Evaluation: the key to success

- How predictive is the model we learned?
- Error on the training data is *not* a good indicator of performance on future data
  - Otherwise 1-NN would be the optimum classifier!
- Simple solution that can be used if lots of (labeled) data is available:
  - Split data into training and test set
- However: (labeled) data is usually limited
  - More sophisticated techniques need to be used

Issues in evaluation

- Statistical reliability of estimated differences in performance ($\rightarrow$ significance tests)
- Choice of performance measure:
  - Number of correct classifications
  - Accuracy of probability estimates
  - Error in numeric predictions
- Costs assigned to different types of errors
  - Many practical applications involve costs

Training and testing I

- Natural performance measure for classification problems: *error rate*
  - *Success*: instance's class is predicted correctly
  - *Error*: instance's class is predicted incorrectly
  - Error rate: proportion of errors made over the whole set of instances
- *Resubstitution error:* error rate obtained from the training data
- Resubstitution error is (hopelessly) optimistic!

Training and testing II

- *Test set*: set of independent instances that have played no part in formation of classifier
    - Assumption: both training data and test data are representative samples of the underlying problem
- Test and training data may differ in nature
    - Example: classifiers built using customer data from two different towns *A* and *B*
        - To estimate performance of classifier from town *A* in completely new town, test it on data from *B*

A note on parameter tuning

- It is important that the test data is not used *in any way* to create the classifier
- Some learning schemes operate in two stages:
  - Stage 1: builds the basic structure
  - Stage 2: optimizes parameter settings
- The test data can't be used for parameter tuning!
- Proper procedure uses *three* sets: *training data*, *validation data*, and *test data*
  - Validation data is used to optimize parameters

Making the most of the data

- Once evaluation is complete, *all the data* can be used to build the final classifier
- Generally, the larger the training data the better the classifier (but returns diminish)
- The larger the test data the more accurate the error estimate
- *Holdout* procedure: method of splitting original data into training and test set
  - Dilemma: ideally we want both, a large training and a large test set

Predicting performance

- Assume the estimated error rate is 25%. How close is this to the true error rate?
  - Depends on the amount of test data
- Prediction is just like tossing a biased (!) coin
  - “Head” is a “success”, “tail” is an “error”
- In statistics, a succession of independent events like this is called a *Bernoulli process*
  - Statistical theory provides us with confidence intervals for the true underlying proportion!

Confidence intervals

- We can say: *p* lies within a certain specified interval with a certain specified confidence
- Example: *S*=750 successes in *N*=1000 trials
  - Estimated success rate: 75%
  - How close is this to true success rate *p*?
    - Answer: with 80% confidence $p \in [73.2, 76.7]$
- Another example: *S*=75 and *N*=100
  - Estimated success rate: 75%
  - With 80% confidence $p \in [69.1, 80.1]$

Mean and variance

- Mean and variance for a Bernoulli trial: p, $p(1-p)$
- Expected success rate $f=S/N$
- Mean and variance for f: p, $p(1-p)/N$
- For large enough N, f follows a normal distribution
- c% confidence interval $[-z \le X \le z]$ for random variable with 0 mean is given by: $\Pr[-z \le X \le z] = c$
- Given a symmetric distribution:

$$\Pr[-z \le X \le z] = 1 - (2 * \Pr[X \ge z])$$

Confidence limits

- Confidence limits for the normal distribution with 0 mean and a variance of 1:

$\Pr[X \geq z]$	z
0.1%	3.09
0.5%	2.58
1%	2.33
5%	1.65
10%	1.28
20%	0.84
40%	0.25

- Thus: $\Pr[-1.65 \leq X \leq 1.65] = 90\%$
- To use this we have to reduce our random variable *f* to have 0 mean and unit variance

Transforming *f*

- Transformed value for *f*: $\dfrac{f-p}{\sqrt{p(1-p)/N}}$

  (i.e. subtract the mean and divide by the *standard deviation*)
- Resulting equation: $\Pr\left[-z \le \dfrac{f-p}{\sqrt{p(1-p)/N}} \le z\right] = c$

- Solving for *p*:

$$p = \left(f + \frac{z^2}{2N} \pm z\sqrt{\frac{f}{N} - \frac{f^2}{N} + \frac{z^2}{4N^2}} \right) \Bigg/ \left(1 + \frac{z^2}{N} \right)$$

