

Data Mining

Classification

- Part 2 -



Outline

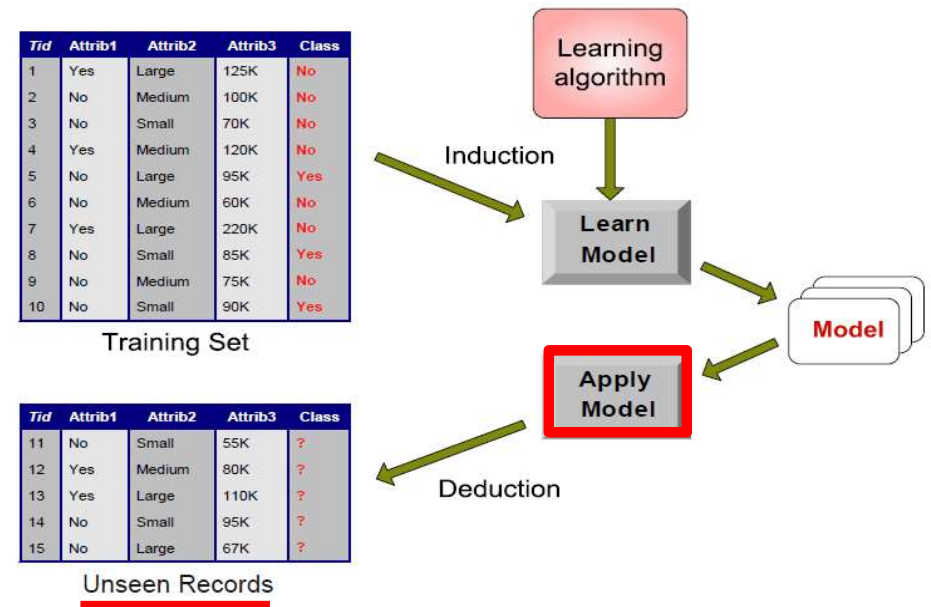
1. What is Classification?
2. K-Nearest-Neighbors
3. Decision Trees
4. Model Evaluation
5. Rule Learning
6. Naïve Bayes
7. Support Vector Machines
8. Artificial Neural Networks
9. Hyperparameter Selection

4. Model Evaluation

Central Question:

How good is a model at classifying unseen records?

(generalization performance)



4.1 Metrics for Model Evaluation

- How to measure the performance of a model?

4.2 Methods for Model Evaluation

- How to obtain reliable estimates?

4.1 Metrics for Model Evaluation

- Focus on the **predictive capability** of a model
 - rather than how much time it takes to classify records or build models
- The confusion matrix counts the correct and false classifications
 - the counts are the basis for calculating different performance metrics

Confusion Matrix

ACTUAL CLASS	PREDICTED CLASS		
		Class=Yes	Class=No
	Class=Yes	True Positives	False Negatives
	Class=No	False Positives	True Negatives

Accuracy and Error Rate

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

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

	PREDICTED CLASS		
		Class= Yes	Class= No
	Class= Yes	TP 25	FN 4
	Class= No	FP 6	TN 15

$$\text{Acc} = \frac{25 + 15}{25 + 15 + 6 + 4} = 0.80$$

The Class Imbalance Problem

- Classes often have **very unequal frequency**
 - Fraud detection: 98% transactions OK, 2% fraud
 - E-commerce: 99% surfers 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 negative examples = 9990
number of positive examples = 10
 - if model predicts all examples to belong to the negative class, the accuracy is $9990/10000 = 99.9\%$
 - **Accuracy is misleading** because model does not detect any positive example

Precision and Recall

Alternative: Use performance metrics from information retrieval which are biased towards the positive class by ignoring 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

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

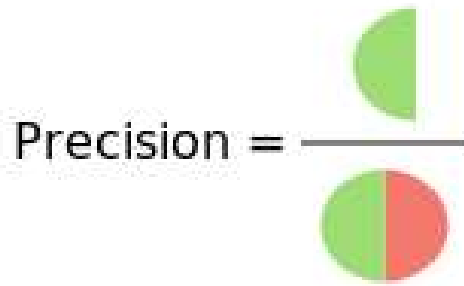
	Classified Positive	Classified Negative
Actual Positive	TP	FN
Actual Negative	FP	TN

Ignored
majority



Precision and Recall - Visualized

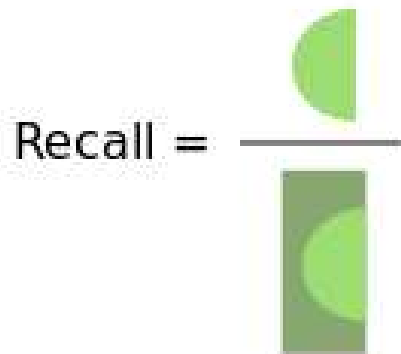
How many examples that are classified positive are actually positive?



Precision =

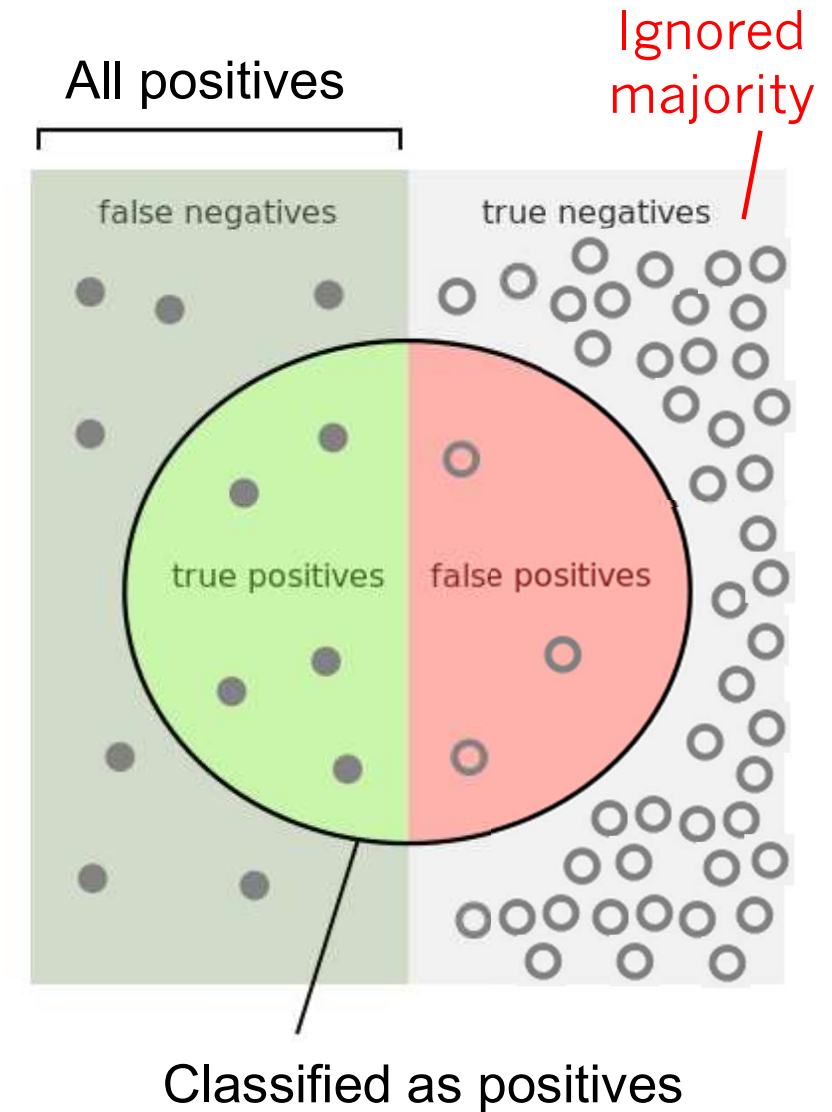
$$p = \frac{TP}{TP + FP}$$

Which fraction of all positive examples is classified correctly?



Recall =

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



Source: Walber

Precision and Recall – A Problematic Case

	Classified Positive	Classified Negative
Actual Positive	1	99
Actual Negative	0	1000

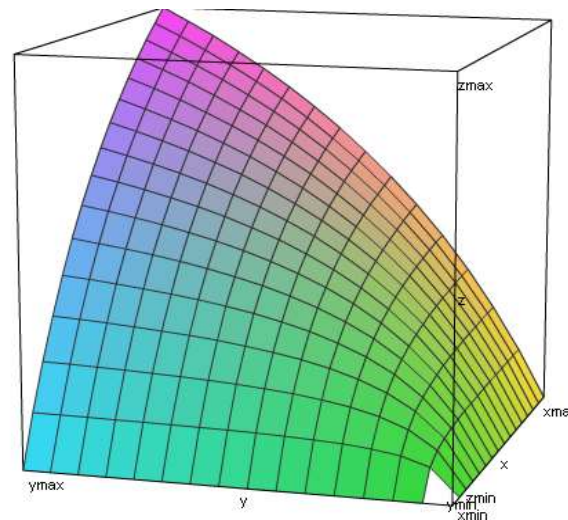
- This confusion matrix gives us
 - precision $p = 100\%$
 - recall $r = 1\%$
- because we only classified one positive example correctly and no negative examples wrongly
- Thus, we want a measure that
 1. combines precision and recall and
 2. is large if both values are large

