StatPatternRecognition: A C++ Package for Multivariate Classification

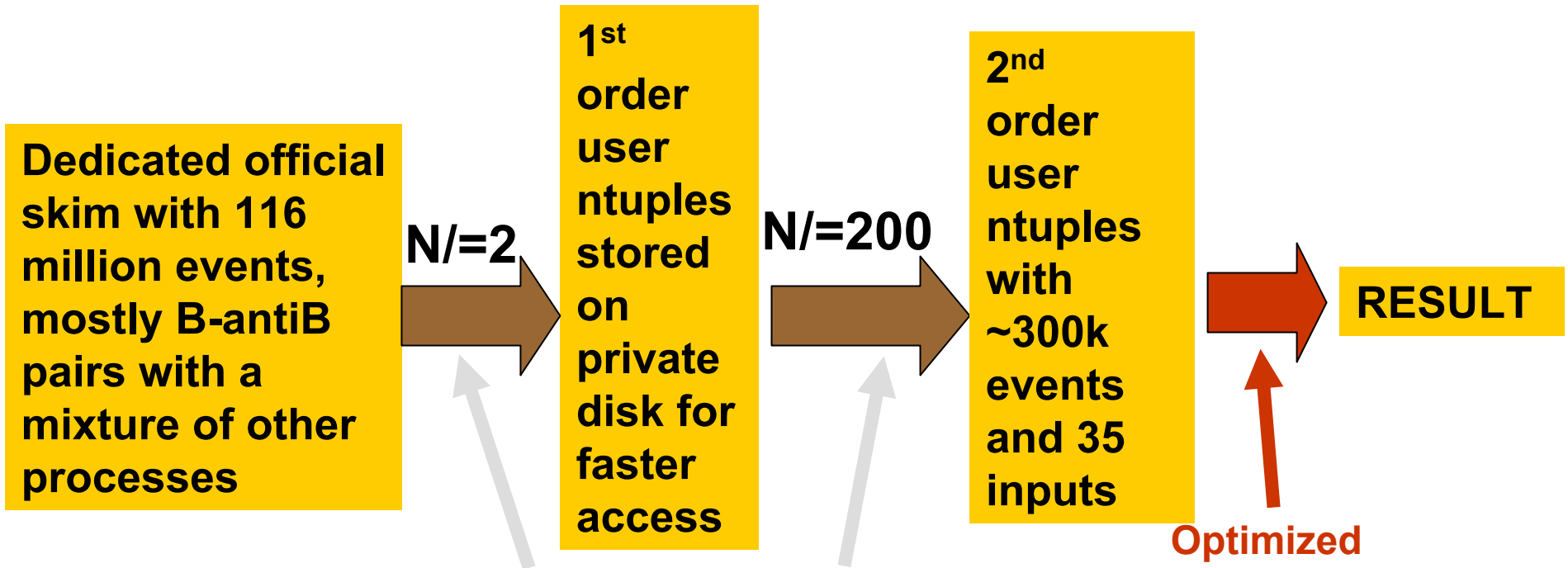
Ilya Narsky, Caltech HEP

Assumptions

- Most people in the audience have an idea of how data analysis in High Energy Physics looks like. The name of the game is to separate weak signal from dominating background processes.
- No time to cover all methods implemented in the package. Focus only on advanced popular methods such as boosted decision trees and random forest.

Example: analysis at BaBar

Searching for $B^+ \to K^+ \nu\nu$ tagged by $B^- \to D^{(*)0} l^- \nu\nu$



Preliminary requirements are more or less well-known: people have been doing similar analyses for decades