Examples

- f=75%, N=1000, c=80% (so that z=1.28):

$$p \in [0.732, 0.767]$$

- f=75%, N=100, c=80% (so that z=1.28):

$$p \in [0.691, 0.801]$$

- Note that normal distribution assumption is only valid for large N (i.e. N > 100)
- f=75%, N=10, c=80% (so that z=1.28):

$$p \in [0.549, 0.881]$$

should be taken with a grain of salt

Holdout estimation

- What shall we do if the amount of data is limited?
- The *holdout* method reserves a certain amount for testing and uses the remainder for training
  - Usually: one third for testing, the rest for training
- Problem: the samples might not be representative
  - Example: class might be missing in the test data
- Advanced version uses *stratification*
  - Ensures that each class is represented with approximately equal proportions in both subsets

Repeated holdout method

- 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 (possibly with stratificiation)
  - The error rates on the different iterations are averaged to yield an overall error rate
- This is called the *repeated holdout* method
- Still not optimum: the different test set overlap
  - Can we prevent overlapping?

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 cross-validation*
- Often the subsets are stratified before the cross-validation is performed
- The error estimates are averaged to yield an overall error estimate

More on cross-validation

- Standard method for evaluation: stratified ten-fold cross-validation
- Why ten? Extensive experiments have shown that this is the best choice to get an accurate estimate
  - There is also some theoretical evidence for this
- Stratification reduces the estimate's variance
- Even better: repeated stratified cross-validation
  - E.g. ten-fold cross-validation is repeated ten times and results are averaged (reduces the variance)

Leave-one-out cross-validation

- Leave-one-out cross-validation is a particular form of cross-validation:
  - The number of folds is set to the number of training instances
  - I.e., a classifier has to be built *n* times, where *n* is the number of training instances
- Makes maximum use of the data
- No random subsampling involved
- Very computationally expensive (exception: NN)

LOO-CV and stratification

- Another disadvantage of LOO-CV: stratification is not possible
  - It *guarantees* a non-stratified sample because there is only one instance in the test set!
- Extreme example: completely random dataset with two classes and equal proportions for both of them
  - Best inducer predicts majority class (results in 50% on fresh data from this domain)
  - LOO-CV estimate for this inducer will be 100%!

The bootstrap

- CV uses sampling *without replacement*
  - The same instance, once selected, can not be selected again for a particular training/test set
- The *bootstrap* is an estimation method that uses sampling with replacement to form the training set
  - A dataset of *n* instances is sampled *n* times with replacement to form a new dataset of *n* instances
  - This data is used as the training set
  - The instances from the original dataset that don't occur in the new training set are used for testing

The 0.632 bootstrap

- This method is also called the *0.632 bootstrap*
  - A particular instance has a probability of 1-1/*n* of *not* being picked
  - Thus its probability of ending up in the test data is:

$$\left(1-\frac{1}{n}\right)^n \approx e^{-1} = 0.368$$

  - This means the training data will contain approximately 63.2% of the instances

Estimating error with the boostrap

- The error estimate on the test data will be very pessimistic
  - It contains only ~63% of the instances
- Thus it is combined with the resubstitution error:

$$err = 0.632 \cdot e_{\text{test instances}} + 0.368 \cdot e_{\text{training instances}}$$

- The resubstituion error gets less weight than the error on the test data
- Process is repeated several time, with different replacement samples, and the results averaged

More on the bootstrap

- It is probably the best way of estimating performance for very small datasets
- However, it has some problems
  - Consider the random dataset from above
  - A perfect memorizes will achieve 0% resubstitution error and ~50% error on test data
  - Bootstrap estimate for this classifier:

$$err = 0.632 \cdot 50\% + 0.368 \cdot 0\% = 31.6\%$$

  - True expected error: 50%

Counting the costs

- In practice, different types of classification errors often incur different costs
- Examples:
  - Predicting when cows are in heat (“in estrus”)
    - “Not in estrus” correct 97% of the time
  - Loan decisions
  - Oil-slick detection
  - Fault diagnosis
  - Promotional mailing

Taking costs into account

- The *confusion matrix*:

		Predicted class	
		Yes	No
Actual class	Yes	True positive	False negative
	No	False positive	True negative

- There many other types of costs!
  - E.g.: cost of collecting training data

Lift charts

- In practice, costs are rarely known
- Decisions are usually made by comparing possible scenarios
- Example: promotional mailout
  - Situation 1: classifier predicts that 0.1% of all households will respond
  - Situation 2: classifier predicts that 0.4% of the 10000 most promising households will respond
- A *lift chart* allows for a visual comparison

