More Data Mining with Weka

Class 4 – Lesson 1

Attribute selection using the “wrapper” method

Ian H. Witten

Department of Computer Science
University of Waikato
New Zealand

weka.waikato.ac.nz
Lesson 4.1: Attribute selection using the “wrapper” method

Class 1 Exploring Weka’s interfaces; working with big data

Class 2 Discretization and text classification

Class 3 Classification rules, association rules, and clustering

Class 4 Selecting attributes and counting the cost

Class 5 Neural networks, learning curves, and performance optimization

Lesson 4.1 “Wrapper” attribute selection

Lesson 4.2 The Attribute Selected Classifier

Lesson 4.3 Scheme-independent selection

Lesson 4.4 Attribute selection using ranking

Lesson 4.5 Counting the cost

Lesson 4.6 Cost-sensitive classification
Lesson 4.1: Attribute selection using the “wrapper” method

Fewer attributes, better classification

- **Data Mining with Weka, Lesson 1.5**
  - Open glass.arff; run J48 (trees>J48): cross-validation classification accuracy 67%
  - Remove all attributes except RI and Mg: 69%
  - Remove all attributes except RI, Na, Mg, Ca, Ba: 74%

- **“Select attributes” panel avoids laborious experimentation**
  - Open glass.arff; attribute evaluator WrapperSubsetEval
    select J48, 10-fold cross-validation, threshold = –1
  - Search method: BestFirst; select Backward
  - Get the same attribute subset: RI, Na, Mg, Ca, Ba: “merit” 0.74

- **How much experimentation?**
  - Set searchTermination = 1
  - Total number of subsets evaluated 36
    complete set (1 evaluation); remove one attribute (9); one more (8); one more (7); one more (6); plus one more (5) to check that removing a further attribute does not yield an improvement; 1+9+8+7+6+5 = 36
Lesson 4.1: Attribute selection using the “wrapper” method

Searching
- Exhaustive search: $2^9 = 512$ subsets
- Searching forward, searching backward
  + when to stop? (searchTermination)

0 attributes (ZeroR)

all 9 attributes
Lesson 4.1: Attribute selection using the “wrapper” method

Trying different searches (WrapperSubsetEval folds = 10, threshold = −1)

- Backwards (searchTermination = 1): RI, Mg, K, Ba, Fe (0.72)
  - searchTermination = 5 or more: RI, Na, Mg, Ca, Ba (0.74)
- Forwards: RI, Al, Ca (0.70)
  - searchTermination = 2 or more: RI, Na, Mg, Al, K, Ca (0.72)
- Bi-directional: RI, Al, Ca (0.70)
  - searchTermination = 2 or more: RI, Na, Mg, Al (0.74)

- Note: local vs global optimum
  - searchTermination > 1 can traverse a valley
- Al is the best single attribute to use (as OneR will confirm)
  - thus forwards search results include Al
- (curiously) Al is the best single attribute to drop
  - thus backwards search results do not include Al
Lesson 4.1: Attribute selection using the “wrapper” method

Cross-validation

Backward (searchTermination=5)

<table>
<thead>
<tr>
<th>number of folds (%)</th>
<th>attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 (100 %)</td>
<td>1 RI</td>
</tr>
<tr>
<td>8 (80 %)</td>
<td>2 Na</td>
</tr>
<tr>
<td>10 (100 %)</td>
<td>3 Mg</td>
</tr>
<tr>
<td>3 (30 %)</td>
<td>4 Al</td>
</tr>
<tr>
<td>2 (20 %)</td>
<td>5 Si</td>
</tr>
<tr>
<td>2 (20 %)</td>
<td>6 K</td>
</tr>
<tr>
<td>7 (70 %)</td>
<td>7 Ca</td>
</tr>
<tr>
<td>10 (100 %)</td>
<td>8 Ba</td>
</tr>
<tr>
<td>4 (40 %)</td>
<td>9 Fe</td>
</tr>
</tbody>
</table>

Definitely choose RI, Mg, Ba; probably Na, Ca; probably not Al, Si, K, Fe

But if we did forward search, would definitely choose Al!
Lesson 4.1: Attribute selection using the “wrapper” method

Gory details
(generally, Weka methods follow descriptions in the research literature)

