StatPatternRecognition: A C++ Package for Multivariate Classification

Ilya Narsky, Caltech

presented by Andy Buckley, IPPP Durham

Why develop a new package when there are so many out there?

- Must be free (and distributed under GPL). Commercial software never took off in the HEP community.
- Must be C++. It is the base language for every HEP collaboration.
- Must implement boosting, bagging and random forest.
- Should accommodate certain specifics of HEP analyses, for example, optimizing HEP-specific figures of merit (signal significance etc).

What users might be looking for:

- Interactive analysis
  - quickly run on small datasets
  - Graphics. Convenient tools for displaying input and output.
  - GUI
- Production
  - ability to process large datasets in many dimensions
  - reasonable CPU and memory consumption, scalability
  - use in production code developed by a HEP collaboration
- SPR is much more of the production package than interactive-analysis package. Although it has tools for IA as well.

A bit of history

- Introduced in early 2005 at BaBar and committed to BaBar CVS.
- Between summer 2005 and summer 2006, SPR was distributed outside Babar as a standalone package with a mandatory dependency on CLHEP and an optional dependency on Root (distributed to at least 50 people)
- In summer 2006 SPR was stripped of CLHEP dependency and equipped with autotools
- In September 2006 SPR was posted at Sourceforge: https://sourceforge.net/projects/statpatrec
- Since then, Sourceforge is the main source of SPR code.
- Currently used by several HEP collaborations (BaBar, D0, MINOS, SNO, searches for supernovae), being promoted within CMS. There are users outside HEP as well. Getting feedback from people on use cases is much harder than you might think.
- Downloaded off Sourceforge ~310 times (as of mid April 2007).
- ~10 analyses and ~20 people using the package in BaBar alone, mostly in rare leptonic and particle ID groups.

Authors

- Creator, main developer – Ilya Narsky
- Random number generation, matrix operations and various math routines – adapted from publicly available packages by Xuan Luo
- Autotools – Andy Buckley and Xuan Luo
- ROOT reader and writer – Jan Strube and Josh Boehm

SPR: Implemented Classifiers

- Decision split, or stump
- Decision trees (2 flavors: regular tree and top-down tree)
- Bump hunter (PRIM, Friedman & Fisher)
- LDA (aka Fisher) and QDA
- Logistic regression
- Boosting: discrete AdaBoost, real AdaBoost, and epsilon-Boost. Can boost any sequence of classifiers.
- Arc-x4 (a variant of boosting from Breiman)
- Bagging. Can bag any sequence of classifiers.
- Random forest
- Backprop neural net with a logistic activation function (original implementation)
- Multi-class learner (Allwein, Schapire and Singer)
- Interfaces to SNNS neural nets (without training): Backprop neural net, and Radial Basis Functions

Rectangular cuts for physicists

- Decision tree and bump hunter implemented in SPR can optimize any figure-of-merit supplied through an abstract interface
- So far, 10 FOM's have been implemented:
  - conventional "statistical" FOM's – Gini index, cross-entropy etc
  - FOM's for physics analysis – signal significance, potential for discovery, expected upper limit etc.
- Analysts mostly use this functionality with the bump hunter to optimize the desired FOM.

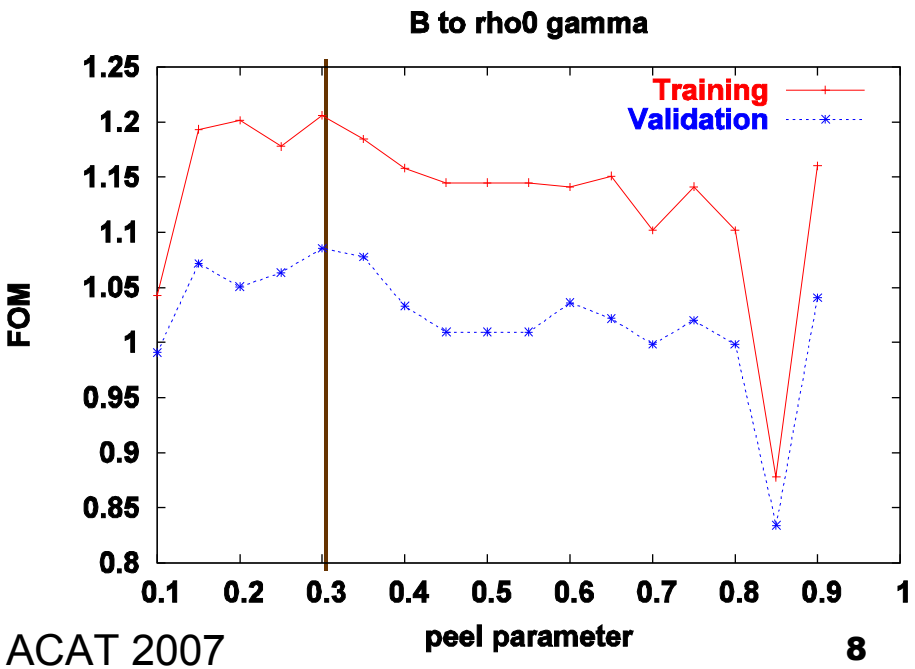
Bump Hunting (PRIM)