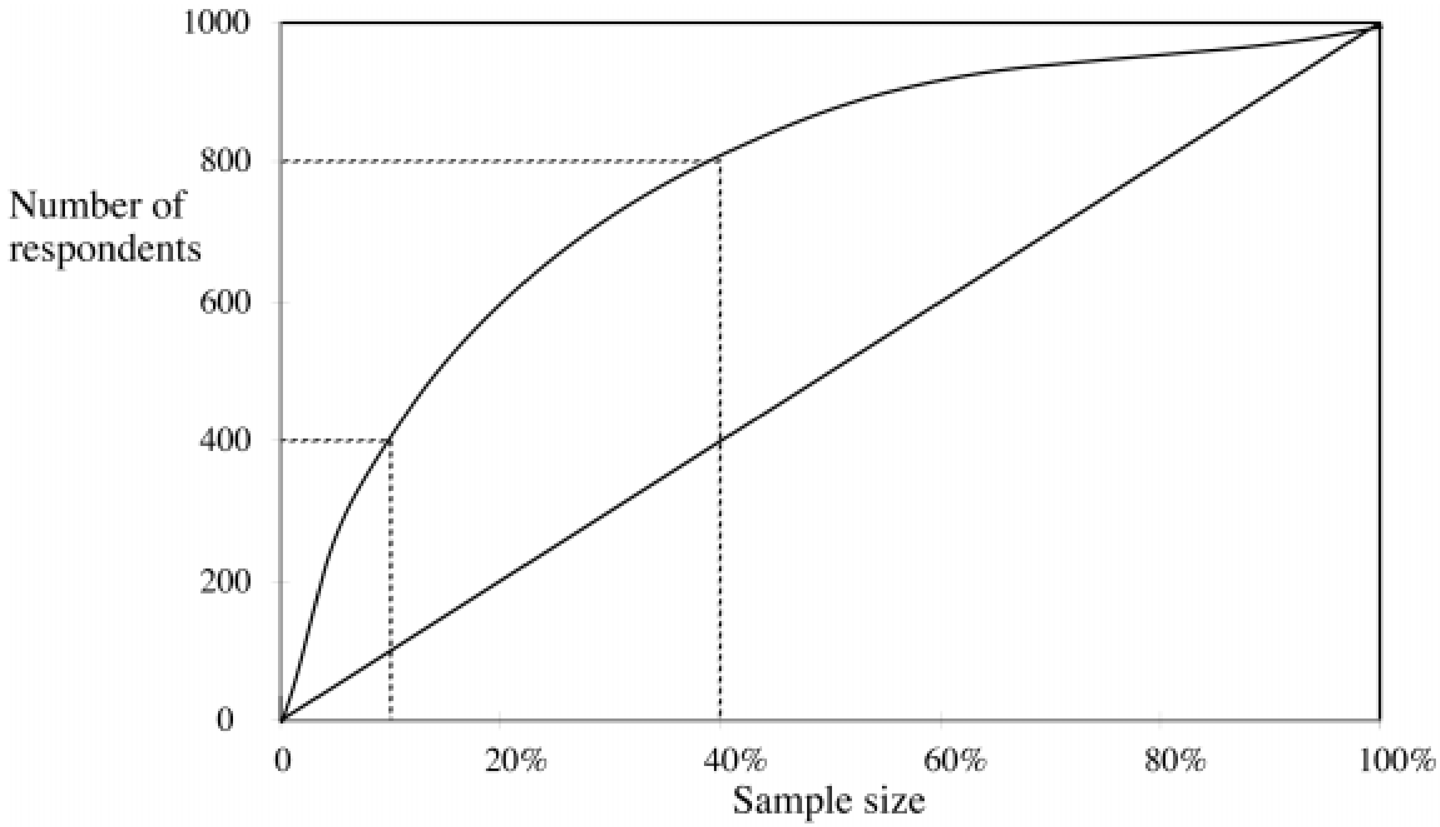
Generating a lift chart

- Instances are sorted according to their predicted probability of being a true positive:

Rank	Predicted probability	Actual class
1	0.95	Yes
2	0.93	Yes
3	0.93	No
4	0.88	Yes
…	…	…

- In lift chart, *x* axis is sample size and *y* axis is number of true positives

