New Developments in Pattern Classification of High Energy Physics Data

Ilya Narsky, Caltech

What is a multivariate classifier?

- Any tool that, given coordinates of a point in a multidimensional space, assigns this point to one of possible categories (e.g., signal and background)
- Classification tools explored by physicists:
  - cut (in statistician's language – binary split, or binary stump)
  - linear discriminant analysis (Fisher)
  - feedforward backpropagation neural nets
  - radial basis function neural nets
  - Bayesian neural nets
  - genetic algorithms
  - decision trees
  - support vector machines
  - kernel density estimation and naïve Bayes
  - local averaging and nearest neighbor methods

“New” developments

- “New” as in “new for HEP, not new in statistics research”
- Methods for combining weak classifiers:
  - boosting
  - bagging and random forest
- Popular choices for the weak classifier:
  - binary split
  - decision tree

Literature

- Hastie, Tibshirani & Friedman “Elements of statistical learning”
- Webb “Statistical pattern recognition”
- Haykin “Neural networks, a comprehensive foundation”
- Kuncheva “Combining classifiers”

Outline

- Basics: terminology and setup
- Overview of classification methods attempted in HEP
- Boosting and bagging decision trees
- How you can use these methods in practice: Software
- Examples from HEP analysis
- (My take on the) Future of pattern classification in HEP

Classification and regression

- Notation
  - $\{(x_i, y_i)\}_{i=1}^{N} \quad x_i = \{x_{ij}\}_{j=1}^{d} \quad y_i = \{y_{ik}\}_{k=1}^{K}$ sample of d-dimensional input vectors x and K-dimensional output vectors y
- Problem
  - find function (predictor) f(x) that approximates y – and use f(x) to predict y on future data samples
- Definition
  - y continuous => *regression* problem
  - y discrete => *classification* problem
- Example of classification problem
  - signal y=1 and background y=0 (K=2)
  - signal y=(1,0,0); 1st bgrnd y=(0,1,0); 2nd bgrnd y=(0,0,1) (K=3)
- One can think of classification as drawing a boundary in the d-dimensional input space that separates categories from each other. This is equivalent to formulation above.

Optimization problem

- What does it mean – *f(x)* approximates *y*?
  - Define per-event *loss*
    - quadratic loss $L_{\text{qua}}(y, f(x)) = (y - f(x))^2$
    - exponential loss $L_{\text{exp}}(y, f(x)) = \exp(-yf(x))$
    - support vector machine loss

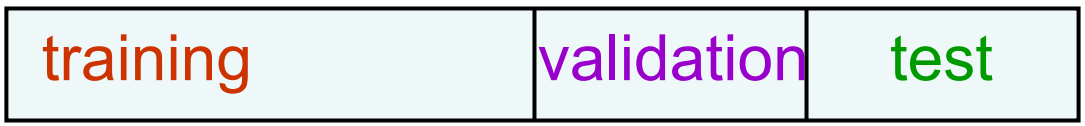
$$L_{\text{SVM}}(y, f(x)) = [1 - yf(x)]_+$$

  - Find f(x) such that minimizes *empirical risk, or classification error*

$$R = \frac{1}{N}\sum_{i=1}^{N} L(y_i, f(x_i))$$

Choice of model

- As you put more and more flexibility into your model f(x), empirical risk gets smaller and smaller… How do you when to stop? And what model f(x) to choose?



Training error optimistically underestimates true model error.

- optimize your model on the training sample
- stop optimization when validation error reaches minimum
- use the test sample to assess model error

What if you have barely enough data for training?

- Data resampling:
  - cross-validation
  - bootstrap
- Quick and “dirty” shortcuts
  - Methods
    - Akaike Information Criterion
    - Bayes (Schwartz) Information Criterion
  - Idea: penalize on the number of free parameters in the model
  - Two questions:
    - a) what approach to use
    - b) what is the number of free parameters in the model?

Cross-validation

- Split data into M subsamples
- Remove one subsample, optimize your model on the kept M-1 subsamples and estimate prediction error for the removed subsample
- Repeat for each subsample

$$m(i):\ \{1,2,\ldots,N\} \mapsto \{1,2,\ldots,M\} \qquad R_{CV} = \frac{1}{N}\sum_{i=1}^{N} L\left(y_i, f_{-m(i)}(x_i)\right)$$

- How to choose M?
  - M=N => CV error estimator is unbiased for the model error but can have large variance
  - M«N => CV error estimator has small variance but can have large bias

Bootstrap

- A seriously underestimated method in HEP analysis.
- Can and should be used to assess estimator bias and variance if neat evaluation of these on a statistically large data sample is not possible.
- Method
  - randomly draw N events with replacement out of the data sample of size N => one bootstrap replica; make, for example, B=100 replicas
  - optimize model on a bootstrap replica and see how well it predicts the behavior of the original sample

$$R_B = \frac{1}{BN}\sum_{b=1}^{B}\sum_{i=1}^{N} L\left(y_i, f_b^*(x_i)\right)$$

  - reduce correlation between bootstrap replica and the original sample => to estimate error at point *i*, use only those replicas that do not contain point *i* ("leave-one-out" method)
  - reduce bias using a ".632 estimator"

$$R_B^{(1)} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{|C_{-i}|}\sum_{b\in C_{-i}} L\left(y_i, f_b^*(x_i)\right)$$

$$P(\text{point } i \in \text{replica } b) = 1-\left(1-\frac{1}{N}\right)^N \approx 1-e^{-1} = 0.632 \qquad R^{(0.632)} = 0.368\cdot R_{training} + 0.632\cdot R_B^{(1)}$$

Bayes (Schwartz) Information Criterion

- Recommended for use with ML fits
- Was derived using Bayes methodology: select model with highest posterior probability
- Assuming that all models have equal prior probabilities and making some generic simplifying assumptions about the model behavior, one obtains:

