UNIVERSITÄT MANNHEIM

Data Mining I
Classification, Part 2

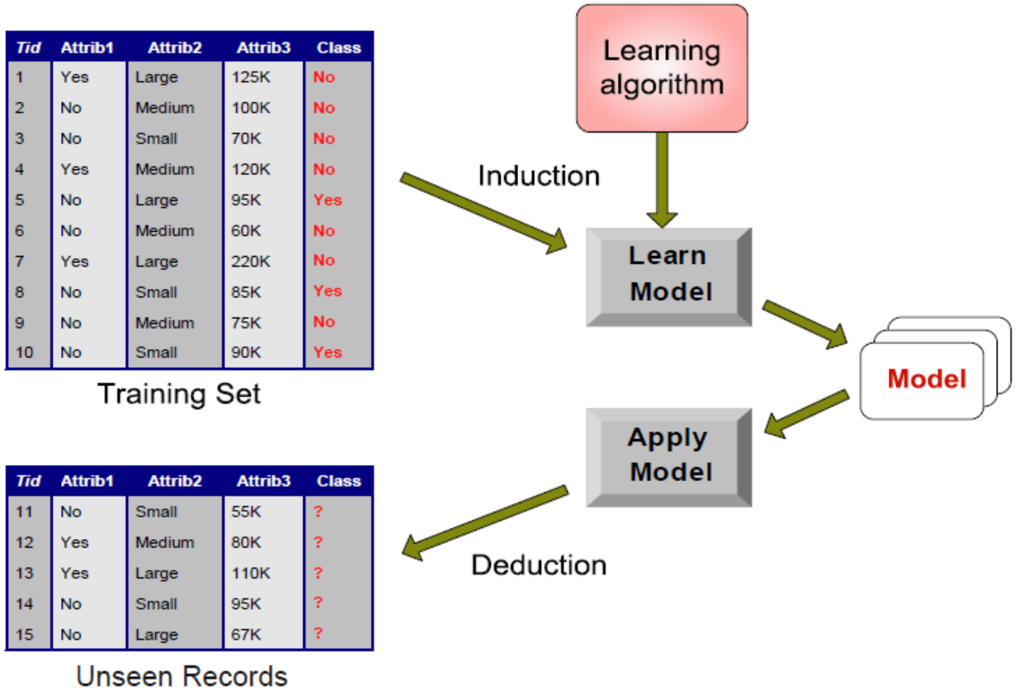
Heiko Paulheim

Outline

1. What is Classification? ✔
2. k Nearest Neighbors ✔
3. Naïve Bayes ✔
4. Decision Trees
5. Evaluating Classification
6. The Overfitting Problem
7. Rule Learning
8. Other Classification Approaches
9. Parameter Tuning

Lazy vs. Eager Learning

- Both k-NN and Naïve Bayes are “lazy” methods
- They do not build an explicit model!
  - “learning” is only performed on demand for unseen records



Today: Eager Learning

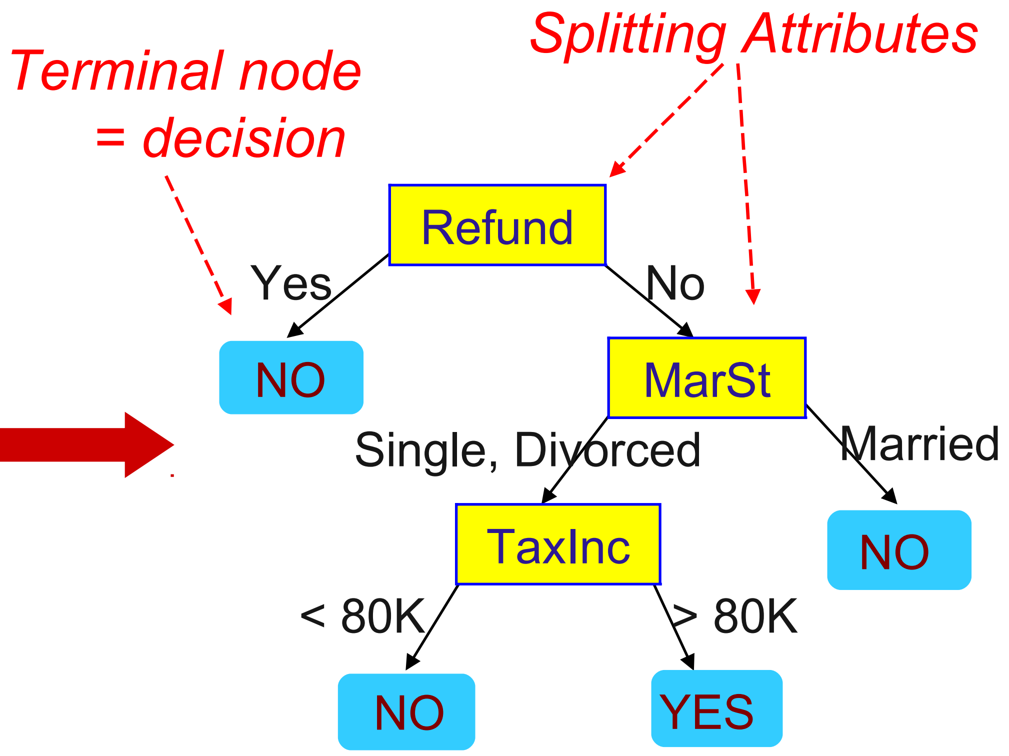
- Actually, we have two goals
  - classify unseen instances
  - learn a model

- Model
  - explains how to classify unseen instances
  - sometimes: interpretable by humans

Decision Tree Classifiers

Tid	Refund (categorical)	Marital Status (categorical)	Taxable Income (continuous)	Cheat (class)
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

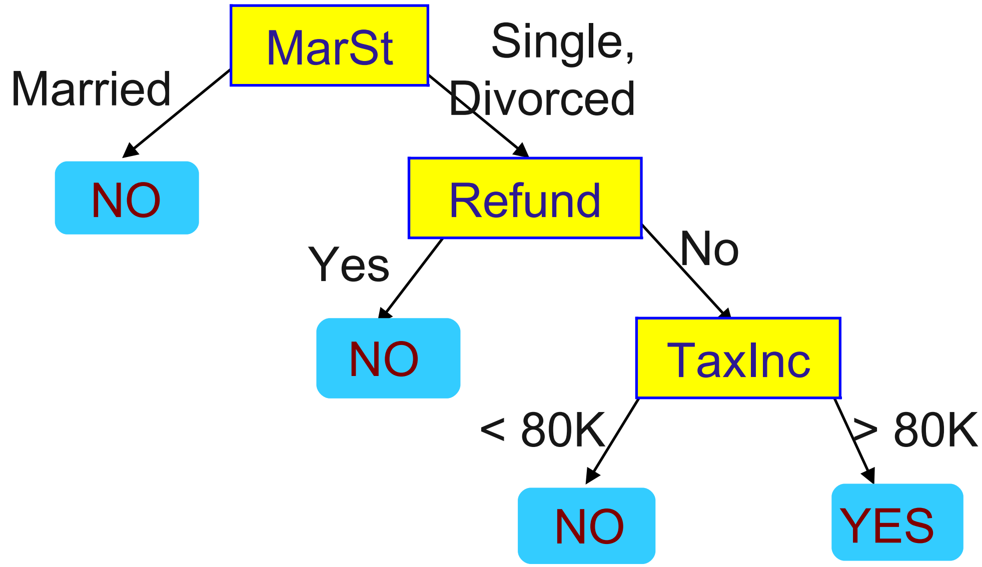


Model: Decision Tree

Another Example of a Possible Decision Tree

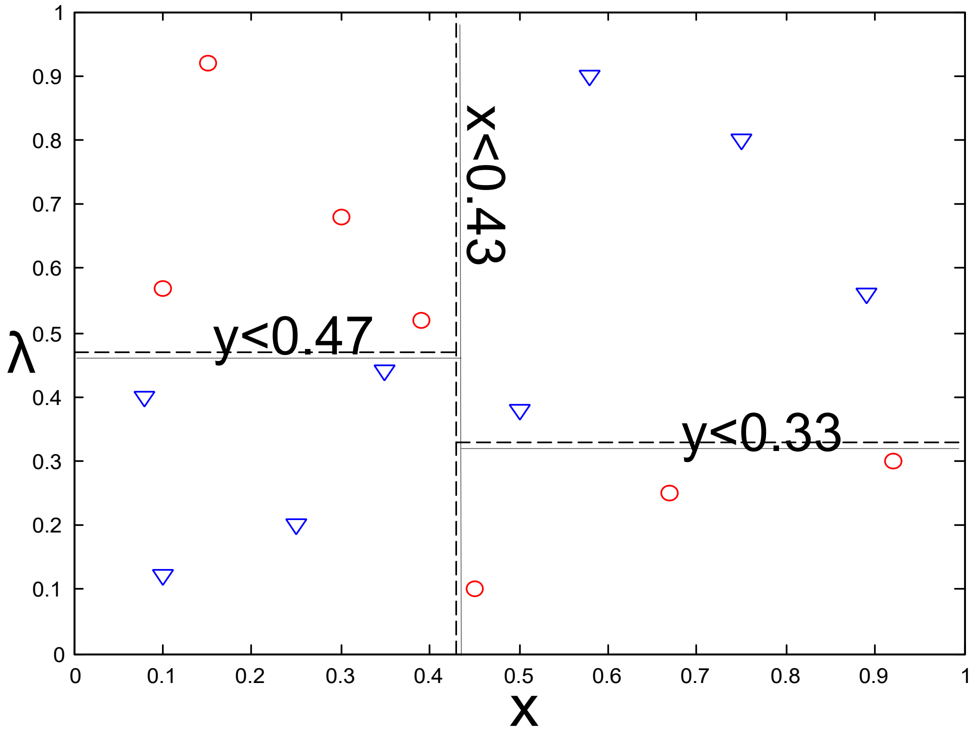
categorical categorical continuous class

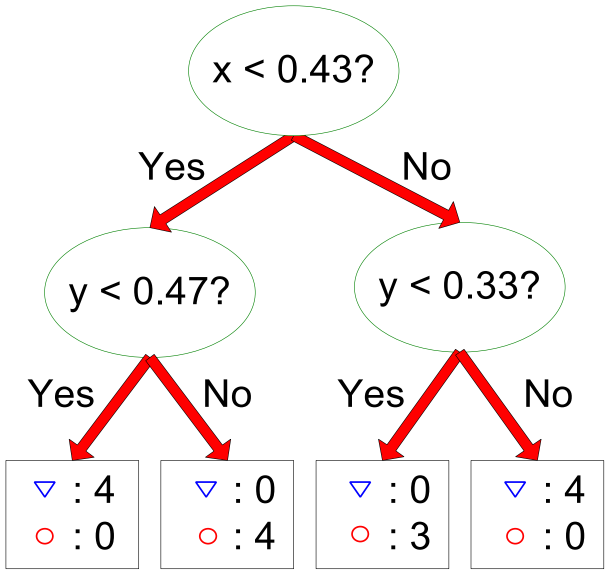
Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes



There can be more than one tree that fits the same data!

Decision Boundary

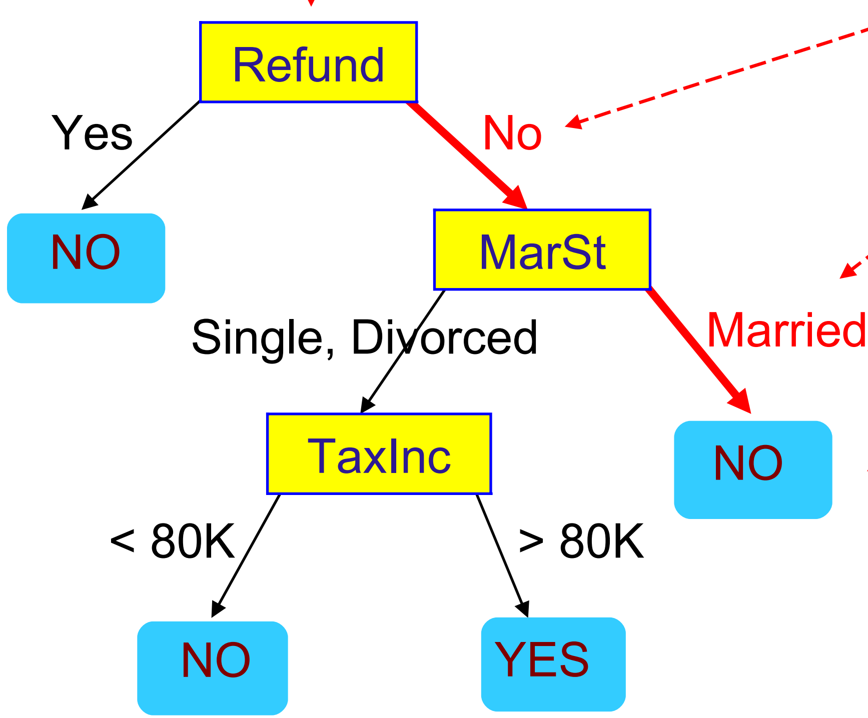




- Border line between two neighboring regions of different classes is known as decision boundary
- Decision boundary is parallel to axes because test condition involves a single attribute at-a-time

Applying a Decision Tree to Test Data

Start from the root of tree.



