UNIVERSITÄT MANNHEIM

Data Mining I
Classification, Part 3

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

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:
    - (Blood Type=Warm) $\wedge$ (Lay Eggs=Yes) $\rightarrow$ Birds
    - (Taxable Income < 50K) $\wedge$ (Refund=Yes) $\rightarrow$ Evade=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 => Bird

Rule R3 covers grizzly bear => 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%

Tid	Refund	Marital Status	Taxable Income	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**

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

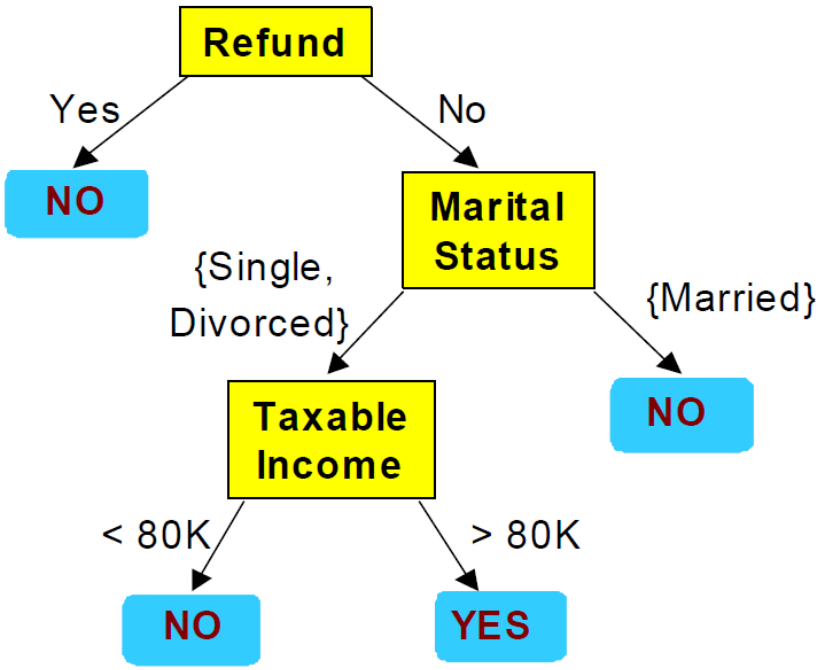
- Mutually exclusive rules
  - Classifier contains mutually exclusive rules if the rules are independent of each other
  - Every example is covered by at most one rule

  → 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

(Refund=Yes) ==> No

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

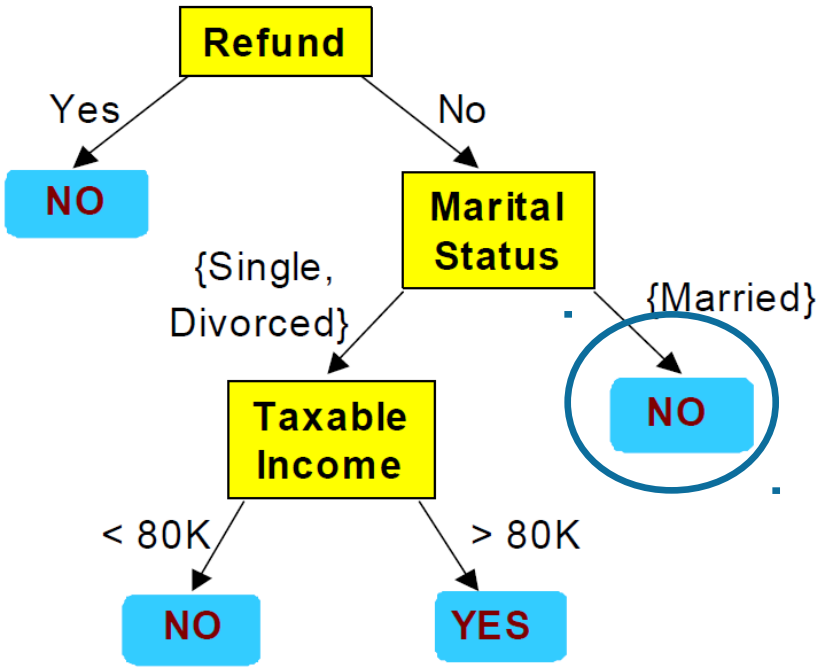
(Refund=No, Marital Status={Single,Divorced}, Taxable Income>80K) ==> Yes

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

Rules are mutually exclusive and exhaustive

Rule set contains as much information as the tree

Rules Can Be Simplified



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

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

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

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

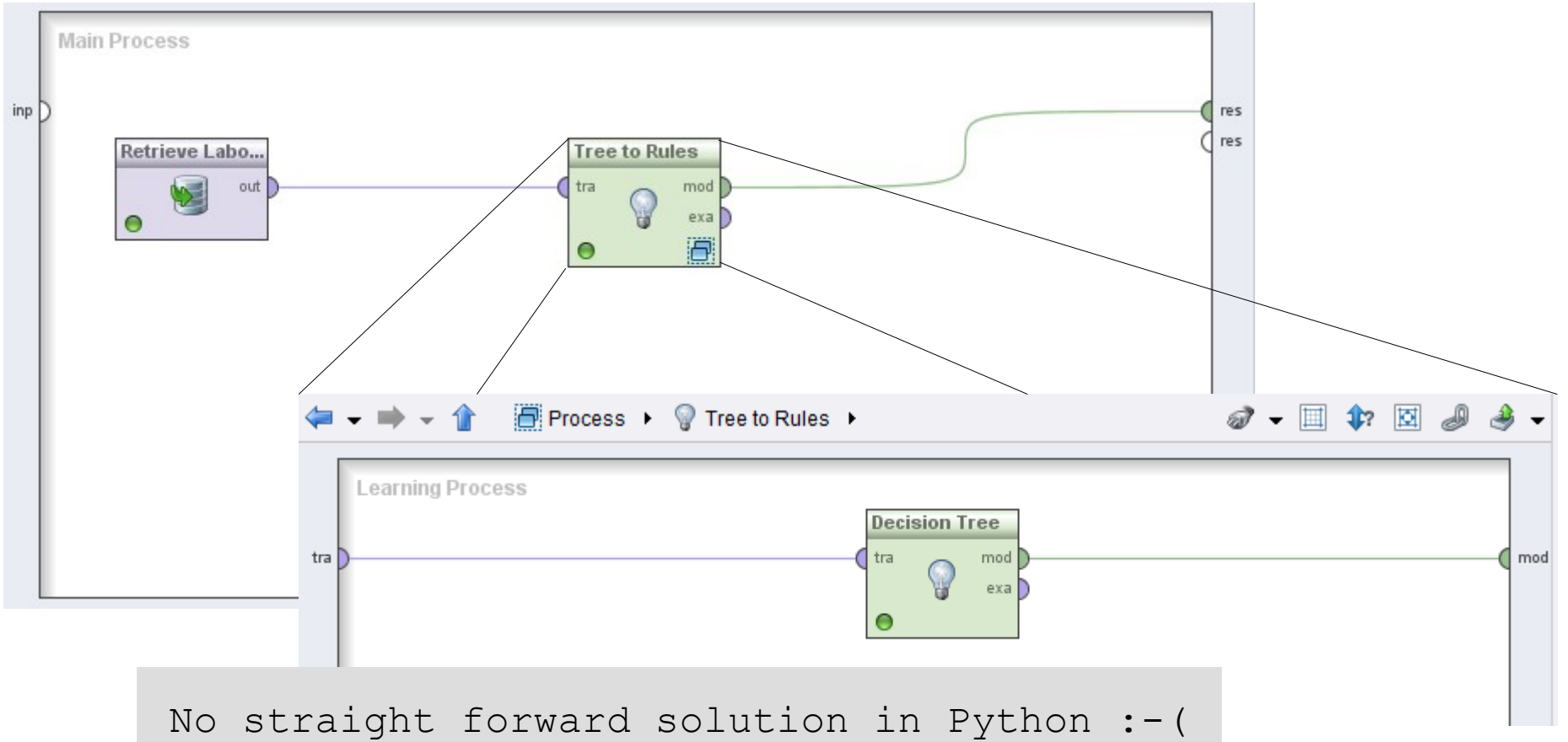
(Refund=No, Marital Status={Married}) ==> No

(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




```
No straight forward solution in Python :-(
```

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

Temperature	Outlook	Humidity	Windy	Play Golf?
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
mild	rain	high	false	yes

- 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

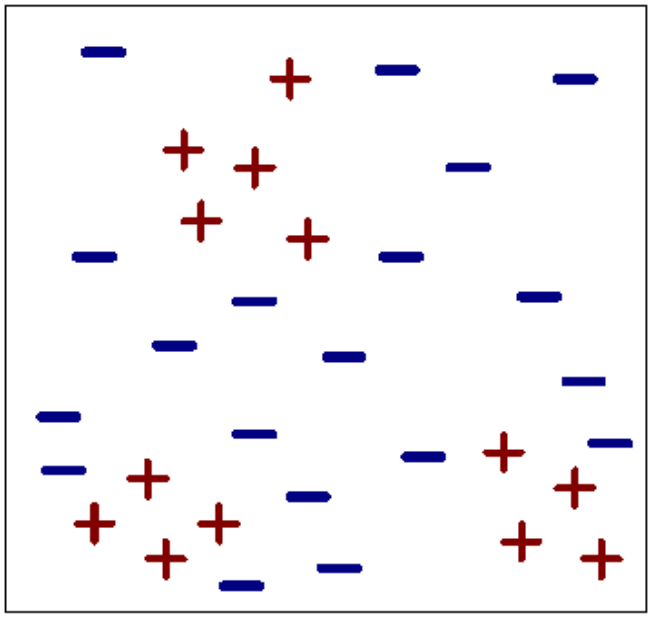
Separate-and-Conquer Rule Learning

- Learn a set of rules, one by one

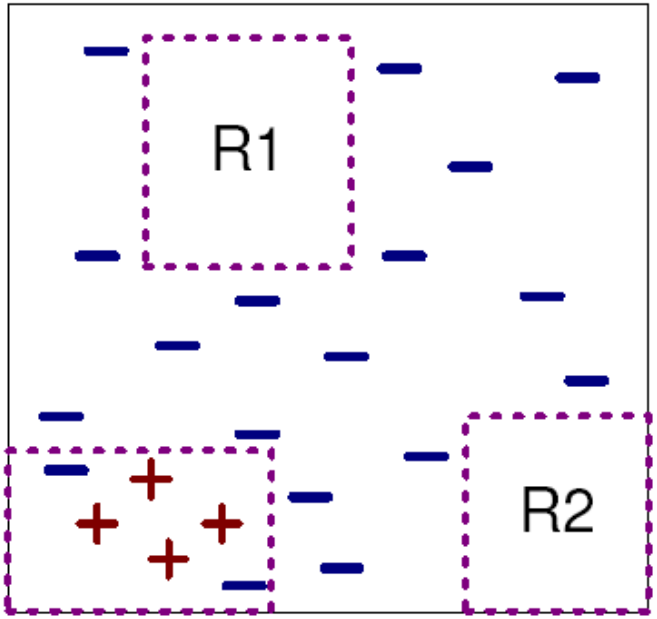
1. Start with an empty theory T and training set E
2. Learn a single (*consistent*) rule R from E and add it to T
3. If T is satisfactory (*complete*), return T
4. Else:
   - » *Separate:* Remove examples explained by R from E
   - » *Conquer:* goto 2.

- 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 incosistent 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 **(true positives)**	P-p **(false negatives)**	P
class -	n **(false positives)**	N-n **(true negatives)**	N
	$p + n$	$P+N-(p+n)$	$P+N$

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 $p \to \infty$
    - grows with $n \to 0$

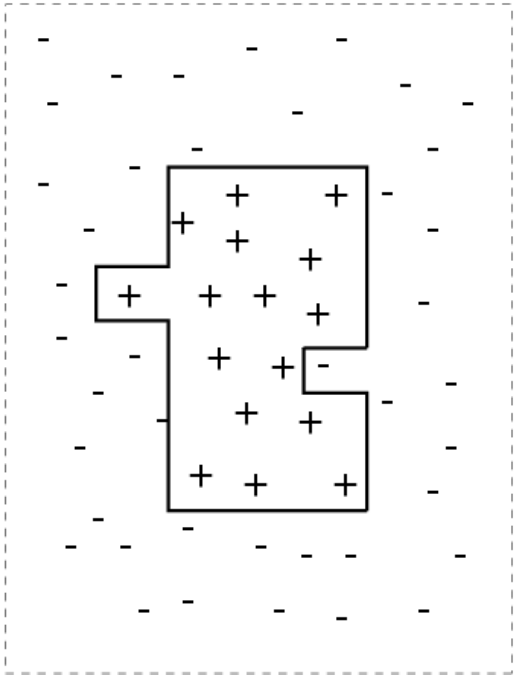
Recap: Overfitting

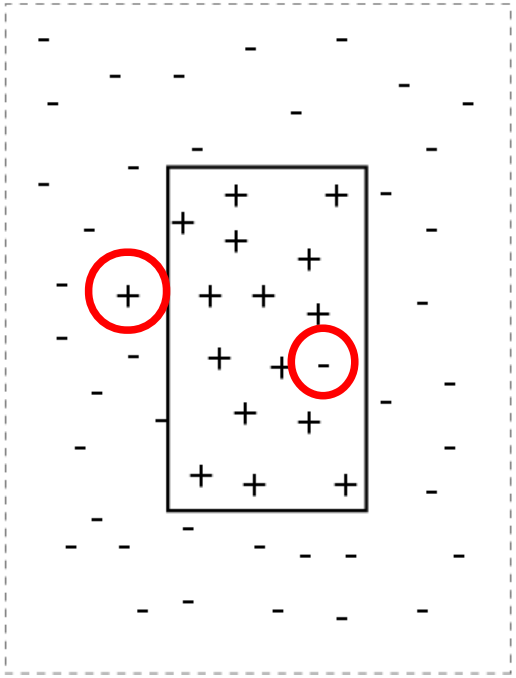
- Overfitting
  - Given
    - a fairly general model class
    - enough degrees of freedom
  - you can always find a model that explains the data
    - even if the data contains errors (noise in the data)
    - in rule learning: each example is a rule

- Such concepts do not generalize well!

