

# Data Mining I

## Classification, Part 2



Heiko Paulheim

# Outline

- 1) What is Classification?
- 2) K-Nearest-Neighbors
- 3) Decision Trees
- 4) Rule Learning
- 5) Decision Boundaries
- 6) Model Evaluation
- 7) Naïve Bayes
- 8) Artificial Neural Networks
- 9) Support Vector Machines
- 10) Parameter Tuning

# Rule-Based Classifiers

- Classify records by using a collection of “if...then...” rules
- Rule:  $(Condition) \rightarrow y$ 
  - where
    - *Condition* is a conjunctions of attributes
    - $y$  is the class label
  - *LHS*: rule antecedent or condition
  - *RHS*: rule consequent
  - Examples of classification rules:
    - $(\text{Blood Type}=\text{Warm}) \wedge (\text{Lay Eggs}=\text{Yes}) \rightarrow \text{Birds}$
    - $(\text{Taxable Income} < 50\text{K}) \wedge (\text{Refund}=\text{Yes}) \rightarrow \text{Evade}=\text{No}$

# Rule-based Classifier (Example)

Name	Blood Type	Give Birth	Can Fly	Live in Water	Class
human	warm	yes	no	no	mammals
python	cold	no	no	no	reptiles
salmon	cold	no	no	yes	fishes
whale	warm	yes	no	yes	mammals
frog	cold	no	no	sometimes	amphibians
komodo	cold	no	no	no	reptiles
bat	warm	yes	yes	no	mammals
pigeon	warm	no	yes	no	birds
cat	warm	yes	no	no	mammals
leopard shark	cold	yes	no	yes	fishes
turtle	cold	no	no	sometimes	reptiles
penguin	warm	no	no	sometimes	birds
porcupine	warm	yes	no	no	mammals
eel	cold	no	no	yes	fishes
salamander	cold	no	no	sometimes	amphibians
gila monster	cold	no	no	no	reptiles
platypus	warm	no	no	no	mammals
owl	warm	no	yes	no	birds
dolphin	warm	yes	no	yes	mammals
eagle	warm	no	yes	no	birds

R1: (Give Birth = no)  $\wedge$  (Can Fly = yes)  $\rightarrow$  Birds

R2: (Give Birth = no)  $\wedge$  (Live in Water = yes)  $\rightarrow$  Fishes

R3: (Give Birth = yes)  $\wedge$  (Blood Type = warm)  $\rightarrow$  Mammals

R4: (Give Birth = no)  $\wedge$  (Can Fly = no)  $\rightarrow$  Reptiles

R5: (Live in Water = sometimes)  $\rightarrow$  Amphibians

# Application of Rule-Based Classifiers

- A rule  $r$  **covers** an instance  $x$  if the attributes of the instance satisfy the condition of the rule

R1: (Give Birth = no)  $\wedge$  (Can Fly = yes)  $\rightarrow$  Birds

R2: (Give Birth = no)  $\wedge$  (Live in Water = yes)  $\rightarrow$  Fishes

R3: (Give Birth = yes)  $\wedge$  (Blood Type = warm)  $\rightarrow$  Mammals

R4: (Give Birth = no)  $\wedge$  (Can Fly = no)  $\rightarrow$  Reptiles

R5: (Live in Water = sometimes)  $\rightarrow$  Amphibians

Name	Blood Type	Give Birth	Can Fly	Live in Water	Class
hawk	warm	no	yes	no	?
grizzly bear	warm	yes	no	no	?

Rule R1 covers hawk  $\Rightarrow$  Bird

Rule R3 covers grizzly bear  $\Rightarrow$  Mammal

# Rule Coverage and Accuracy

- Coverage of a rule:
  - Fraction of records that satisfy the antecedent of a rule
- Accuracy of a rule:
  - Fraction of records that satisfy **both** the antecedent **and** consequent of a rule

**(Status=Single) → No**

**Coverage = 40%, Accuracy = 50%**

<i>Tid</i>	Refund	Marital Status	Taxable Income	C
1	Yes	Single	125K	N
2	No	Married	100K	N
3	No	Single	70K	N
4	Yes	Married	120K	N
5	No	Divorced	95K	Y
6	No	Married	60K	N
7	Yes	Divorced	220K	N
8	No	Single	85K	Y
9	No	Married	75K	N

# How does a Rule-based Classifier Work?

R1: (Give Birth = no)  $\wedge$  (Can Fly = yes)  $\rightarrow$  Birds

R2: (Give Birth = no)  $\wedge$  (Live in Water = yes)  $\rightarrow$  Fishes

R3: (Give Birth = yes)  $\wedge$  (Blood Type = warm)  $\rightarrow$  Mammals

R4: (Give Birth = no)  $\wedge$  (Can Fly = no)  $\rightarrow$  Reptiles

R5: (Live in Water = sometimes)  $\rightarrow$  Amphibians

Name	Blood Type	Give Birth	Can Fly	Live in Water	Class
lemur	warm	yes	no	no	?
turtle	cold	no	no	sometimes	?
dogfish shark	cold	yes	no	yes	?

A lemur triggers rule R3, so it is classified as a mammal

A turtle triggers both R4 and R5

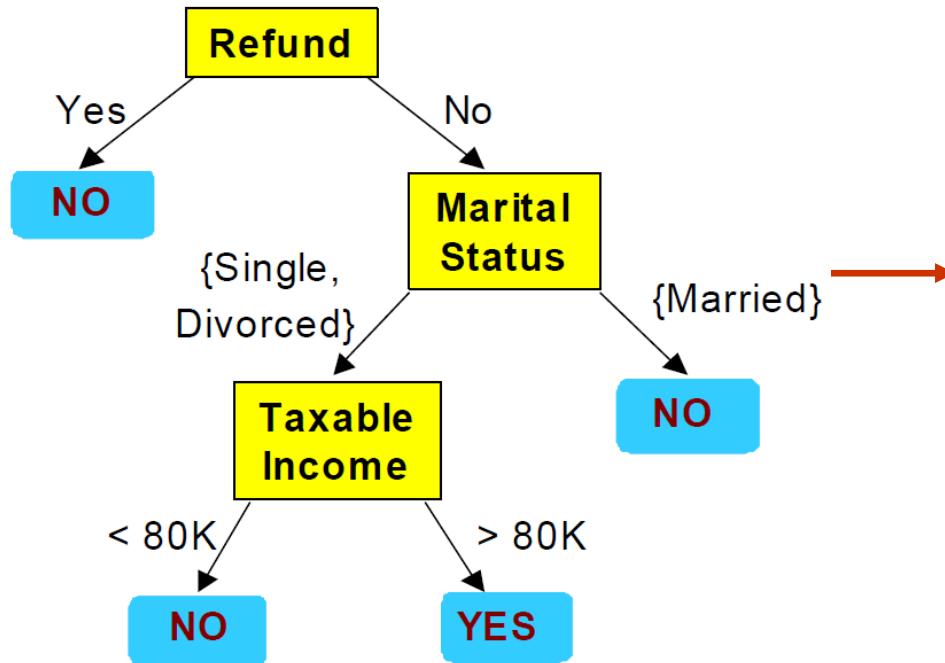
A dogfish shark triggers none of the rules

# Characteristics of Rule-Based Classifiers

- Mutually exclusive rules
  - Classifier contains mutually exclusive rules if the rules are independent of each other
  - Every example is covered by at most one rule→ avoids conflicts
- Exhaustive rules
  - Classifier has exhaustive coverage if it accounts for every possible combination of attribute values
  - Each record is covered by at least one rule→ enforces each example to be classified



# From Decision Trees To Rules



## Classification Rules

$(\text{Refund}=\text{Yes}) \implies \text{No}$

$(\text{Refund}=\text{No}, \text{Marital Status}=\{\text{Single}, \text{Divorced}\}, \text{Taxable Income} < 80\text{K}) \implies \text{No}$

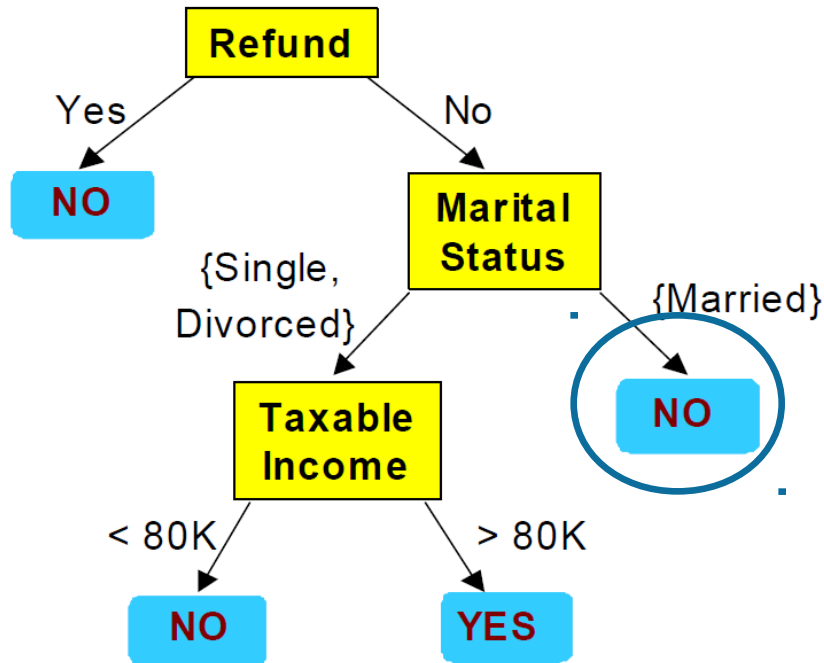
$(\text{Refund}=\text{No}, \text{Marital Status}=\{\text{Single}, \text{Divorced}\}, \text{Taxable Income} > 80\text{K}) \implies \text{Yes}$

$(\text{Refund}=\text{No}, \text{Marital Status}=\{\text{Married}\}) \implies \text{No}$

**Rules are mutually exclusive and exhaustive**

**Rule set contains as much information as the tree**

# Rules Can Be Simplified



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

Initial Rule:  $(\text{Refund}=\text{No}) \wedge (\text{Status}=\text{Married}) \rightarrow \text{No}$

Simplified Rule:  $(\text{Status}=\text{Married}) \rightarrow \text{No}$

# Possible Effects of Rule Simplification

- Rules are no longer mutually exclusive
  - A record may trigger more than one rule
  - Solution?
    - Ordered rule set
    - Unordered rule set – use voting schemes
- Rules are no longer exhaustive
  - A record may not trigger any rules
  - Solution?
    - Use a default class

# Ordered Rule Set

- Rules are ranked ordered according to their priority
  - An ordered rule set is known as a *decision list*
- When a test record is presented to the classifier
  - It is assigned to the class label of the highest ranked rule it has triggered
  - If none of the rules fired, it is assigned to the default class

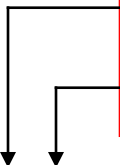
R1: (Give Birth = no)  $\wedge$  (Can Fly = yes)  $\rightarrow$  Birds

R2: (Give Birth = no)  $\wedge$  (Live in Water = yes)  $\rightarrow$  Fishes

R3: (Give Birth = yes)  $\wedge$  (Blood Type = warm)  $\rightarrow$  Mammals

R4: (Give Birth = no)  $\wedge$  (Can Fly = no)  $\rightarrow$  Reptiles

R5: (Live in Water = sometimes)  $\rightarrow$  Amphibians



Name	Blood Type	Give Birth	Can Fly	Live in Water	Class
turtle	cold	no	no	sometimes	?