Machine Learning setup

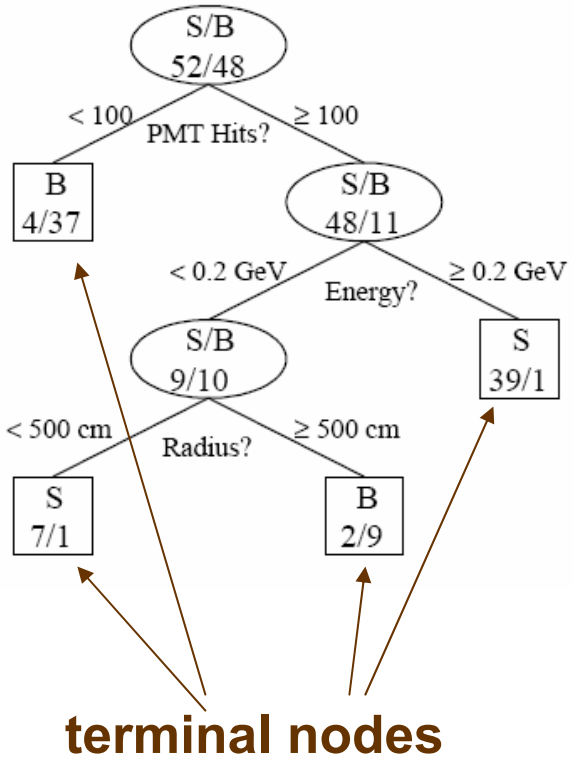
- SPR implements tools for supervised learning
  - learning on training datasets where signal and background are exactly known
  - training sets are typically obtained by Monte Carlo simulation or from real-data control samples of high purity
- Unsupervised learning: Clustering and feature selection
  - another chapter from statistical textbooks
  - No good knowledge of what you should call “signal” and “background”. Exploration.
  - the only tool in SPR that can be used for unsupervised learning is the bump hunter
- In HEP training sets can be arbitrarily large. They inevitably need to be reduced to a manageable size before sophisticated multivariate techniques can be deployed. “Manageable” is defined by your time and the software you use.

Tools for Multivariate Classification in HEP

- Classifiers actively explored by physicists
  - cut (binary split, or binary stump)
  - linear discriminant analysis (Fisher)
  - Idiot's Bayes (max likelihood fit)
  - feedforward backpropagation neural nets
  - radial basis function neural nets
  - Bayesian neural nets
  - genetic algorithms
  - decision trees
  - support vector machines
  - kernel density estimation
  - local averaging and nearest neighbor methods
  - boosting and bagging
  - random forest

Decision Trees

Decision trees emerged in mid 80's: CART (Breiman, Friedman etc), C4.5 (Quinlan) etc



Popular criteria used for splitting

(p = fraction of correctly classified events)

$$Q(p) = p \qquad \text{purity}$$

$$Q(p) = -2p(1-p) \qquad \text{Gini index}$$

$$Q(p) = p\log p + (1-p)\log(1-p) \qquad \text{cross-entropy}$$

Parent node with *W* events and correctly classified *p*W* events is split into two daughters nodes iff

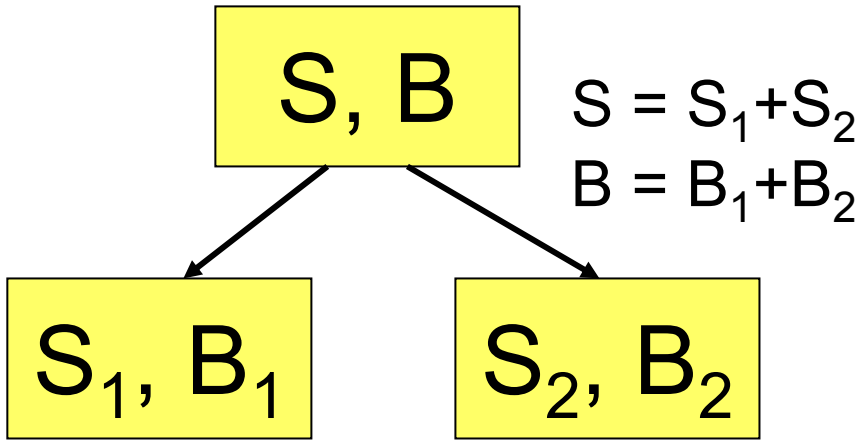
$$WQ(p) < W_1 Q(p_1) + W_2 Q(p_2)$$

Split nodes recursively until a stopping criterion is satisfied.

