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

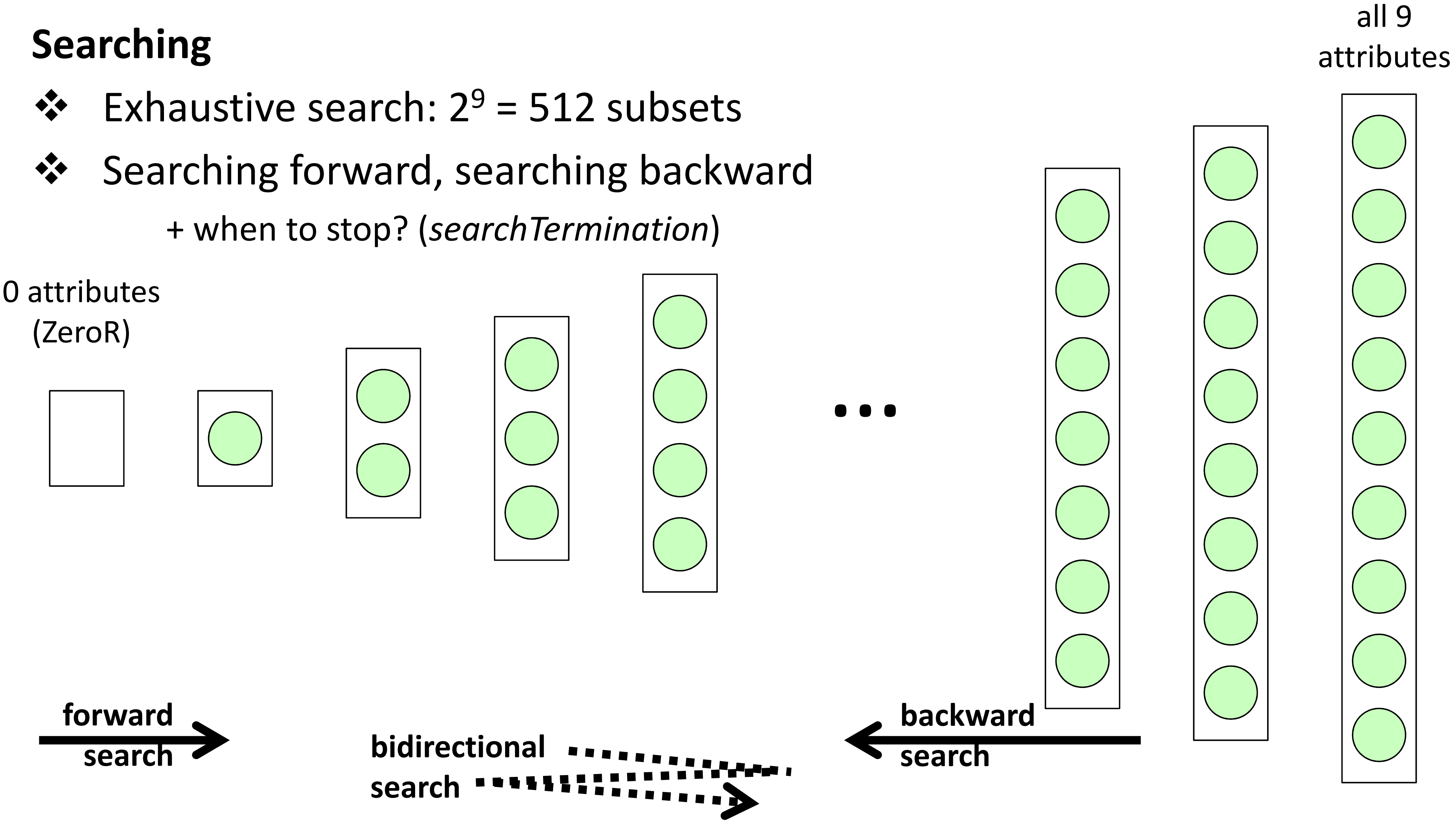
- **Data Mining with Weka, Lesson 1.5**
  - *Open glass.arff; run J48 (trees>J48): cross-validation classification accuracy 67%*
  - *Remove all attributes except RI and Mg: 69%*
  - *Remove all attributes except RI, Na, Mg, Ca, Ba: 74%*
- **"Select attributes" panel avoids laborious experimentation**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