  → Solution: Rule and rule set pruning

Overfitting Avoidance

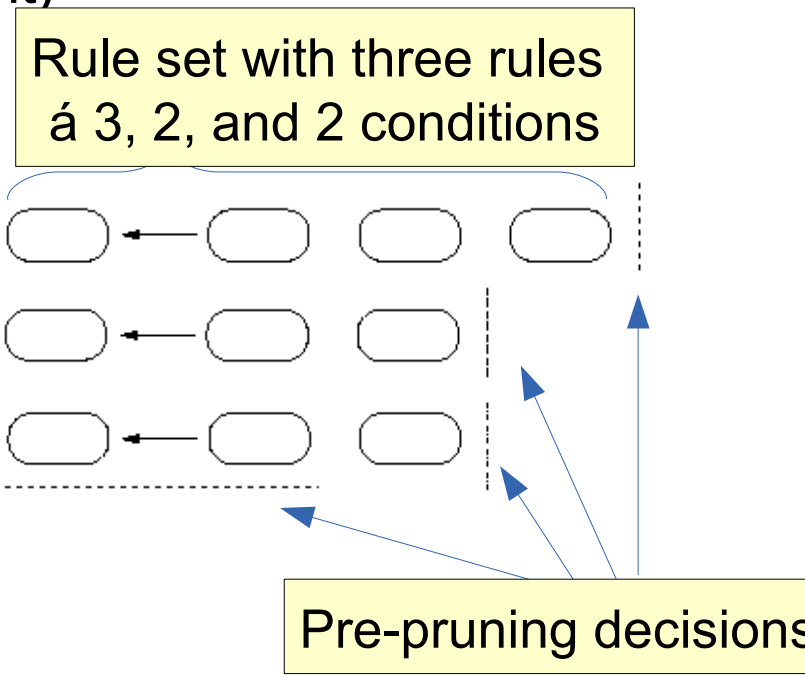




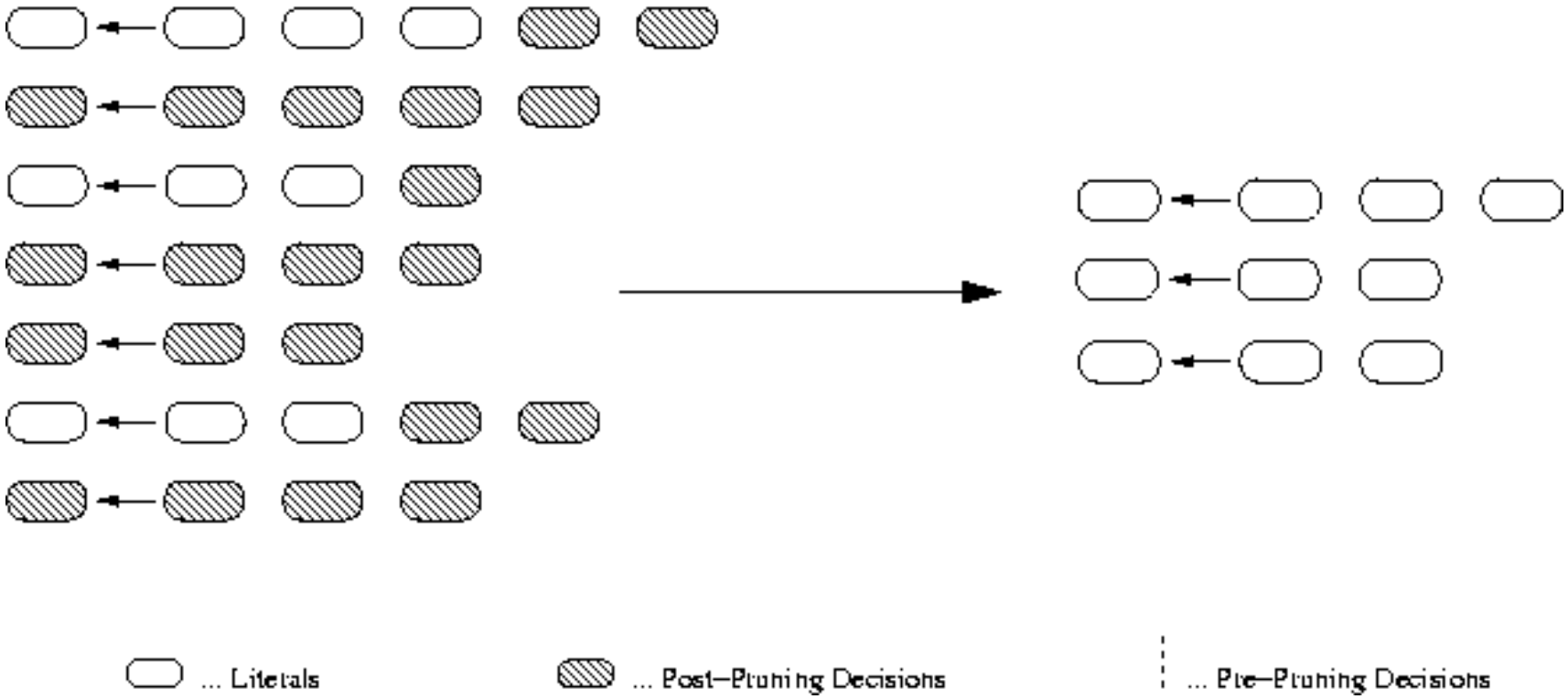
– learning concepts so that
  - not all positive examples have to be covered by the theory
  - some negative examples may be covered by the theory

