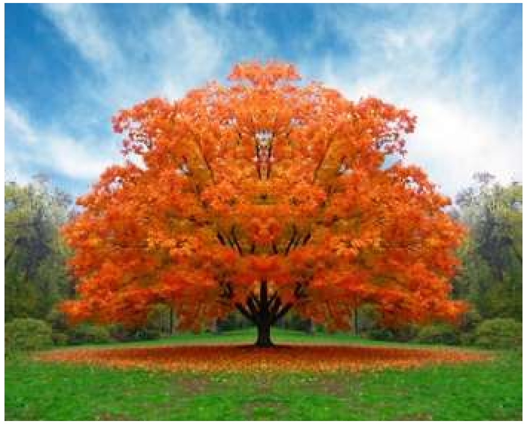
3.3 Overfitting

- We want to learn models that are good at classifying unseen records
- Overfitting: Learned models can fit the training data too closely and therefore work poorly on unseen data

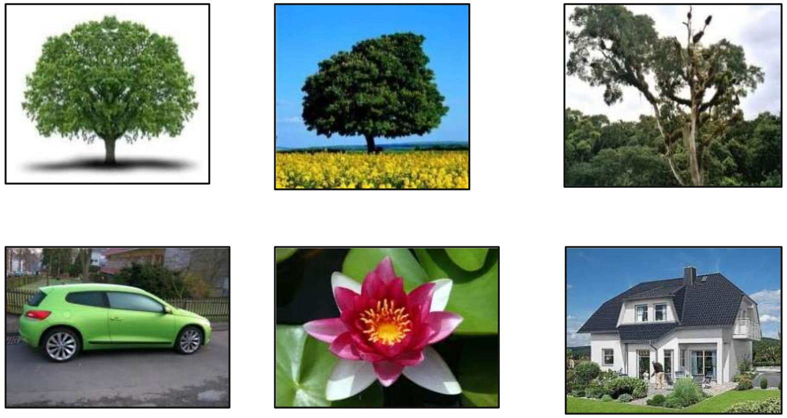
- Model perfectly fitting the training data:

  "Trees are big, green plants that have a trunk and no wheels“

- Unseen example:



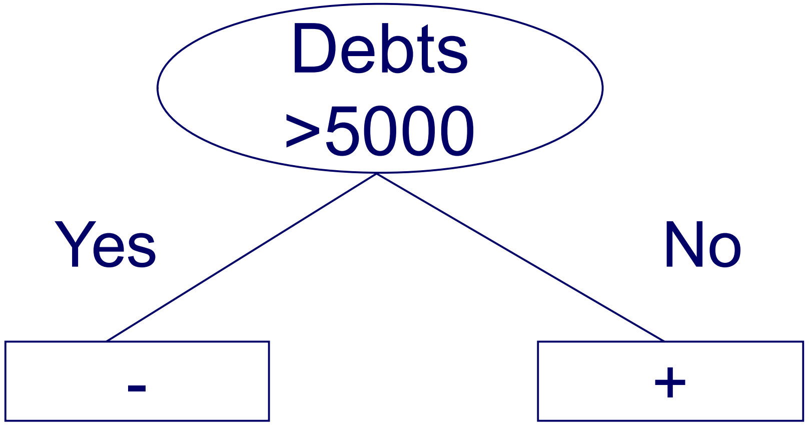
Training data



- Goal: Find good compromise between specificness and generality of the learned model

Overfitting: Second Example

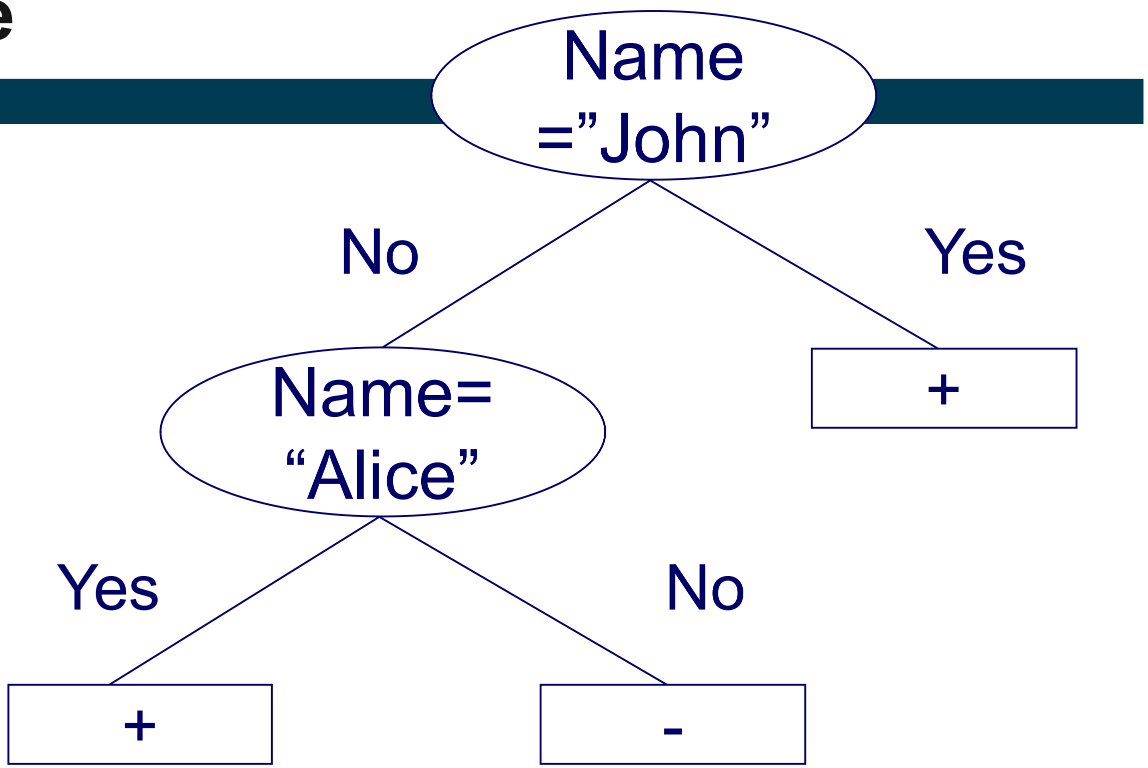
- Example: Predict credit rating
  - possible decision tree:



Name	Net Income	Job status	Debts	Rating
John	40000	employed	0	+
Mary	38000	employed	10000	-
Stephen	21000	self-employed	20000	-
Eric	2000	student	10000	-
Alice	35000	employed	4000	+