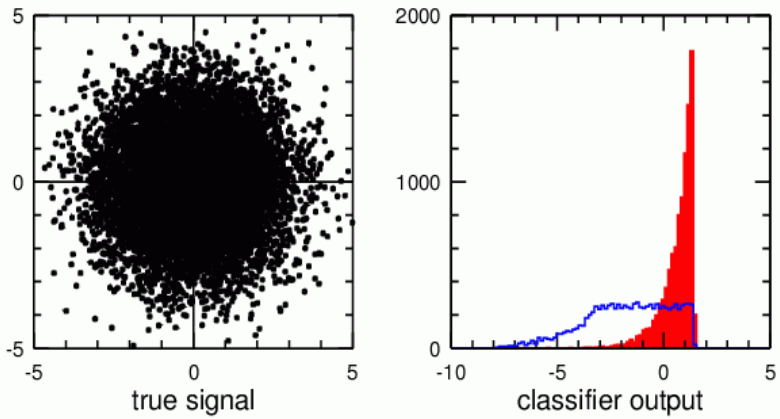
$$BIC = -2\log L(\hat{\theta} \mid x) + p\log N$$

where L is likelihood maximized over the parameter space

p is the number of free parameters in the likelihood

- Choose model with minimal BIC

Linear and Quadratic Discriminant Analysis



Separation of a bivariate Gaussian from uniform background by quadratic discriminant analysis.

- Each class density is a multivariate Gaussian

$$f_k(x) = \frac{1}{(2\pi)^{d/2}|\Sigma_k|^{1/2}} \exp\left[-\frac{1}{2}(x-\mu_k)^T \Sigma_k^{-1}(x-\mu_k)\right]$$

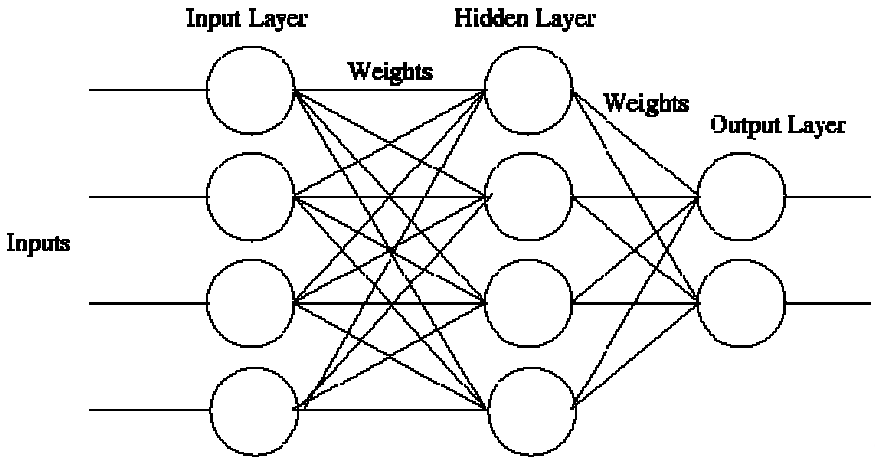
- Look at the log-ratio of two class probabilities:

$$\log\frac{f_1(x)}{f_2(x)} = C + x^T\left(\Sigma_1^{-1}\mu_1 - \Sigma_2^{-1}\mu_2\right) - \frac{1}{2}x^T\left(\Sigma_1^{-1} - \Sigma_2^{-1}\right)x$$

- Assume $\Sigma_1 = \Sigma_2$ => linear Fisher
- Don't assume $\Sigma_1 = \Sigma_2$ => quadratic Fisher

Neural Networks

Milestones:
McCulloch & Pitts (1943) – early concept
Rosenblatt (1958) – learning perceptron
Hopfield (1982)
Rumelhart et al. (1986) – back-propagation algorithm



input layer: $y_i^{(0)} = x_i;\ i = 1,\ldots,d$

hidden and output layers:

$$v_i^{(n+1)} = \sum_{j=1}^{d^{(n)}} w_{ij}^{(n)} y_j^{(n)}; \quad y_i^{(n+1)} = \phi\left(v_i^{(n+1)}\right)$$

1) Forward propagation from layer n to layer (n+1) using linear weights and an activation function phi.

Now compute classification error in the output layer:

$$R = \sum_{k=1}^{K} \left(y_k - o_k\right)^2$$

(sum-squared error over K categories)

2) Propagate classification error back to update weights

$$\Delta w_{ij}^{(n)} = -\eta \frac{\partial R}{\partial w_{ij}^{(n)}}$$

Logistic, or sigmoid, activation function:

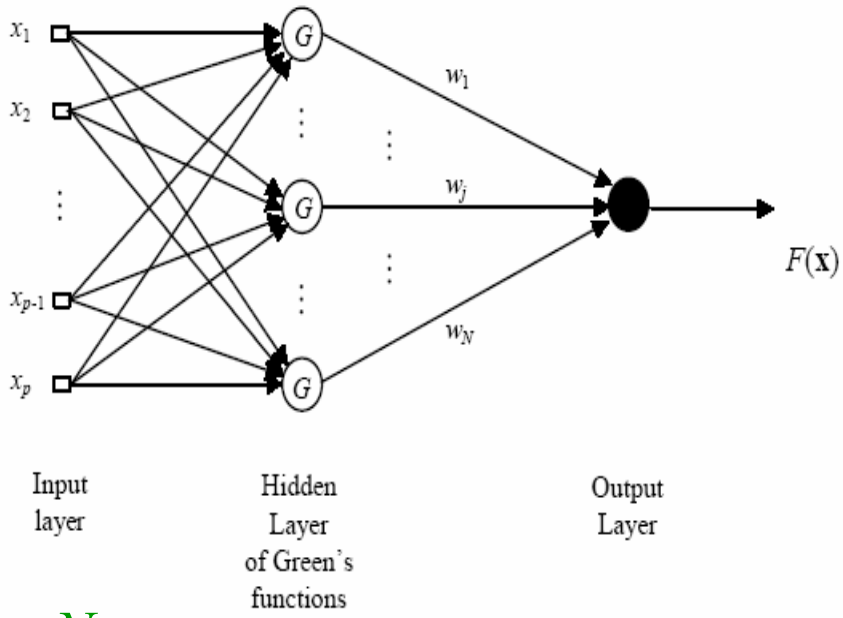
$$\phi(v) = \frac{1}{1+\exp(-av)} \qquad \frac{d\phi}{dv} = a\phi(1-\phi)$$

Changes fastest in midrange => contributes to the stability of the network.

