Experimental Methodology for Benchmarking

Thomas G. Dietterich
Department of Computer Science
Oregon State University
Corvallis, Oregon 97331
tgd@cs.orst.edu

Some Reasons for Experimental Testing of Learning Algorithms

- **Algorithm Testing and Debugging.** Demonstrate that the program is working properly.
- **Estimating Performance on New Data.** How much can users trust the results?
- **Estimating Performance on Similar Tasks.** Many applications involve a series of related tasks.
- **Comparison With Other Algorithms:**
  - **Which algorithm is best on this task?**
  - **Which algorithm is best on an ensemble of tasks?**
  - **Are the two algorithms learning the same thing?**
- **Understanding Algorithm Behavior.**
  - **Why does algorithm A out-perform B on this problem?**
  - **On what kinds of problems will A out-perform B?**

The Standard Experiment for Evaluating an Hypothesis

- **Goal: Estimate the predictive accuracy of an hypothesis.**
- **Method: Evaluate the hypothesis on an independent test set of data.**
- **Statistic: Proportion of examples correctly classified by this hypothesis.**
- **Sources of uncertainty: Random sampling to create test set.**
- **Assessment of uncertainty: Confidence intervals**

  Exact binomial

  Normal approximation to the binomial

  Bootstrap

The Standard Experiment for Evaluating an Algorithm in a Single Domain (Eval One)

- **Goal: Estimate predictive accuracy of algorithm A when trained on data sets of size m from this domain**
- **One Method: k-fold cross-validation using training sets of size m**
- **Statistic: k independent estimates of the probability of misclassification**
- **Sources of Uncertainty:**

  Size of the k test sets

  Variance of the learning algorithm
- **Assessment of uncertainty: Confidence interval based on the t distribution.**

The Standard Experiment for Comparing Two Algorithms in a Single Domain (Eval Pair)

- **Goal: Estimate the difference in predictive accuracy of algorithms A and B when trained on data sets of size m from this domain.**
- **One Method: Perform identical k-fold cross-validations on the two algorithms**
- **Statistic: k independent estimates of the difference in probability of misclassification**
- **Sources of Uncertainty:**
  Size of the k test sets
  Variance of each learning algorithm
- **Assessment of uncertainty: Confidence interval (paired-differences t test)**

Note: At the workshop, Brian Ripley suggested that McNemar's test would be more appropriate. In general, the choice of the most appropriate statistical test appears to be an open problem. Ripley also suggested that we consult the Operations Research literature to see how they conducted statistical tests on computer simulations.

Some Other Designs That Are Wrong

- **t tests based on repeated random train/test splits**

  The test sets are not disjoint so the estimates are not independent. By performing enough random trials, any degree of "statistical signficance" can be obtained.

- **Test for the difference of two proportions based on a single train/test split**

  This accounts for the uncertainty due to the size of the test set, but it does not assess the uncertainty due to variance of the learning algorithms.

Error Cross-Tabulation

Record how each algorithm does on each test example and tabulate.

		Algorithm B: Correct	Algorithm B: Incorrect
Algorithm A	Correct	4239	512
	Incorrect	873	1618

The sum of the off-diagonal counts measures the disagreements between the two algorithms.

It is important to measure the disagreement between an algorithm and itself on multiple runs (e.g., with different random weights or different training sets).

A related method is to construct a $k \times k$ confusion matrix where the rows indicate how algorithm A classified each example and the columns indicate how algorithm B classified each example.

Understanding Algorithm Behavior via Algorithm Mutation

Suppose we observe a difference in performance between algorithms A and B.

Suppose there are a set of potential changes to A, each of which is hypothesized to account for some of the differences between A and B. We can apply each of these potential changes separately or in combination and compare the resulting algorithm to both A and B.

Example: Comparing backprop and nearest neighbor.
A_1: backprop trained on the k nearest neighbors of the query point (local backprop)
A_2: nearest neighbor using only the features with significantly non-zero backprop weights (relevant features)
A_3: both of these changes.

Understanding Algorithm Behavior via Data Mutation

We can perform a similar analysis by mutating the synthetic data set.

Suppose the data set has certain properties that are believed to make algorithm A perform better than B. We can generate new data sets that remove these properties individually or in combination.

Test both algorithms on the new data sets. Each change should leave the performance of B unchanged while hurting the performance of A.

Care must be taken to avoid making the learning task harder (so that all signal is removed from the data).