Overfitting: Second Example

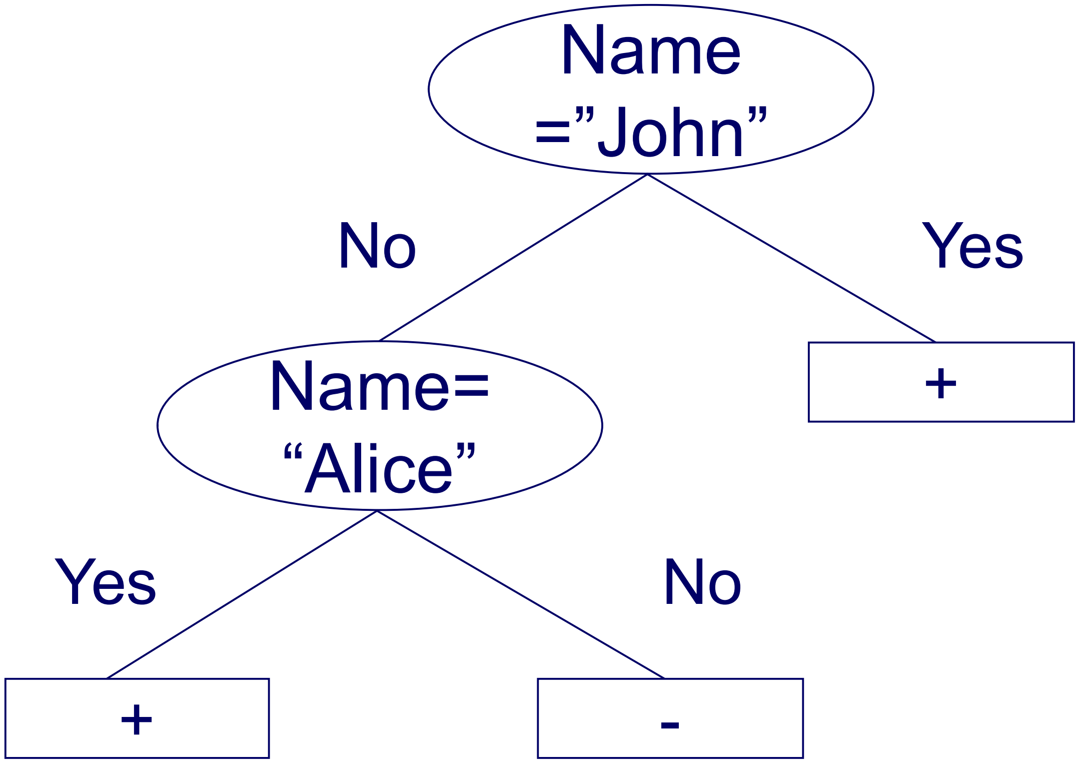
- Example: Predict credit rating
  - alternative decision tree:

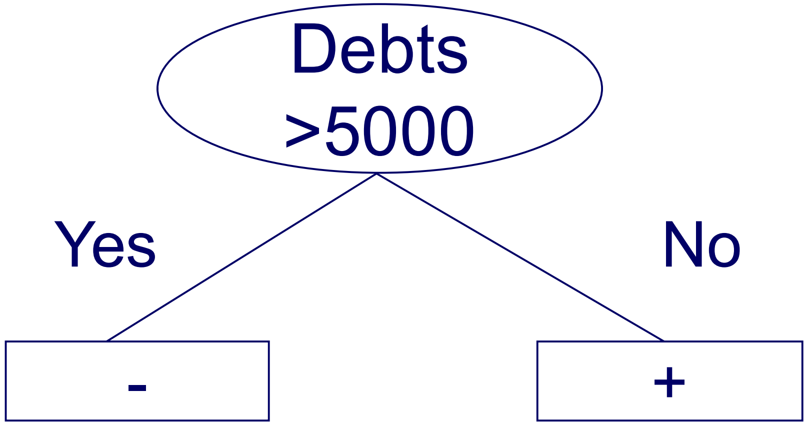


Name	Net Income	Job status	Debts	Rating
John	40000	employed	0	+
Mary	38000	employed	10000	-
Stephen	21000	self-employed	20000	-
Eric	2000	student	10000	-
Alice	35000	employed	4000	+

Overfitting: Second Example

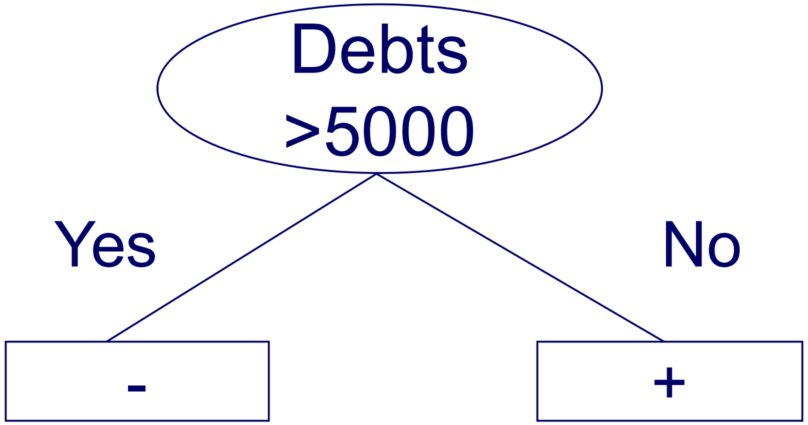
- Both trees seem equally good
  - as they classify all instances in the training set correctly
- Which one do you prefer?