From now on, I will refer to this as "standard NN."

Radial Basis Functions networks

- RBF as a network
  - one hidden layer
  - one hidden unit per expansion center
  - M expansion centers for N training points



$$f(x) = \sum_{i=1}^{M} w_i G(\|x - t_i\|); \quad i = 1,\ldots,M; \quad M < N$$

$$w = \left(G^T G + \lambda T\right)^{-1} G^T y; \quad G_{ij}^{(N\times M)} = G(x_i, t_j); \quad T_{ij}^{(M\times M)} = G(t_i, t_j)$$

Radial Basis: $G(x, x') = G(\|x - x'\|)$

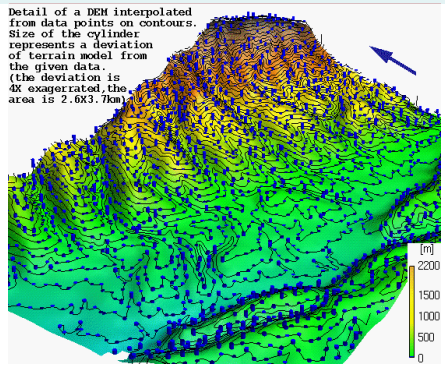
- 1) Smoothing splines

$$G(r) = \begin{cases} (r/\rho)^{2m-d} \log(r/\rho) & d \text{ even} \\ (r/\rho)^{2m-d} & d \text{ odd} \end{cases}$$

$$f(x) = \sum_{i=1}^{N} w_i G\left(\left|x - x_i\right|\right) + P^{(m-1)}(x)$$

  - d=1 m=2 => natural cubic spline
  - d=2 m=2 => thin-plate spline $G(r) = r^2 \log(r)$

Example: Geographic Information Systems (d=2,3,4)
http://skagit.meas.ncsu.edu/~helena/gmslab/viz/sinter.html



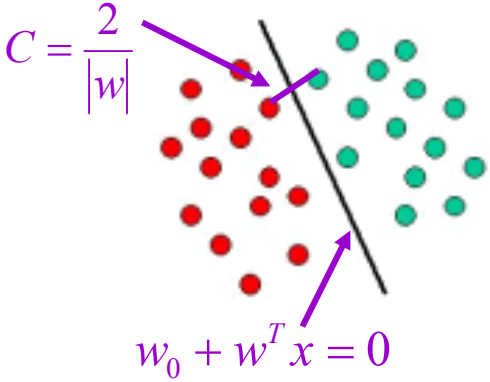
- 2) Gaussian kernel regression

$$G(r) \propto \exp\left(-\frac{r^2}{2\sigma^2}\right)$$

  - Nadaraya-Watson estimator $\lambda = 0 \Rightarrow f(x) = \dfrac{\sum_{i=1}^{N} y_i G\left(\left|x - x_i\right|\right)}{\sum_{i=1}^{N} G\left(\left|x - x_i\right|\right)}$

Support Vector Machines Vapnik et al., 1992

- Idea: draw an optimal separating hyperplane between two *perfectly separated* classes



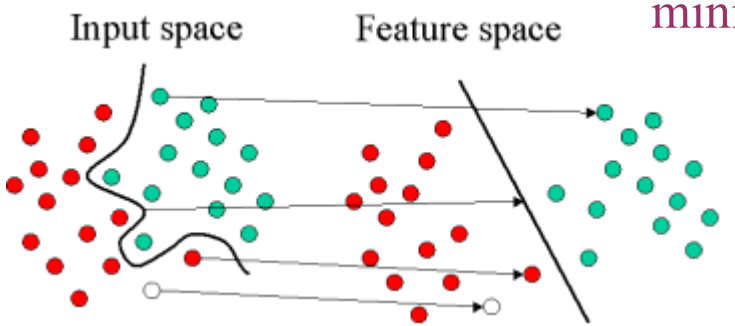
$$y_i = -1 \text{ (bgrnd) or } 1 \text{ (signal)}$$

Optimization problem:

$$\max C \text{ such that } y_i\left(w_0 + w^T x_i\right) \geq C;\ i = 1,\ldots,N$$
$$|w| = 1$$

- Of course, in reality the two classes are not perfectly separated. Solution: map inputs onto a high-dimensional space

$$\{x_i\}_{i=1}^{d} \mapsto \{h_m(x)\}_{m=1}^{M};\quad M >> d$$



$$\text{minimize } R = \sum_{i=1}^{N}\left[1 - y_i f(x_i)\right]_+ + \lambda|w|^2;\quad f(x) = w_0 + w^T h(x)$$

$$f(x) = \alpha_0 + \sum_{i=1}^{N}\alpha_i y_i G(x, x_i) \qquad \alpha_i \neq 0 \Rightarrow \text{support vector}$$

Again, kernel expansion but with a different error term, a different penalty term, and not necessarily radially-symmetric kernel.

Kernel density classification and Naïve Bayes

- Choose the class with the highest posterior density

$$g(\omega_i \mid x) = P(\omega_i)\, p(x \mid \omega_i) \qquad i = 1,\ldots,C$$

- Max likelihood fit $L(x) = \sum_{i=1}^{C} \hat{\alpha}_i\, p(x \mid \omega_i)$
- Idiot's Bayes: input variables are independent

$$p\left(x \mid \omega_i\right) = \prod_{k=1}^{D} p_k\left(x_k \mid \omega_i\right)$$

- Typically gives a mediocre classification error

Local averaging and k-NN methods

- Local averaging (Parzen window)
  - hypersphere or hypercube with volume V and n points of which n_i come from class i

$$p(x \mid \omega_i) = \frac{n_i}{N_i V} \qquad p(\omega_i \mid x) = \frac{n_i}{n}$$