F₁-Measure

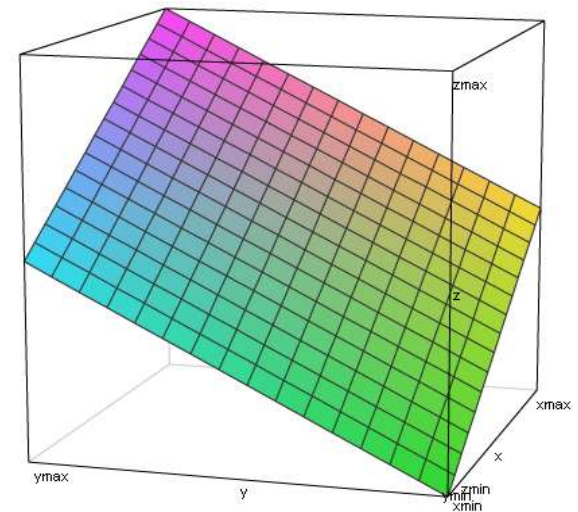
- F₁-score **combines precision and recall** into one measure
- F₁-score is the harmonic mean of precision and recall
 - the harmonic mean of two numbers tends to be closer to the smaller of the two
 - thus for the F₁-score to be large, both p and r must be large

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

Harmonic mean



Arithmetic mean



Example: Alternative Metrics on Imbalanced Data

	PREDICTED CLASS		
		Class=Yes	Class=No
	Class=Yes	10	0
	Class=No	10	980

	PREDICTED CLASS		
		Class=Yes	Class=No
	Class=Yes	1	9
	Class=No	0	990

$$\text{Precision (p)} = \frac{10}{10+10} = 0.5$$

$$\text{Recall (r)} = \frac{10}{10+0} = 1$$

$$F_1\text{-measure (F}_1\text{)} = \frac{2 * 1 * 0.5}{1 + 0.5} = 0.62$$

$$\text{Accuracy} = \frac{990}{1000} = 0.99$$

$$\text{Precision (p)} = \frac{1}{1+0} = 1$$

$$\text{Recall (r)} = \frac{1}{1+9} = 0.1$$

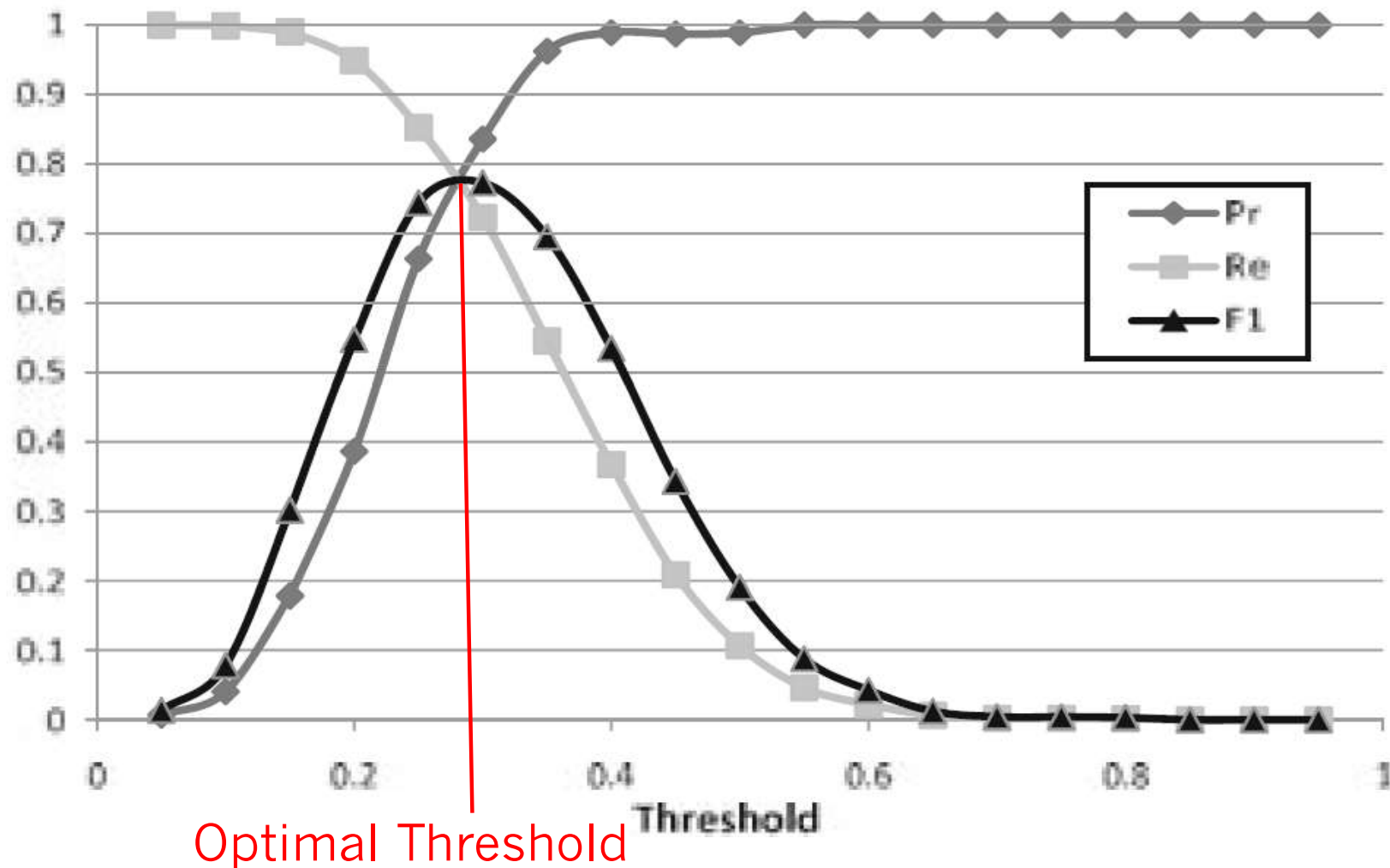
$$F_1\text{-measure (F}_1\text{)} = \frac{2 * 0.1 * 1}{1 + 0.1} = 0.18$$

$$\text{Accuracy} = \frac{991}{1000} = 0.991$$

F₁-Measure Graph

Low threshold: Low precision, high recall

Restrictive threshold: High precision, low recall



Cost-Sensitive Model Evaluation

$C(i|j)$: Cost of misclassifying a class j record as class i

	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	$C(\text{Yes} \text{No})$	$C(\text{No} \text{No})$

Example: Cost-Sensitive Model Evaluation

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

- Use case: Credit card fraud
- it is expensive to miss fraudulent transactions
 - false alarms are not too expensive

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

Accuracy = 67%

Cost = 3798 ← Better model

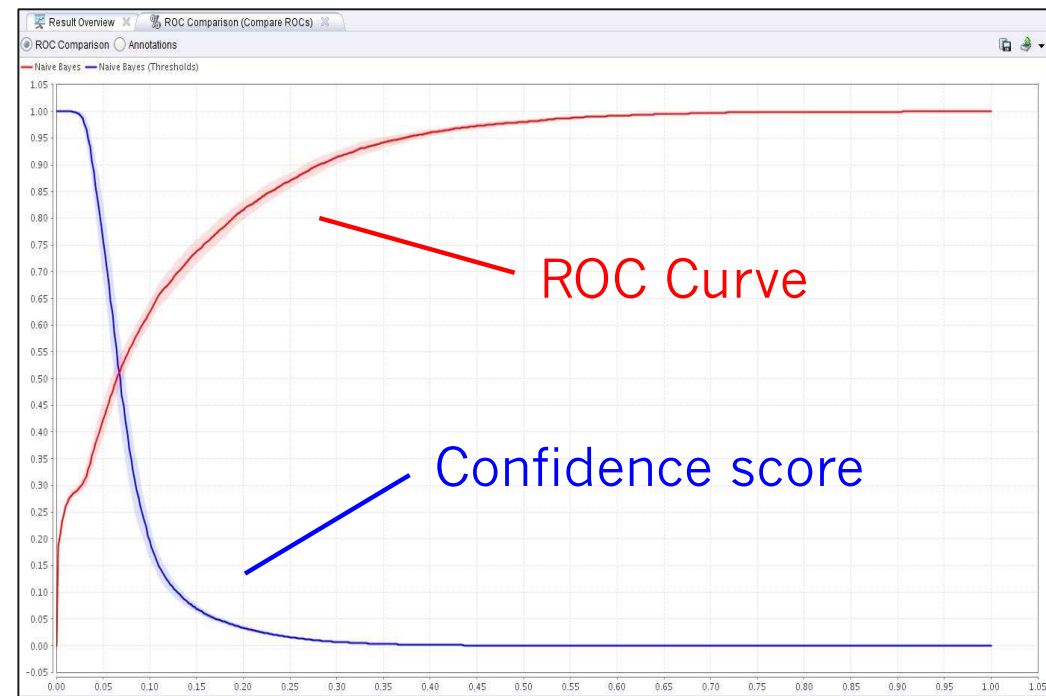
Model M_2	PREDICTED CLASS		
ACTUAL CLASS		+	-
	+	155	45
	-	5	395