Test Data

Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?

Assign Cheat to “No”

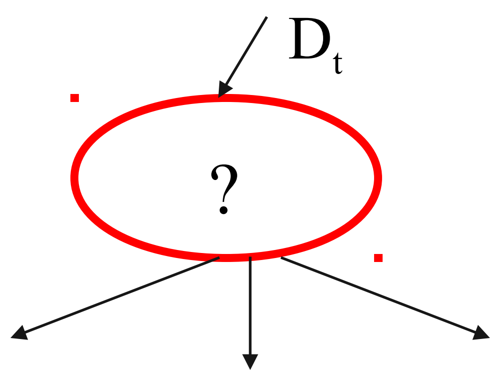
Decision Tree Induction

- How to learn a decision Tree from test data?
- Finding an optimal decision tree is NP-hard
- Tree building algorithms use a greedy, top-down, recursive partitioning strategy to induce a reasonable solution
  - also known as: divide and conquer
- Many different algorithms have been proposed:
  - Hunt’s Algorithm
  - ID3
  - CHAID
  - C4.5

General Structure of Hunt's Algorithm

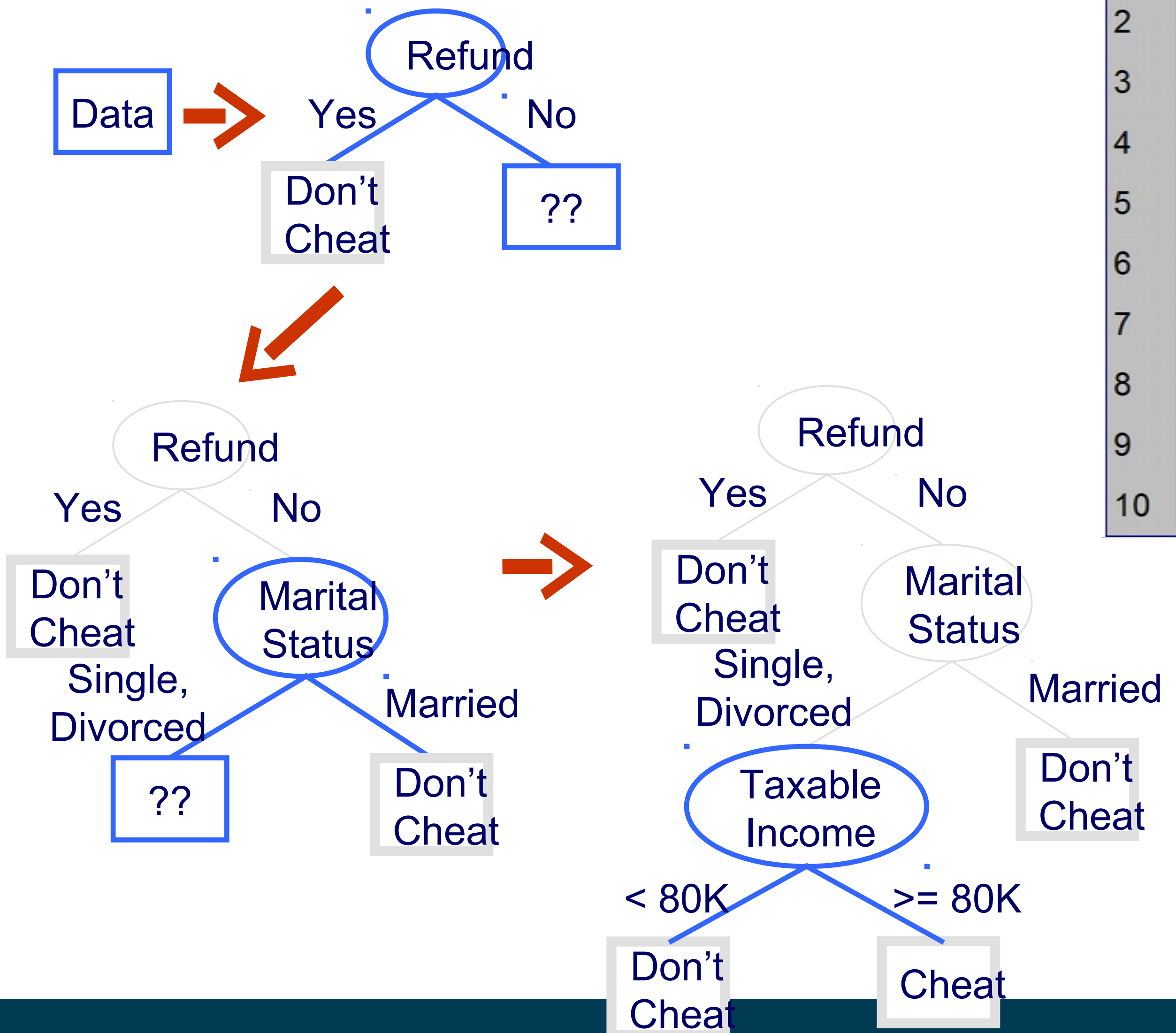
- Let D_t be the set of training records that reach a node t
- General Procedure:
  - If D_t contains only records that belong to the same class y_t, then t is a leaf node labeled as y_t
  - If D_t contains records that belong to more than one class, use an attribute test to split the data into smaller subsets
  - Recursively apply the procedure to each subset

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

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



Tree Induction Issues

- Determine how to split the records
  - How to specify the attribute test condition?
  - How to determine the best split?
- Determine when to stop splitting

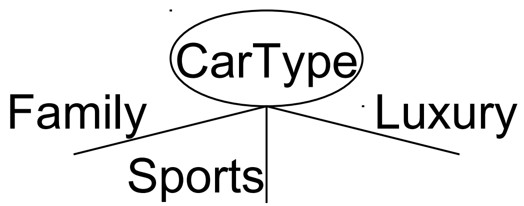
How to Specify the Attribute Test Condition?

- Depends on attribute types
  - Nominal
  - Ordinal
  - Continuous

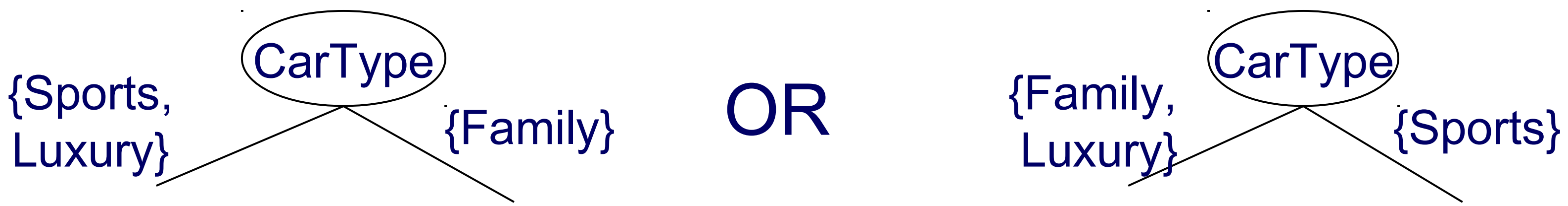
- Depends on number of ways to split
  - 2-way split
  - Multi-way split

Splitting Based on Nominal Attributes

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

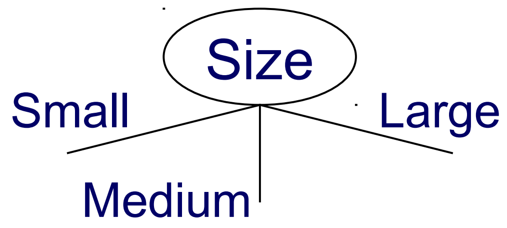


- Binary split: Divides values into two subsets. Need to find optimal partitioning

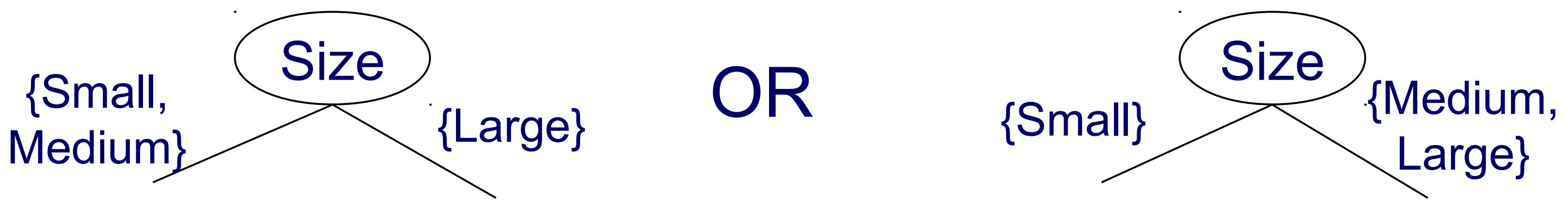


Splitting Based on Ordinal Attributes

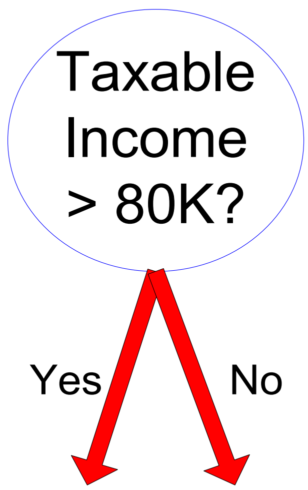
- Multi-way split: Use as many partitions as distinct values.



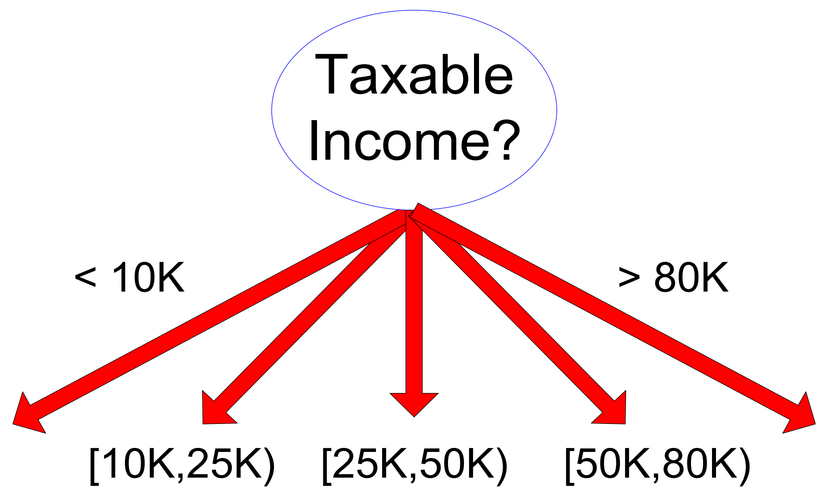
- Binary split: Divides values into two subsets, while keeping the order. Need to find optimal partitioning.



Splitting Based on Continuous Attributes



(i) Binary split



(ii) Multi-way split

Splitting Based on Continuous Attributes

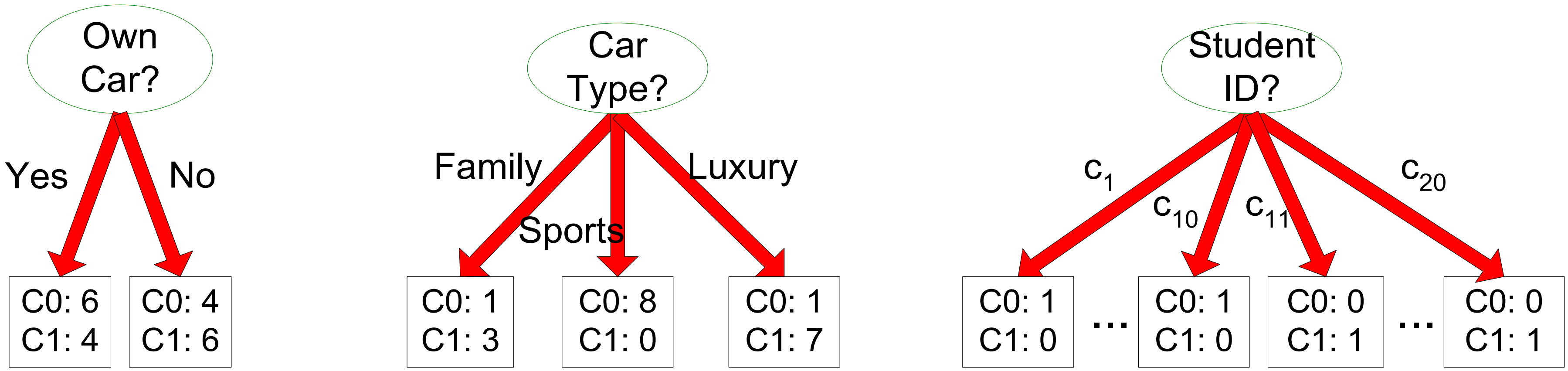
- Different ways of handling
  - Discretization to form an ordinal categorical attribute
    - equal-interval binning
    - equal-frequency binning
    - binning based on user-provided boundaries
  - Binary Decision: $(A < v)$ or $(A \geq v)$
    - usually sufficient in practice
    - consider all possible splits
    - find the best cut (i.e., the best v) based on a purity measure (see later)
    - can be computationally expensive

Discretization Example

- Attribute values (for one attribute e.g., age):
  - 0, 4, 12, 16, 16, 18, 24, 26, 28
- Equal-width 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
    - ∞ denotes 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

How to determine the Best Split?