# Rule Ordering Schemes

- Rule-based ordering
  - Individual rules are ranked based on their quality (e.g., accuracy)
- Class-based ordering
  - Rules that belong to the same class appear together

## Rule-based Ordering

(Refund=Yes) ==> No

(Refund=No, Marital Status={Single,Divorced},  
Taxable Income<80K) ==> No

(Refund=No, Marital Status={Single,Divorced},  
Taxable Income>80K) ==> Yes

(Refund=No, Marital Status={Married}) ==> No

## Class-based Ordering

(Refund=Yes) ==> No

(Refund=No, Marital Status={Single,Divorced},  
Taxable Income<80K) ==> No

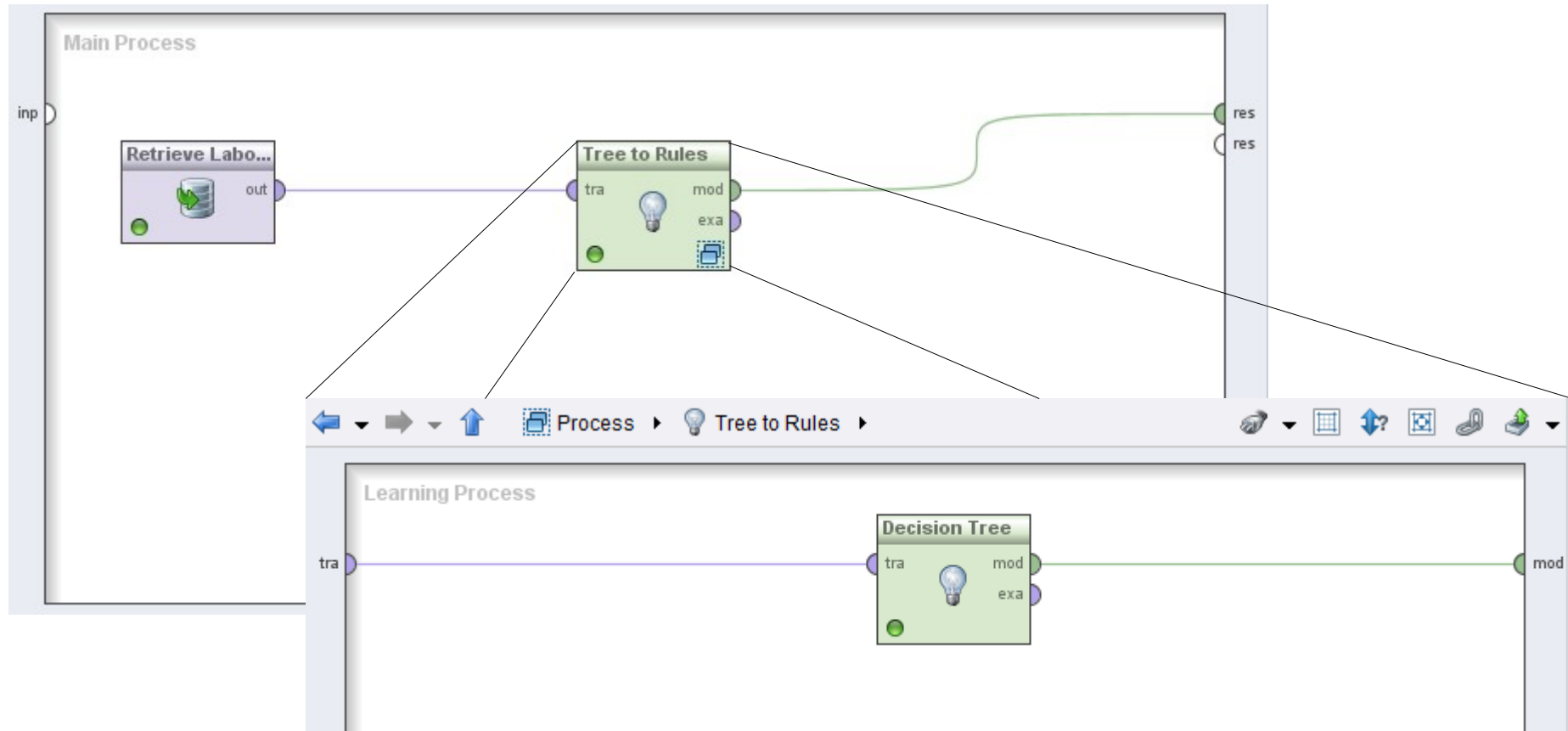
(Refund=No, Marital Status={Married}) ==> No

(Refund=No, Marital Status={Single,Divorced},  
Taxable Income>80K) ==> Yes

# Indirect Method: C4.5rules

- Extract rules from an unpruned decision tree
- For each rule,  $r: A \rightarrow y$ ,
  - consider an alternative rule  $r': A' \rightarrow y$  where  $A'$  is obtained by removing one of the conjuncts in  $A$
  - Compare the pessimistic generalization error for  $r$  against all  $r'$
  - Prune if one of the  $r'$ s has a lower pessimistic generalization error
  - Repeat until we can no longer improve generalization error

# Indirect Method in RapidMiner



# Direct vs. Indirect Rule Learning Methods

- Direct Method:
  - Extract rules directly from data
  - e.g.: RIPPER, CN2, Holte's 1R
- Indirect Method:
  - Extract rules from other classification models (e.g. decision trees, neural networks, etc).
  - Example: C4.5rules



# Direct methods

- Do not derive rules from another type of model
  - but learn the rules directly
- practical algorithms use different approaches
  - covering or separate-and-conquer algorithms
  - based on heuristic search

The following slides are based on the machine learning course by Johannes Fürnkranz, Technische Universität Darmstadt

# A sample task

<i>Temperature</i>	<i>Outlook</i>	<i>Humidity</i>	<i>Windy</i>	<i>Play Golf?</i>
hot	sunny	high	false	no
hot	sunny	high	true	no
hot	overcast	high	false	yes
cool	rain	normal	false	yes
cool	overcast	normal	true	yes
mild	sunny	high	false	no
cool	sunny	normal	false	yes
mild	rain	normal	false	yes
mild	sunny	normal	true	yes
mild	overcast	high	true	yes
hot	overcast	normal	false	yes
mild	rain	high	true	no
cool	rain	normal	true	no

- Task:
  - Find a rule set that correctly predicts the dependent variable from the observed variables

# A Simple Solution

IF	T=hot	AND	H=high	AND	O=overcast	AND	W=false	THEN	yes
IF	T=cool	AND	H=normal	AND	O=rain	AND	W=false	THEN	yes
IF	T=cool	AND	H=normal	AND	O=overcast	AND	W=true	THEN	yes
IF	T=cool	AND	H=normal	AND	O=sunny	AND	W=false	THEN	yes
IF	T=mild	AND	H=normal	AND	O=rain	AND	W=false	THEN	yes
IF	T=mild	AND	H=normal	AND	O=sunny	AND	W=true	THEN	yes
IF	T=mild	AND	H=high	AND	O=overcast	AND	W=true	THEN	yes
IF	T=hot	AND	H=normal	AND	O=overcast	AND	W=false	THEN	yes
IF	T=mild	AND	H=high	AND	O=rain	AND	W=false	THEN	yes

- The solution is
  - a set of rules
  - that is complete and consistent on the training examples
- “**Overfitting** is like memorizing the answers to a test instead of understanding the principles.” (Bob Horton, 2015)

# A Better Solution

IF Outlook = overcast	THEN yes
IF Humidity = normal AND Outlook = sunny	THEN yes
IF Outlook = rainy AND Windy = false	THEN yes

# A Simple Algorithm: Batch-Find

- Abstract algorithm for learning a single rule:

1. Start with an empty theory  $T$  and training set  $E$
2. Learn a single (*consistent*) rule  $R$  from  $E$  and add it to  $T$
3. return  $T$

- Problem:
  - the basic assumption is that the found rules are complete, i.e., they cover all positive examples
  - What if they don't?
- Simple solution:
  - If we have a rule that covers part of the positive examples, add some more rules that cover the remaining examples

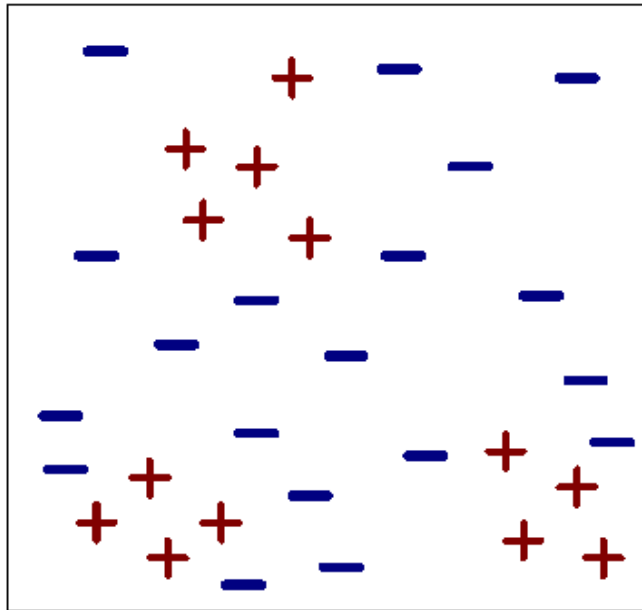
# Separate-and-Conquer Rule Learning

- Learn a set of rules, one by one

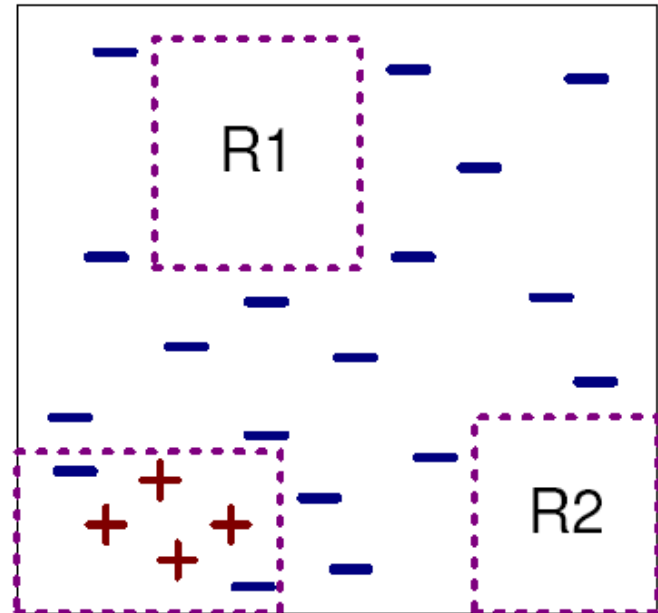
1. Start with an empty theory  $T$  and training set  $E$
2. Learn a single (*consistent*) rule  $R$  from  $E$  and add it to  $T$
3. If  $T$  is *satisfactory (complete)*, return  $T$
4. Else:
  - » Separate: Remove examples explained by  $R$  from  $E$
  - » Conquer: goto 2.

- One of the oldest family of learning algorithms
  - goes back to AQ (Michalski, 60s)
  - FRINGE, PRISM and CN2: relation to decision trees (80s)
  - popularized in ILP (FOIL and PROGOL, 90s)
  - RIPPER brought in good noise-handling
- Different learners differ in how they find a single rule

# Separate-and-Conquer Rule Learning



(i) Original Data



(iv) Step 3

# Relaxing Completeness and Consistency

- So far we have always required a learner to learn a complete and consistent theory
  - e.g., one rule that covers all positive and no negative examples
- This is not always a good idea (→ [overfitting](#))
- Example: Training set with 200 examples, 100 positive and 100 negative
  - **Theory A** consists of 100 complex rules, each covering a single positive example and no negatives
    - Theory A is **complete and consistent** on the training set
  - **Theory B** consists of one simple rule, covering 99 positive and 1 negative example
    - Theory B is **incomplete and inconsistent** on the training set
- Which one will generalize better to unseen examples?



# Top-Down Hill-Climbing

- **Top-Down** Strategy: A rule is successively *specialized*

1. Start with the universal rule R that covers all examples
2. Evaluate all possible ways to add a condition to R
3. Choose the best one (according to some heuristic)
4. If R is satisfactory, return it
5. Else goto 2.

