Data Mining I
Classification, Part 3

Heiko Paulheim, Volha Bryl, Robert Meusel, Oliver Lehmberg
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 Classifier

- Classify records by using a collection of “if…then…” rules

- Rule: \((Condition) \rightarrow y\)
  - where
    - \(Condition\) is a conjunction of attributes
    - \(y\) is the class label
  - \(LHS\): rule antecedent or condition
  - \(RHS\): rule consequent
  - Examples of classification rules:
    - \((\text{Blood Type}=\text{Warm}) \land (\text{Lay Eggs}=\text{Yes}) \rightarrow \text{Birds}\)
    - \((\text{Taxable Income} < 50\text{K}) \land (\text{Refund}=\text{Yes}) \rightarrow \text{Evade}=\text{No}\)
Rule-based Classifier (Example)

R1: (Give Birth = no) \(\land\) (Can Fly = yes) \(\rightarrow\) Birds
R2: (Give Birth = no) \(\land\) (Live in Water = yes) \(\rightarrow\) Fishes
R3: (Give Birth = yes) \(\land\) (Blood Type = warm) \(\rightarrow\) Mammals
R4: (Give Birth = no) \(\land\) (Can Fly = no) \(\rightarrow\) Reptiles
R5: (Live in Water = sometimes) \(\rightarrow\) Amphibians
Application of Rule-Based Classifier

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

\[
\begin{align*}
&\text{R1: (Give Birth = no) \land (Can Fly = yes) \rightarrow Birds} \\
&\text{R2: (Give Birth = no) \land (Live in Water = yes) \rightarrow Fishes} \\
&\text{R3: (Give Birth = yes) \land (Blood Type = warm) \rightarrow Mammals} \\
&\text{R4: (Give Birth = no) \land (Can Fly = no) \rightarrow Reptiles} \\
&\text{R5: (Live in Water = sometimes) \rightarrow Amphibians}
\end{align*}
\]

<table>
<thead>
<tr>
<th>Name</th>
<th>Blood Type</th>
<th>Give Birth</th>
<th>Can Fly</th>
<th>Live in Water</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>hawk</td>
<td>warm</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>?</td>
</tr>
<tr>
<td>grizzly bear</td>
<td>warm</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>?</td>
</tr>
</tbody>
</table>

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 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%

<table>
<thead>
<tr>
<th>Tid</th>
<th>Refund</th>
<th>Marital Status</th>
<th>Taxable Income</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Yes</td>
<td>Single</td>
<td>125K</td>
<td>No</td>
</tr>
<tr>
<td>2</td>
<td>No</td>
<td>Married</td>
<td>100K</td>
<td>No</td>
</tr>
<tr>
<td>3</td>
<td>No</td>
<td>Single</td>
<td>70K</td>
<td>No</td>
</tr>
<tr>
<td>4</td>
<td>Yes</td>
<td>Married</td>
<td>120K</td>
<td>No</td>
</tr>
<tr>
<td>5</td>
<td>No</td>
<td>Divorced</td>
<td>95K</td>
<td>Yes</td>
</tr>
<tr>
<td>6</td>
<td>No</td>
<td>Married</td>
<td>60K</td>
<td>No</td>
</tr>
<tr>
<td>7</td>
<td>Yes</td>
<td>Divorced</td>
<td>220K</td>
<td>No</td>
</tr>
<tr>
<td>8</td>
<td>No</td>
<td>Single</td>
<td>85K</td>
<td>Yes</td>
</tr>
<tr>
<td>9</td>
<td>No</td>
<td>Married</td>
<td>75K</td>
<td>No</td>
</tr>
<tr>
<td>10</td>
<td>No</td>
<td>Single</td>
<td>90K</td>
<td>Yes</td>
</tr>
</tbody>
</table>
How does a Rule-based Classifier Work?

R1: (Give Birth = no) \( \land \) (Can Fly = yes) \( \rightarrow \) Birds
R2: (Give Birth = no) \( \land \) (Live in Water = yes) \( \rightarrow \) Fishes
R3: (Give Birth = yes) \( \land \) (Blood Type = warm) \( \rightarrow \) Mammals
R4: (Give Birth = no) \( \land \) (Can Fly = no) \( \rightarrow \) Reptiles
R5: (Live in Water = sometimes) \( \rightarrow \) Amphibians