Stopping criteria:

- **unable to find a split that satisfies the split criterion**
- **maximal number of terminal nodes in the tree**
- **minimal number of events per node**

One decision split under the microscope



Conventional decision tree, e.g., CART:

- Each split minimizes Gini index:

$$\frac{S_1 B_1}{S_1 + B_1} + \frac{S_2 B_2}{S_2 + B_2} < \frac{SB}{S + B}$$

- Ideally **S_1=0; B_2=0** or **S_2=0; B_1=0**

Decision tree in StatPatternRecognition:

- Each split maximizes any user-supplied criterion, e.g., signal significance:

$$Signif = \max\left(\frac{S_1}{\sqrt{S_1 + B_1}}, \frac{S_2}{\sqrt{S_2 + B_2}} \right)$$

Classifying new events:

- **discrete scoring: 1 if an event lands on a signal node, 0 otherwise**
- **continuous scoring: for example, signal purity, $S_1/(S_1+B_1)$**

What is so good about Decision Trees?

- Can easily deal with strongly correlated inputs and inputs of mixed type (real and integer).
- Distances between training points do not matter. Curse of dimensionality defeated.
- Training time scales linearly vs dimensionality
  - Need O(N·log(N)) operations to sort training points in each dimension.
- If a tree has only a few leaves, it is easy to interpret and visualize.
- Of course, a single decision tree typically gives inferior predictive power…

Weak Classifier = a classifier whose classification error is a little better than 50%

Combining weak classifiers: Boosting, Bagging and Random Forest

Boosted Decision Trees and Random Forest are the best off-the-shelf tools for multivariate classification

(Discrete) AdaBoost

Freund and Schapire, 1996

Idea: Combine weak classifiers by applying them sequentially

- Build a new weak classifier on the weighted training set and apply it to the same training set
- Enhance weights of misclassified events and reduce weights of correctly classified events
- Continue as long as weighted misclassified fraction less than 50%

$$\text{iteration } 0: \quad w_n^{(0)} = 1/N; \quad n = 1,\ldots,N$$

$$\text{iteration m}: \quad f_m(x): \ \varepsilon_m = \sum_{\text{misclassified}} w_n^{(m-1)} < 0.5$$

$$\text{correctly classified events}: \quad w_n^{(m)} = \frac{w_n^{(m-1)}}{2(1-\varepsilon_m)}$$

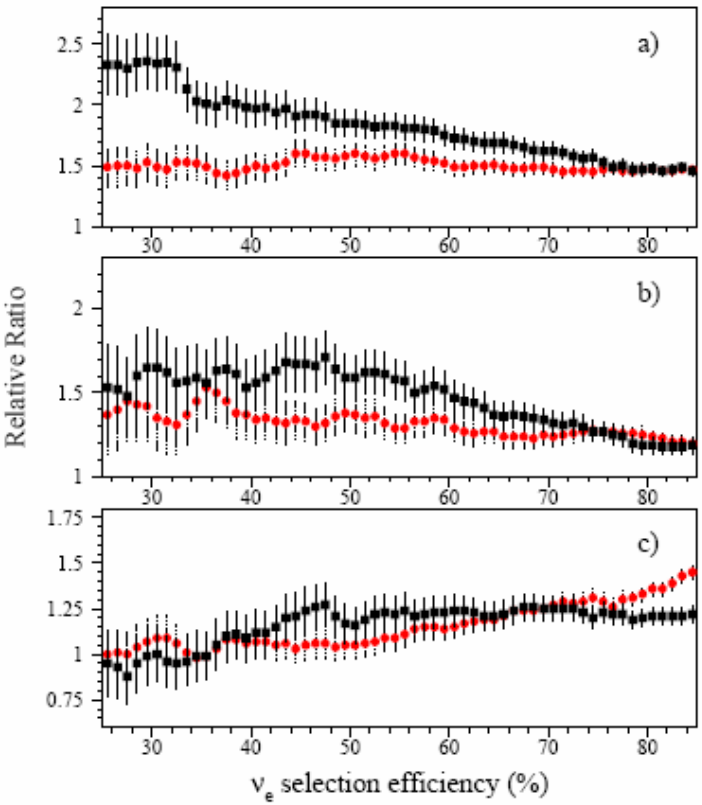
$$\text{misclassified events}: \quad w_n^{(m)} = \frac{w_n^{(m-1)}}{2\varepsilon_m}$$