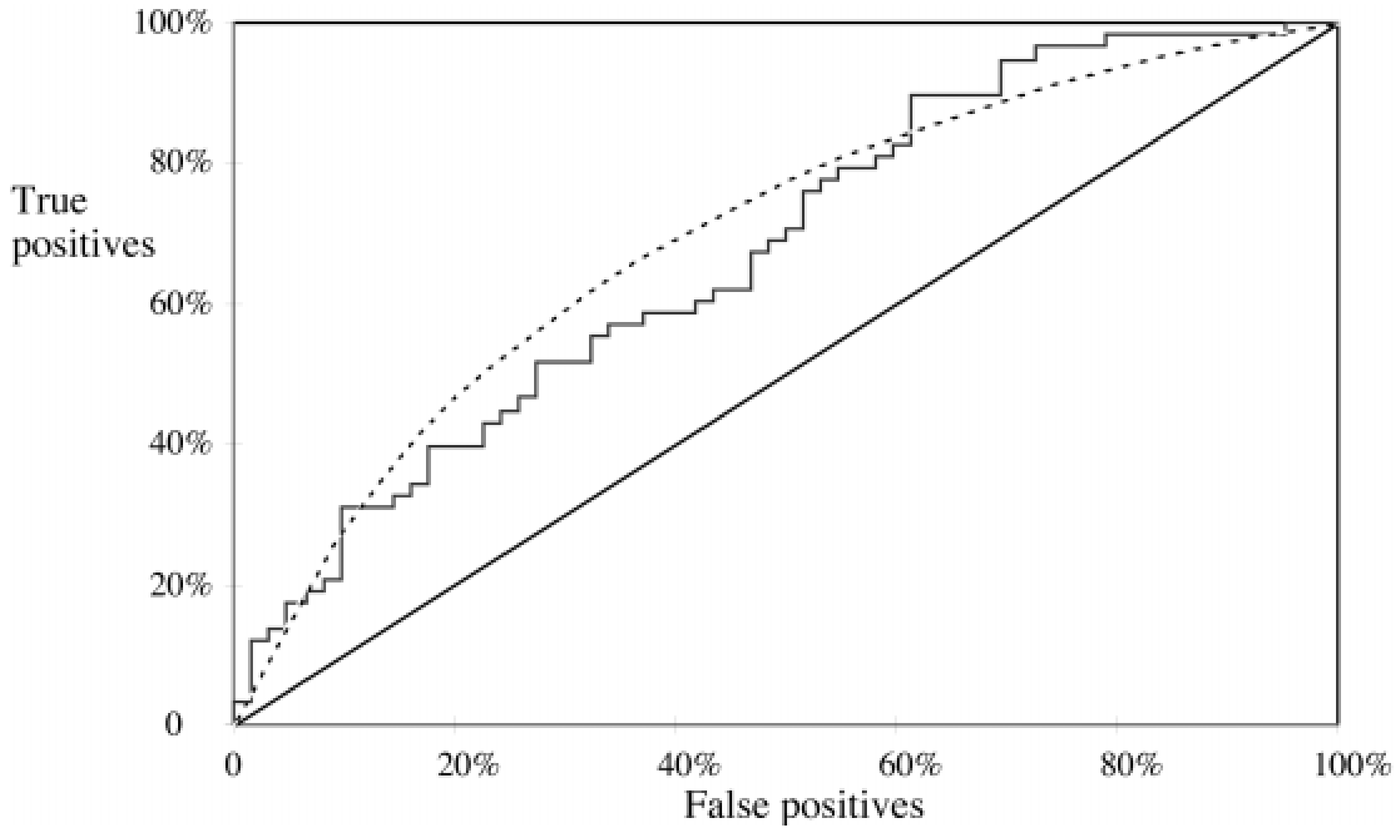
A hypothetical lift chart



ROC curves

- *ROC curves* are similar to lift charts
  - "ROC" stands for "receiver operating characteristic"
  - Used in signal detection to show tradeoff between hit rate and false alarm rate over noisy channel
- Differences to lift chart:
  - *y* axis shows percentage of true positives in sample (rather than absolute number)
  - *x* axis shows percentage of false positives in sample (rather than sample size)