- **WrapperSubsetEval** attribute evaluator
  - Default: 5-fold cross-validation
  - Does at least 2 and up to 5 cross-validation runs and takes average accuracy
  - Stops when the standard deviation across the runs is less than the user-specified threshold times the mean (default: 1% of the mean)
  - Setting a negative threshold forces a single cross-validation

- **BestFirst** search method
  - `searchTermination` defaults to 5 for traversing valleys

- Choose **ClassifierSubsetEval** to use the wrapper method, but with a separate test set instead of cross-validation
Lesson 4.1: Attribute selection using the “wrapper” method

- Use a classifier to find a good attribute set (“scheme-dependent”)
  - we used J48; in the associated Activity you will use ZeroR, OneR, IBk
- Wrap a classifier in a cross-validation loop
- Involves both an Attribute Evaluator and a Search Method
- Searching can be greedy forward, backward, or bidirectional
  - computationally intensive; $m^2$ for $m$ attributes
  - there’s also has an “exhaustive” search method ($2^m$), used in the Activity
- Greedy searching finds a local optimum in the search space
  - you can traverse valleys by increasing the searchTermination parameter

Course text
- Section 7.1 Attribute selection
More Data Mining with Weka

Class 4 – Lesson 2

The Attribute Selected Classifier

Ian H. Witten

Department of Computer Science
University of Waikato
New Zealand

weka.waikato.ac.nz
Lesson 4.2: The Attribute Selected Classifier

Class 1  Exploring Weka’s interfaces; working with big data

Lesson 4.1 “Wrapper” attribute selection

Class 2  Discretization and text classification

Lesson 4.2 The Attribute Selected Classifier

Class 3  Classification rules, association rules, and clustering

Lesson 4.3 Scheme-independent selection

Class 4  Selecting attributes and counting the cost

Lesson 4.4 Attribute selection using ranking

Lesson 4.5 Counting the cost

Class 5  Neural networks, learning curves, and performance optimization

Lesson 4.6 Cost-sensitive classification
Lesson 4.2: The Attribute Selected Classifier

- Select attributes and apply a classifier to the result
  - glass.arff  default parameters everywhere
    - J48  67%
  - Wrapper selection with J48  {RI, Mg, Al, K, Ba}
    - J48  71%
    - IBk  78%
  - with IBk  {RI, Mg, Al, K, Ca, Ba}
    - IBk  71%

- Is this cheating? – yes!

- AttributeSelectedClassifier (in meta)
  - Select attributes based on training data only
    - ... then train the classifier and evaluate it on the test data
  - like the FilteredClassifier used for supervised discretization (Lesson 2.2)
  - Use AttributeSelectedClassifier to wrap J48
    - J48  72%
    - IBk  74%
  - Use AttributeSelectedClassifier to wrap IBk
    - IBk  69%
    - IBk  71%
      (slightly surprising)
Lesson 4.2: The Attribute Selected Classifier

- **Check the effectiveness of the AttributeSelectedClassifier**
  - `diabetes.arff` 76.3%
  - `AttributeSelectedClassifier, NaiveBayes, WrapperSubsetEval, NaiveBayes` 75.7%

- **Add copies of an attribute**
  - Copy the first attribute (preg); NaiveBayes 75.7%
  - `AttributeSelectedClassifier as above` 75.7%
  - Add 9 further copies of preg; NaiveBayes 68.9%
  - `AttributeSelectedClassifier as above` 75.7%
  - Add further copies: NaiveBayes even worse
  - `AttributeSelectedClassifier as above` 75.7%

- Attribute selection does a good job of removing redundant attributes
Lesson 4.2: The Attribute Selected Classifier

- AttributeSelectedClassifier selects based on training set only
  - even when cross-validation is used for evaluation
  - this is the right way to do it!
  - we used J48; in the associated Activity you will use ZeroR, OneR, IBk
- (probably) Best to use the same classifier within the wrapper
  - e.g. wrap J48 to select attributes for J48
- One-off experiments in the Explorer may not be reliable
  - the associated Activity uses the Experimenter for more repetition