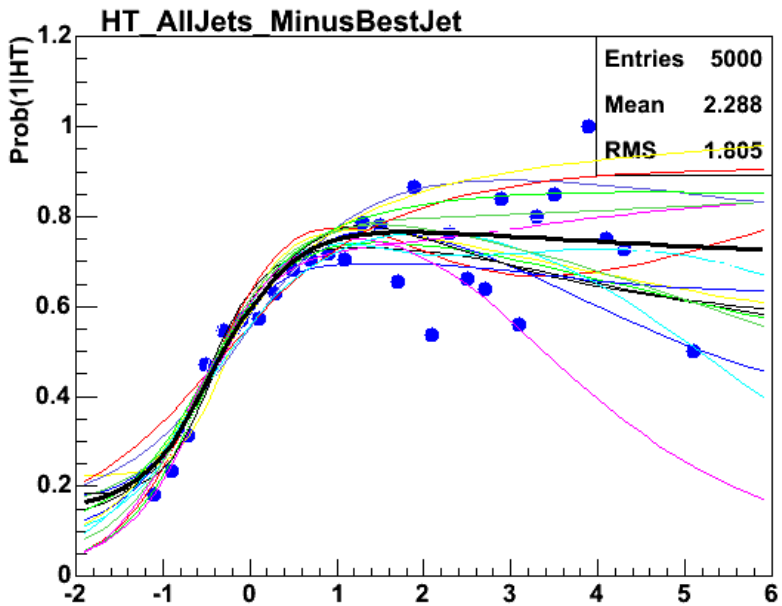
- k-NN method
  - classify a point by the majority vote of k nearest neighbors
- Local methods typically provide a good predictive power in low-dimensional problems
- Disadvantages:
  - computationally expensive (memory-based)
  - k-NN is sensitive to the choice of distance metric
  - curse of dimensionality

Bayesian Neural Networks

- Conventional NN:
  - find one optimal set of NN parameters
- Bayesian NN
  - obtain a posterior pdf of NN parameters by consecutive updating
- D0 uses Radford Neal's package
  - http://www.cs.utoronto.ca/~radford/fbm.software.html
- Advantage: approximates P(Y=1|X)
- Disadvantage: must integrate over an ensemble of Neural Nets with varying parameters => slow

$$\prod_{c=1}^{n} P(y^{(c)} \mid x^{(c)}, w)$$

$$P(y^{(n+1)} \mid x^{(n+1)}, (x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})) = \int [dw]\, P(y^{(n+1)} \mid x^{(n+1)}, w) \times P(w \mid (x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)}))$$

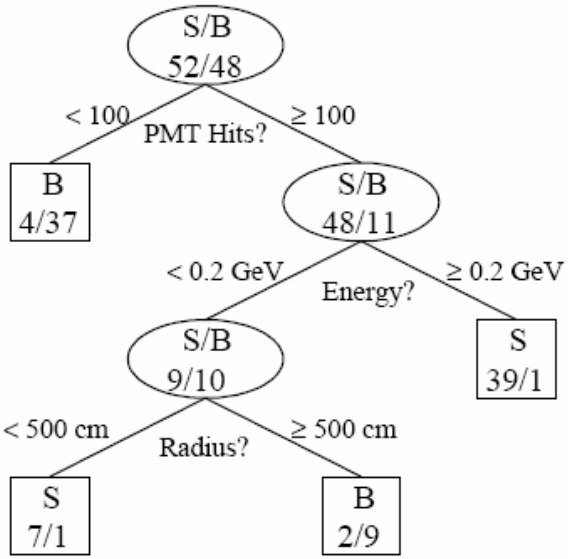


Genetic algorithms

- Construct an ensemble of classifiers that are
  - accurate: each classifier gives a low classification error
  - diverse: maximize disagreement between individual classifiers
- Roots in evolution of biological species

Decision Trees

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



Criteria used for commercial trees

(p = fraction of correctly classified events)

$$Q(p) = p$$

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

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

Split nodes recursively until a stopping criterion is satisfied.

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

$$WQ(p) < W_1Q(p_1) + W_2Q(p_2)$$

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**

Output of a decision tree is discrete: 1 if an event falls into a signal node, 0 otherwise.

New powerful classifiers on the market

- In the last decade, there have been quite a few developments in statistics research and practice.
- Boosting (Freund and Schapire, 1997)
- Bagging (Breiman, 1996) and random forest (Breiman, 2001)
- Boosted and bagged (in combination with random forest) decision trees are believed to be the most powerful off-the-shelf multivariate classifiers!

AdaBoost

Freund and Schapire, 1997

- Combines weak classifiers by applying them sequentially
- At each step enhances weights of misclassified events and reduces weights of correctly classified events
- Iterates as long as weighted misclassified fraction less than 50%
- Final decision: weighted vote of all classifiers

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

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

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

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

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

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

AdaBoost - II

- Formally, AdaBoost can be derived from exponential loss:

$$L(y, f(x)) = \exp(-yf(x))$$

$$(\beta_K, G_K) = \arg\min_{\beta, G} \sum_{i=1}^{N} \exp[-y_i (f_{K-1}(x_i) + \beta G(x_i))]$$

one can show that

$$\beta_K = \frac{1}{2} \log \frac{1-\varepsilon_K}{\varepsilon_K}$$

$$w_i^{(K)} = w_i^{(K-1)} \exp(-\beta_K y_i G_K(x_i))$$

- AdaBoost implements a weak learning algorithm:

$$\varepsilon \le 2^C \prod_{K=1}^{C} \sqrt{\varepsilon_K (1-\varepsilon_K)} \le \exp\left(-2 \sum_{K=1}^{C} \left(\frac{1}{2} - \varepsilon_K \right)^2 \right)$$

- A nice probabilistic property:

$$\frac{P(y=+1 \mid x)}{P(y=-1 \mid x)} = \exp(2 f(x))$$

that is, the ratio of probability of signal over the probability of background at a given point in space is directly given by the AdaBoost output

Bagging (Breiman, 1996)

- Acronym for Bootstrap AGGregatING
- One bootstrap replica = N points drawn with replacement out of sample of size N
- Idea: build many decision trees on independently-drawn bootstrap replicas of the training sample
- Classify new data by the majority vote of the built trees
- Unlike AdaBoost, this is a parallel algorithm
- Boost or bag?
  - Boosting generally gives better predictive power but bagging tends to perform better in the presence of outliers and noise
  - Bauer and Kohavi, “An empirical comparison of voting classification algorithms”, Machine Learning 36 (1999)
  - Short answer: Try both

Random Forest (Breiman, 2001)

- "[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
- http://www.stat.berkeley.edu/users/breiman/RandomForests/
- How does it work? Makes decision trees less correlated with each other => increases the overall classification power of the majority vote
- If the number of randomly selected features at each decision split is small, the algorithm is much faster than simple bagging

What is so good about boosted and bagged decision trees?