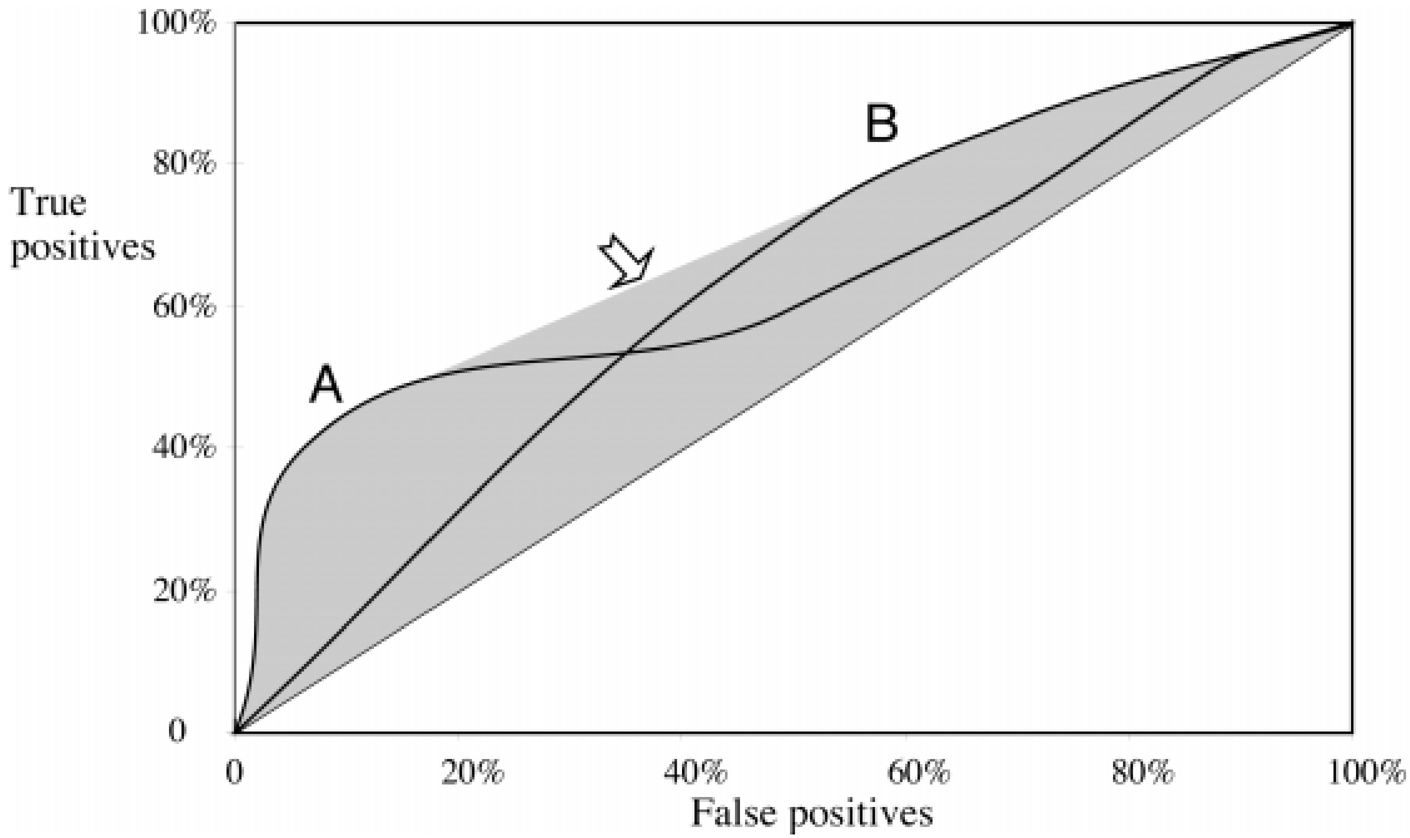
A sample ROC curve



Cross-validation and ROC curves

- Simple method of getting a ROC curve using cross-validation:
  - Collect probabilities for instances in test folds
  - Sort instances according to probabilities
- This method is implemented in WEKA
- However, this is just one possibility
  - The method described in the book generates an ROC curve for each fold and averages them

ROC curves for two schemes



The convex hull

- Given two learning schemes we can achieve any point on the convex hull!
- TP and FP rates for scheme 1: t_1 and f_1
- TP and FP rates for scheme 2: t_2 and f_2
- If scheme 1 is used to predict $100 \times q$% of the cases and scheme 2 for the rest, then we get:
  - TP rate for combined scheme: $q \times t_1 + (1-q) \times t_2$
  - FP rate for combined scheme: $q \times f_2 + (1-q) \times f_2$