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

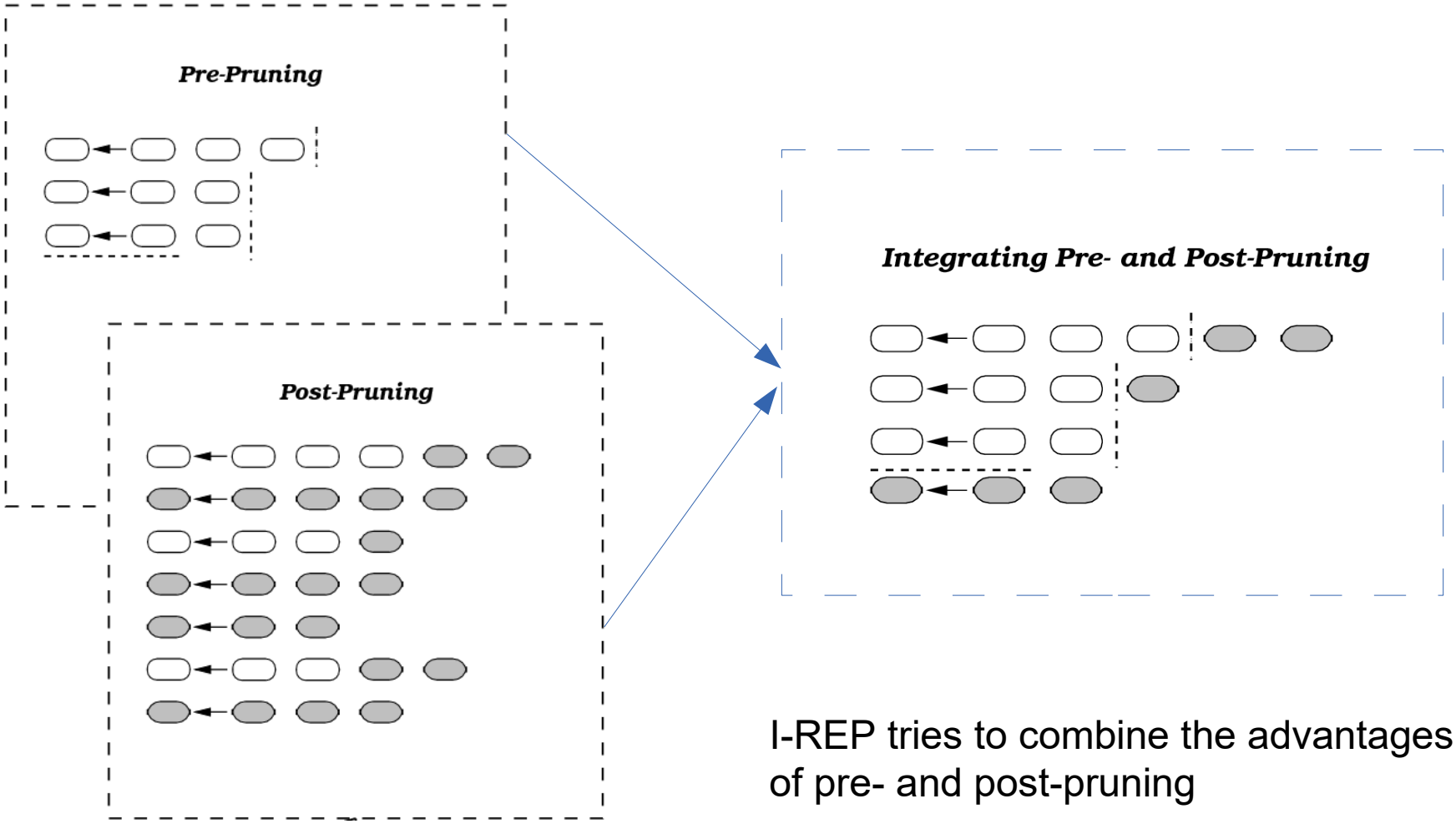
Reduced Error Pruning

- 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

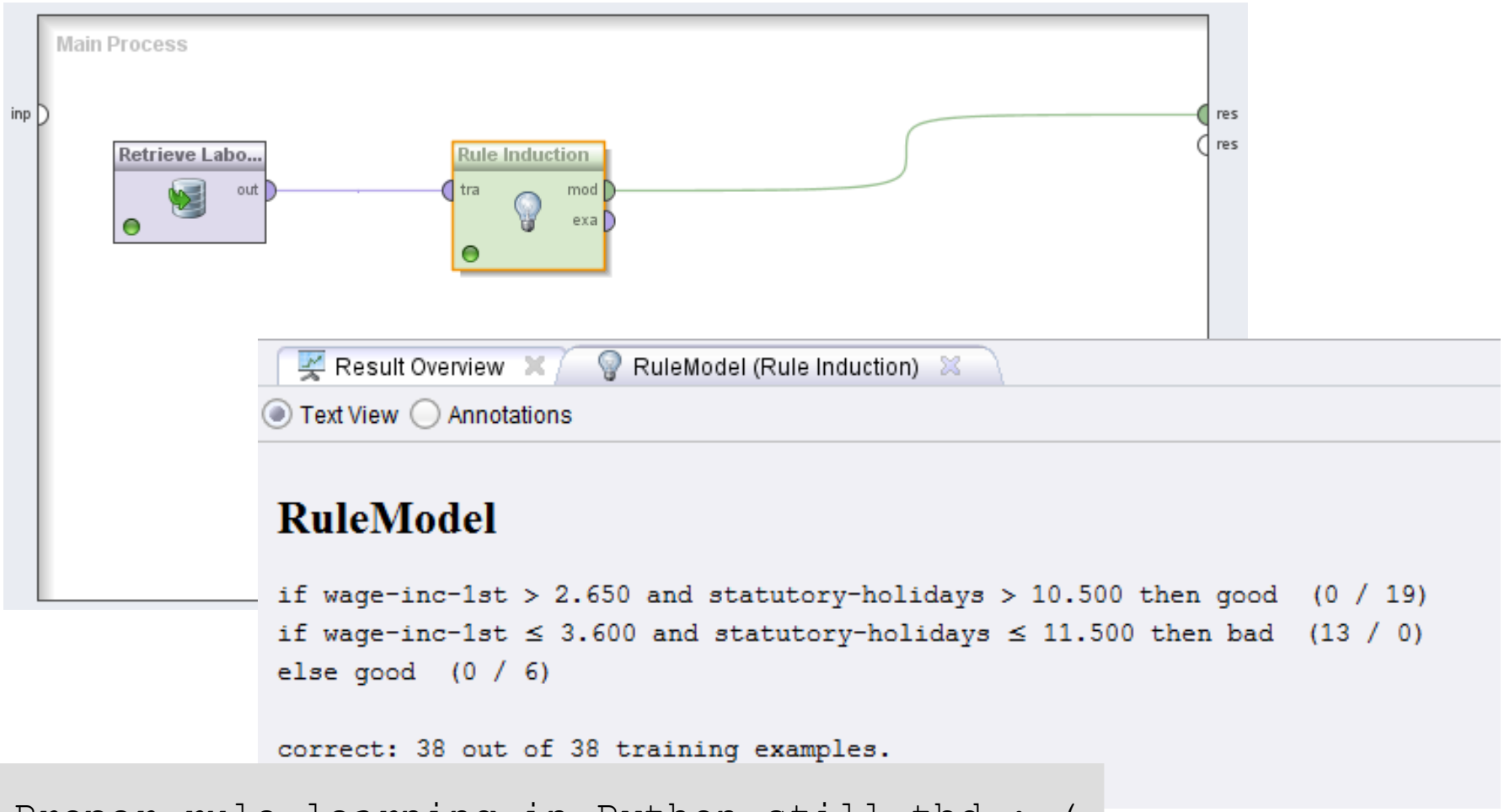
Incremental Reduced Error Pruning

- 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



Proper rule learning in Python still tbd :-(

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

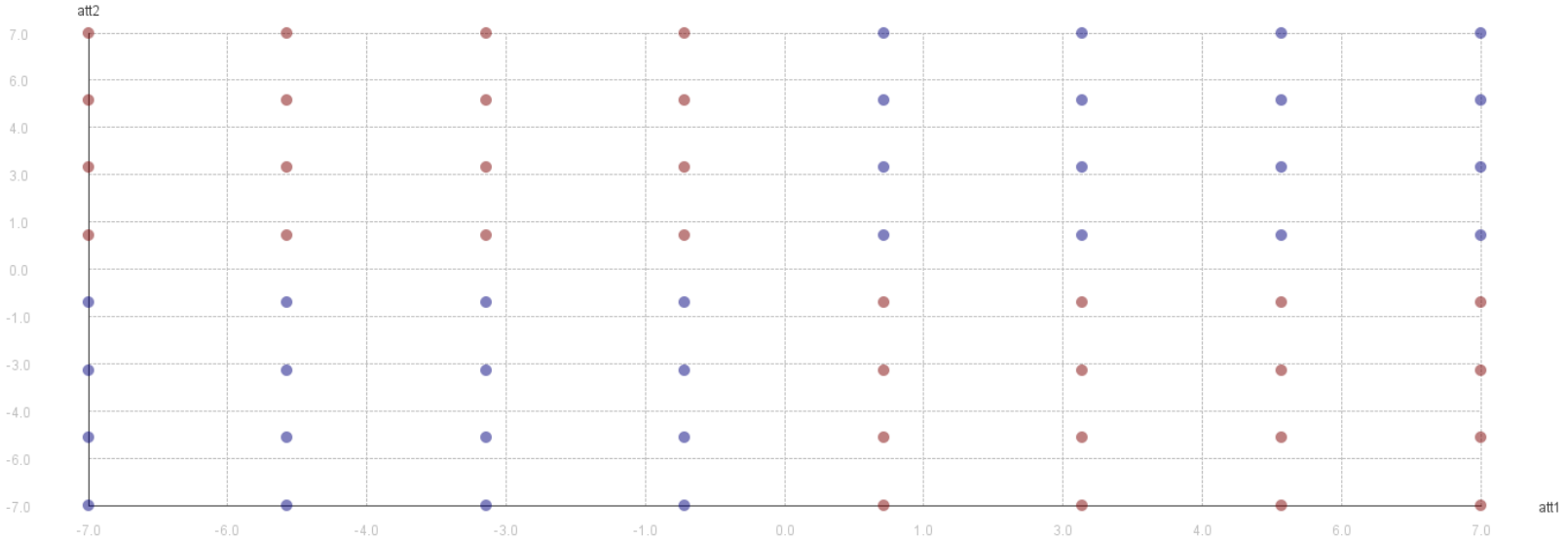
Decision Boundaries: Theory and Practice

- 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

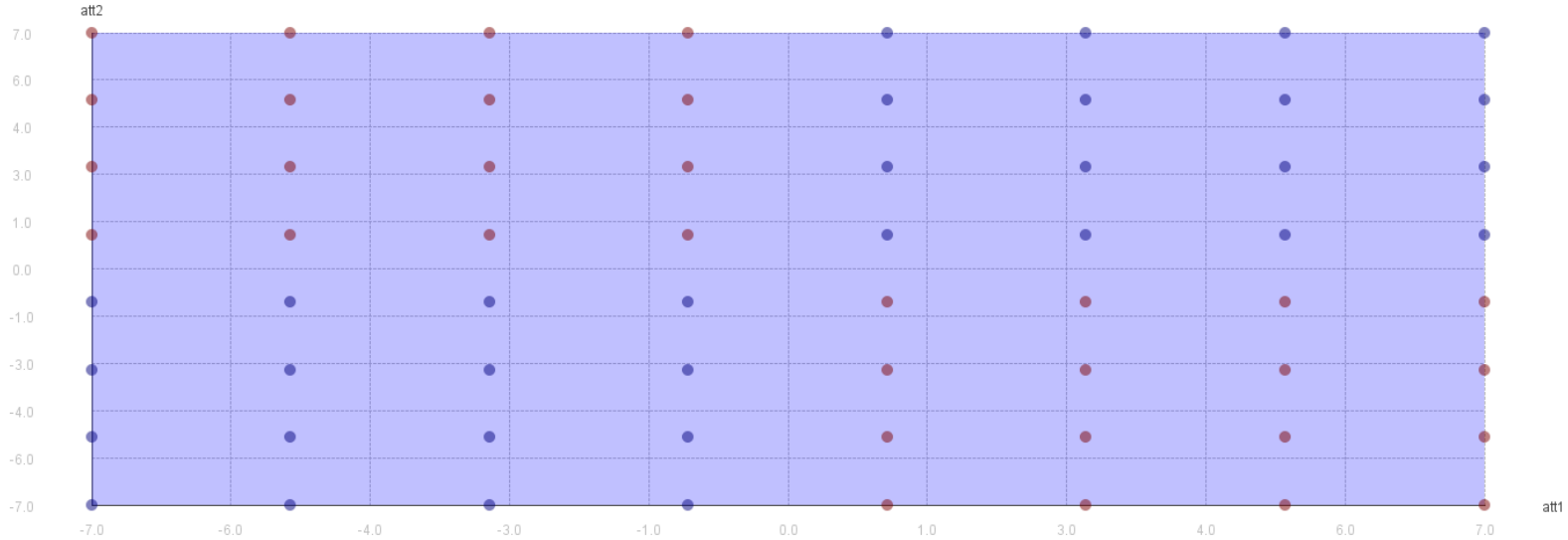
- Example 1: a checkerboard dataset
  - positive and negative points come in four quadrants
  - can be perfectly described with rectangles



Decision Boundaries: Theory and Practice

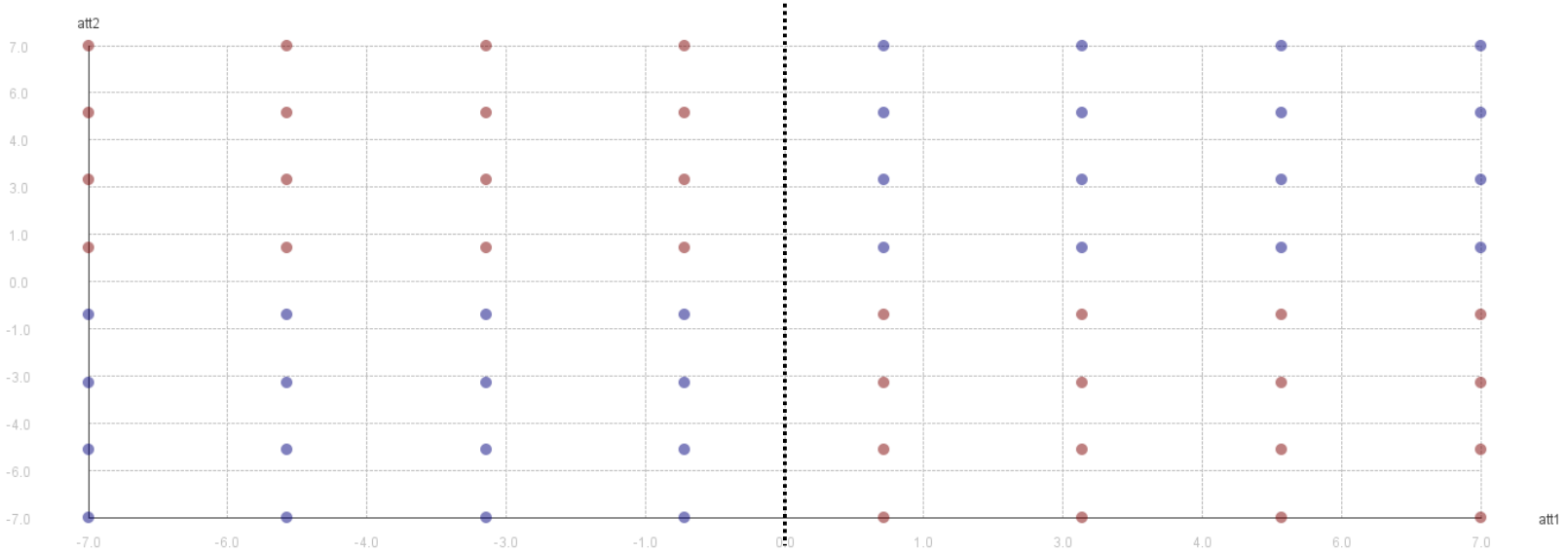
- Example 1: a checkerboard dataset
  - positive and negative points come in quadrants
  - can be perfectly described with rectangles

- Model learned by a decision tree:
  - only the default tree (one node)



Decision Boundaries: Theory and Practice

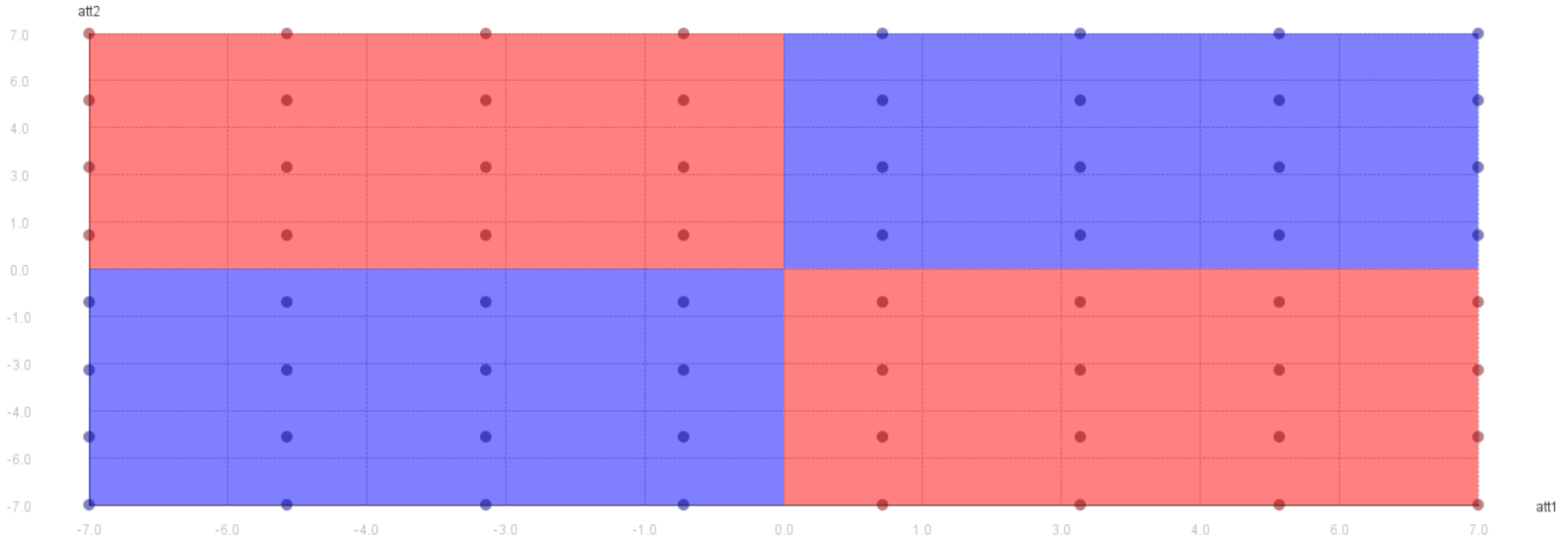
- What is going on here?
  - No possible split improves the purity
  - We always have 50% positive and negative examples
  - i.e., Gini index is always 0.5
    - same holds for other purity measures



Decision Boundaries: Theory and Practice

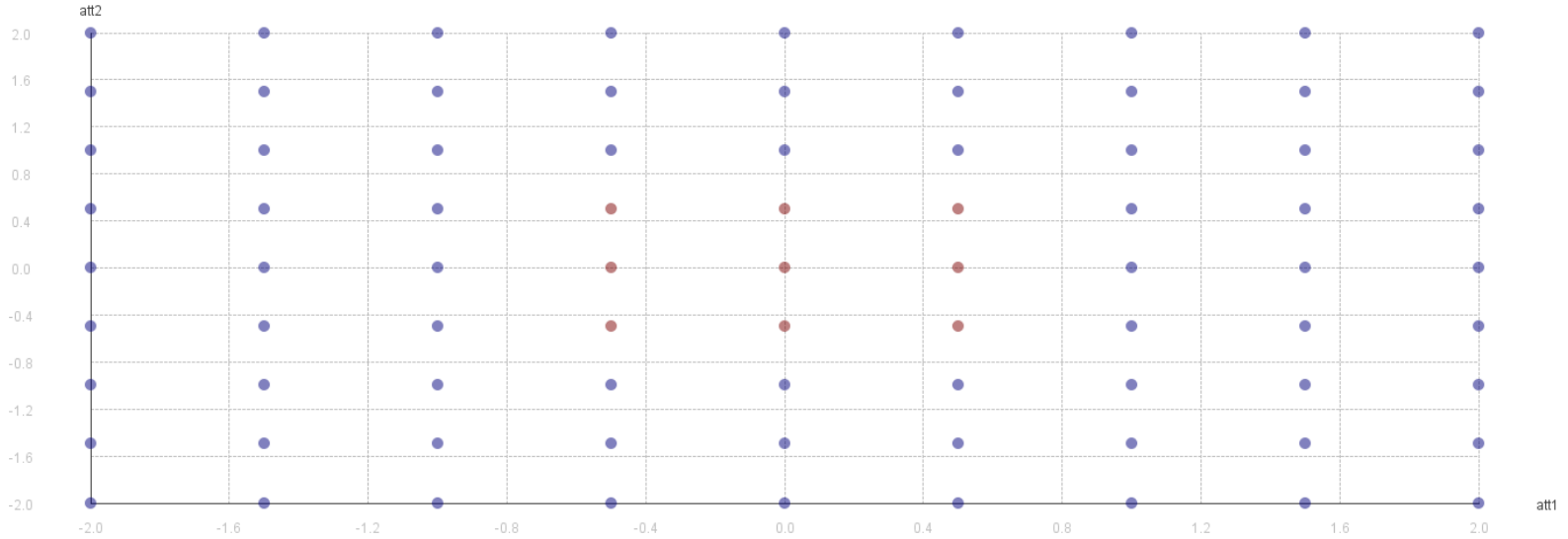
- Example 1: a checkerboard dataset
  - positive and negative points come in quadrants
  - can be perfectly described with rectangles

- Model learned by a rule learner:



Decision Boundaries: Theory and Practice

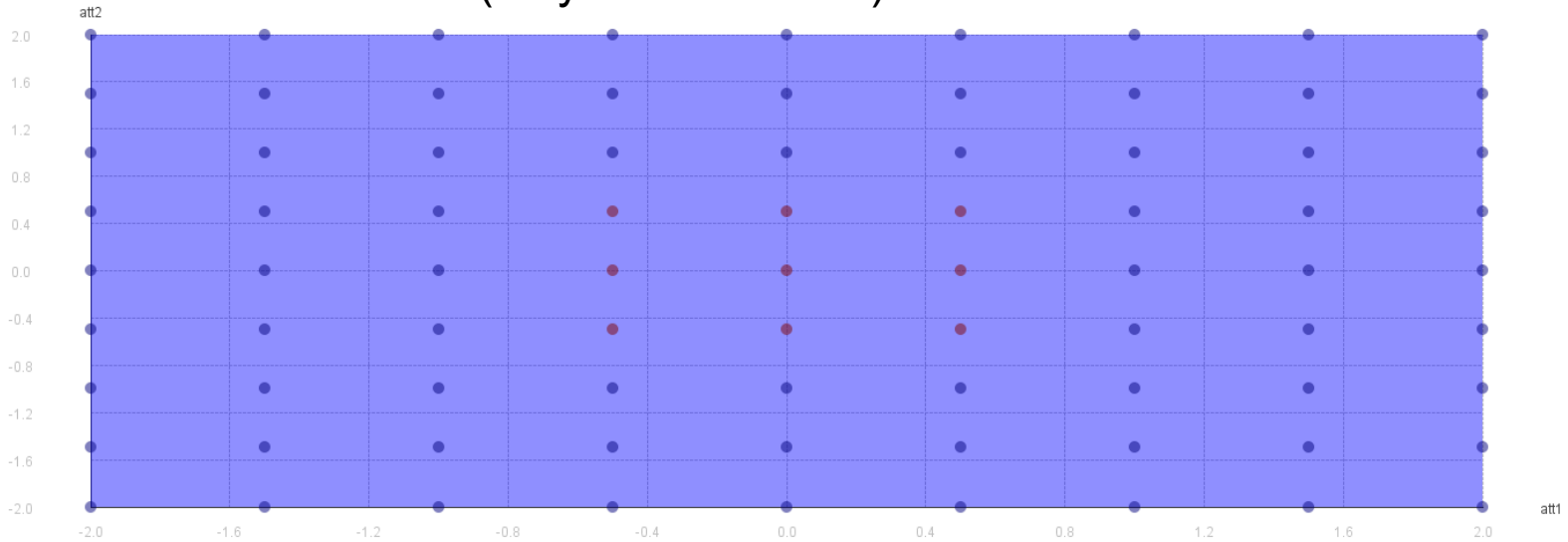
- Example 2:
  - a small class inside a large one
  - again: can be perfectly described with rectangles



Decision Boundaries: Theory and Practice

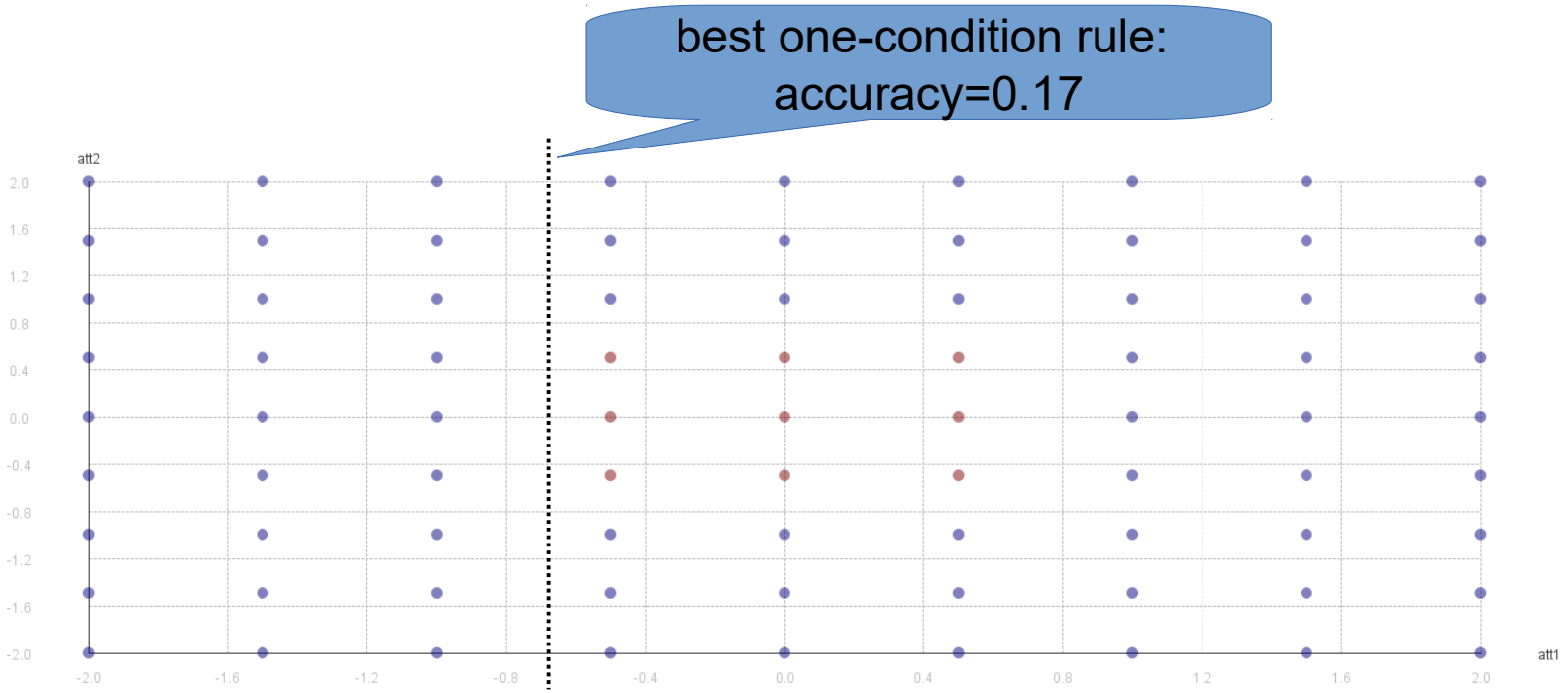
- Example 2:
  - a small class inside a large one
  - again: can be perfectly described with rectangles

- Output of a rule-based classifier:
  - no model learned (only a default rule)



Decision Boundaries: Theory and Practice

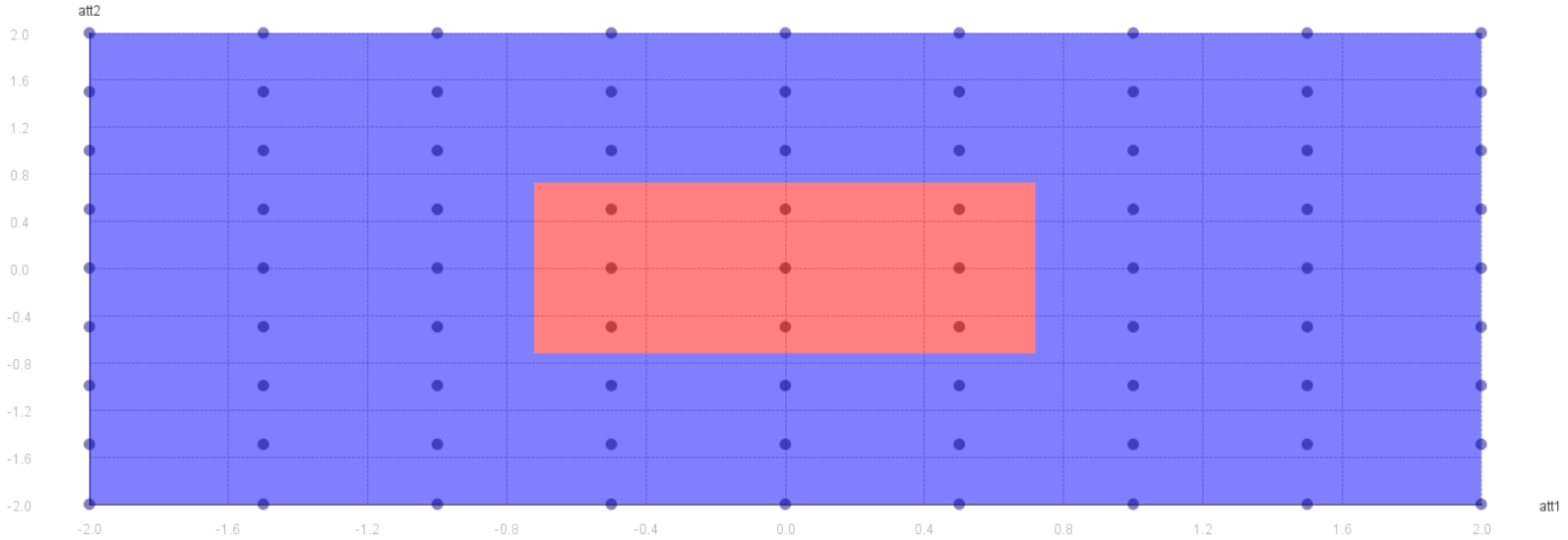
- What is happening here?
  - We try to learn the *smallest* class
  - ...and pre-pruning requires a *minimum* heuristic
  - no initial condition can be selected that exceeds that minimum



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: 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

Alternative Classification Methods

- There are various methods of classification
  - e.g., >50 methods in basic RapidMiner edition
  - plus many more using the Weka extension

- So far, we have seen
  - k-NN
  - Naive Bayes
  - Decision Trees
  - C4.5 and Ripper

- Brief intro
  - Artificial Neural Networks
  - Support Vector Machines

Example: Credit Rating

- Consider the following example:
  - and try to build a model
  - which is as small as possible (recall: Occam's Razor)

Person	Employed	Owns House	Balanced Account	Get Credit
Peter Smith	yes	yes	no	yes
Julia Miller	no	yes	no	no
Stephen Baker	yes	no	yes	yes
Mary Fisher	no	no	yes	no
Kim Hanson	no	yes	yes	yes
John Page	yes	no	no	no

Example: Credit Rating

- Smallest model:
  - if at least two of Employed, Owns House, and Balanced Account are yes
    → Get Credit is yes
- Not nicely expressible in trees and rule sets
  - as we know them (attribute-value conditions)

Person	Employed	Owns House	Balanced Account	Get Credit
Peter Smith	yes	yes	no	yes
Julia Miller	no	yes	no	no
Stephen Baker	yes	no	yes	yes
Mary Fisher	no	no	yes	no
Kim Hanson	no	yes	yes	yes
John Page	yes	no	no	no

Example: Credit Rating

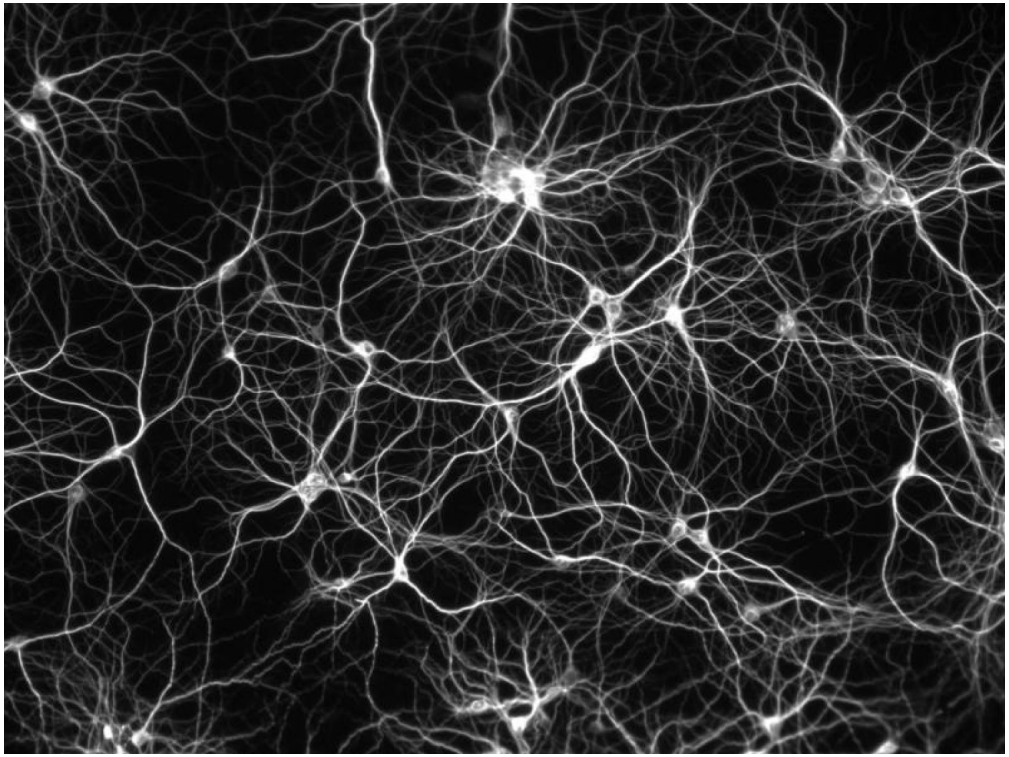
- Smallest model:
  - if at least two of Employed, Owns House, and Balance Account are yes → Get Credit is yes
- As rule set:

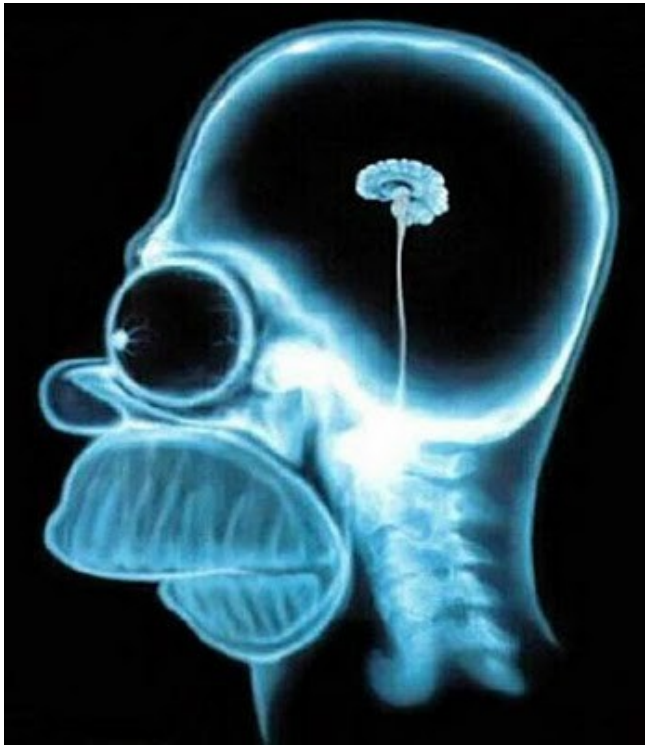
  Employed=yes and OwnsHouse=yes => yes
  Employed=yes and BalanceAccount=yes => yes
  OwnsHouse=yes and BalanceAccount=yes => yes
  => no

- General case:
  - at least m out of n attributes need to be yes => yes
  - this requires $\binom{n}{m}$ rules, i.e., $\frac{n!}{m! \cdot (n-m)!}$
  - e.g., “5 out of 10 attributes need to be yes” requires more than 15,000 rules!