<table>
<thead>
<tr>
<th>Name</th>
<th>Blood Type</th>
<th>Give Birth</th>
<th>Can Fly</th>
<th>Live in Water</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>lemur</td>
<td>warm</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>?</td>
</tr>
<tr>
<td>turtle</td>
<td>cold</td>
<td>no</td>
<td>no</td>
<td>sometimes</td>
<td>?</td>
</tr>
<tr>
<td>dogfish shark</td>
<td>cold</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>?</td>
</tr>
</tbody>
</table>

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 Classifier

• 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

• 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
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

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

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Blood Type</th>
<th>Give Birth</th>
<th>Can Fly</th>
<th>Live in Water</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>turtle</td>
<td>cold</td>
<td>no</td>
<td>no</td>
<td>sometimes</td>
<td>?</td>
</tr>
</tbody>
</table>

R1: (Give Birth = no) \(\land\) (Can Fly = yes) \(\rightarrow\) Birds
R2: (Give Birth = no) \(\land\) (Live in Water = yes) \(\rightarrow\) Fishes
R3: (Give Birth = yes) \(\land\) (Blood Type = warm) \(\rightarrow\) Mammals
R4: (Give Birth = no) \(\land\) (Can Fly = no) \(\rightarrow\) Reptiles
R5: (Live in Water = sometimes) \(\rightarrow\) Amphibians
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 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

<table>
<thead>
<tr>
<th>Temperature</th>
<th>Outlook</th>
<th>Humidity</th>
<th>Windy</th>
<th>Play Golf?</th>
</tr>
</thead>
<tbody>
<tr>
<td>hot</td>
<td>sunny</td>
<td>high</td>
<td>false</td>
<td>no</td>
</tr>
<tr>
<td>hot</td>
<td>sunny</td>
<td>high</td>
<td>true</td>
<td>no</td>
</tr>
<tr>
<td>hot</td>
<td>overcast</td>
<td>high</td>
<td>false</td>
<td>yes</td>
</tr>
<tr>
<td>cool</td>
<td>rain</td>
<td>normal</td>
<td>false</td>
<td>yes</td>
</tr>
<tr>
<td>cool</td>
<td>overcast</td>
<td>normal</td>
<td>true</td>
<td>yes</td>
</tr>
<tr>
<td>mild</td>
<td>sunny</td>
<td>high</td>
<td>false</td>
<td>no</td>
</tr>
<tr>
<td>cool</td>
<td>sunny</td>
<td>normal</td>
<td>false</td>
<td>yes</td>
</tr>
<tr>
<td>mild</td>
<td>rain</td>
<td>normal</td>
<td>false</td>
<td>yes</td>
</tr>
<tr>
<td>mild</td>
<td>sunny</td>
<td>normal</td>
<td>true</td>
<td>yes</td>
</tr>
<tr>
<td>mild</td>
<td>overcast</td>
<td>high</td>
<td>true</td>
<td>yes</td>
</tr>
<tr>
<td>hot</td>
<td>overcast</td>
<td>normal</td>
<td>false</td>
<td>yes</td>
</tr>
<tr>
<td>mild</td>
<td>rain</td>
<td>high</td>
<td>true</td>
<td>no</td>
</tr>
<tr>
<td>cool</td>
<td>rain</td>
<td>normal</td>
<td>true</td>
<td>no</td>
</tr>
<tr>
<td>mild</td>
<td>rain</td>
<td>high</td>
<td>false</td>
<td>yes</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>IF</th>
<th>THEN</th>
</tr>
</thead>
<tbody>
<tr>
<td>T=hot AND H=high AND O=overcast AND W=false</td>
<td>yes</td>
</tr>
<tr>
<td>T=cool AND H=normal AND O=rain AND W=false</td>
<td>yes</td>
</tr>
<tr>
<td>T=cool AND H=normal AND O=overcast AND W=true</td>
<td>yes</td>
</tr>
<tr>
<td>T=cool AND H=normal AND O=sunny AND W=false</td>
<td>yes</td>
</tr>
<tr>
<td>T=mild AND H=normal AND O=rain AND W=false</td>
<td>yes</td>
</tr>
<tr>
<td>T=mild AND H=normal AND O=sunny AND W=true</td>
<td>yes</td>
</tr>
<tr>
<td>T=mild AND H=high AND O=overcast AND W=true</td>
<td>yes</td>
</tr>
<tr>
<td>T=hot AND H=normal AND O=overcast AND W=false</td>
<td>yes</td>
</tr>
<tr>
<td>T=mild AND H=high AND O=rain AND W=false</td>
<td>yes</td>
</tr>
</tbody>
</table>

• The solution is
  – a set of rules
  – that is complete and consistent on the training examples

• but it does not generalize well to new examples
  – plus: danger of overfitting
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
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