- Almost all greedy s&c rule learning systems use this strategy

# Recap: Terminology

- training examples
  - $P$ : total number of positive examples
  - $N$ : total number of negative examples
- examples covered by the rule (predicted positive)
  - **true positives**  $p$ : positive examples covered by the rule
  - **false positives**  $n$ : negative examples covered by the rule
- examples not covered the rule (predicted negative)
  - **false negatives**  $P-p$ : positive examples not covered by the rule
  - **true negatives**  $N-n$ : negative examples not covered by the rule

	predicted +	predicted -	
class +	$p$ ( <b>true positives</b> )	$P-p$ ( <b>false negatives</b> )	$P$
class -	$n$ ( <b>false positives</b> )	$N-n$ ( <b>true negatives</b> )	$N$

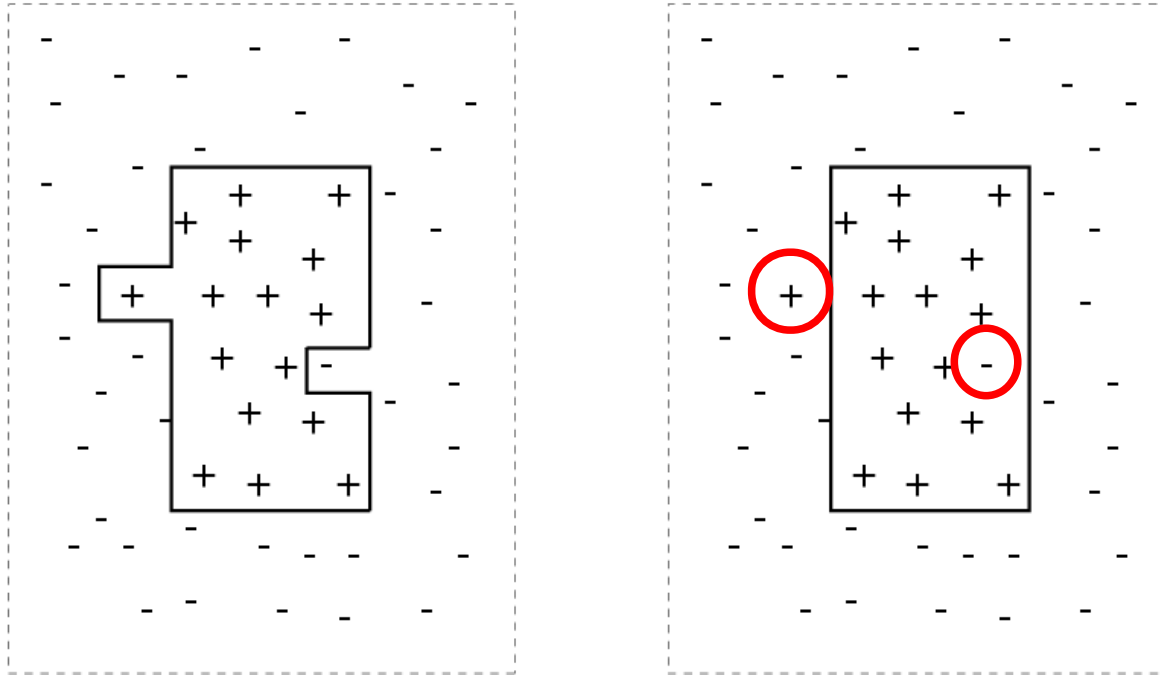
# Rule Learning Heuristics

- Adding a rule should
  - increase the number of covered negative examples as little as possible (do **not decrease consistency**)
  - increase the number of covered positive examples as much as possible (**increase completeness**)
- An evaluation heuristic should therefore trade off these two extremes
  - Example: **Laplace heuristic**  $h_{Lap} = \frac{p+1}{p+n+2}$ 
    - grows with  $p \rightarrow \infty$
    - grows with  $n \rightarrow 0$

# Recap: Overfitting

- Overfitting
  - Given
    - a fairly general model class
    - enough degrees of freedom
  - you can always find a model that explains the data
    - even if the data contains errors (noise in the data)
    - in rule learning: each example is a rule
- Such concepts do not generalize well!
  - Solution: Rule and rule set pruning

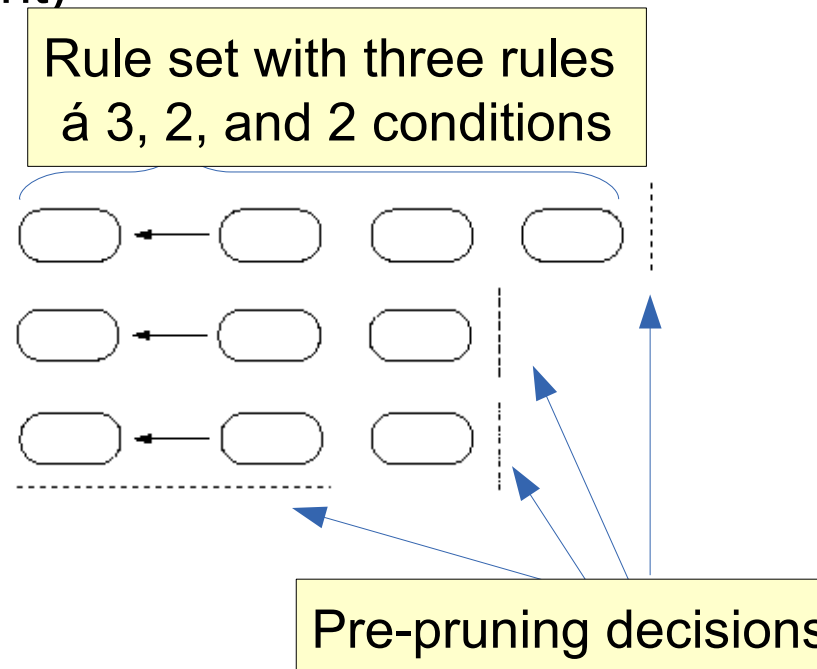
# Overfitting Avoidance



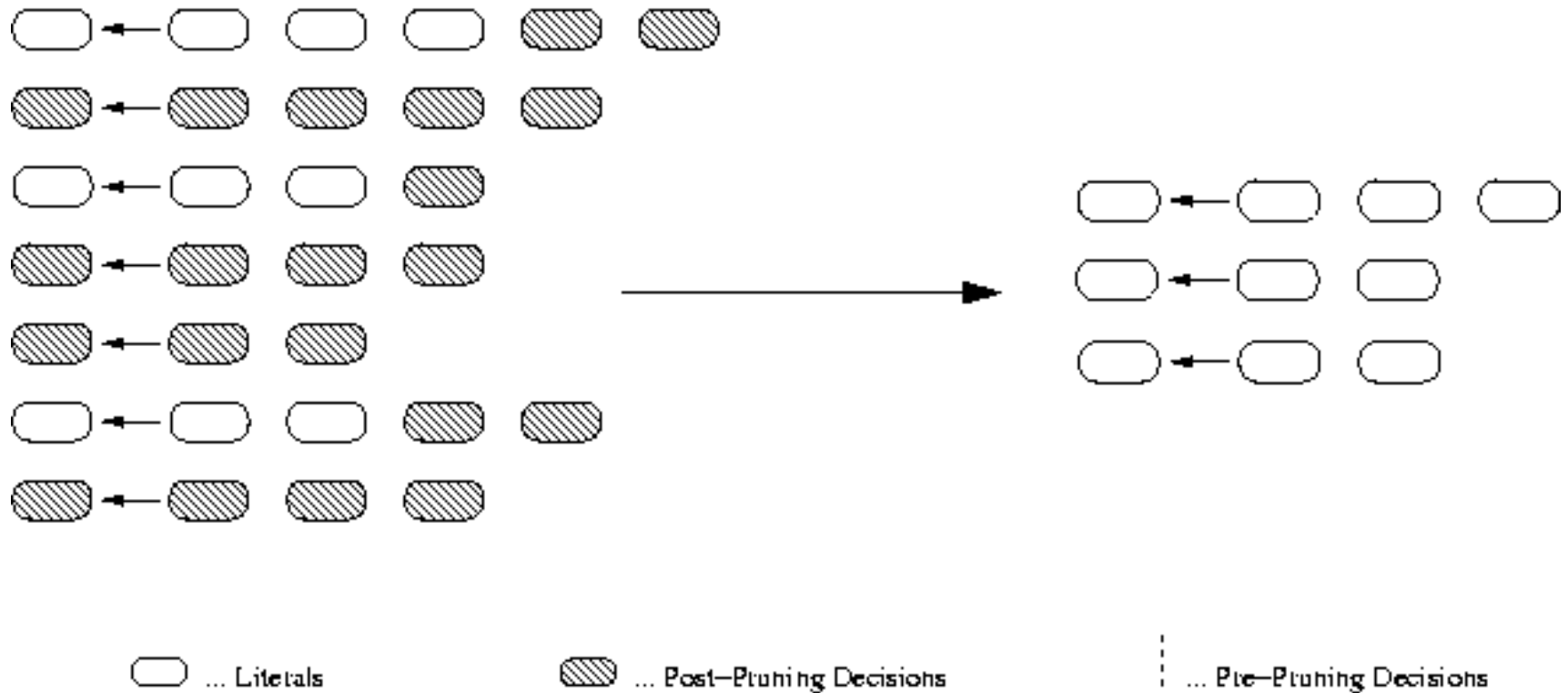
- learning concepts so that
  - not all positive examples have to be covered by the theory
  - some negative examples may be covered by the theory

# Pre-Pruning

- keep a theory simple *while* it is learned
  - decide when to **stop adding conditions** to a rule (*relax consistency* constraint)
  - decide when to **stop adding rules** to a theory (*relax completeness* constraint)
- efficient but not accurate



# Post Pruning



# Post-Pruning: Example

IF	T=hot	AND	H=high	AND	O=sunny	AND	W=false	THEN	no
IF	T=hot	AND	H=high	AND	O=sunny	AND	W=true	THEN	no
IF	T=hot	AND	H=high	AND	O=overcast	AND	W=false	THEN	yes
IF	T=cool	AND	H=normal	AND	O=rain	AND	W=false	THEN	yes
IF	T=cool	AND	H=normal	AND	O=overcast	AND	W=true	THEN	yes
IF	T=mild	AND	H=high	AND	O=sunny	AND	W=false	THEN	no
IF	T=cool	AND	H=normal	AND	O=sunny	AND	W=false	THEN	yes
IF	T=mild	AND	H=normal	AND	O=rain	AND	W=false	THEN	yes
IF	T=mild	AND	H=normal	AND	O=sunny	AND	W=true	THEN	yes
IF	T=mild	AND	H=high	AND	O=overcast	AND	W=true	THEN	yes
IF	T=hot	AND	H=normal	AND	O=overcast	AND	W=false	THEN	yes
IF	T=mild	AND	H=high	AND	O=rain	AND	W=true	THEN	no
IF	T=cool	AND	H=normal	AND	O=rain	AND	W=true	THEN	no
IF	T=mild	AND	H=high	AND	O=rain	AND	W=false	THEN	yes