Course text
- Section 7.1 Attribute selection
More Data Mining with Weka

Class 4 – Lesson 3

Scheme-independent attribute selection

Ian H. Witten

Department of Computer Science
University of Waikato
New Zealand

weka.waikato.ac.nz
Lesson 4.3: Scheme-independent attribute selection

Class 1  Exploring Weka's interfaces; working with big data

Class 2  Discretization and text classification

Class 3  Classification rules, association rules, and clustering

Class 4  Selecting attributes and counting the cost

Class 5  Neural networks, learning curves, and performance optimization

Lesson 4.1 “Wrapper” attribute selection

Lesson 4.2 The Attribute Selected Classifier

Lesson 4.3 Scheme-independent selection

Lesson 4.4 Attribute selection using ranking

Lesson 4.5 Counting the cost

Lesson 4.6 Cost-sensitive classification
Lesson 4.3: Scheme-independent attribute selection

Wrapper method is simple and direct – but slow
❖ Either:
   1. use a single-attribute evaluator, with ranking (*Lesson 4.4*)
      – *can eliminate* irrelevant attributes
   2. combine an attribute subset evaluator with a search method
      – *can eliminate* redundant attributes as well
❖ We’ve already looked at search methods (*Lesson 4.1*)
   – greedy forward, backward, bidirectional
❖ Attribute subset evaluators
   – wrapper methods are *scheme-dependent* attribute subset evaluators
   – other subset evaluators are *scheme-independent*
**Lesson 4.3: Scheme-independent attribute selection**

**CfsSubsetEval**: a scheme-independent attribute subset evaluator

- An attribute subset is good if the attributes it contains are
  - *highly correlated with the class attribute*
  - *not strongly correlated with one another*

- Goodness of an attribute subset =
  \[
  \frac{\sum_{\text{all attributes } x} C(x, \text{class})}{\sqrt{\sum_{\text{all attributes } x} \sum_{\text{all attributes } y} C(x, y)}}
  \]

- \( C \) measures the correlation between two attributes
- An entropy-based metric called the “symmetric uncertainty” is used
Lesson 4.3: Scheme-independent attribute selection

Compare CfsSubsetEval with Wrapper selection on ionosphere.arff

- No attribute selection  
  83% 86% 91%

- With attribute selection (using AttributeSelectedClassifier)
  - CfsSubsetEval (very fast)  
    89% 89% 92%
  - Wrapper selection (very slow)  
    91% 89% 90%

*(the corresponding classifier is used in the wrapper, e.g. the wrapper for IBk uses IBk)*

- Conclusion: CfsSubsetEval is nearly as good as Wrapper, and much faster
Lesson 4.3: Scheme-independent attribute selection

Attribute subset evaluators in Weka

Scheme-dependent
- WrapperSubsetEval (internal cross-validation)
- ClassifierSubsetEval (separate held-out test set)

Scheme-independent
- CfsSubsetEval
  - consider predictive value of each attribute, along with the degree of inter-redundancy
- ConsistencySubsetEval
  - measures consistency in class values of training set with respect to the attributes
  - seek the smallest attribute set whose consistency is no worse than for the full set

(There are also meta-evaluators, which incorporate other operations)
Lesson 4.3: Scheme-independent attribute selection

- Attribute subset selection involves
  - a subset evaluation measure
  - a search method

- Some measures are scheme-dependent
  - e.g. the wrapper method; but very slow

- ... and others are scheme-independent
  - e.g. CfsSubsetEval; quite fast

- Even faster ... single-attribute evaluator, with ranking (next lesson)

Course text
- Section 7.1 Attribute selection
More Data Mining with Weka

Class 4 – Lesson 4

Fast attribute selection using ranking

Ian H. Witten

Department of Computer Science
University of Waikato
New Zealand

weka.waikato.ac.nz
Lesson 4.4: Fast attribute selection using ranking
Lesson 4.4: Fast attribute selection using ranking

- Attribute subset selection involves:
  - subset evaluation measure
  - search method
- Searching is slow!

- Alternative: use a single-attribute evaluator, with ranking
  - can eliminate irrelevant attributes
    ... but not redundant attributes
- Choose the “ranking” search method when selecting a single-attribute evaluator
Lesson 4.4: Fast attribute selection using ranking

Metrics for evaluating attributes: we’ve seen some before
- OneR uses the accuracy of a single-attribute classifier
- C4.5 (i.e. J48) uses information gain
  ... actually, it uses gain ratio
- CfsSubsetEval uses “symmetric uncertainty”

The “ranker” search method sorts attributes according to their evaluation
- parameters
  - number of attributes to retain (default: retain all)
  - or discard attributes whose evaluation falls below a threshold (default: −∞)
  - can specify a set of attributes to ignore
Lesson 4.3: Scheme-independent attribute selection