- Training stability
  - can easily handle strongly correlated inputs
  - can easily handle mixed input types (continuous+discrete)
  - can easily handle irrelevant inputs
- Do not suffer from the "curse of dimensionality"
  - Data become sparse in many dimensions
  - Methods that depend on distances between sample points (kernel-based methods, support vector machines, radial basis functions, nearest neighbor methods and density estimators) become unstable
- Scale linearly with the number of dimensions
- A perfect tool for use in many dimensions

	Neural Net	RBF	SVM	Trees (CART)	Boosted and bagged trees	MARS	k-NN	VAB
Predictive power	●	●	●	●	●	●	●	●
Ability to deal with irrelevant inputs	●	●	●	●	●	●	●	●
Interpretability	●	●	●	●	●	●	●	●
Curse of dimensionality	●	●	●	●	●	●	●	●
Computational scalability with adding new dimensions	●	●	●	●	●	●	●	●
Training stability	●	●	●	●	●	●	○	●
Response time	●	●	●	●	●	●	●	●

● good ● fair ● poor ● horrible

Scalability of training time in high dimensions

- Decision tree $O(DN \log N)$
- Boosted or bagged decision trees $O(TDN \log N)$
- Random Forest $O(TFN \log N)$
- Boosted binary splits $O(TDN)$
- Standard neural net with one hidden layer $O(THDN)$
  - to capture the data structure in many dimensions, you typically need $H \propto D$ and more than one hidden layer is needed!
  - scales as $O(D^2)$ or worse

D – number of dimensions

N – training sample size

T – number of training cycles

H – number of NN hidden units

F – number of selected features for Random Forest

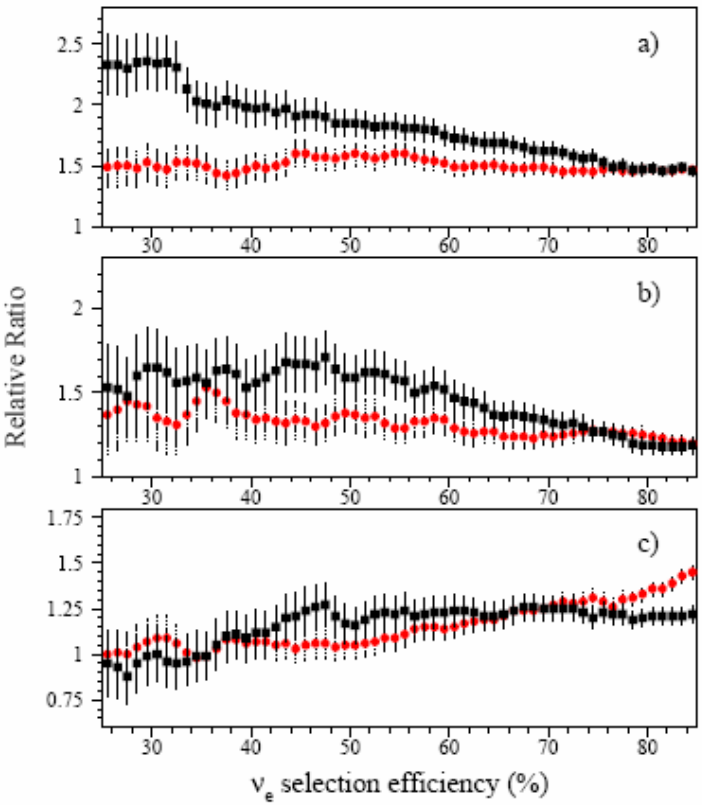
Boosted and bagged decision trees at Phystat 2005

- Byron Roe et al.
  - boosted decision trees for PID at MiniBoone
  - implementation: their own, as far as I can tell
- Sarda et al.
  - decision trees, random forest and a bunch of other methods to separate $\gamma p \rightarrow \Lambda K^{*+}$ from background processes at CLAS
  - implementation: WEKA (open source Java)
- Narsky (presented by Harrison Prosper)
  - boosted and bagged decision trees for 3 various physics analyses at BaBar and D0
  - implementation: StatPatternRecognition

PID at MiniBoone with BDT

Byron Roe et al.,

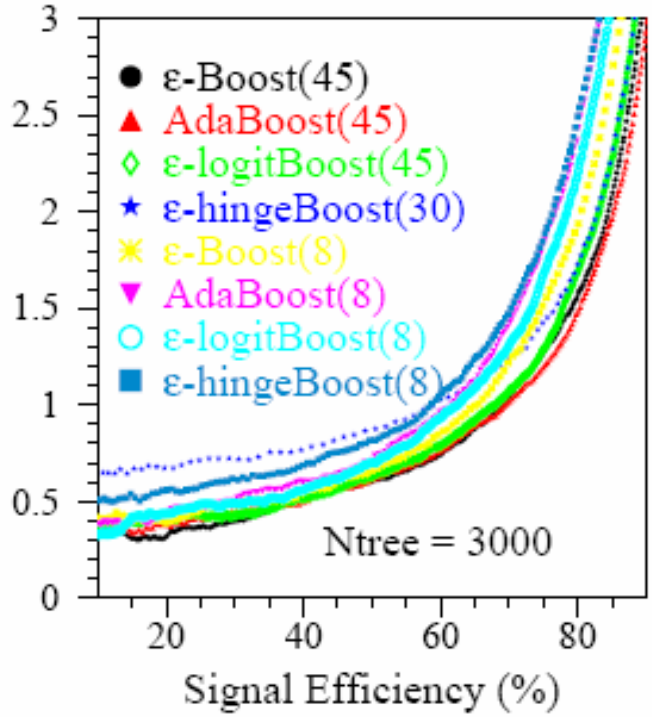
physics/0408124, physics/0508045



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).