Before Splitting: 10 records of class 0,
10 records of class 1



Which test condition is the best?

How to determine the Best Split?

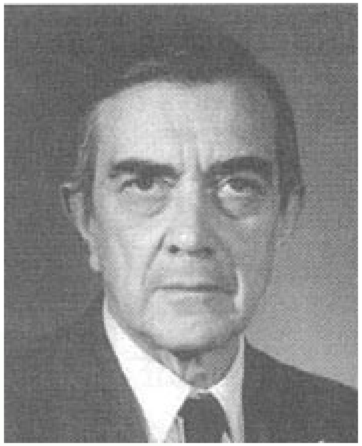
- Nodes with homogeneous class distribution are preferred
- Need a measure of node impurity:

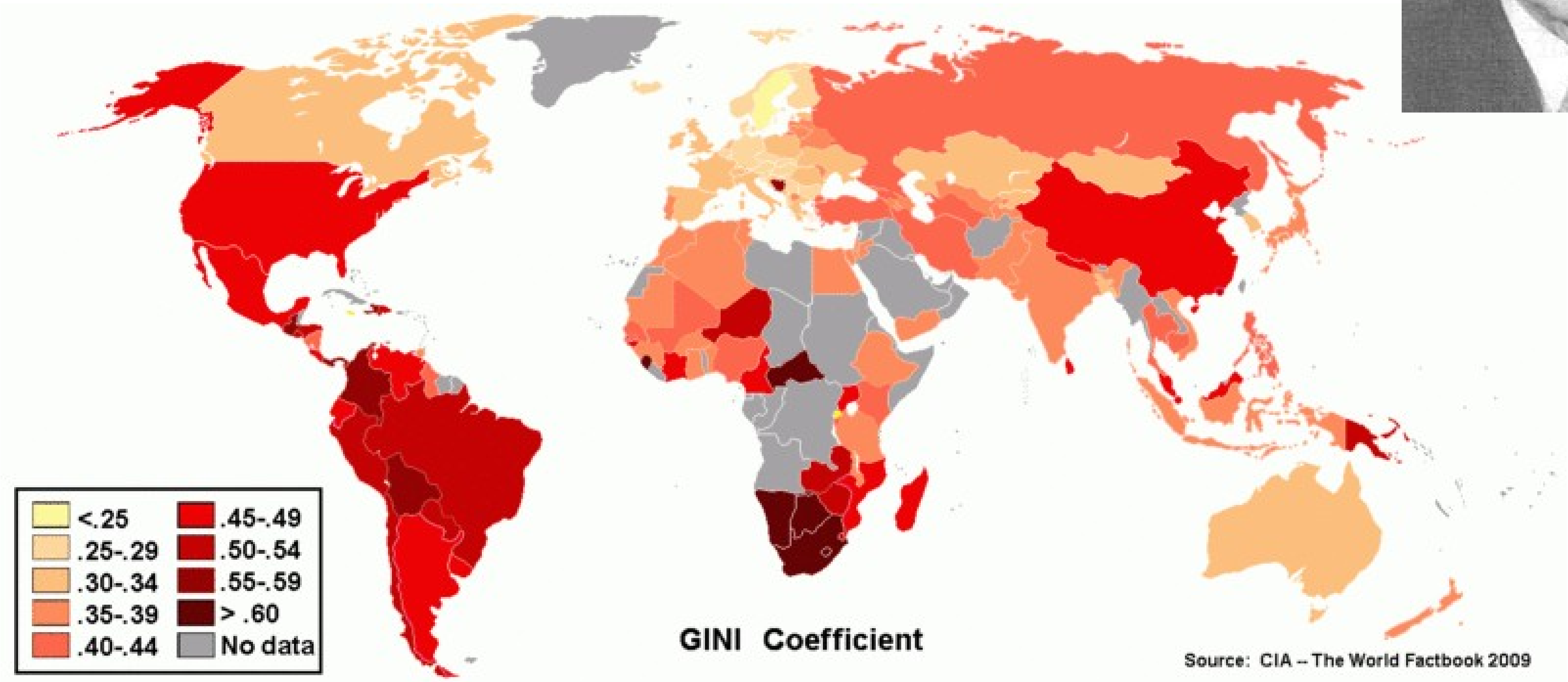
C0: 5 C1: 5	C0: 9 C1: 1
Non-homogeneous, High degree of impurity	Homogeneous, Low degree of impurity

- Common measures of node impurity:
  - Gini Index
  - Entropy
  - Misclassification error

Gini Index

- Named after Corrado Gini (1885-1965)
- Used to measure the distribution of income
  - 1: somebody gets everything
  - 0: everybody gets an equal share





Measure of Impurity: GINI

- Gini-based purity measure for a given node t :

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

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

– Maximum ($1 - 1/n_c$) when records are equally distributed among all classes, implying least interesting information

– Minimum (0.0) when all records belong to one class, implying most interesting information

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

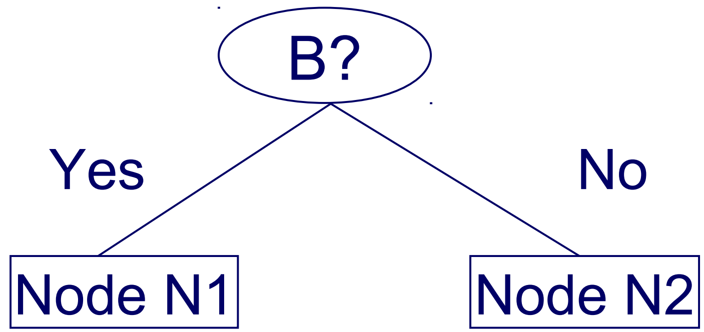
- When a node p is split into k partitions (children), the quality of split is computed as

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

  - where n_i = number of records at child i,
  - n = number of records at node p.
- Intuition:
  - The GINI index of each partition is weighted
  - according to the partition's size

Binary Attributes: Computing GINI Index

- Splits into two partitions



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

Gini(N1)
$= 1 – (5/7)^2 – (2/7)^2$
$= 0.408$

Gini(N2)
$= 1 – (1/5)^2 – (4/5)^2$
$= 0.320$

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

Gini(Children)
$= 7/12 * 0.408 +$
$5/12 * 0.320$
$= 0.371$

Categorical Attributes: Computing Gini Index

- For each distinct value, gather counts for each class in the dataset
- Use the count matrix to make decisions

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)

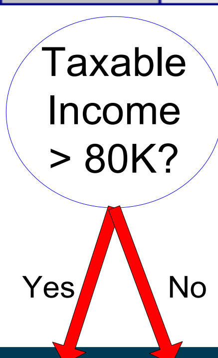
	CarType	
	{Sports, Luxury}	{Family}
C1	3	1
C2	2	4
Gini	0.400	

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

Continuous Attributes: Computing Gini Index

- Use Binary Decisions based on one value
- Several Choices for the splitting value
  - Number of possible splitting values = Number of distinct values
- Each splitting value has a count matrix associated with it
  - Class counts in each of the partitions, A < v and A ≥ v
- Simple method to choose best v
  - For each v, scan the database to gather count matrix and compute its Gini index
  - Computationally Inefficient! Repetition of work

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



Continuous Attributes: Computing Gini Index

- For efficient computation: for each attribute,
  - Sort the attribute on values
  - Linearly scan these values, each time updating the count matrix and computing gini index
  - Choose the split position that has the least gini index

Sorted Values →

Cheat	No	No	No	Yes	Yes	Yes	No	No	No	No
Taxable Income										
	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	

Continuous Attributes: Computing Gini Index

- Note: it is enough to compute the GINI for those positions where the label changes!

Cheat	No	No	No	Yes	Yes	Yes	No	No	No	No
	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	

Alternative Splitting Criteria: Information Gain

- Entropy at a given node t:

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

(NOTE: p(j | t) is the relative frequency of class j at node t).

  – Measures homogeneity of a node
    - Maximum (log nc) when records are equally distributed among all classes implying least information
    - Minimum (0.0) when all records belong to one class, implying most information
  – Entropy based computations are similar to the GINI index computations

Splitting Based on Information Gain

- Information Gain:

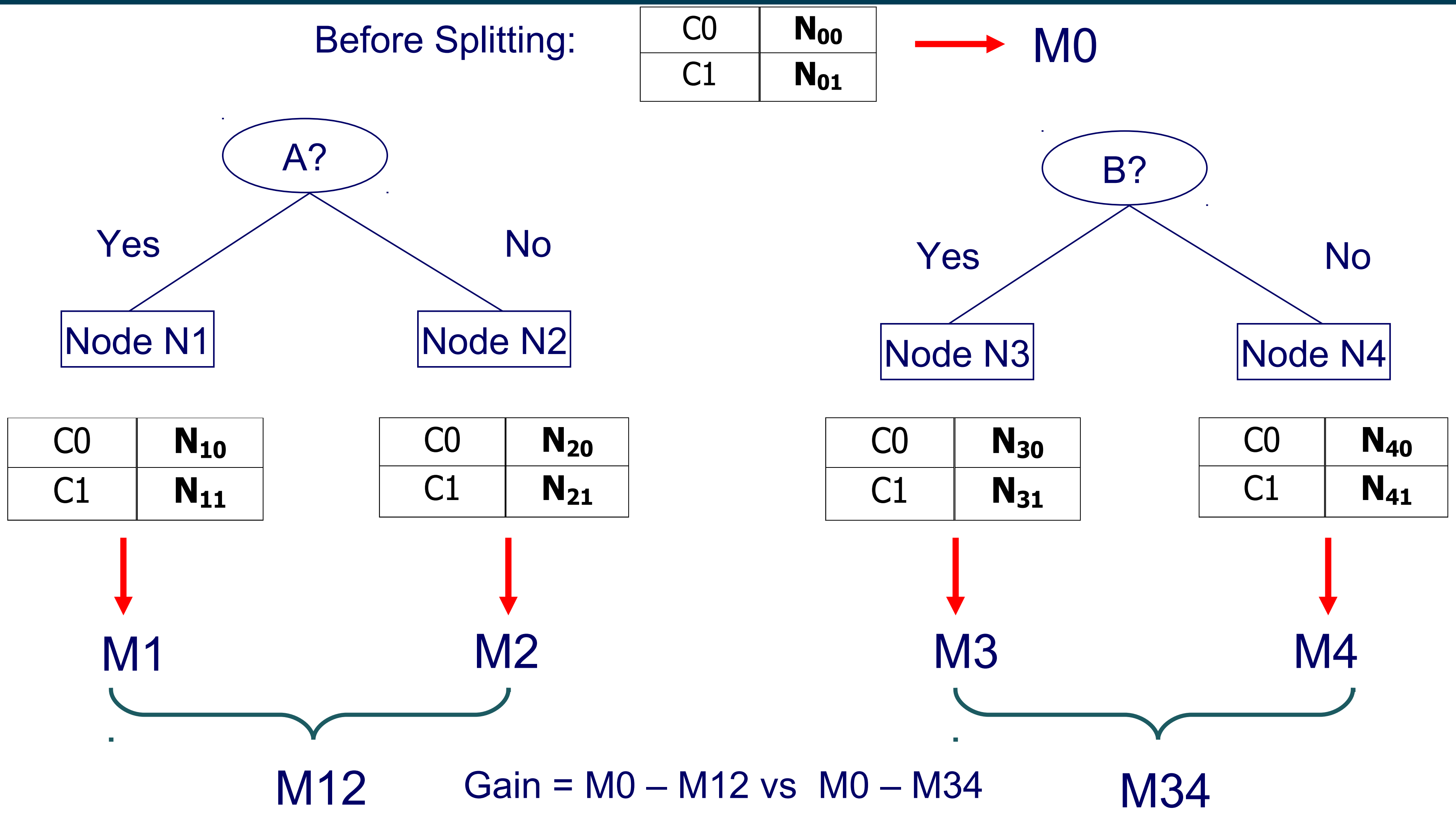
$$GAIN_{split} = Entropy(p) - \left(\sum_{i=1}^{k} \frac{n_i}{n} Entropy(i) \right)$$

  - Parent Node, p is split into k partitions;
  - n_i is number of records in partition i

– Measures reduction in entropy achieved because of the split
  - Choose the split that achieves most reduction (maximizes GAIN)

– Disadvantage: Tends to prefer splits that result in large number of partitions, each being small but pure
  - e.g,. split by ID attribute

How to Find the Best Split



Alternative Splitting Criteria: GainRATIO

- Gain Ratio:

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

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

  - Parent Node, p is split into k partitions
  - n_i is the number of records in partition I

– Adjusts Information Gain by the entropy of the partitioning (SplitINFO)

  - Higher entropy partitioning (large number of small partitions) is penalized!