Cost-sensitive learning

- Most learning schemes do not perform cost-sensitive learning
  - They generate the same classifier no matter what costs are assigned to the different classes
  - Example: standard decision tree learner
- Simple methods for cost-sensitive learning:
  - Resampling of instances according to costs
  - Weighting of instances according to costs
- Some schemes are inherently cost-sensitive, e.g. naïve Bayes

Measures in information retrieval

- Percentage of retrieved documents that are relevant: *precision*=TP/TP+FP
- Percentage of relevant documents that are returned: *recall* =TP/TP+FN
- Precision/recall curves have hyperbolic shape
- Summary measures: average precision at 20%, 50% and 80% recall (*three-point average recall*)
- *F-measure*=(2×recall×precision)/(recall+precision)

Summary of measures

	Domain	Plot	Explanation
Lift chart	Marketing	TP Subset size	TP (TP+FP)/ (TP+FP+TN+FN)
ROC curve	Communications	TP rate FP rate	TP/(TP+FN) FP/(FP+TN)
Recall-precision curve	Information retrieval	Recall Precision	TP/(TP+FN) TP/(TP+FP)

The MDL principle

- MDL stands for *minimum description length*
- The description length is defined as:

space required to describe a theory

+

space required to describe the theory's mistakes

- In our case the theory is the classifier and the mistakes are the errors on the training data
- Aim: we want a classifier with minimal DL
- MDL principle is a *model selection criterion*

Model selection criteria

- Model selection criteria attempt to find a good compromise between:
  A. The complexity of a model
  B. Its prediction accuracy on the training data
- Reasoning: a good model is a simple model that achieves high accuracy on the given data
- Also known as *Occam's Razor*: the best theory is the smallest one that describes all the facts

Elegance vs. errors

- Theory 1: very simple, elegant theory that explains the data almost perfectly
- Theory 2: significantly more complex theory that reproduces the data without mistakes
- Theory 1 is probably preferable
- Classical example: Kepler's three laws on planetary motion
  - Less accurate than Copernicus's latest refinement of the Ptolemaic theory of epicycles

MDL and compression

- The MDL principle is closely related to data compression:
  - It postulates that the best theory is the one that compresses the data the most
  - I.e. to compress a dataset we generate a model and then store the model and its mistakes
- We need to compute (a) the size of the model and (b) the space needed for encoding the errors
- (b) is easy: can use the informational loss function
- For (a) we need a method to encode the model

DL and Bayes's theorem

- $L[T]$="length" of the theory
- $L[E|T]$=training set encoded wrt. the theory
- Description length= $L[T] + L[E|T]$
- Bayes's theorem gives us the a posteriori probability of a theory given the data:

$$\Pr[T \mid E] = \frac{\Pr[E \mid T]\Pr[T]}{\Pr[E]}$$

- Equivalent to:

$$-\log \Pr[T \mid E] = -\log \Pr[E \mid T] - \log \Pr[T] + \overbrace{\log \Pr[E]}^{constant}$$

MDL and MAP

- MAP stands for *maximum a posteriori probability*
- Finding the MAP theory corresponds to finding the MDL theory
- Difficult bit in applying the MAP principle: determining the prior probability Pr[*T*] of the theory
- Corresponds to difficult part in applying the MDL principle: coding scheme for the theory
- I.e. if we know a priori that a particular theory is more likely we need less bits to encode it

Discussion of the MDL principle

- Advantage: makes full use of the training data when selecting a model
- Disadvantage 1: appropriate coding scheme/prior probabilities for theories are crucial
- Disadvantage 2: no guarantee that the MDL theory is the one which minimizes the expected error
- Note: Occam's Razor is an axiom!
- Epicurus's *principle of multiple explanations*: keep all theories that are consistent with the data

Bayesian model averaging

- Reflects Epicurus's principle: all theories are used for prediction weighted according to P[*T*|*E*]
- Let *I* be a new instance whose class we want to predict
- Let *C* be the random variable denoting the class
- Then BMA gives us the probability of *C* given *I*, the training data *E*, and the possible theories T_j:

$$\Pr[C \mid I, E] = \sum_j \Pr[C \mid I, T_j] \Pr[T_j \mid E]$$

MDL and clustering

- DL of theory: DL needed for encoding the clusters (e.g. cluster centers)
- DL of data given theory: need to encode cluster membership and position relative to cluster (e.g. distance to cluster center)
- Works if coding scheme needs less code space for small numbers than for large ones
- With nominal attributes, we need to communicate probability distributions for each cluster