<table>
<thead>
<tr>
<th></th>
<th>predicted +</th>
<th>predicted -</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>class +</td>
<td>( p ) (true positives)</td>
<td>( P-p ) (false negatives)</td>
<td>( P )</td>
</tr>
<tr>
<td>class -</td>
<td>( n ) (false positives)</td>
<td>( N-n ) (true negatives)</td>
<td>( N )</td>
</tr>
<tr>
<td></td>
<td>( p + n )</td>
<td>( P+N - (p+n) )</td>
<td>( P+N )</td>
</tr>
</tbody>
</table>
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 error (noise in the data)
    - in rule learning: each example is a rule

- Such concepts do not generalize well!
  → 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
    \((relax\ \text{consistency}\ \text{constraint})\)
  • decide when to stop adding rules to a theory
    \((relax\ \text{completeness}\ \text{constraint})\)
  – efficient but not accurate

Rule set with three rules á 3, 2, and 2 conditions

Pre-pruning decisions
Pre-Pruning Heuristics

1. **Thresholding** a heuristic value
   - require a certain minimum value of the search heuristic
   - e.g.: Precision > 0.8.

2. Foil's **Minimum Description Length** Criterion
   - the length of the theory plus the exceptions (misclassified examples) must be shorter than the length of the examples by themselves
   - lengths are measured in bits (information content)

3. CN2's **Significance Test**
   - tests whether the distribution of the examples covered by a rule deviates significantly from the distribution of the examples in the entire training set
   - if not, discard the rule
Pruning: Removing Rules

- By coverage:
  - remove rules that do not cover a significant amount of examples

- By support:
  - remove rules that do not cover a significant amount of positive examples

- By confidence:
  - remove rules that are not precise enough
  - i.e., the heuristic is below a certain threshold
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 $H=\text{high}$ AND $O=\text{sunny}$ THEN no

IF $O=\text{rain}$ AND $W=\text{true}$ 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 concept
  – REP: $O(n^4)$ I-REP: $O(n \log^2 n)$

• Subsequently used in RIPPER rule learner (Cohen, 1995)
RIPPER in RapidMiner

RuleModel

if wage-inc-1st > 2.650 and statutory-holidays > 10.500 then good (0 / 19)
if wage-inc-1st ≤ 3.600 and statutory-holidays ≤ 11.500 then bad (13 / 0)
else good (0 / 6)

correct: 38 out of 38 training examples.
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
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)
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
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
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)
• 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

![Graph showing decision boundaries]
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., 53 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)

<table>
<thead>
<tr>
<th>Person</th>
<th>Employed</th>
<th>Owns House</th>
<th>Balanced Account</th>
<th>Get Credit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peter Smith</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Julia Miller</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Stephen Baker</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Mary Fisher</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Kim Hanson</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>John Page</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>
Example: Credit Rating

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

- Not nicely expressible in trees and rule sets
  - as we know them (attribute-value conditions)

<table>
<thead>
<tr>
<th>Person</th>
<th>Employed</th>
<th>Owns House</th>
<th>Balanced Account</th>
<th>Get Credit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peter Smith</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Julia Miller</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Stephen Baker</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Mary Fisher</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Kim Hanson</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>John Page</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>
**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
Output $Y$ is 1 if at least two of the three inputs are equal to 1.
Artificial Neural Networks (ANN)

\[ Y = I(0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4 > 0) \]

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

\[
Y = I\left(\sum_i w_i X_i - t\right) \text{ or } Y = \text{sign}\left(\sum_i w_i X_i - t\right)
\]
General Structure of ANN

Training ANN means learning the weights of the neurons.
Algorithm for learning ANN

• Initialize the weights \((w_0, w_1, \ldots, w_k)\), 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
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

- Dealing with non-linear separation
  - transform data in higher dimensional space
  - where there is a linear separation
  - details: higher mathematics (refer to books if you are interested)
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

![Diagram of SVM process in RapidMiner]

- SVM type: C-SVC
- Kernel type: rbf
- Gamma: 0.0
- C: 0.0
- Cache size: 80
- Epsilon: 0.001
- Confidence for multiclass

[Diagram showing the SVM process and parameters]
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
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?
  – playing around in RapidMiner
  – 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

ParameterSet

Parameter set:

Performance:
PerformanceVector [
-----accuracy: 83.87%
ConfusionMatrix:
True: Rock Mine
Rock: 23 3
Mine: 7 29
]
k-NN.k = 5
k-NN.measure_types = BregmanDivergences
Questions?