$$\text{weight of classifier m}: \beta_m = \frac{1}{2}\log\left(\frac{1-\varepsilon_m}{\varepsilon_m}\right)$$

To classify new data: weighted vote of all classifiers

$$f(x) = \sum_{m=1}^{M} \beta_m f_m(x)$$

PID at MiniBoone: Boosted Decision Trees vs Neural Net



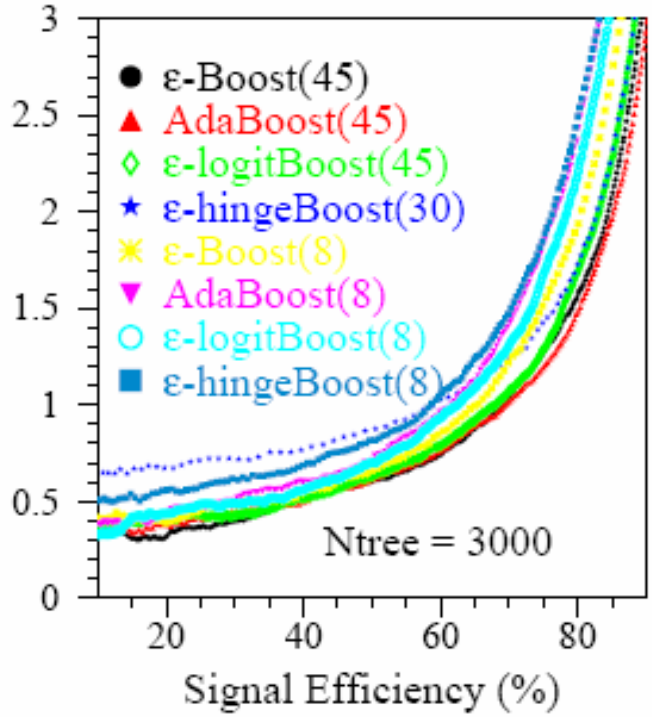
Ratio of background contributions NN/BDT (2 upper plots):
21 input variables(red)
52 input variables (black)

Ratio of BDT with 21(red) and 22(black) variables over BDT with 52 input variables (lower plot).

Byron Roe et al.,

physics/0408124, physics/0508045

Comparison of various boosting algorithms. Background/signal ratio (in relative units) vs signal efficiency.



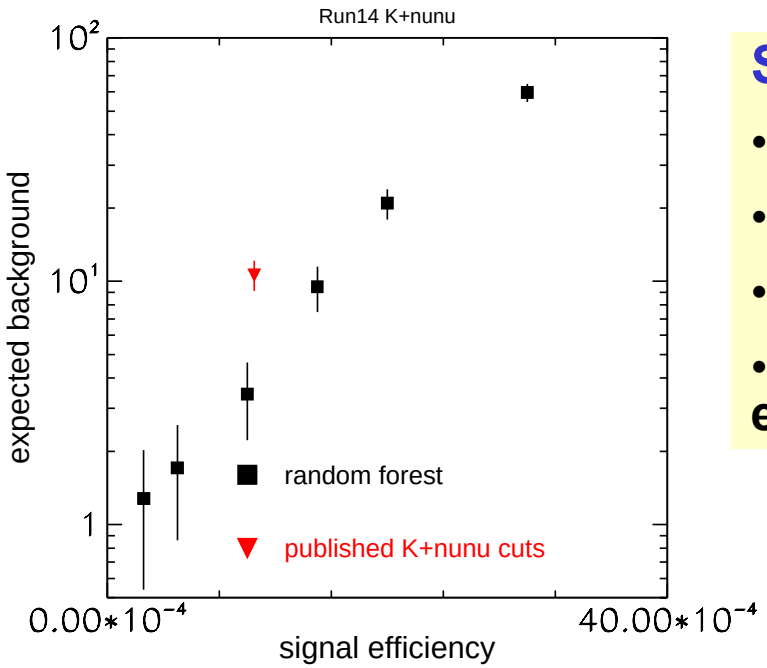
Bagging (Breiman, 1996)