– Designed to overcome the tendency to generate a large number of small partitions

Alternative Splitting Criteria: Classification Error

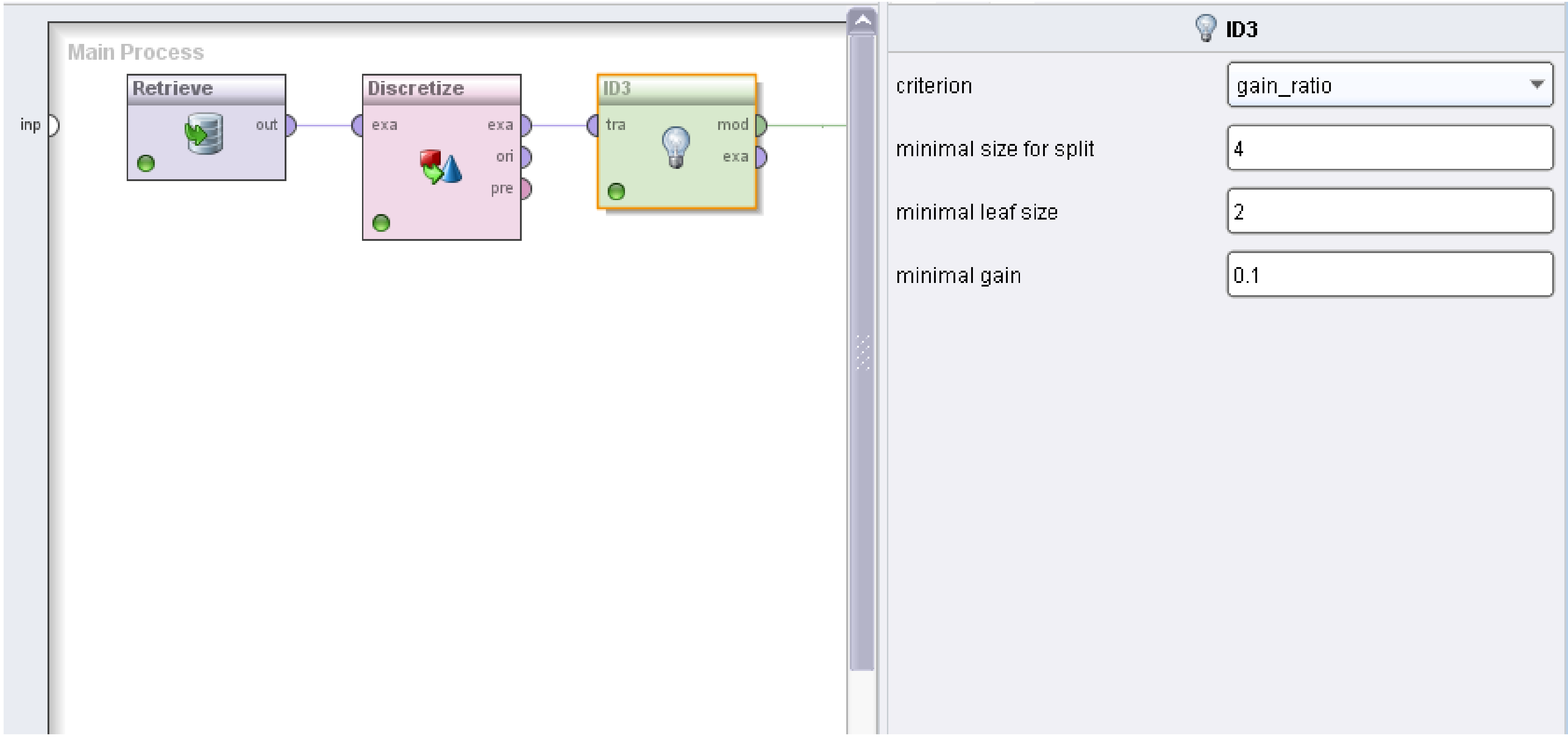
- Classification error at a node t :

$$Error(t) = 1 - \max_i P(i \mid t)$$

- Measures misclassification error made by a node.
  - Assumption: The node classifies every example to belong to the majority class
  - Maximum ($1 - 1/n_c$) when records are equally distributed among all classes, implying least interesting information
  - Minimum (0.0) when all records belong to one class, implying most interesting information

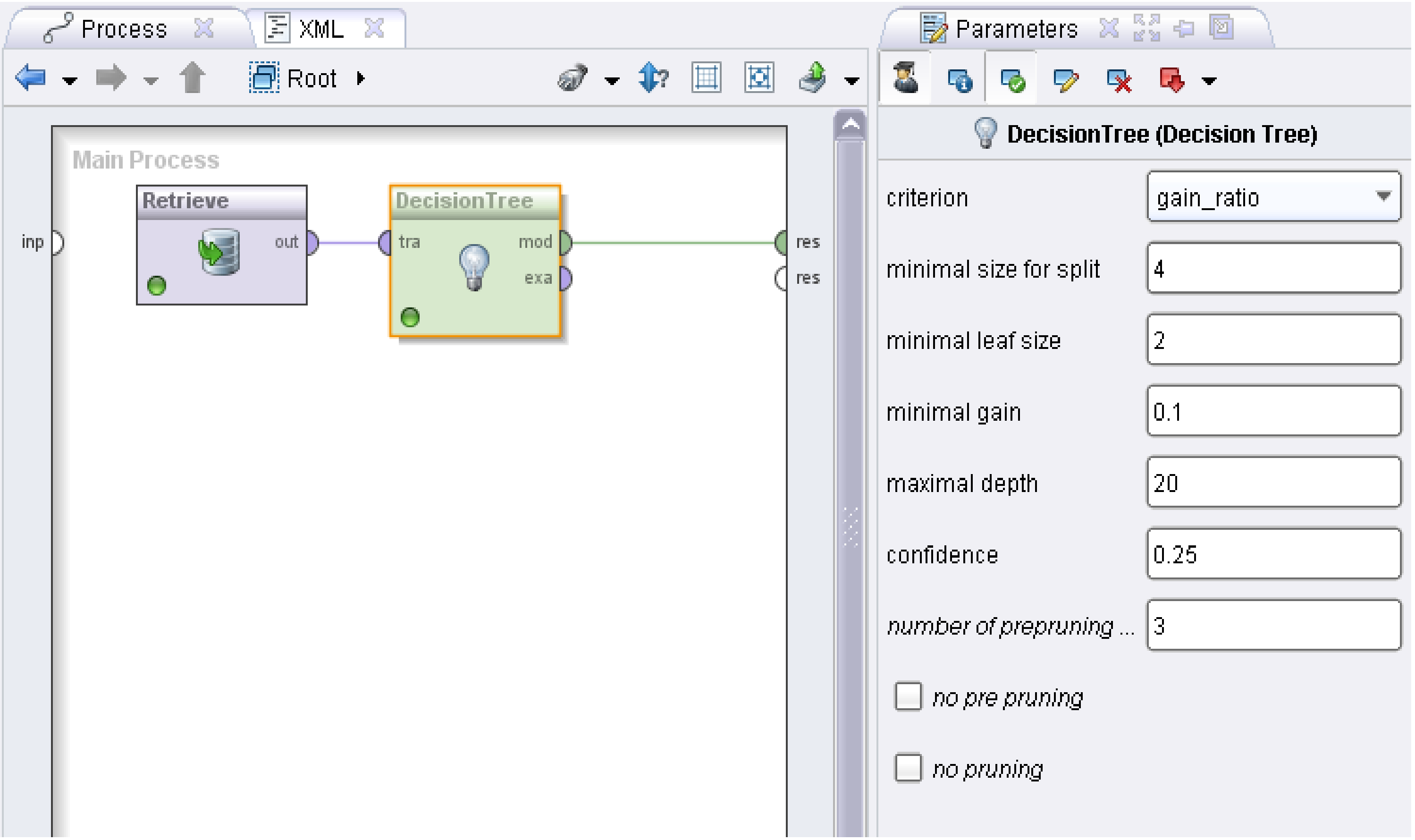
Decision Trees in RapidMiner (ID3)

Learns an un-pruned decision tree from nominal attributes only.



Decision Trees in RapidMiner

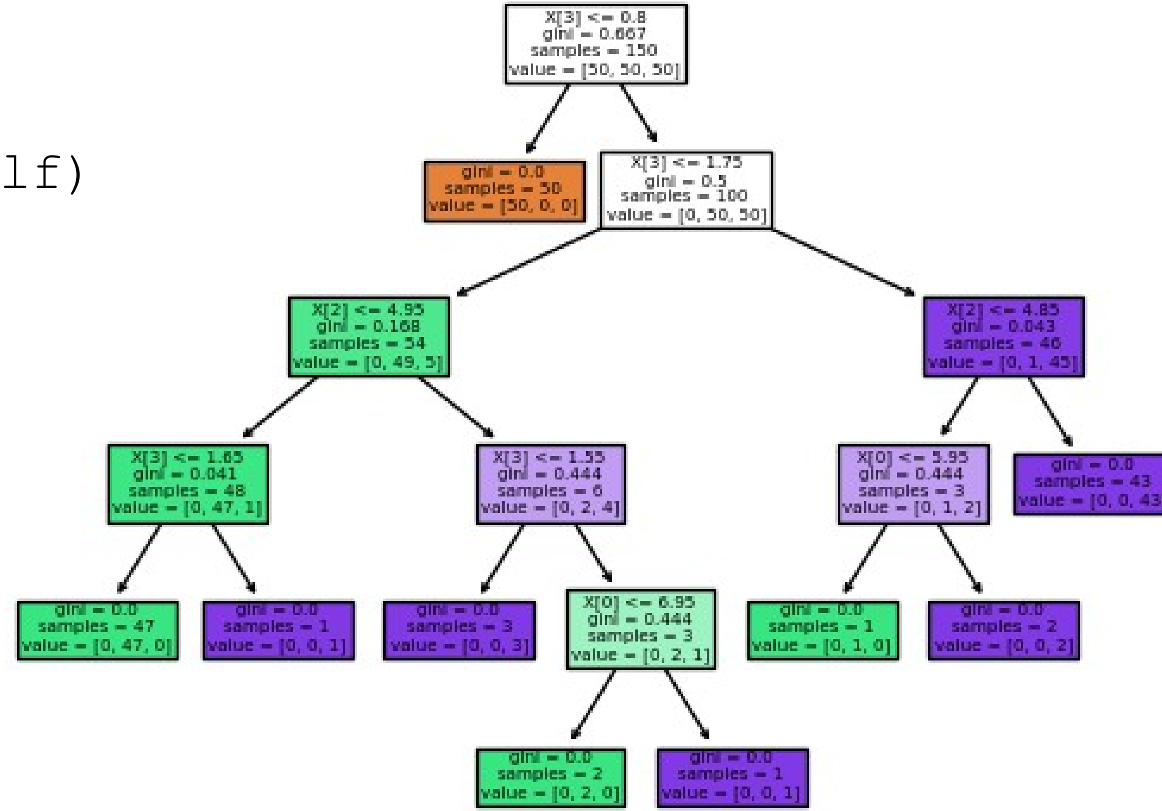
More flexible algorithm that includes pruning and discretization



Tree Induction in Python

```
clf = DecisionTreeClassifier()
clf = clf.fit(X,X_labels)

# Visualization
tree.plot_tree(clf)
```



Model Evaluation

- Metrics
  - how to measure performance?
- Evaluation methods
  - how to obtain meaningful and reliable estimates?

Model Evaluation

- Models are evaluated by looking at
  - correctly and incorrectly classified instances

- For a two-class problems, four cases can occur:
  - true positives: positive class correctly predicted
  - false positives: positive class incorrectly predicted
  - true negatives: negative class correctly predicted
  - false negatives: negative class incorrectly predicted

Metrics for Performance Evaluation

- Focus on the predictive capability of a model
- Rather than how fast it takes to classify or build models

- Confusion Matrix:

	PREDICTED CLASS		
ACTUAL CLASS		Class=Yes	Class=No
	Class=Yes	TP	FN
	Class=No	FP	TN

Metrics for Performance Evaluation

- Most frequently used metrics:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\text{Error Rate} = 1 - Accuracy$$

	PREDICTED CLASS	
ACTUAL CLASS	Class=Yes	Class=No
Class=Yes	TP	FN
Class=No	FP	TN

What is a Good Accuracy?

- i.e., when are you done?
  - at 75% accuracy?
  - at 90% accuracy?
  - at 95% accuracy?

- Depends on difficulty of the problem!
- Baseline: naive guessing
  - always predict majority class

- Compare
  - Predicting coin tosses with accuracy of 50%
  - Predicting dice roll with accuracy of 50%
  - Predicting lottery numbers (6 out of 49) wth accuracy of 50%

Limitation of Accuracy: Unbalanced Data

- Sometimes, classes have very unequal frequency
  - Fraud detection: 98% transactions OK, 2% fraud
  - eCommerce: 99% don’t buy, 1% buy
  - Intruder detection: 99.99% of the users are no intruders
  - Security: >99.99% of Americans are not terrorists
- The class of interest is commonly called the positive class, and the rest negative classes.
- Consider a 2-class problem
  - Number of Class 0 examples = 9990, Number of Class 1 examples = 10
  - If model predicts everything to be class 0,
    accuracy is 9990/10000 = 99.9 %
  - Accuracy is misleading because model does not detect
    any class 1 example

Precision and Recall

Alternative: Use measures from information retrieval which are biased towards the positive class.

