StatPatternRecognition: A C++ Package for Multivariate Classification of HEP Data

Ilya Narsky, Caltech
Introduction

• Motivation
  – advanced classification tools in a convenient C++ package for HEP researchers

• Description of math formalism:
  – I. Narsky, physics/0507157, Optimization of Signal Significance by Bagging Decision Trees

• Practical instructions:
  – README included in the package
Implemented Methods

• Classifiers:
  – Binary split
  – Linear and quadratic discriminant analysis
  – Decision trees
  – Bump hunting algorithm (PRIM, Friedman & Fisher)
  – AdaBoost
  – Bagging and random forest algorithms
  – AdaBoost and Bagger are capable of boosting/bagging any classifier implemented in the package

• Methods of general use:
  – Bootstrap
  – Calculation of data moments (mean, covariance, kurtosis)
  – Test of zero correlation for variables with a joint elliptical distribution
Linear and Quadratic Discriminant Analysis

- Each class density is a multivariate Gaussian
  \[ f_k(x) = \frac{1}{(2\pi)^{d/2}|\Sigma_k|^{1/2}} \exp \left[ -\frac{1}{2} (x - \mu_k)^T \Sigma_k^{-1} (x - \mu_k) \right] \]

- Look at the log-ratio of two class probabilities:
  \[ \log \frac{f_1(x)}{f_2(x)} = C + x^T (\Sigma_1^{-1} \mu_1 - \Sigma_2^{-1} \mu_2) - \frac{1}{2} x^T (\Sigma_1^{-1} - \Sigma_2^{-1}) x \]

- Assume \( \Sigma_1 = \Sigma_2 \) \( \Rightarrow \) linear Fisher

- Don’t assume \( \Sigma_1 = \Sigma_2 \) \( \Rightarrow \) quadratic Fisher

Separation of a bivariate Gaussian from uniform background by quadratic discriminant analysis.
Decision trees in StatPatternRecognition

\[ S = S_1 + S_2 \]
\[ B = B_1 + B_2 \]

StatPatternRecognition allows the user to supply an arbitrary criterion for tree optimization by providing an implementation to the abstract C++ interface.

At the moment, following criteria are implemented:

- Conventional: Gini index, cross-entropy, and correctly classified fraction of events.
- Physics: signal purity, \( S/\sqrt{S+B} \), 90% Bayesian UL, and \( 2*(\sqrt{S+B} - \sqrt{B}) \).

Conventional decision tree, e.g., CART:
- Each split minimizes Gini index:

\[ Gini = \frac{S_1 B_1}{S_1 + B_1} + \frac{S_2 B_2}{S_2 + B_2} \]
- The tree spends 50% of time finding clean signal nodes and 50% of time finding clean background nodes.

Decision tree in StatPatternRecognition:
- Each split maximizes signal significance:

\[ Signif = \max \left( \frac{S_1}{\sqrt{S_1 + B_1}}, \frac{S_2}{\sqrt{S_2 + B_2}} \right) \]
Decision Trees: An Idealistic Example

Separation of two bivariate Gaussians from uniform background by decision trees optimizing various figures of merit:

- signal significance (top right)
- signal purity (bottom left)
- Gini index (bottom right)

True signal is shown in the top left.
Algorithm finds a box in the multidimensional space in two stages:

• shrinkage: reduce the box size; peel parameter = max fraction of points that can be peeled off in one iteration

• expansion

Above: The bump hunter finds two tiny signal peaks on top of uniform background (right). Full signal+background distribution (left).
AdaBoost

- Method for combining classifiers
- Applies classifiers sequentially:
  - enhances weights of misclassified events
  - computes relative weight of each classifier
- Iterate as long as $\varepsilon < 0.5$
- Final decision: weighted vote of all classifiers

iteration 0: $w_i^{(0)} = 1/N; \quad i = 1, \ldots, N$

iteration $K$: $f^{(K)}(x): \varepsilon_K = \sum_{\text{misclassified}} w_i^{(K-1)} < 0.5$

correctly classified events: $w_i^{(K)} = \frac{W_i^{(K-1)}}{2(1 - \varepsilon_K)}$

misclassified events: $w_i^{(K)} = \frac{W_i^{(K-1)}}{2\varepsilon_K}$

weight of classifier $K$: $\beta_K = \log \left( \frac{1 - \varepsilon_K}{\varepsilon_K} \right)$

$f(x) = \sum_{K=1}^{C} \beta_K f^{(K)}(x)$
Bagging (Bootstrap AGGregatING)

- **Algorithm:**
  - draw a bootstrap replica of the training sample
  - train a new classifier on this replica
  - classify new data by the majority vote of the built classifiers

- **Parallel algorithm:** uses classifiers built on independently-drawn bootstrap replicas

- **Random forest:** A more generic version of bagging
  - sample not only from training events but also from input dimensions
  - for now SPR bootstraps all input dimensions (you can’t choose how many you include)
Adaboost and Bagging: An Idealistic Example

Separation of two bivariate Gaussians from uniform background by boosted binary splits. Background is uniform on $-10<x<10; -10<y<10$.

Separation of two bivariate Gaussians from uniform background by bagged trees. Background is uniform on $-10<x<10; -10<y<10$. 
Why boosted and bagged decision trees are good?

• The best off-the-shelf method for classification in many dimensions.

• Unlike kernel-based methods, methods based on local averaging and nearest-neighbor methods, trees do not suffer from the “curse of dimensionality”.

• CPU time generally scales linearly with dimensionality.

• Training mechanism is robust. Can deal with
  – strongly correlated inputs
  – mixed input types (continuous and discrete)
Separation of $B \to \gamma l \nu$ from combined background at BaBar by bagged decision trees optimizing the signal significance with 11 input variables.

Time needed to train 100 bagged trees on 500k events = several hours.

Gives a 25% improvement in the signal significance compared to the method originally used by the analysts.

B0/B0bar tagging at BaBar by boosted decision trees with 135 input variables.

Time needed to train 50 boosted trees on 500k events = 1+ day.

At the moment, gives a somewhat worse B0/B0bar separation than the clever algorithm that builds 9 neural nets on low-dimensional subtaggers and then combines them. But first results showed promise.
All results shown for test data.

Separation of $pp\bar{p} \rightarrow tqb$ from $pp\bar{p} \rightarrow Wbb$ by bagged decision trees in 11 dimensions.

Time needed to train 200 decision trees on 5000 events = several minutes.

Separation of $pp\bar{p} \rightarrow tqb$ from $pp\bar{p} \rightarrow Wbb$ by boosted binary splits in 11 dimensions.

Time needed to train 1100 binary splits on 5000 events = seconds.
Installation and use outside BaBar

• Send email to narsky@hep.caltech.edu
• Get a copy of the package
• Edit SprExperiment.hh to remove reference to BaBar.hh (one line)
• Provide a Makefile that resolves references to CLHEP headers and CERN libraries
• Either provide your own implementation of SprTupleWriter (depends on BaBar-specific histogramming utils) or replace it with SprAsciiWriter provided in the package
• You are ready to go!
• Feedback is very much appreciated