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)

```
number of folds (%)  attribute
          10(100 %)    1 RI
           8( 80 %)    2 Na
          10(100 %)    3 Mg
           3( 30 %)    4 Al
           2( 20 %)    5 Si
           2( 20 %)    6 K
           7( 70 %)    7 Ca
          10(100 %)    8 Ba
           4( 40 %)    9 Fe
```

In how many folds does that attribute appear in the final subset?

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

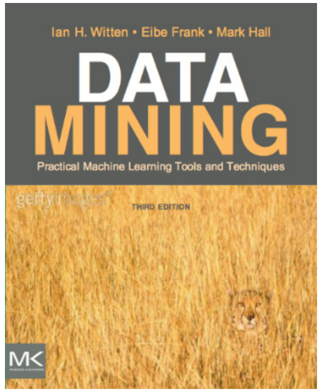
(generally, Weka methods follow descriptions in the research literature)

- *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**
- Class 2 **Discretization and text classification**
- Class 3 **Classification rules, association rules, and clustering**
- Class 4 **Selecting attributes and counting the cost**
  - **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**
- Class 5 **Neural networks, learning curves, and performance optimization**

Lesson 4.2: The Attribute Selected Classifier

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

❖ **Is this cheating? – yes!**

❖ **AttributeSelectedClassifier** (in meta)

	J48	IBk
– *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*	72%	74%
– *Use AttributeSelectedClassifier to wrap IBk*	69%	71%

(slightly surprising)

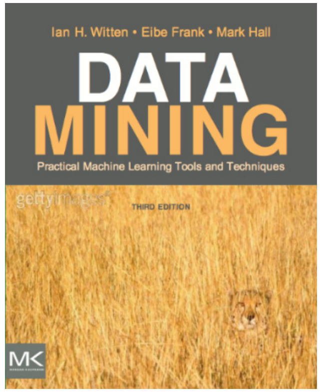
Lesson 4.2: The Attribute Selected Classifier

❖ **Check the effectiveness of the AttributeSelectedClassifier**	**NaiveBayes**
– *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 = $\dfrac{\sum_{\text{all attributes } x} C(x, \text{class})}{\sqrt{\sum_{\text{all attributes } x} \sum_{\text{all attributes } y} C(x, y)}}$
- *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

	NaiveBayes	*IBk*	*J48*
❖ No attribute selection	83%	86%	91%
❖ With attribute selection (using AttributeSelectedClassifier)			
– CfsSubsetEval (very fast)	89%	89%	90%
– Wrapper selection (very slow)	91%	89%	92%

(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)

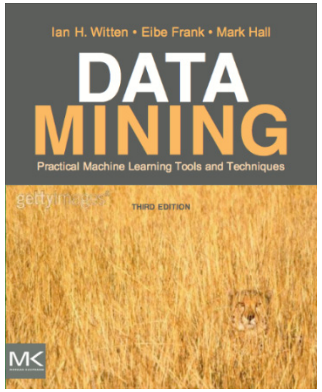
Scheme-independent

- 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

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 4.4: 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 — OneRAttributeEval
- ❖ C4.5 (i.e. J48) uses information gain — InfoGainAttributeEval
  … actually, it uses gain ratio — GainRatioAttributeEval
- ❖ CfsSubsetEval uses "symmetric uncertainty" — SymmetricalUncertAttributeEval

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: $-\infty$)*
  - *can specify a set of attributes to ignore*

Lesson 4.4: Scheme-independent attribute selection

Compare GainRatioAttributeEval with others on ionosphere.arff

	NaiveBayes	*IBk*	*J48*
❖ 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)			
– GainRatioAttributeEval, retaining 7 attributes	90%	90%	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