# Post-Pruning: Example

IF H=high AND O=sunny THEN no

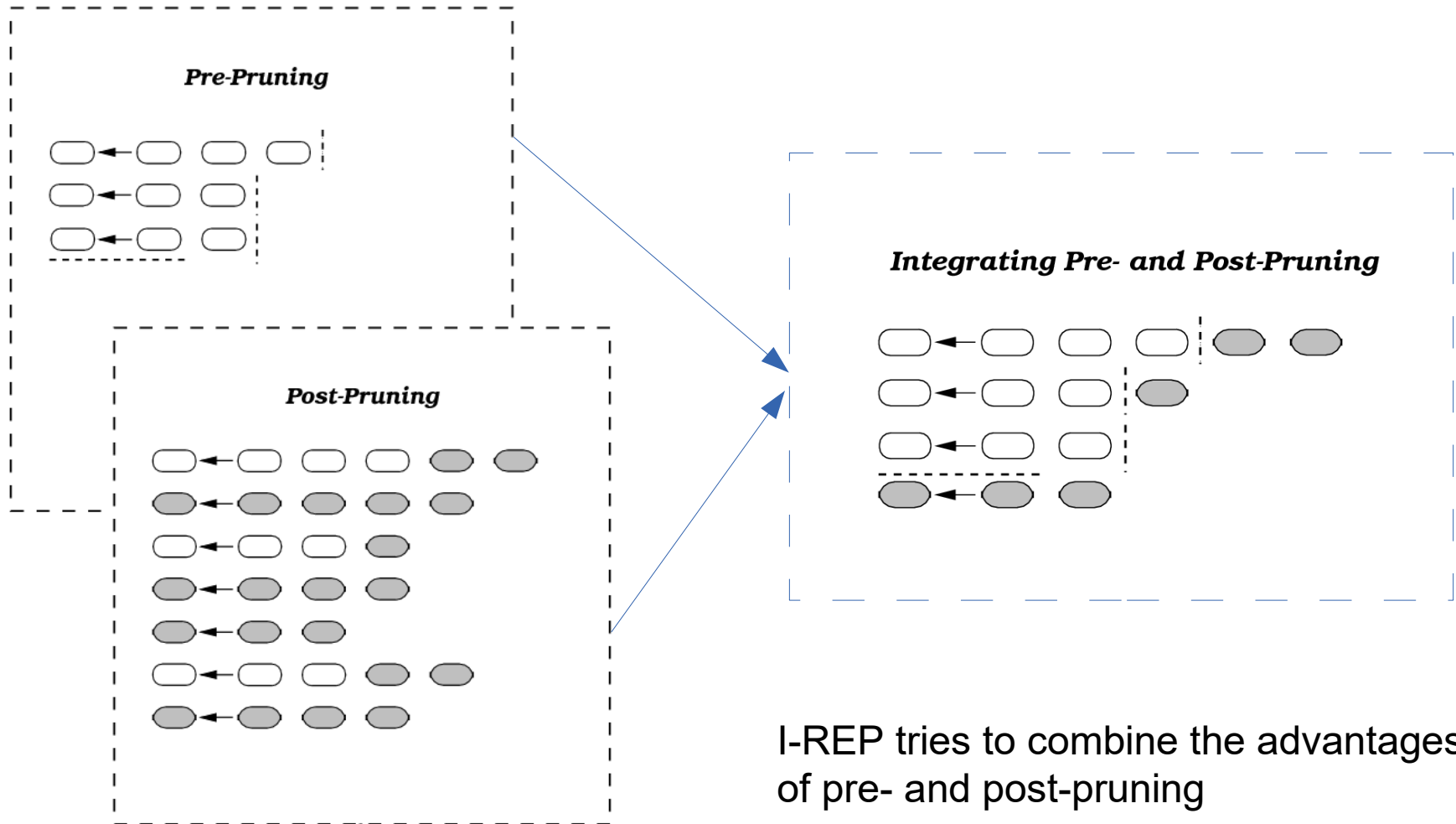
IF O=rain AND W=true THEN no

ELSE yes

# Reduced Error Pruning

- basic idea
    - optimize the accuracy of a rule set on a separate pruning set
1. split training data into a growing and a pruning set
  2. learn a complete and consistent rule set covering all positive examples and no negative examples
  3. as long as the error on the pruning set does not increase
    - delete condition or rule that results in the largest reduction of error on the pruning set
  4. return the remaining rules
- REP is accurate but not efficient
    - $O(n^4)$

# Incremental Reduced Error Pruning

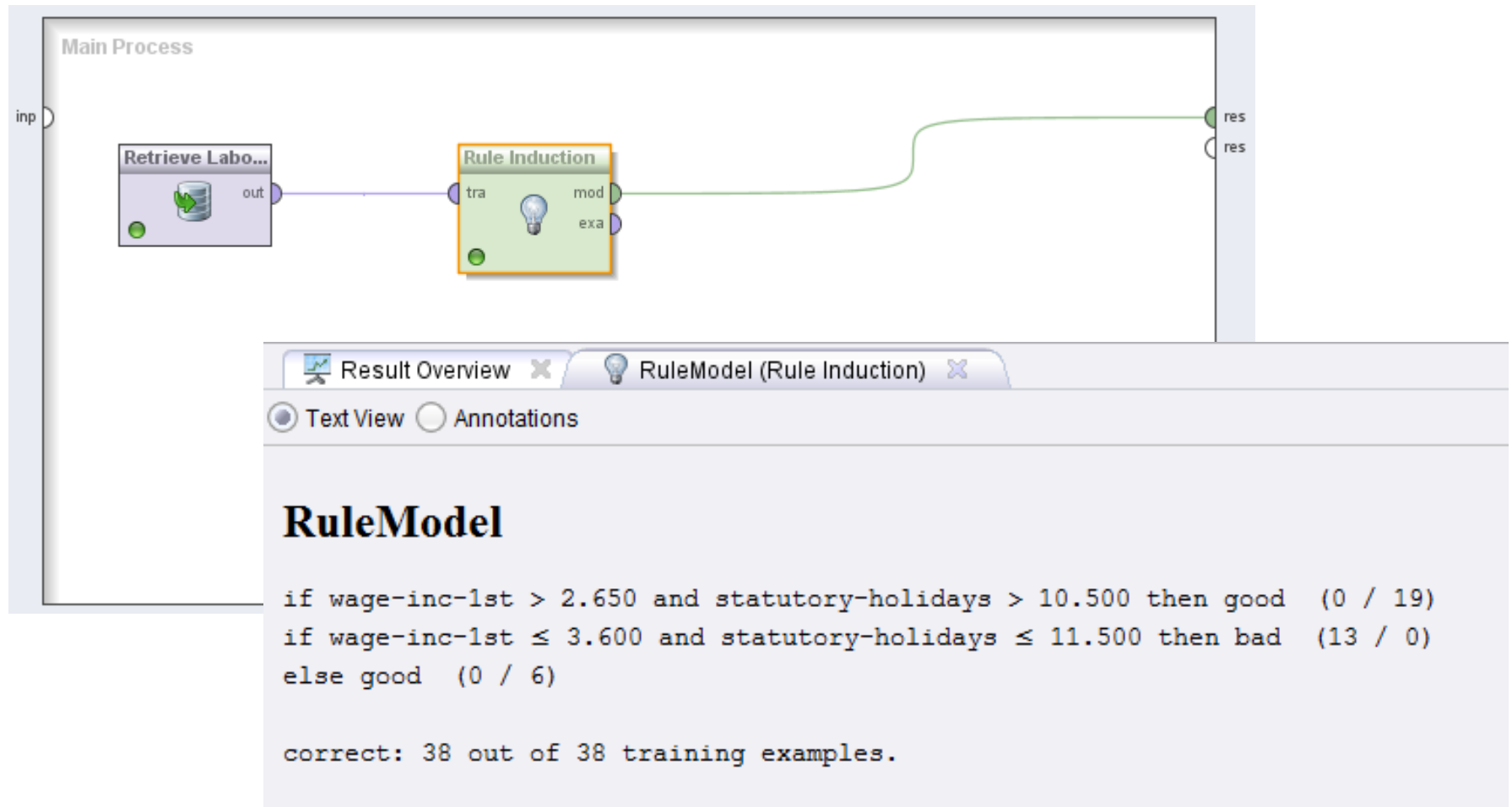


I-REP tries to combine the advantages of pre- and post-pruning

# Incremental Reduced Error Pruning

- Prune each rule right after it is learned:
  1. split training data into a growing and a pruning set
  2. learn a consistent rule covering only positive examples
  3. delete conditions as long as the error on the pruning set does not increase
  4. if the rule is better than the default rule
    - add the rule to the rule set
    - goto 1.
- More accurate, much more efficient
  - because it does not learn overly complex intermediate concepts
  - REP:  $O(n^4)$       I-REP:  $O(n \log^2 n)$
- Subsequently used in RIPPER rule learner (Cohen, 1995)

# RIPPER in RapidMiner



# Advantages of Rule-Based Classifiers

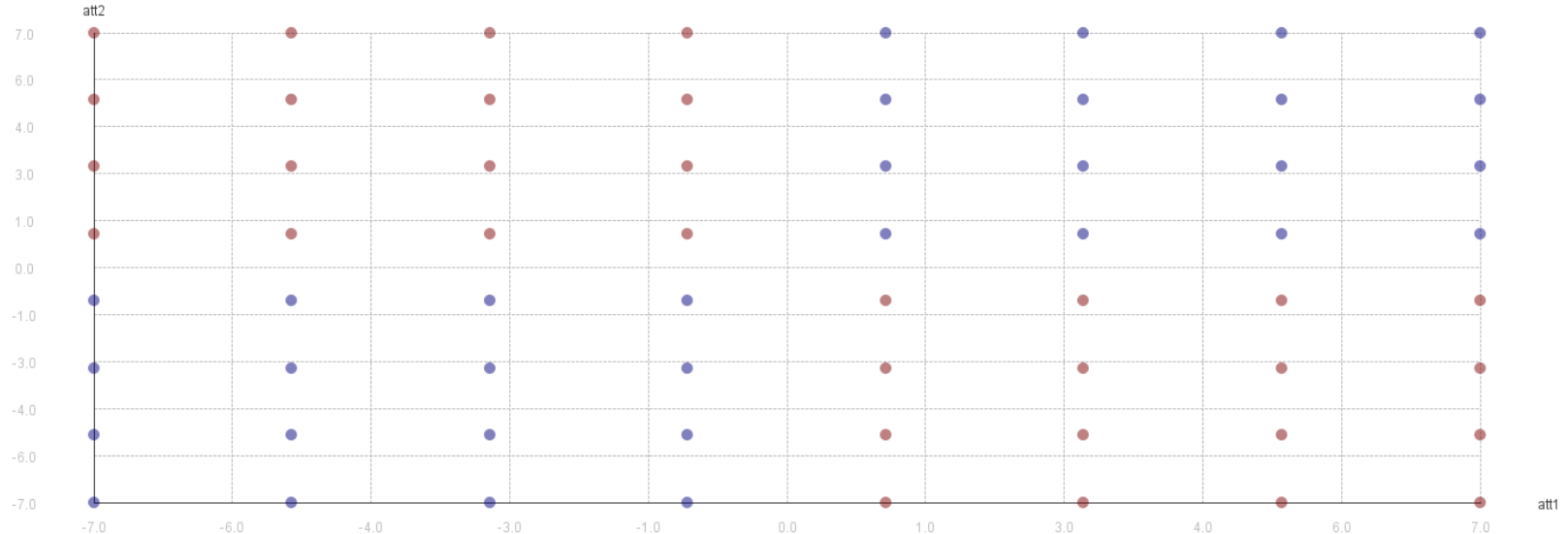
- As highly expressive as decision trees
- Easy to interpret
- Easy to generate
- Can classify new instances rapidly
- Performance comparable to decision trees

# Decision Boundaries: Theory and Practice

- We have seen decision boundaries of rule-based classifiers and decision trees
  - both are parallel to the axes
  - i.e., both can learn models that are a collection of rectangles
- What does that mean for comparing their *performance*
  - if they learn the same sort of models
  - are they equivalent?