Accuracy = 92%

Cost = 4350

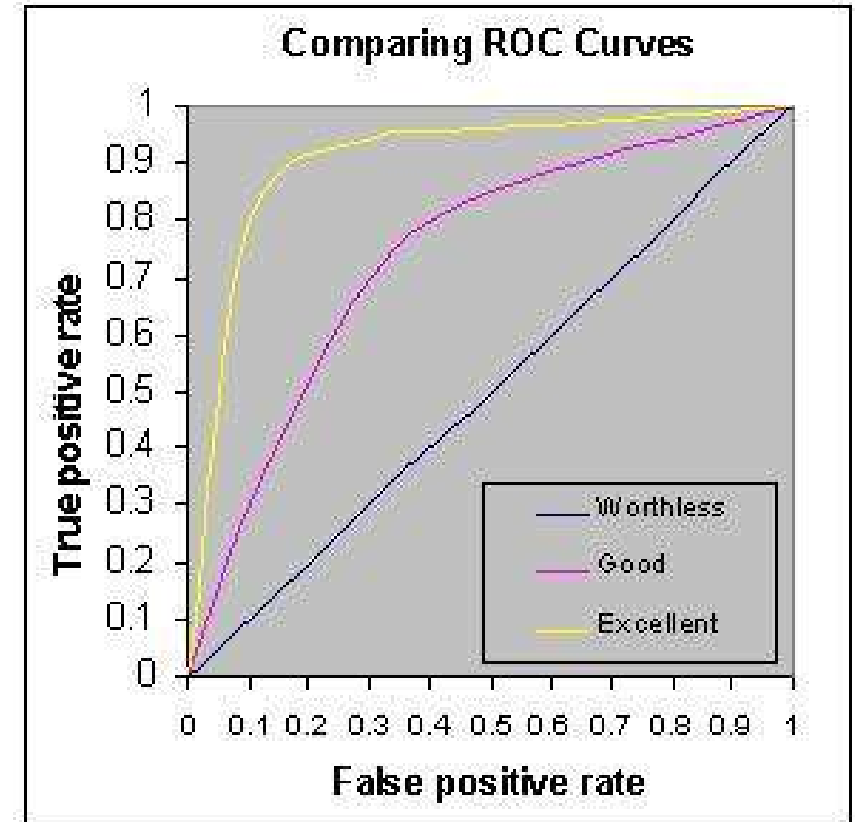
ROC Curves

- Graphical approach for displaying trade-off between detection rate and false alarm rate
- Some classification algorithms provide **confidence scores**
 - how sure the algorithm is with its prediction
 - e.g., KNN (the neighbor's vote), Naive Bayes (the probability)
- ROC curves visualize **true positive rate** and **false positive rate** in relation to the algorithm's confidence
- Drawing a ROC Curve
 - Sort classifications according to confidence scores
 - Scan over all classifications
 - right prediction: draw one step up
 - wrong prediction: draw one step to the right
 - Exact method: Tan, Chapter 6.11



Interpreting ROC Curves

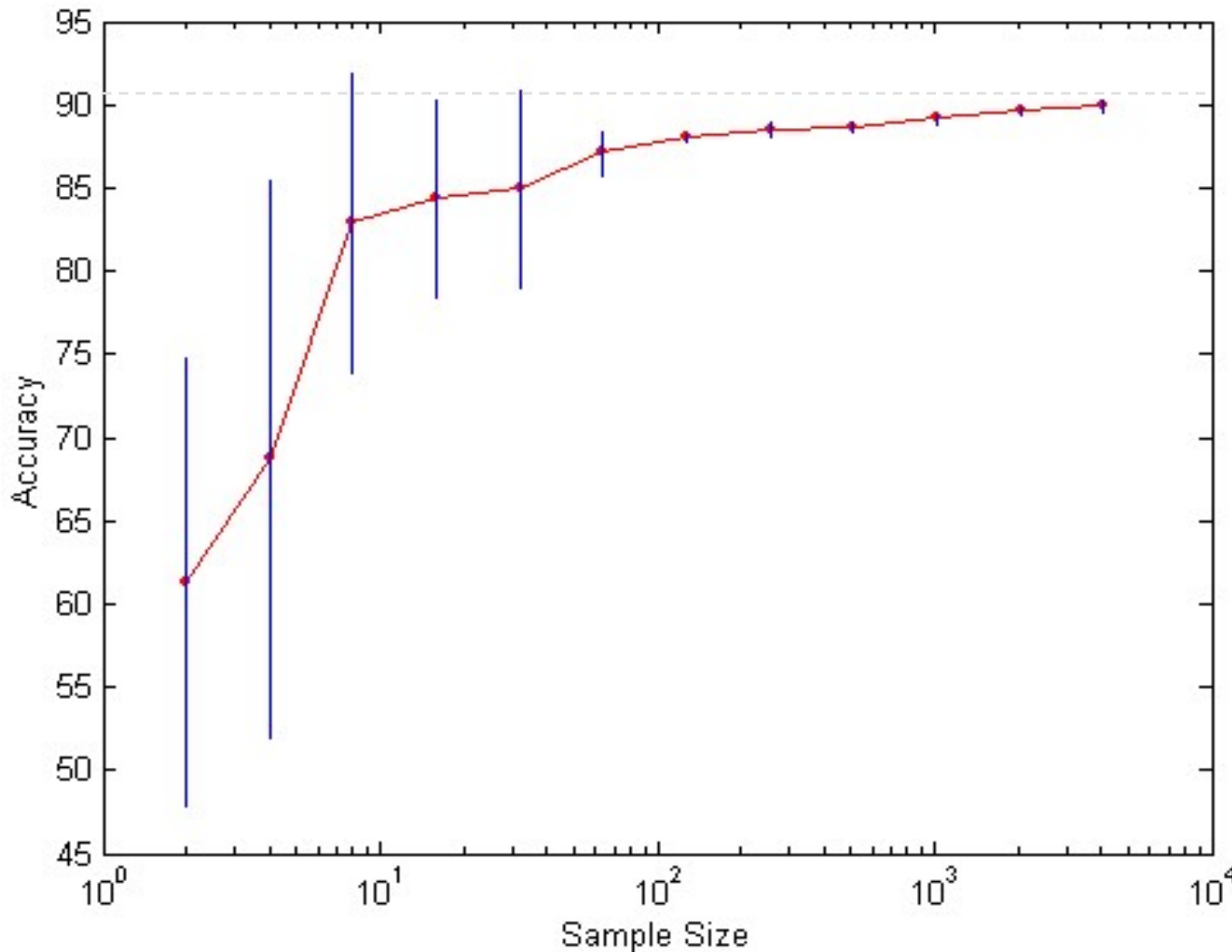
- The steeper, the better
 - random guessing results in the diagonal
 - so a decent classification model should result in a curve above the diagonal
- Comparing models:
 - Curve A above curve B means model A better than model B
- Measure for comparing models
 - Area under ROC curve (AUC)



4.2 Methods for Model Evaluation

- How to obtain a **reliable estimate** of the **generalization performance**?
- General approach: Split labeled records into a **training set** and a **test set**
- Never ever test a model on data that was used for training!
 - Because model has been fit to training data, evaluating on training data does not result in a suitable estimate of the performance on unseen data
 - We need to keep training set and test set strictly separate
- Which labeled records to use for training and which for testing?
- Alternative splitting approaches:
 1. Holdout Method
 2. Random Subsampling
 3. Cross Validation

Learning Curve



- The learning curve shows how accuracy changes with growing training set size
- **Conclusion:**
 - If model performance is low and unstable, get more training data
 - **Use labeled data rather for training than testing**
- **Problem:**
 - Labeling additional data is often expensive due to manual effort involved

Holdout Method

- The **holdout method** reserves a certain amount of the labeled data for testing and uses the remainder for training
- Usually: 1/3 for testing, 2/3 for training (or even better 20% / 80%)



- For imbalanced datasets, random samples might not be representative
 - few or no records of the minority class (aka positive class) in training or test set
- **Stratified sample**: Sample each class independently, so that records of the minority class are present in each sample

Random Subsampling

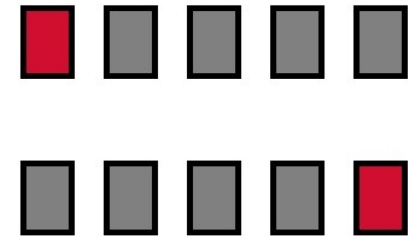
- Holdout estimate can be made more reliable by repeating the process with different subsamples
 - in each iteration, a certain proportion is **randomly selected** for training
 - the performance of the different iterations is **averaged**



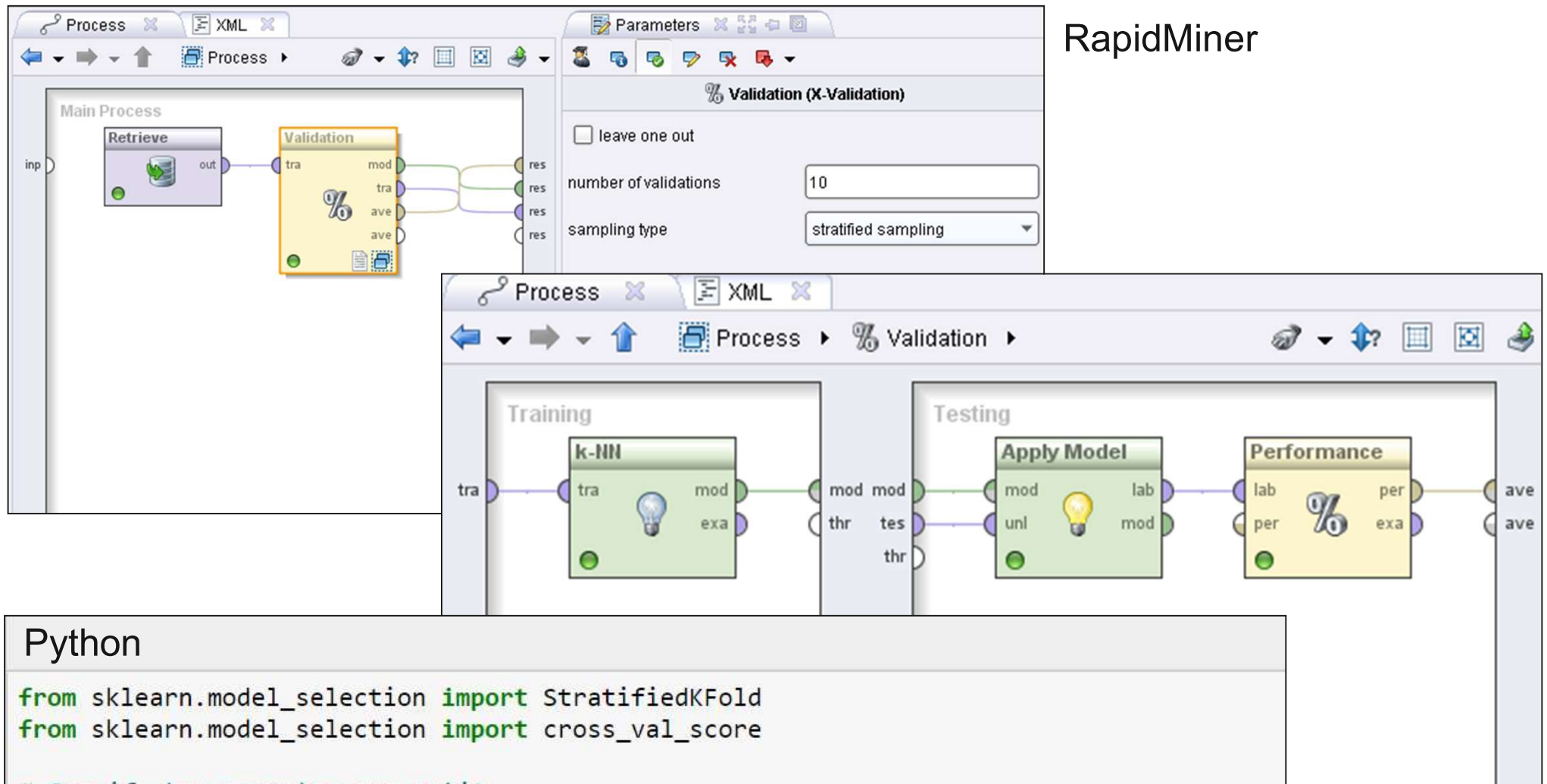
- Still not optimal as the different test sets may overlap
 1. problem: some outliers might always end up in the test sets
 2. problem: important records for learning (red tree) might always be in test sets