- Acronym for Bootstrap AGGregatING
- One bootstrap replica = N points drawn with replacement out of sample of size N
- Idea: build many classifiers on independently-drawn bootstrap replicas of the training sample
- Classify new data by the majority vote of the built classifiers
- The underlying classifier must be unstable, i.e., produce substantially different output when trained on resampled events. Bagging a stable classifier won't help.
  - Unstable: Decision tree with fine leaves
  - Stable: Binary split (or stump), Nearest Neighbor methods
- Unlike boosting, this is a parallel algorithm. Training weights for each weak classifier do not depend on classifiers built so far.

Random Forest (Breiman, 2001)

- http://www.stat.berkeley.edu/users/breiman/RandomForests/
  "[Random Forest] is unexcelled in accuracy among current algorithms." – Leo Breiman, home page
- Select a random sub-set of variables for each decision split; the size of the selected sub-set is controlled by the user
- Usually applied in conjunction with bagging
- RF makes decision trees less correlated with each other => increases the overall classification power of the majority vote
- If the number of randomly selected variables for every decision split is small, it takes significantly less time to construct each decision tree

Random Forest in HEP

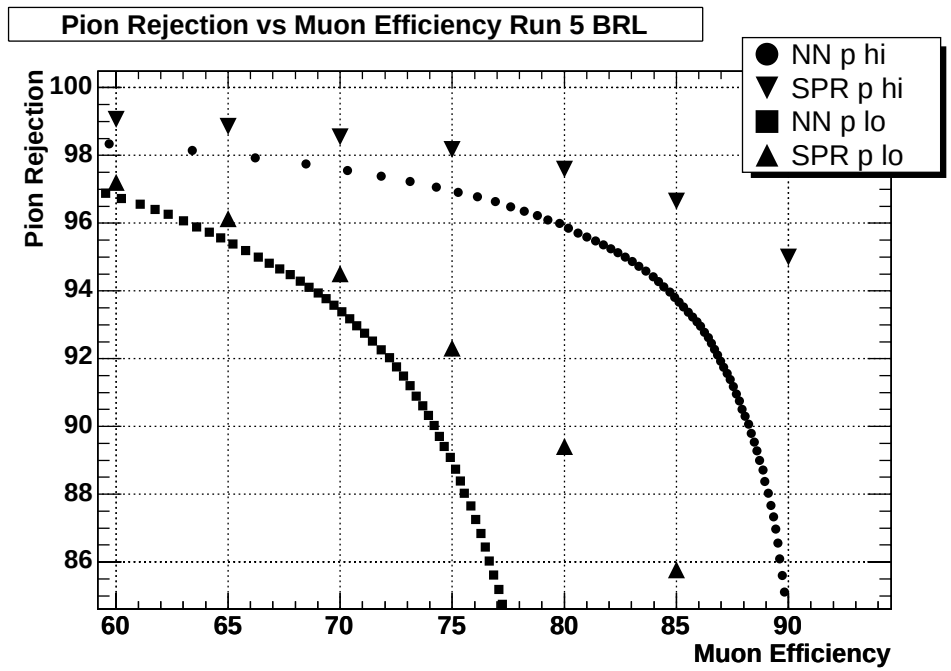


Search for $B^+ \rightarrow K^+ \nu\nu$ at BaBar

- 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.