Occam's Razor

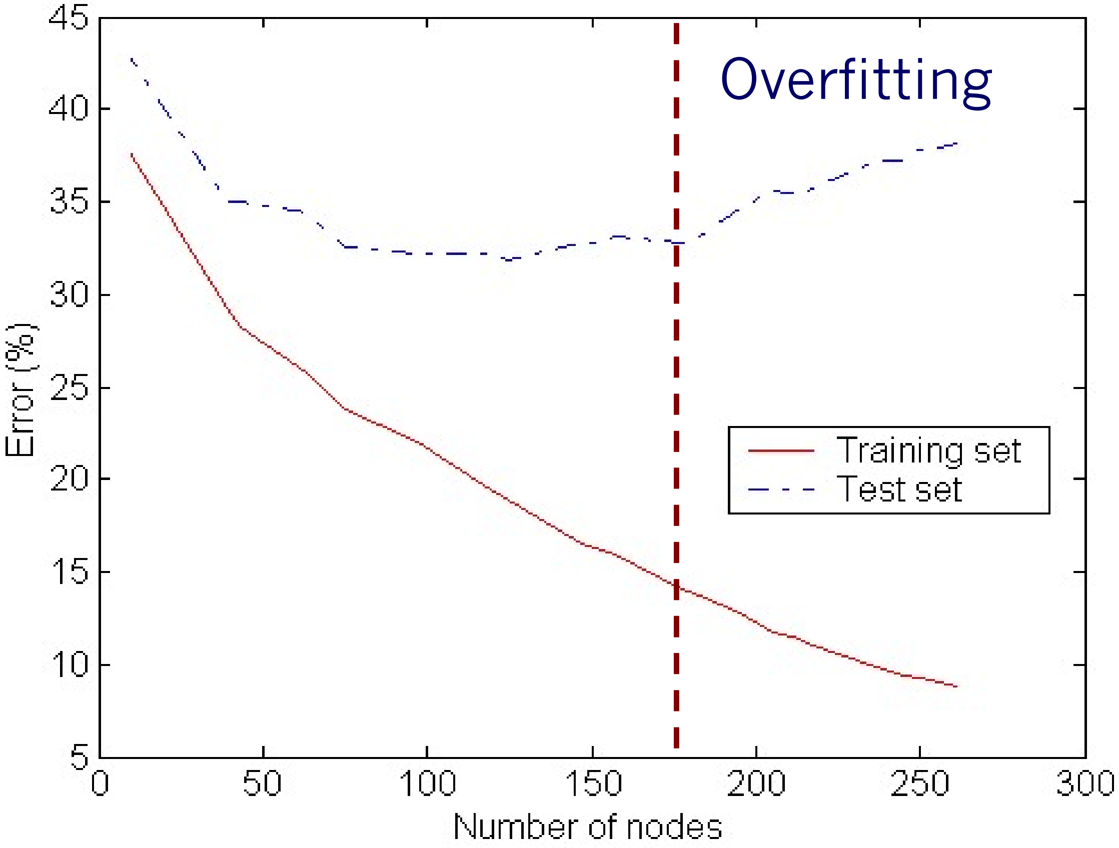
- Named after William of Ockham (1287-1347)
- A fundamental principle of science
  - if you have two theories
  - that explain a phenomenon equally well
  - choose the simpler one
- Tree that likely generalizes better according to Occam's razor



Overfitting: Symptoms and Causes

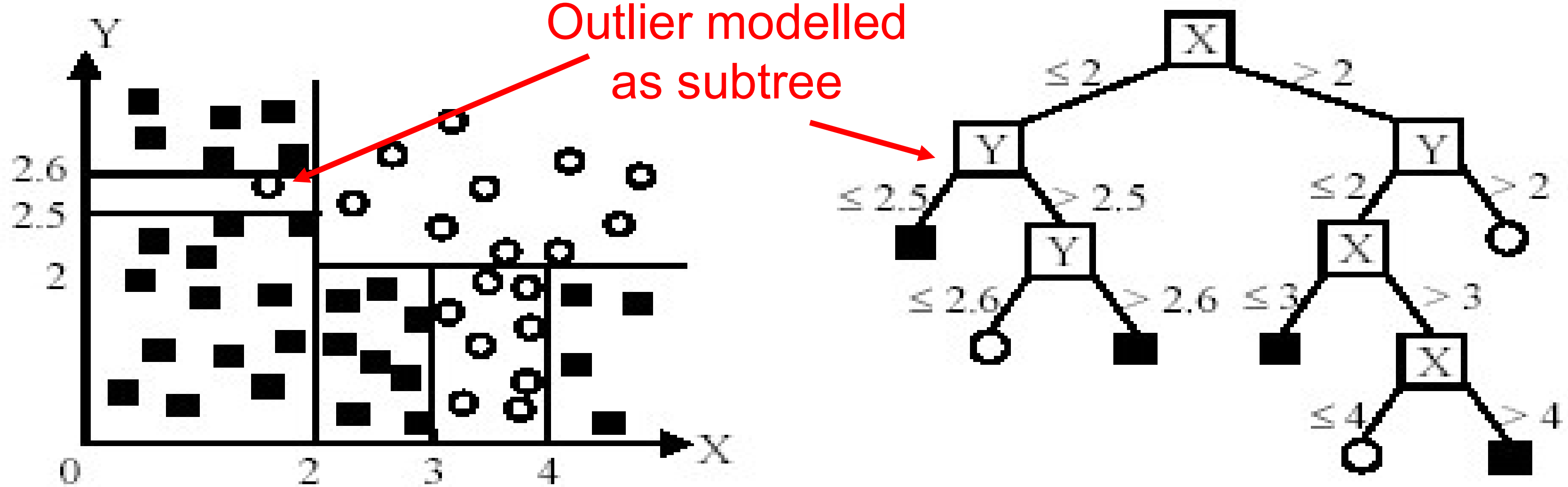
- Symptoms:
  1. decision tree too deep
  2. too many branches
  3. model works well on training set but performs bad on test set

- Typical causes of overfitting
  1. noise / outliers in training data
  2. too little training data
  3. high model complexity



An overfitted model does not generalize well to unseen data.

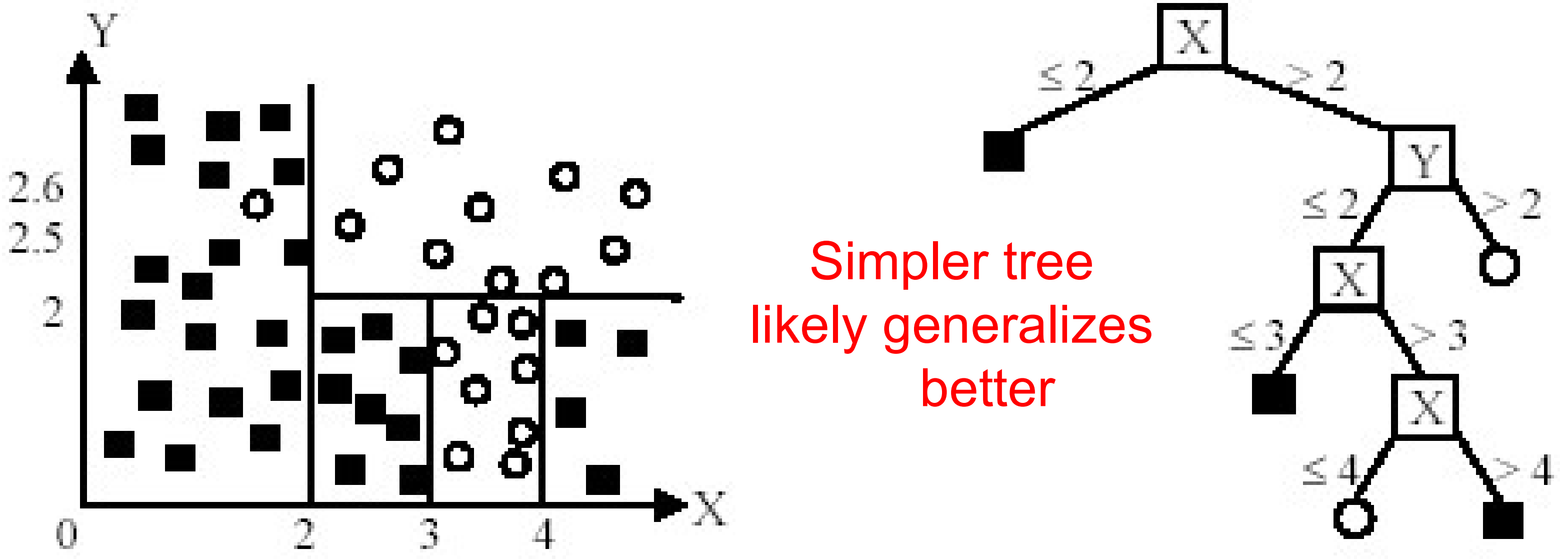
Example of an Outlier causing Overfitting