Cross-Validation

- 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 x-validation
- Every record is used exactly once for testing
- The performance estimates of all runs are averaged to yield overall performance estimate
- Frequently used: $k = 10$ (90% training, 10% testing)
 - why ten? Experiments have shown that this is the good choice to get an accurate estimate and still use as much data as possible for training
- Often the subsets are generated using stratified sampling
 - in order to deal with class imbalance



Cross-Validation in RapidMiner and Python



Python

```
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score

# Specify how examples are split
cross_val = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

# Run cross-validation and calculate performance metric
accuracy = cross_val_score(estimator, data, target, cv=cross_val, scoring='accuracy')
```

Cross-Validation Results in RapidMiner

Average accuracy over all 10 runs (test sets)

Standard deviation of accuracy values over all 10 runs (test sets)

accuracy: 92.00% +/- 5.26% (micro average: 92.00%)

	true Iris-setosa	true Iris-versicolor	true Iris-virginica	class precision
pred. Iris-setosa	50	0	0	100.00%
pred. Iris-versicolor	0	46	8	85.19%
pred. Iris-virginica	0	4	42	91.30%
class recall	100.00%	92.00%	84.00%	

Recall given that we define Iris-setosa as positive class

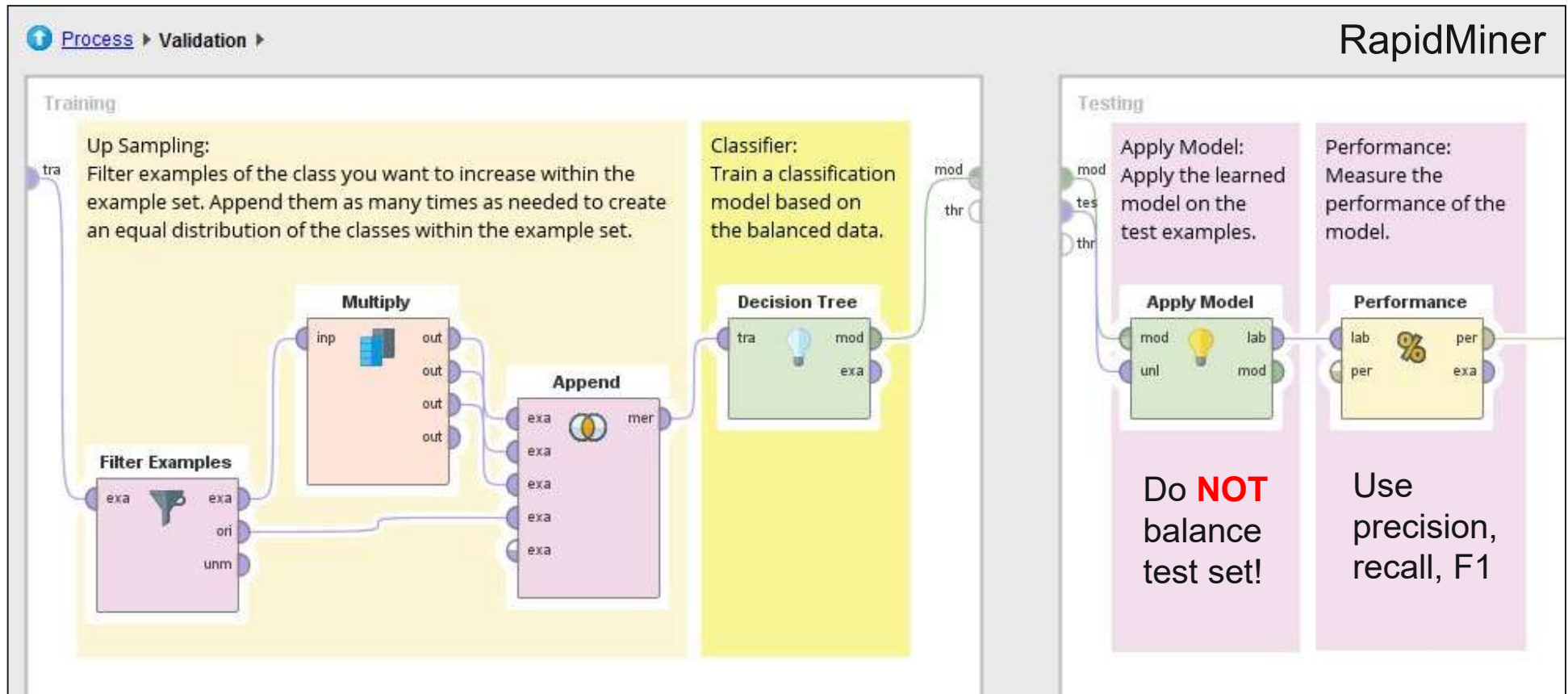
Number of correctly classified Iris-versicolor examples in all runs (test sets)

Each record is used exactly once for testing → The numbers in the confusion matrix sum up to the size of the labeled dataset

Evaluation Summary

- Performance metrics
 - Default: **use accuracy**
 - If interesting class is infrequent: use precision, recall, and F1
- Estimation of metric
 - Default: **use cross-validation**
 - If labeled dataset is large (>5000 examples) and
 - computation takes too much time or
 - exact replicability of results matters, e.g. for data science competitionsuse the holdout method with fixed split
- To increase model performance
 1. balance “imbalanced” data by increasing the number of positive examples in the training set (oversampling)
 2. optimize the hyperparameters of the learning algorithm (see next slide set)
 3. avoid overfitting (see previous slide set)

Dealing with Class Imbalance in Training and Testing



Python

```
from imblearn.over_sampling import RandomOverSampler

# Up-sample positive class
sampler = RandomOverSampler()
balanced_training_data, balanced_target = sampler.fit_resample(training_data, target)
```

5. Rule-based Classification

- Classify records by using a collection of “if...then...” rules
- Classification rule: *Condition* $\rightarrow y$
 - *Condition* is a conjunction of attribute tests (**rule antecedent**)
 - *y* is the class label (**rule consequent**)
- Examples of classification rules:
R1: (Blood Type=Warm) \wedge (Lay Eggs=Yes) \rightarrow Birds
R2: (Taxable Income < 50K) \wedge (Refund=Yes) \rightarrow Cheat = No
- **Rule-based classifier**
 - set of classification rules
 - predictions can easily be explained (eager learning)

Example: Rule-based Classifier

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
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

5.1 Applying a Rule-based Classifier

- A rule r **covers** a record x if the attributes of the record 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	?

- The rule R1 covers hawk \rightarrow Bird
- The rule R3 covers grizzly bear \rightarrow Mammal

Rule Coverage and Accuracy

– Coverage of a rule

- fraction of all records that satisfy the condition of a rule.

– Accuracy of a rule

- fraction of covered records that satisfy the consequent of a rule

– Example

- R1: (Status=Single) → No
- Coverage = 40%
- Accuracy = 50%

<i>Tid</i>	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

Characteristics of Rule-based Classifiers

– Mutually Exclusive Rule Set

- the rules in a rule set are mutually exclusive if no two rules are triggered by the same record
- ensures that every record is covered by at most one rule

– Exhaustive Rule Set

- a rule set has exhaustive coverage if there is a rule for every combination of attribute values
- ensures that every record is covered by at least one rule

A Rule Set that is not Mutually Exclusive and Exhaustive

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 turtle triggers both R4 and R5 \rightarrow not mutually exclusive
- A dogfish shark triggers none of the rules \rightarrow not exhaustive

Fixes for not Mutually Exclusive and Exhaustive Rule Sets

- Not Exhaustive Rule Set
 - Problem: Some records are not covered by the rules
 - Solution: Add default rule: $() \rightarrow Y$
- Not Mutually Exclusive Rule Set
 - Problem: An record might be covered by multiple rules
 - Solution 1: Ordered Rules
 - order rules (e.g. prefer rules with high accuracy)
 - classify record according to the highest-ranked rule
 - Solution 2: Voting
 - let all matching rules vote and assign the majority class label
 - the votes may be weighted by rule quality (e.g. accuracy)

Example: Ordered Rule Set

- Rules are ordered according to their priority (e.g. accuracy)
- 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 fires, it is assigned to the default class