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



Software

- Various HEP collaborations stick to various HEP tools:
  - BaBar – Stuttgart Neural Network Simulator (B0/B0bar tagging) and Jetnet
  - D0 – TerraFerma, Jetnet, C2.4 (decision tree), Bayesian NN
  - WEKA is becoming popular (open source Java)
    - http://www.cs.waikato.ac.nz/ml/weka/ (U of Waikato)
    - not to mix with http://www.weka.net.nz/ (New Zealand's disability information web site)
- There is a great abundance of software for multivariate classification floating around:
  - C, Fortran, R, Java, C++ etc
- Professional statisticians who don't want to buy commercial software typically opt for R
- If you want to find a piece of software, browse software repositories (or talk to me)

Statistical Resources on the Web

- http://www.pa.msu.edu/people/linnemann/stat_resources.html
  - lots of links; probably the most complete list maintained by a physicist
- No central repository for statistics software in HEP
- Repositories maintained by other communities:
  - StatLib at CMU http://lib.stat.cmu.edu/
  - StatCodes at Penn State http://www.astro.psu.edu/statcodes/
  - and more!!!

StatPatternRecognition, a C++ package for multivariate classification

- Deliver advanced classification tools in a convenient C++ framework to HEP researchers
- Why C++? Ultimate flexibility – not just run executables but use classes in your analysis code
- Why write a new package instead of downloading one off internet?
  - must be free
  - must be C++
  - must implement boosted and bagged decision trees
  - must be easily extendable for HEP needs
- I haven't found a package that satisfies these requirements

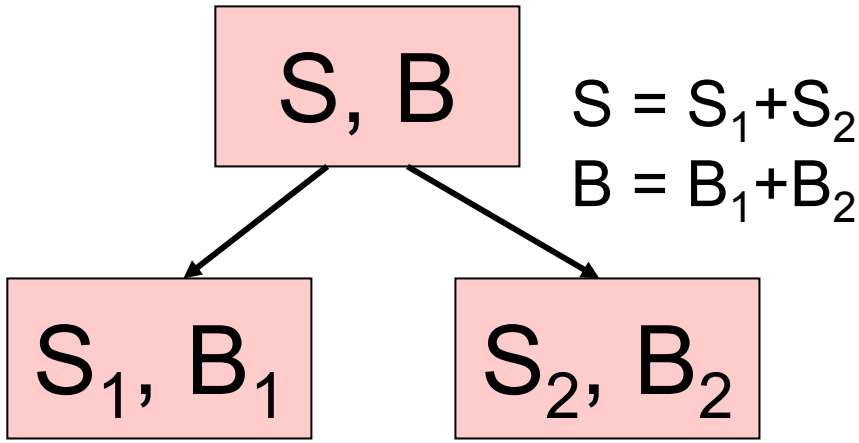
What's in StatPatternRecognition – I

- Classifiers:
  - binary split
  - linear and quadratic discriminant analysis
  - decision trees (original implementation)
  - bump hunting (PRIM, Friedman & Fisher)
  - AdaBoost
  - bagging and random forest
  - AdaBoost and Bagger are capable of boosting/bagging any classifier implemented in the package!
  - interfaces to SNNS feedforward Neural Net and radial basis function Neural Net – not for training! Just for reading the saved net configuration and classifying new data.

What's in StatPatternRecognition – II

- Methods of general use:
  - bootstrap
  - cross-validation
  - estimation of data means, covariance matrix and kurtosis
  - test of zero correlation between two elliptically distributed variables
- Framework
  - tools for imposing selection criteria
  - tools for saving input/output into HBOOK or ROOT
  - input data can be read either from Ascii or Root
  - SPR accepts weighted data
  - can save classifier configuration into an ascii file and resume training from the saved configuration
  - OO design – the package can be extended by supplying new implementations to existing interfaces

Decision trees in StatPatternRecognition



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

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:

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

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/√S+B, 90% Bayesian UL, and 2*(√S+B- √B).**

Status of StatPatternRecognition

- Bulk of the package written in January-July 2005; still adding various features but at a slower pace
- Presented at:
  - two notes posted at physics archive in mid July
  - Fermilab Workshop on Statistical Software Repositories, August
  - Phystat, September 2005
  - plenary session at BaBar Collaboration Meeting, Sept 2005
- 8 subscribers outside BaBar (installation is easy!)
  - Harrison Prosper: “I was able to get your package up and running. You've created a very elegant and simple interface. So simple, in fact, I was able to wrap (automatically!) all your classes to build a Python module. I ran both the AdaBoostDecisionTree executable as well as a Python version thereof without problems.”
- The package is in BaBar CVS
  - at least 3 BaBarians tried it and told me that it worked fine for them; more expressed interest
  - Jan Strube (U Oregon) provided a tool for reading input data directly from Root

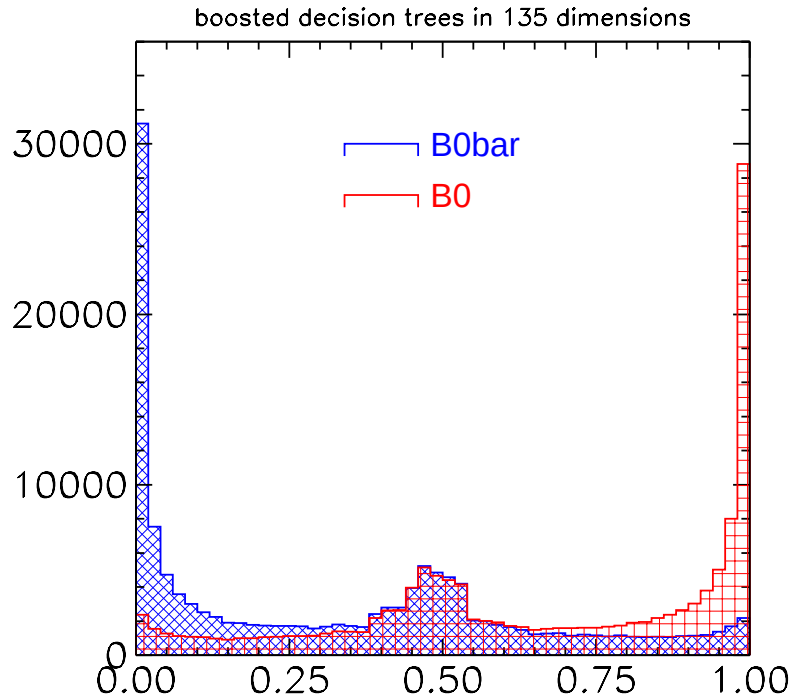
How can you learn the package?