Compare GainRatioAttributeEval with others on ionosphere.arff

- No attribute selection
  - NaiveBayes 83%
  - IBk 86%
  - J48 91%

- With attribute selection (using AttributeSelectedClassifier)
  - CfsSubsetEval (very fast)
    - NaiveBayes 89%
    - IBk 89%
    - J48 92%
  - Wrapper selection (very slow)
    - NaiveBayes 91%
    - IBk 89%
    - J48 90%

  (the corresponding classifier is used in the wrapper, e.g. the wrapper for IBk uses IBk)

- GainRatioAttributeEval, retaining 7 attributes
  - NaiveBayes 90%
  - IBk 86%
  - J48 91%

- Lightning fast ...
  - but performance is sensitive to the number of attributes retained
Lesson 4.4: Fast attribute selection using ranking

Attribute evaluators in Weka
- OneRAttributeEval
- InfoGainAttributeEval
- GainRatioAttributeEval
- SymmetricalUncertaintyAttributeEval

plus
- ChiSquaredAttributeEval – compute the $\chi^2$ statistic of each attribute wrt the class
- SVMAttributeval – use SVM to determine the value of attributes
- ReliefFAttributeEval – instance-based attribute evaluator
- PrincipalComponents – principal components transform, choose largest eigenvectors
- LatentSemanticAnalysis – performs latent semantic analysis and transformation

(There are also meta-evaluators, which incorporate other operations)
Lesson 4.4: Fast attribute selection using ranking

- Attribute subset evaluation
  - involves searching and is bound to be slow

- Single-attribute evaluation
  - involves ranking, which is far faster
  - difficult to specify a suitable number of attributes to retain
    (involves experimentation)
  - does not cope with redundant attributes
    (e.g. copies of an attribute will be repeatedly selected)

- Many single-attribute evaluators are based on ML methods

Course text
- Section 7.1 Attribute selection
More Data Mining with Weka

Class 4 – Lesson 5

Counting the cost

Ian H. Witten

Department of Computer Science
University of Waikato
New Zealand

weka.waikato.ac.nz
Lesson 4.5: Counting the cost

Class 1 Exploring Weka's interfaces; working with big data

Lesson 4.1 “Wrapper” attribute selection

Class 2 Discretization and text classification

Lesson 4.2 The Attribute Selected Classifier

Class 3 Classification rules, association rules, and clustering

Lesson 4.3 Scheme-independent selection

Class 4 Selecting attributes and counting the cost

Lesson 4.4 Attribute selection using ranking

Lesson 4.5 Counting the cost

Class 5 Neural networks, learning curves, and performance optimization

Lesson 4.6 Cost-sensitive classification
Lesson 4.5: Counting the cost

What is success?

- So far, the classification rate
  *(measured by test set, holdout, cross-validation)*
- Different kinds of error may have different costs
- Minimizing total errors is inappropriate
  *With 2-class classification, the ROC curve summarizes different tradeoffs*
- Credit dataset **credit-g.arff**
  *It’s worse to class a customer as good when they are bad than to class a customer as bad when they are good*
- Economic model: error cost of 5 vs. 1
**Lesson 4.5: Counting the cost**

**Weka: Cost-sensitive evaluation**

- **Credit dataset** `credit-g.arff`
- **J48 (70%)**

- **Classify Panel “More options”: Cost-sensitive evaluation**
  
  ```
<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>&lt;-- classified as</th>
</tr>
</thead>
<tbody>
<tr>
<td>588</td>
<td>112</td>
<td>a = good</td>
</tr>
<tr>
<td>183</td>
<td>117</td>
<td>b = bad</td>
</tr>
</tbody>
</table>
  
  cost: $183 \times 5 + 112 \times 1$
  
  = 1027 (1.027/instance)
  ```

- **Baseline (ZeroR)**
  
  ```
<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>&lt;-- classified as</th>
</tr>
</thead>
<tbody>
<tr>
<td>700</td>
<td>0</td>
<td>a = good</td>
</tr>
<tr>
<td>300</td>
<td>0</td>
<td>b = bad</td>
</tr>
</tbody>
</table>
  
  cost: $300 \times 5 = 1500$
  ```

- if you were to classify everything as *bad* the total cost would be only 700
Lesson 4.5: Counting the cost