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

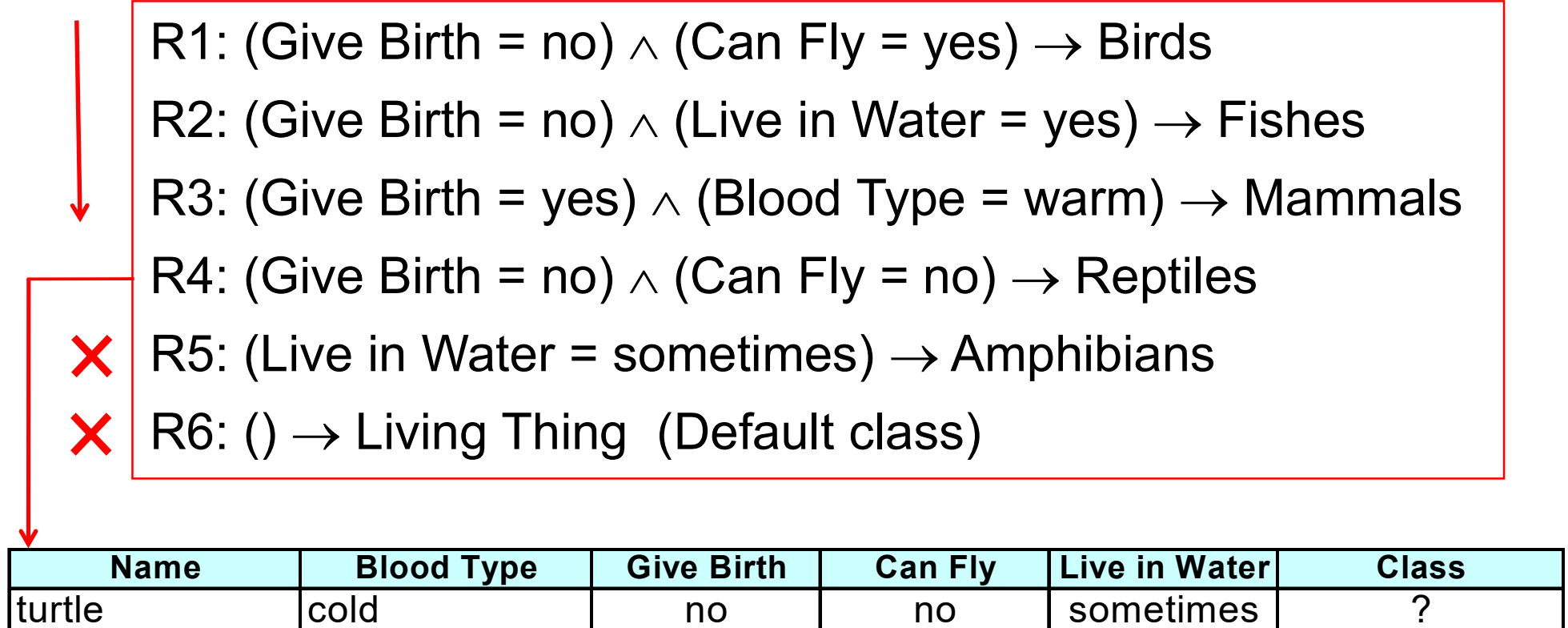
R2: (Give Birth = no) \wedge (Live in Water = yes) \rightarrow Fishes

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

✗ R6: () \rightarrow Living Thing (Default class)



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

5.2 Learning Rule-based Classifiers

1. Direct Method

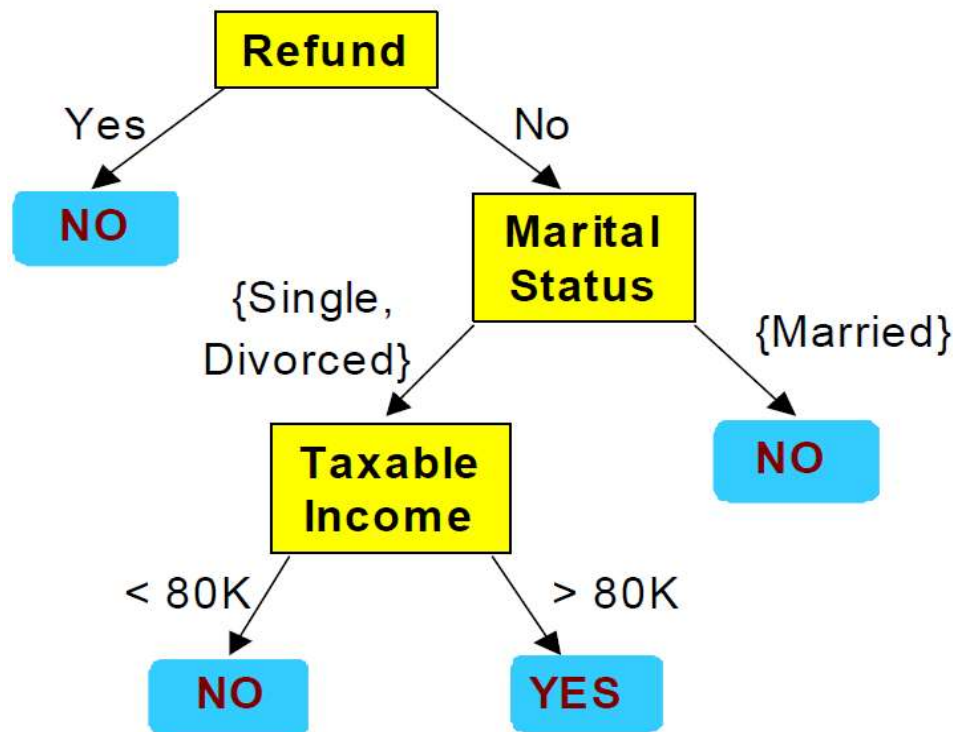
- Extract rules directly from data
- Example algorithm: RIPPER

2. Indirect Method

- Extract rules from other classification models (e.g. decision trees)
- Example: C4.5rules

5.2.1 Indirect Method: From Decision Trees To Rules

- Approach: Generate a rule **for every path** from the root to one of the leaf nodes in the decision tree
- Rule set contains as much information as the tree
- The generated rules are mutually exclusive and exhaustive



Classification Rules

(Refund=Yes) ==> No

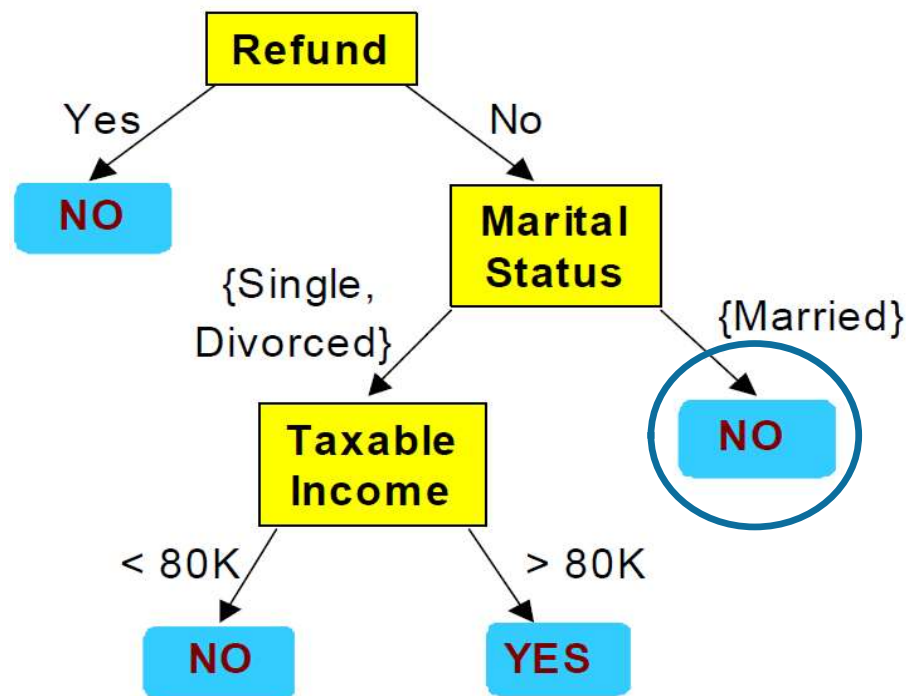
(Refund=No, Marital Status={Single, Divorced}, Taxable Income<80K) ==> No