Outlier modelled as subtree

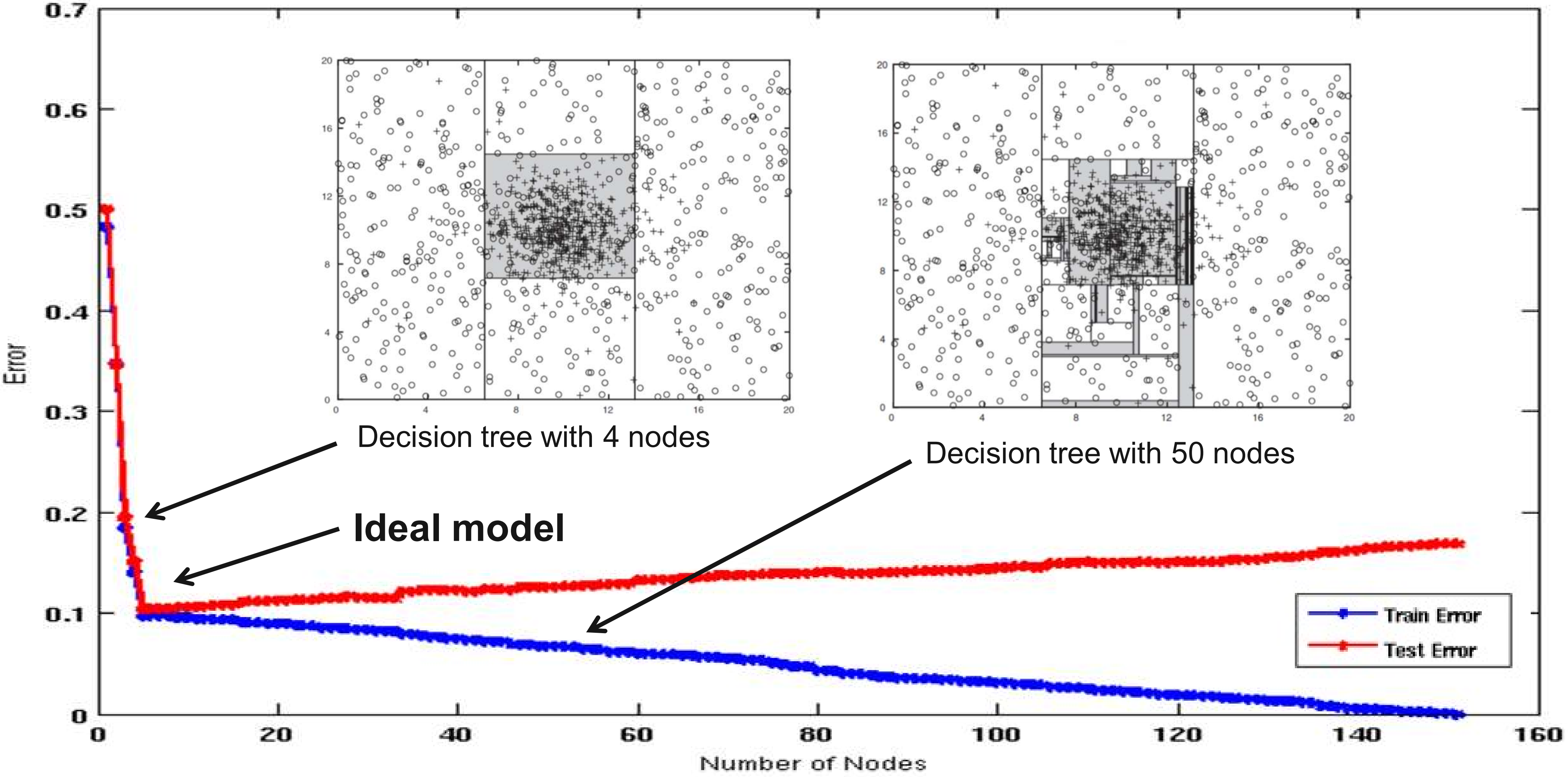
(A) A partition of the data space

(B). The decision tree



Simpler tree likely generalizes better

Underfitting versus Overfitting



Decision tree with 4 nodes

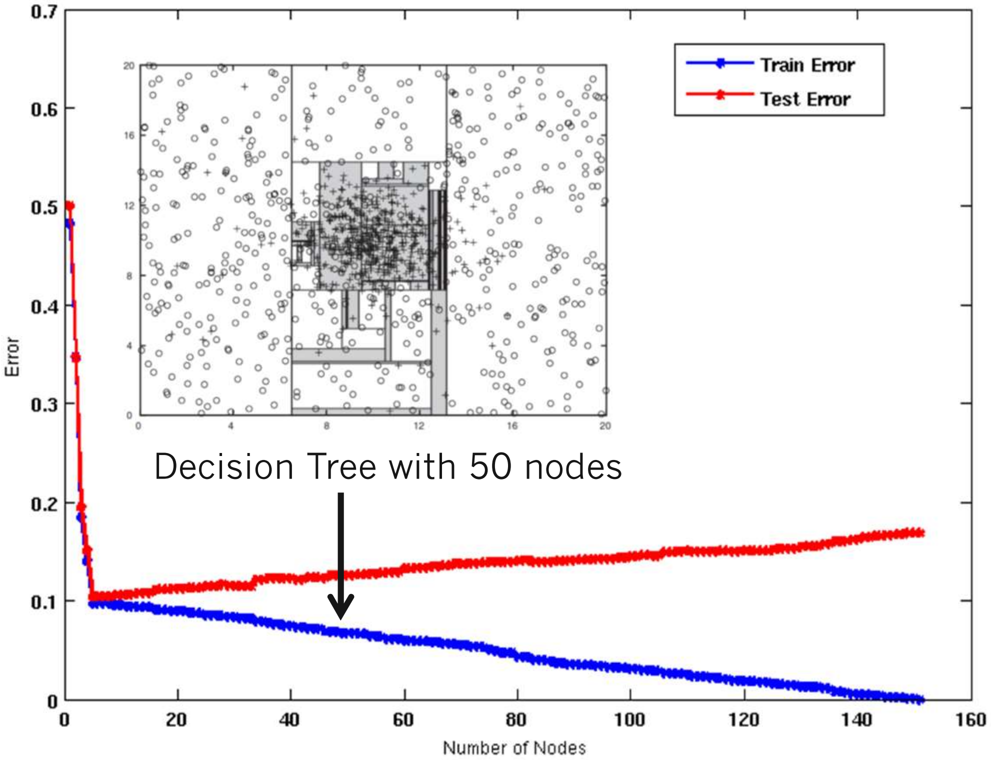
Decision tree with 50 nodes

Ideal model

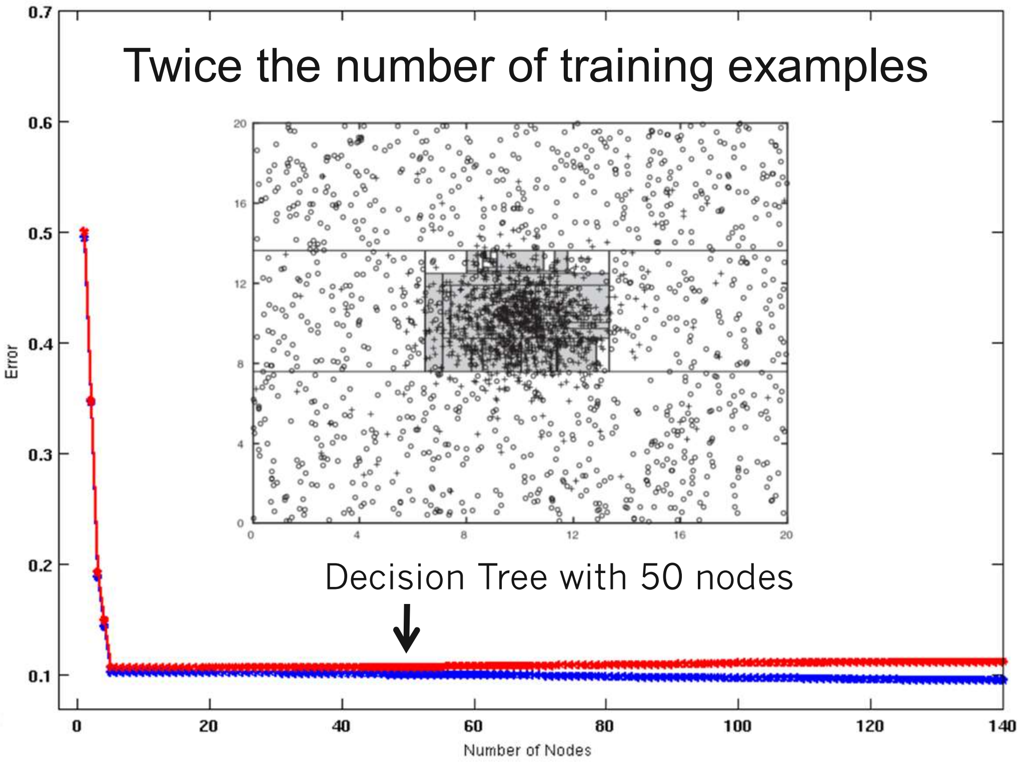
Underfitting: when model is too simple, both training and test errors are large

Overfitting: when model is too complex, training error is small but test error is large

How to Prevent Overfitting 1: Use More Training Data



Decision Tree with 50 nodes



Twice the number of training examples

Decision Tree with 50 nodes

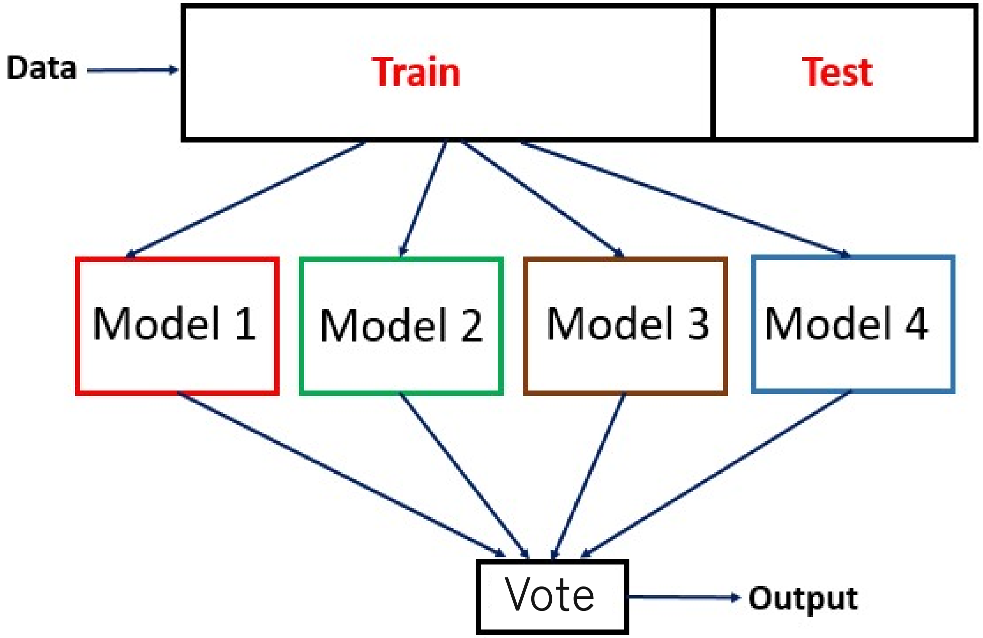
- If training data is under-representative, training errors decrease but testing errors increase on increasing number of nodes
- Increasing the size of training set reduces the difference between training and testing errors at a given number of nodes

How to Prevent Overfitting 2: Pre-Pruning

- Stop the algorithm before tree becomes fully-grown
  - shallower tree potentially generalizes better (Occam’s razor)
- Normal stopping conditions for a node (no pruning):
  - Stop if all instances belong to the same class
  - Stop if all the attribute values are the same
- Early stopping conditions (pre-pruning):
  - Stop if number of instances within a leaf node is less than some user-specified threshold (e.g. leaf size < 4)
  - Stop if expanding the current node only slightly improves the impurity measure (e.g. gain < 0.01)
  - Stop splitting at a specific depth (e.g. maxDepth = 5)