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

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

		Predicted Class	
		Yes	No
Actual Class	Yes	TP	FN
	No	FP	TN

Precision *p* is the number of correctly classified positive examples divided by the total number of examples that are classified as positive

Recall *r* is the number of correctly classified positive examples divided by the total number of actual positive examples in the test set

Precision and Recall Example

	Predicted positive	Predicted negative
Actual positive	1	99
Actual negative	0	1000

- This confusion matrix gives us
  - precision p = 100% and
  - recall r = 1%
- because we only classified one positive example correctly and no negative examples wrongly
- We want a measure that combines precision and recall

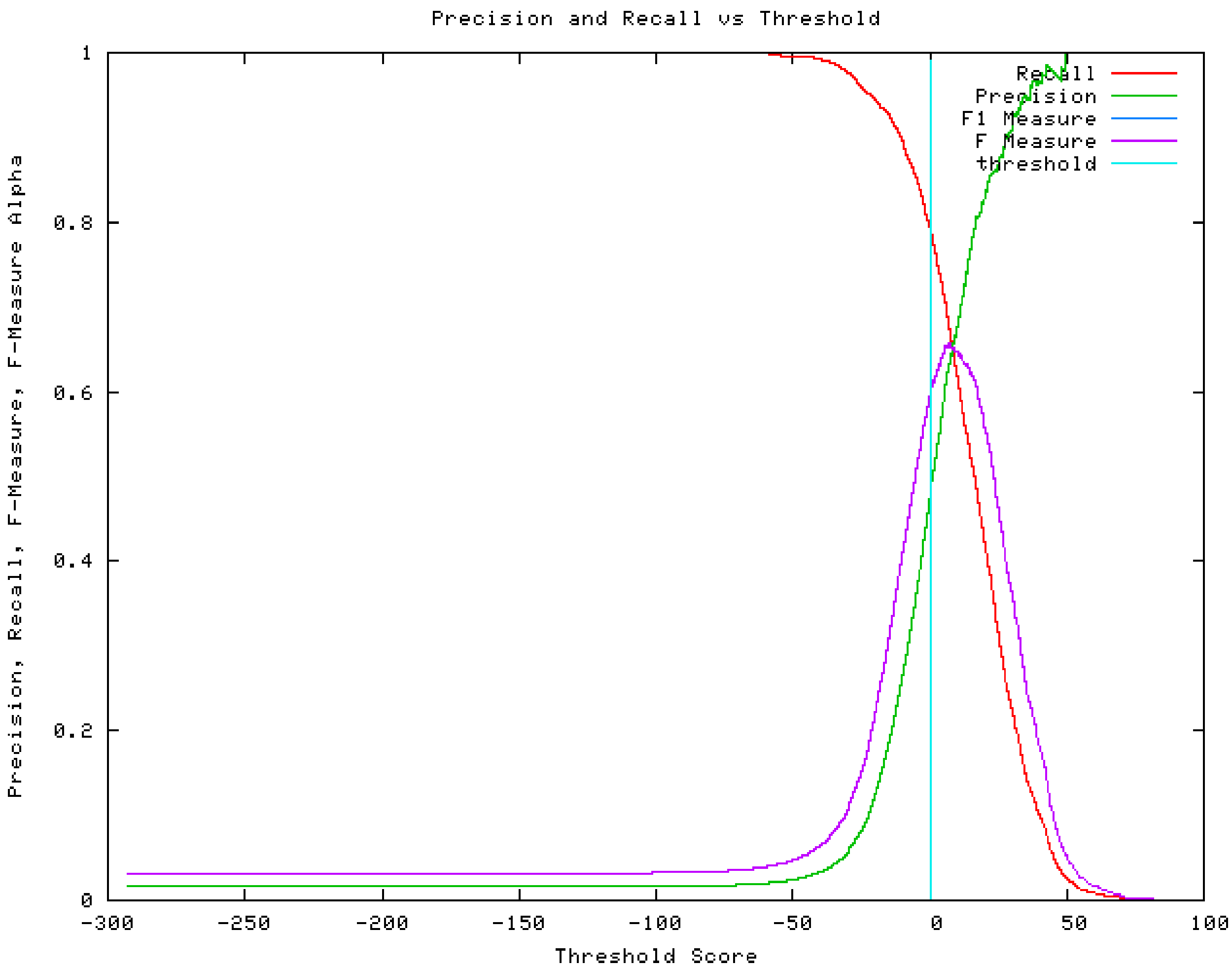
F_1-Measure

- It is hard to compare two classifiers using two measures
- F_1-Score combines precision and recall into one measure
  – by using the *harmonic mean*

$$F_1 = \frac{2}{\frac{1}{p} + \frac{1}{r}} = \frac{2pr}{p+r}$$

- The harmonic mean of two numbers tends to be closer to the smaller of the two
- For F_1-value to be large, both p and r must be large

F_1-Measure



Precision and Recall vs Threshold
Recall
Precision
F1 Measure
F Measure
threshold
Precision, Recall, F-Measure, F-Measure Alpha
Threshold Score
1
0.8
0.6
0.4
0.2
0
-300
-250
-200
-150
-100
-50
0
50
100

Alternative for Unbalanced Data: Cost Matrix

	PREDICTED CLASS		
ACTUAL CLASS	C(i|j)	**Class=Yes**	**Class=No**
	Class=Yes	C(Yes|Yes)	C(No|Yes)
	Class=No	C(Yes|No)	C(No|No)

C(i|j): Cost of misclassifying class j example as class i

Computing Cost of Classification

Cost Matrix	PREDICTED CLASS		
ACTUAL CLASS	C(i|j)	+	-
	+	0	100
	-	1	0

Model M_1	PREDICTED CLASS		
ACTUAL CLASS		+	-
	+	162	38
	-	160	240

Accuracy = 67%

Cost = 3798

Model M_2	PREDICTED CLASS		
ACTUAL CLASS		+	-
	+	155	45
	-	5	395

Accuracy = 92%

Cost = 4350

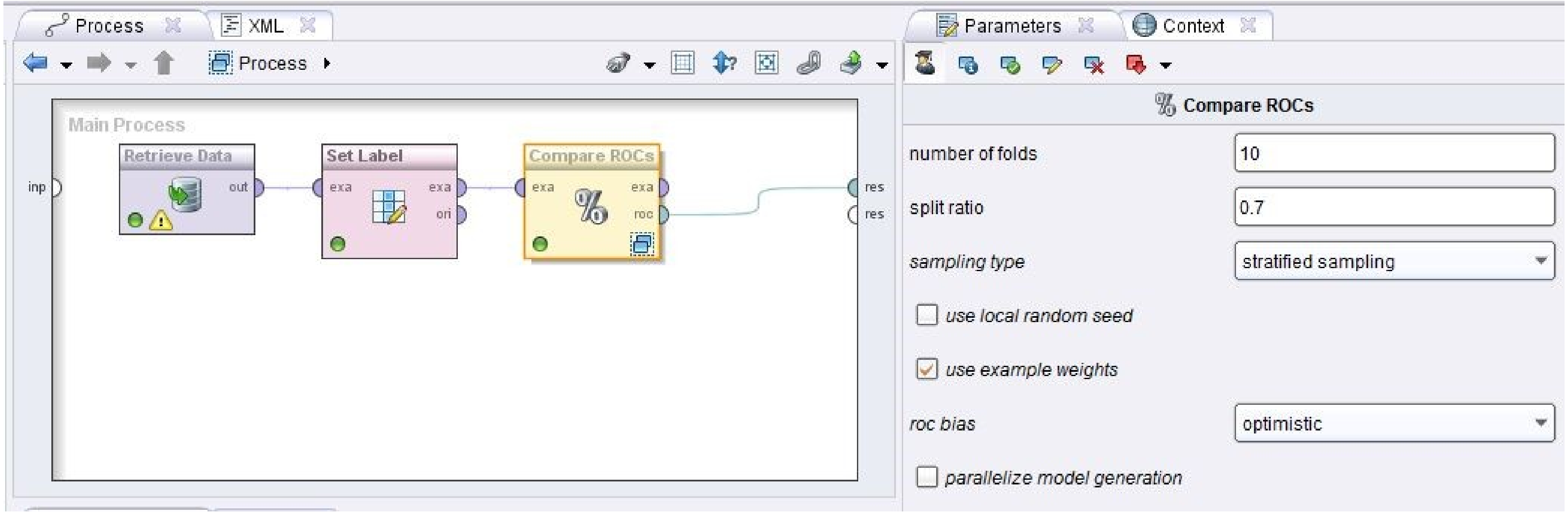
ROC Curves

- Some classification algorithms provide confidence scores
  - how sure the algorithms is with its prediction
  - e.g., Naive Bayes: the probability
  - e.g., Decision Trees: the purity of the respective leaf node

- Drawing a ROC Curve
  - Sort classifications according to confidence scores
  - Evaluate
    - correct prediction: draw one step up
    - incorrect prediction: draw one step to the right