Friedman and Fisher, 1999

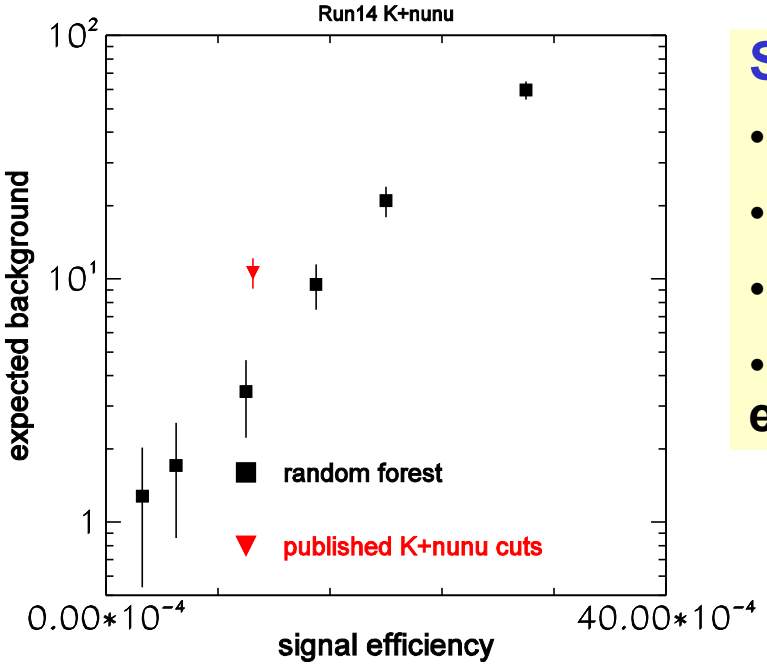
- Works in two stages:
- Shrinkage. Gradually reduce the size of the box. At each step tries all possible reductions in all dimensions and chooses the most optimal one. Rate of shrinkage is controlled by the "peel" parameter = fraction of events that can be removed from the box in one step.
- Expansion. Now tries all possible expansions in all dimensions and chooses the most optimal one.

Search for $B^0 \to \rho^0 \gamma$ at BaBar:

- Optimize **S/sqrt(S+B)** in 8D space
- ~45k signal and ~90k background training events
- Improvement in **S/sqrt(S+B)** by a factor of ~1.5 over manually optimized cuts



Random Forest in HEP

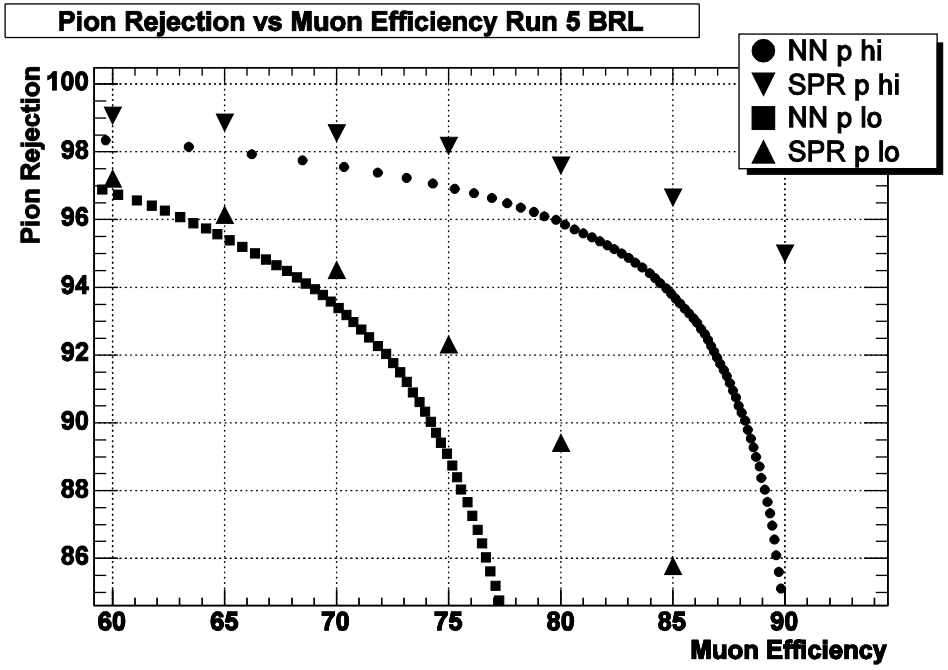


Search for $B^+ \to K^+ \nu\nu$ at BaBar

- 60D
- several hundred k's in the training sample
- 100 decision trees
- 20 input variables randomly selected for each split (F=20)

Muon PID at BaBar

- 17D
- EMC and IFR output combined
- separate classifiers for barrel/endcap and highP/lowP
- an impressive improvement over Neural Net
- 100 decision trees with F=10



Two examples presented at CHEP 2006.