How to Prevent Overfitting 3: Ensembles

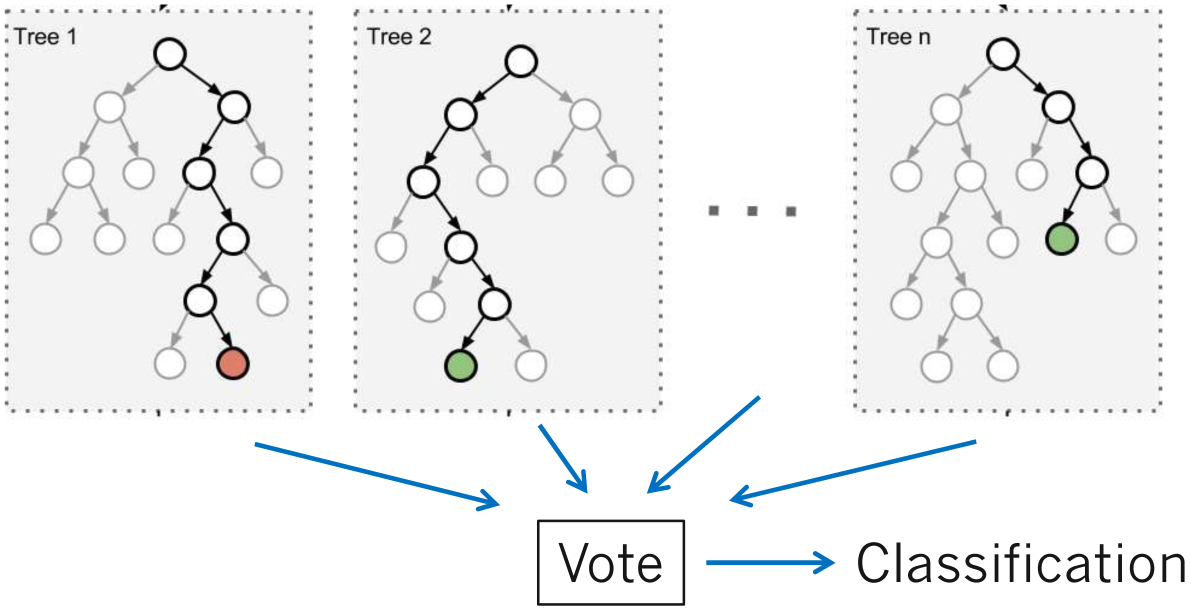
– Lean multiple different models (base learners)
– Have them vote on the final classification decision



– Idea: Wisdom of the crowds applied to classification
  - A single classifier might focus too much on one aspect
  - Multiple classifiers can focus on different aspects

Random Forest

– Ensemble consisting of a large number of different decision trees



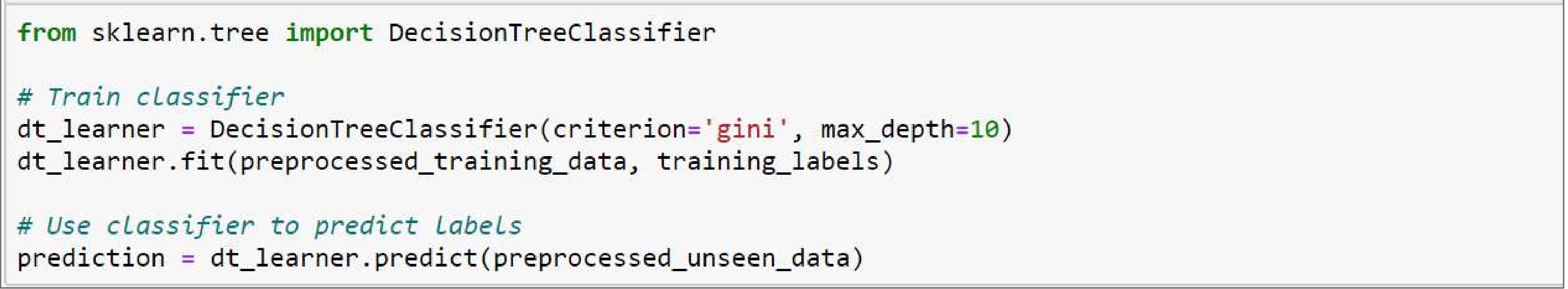
Vote → Classification

– Independence of trees achieved by introducing randomness into the learning process
  - only use a random subset of the attributes at each split
  - learn on different random subsets of the data (bagging)

– Random forests usually outperform single decision trees

Decision Tree Classification in RapidMiner and Python

Python

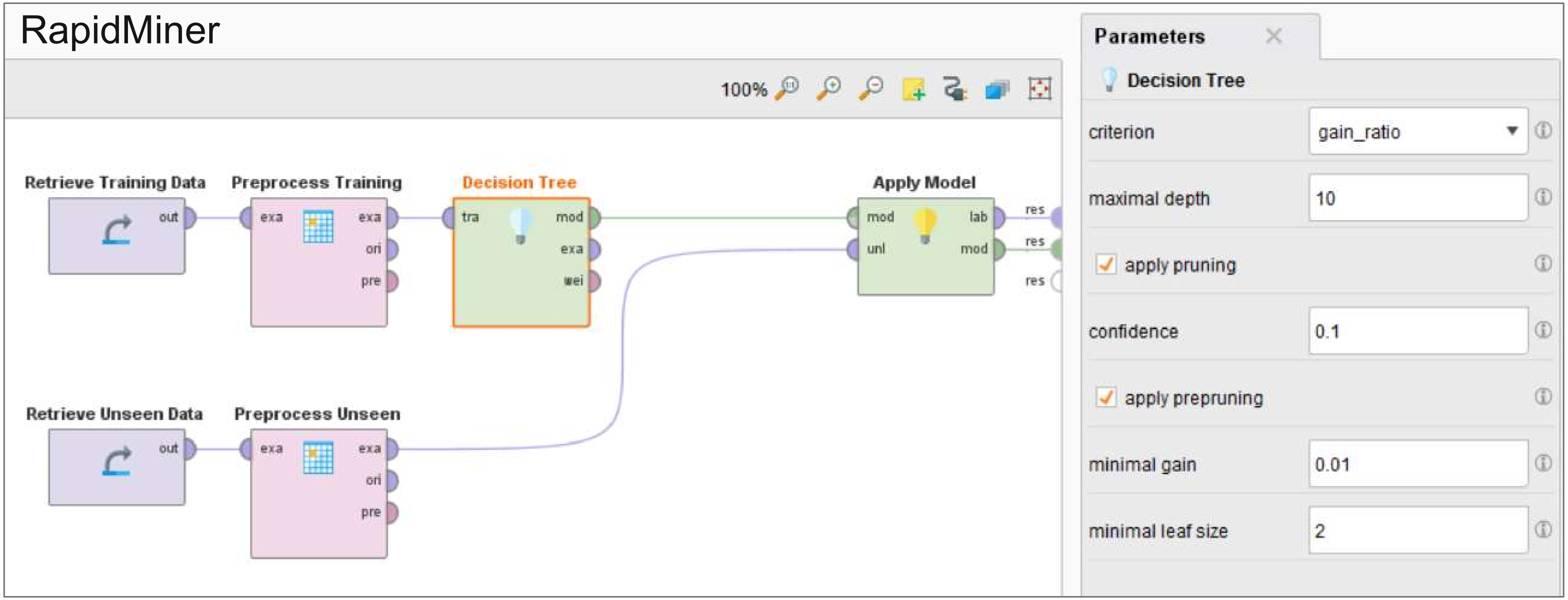


```
from sklearn.tree import DecisionTreeClassifier

# Train classifier
dt_learner = DecisionTreeClassifier(criterion='gini', max_depth=10)
dt_learner.fit(preprocessed_training_data, training_labels)

# Use classifier to predict labels
prediction = dt_learner.predict(preprocessed_unseen_data)
```