- Documentation
  - physics/0507143, *StatPatternRecognition: A C++ Package for Statistical Analysis of High Energy Physics Data*
  - physics/0507157, *Optimization of Signal Significance by Bagging Decision Trees*
- http://www.hep.caltech.edu/~narsky/spr.html

```
Contents of README
------------------
1. Documentation
2. Classifiers
        2.1  AdaBoost
        2.2  Bagging
        2.3  Decision trees
        2.4  Bump hunting
        2.5  Fisher
        2.6  Interfaces to SNNS
        2.7  Combining classifiers
        2.8  Recommendations: What
classifiers to use and when
        2.9  How to choose parameters for
boosted and bagged decision trees
        2.10 Cross-validation
3. Data and filters
4. Imposing cuts
5. Data input
6. Output
7. Installation instructions for people
outside BaBar
8. Summary
```

Physics Examples - 1

All results shown for test data.



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

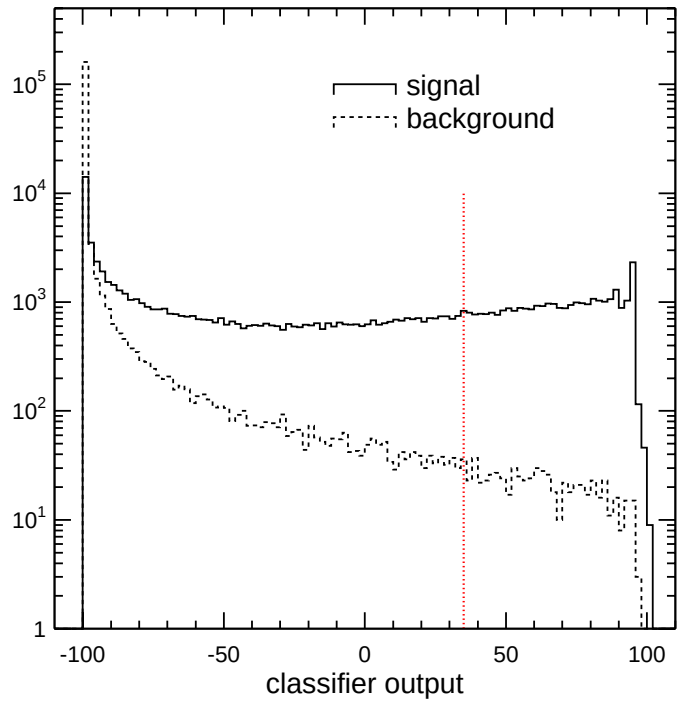
Time needed to train 50 boosted trees on 500k events = 1+ day.

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. More details in Saturday workshop.

Separation of $B \to \gamma l \nu$ from combined background at BaBar by bagged decision trees optimizing the signal significance with 11 input variables.

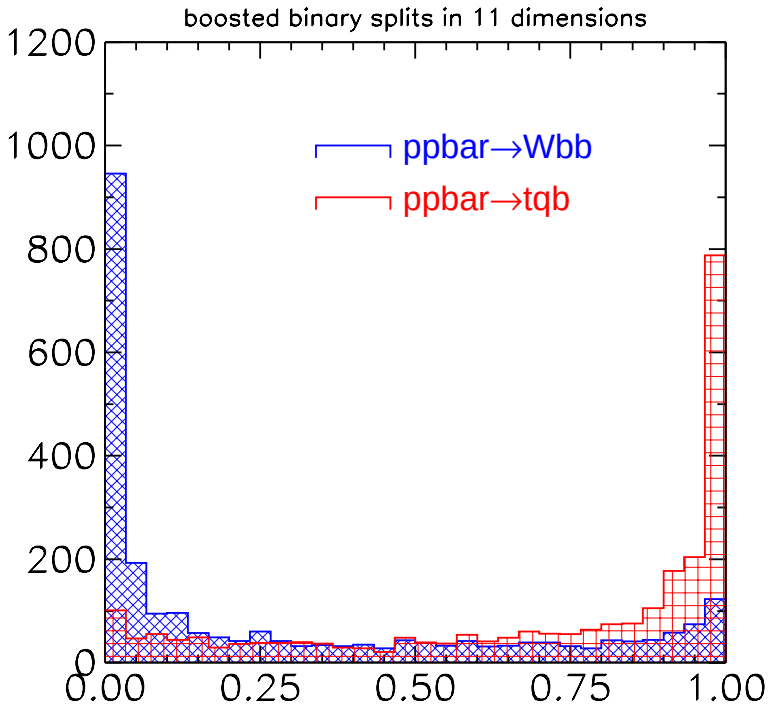
Time needed to train 100 bagged trees on 500k events = several hours.

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



Physics Examples - 2

All results shown for test data.