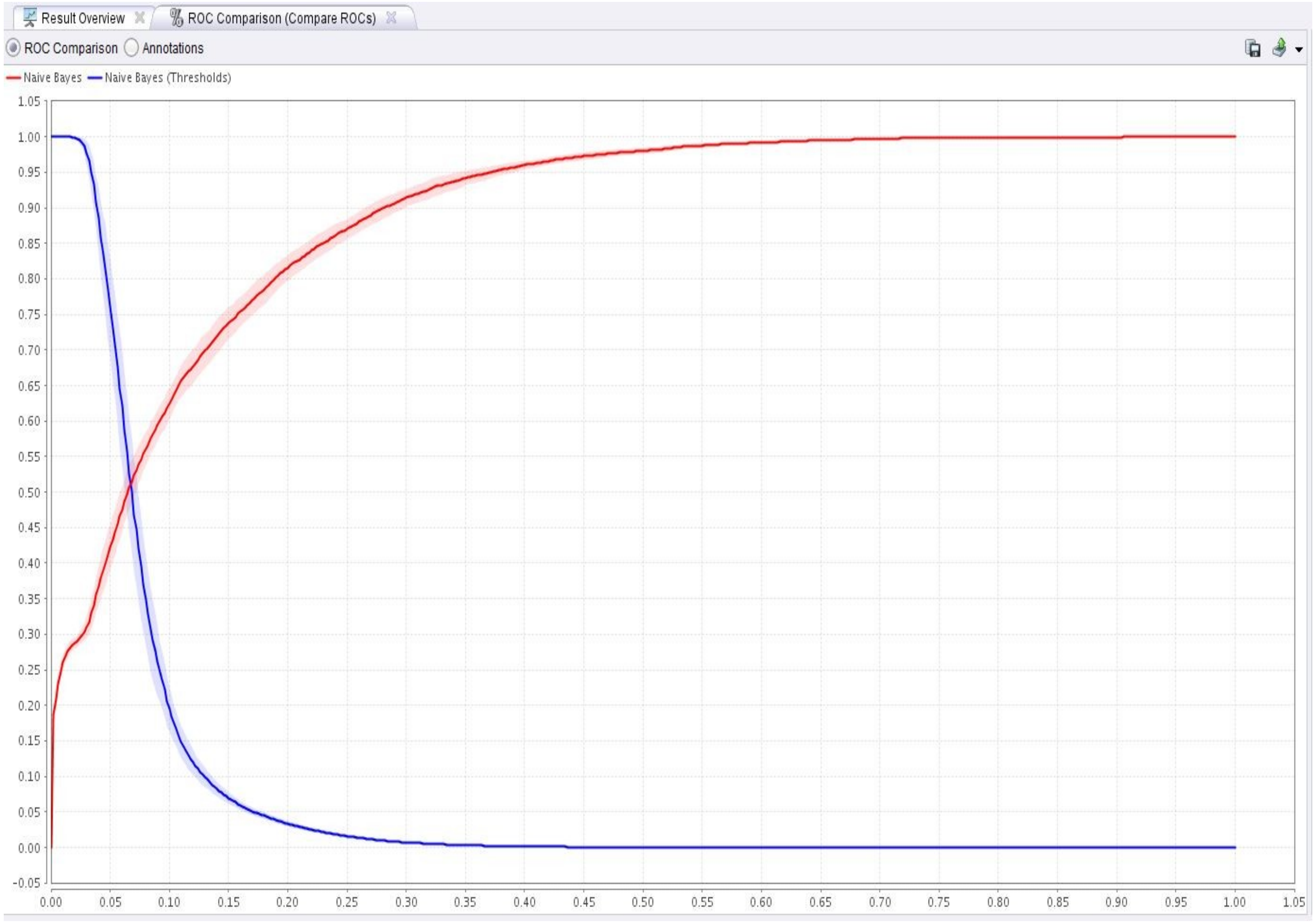
ROC Curves

- Drawing ROC Curves in RapidMiner & Python

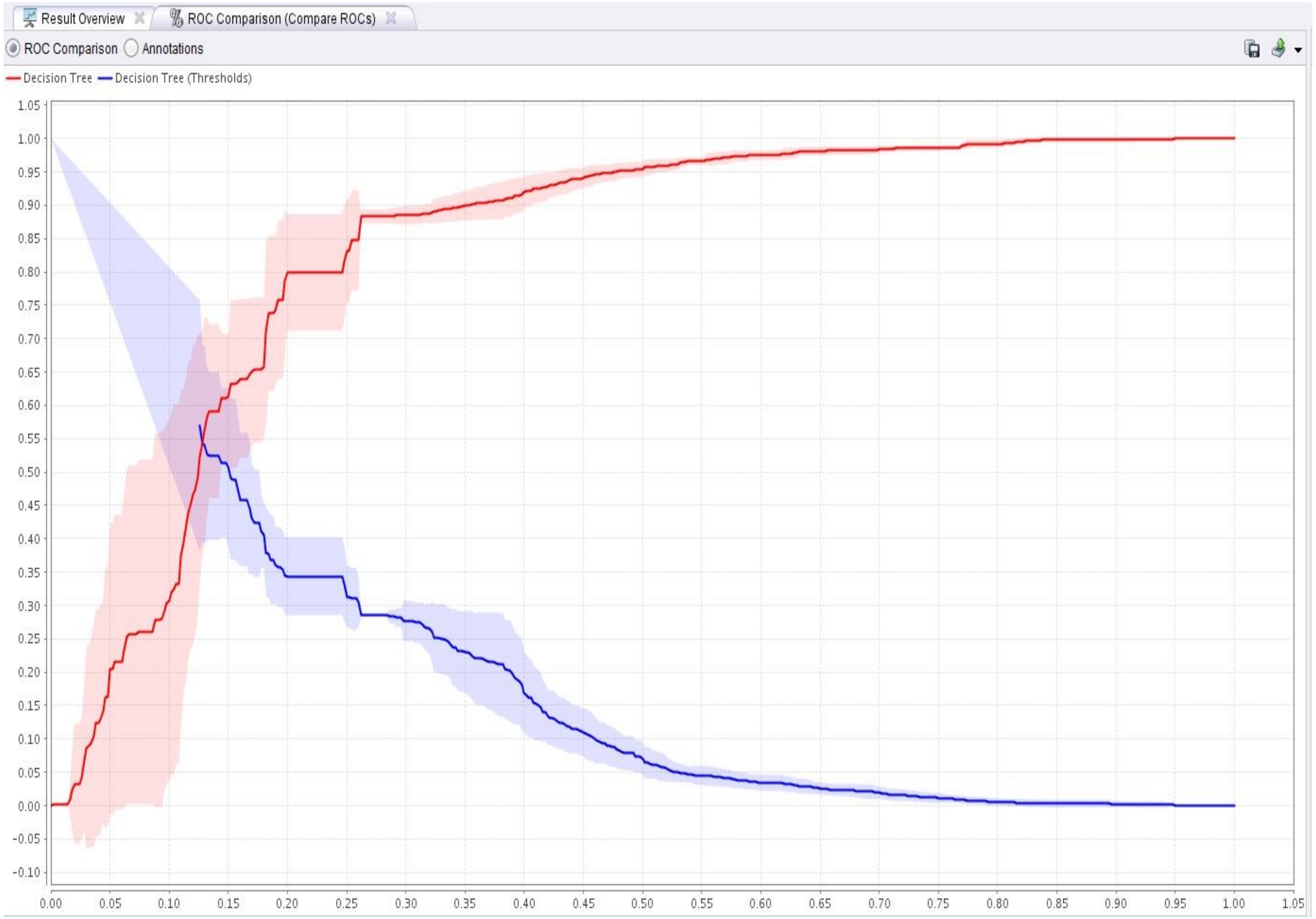



```
fpr, tpr, thresholds = roc_curve(actual, predictions)
plt.plot(fpr, tpr)
```

Example ROC Curve of Naive Bayes

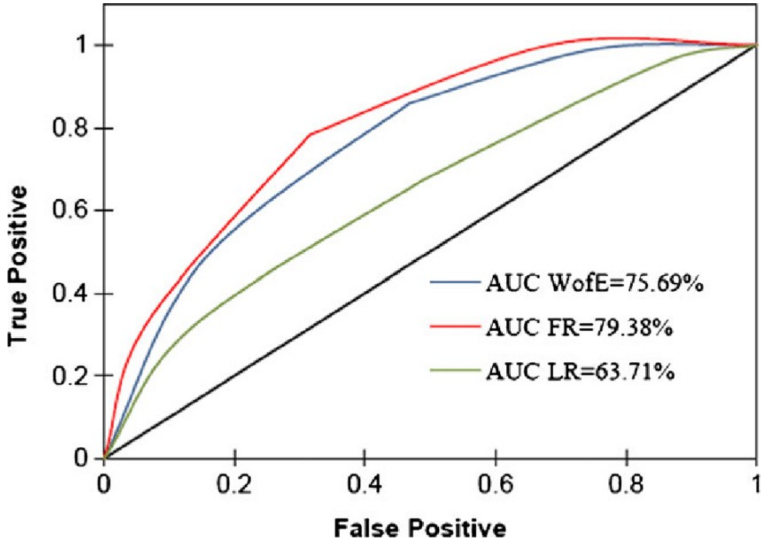


Example ROC Curve of Decision Tree Learner



Interpreting ROC Curves

- Best possible result:
  - all correct predictions have higher confidence than all incorrect ones
- The steeper, the better
  - random guessing results in the diagonal
  - so a decent algorithm should result in a curve significantly above the diagonal
- Comparing algorithms:
  - Curve A above curve B means algorithm A better than algorithm B
- Frequently used criterion
  - area under curve (aka ROC AUC)
  - normalized to 1

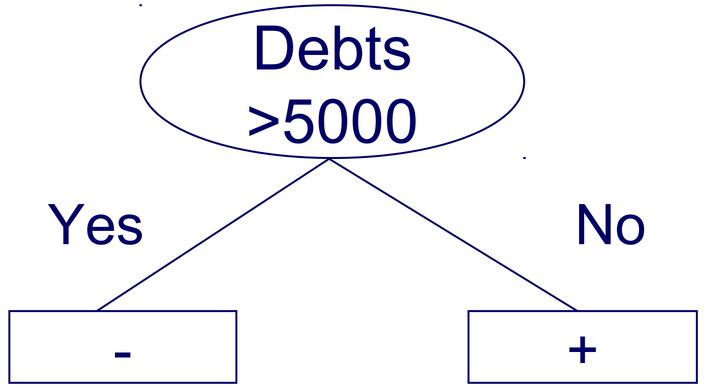


Methods for Performance Evaluation

- How to obtain a reliable estimate of performance?
- Performance of a model may depend on other factors besides the learning algorithm:
  - Size of training and test sets (it often expensive to get labeled data)
  - Class distribution (balanced, skewed)
  - Cost of misclassification (your goal)
- Methods for estimating the performance
  - Holdout
  - Random Subsampling
  - Cross Validation

Practical Issue: Overfitting

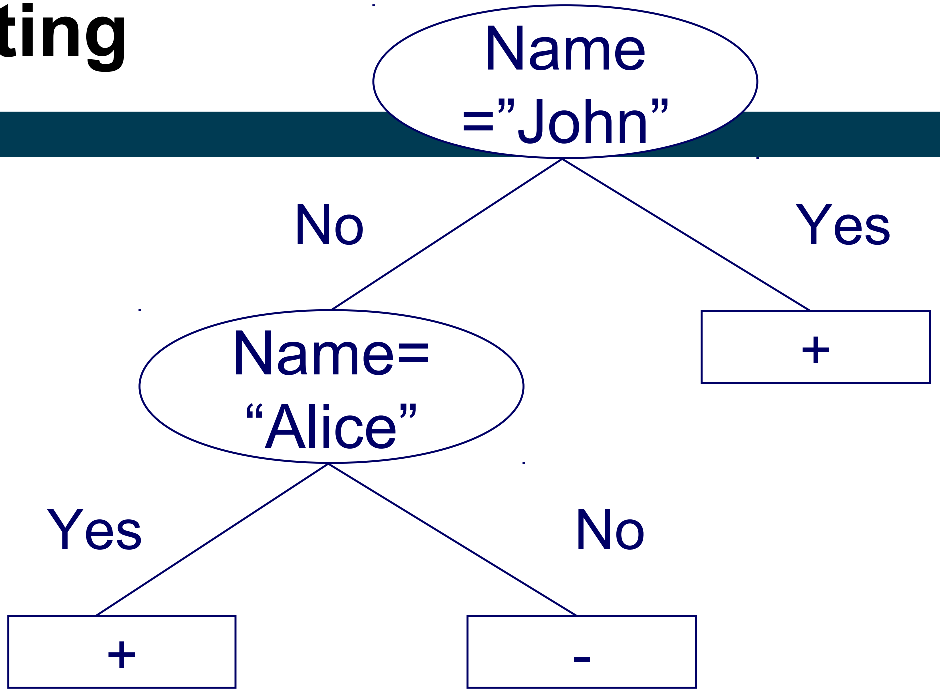
- Example: predict credit rating
  - possible decision tree:



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	+

Practical Issue: Overfitting

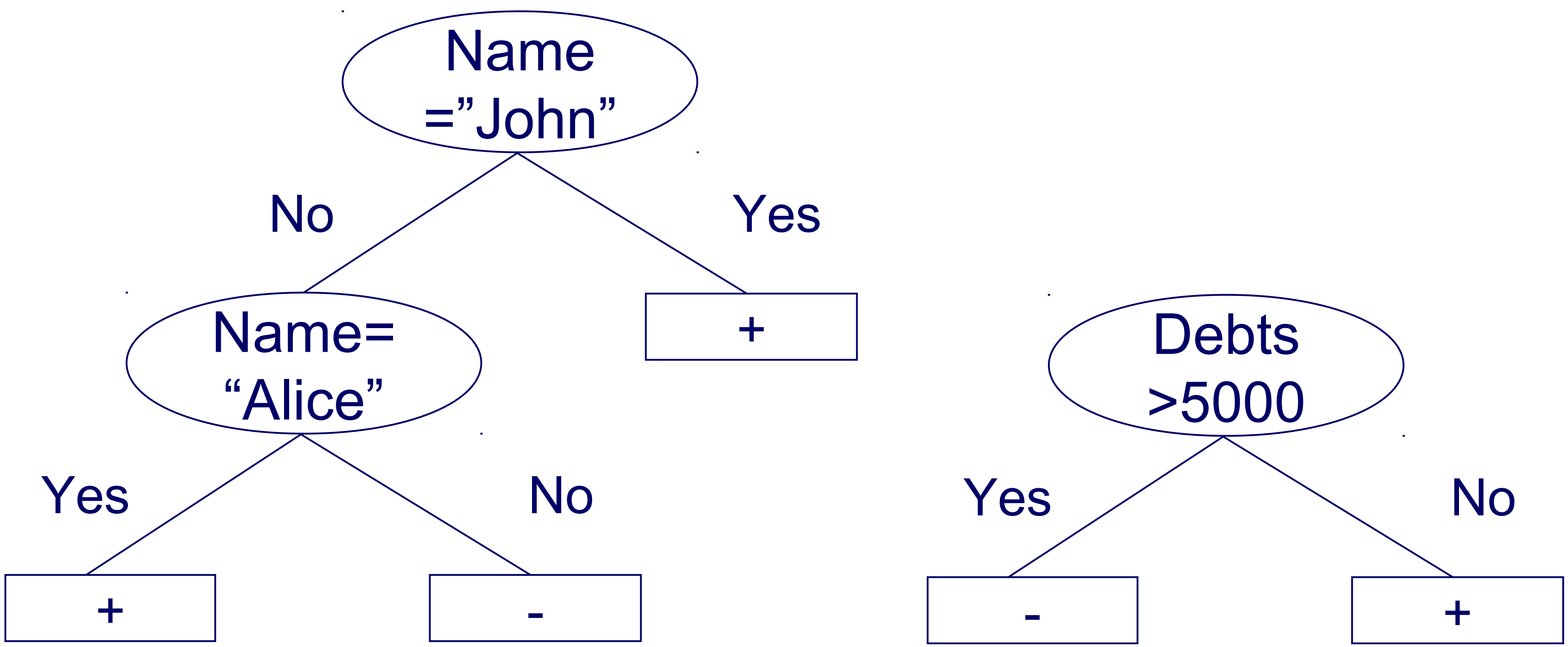
- Example: predict credit rating
  - alternative decision tree:



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	+

Practical Issue: Overfitting

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



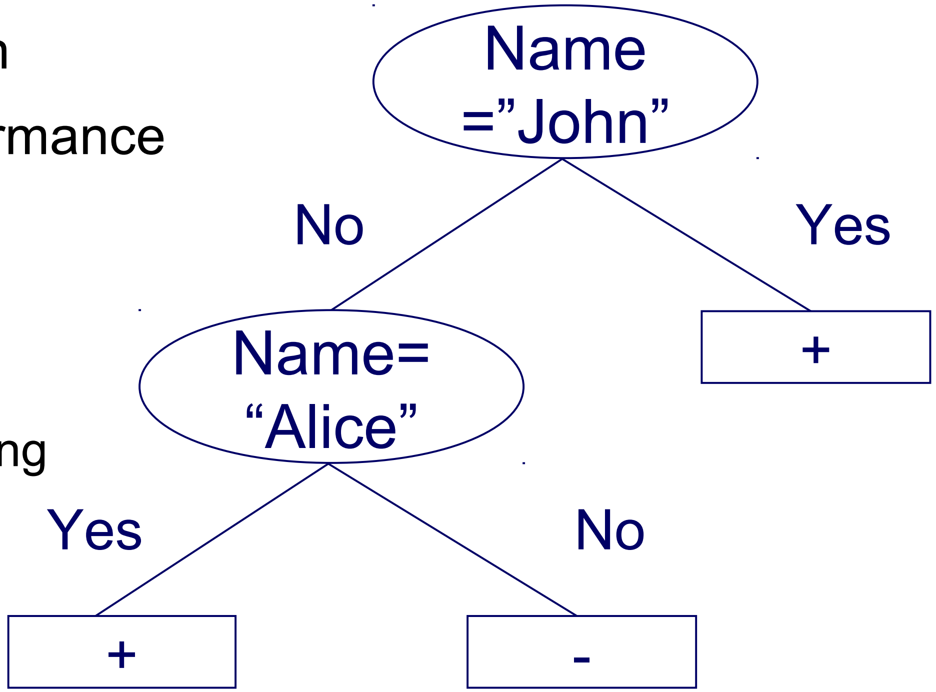
Occam's Razor

- 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

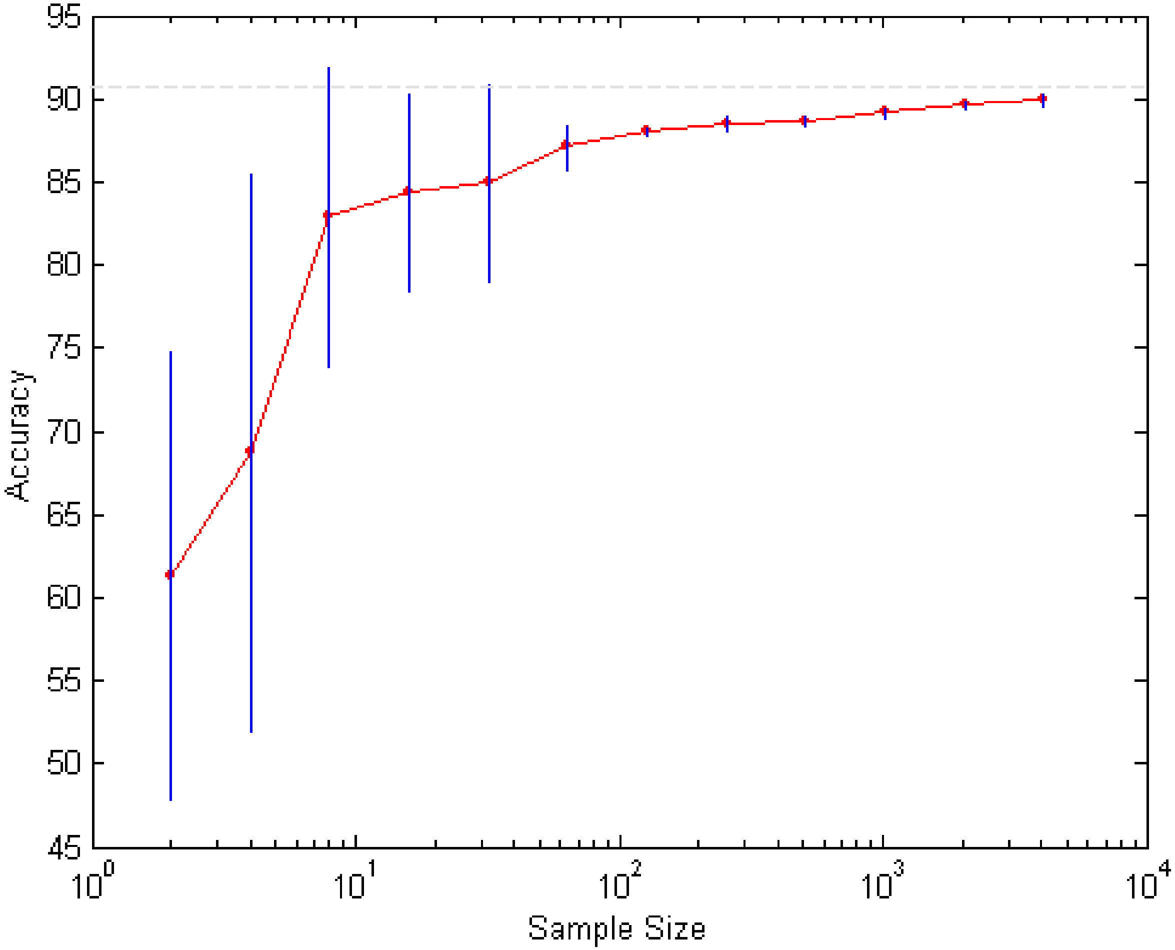
- Example:
  - phenomenon: the street is wet
  - theory 1: *it has rained*
  - theory 2: *a beer truck has had an accident, and beer has spilled. The truck has been towed, and magpies picked the glass pieces, so only the beer remains*

Training and Testing Data

- Consider the decision tree again
- Assume you measure the performance using the training data
- Conclusion:
  - We need separate data for testing



Learning Curve



- Learning curve shows how accuracy changes with varying sample size
- Conclusion: Use as much data as possible for training
- At the same time: variation drops with larger evaluation sets
- Conclusion: use as much data as possible for evaluation

Holdout Method

- The *holdout* method reserves a certain amount for testing and uses the remainder for training
- Usually: one third for testing, the rest for training
- applied when lots of sample data is available
- For unbalanced datasets, samples might not be representative
  - Few or none instances of some classes
- *Stratified sample: balances the data*
  - Make sure that each class is represented with approximately equal proportions in both subsets

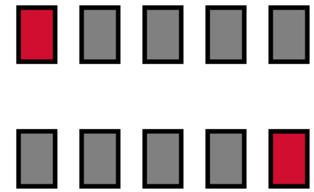
Leave One Out

- Iterate over all examples
  - train a model on all examples but the current one
  - evaluate on the current one

- Yields a very accurate estimate
- Uses as much data for training as possible
  - but is computationally infeasible in most cases

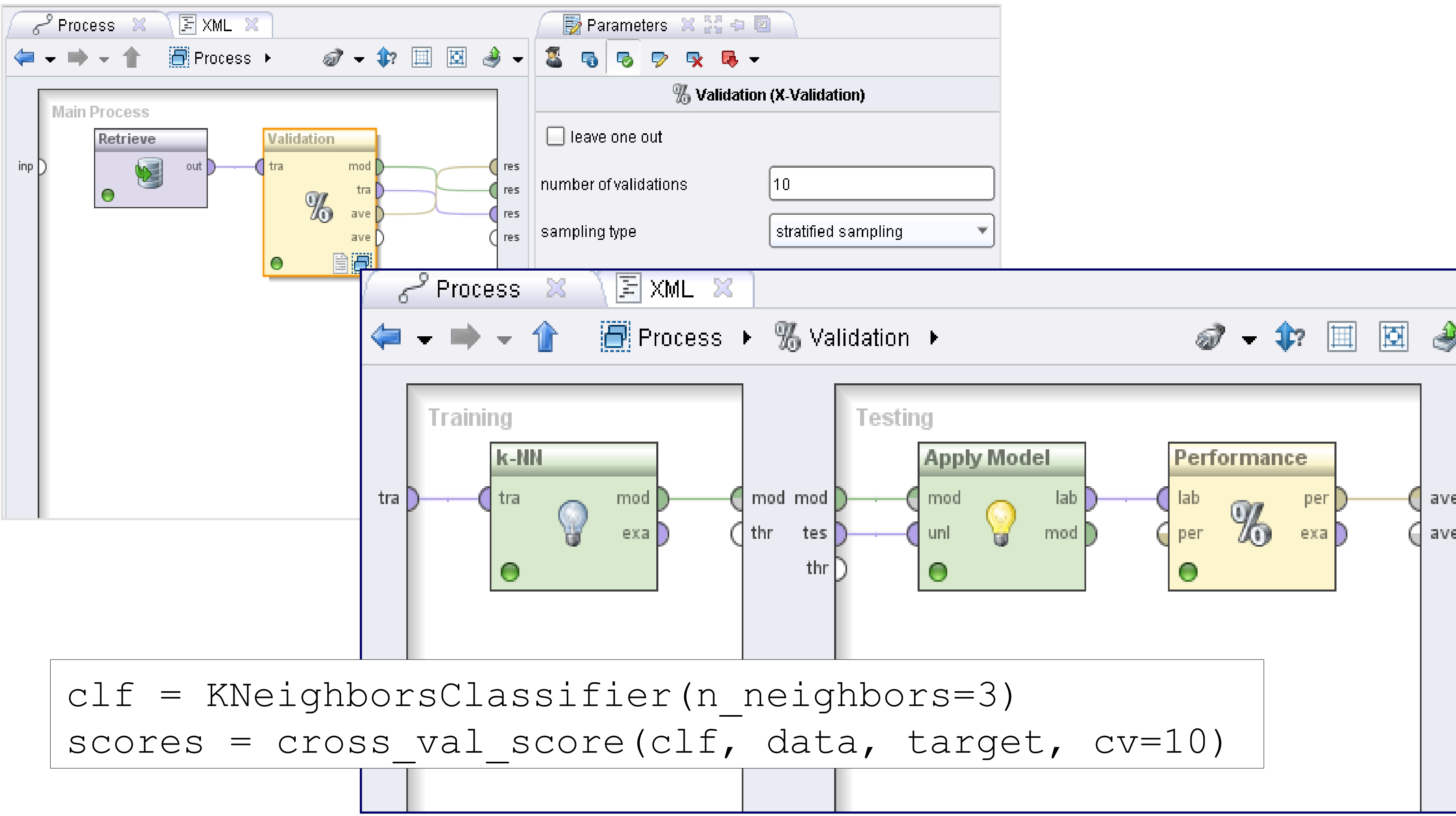
- Imagine: a dataset with a million instances
  - one minute to train a single model
  - Leave one out would take almost two years

Cross-Validation



- Compromise of Leave One Out and decent runtime
- *Cross-validation* avoids overlapping test sets
  - First step: data is split into *k* subsets of equal size
  - Second step: each subset in turn is used for testing and the remainder for training
- This is called *k-fold cross-validation*
- The error estimates are averaged to yield an overall error estimate
- Frequently used value for k : 10
  - Why ten? Extensive experiments have shown that this is the good choice to get an accurate estimate
- Often the subsets are stratified before the cross-validation is performed