# Decision Boundaries: Theory and Practice

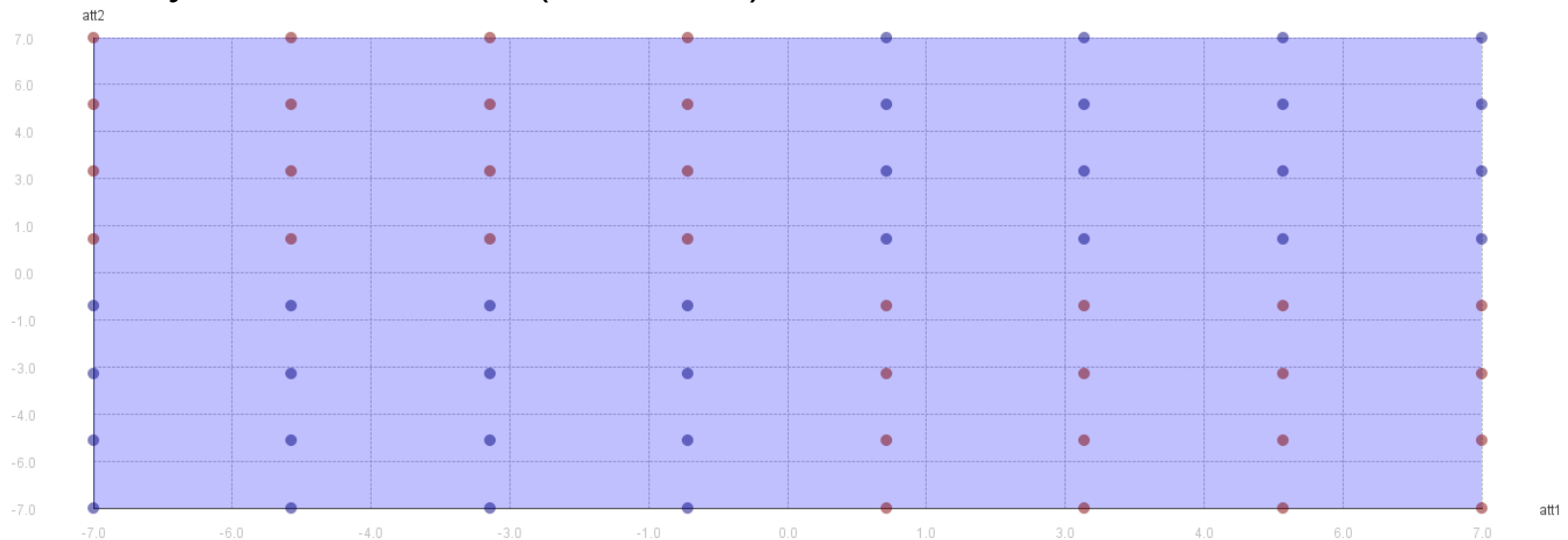
- Example 1: a checkerboard dataset
  - positive and negative points come in four quadrants
  - can be perfectly described with rectangles





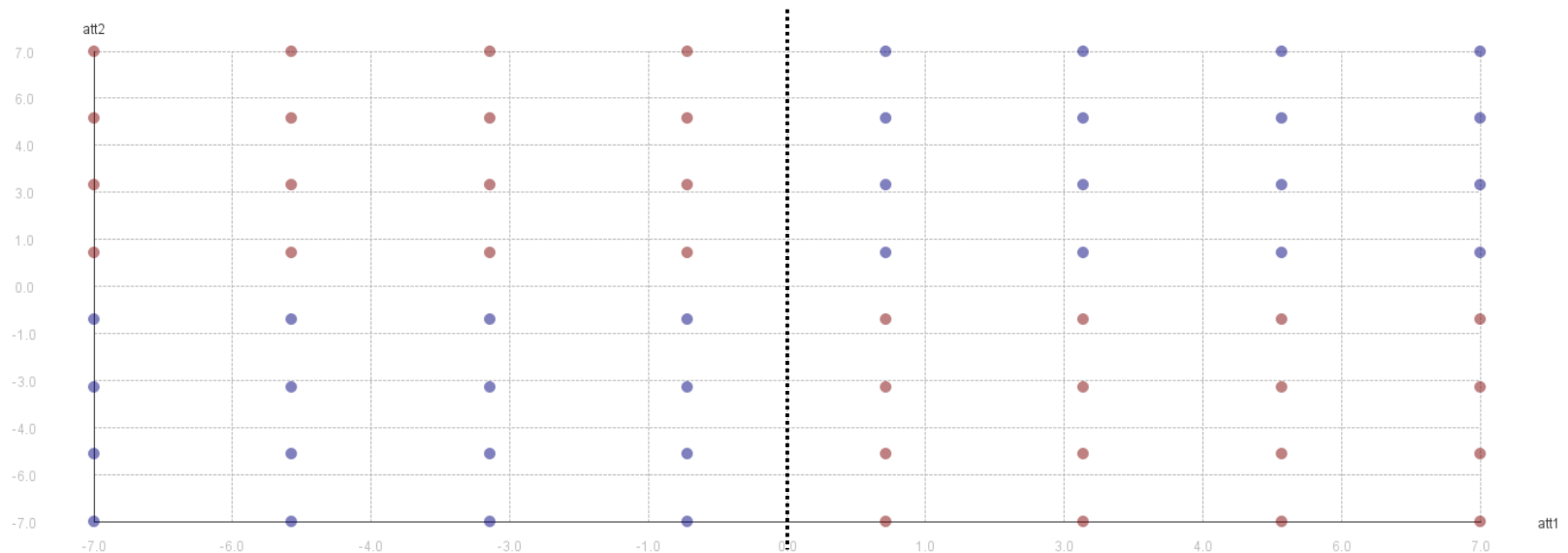
# Decision Boundaries: Theory and Practice

- Example 1: a checkerboard dataset
  - positive and negative points come in quadrants
  - can be perfectly described with rectangles
- Model learned by a decision tree:
  - only the default tree (one node)



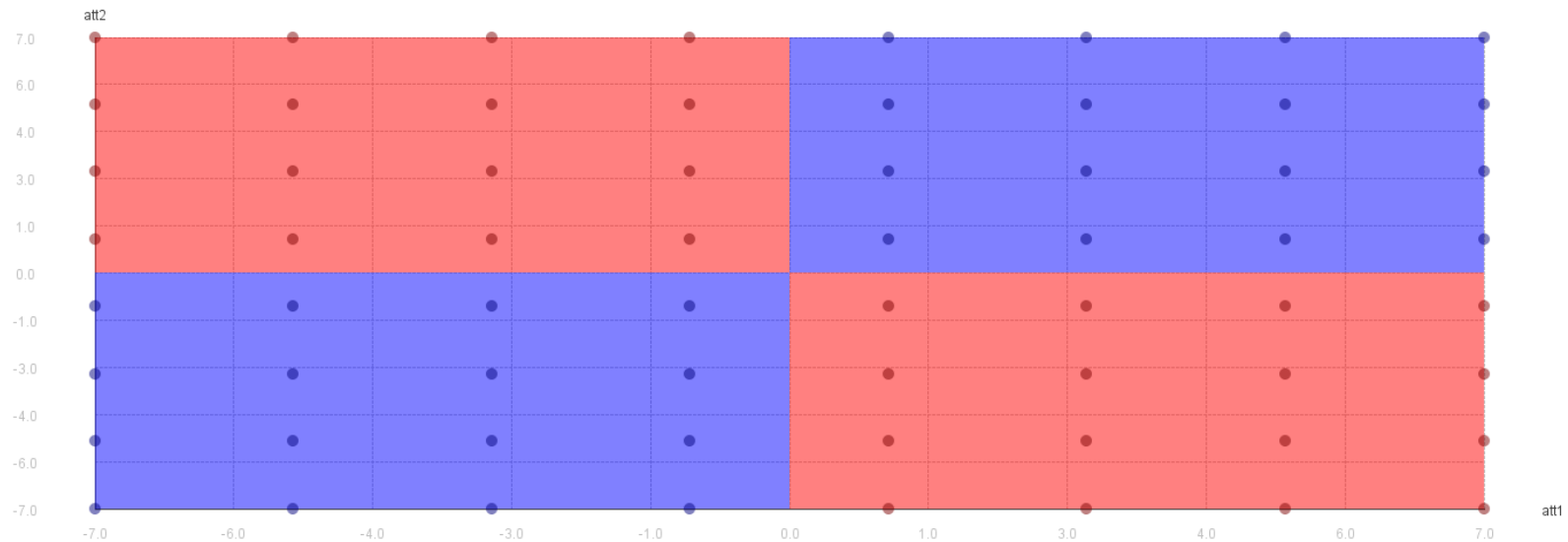
# Decision Boundaries: Theory and Practice

- What is going on here?
  - No possible split improves the purity
  - We always have 50% positive and negative examples
  - i.e., Gini index is always 0.5
    - same holds for other purity measures



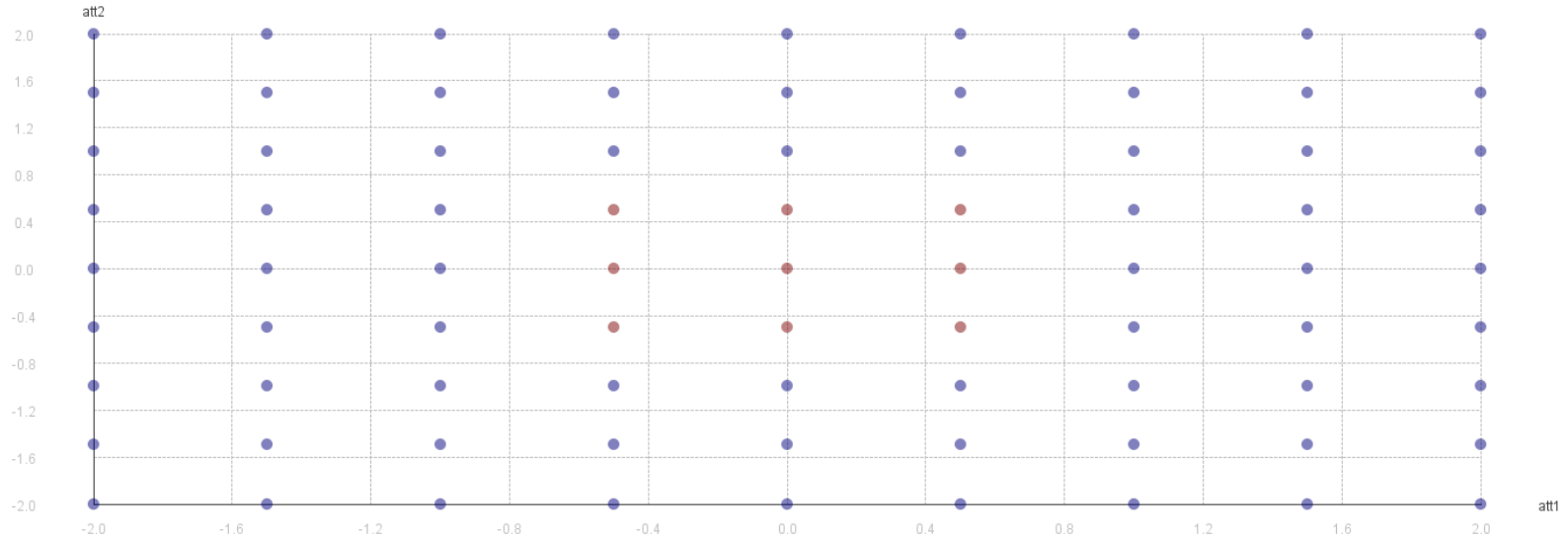
# Decision Boundaries: Theory and Practice

- Example 1: a checkerboard dataset
  - positive and negative points come in quadrants
  - can be perfectly described with rectangles
- Model learned by a rule learner:



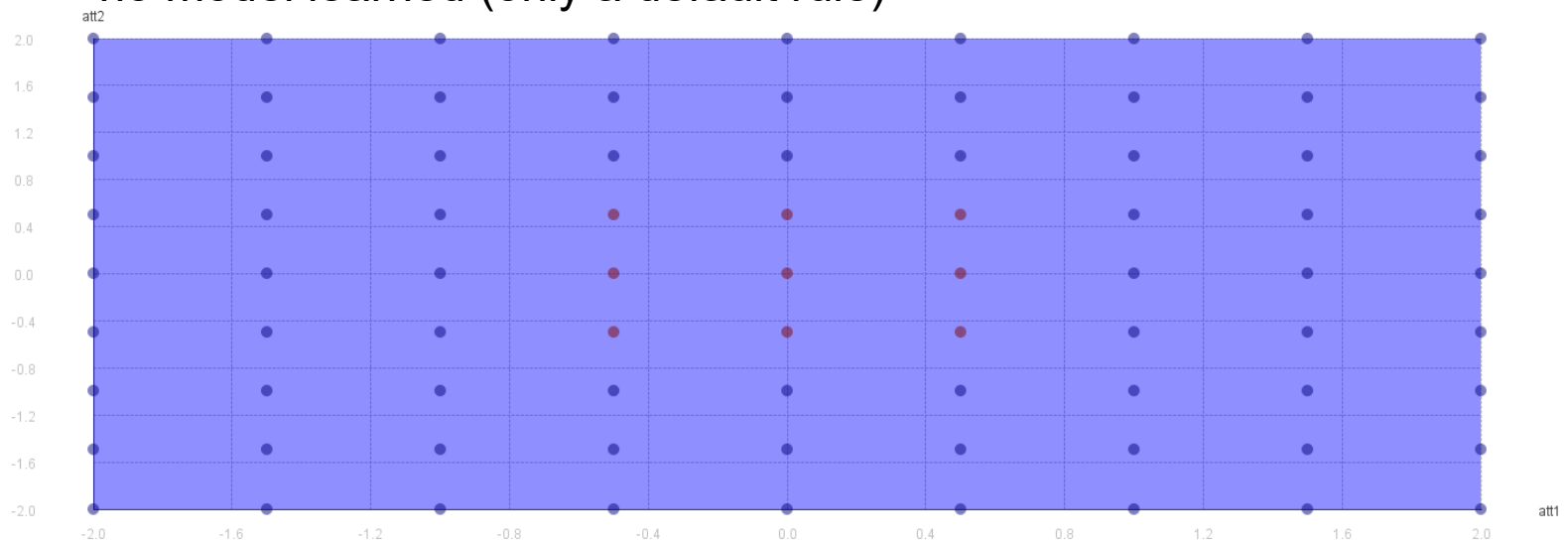
# Decision Boundaries: Theory and Practice