	Neural Net	RBF	SVM	Decision Tree	BDT and RF	k-NN
Predictive power	good	good	good	poor	good	good
Ability to deal with irrelevant inputs	poor	poor	poor	good	good	poor
Interpretability	poor	poor	poor	good	poor	poor
Curse of dimensionality	good	poor	fair	good	good	poor
Computational scalability with adding new dimensions	poor	poor	poor	good	good	poor
Training stability	poor	poor	poor	fair	good	none
Response time	good	fair	fair	good	fair	poor

good fair poor

Mostly copied from Hastie, Tibshirani & Friedman

BDT and RF… what else?

- Boosted and bagged (plus random forest) decision trees are a popular choice. But not the only one.
- Boosted random forests
  - Byron Roe et al., physics/0508045, PID at MiniBOONE. Performance same as that of optimal boosted decision trees.
- Boosted neural nets
  - Statistics literature: plenty, e.g., H. Drucker, "Boosting Using Neural Networks", in *Combining Artificial Neural Nets*, ed. A. Sharkey, Springer Series in Perspectives in Neural Computing
  - Physics literature: Meiling, Mingmei and Lianshou, hep-ph/0606257, classification of quark and gluon jets from e+e- collisions at 91 GeV
- Bagged neural nets
  - none in physics; examples in stats literature, e.g., H. Drucker…
- Cascade techniques:
  - **logistic regression + decision tree** Zhao and Ram, "Constrained Cascade Generalization of Decision Trees", in *IEEE Transcations on Knowledge and Engineering* (2004)
  - **neural net + boosted decision trees** Liu and Stancu, "Cascade Training Technique for Particle Identification", physics/0611267 (MiniBOONE)

What to do if you want to learn more

- Hastie, Tibshirani and Friedman, *The Elements of Statistical Learning,* Springer Series in Statistics 2001
- Webb, *Statistical Pattern Recognition,* 2nd ed., John Wiley & Sons 2002
- Haykin, *Neural Networks,* 2nd ed., Prentice Hall 1999
- Kuncheva, *Combining Pattern Classifiers,* John Wiley & Sons 2004
- http://www-group.slac.stanford.edu/sluo/Lectures/Stat2006_Lectures.html

Software for Machine Learning

What software physicists use?

- Different groups choose different tools:
  - BaBar – Stuttgart Neural Network Simulator (C) and JetNet (Fortran)
  - D0 – TerraFerma, JetNet, C2.4, Radford Neal's Bayesian NN package
  - WEKA (open source Java); http://www.cs.waikato.ac.nz/ml/weka/ (U of Waikato)
  - R http://www.r-project.org/ Open source command-line interpreter popular among university statisticians. Promoted by a group of physicists from Fermilab for HEP analysis
- Since 2005:
  - SPR (early 2005)
  - TMVA (late 2005). Root-based package developed mostly by ATLAS members at CERN

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 our 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).

Software setup

- 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 (thanks to Andy Buckley and Xuan Luo)
- 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. Mostly HEP users but also users outside HEP. Getting feedback from people on use cases is much harder than you might think.
- Downloaded off Sourceforge ~250 times.
- ~10 analyses and ~20 people using the package in BaBar alone, mostly in rare leptonic and particle ID groups.

SPR: Implemented Classifiers

- Decision split, or stump
- Decision trees (2 flavors)
- 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

Other tools

- Cross-validation
- Bootstrap
- Tools for variable selection
- Decision trees and bump hunter can optimize any of 10 figures of merit implemented in the package
- 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

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 is 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

Plans

- Write a set of scripts to adapt the Sourceforge version to Babar and CMS environments. Other people promised to do this.
- Build a web service using Clarens framework. Clarens has been developed at Caltech and used for the most part at CMS. Use a web client to submit jobs to remote published resources.
- If you are interested in contributing, I have a laundry list of methods that would be good to implement.