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

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

The Generated Rules Can Be Simplified

- to avoid overfitting
- to be easier to understand



<i>Tid</i>	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

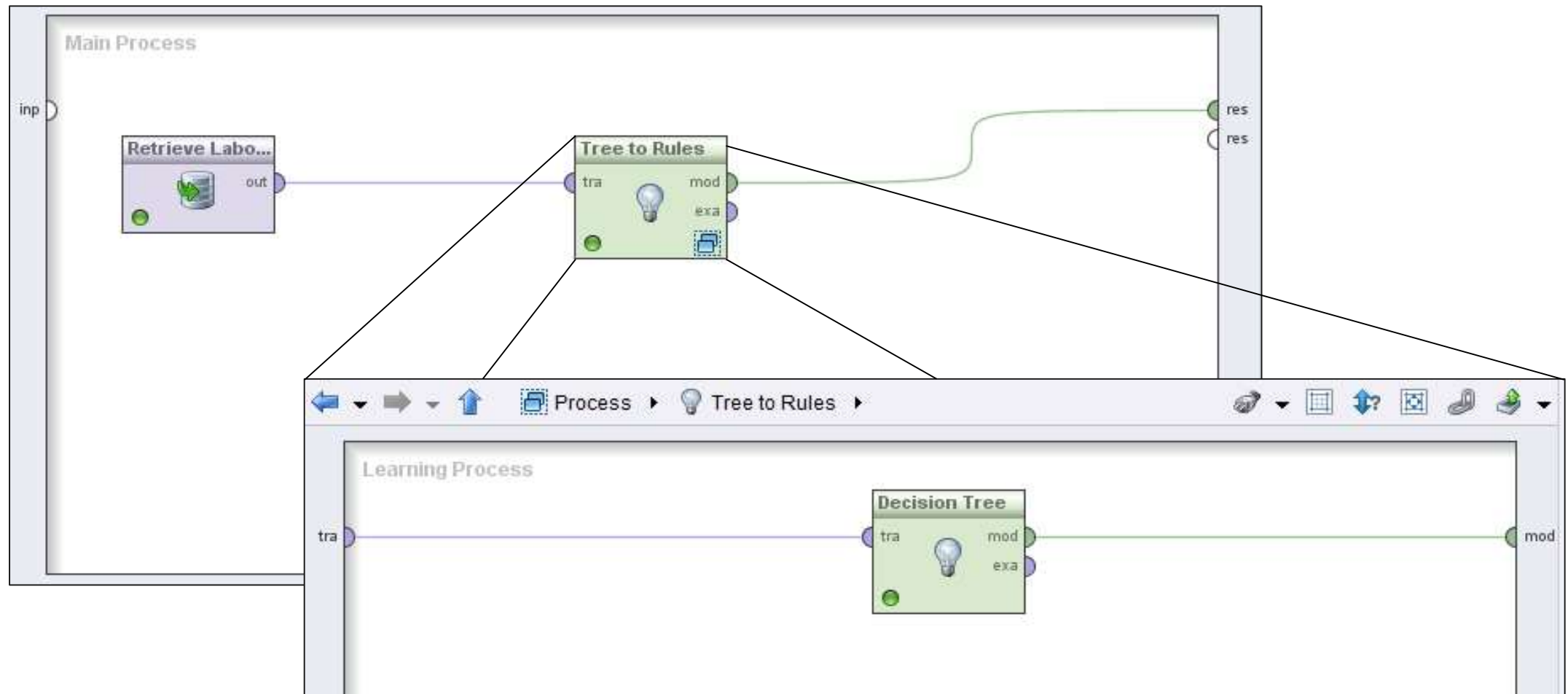
Initial Rule: $(\text{Refund}=\text{No}) \wedge (\text{Status}=\text{Married}) \rightarrow \text{No}$

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

Indirect Method: C4.5rules

1. Extract rules from an unpruned decision tree
2. For each rule, $r: A \rightarrow y$,
 1. consider an alternative rule $r': A' \rightarrow y$
where A' is obtained by **removing one of the conjuncts** in A
 2. **compare the estimated error rate** for r against all r 's
 - estimate error rate using training data plus a length penalty
 - or measure error using a validation dataset
 3. prune if one of the r 's has lower error rate
 4. repeat until we can no longer improve the generalization error
- Effect of rule simplification: Rule set is no longer mutually exclusive
 - A record may trigger more than one rule
 - Solution?
 - use ordered rule set or unordered rule set and voting scheme

Indirect Method in RapidMiner



Direct Method: RIPPER

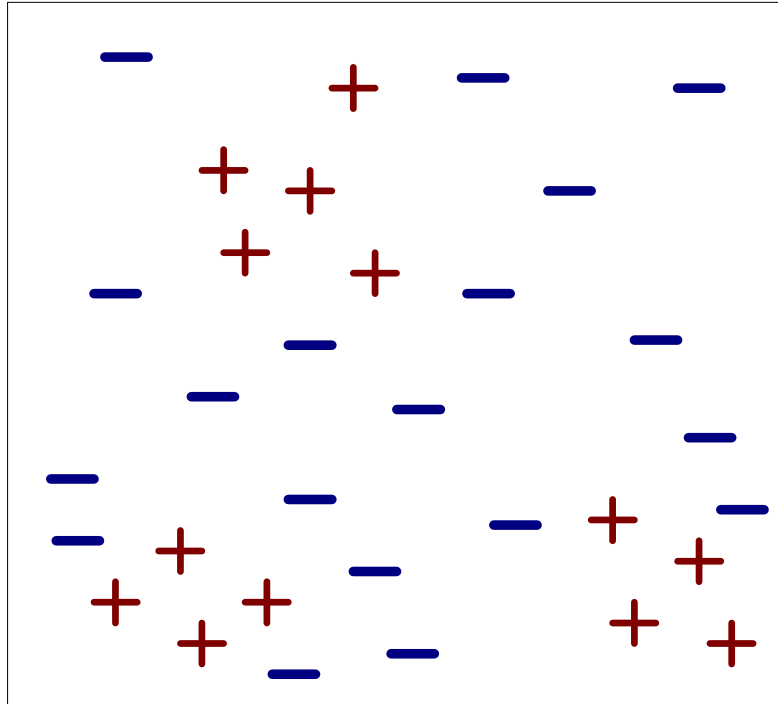
- Learns **ordered rule set** from training data
- For 2-class problem
 - choose the less frequent class as positive class and the other as negative class
 - learn rules for the positive class
 - negative class will be default class
- For multi-class problem
 - order the classes according to increasing class prevalence (fraction of instances that belong to a particular class)
 - learn the rule set for smallest class first, treat the rest as negative class
 - repeat with next smallest class as positive class

Sequential Covering

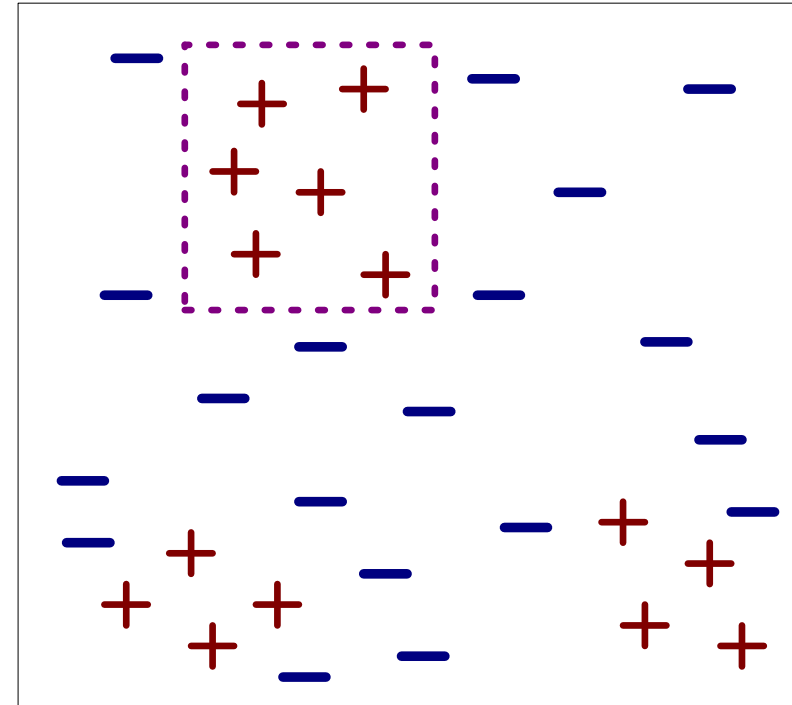
RIPPER uses sequential covering to learn a rule list for each class.

1. Start from an empty rule list
2. Grow a rule that covers as many positive examples as possible and is rather accurate
3. Remove training records covered by the rule
4. Repeat steps 2 and 3 until stopping criterion is met

Example of Sequential Covering ...