RapidMiner



Parameters

Decision Tree

criterion: gain_ratio

maximal depth: 10

☑ apply pruning

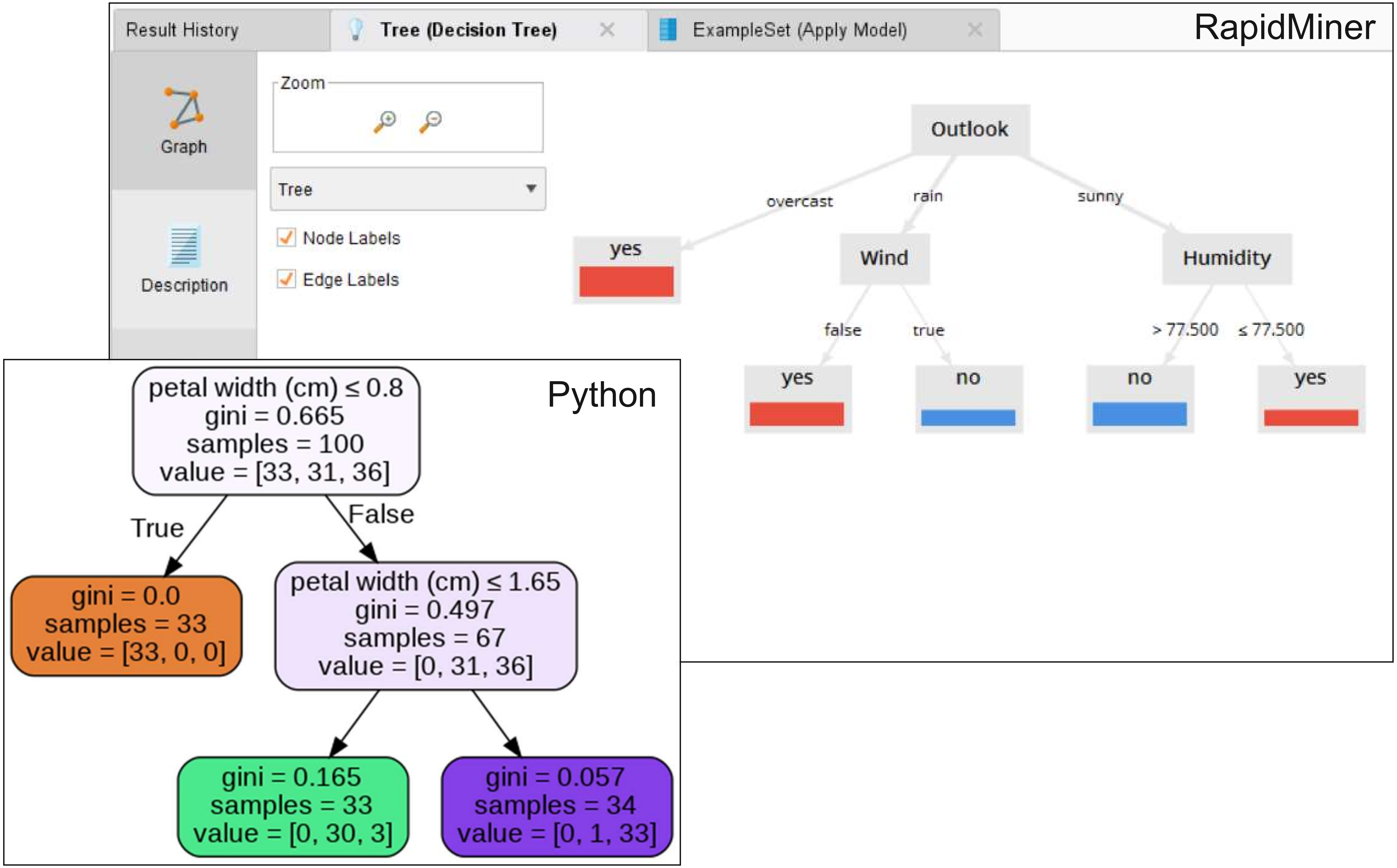
confidence: 0.1

☑ apply prepruning

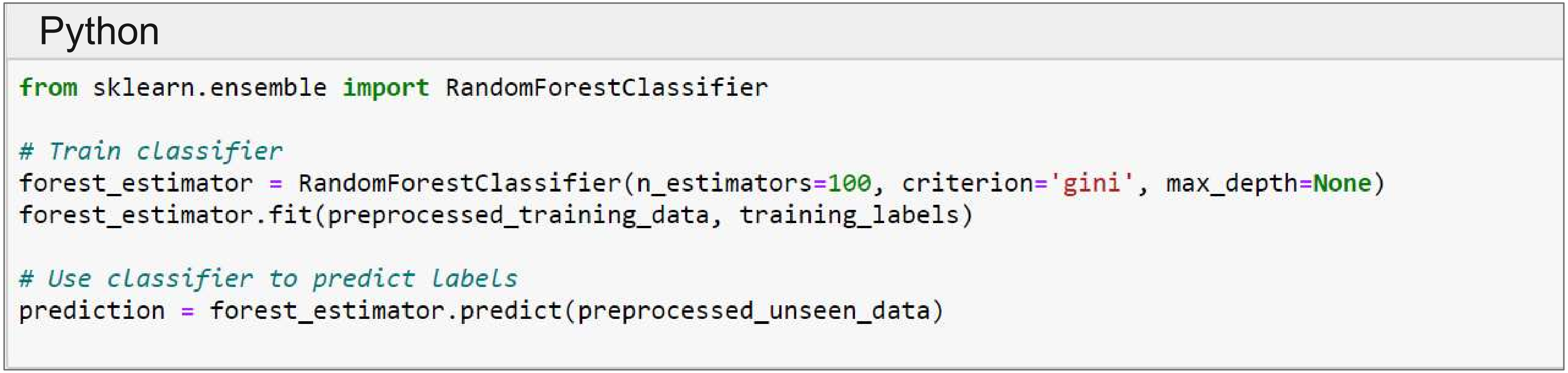
minimal gain: 0.01

minimal leaf size: 2

Examples of Learned Decision Trees



Random Forests in RapidMiner and Python

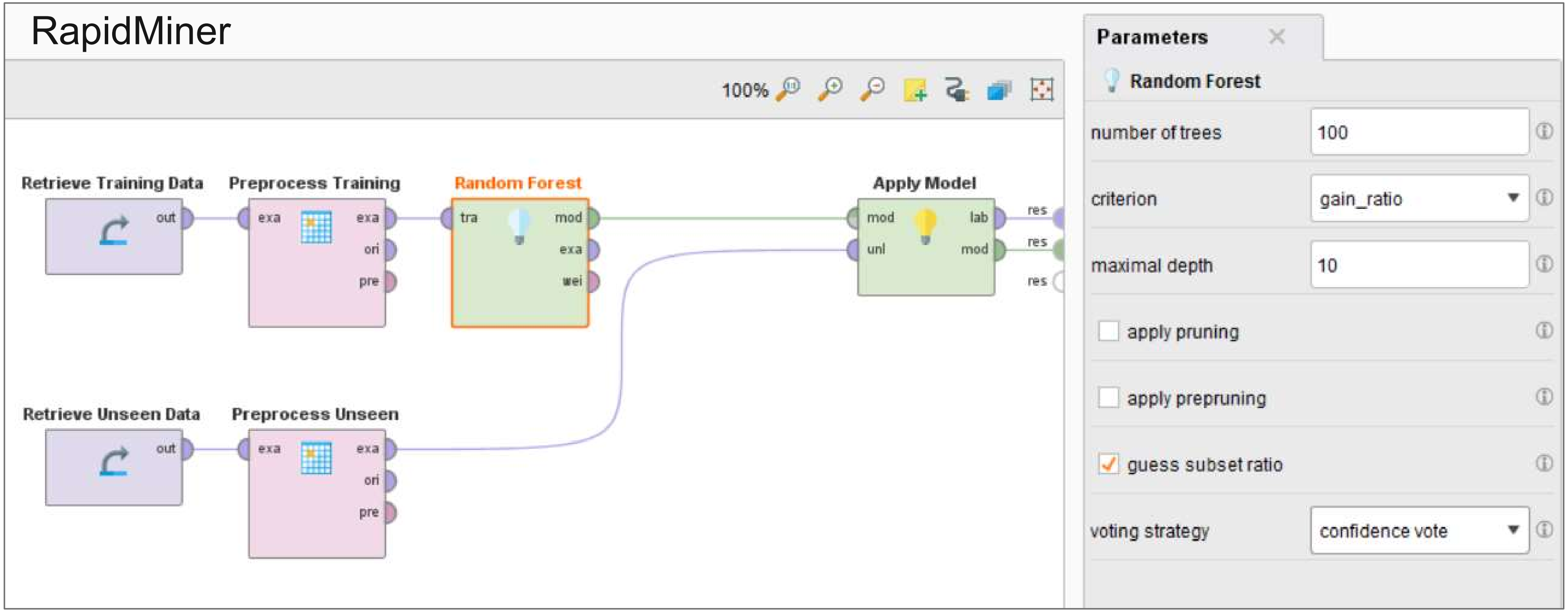


Python

```
from sklearn.ensemble import RandomForestClassifier

# Train classifier
forest_estimator = RandomForestClassifier(n_estimators=100, criterion='gini', max_depth=None)
forest_estimator.fit(preprocessed_training_data, training_labels)

# Use classifier to predict labels
prediction = forest_estimator.predict(preprocessed_unseen_data)
```



RapidMiner

Parameters

Random Forest

number of trees: 100

criterion: gain_ratio ▼

maximal depth: 10

□ apply pruning

□ apply prepruning

☑ guess subset ratio

voting strategy: confidence vote ▼

3.4 Discussion of Decision Trees

– Advantages