Other tools

- Cross-validation
- Bootstrap
- Tools for variable selection
- Computation of data moments (mean, variance, covariance, kurtosis etc)
- Arbitrary grouping of input classes in two categories (signal and background)
- Multivariate GoF method proposed by Friedman at Phystat 2003
- Tools for combining classifiers (training a classifier in the space of outputs of several sub-classifiers)

Download and install

- Get the package from https://sourceforge.net/projects/statpatrec
- README and INSTALL should be (more than) enough for you to get started.
- SPR can be built in two versions:
  - standalone with Ascii I/O; no external dependencies
  - Root I/O
- Desired build options are chosen by the ./configure script. See INSTALL. Run `./configure --help` for a full list of options.
- The Ascii version of the package builds smoothly. The Root version build can bail out with errors related to Root classes. The fix is to change compilation flags. Again, see INSTALL for detail.
- Successful builds on 32-bit SL3, SL4, RH4, 64-bit Fedora and 64-bit Debian. Enthusiasts adapted SPR to WinXP and MacOS.

How it works

- Training cycle:
  - grab all input data (either from Root or Ascii) and convert into internal SPR format (SprData)
  - every number is coded as double
  - all input data are kept in memory
  - Implementation of decision trees: minimize memory consumption at the expense of CPU. SPR decision trees are relatively slow for small datasets (when you really don't care because it is fast anyway) but scale well for large datasets.
  - save trained classifier configuration into an Ascii file
- Test/production cycle:
  - read saved classifier configuration from this Ascii file
  - apply this classifier to a new event

Touch and feel

- **`cat booster.config`**

```
TopdownTree 5 0 8000
StdBackprop 30:15:7:1 100 0.1 0 0.1
```

- **`SprBoosterApp -M 1 -n 300 -g 2 -t validation.pat -d 1 -f booster.spr train.pat booster.config`**

(Train 300 cycles of Discrete AdaBoost and display exponential loss on each training cycle for validation data. When finished, save classifier configuration into booster.spr)

- **`SprBaggerApp -n 100 -g 1 -t validation.pat -d 1 -f bagger.spr train.pat bagger.config`**

(Train 100 cycles of bagger and display quadratic loss for validation data. Save configuration into bagger.spr)

- **`SprOutputWriterApp -C 'boost,bag' 'booster.spr,bagger.spr' test.pat test.root`**

(Read classifier configurations from booster.spr and bagger.spr and apply them to test data. Save input variables and classifier output into test.root. Classifier responses will be saved with names “boost” and “bag”.)

Using trained classifiers in C++ code

```
#include "StatPatternRecogniion/SprClassifierReader.hh"

……

SprAbsTrainedClassifier* ada =
SprClassifierReader::readTrained("saved_adaboost.spr");

assert( ada != 0 );

vector<double> input = ...;

double ada_output = ada->response(input);

……

delete ada;// delete after you are done
```

User tools

- SprInteractiveAnalysisApp
  - Interactive selection of classifiers and comparison of their performance on test data
  - starts a dialogue with the user
  - Saves user entries and classifier output for used datasets into disk cache. If you want to change just one classifier, you do not need to re-enter parameters and re-train all others.
  - Should be used only on small datasets in not too many dimensions
- SprOutputWriterApp
  - For reading stored classifier configurations and applying them to test data. Can handle any classifier and can read several classifiers at once.
- SprOutputAnalyzerApp
  - For people who don't like Root. Prints out efficiency curves for data and classifier responses stored in Ascii format.

Benchmarks

- Example: ($B \to \rho/\omega\gamma$ analysis at BaBar)
  - 2 GHz Opteron CPU with 1 GB/core
  - training sample with 120k events in 300 dimensions
  - ~3.5 hours (user cpu time) to build 100 bagged decision trees with 10 events per leaf
- Against R
  - 90k training set in 4D (B0/B0bar tagging)
  - SPR random forest: 164.360u 5.170s 3:00.55 93.8%
  - R randomForest: 708.970u 940.830s 31:13.56 88.0%
- Against m-boost (C, used at MiniBOONE):
  - SPR slower by a factor of ~2 on a sample of 50k events in 50D
  - SPR faster by a factor of ~2 on a sample of 150k events in 60D
- More benchmarks are needed

Outlook

- More info can be found on the main author's web page:
  http://www.hep.caltech.edu/~narsky/spr.html

  http://www-group.slac.stanford.edu/sluo/Lectures/Stat2006_Lectures
- The package is under development. Check for the latest version at Sourceforge
- Volunteers are always welcome:
  - development of new methods
  - testing builds on various platforms and fixing Makefiles
  - benchmarking