- Example 2:
  - a small class inside a large one
  - again: can be perfectly described with rectangles



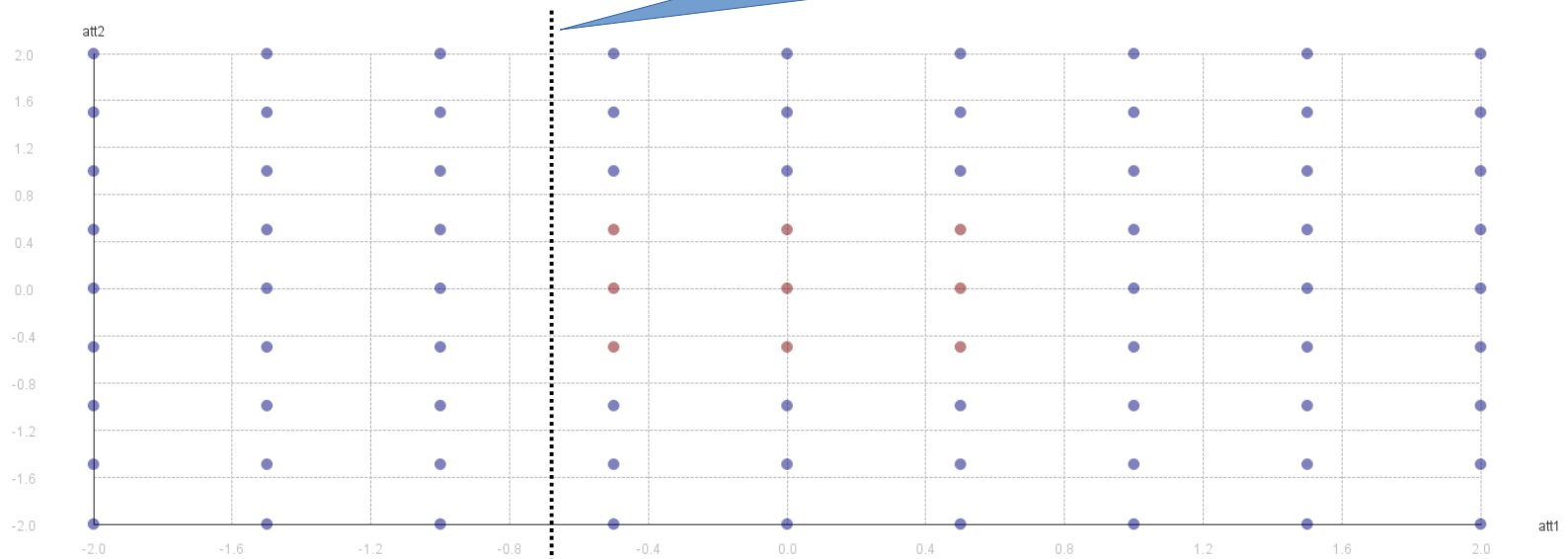
# Decision Boundaries: Theory and Practice

- Example 2:
  - a small class inside a large one
  - again: can be perfectly described with rectangles
- Output of a rule-based classifier:
  - no model learned (only a default rule)



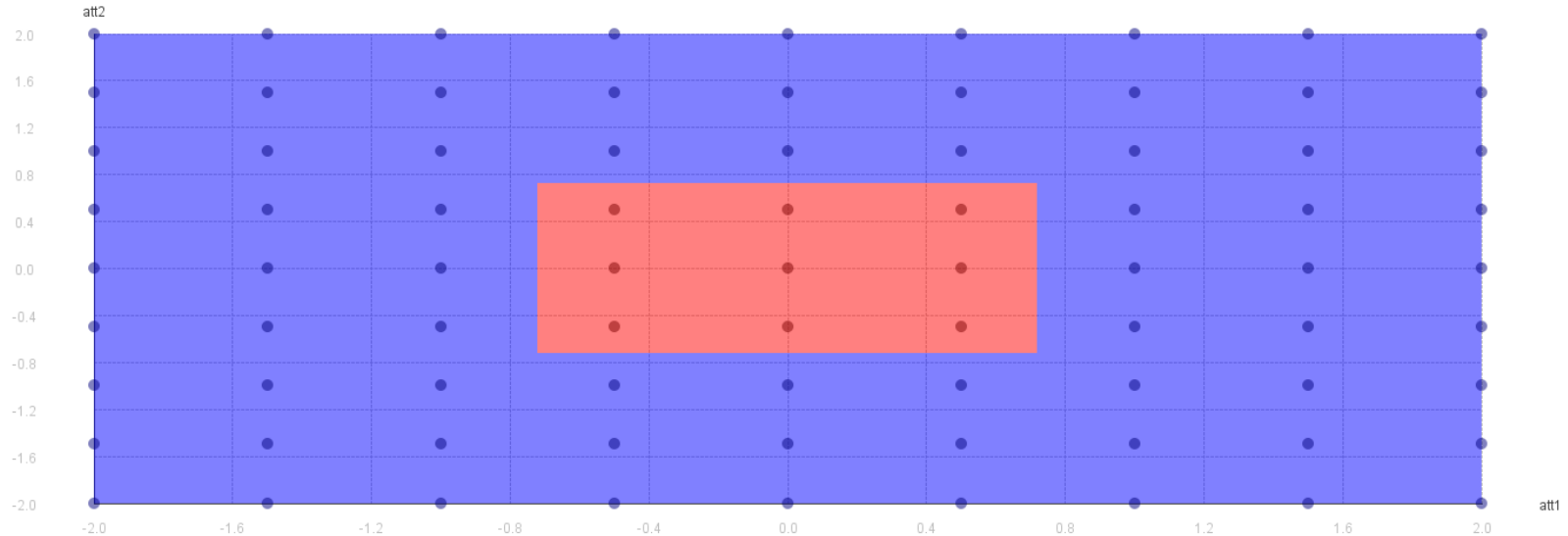
# Decision Boundaries: Theory and Practice

- What is happening here?
  - We try to learn the *smallest* class
  - ...and pre-pruning requires a *minimum* heuristic
  - no initial condition can be selected that exceeds that minimum



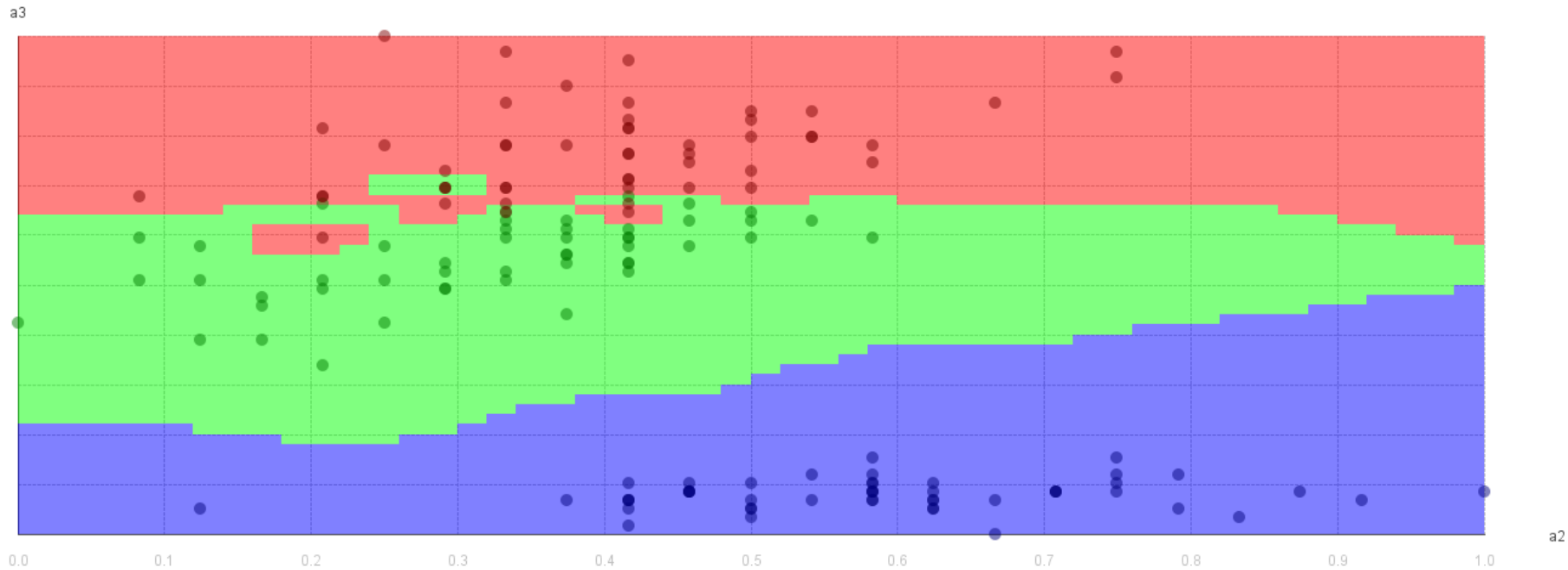
# Decision Boundaries: Theory and Practice

- Example 2:
  - a small class inside a large one
  - again: can be perfectly described with rectangles
- Decision boundaries of a decision tree classifier:



# Decision Boundaries of a k-NN Classifier

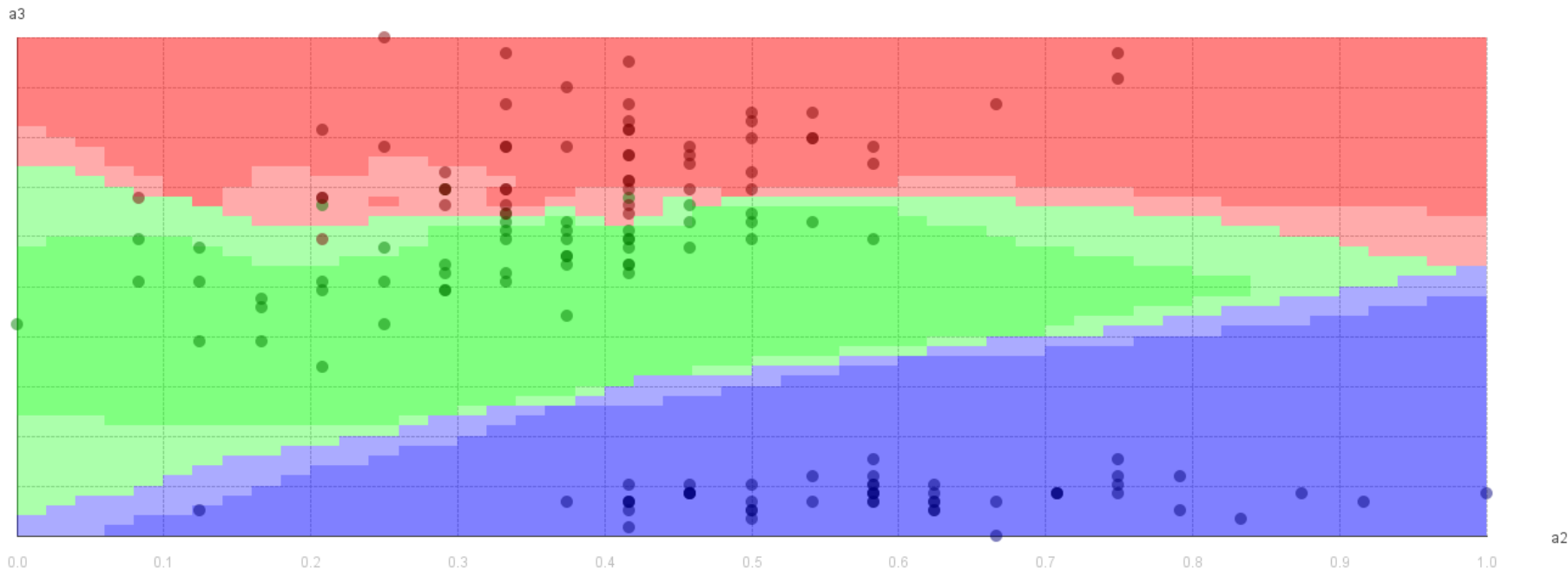
- $k=1$
- Single noise points have influence on model