Weka: cost-sensitive classification

- The classifier should know the costs when learning!
- meta > CostSensitiveClassifier
- Select J48
- Define cost matrix:
  
  \[
  \begin{array}{cc}
  0 & 1 \\
  5 & 0 \\
  \end{array}
  \]
- Worse classification error (61% vs. 70%)
- Lower average cost (0.66 vs. 1.027)
- Effect of error on confusion matrix

\[
\begin{array}{cc|c}
\text{a} & \text{b} & \text{old} \\
588 & 112 & \text{a = good} \\
183 & 117 & \text{b = bad} \\
\end{array} \quad \begin{array}{cc|c}
\text{a} & \text{b} & \text{new} \\
372 & 328 & \text{a = good} \\
66 & 234 & \text{b = bad} \\
\end{array}
\]

- ZeroR: average cost 0.7
Lesson 4.5: Counting the cost

- Is classification accuracy the best measure?
- Economic model: cost of errors
  - or consider the tradeoff between error rates – the ROC curve
- Cost-sensitive evaluation
- Cost-sensitive classification
- meta > CostSensitiveClassifier
  - makes any classifier cost-sensitive

- Section 5.7 Counting the cost
More Data Mining with Weka

Class 4 – Lesson 6

Cost-sensitive classification vs. cost-sensitive learning

Ian H. Witten

Department of Computer Science
University of Waikato
New Zealand

weka.waikato.ac.nz
Lesson 4.6: Cost-sensitive classification vs. cost-sensitive learning

Class 1  Exploring Weka's interfaces; working with big data

Class 2  Discretization and text classification

Class 3  Classification rules, association rules, and clustering

Class 4  Selecting attributes and counting the cost

Class 5  Neural networks, learning curves, and performance optimization

Lesson 4.1 “Wrapper” attribute selection
Lesson 4.2 The Attribute Selected Classifier
Lesson 4.3 Scheme-independent selection
Lesson 4.4 Attribute selection using ranking
Lesson 4.5 Counting the cost
Lesson 4.6 Cost-sensitive classification
Lesson 4.6: Cost-sensitive classification vs. cost-sensitive learning

Making a classifier cost-sensitive: Method 1: Cost-sensitive classification

Adjust a classifier’s output by recalculating the probability threshold

- Credit dataset credit-g.arff
- NaiveBayes, Output predictions

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>p_{good}</th>
</tr>
</thead>
<tbody>
<tr>
<td>605</td>
<td>95</td>
<td>0.999</td>
</tr>
<tr>
<td>151</td>
<td>149</td>
<td>0.999</td>
</tr>
</tbody>
</table>

Threshold: 0.5
- predicts 756 good, with 151 mistakes
- 244 bad, with 95 mistakes
Lesson 4.6: Cost-sensitive classification vs. cost-sensitive learning

Recalculating the probability threshold

- Cost matrix
  
  \[
  \begin{array}{cc}
  a & b \\
  0 & 1 \\
  5 & 0
  \end{array}
  \]

  \(a = \text{good} \)  \(b = \text{bad}\)

- Threshold = 5/6 = 0.833

- General cost matrix:

<table>
<thead>
<tr>
<th>actual</th>
<th>predicted</th>
<th>(p_{\text{good}})</th>
</tr>
</thead>
<tbody>
<tr>
<td>good</td>
<td>good</td>
<td>0.999</td>
</tr>
<tr>
<td>good</td>
<td>good</td>
<td>0.991</td>
</tr>
<tr>
<td>good</td>
<td>good</td>
<td>0.983</td>
</tr>
<tr>
<td>good</td>
<td>good</td>
<td>0.975</td>
</tr>
<tr>
<td>good</td>
<td>good</td>
<td>0.965</td>
</tr>
<tr>
<td>good</td>
<td>bad</td>
<td>0.951</td>
</tr>
<tr>
<td>bad</td>
<td>good</td>
<td>0.934</td>
</tr>
<tr>
<td>good</td>
<td>good</td>
<td>0.917</td>
</tr>
<tr>
<td>good</td>
<td>good</td>
<td>0.896</td>
</tr>
<tr>
<td>good</td>
<td>good</td>
<td>0.873</td>
</tr>
<tr>
<td>good</td>
<td>good</td>
<td>0.836</td>
</tr>
<tr>
<td>good</td>
<td>good</td>
<td>0.776</td>
</tr>
<tr>
<td>bad</td>
<td>good</td>
<td>0.715</td>
</tr>
<tr>
<td>good</td>
<td>good</td>
<td>0.663</td>
</tr>
<tr>
<td>good</td>
<td>good</td>
<td>0.587</td>
</tr>
<tr>
<td>bad</td>
<td>good</td>
<td>0.508</td>
</tr>
<tr>
<td>good</td>
<td>bad</td>
<td>0.416</td>
</tr>
<tr>
<td>bad</td>
<td>bad</td>
<td>0.297</td>
</tr>
<tr>
<td>good</td>
<td>bad</td>
<td>0.184</td>
</tr>
<tr>
<td>bad</td>
<td>bad</td>
<td>0.04</td>
</tr>
</tbody>
</table>