Artificial Neural Networks

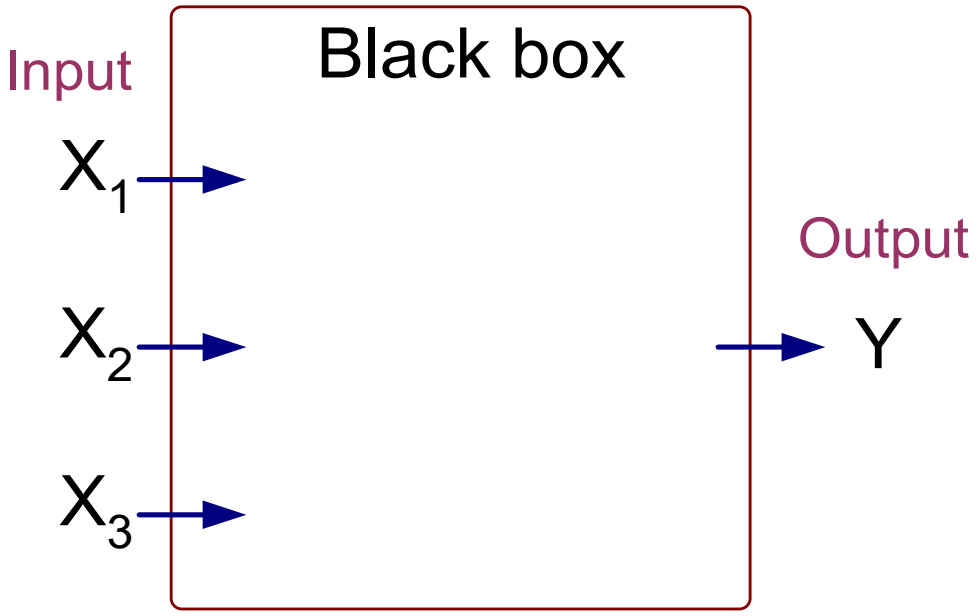
- Inspiration
  - one of the most powerful super computers in the world





Artificial Neural Networks (ANN)

X_1	X_2	X_3	Y
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	0
0	1	0	0
0	1	1	1
0	0	0	0



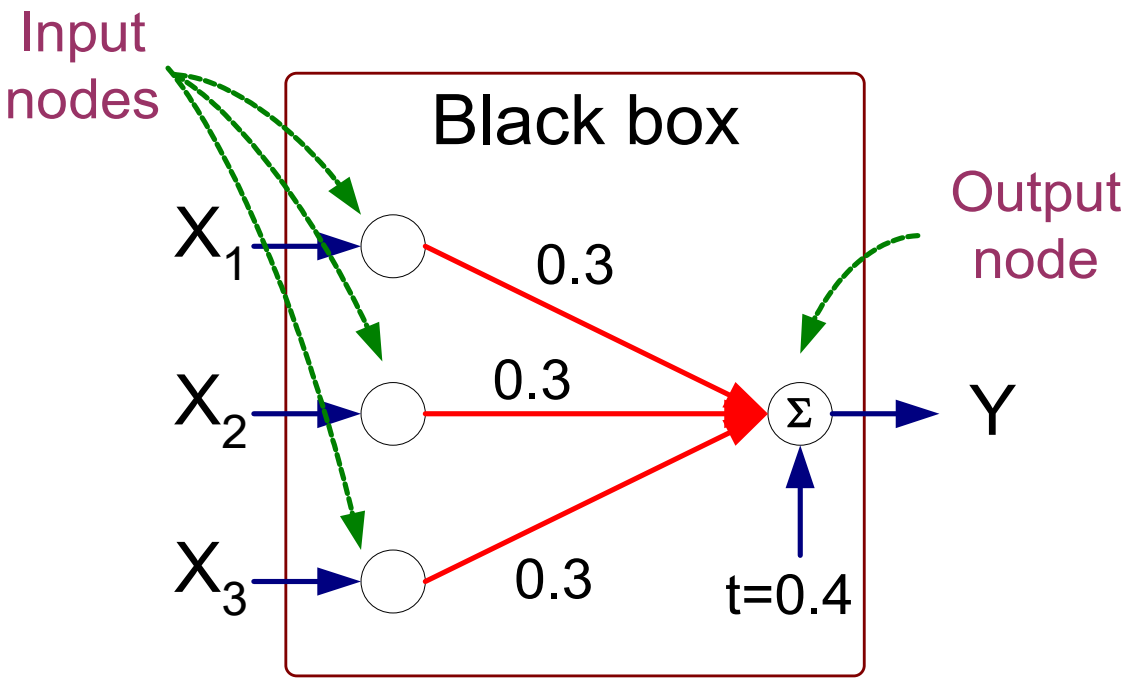
Output Y is 1 if at least two of the three inputs are equal to 1.

Example: Credit Rating

- Smallest model:
  - if at least two of Employed, Owns House, and Balance Account are yes
    → Get Credit is yes
- Given that we represent yes and no by 1 and 0, we want
  - if(Employed + Owns House + Balance Acount)>1.5
    → Get Credit is yes

Artificial Neural Networks (ANN)

X_1	X_2	X_3	Y
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	0
0	1	0	0
0	1	1	1
0	0	0	0

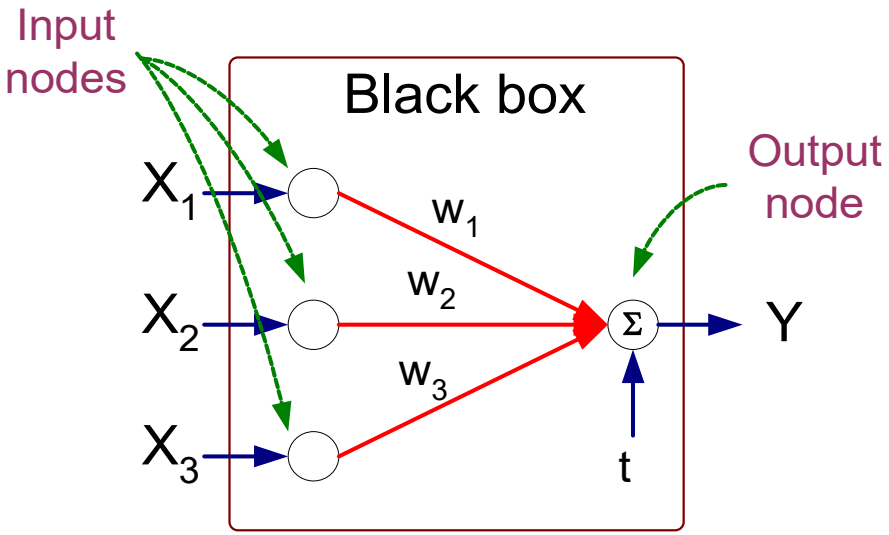


$$Y = I(0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4 > 0)$$

$$\text{where } I(z) = \begin{cases} 1 & \text{if } z \text{ is true} \\ 0 & \text{otherwise} \end{cases}$$

Artificial Neural Networks (ANN)

- Model is an assembly of inter-connected nodes and weighted links
- Output node sums up each of its input value according to the weights of its links
- Compare output node against some threshold t

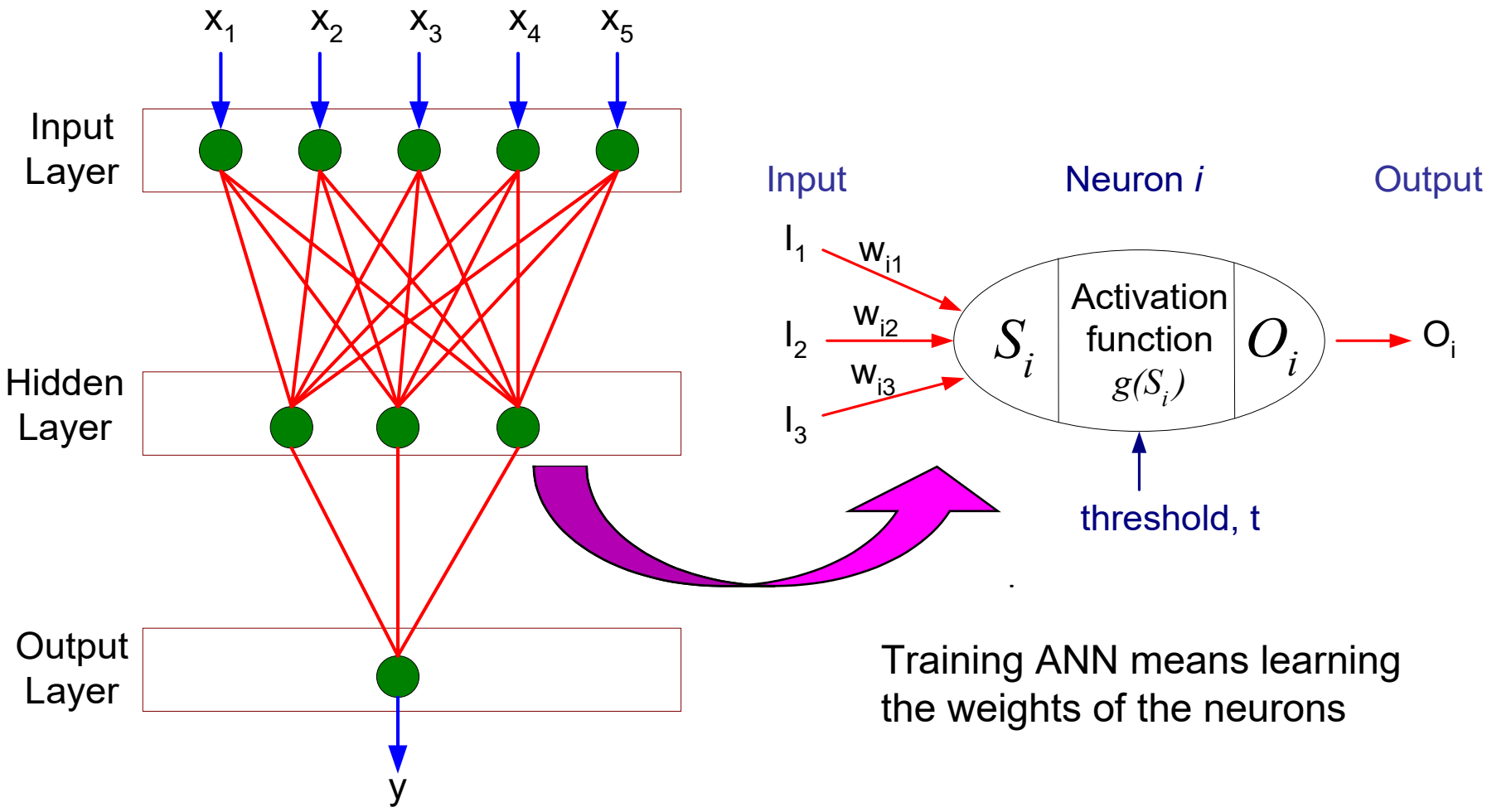


Perceptron Model

$$Y = I(\sum_i w_i X_i - t) \quad \text{or}$$

$$Y = sign(\sum_i w_i X_i - t)$$

General Structure of ANN



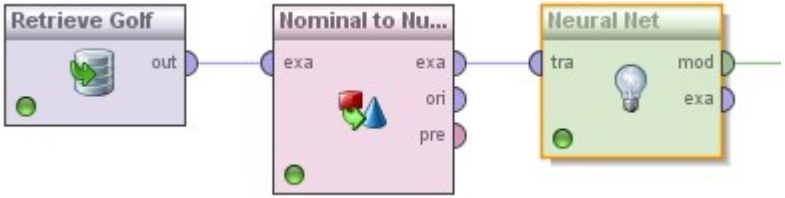
Training ANN means learning the weights of the neurons