# Decision Boundaries of a k-NN Classifier

- $k=3$
- Boundaries become smoother
- Influence of noise points is reduced



# Decision Boundaries: Theory and Practice

- Decision boundaries are a useful tool
  - they show us what model *can be* expressed by a learner
  - e.g., circular shapes are not learned by a decision tree learner
  - good for a pre-selection of learners for a problem
- What *can be expressed* and what *is actually learned*
  - are two different stories
  - answer depends heavily on the learning algorithm (and its parameters)
  - requires (and provides) more insights into the learning algorithm

# Model Evaluation

- Metrics
  - how to measure performance?
- Evaluation methods
  - how to obtain meaningful 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:

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

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

ACTUAL CLASS	PREDICTED CLASS	
	Class=Yes	Class=No
	Class=Yes	Class=No
	TP	FN
	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%

# 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.

	Classified Positive	Classified Negative
Actual Positive	TP	FN
Actual Negative	FP	TN

$$p = \frac{TP}{TP + FP} \quad r = \frac{TP}{TP + FN}$$

**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

	Classified Positive	Classified 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<sub>1</sub>-Measure

- It is hard to compare two classifiers using two measures
- F<sub>1</sub>-Score combines precision and recall into one measure

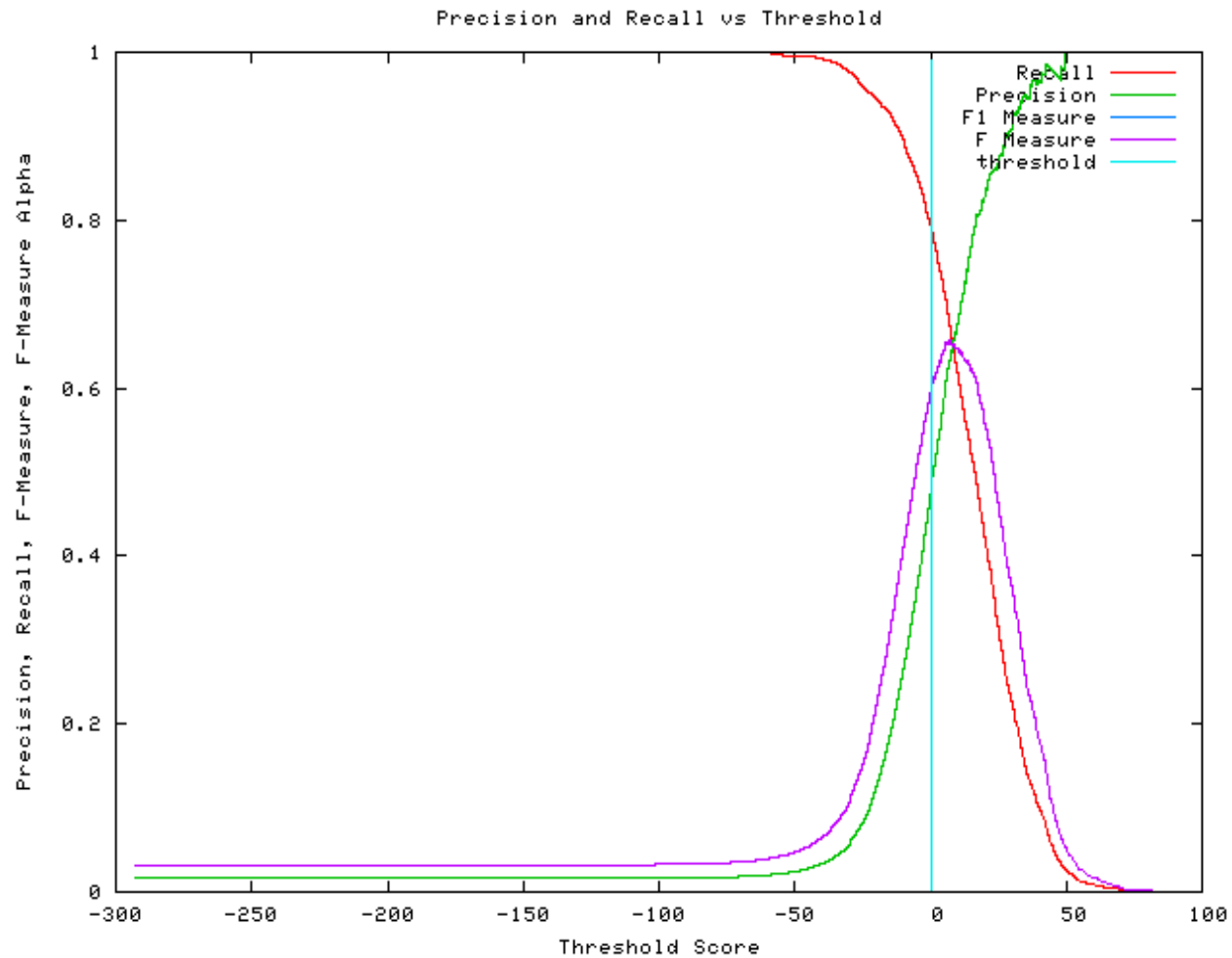
$$F_1 = \frac{2pr}{p+r}$$

F<sub>1</sub>-score is the harmonic mean of precision and recall.

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

- The harmonic mean of two numbers tends to be closer to the smaller of the two.
- For F<sub>1</sub>-value to be large, both  $p$  and  $r$  must be large

# F<sub>1</sub>-Measure



# Alternative for Unbalanced Data: Cost Matrix

	PREDICTED CLASS		
	$C(i j)$	Class=Yes	Class=No
ACTUAL CLASS	Class=Yes	$C(\text{Yes} \text{Yes})$	$C(\text{No} \text{Yes})$
	Class=No	$C(\text{Yes} \text{No})$	$C(\text{No} \text{No})$

$C(i|j)$ : 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

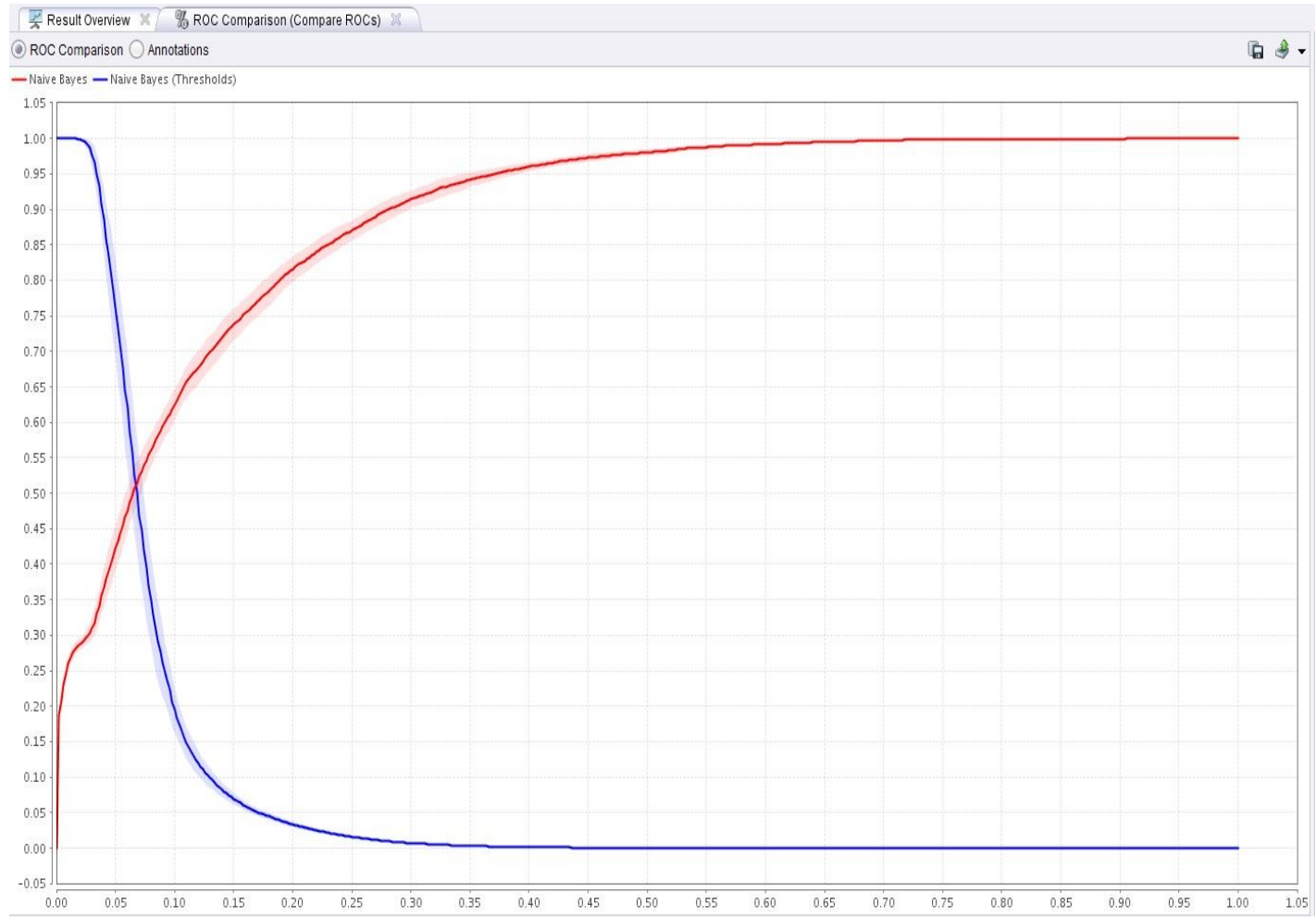
- Drawing ROC Curves in RapidMiner

The screenshot displays the RapidMiner software interface. On the left, the 'Main Process' canvas shows a workflow starting with an 'inp' port, followed by a 'Retrieve Data' node, a 'Set Label' node, and a 'Compare ROCs' node. The 'Compare ROCs' node is highlighted with a yellow border. On the right, the 'Parameters' panel for the 'Compare ROCs' node is visible. It includes the following settings:

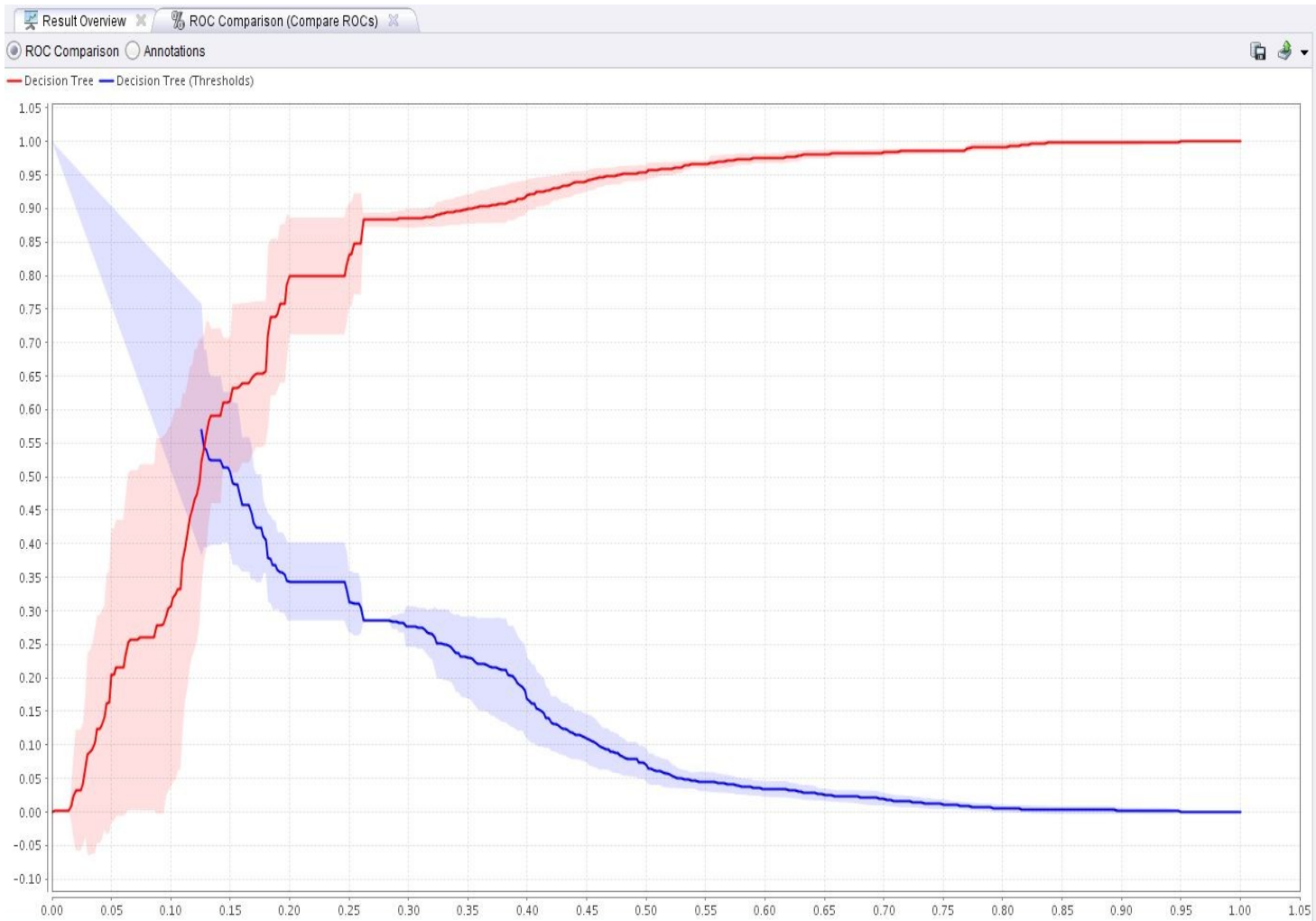
- number of folds: 10
- split ratio: 0.7
- sampling type: stratified sampling
- ☐ use local random seed
- ☒ use example weights
- roc bias: optimistic
- ☐ parallelize model generation



# Example ROC Curve of Naive Bayes

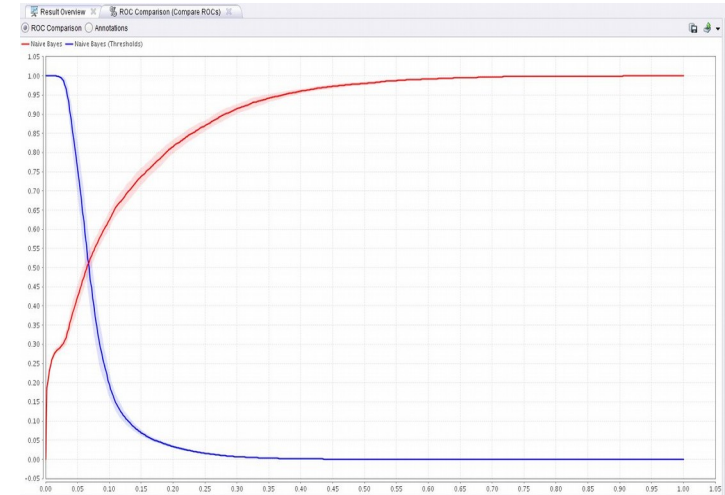


# Example ROC Curve of Decision Tree Learner



# Interpreting ROC Curves

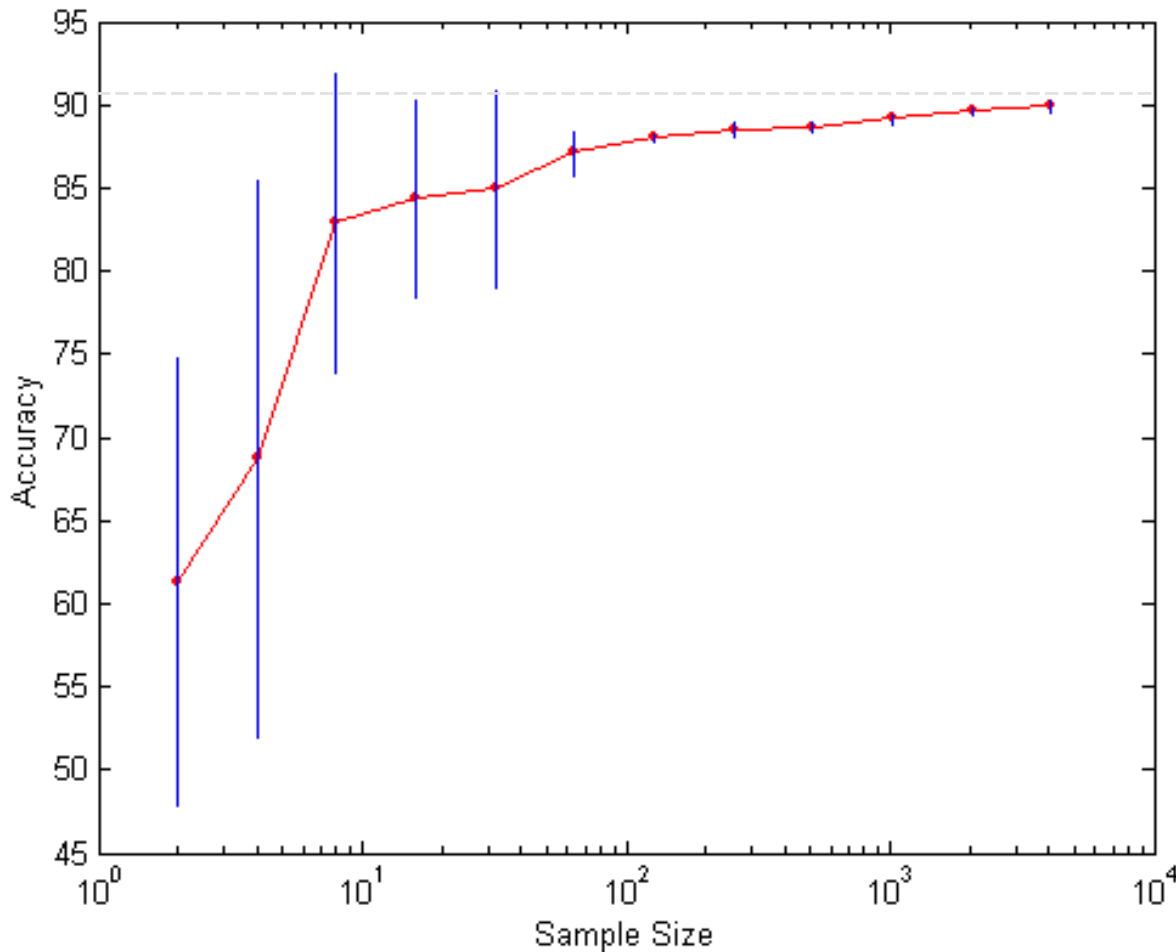
- Best possible result:
  - all correct predictions have higher confidence than all incorrect ones
- The steeper, the better
  - random guessing results in the diagonal
  - so a decent algorithm should result in a curve significantly above the diagonal
- Comparing algorithms:
  - Curve A above curve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

# Learning Curve



- Learning curve shows how accuracy changes with varying sample size
- Conclusion: Use as much data as possible for training



# 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

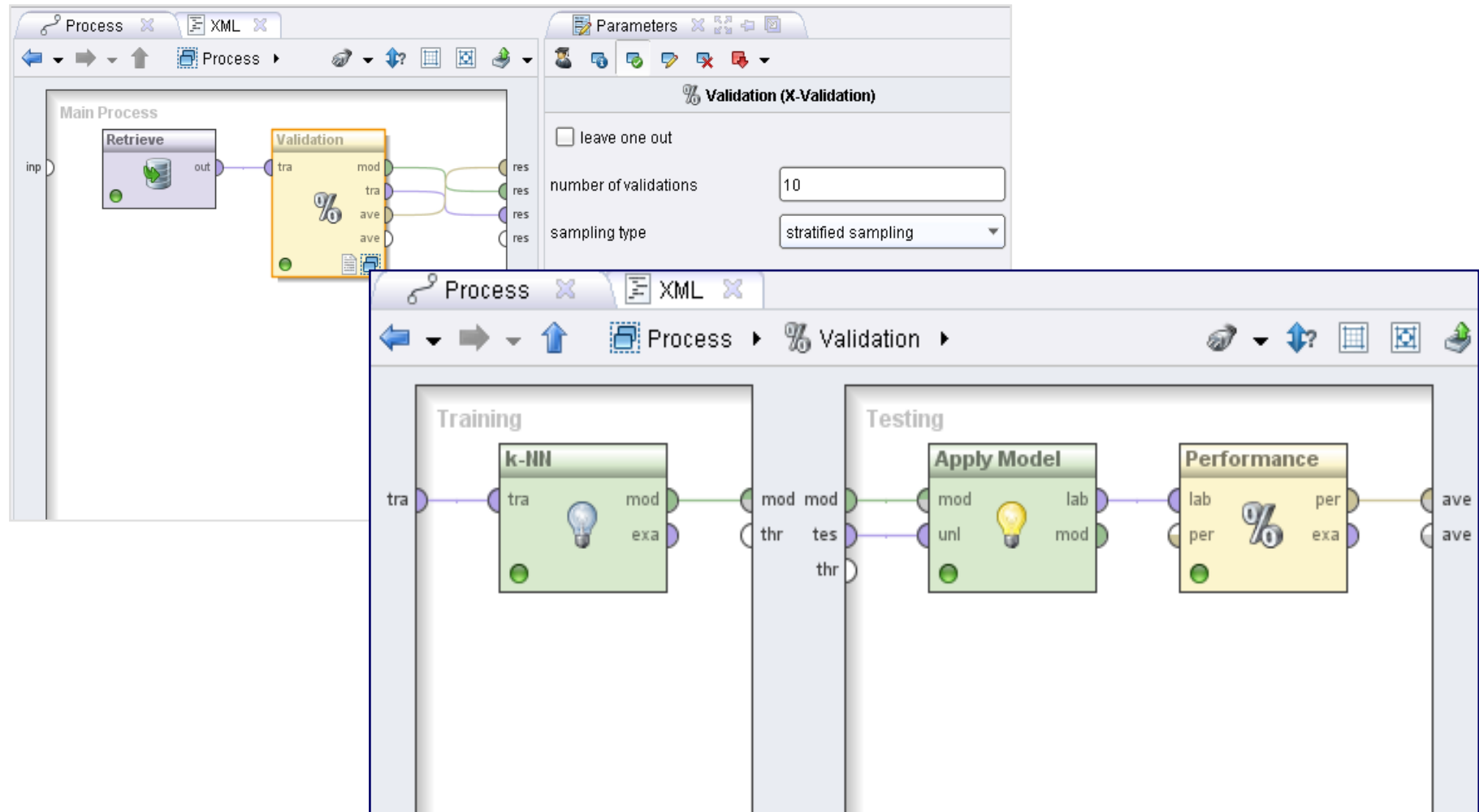
- Iterate over all examples
  - train a model on all examples but the current one
  - evaluate on the current one
- Yields a very accurate estimate
- Uses as much data for training as possible
  - but is computationally infeasible in most cases
- Imagine: a dataset with a million instances
  - one minute 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



# Summary

- Rule learning
  - another eager method to learn *interpretable* models
  - similar to decision trees
  - rectangular decision boundaries
- Evaluation methods
  - Accuracy,
  - Recall, precision, F1
  - ROC Curves
  - Cost based evaluation
  - Evaluation setups: holdout sets, cross validation

# Questions?