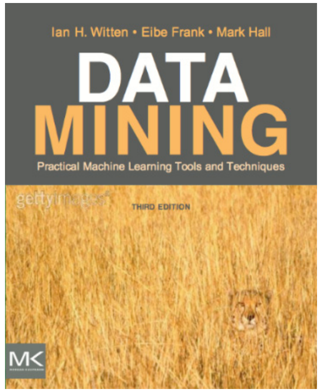
plus

- ChiSquaredAttributeEval – *compute the χ^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**

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 4.5: 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%)

a	b		<-- classified as
588	112	|	a = good
183	117	|	b = bad

cost: 295 incorrectly classified instances

- Classify Panel "More options": Cost-sensitive evaluation

  Cost matrix:

0	1
5	0

cost: 183 × 5 + 112 × 1
= 1027 (1.027/instance)

- Baseline (ZeroR)

a	b		<-- classified as
700	0	|	a = good
300	0	|	b = bad

cost: 300 × 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:

0	1
5	0

- Worse classification error (61% vs. 70%)
- Lower average cost (0.66 vs. 1.027)
- Effect of error on confusion matrix

old

a	b		
588	112	|	a = good
183	117	|	b = bad

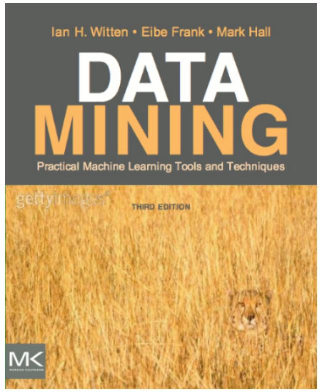
new

a	b		
372	328	|	a = good
66	234	|	b = bad

- 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

```
  a    b   <-- classified as
605   95 |   a = good
151  149 |   b = bad
```

- Threshold: 0.5
  - predicts 756 *good*, with 151 mistakes
  - 244 *bad*, with 95 mistakes

	actual	predicted	p_{good}
0	good	good	0.999
50	good	good	0.991
100	good	good	0.983
150	good	good	0.975
200	good	good	0.965
250	bad	good	0.951
300	bad	good	0.934
350	good	good	0.917
400	good	good	0.896
450	good	good	0.873
500	good	good	0.836
550	good	good	0.776
600	bad	good	0.715
650	good	good	0.663
700	good	good	0.587
750	bad	good	0.508
800	good	bad	0.416
850	bad	bad	0.297
900	good	bad	0.184
950	bad	bad	0.04

Lesson 4.6: Cost-sensitive classification *vs. cost-sensitive* learning

Recalculating the probability threshold

- Cost matrix

a	b		
0	1	|	a = good
5	0	|	b = bad

- Threshold = 5/6 = 0.833

a	b		<-- classified as
448	252	|	a = good
53	247	|	b = bad

total cost 517 (vs. 850)