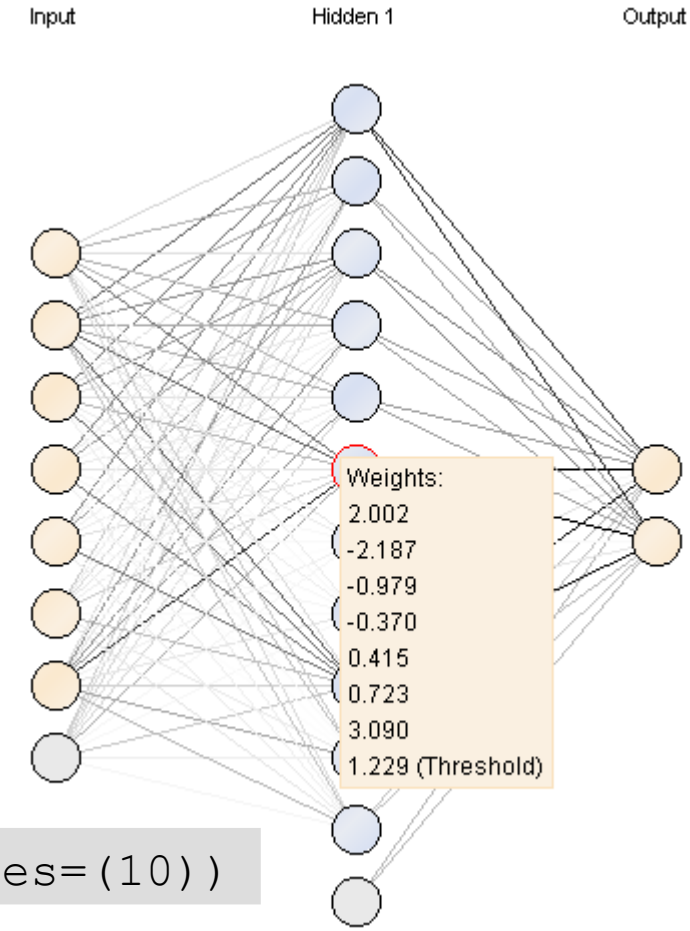
Algorithm for learning ANN

- Initialize the weights (w_0, w_1, …, w_k), e.g., all with 1

- Adjust the weights in such a way that the output of ANN is consistent with class labels of training examples
  - Objective function: $$E = \sum_i \left[Y_i - f(w_i, X_i) \right]^2$$
  - Find the weights w_i's that minimize the above objective function
    - e.g., back propagation algorithm (see books)

ANN in RapidMiner & Python

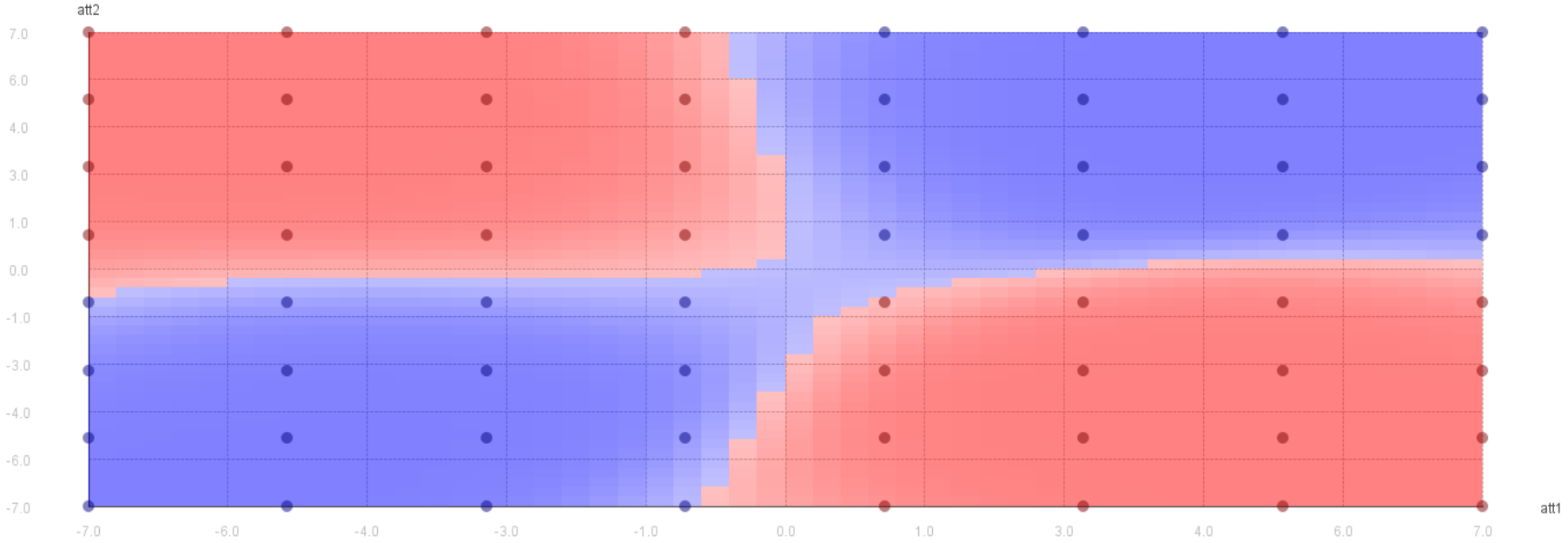




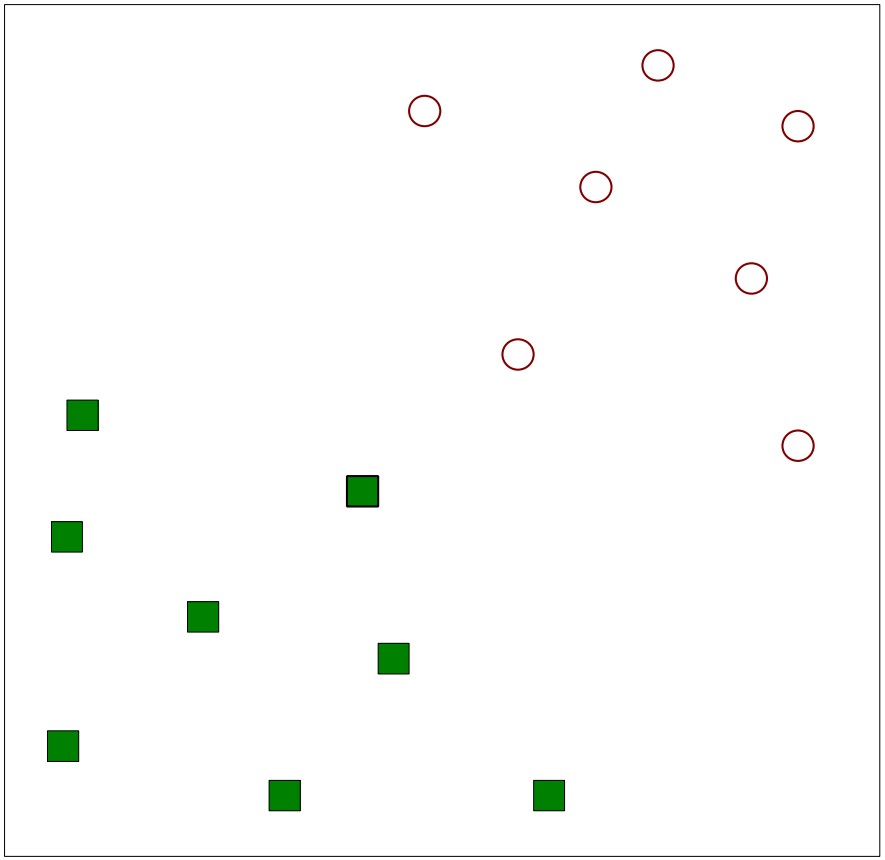

```
clf = MLPClassifier(hidden_layer_sizes=(10))
```

Decision Boundaries of ANN

- Arbitrarily shaped objects
- Fuzzy boundaries

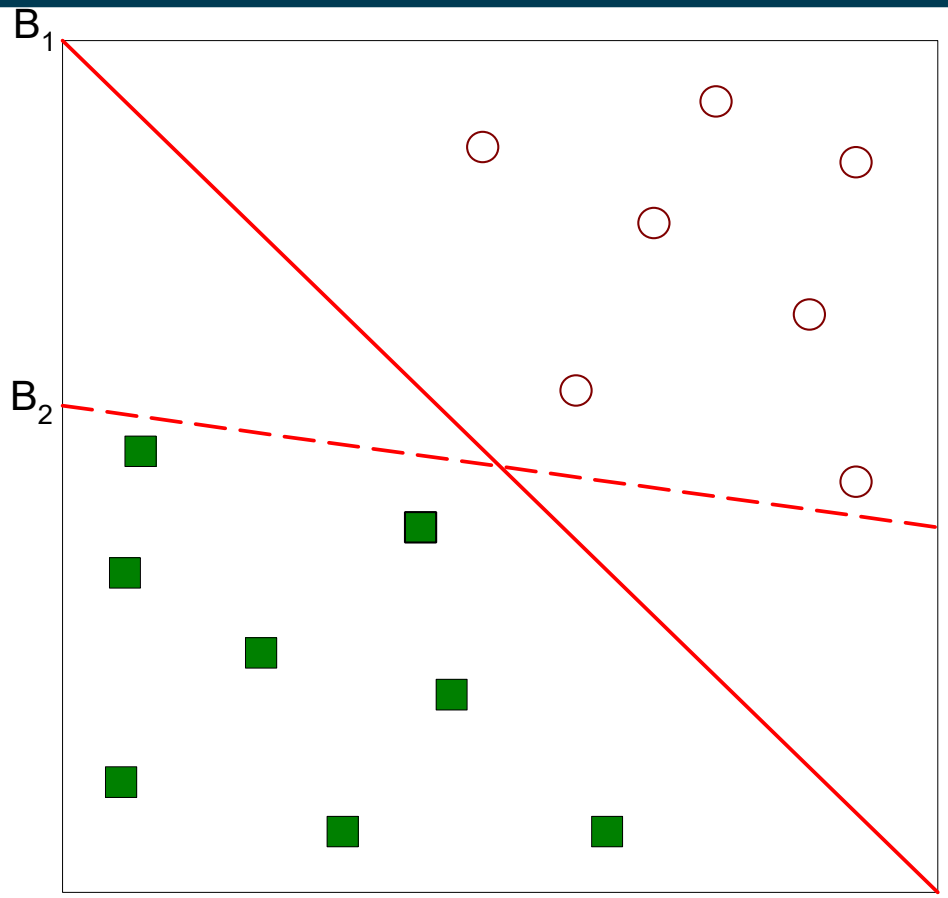


Support Vector Machines



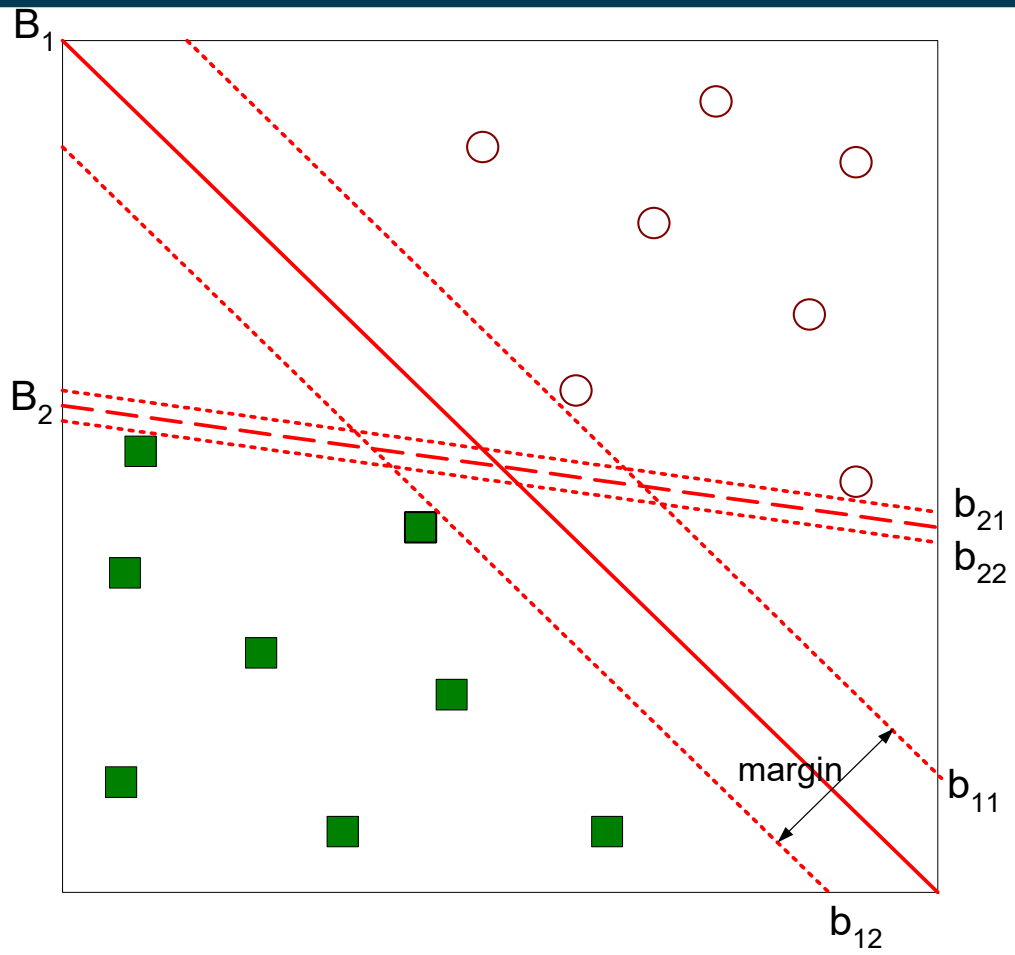
- Find a linear hyperplane (decision boundary) that will separate the data

Support Vector Machines



- Which one is better? B1 or B2?
- How do you define “better”?