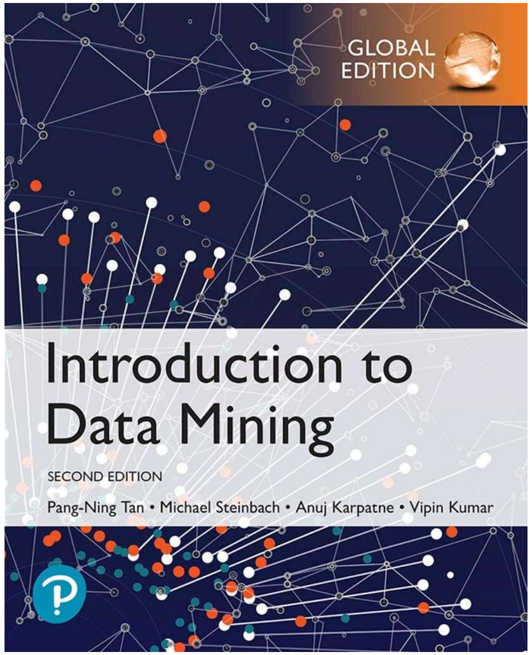
- Inexpensive to construct
- Extremely fast at classifying unknown records
- Easy to interpret by humans for small-sized trees (eager learning)
- Ignore irrelevant attributes (automatic feature selection)
- Accuracy is comparable to other classification techniques for many low dimensional data sets (not texts and images)

– Disadvantages

- Space of possible decision trees is exponentially large. Greedy approaches are often unable to find the best tree
- Trees do not consider interactions between attributes

Literature for this Slideset

Pang-Ning Tan, Michael Steinbach, Anuj Karpatne, Vipin Kumar: **Introduction to Data Mining.** 2nd Edition. Pearson.


GLOBAL EDITION
Introduction to Data Mining
SECOND EDITION
Pang-Ning Tan • Michael Steinbach • Anuj Karpatne • Vipin Kumar

Chapter 3: Classification

Chapter 6.3: Nearest Neighbor Classifiers

Chapter 3.3: Decision Tree Classifier

Chapter 3.4: Overfitting

Chapter 6.10.6: Random Forests