  - To minimize expected cost, classify as \(\text{good}\) if

  \[p_{\text{good}} > \frac{\mu}{\lambda + \mu}\]
Lesson 4.6: Cost-sensitive classification vs. cost-sensitive learning

What about methods that don’t produce probabilities?

- They (almost) all do 😊
- J48 with minNumObj = 100 (to get small tree)
- from tree,
  \[1 - \frac{37}{108} = 0.657, \frac{68}{166} = 0.410, 1 - \frac{44}{152} = 0.711, \text{etc}\]
- Other methods (e.g. rules) are similar
Lesson 4.6: Cost-sensitive classification vs. cost-sensitive learning

CostSensitiveClassifier with minimizeExpectedCost = true

- Credit dataset credit-g.arff; J48
- Cost matrix

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a = good</th>
<th>b = bad</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Cost 1027

- meta > CostSensitiveClassifier; minimizeExpectedCost = true; set cost matrix

- select J48

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a = good</th>
<th>b = bad</th>
</tr>
</thead>
<tbody>
<tr>
<td>455</td>
<td>245</td>
<td></td>
<td></td>
</tr>
<tr>
<td>105</td>
<td>195</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Cost 770

- use bagging (Data Mining with Weka, Lesson 4.6)

  ... J48 produces a restricted set of probs

- bagged J48

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a = good</th>
<th>b = bad</th>
</tr>
</thead>
<tbody>
<tr>
<td>367</td>
<td>333</td>
<td></td>
<td></td>
</tr>
<tr>
<td>54</td>
<td>246</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Cost 603
Cost-sensitive classification adjusts the output of a classifier
Cost-sensitive learning learns a different classifier
Create a new dataset with some instances replicated
To simulate the cost matrix

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>a = good</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>b = bad</td>
</tr>
</tbody>
</table>

add 4 copies of every bad instance

Dataset credit-g has 700 good and 300 bad instances (1000)
⇒ new version has 700 good and 1500 bad (2200)

... and re-learn!

In practice, re-weight the instances, don’t copy them
Lesson 4.6: Cost-sensitive classification vs. cost-sensitive learning

Cost-sensitive learning in Weka:
CostSensitiveClassifier with minimizeExpectedCost = false (default)

- Credit dataset, cost matrix as before credit-g.arff; J48
- meta > CostSensitiveClassifier; minimizeExpectedCost = false
- NaïveBayes
  | a  | b  | <-- classified as |
  | 445| 255| a = good         |
  | 55 | 245| b = bad          |
  cost 530

- J48
  | a  | b  | <-- classified as |
  | 372| 328| a = good         |
  | 66 | 234| b = bad          |
  cost 658

- bagged J48
  | a  | b  | <-- classified as |
  | 404| 296| a = good         |
  | 57 | 243| b = bad          |
  cost 581
Lesson 4.6: Cost-sensitive classification vs. cost-sensitive learning

- Cost-sensitive classification: adjust a classifier’s output
- Cost-sensitive learning: learn a new classifier
  - by duplicating instances appropriately (inefficient!)
  - or by internally reweighting the original instances
- meta > CostSensitiveClassifier
  - implements both cost-sensitive classification and cost-sensitive learning
- Cost matrix can be stored and loaded automatically
  - e.g. german-credit.cost
- Section 5.7 Counting the cost
More Data Mining with Weka

Department of Computer Science
University of Waikato
New Zealand

Creative Commons Attribution 3.0 Unported License
creativecommons.org/licenses/by/3.0/

weka.waikato.ac.nz