Support Vector Machines



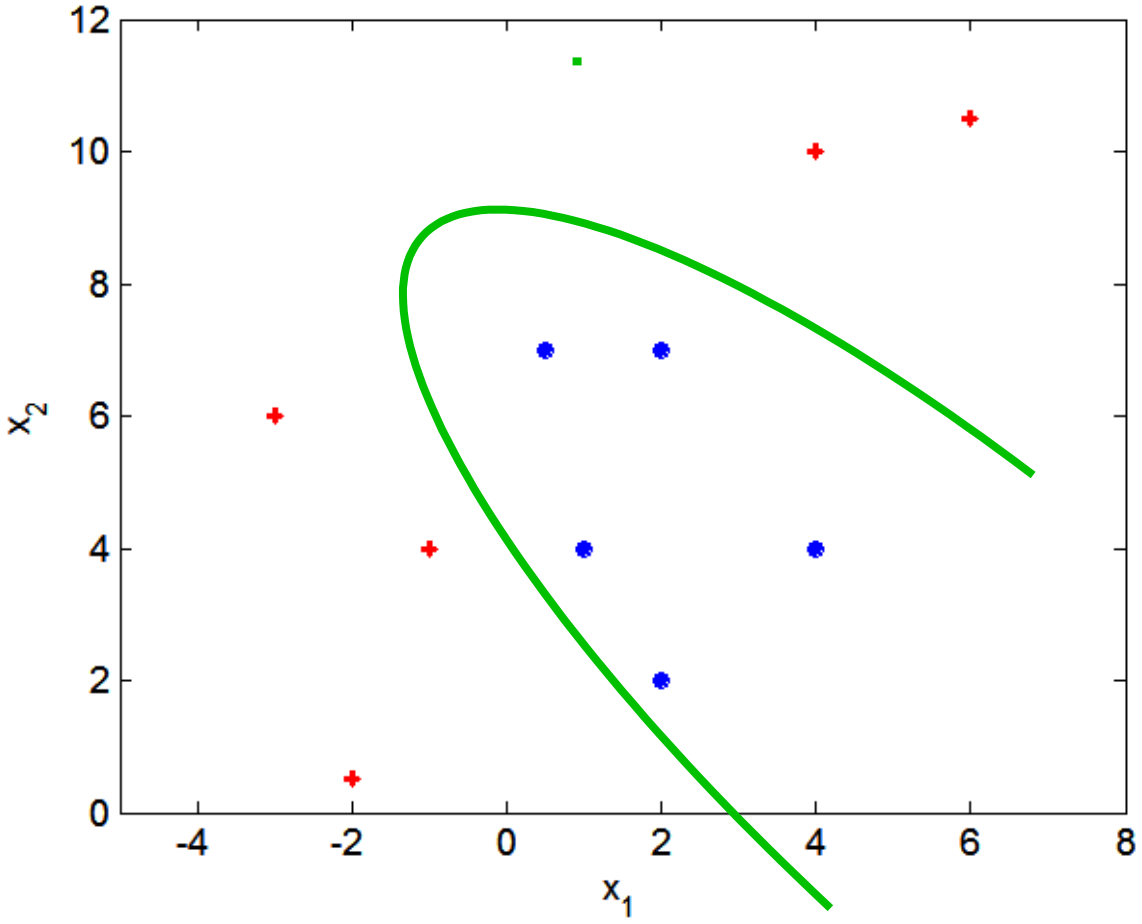
- Find hyperplane maximizes the margin => B1 is better than B2

Support Vector Machines

- What is computed
  - a separating hyper plane
  - defined by its support vectors (hence the name)
- Challenges
  - Computing an optimal separation is expensive
  - requires good approximations
- Dealing with noisy data
  - introducing “slack variables” in margin computation

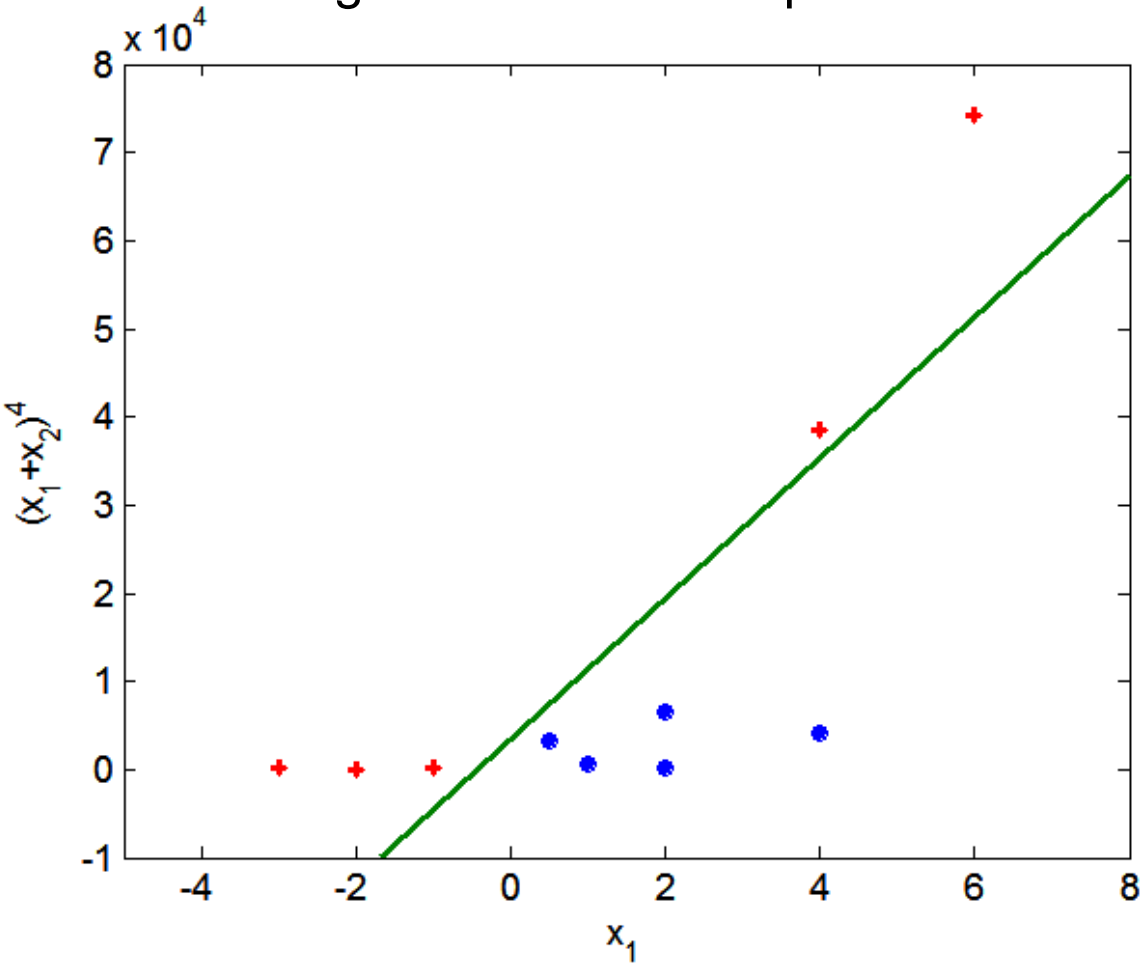
Nonlinear Support Vector Machines

- What if decision boundary is not linear?



Nonlinear Support Vector Machines

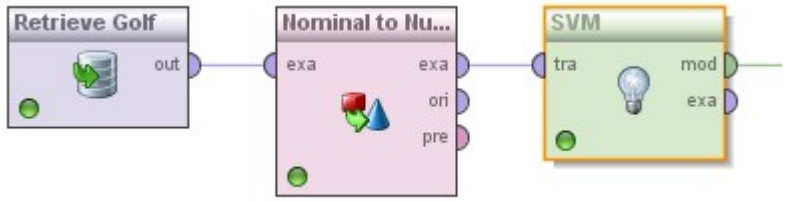
- Transform data into higher dimensional space

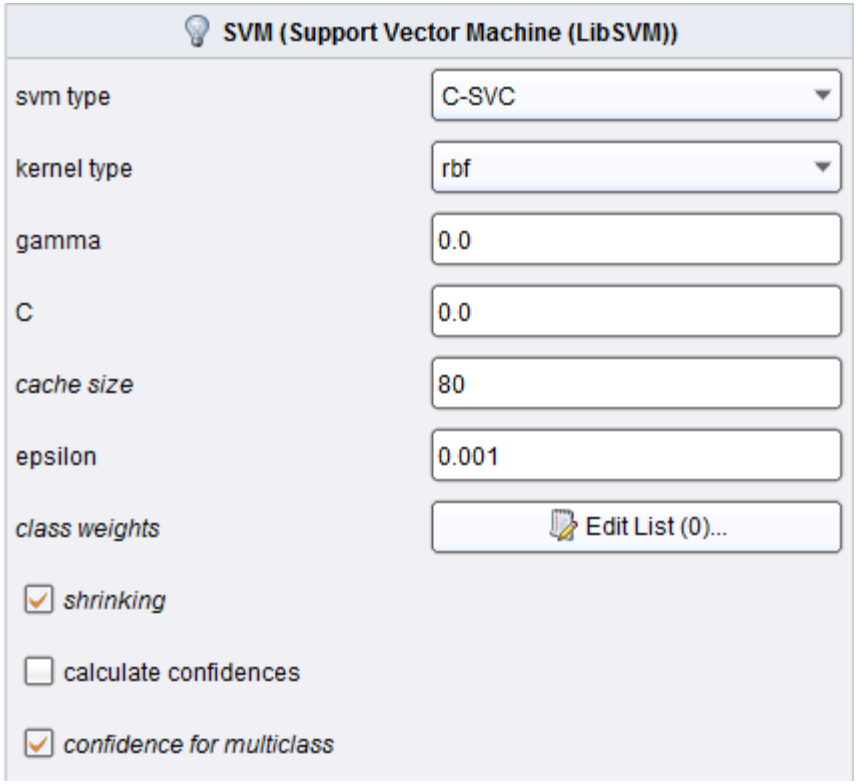


Nonlinear Support Vector Machines

- Transformation in higher dimensional space
  - Uses so-called Kernel function
  - Different variants: polynomial function, radial basis function, …
- Finding a hyperplane in higher dimensional space
  - is computationally expensive
  - Kernel trick: expensive parts of the calculation can be performed in lower dimensional space
  - Details: see books...