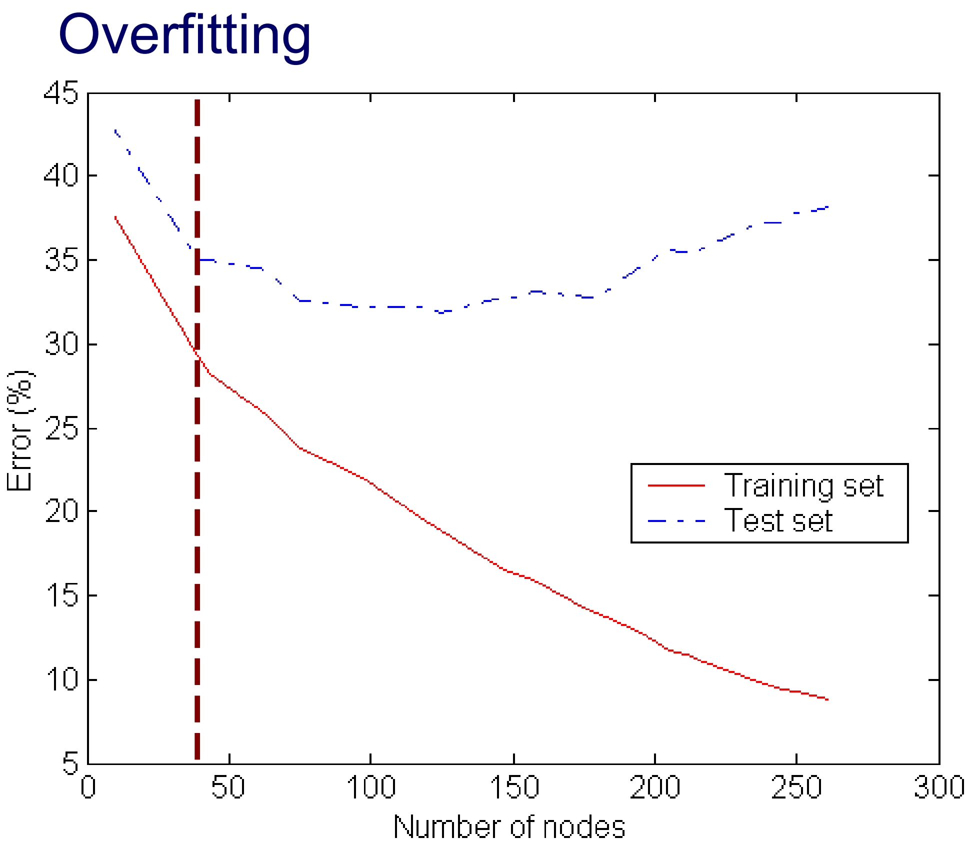
Cross-Validation in RapidMiner



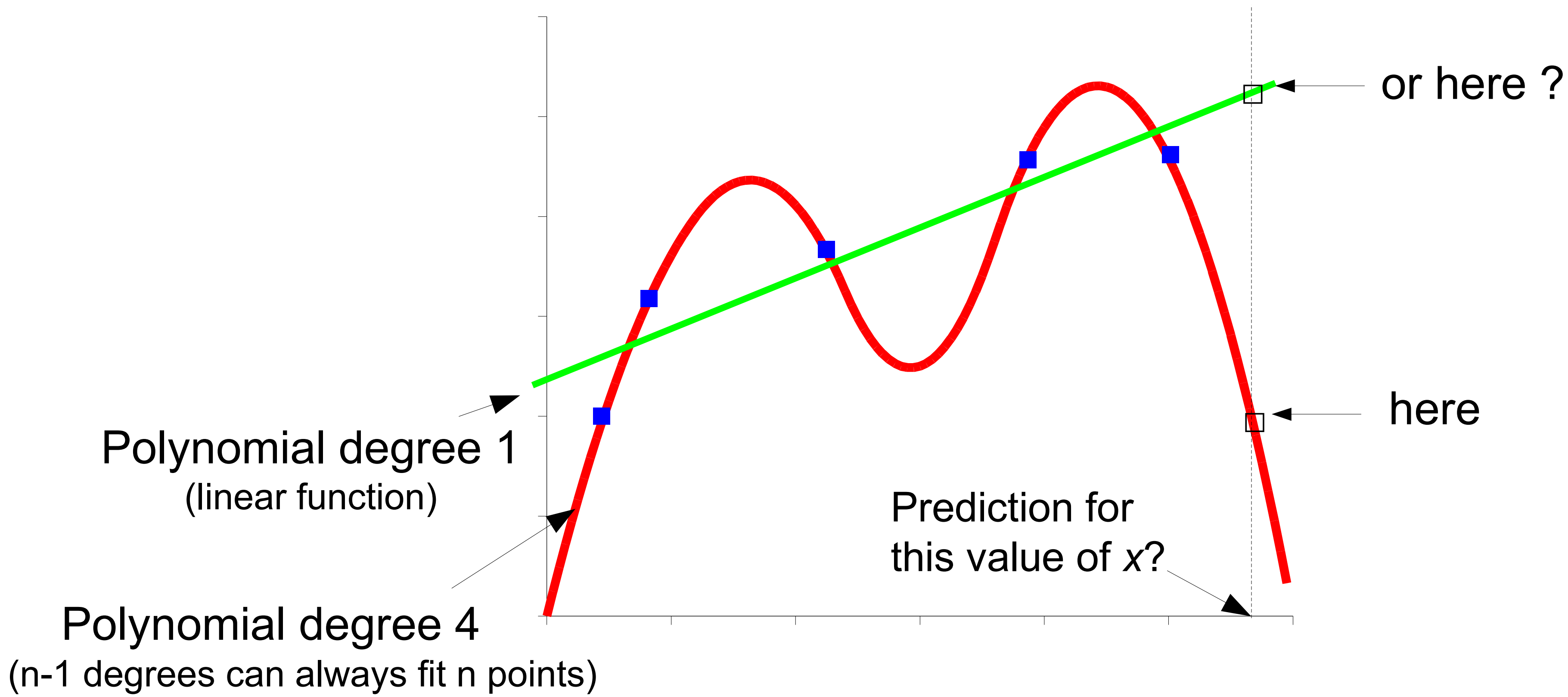

```
clf = KNeighborsClassifier(n_neighbors=3)
scores = cross_val_score(clf, data, target, cv=10)
```

Back to Overfitting

- Overfitting: Good accuracy on training data, but poor on test data.
- Symptoms: Tree too deep and too many branches
- Typical causes of overfitting
  - too little training data
  - noise
  - poor learning algorithm

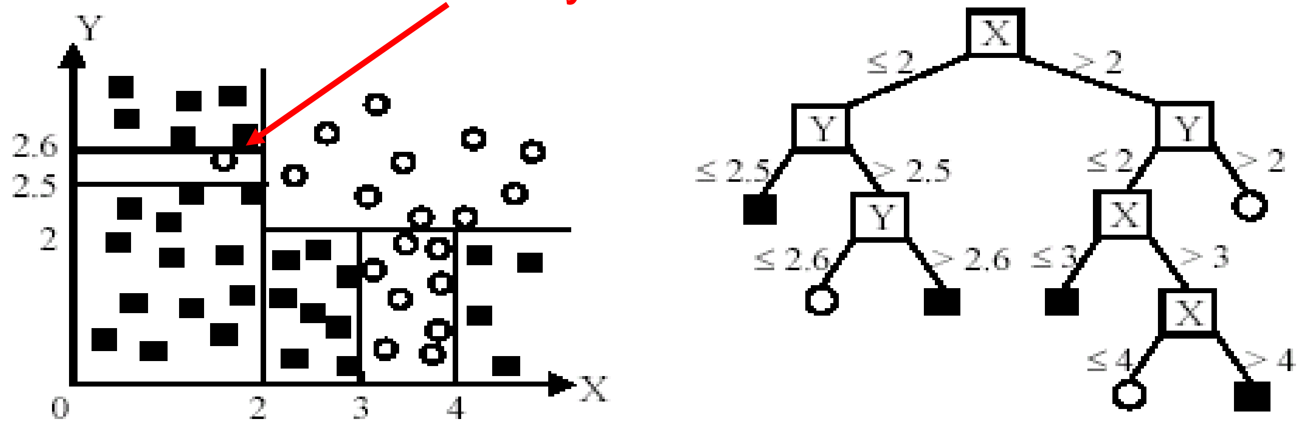


Overfitting - Illustration

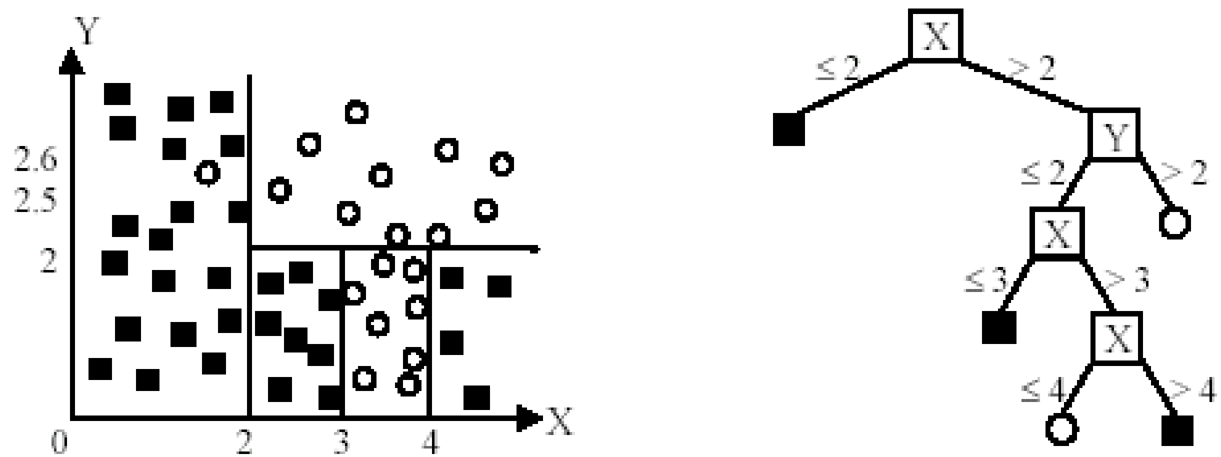


Overfitting and Noise

Likely to overfit the data



(A) A partition of the data space (B). The decision tree



How to Address Overfitting?

- Pre-Pruning (Early Stopping Rule)
  - Stop the algorithm before it becomes a fully-grown tree
  - Typical stopping conditions for a node:
    - Stop if all instances belong to the same class
    - Stop if all the attribute values are the same
  - Less restrictive conditions:
    - Stop if number of instances within a node is less than some user-specified threshold
    - Stop if expanding the current node only slightly improves the impurity measure (user-specified threshold)

How to Address Overfitting?

- Post-pruning
  1. Grow decision tree to its entire size
  2. Trim the nodes of the decision tree in a bottom-up fashion
     - using a validation data set
     - or an estimate of the generalization error
  3. If generalization error improves after trimming
     - replace sub-tree by a leaf node
     - Class label of leaf node is determined from majority class of instances in the sub-tree

Training vs. Generalization Errors

- Training error
  - also: resubstitution error, apparent error
  - errors made in training
  - evidence: misclassified training instances
- Generalization error
  - errors made on unseen data
  - evidence: no apparent evidence

- Training error can be computed
- Generalization error must be estimated

Estimating the Generalization Error

- Training errors: error on training (Σ e(t))
- Generalization errors: error on testing (Σ e'(t))
- Methods for estimating generalization errors:
  1. (Too) Optimistic approach: e'(t) = e(t)
  2. Pessimistic approach:
     - For each leaf node: e'(t) = (e(t)+0.5)
       (user-defined 0.5 penalty for large trees)
     - Total errors: e'(T) = e(T) + N $\times$ 0.5
       (N: number of leaf nodes)
     - For a tree with 30 leaf nodes and 10 errors on training (out of 1000 instances):
       Training error = 10/1000 = 1%
       Generalization error = (10 + 30$\times$0.5)/1000 = 2.5%
  3. Reduced Error Pruning (REP):
     - use validation data set to estimate generalization error

Example of Post-Pruning

Class = Yes	20
Class = No	10
Error = 10/30	

Training Error (Before splitting) = 10/30

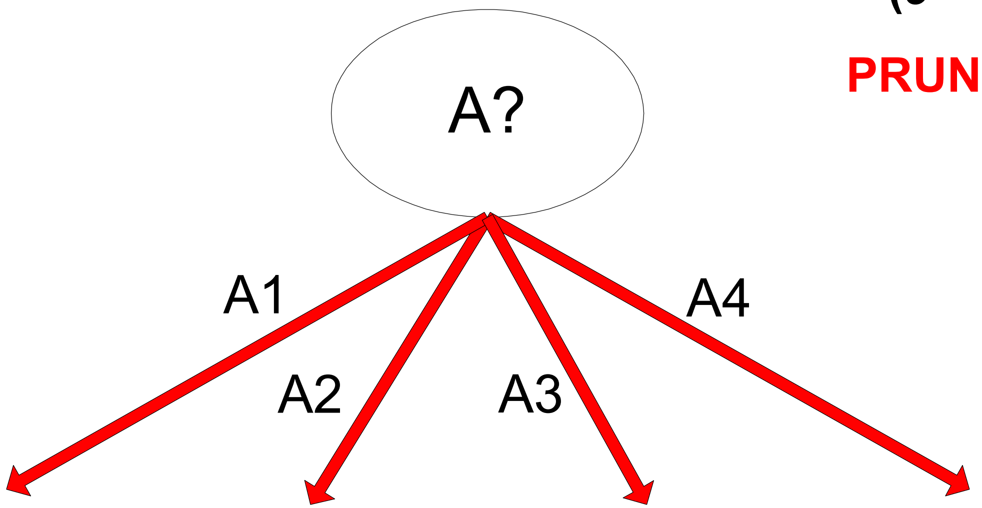
Pessimistic error = (10 + 0.5)/30 = 10.5/30

Training Error (After splitting) = 9/30

Pessimistic error (After splitting)

= (9 + 4 × 0.5)/30 = 11/30

PRUNE!



Class = Yes	8
Class = No	4

Class = Yes	3
Class = No	4

Class = Yes	4
Class = No	1

Class = Yes	5
Class = No	1

Discussion of Decision Trees

- Advantages:
  - Inexpensive to construct
  - Fast at classifying unknown records
  - Easy to interpret by humans for small-sized trees
  - Accuracy is comparable to other classification techniques for many simple data sets
- Disadvantages:
  - Decisions are based only one a single attribute at a time
  - Can only represent decision boundaries that are parallel to the axes

Comparing Decision Trees and k-NN

- Decision boundaries
  - k-NN: arbitrary
  - Decision trees: rectangular
- Sensitivity to scales
  - k-NN: needs normalization
  - Decision tree: does not require normalization (why?)
- Runtime & memory
  - k-NN is cheap to train, but expensive for classification
  - decision tree is expensive to train, but cheap for classification

Questions?