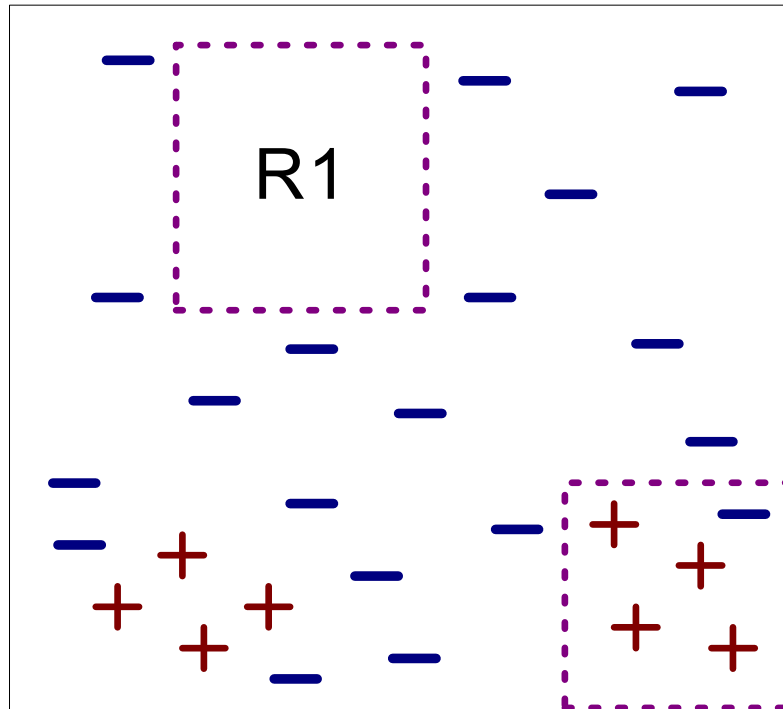


(i) Original Data

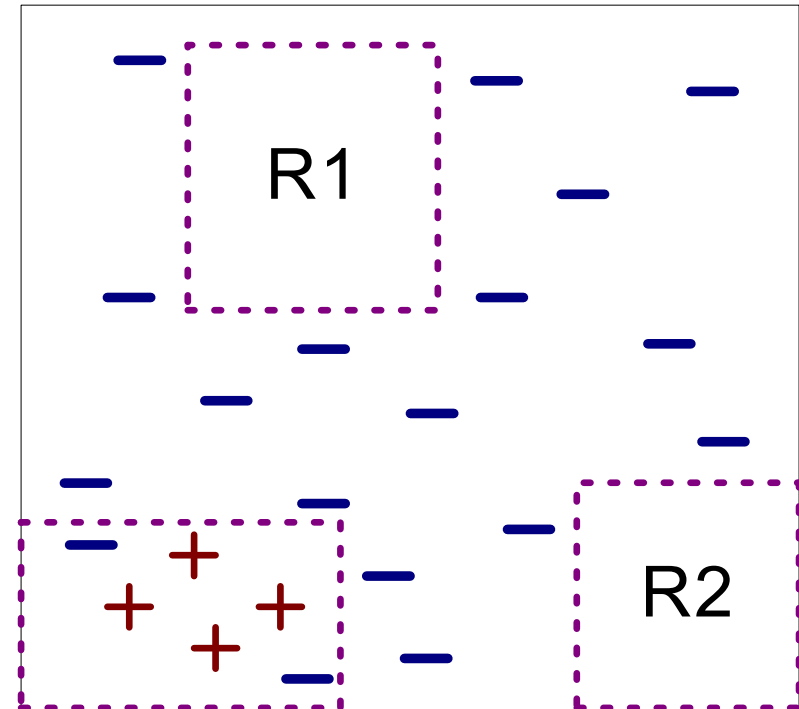


(ii) Step 1

Example of Sequential Covering



(iii) Step 2



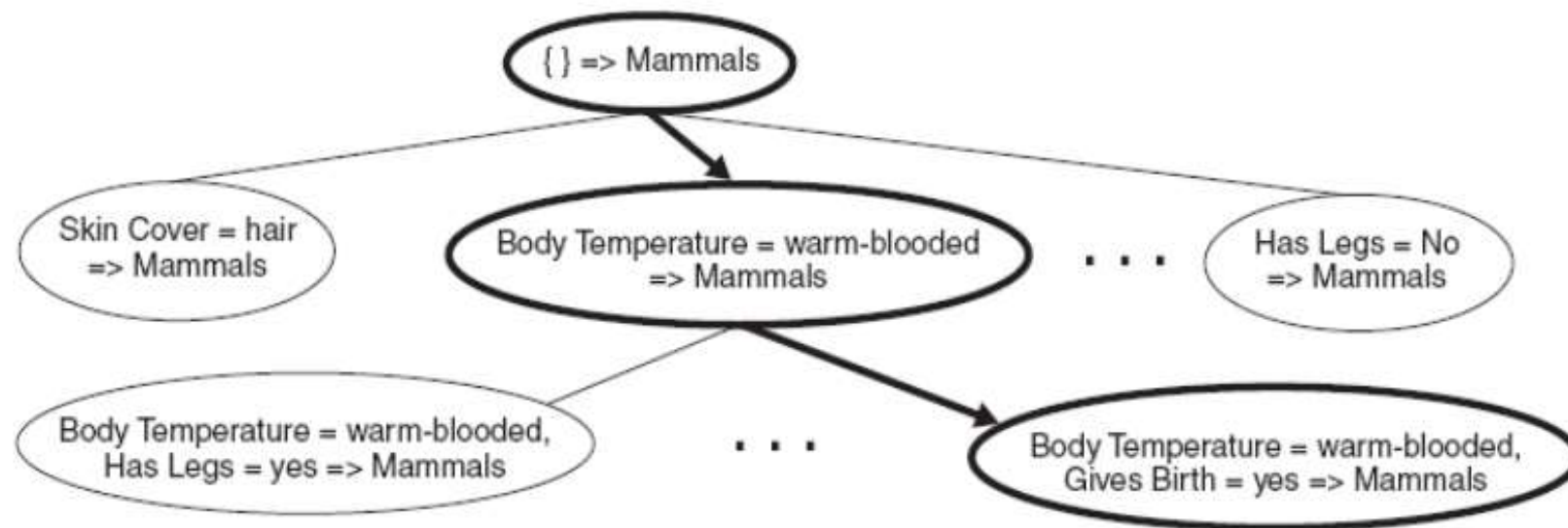
(iv) Step 3

Aspects of Sequential Covering

1. Rule Growing
2. Rule Pruning
3. Instance Elimination
4. Stopping Criterion

Rule Growing within the RIPPER Algorithm

- Start from an empty rule: $\{\} \rightarrow \text{class}$
- Step by step add conjuncts so that
 1. the accuracy of the rule improves
 2. the rule still covers many examples



Rule Growing Procedure

- Goal: Prefer rules with high accuracy and high support count
- Add conjunct that maximizes **FOIL's information gain measure**
 - $R_0: \{\} \rightarrow \text{class}$ (initial rule)
 - $R_1: \{A\} \rightarrow \text{class}$ (rule after adding conjunct)
- Stop when rule no longer covers negative examples

$$\text{Gain}(R_0, R_1) = p_1 [\log_2 (p_1/(p_1+n_1)) - \log_2 (p_0/(p_0 + n_0))]$$

where

p_1 : number of positive instances covered by R_1

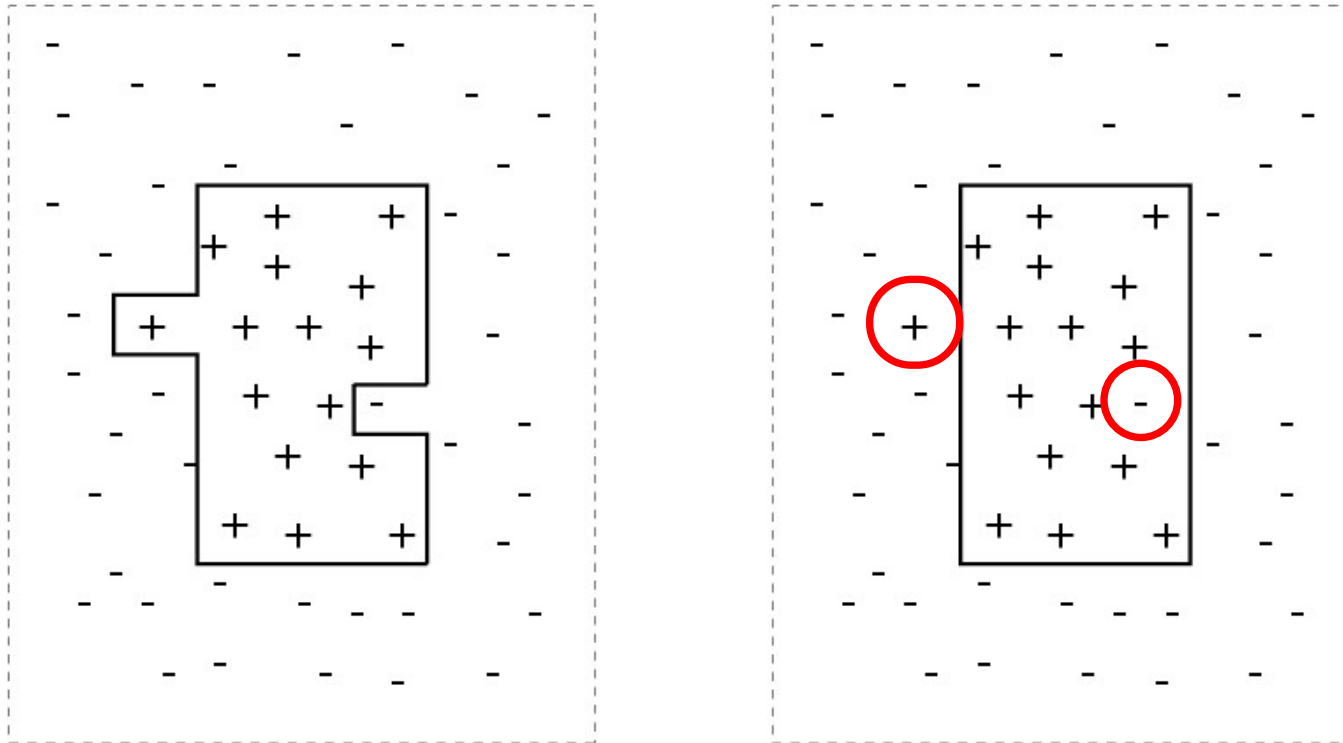
n_1 : number of negative instances covered by R_1

p_0 : number of positive instances covered by R_0

n_0 : number of negative instances covered by R_0

Rule Pruning

- Because of the stopping criterion, the learned rule is likely to overfit the data



- Thus, the rule is pruned afterwards using a validation dataset

Rule Pruning Procedure

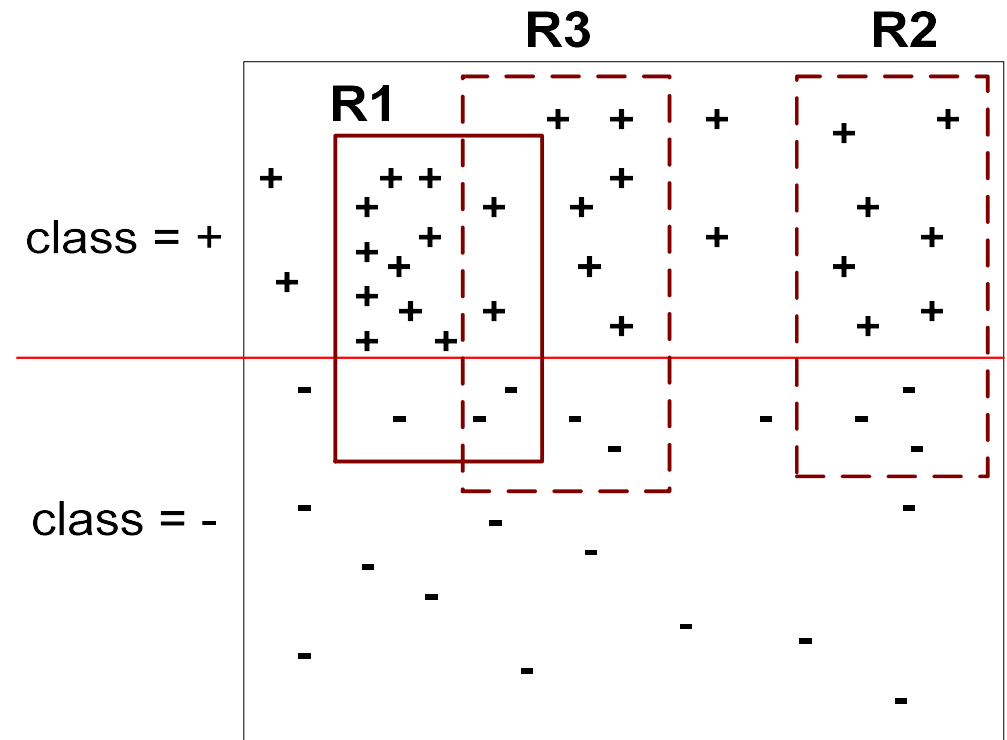
- Goal: Decrease generalization error of the rule
- Procedure
 1. remove one of the conjuncts in the rule
 2. compare error rates on a validation dataset before and after pruning
 3. if error improves, prune the conjunct
- Measure for pruning

$$v = (p - n) / (p + n)$$

p: number of positive examples covered by the rule in the validation set
n: number of negative examples covered by the rule in the validation set