SVMs in RapidMiner & Python

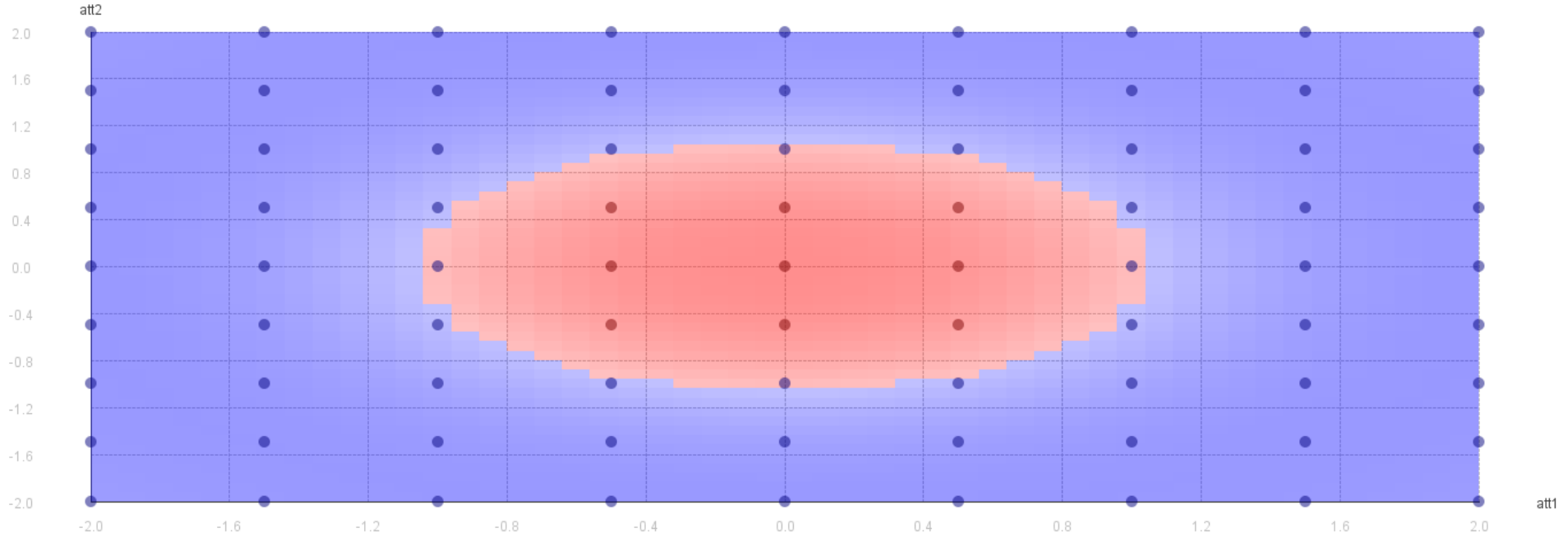





```
clf = SVC(kernel='rbf')
```

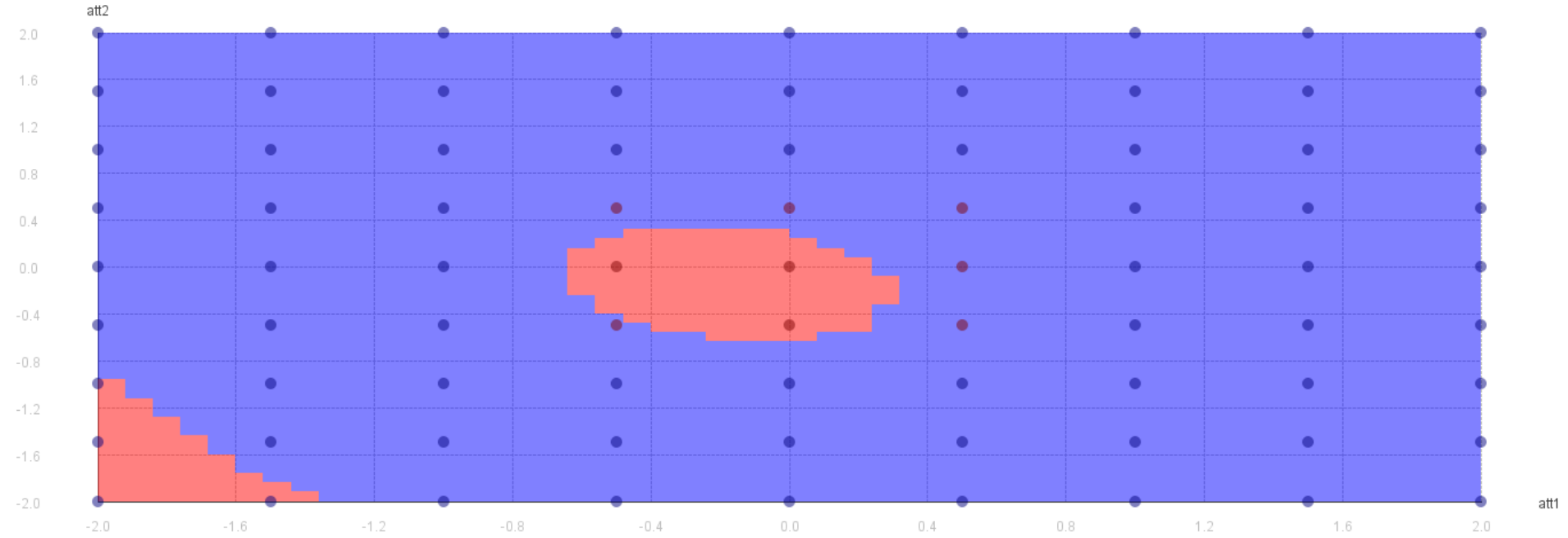
Decision Boundaries of SVMs

- Small class in large one
  - using radial basis function kernel



Decision Boundaries of SVMs

- Small class in large one
  - using a Gaussian function kernel



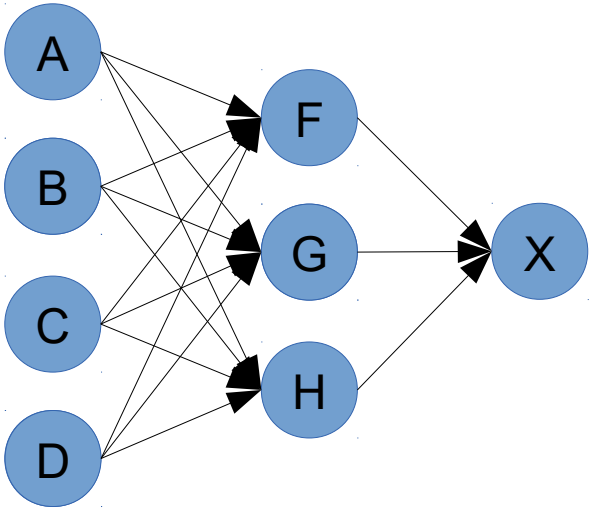
More Exotic Problems

- Consider
  - Four binary features A,B,C,D
  - Goal: Classify true if the number of TRUE values is even (i.e., 0, 2, or 4)

- Very hard for classic machine learning problems
  - Approximate solution can be learned with neural network

More Exotic Problems

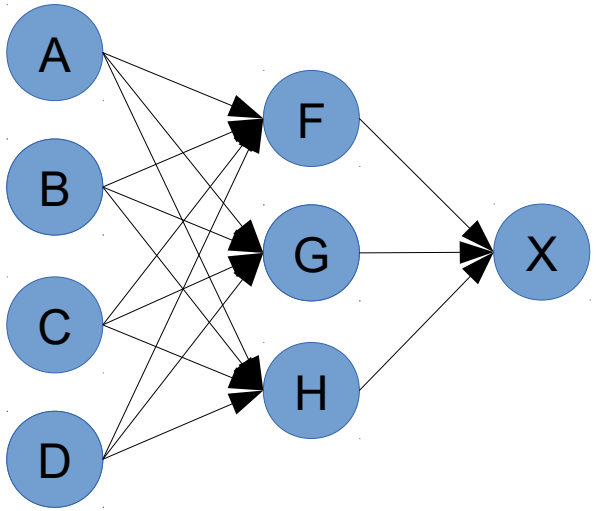
- Consider
  - Four binary features A,B,C,D
  - Goal: Classify true if the number of TRUE values is even (i.e., 0, 2, or 4)



$$F = A + B + C - D < 3$$
$$G = A + B + C - D < 2$$
$$H = A + B + C - D < 1$$

$$X = F + G - H < 2$$

More Exotic Problems



$F = A + B + C - D < 3$
$G = A + B + C - D < 2$
$H = A + B + C - D < 1$

$X = F + G - H < 2$

A	B	C	D	F	G	H	X	C
0	0	0	0	1	1	1	TRUE	TRUE
1	0	0	0	1	1	0	FALSE	FALSE
0	1	0	0	1	1	0	FALSE	FALSE
0	0	1	0	1	1	0	FALSE	FALSE
0	0	0	1	1	1	1	TRUE	FALSE
1	1	0	0	1	0	0	TRUE	TRUE
0	1	1	0	1	0	0	TRUE	TRUE
0	0	1	1	1	1	1	TRUE	TRUE
1	0	0	1	1	1	1	TRUE	TRUE
1	0	1	0	1	0	0	TRUE	TRUE
0	1	0	1	1	1	1	TRUE	TRUE
1	1	1	0	0	0	0	TRUE	FALSE
1	1	0	1	1	1	0	FALSE	FALSE
1	0	1	1	1	1	0	FALSE	FALSE
0	1	1	1	1	1	0	FALSE	FALSE
1	1	1	1	1	0	0	TRUE	TRUE

Parameter Tuning

- Many learning methods require parameters
  - k for k nearest neighbors
  - pruning thresholds for trees and rules
  - hidden layer configuration for ANN
  - kernel function, epsilon and gamma for SVM
  - …

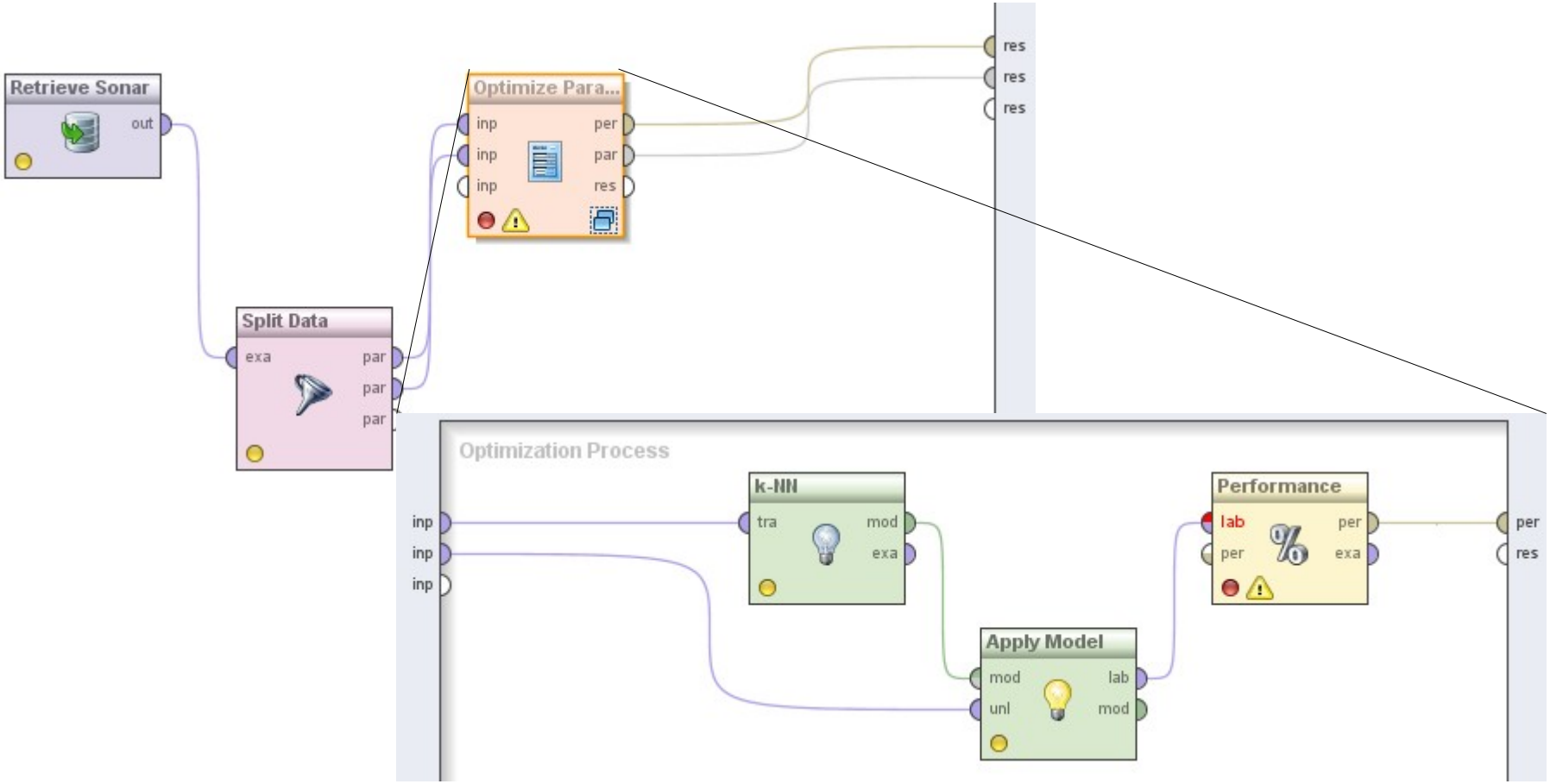
- How to define optimal parameters?
  - Trying out different configurations
  - systematic approaches

Parameter Tuning

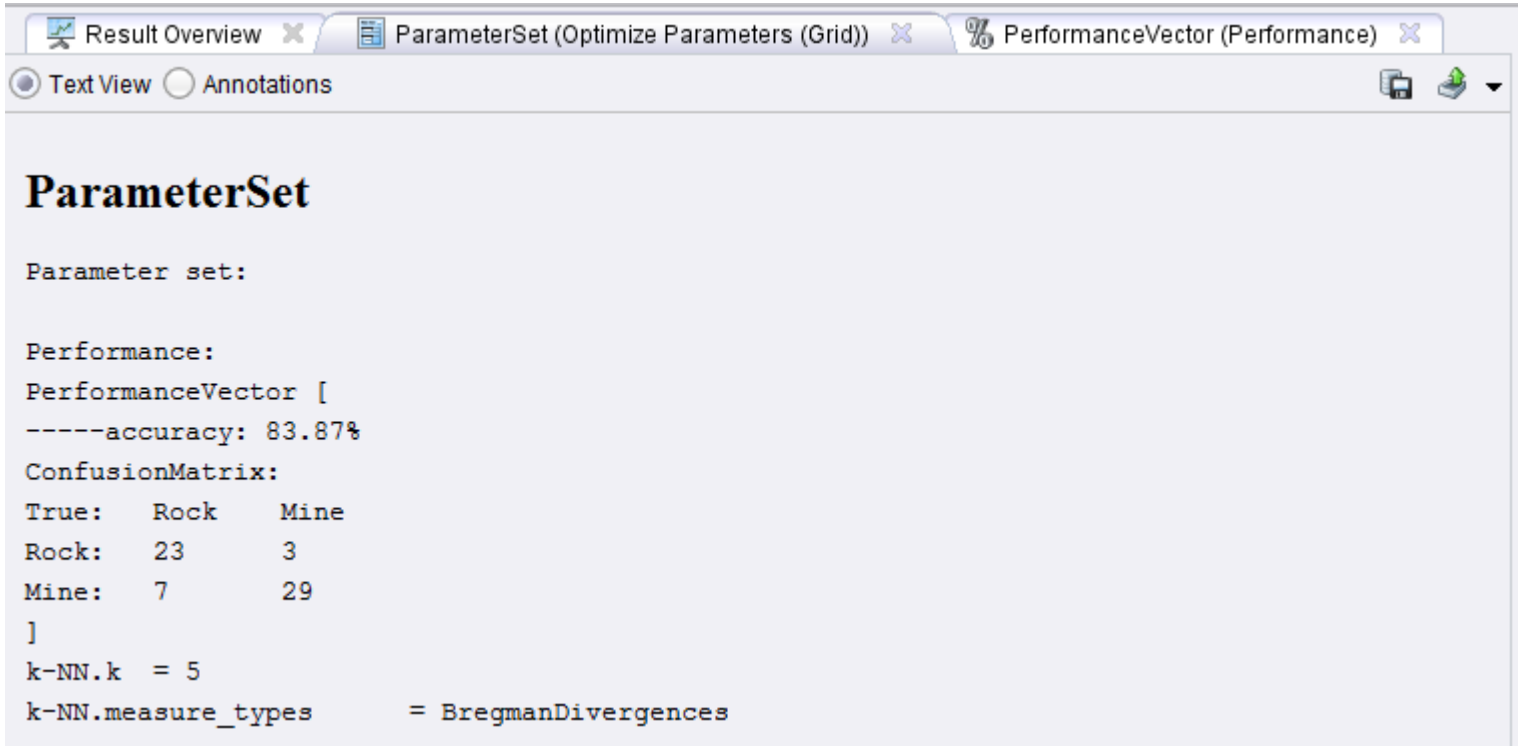
- Some approaches often work rather poorly with default parameters
  - SVMs are a typical example here
- Systematic approaches (see Data Mining 2)
  - GridSearch: search space for possible combinations
    - Gamma=0, Eps=0; Gamma=0.1, Eps=0; Gamma=0.2, Eps=0...
  - Local Hill Climbing, Beam Search
  - Evolutionary algorithms, genetic programming
  - …

- Attention
  - use hold out set (or cross validation) for parameter tuning
  - there may be an overfitting problem here as well!

Parameter Tuning in RapidMiner



Parameter Tuning in RapidMiner



Parameter Tuning in Python

```
GridSearchCV(cv=10,
        estimator=SVC(C=1.0, kernel='rbf',
                      gamma=0.1),
        param_grid = [
             { 'C': [1, 10, 100, 1000],
               'kernel': ['linear','rbf],
               'gamma: [0.1,0.01,0.001]})
```

Issues in Parameter Tuning

- Which parameters are sensitive and interesting?
- Which combinations to test?
  - Computational explosion
- Avoiding overfitting
  - Split or cross validation
  - Never use the test set for parameter tuning!

Summary

- Classification approaches
  - There are quite a few: Nearest Neighbors, Naive Bayes, Decision Trees, Rules, SVMs, Neural Networks
- Distinctions
  - Lazy vs. eager
  - Performance (accuracy, training time, testing time, model size)
  - Decision Boundaries (theory and practice)
- Issues
  - Overfitting
  - Parameter tuning

Questions?