- General cost matrix: $\begin{matrix} 0 & \lambda \\ \mu & 0 \end{matrix}$

- To minimize expected cost, classify as *good* if

$$p_{good} > \frac{\mu}{\lambda + \mu}$$

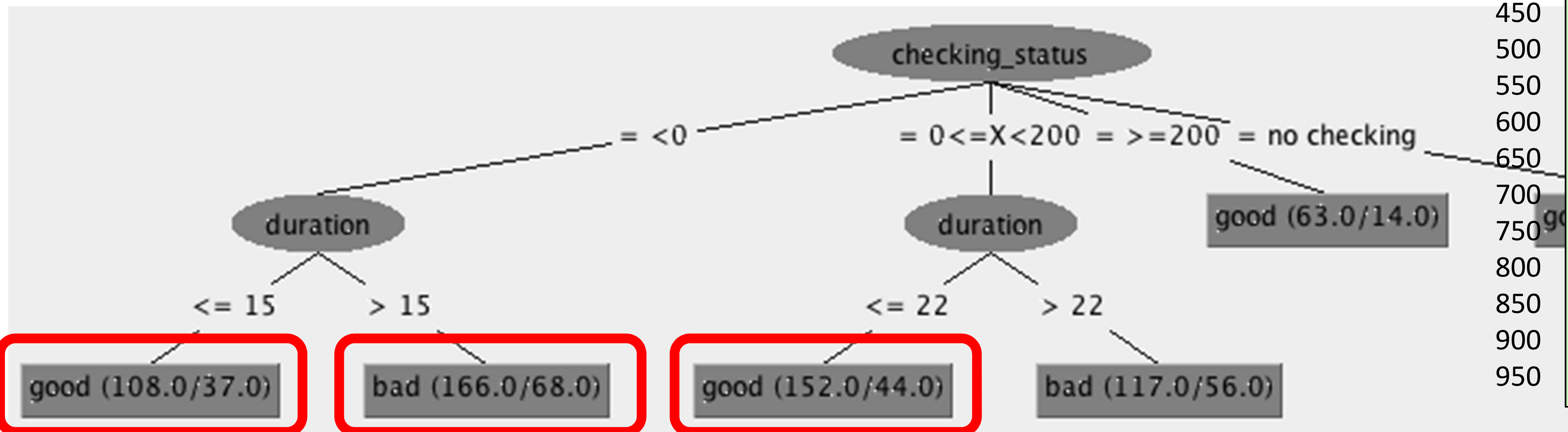
	actual	predicted	p_{good}
0	good	good	0.999
50	good	good	0.991
100	good	good	0.983
150	good	good	0.975
200	good	good	0.965
250	bad	good	0.951
300	bad	good	0.934
350	good	good	0.917
400	good	good	0.896
450	good	good	0.873
500	good	good	0.836
550	good	good	0.776
600	bad	good	0.715
650	good	good	0.663
700	good	good	0.587
750	bad	good	0.508
800	good	bad	0.416
850	bad	bad	0.297
900	good	bad	0.184
950	bad	bad	0.04

What about methods that don’t produce probabilities?

- They (almost) all do ☺
- J48 with minNumObj = 100 (to get small tree)
- from tree,

  1 – 37/108 = 0.657, 68/166=0.410, 1 – 44/152 = 0.711, etc
- Other methods (e.g. rules) are similar

	actual	predicted	p_{good}
0	good	good	0.883
50	good	good	0.883
100	good	good	0.883
150	good	good	0.883
200	good	good	0.883
250	good	good	0.883
300	good	good	0.883
350	good	good	0.883
400	good	good	0.778
450	bad	good	0.778
500	bad	good	0.711
550	good	good	0.711
600	good	good	0.711
650	good	good	0.657
700	bad	good	0.657
750	good	bad	0.479
800	good	bad	0.479
850	bad	bad	0.410
900	good	bad	0.410
950	bad	bad	0.410



Lesson 4.6: Cost-sensitive classification *vs. cost-sensitive* learning

CostSensitiveClassifier with minimizeExpectedCost = true

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

a	b		
0	1	|	a = good
5	0	|	b = bad

a	b		<-- classified as
588	112	|	a = good
183	117	|	b = bad

cost 1027

- meta > CostSensitiveClassifier; minimizeExpectedCost = true; set cost matrix
- select J48

a	b		<-- classified as
455	245	|	a = good
105	195	|	b = bad

cost 770

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

  … *J48 produces a restricted set of probs*

- bagged J48

a	b		<-- classified as
367	333	|	a = good
54	246	|	b = bad

cost 603

Lesson 4.6: Cost-sensitive classification *vs. cost-sensitive* learning

Method 2: Cost-sensitive *learning*

- 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

a	b		
0	1	|	a = good
5	0	|	b = bad

- 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

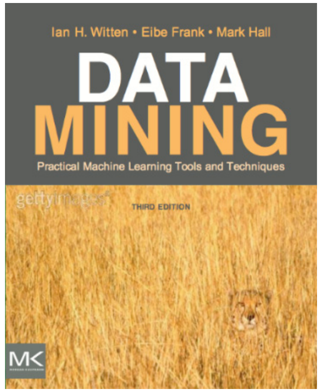
- bagged J48

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

weka.waikato.ac.nz