Instance Elimination

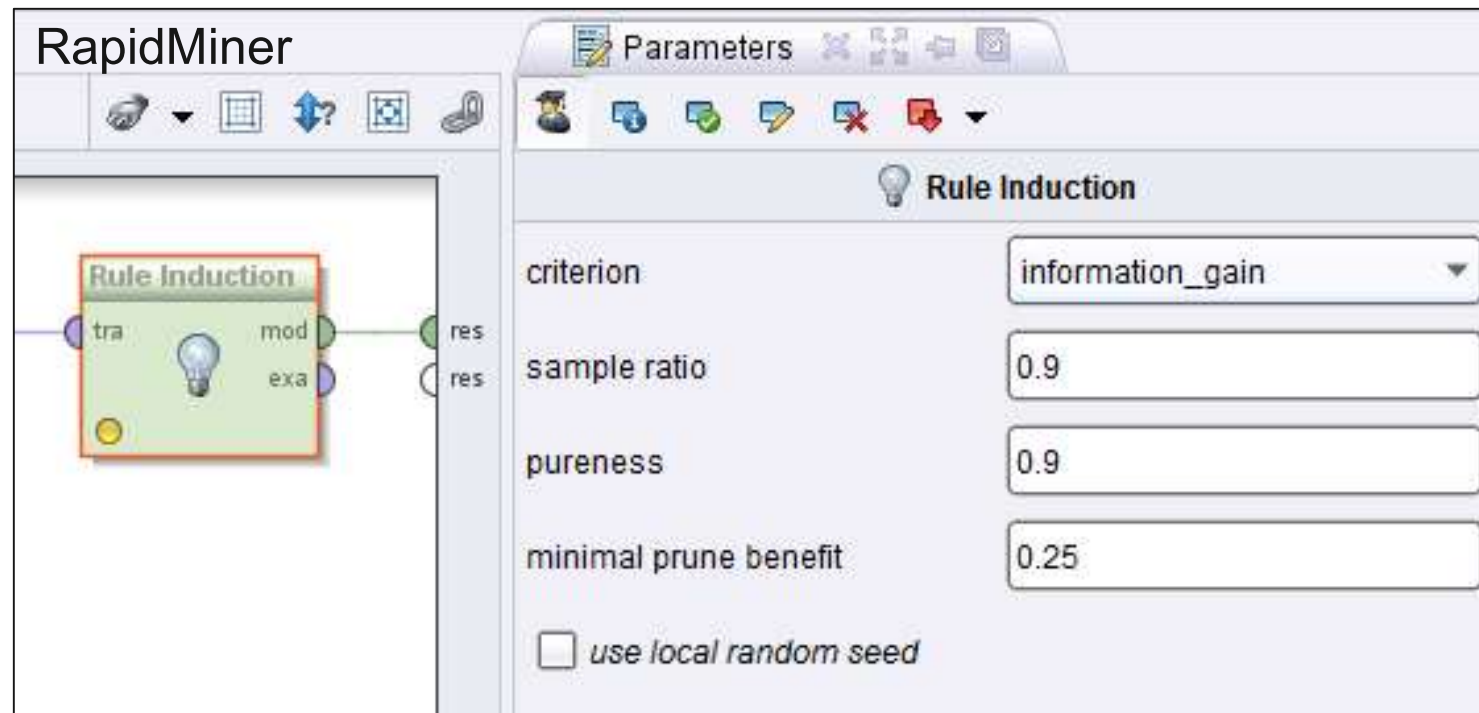
- Why do we remove positive instances?
 - otherwise, the next rule is identical to previous rule
- Why do we remove negative instances?
 - prevent underestimating accuracy of rule
 - compare rules R2 and R3 in the diagram
 - 3 errors vs. 2 errors



Stopping Criterion

- When to stop adding new rules to the rule set?
- RIPPER
 - error rate of new rule on validation set must not exceed 50%
 - minimum description length should not increase more than d bits

RIPPER in RapidMiner and Python



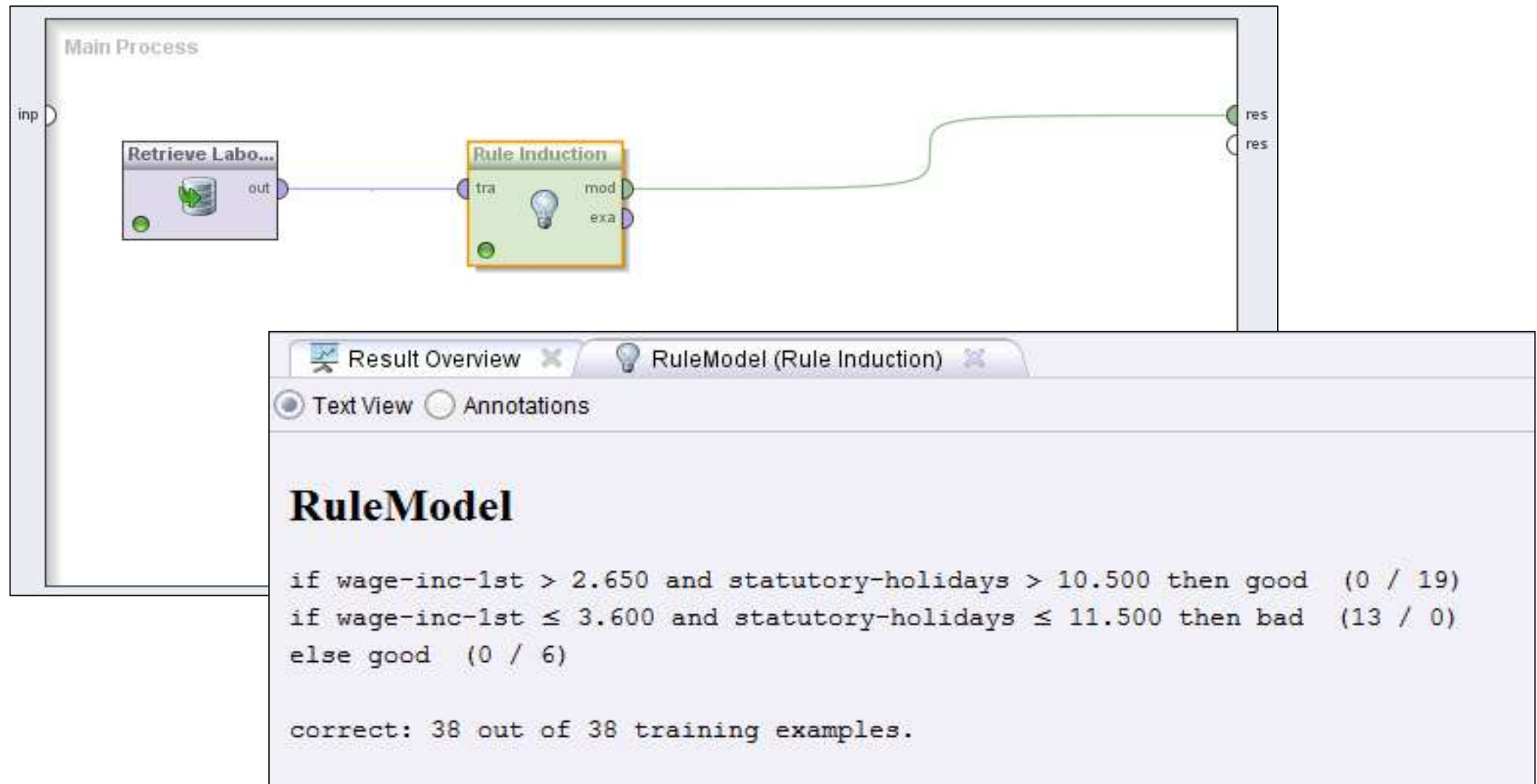
```
In [ ]: # Import rule learner (requires pip install wittgenstein)
import wittgenstein as lw

#Train classifier
ripper_clf = lw.RIPPER()
ripper_clf.fit(training_data, training_labels)
```

Python

<https://github.com/imoscovitz/wittgenstein>

RIPPER in RapidMiner



Advantages of Rule-based Classifiers

- Easy to interpret for humans
 - classification decisions are explainable
 - eager learning
- Performance comparable to decision trees
- Ignore irrelevant attributes (automated feature selection)
- Can classify unseen instances rapidly
- Are well suited to handle imbalanced data sets
 - as they learn rules for the minority class first

Literature for this Slideset

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

Chapter 3.6: Model Evaluation

Chapter 6.11: Class Imbalance Problem

Chapter 6.2: Rule-Based Classifiers