Answering Our Questions With These Experiment Designs

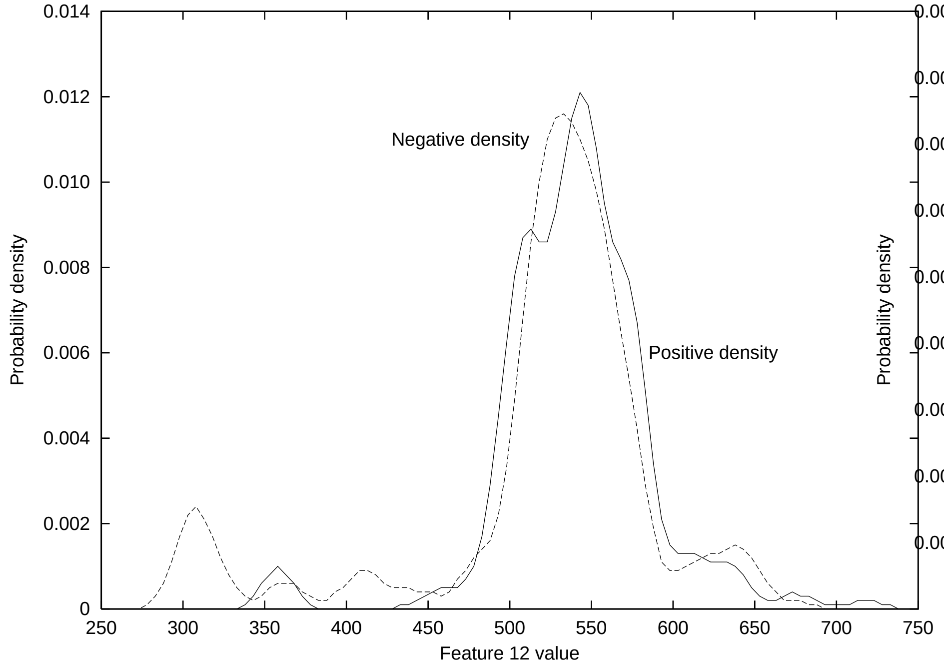
- **Algorithm Testing and Debugging.** Eval One.
- **Estimating Performance on New Data.** Eval One.
- **Estimating Performance on Similar Tasks.** Eval One on several similar data sets.
- **Comparison With Other Algorithms:**
  - **Which algorithm is best on this task?** Eval Pair.
  - **Which algorithm is best on an ensemble of tasks?** Eval Pair on a sample from the ensemble.
  - **Are the two algorithms learning the same thing?** Eval Pair with error cross-tabulation.
- **Understanding Algorithm Behavior.**
  - **Why does algorithm A out-perform B on this problem?** Algorithm mutation.
  - **On what kinds of problems will A out-perform B?** Data mutation.

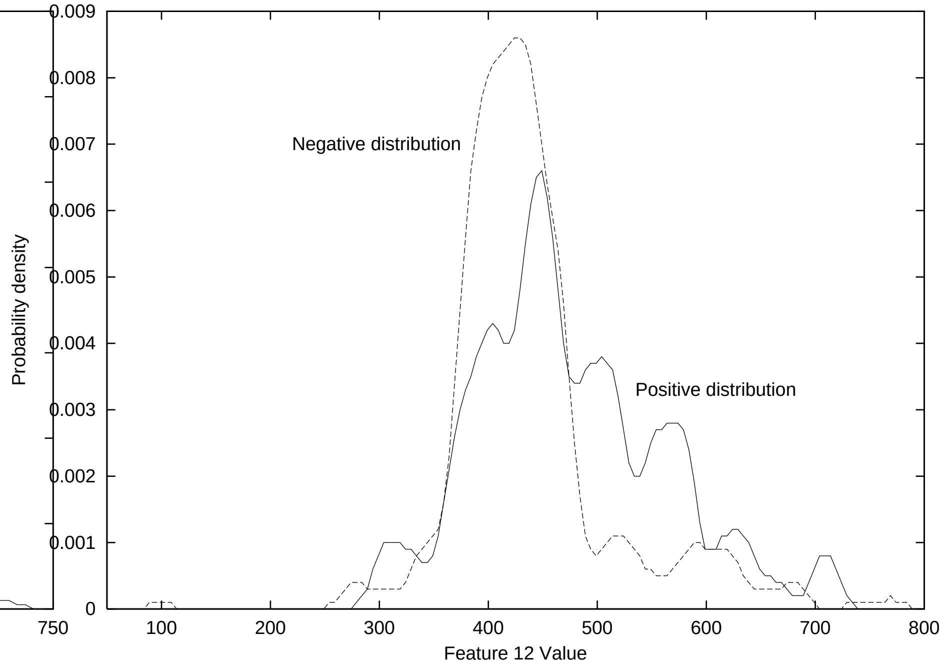
Sources of Data

- **Real World Data Sets.** Useful for demonstrating that our algorithms are solving real problems. Useful for identifying algorithm shortcomings.
- **Synthetic Data Sets.** Useful whenever it is essential to know the "right answer", especially for algorithm debugging and understanding algorithm behavior.
- **Realistic Data Sets.** Data sets constructed by developing generative statistical models of a real data set and then running those models to generate new data.

Example of Realistic Data: Arris Musk Data Set

- **Characterize number of relevant features**. By using a simple learning algorithm (smallest axis-parallel rectangle covering the positive examples)
- **Characterize marginal distribution of data along each feature.** Both positive and negative feature values are approximately normally distributed, but the negative distribution is wider.
- **Generate fake data with similar statistics**





Synthetic versus Real World

- **Algorithm Testing and Debugging.** Synthetic.
- **Estimating Performance on New Data.** Real.
- **Estimating Performance on Similar Tasks.** Realistic.
- **Comparison With Other Algorithms:**
  - **Which algorithm is best on this task?** Real.
  - **Which algorithm is best on an ensemble of tasks?** Real and realistic data.
  - **Are the two algorithms learning the same thing?** Any.
- **Understanding Algorithm Behavior.**
  - **Why does algorithm A out-perform B on this problem?** Any.
  - **On what kinds of problems will A out-perform B?** Synthetic.

References

Aha, D. W. (1992). Generalizing from case studies: A case study. In *Proceedings of the Ninth International Conference on Machine Learning* (pp. 1–10). San Francisco: Morgan Kaufmann.

Dietterich, T. G., Lathrop, R. H., Lozano-Perez, T. (to appear) Solving the multiple-instance problem with axis-parallel rectangles. Accepted for publication in *Artificial Intelligence*.

Data Generators

- **DPG/2 (Benedict)**. Generates synthetic data with real-valued features. Each class is generated by a mixture of gaussians. Only two classes. (Available from Irvine repository.)

Data Generators (2)

- **Chains (Dietterich, under development).** Generate synthetic data with real-valued features. For k classes, generates k chains of points by a random walk in n-dimensional space.

  Generate data by drawing points at random, labeling them according to the nearest-neighbor rule (using the points on the chains), and then embedding them in a higher-dimensional space (via an $n \times m$ matrix).