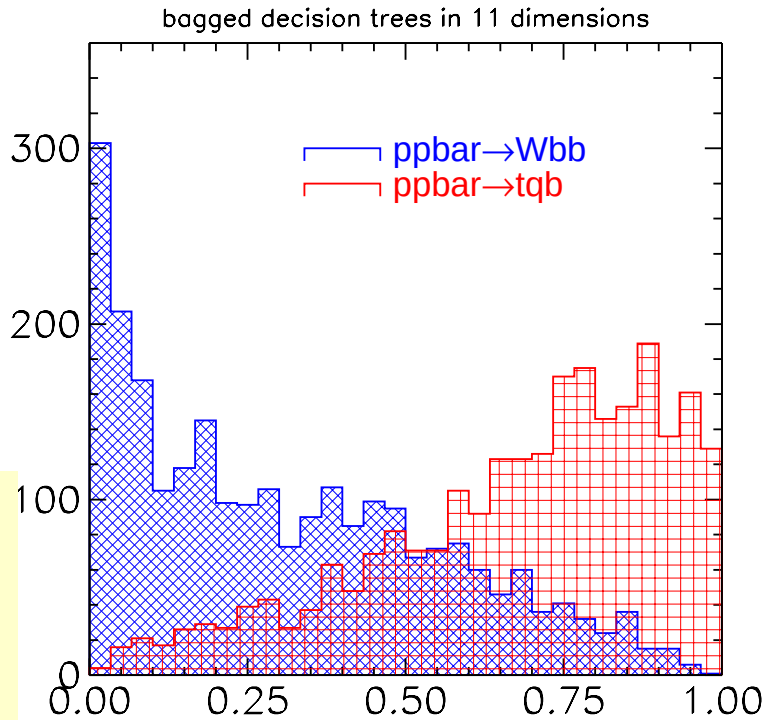
Separation of ppbar→tqb from ppbar→Wbb by boosted binary splits in 11 dimensions.

Time needed to train 1100 binary splits on 5000 events = 2 seconds on noric.

(D0 data from Harrison Prosper)

Separation of ppbar→tqb from ppbar→Wbb by bagged decision trees in 11 dimensions.

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

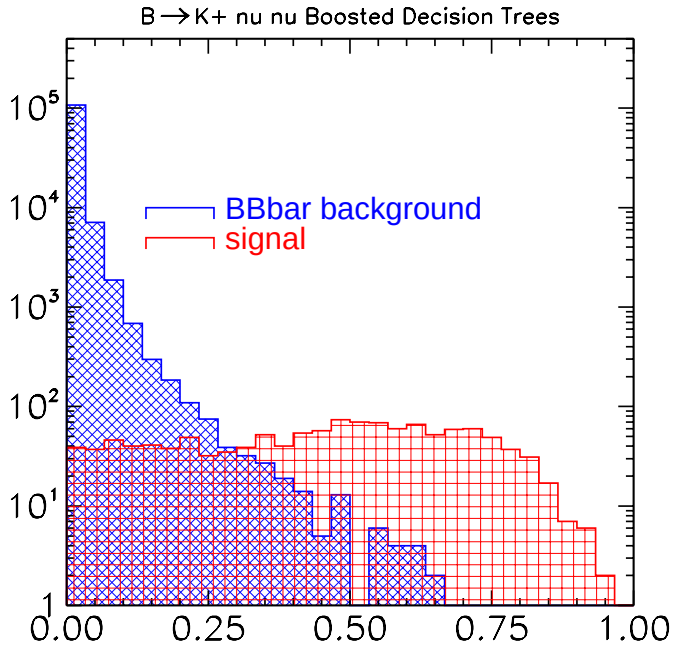


D0 uses Bayesian Neural Nets for this problem.

According to Harrison, gives a similar performance. There is, however, a substantial gain in CPU speed: The Bayesian NN algorithm takes several hours!!!

Physics Examples - 3

All results shown for test data.



Separation of $B^+ \to K^+ \nu\nu$ from B+B- and B0B0bar background by boosted decision trees in 61 dimension.

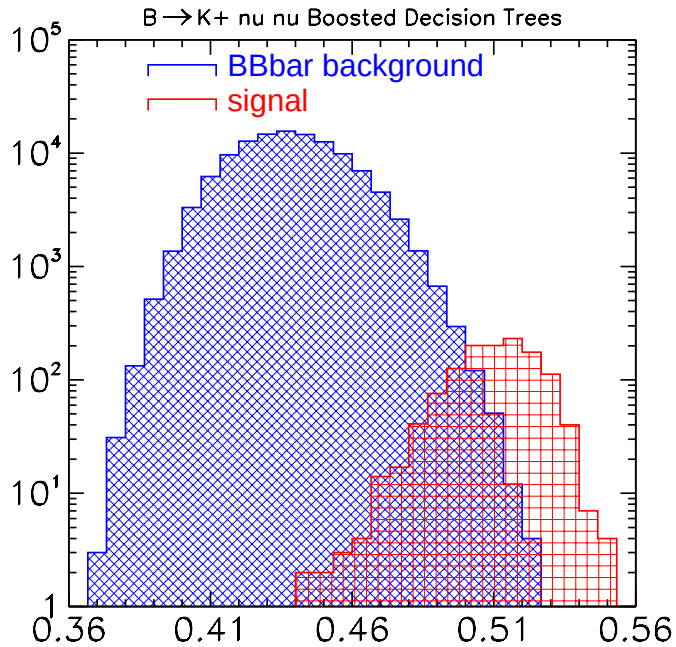
Time needed to train 70 decision trees on 120k events = several hours (~1/3 of xlong limit).

A very preliminary conclusion: compared to the published $B^+ \to K^+ \nu\nu$ analysis, signal efficiency is increased by 1.7 at the same background rejection level.

Separation of $B^+ \to K^+ \nu\nu$ from B+B- and B0B0bar background by boosted binary splits in 61 dimension.

Time needed to train 3050 splits on 120k events = several minutes on noric.

Performance is somewhat worse than that of the boosted decision trees.



Muon PID challenge (Kevin Flood, BaBar)

A bottle of French champagne if I can improve the pion suppression by 3% at the same muon efficiency.

The challenge won't be resolved until Kevin shows results obtained with the Neural Net… but I have a feeling the bottle is mine.

2<P<4 GeV	60%	70%	80%	85%
boosted splits	0.60	0.78	1.25	1.56
boosted decision trees	0.65	0.92	1.30	1.61
Random Forest	0.54	0.70	0.97	1.29

0.5<P<2 GeV	60%	70%	80%	85%
boosted splits	1.24	2.00	4.07	6.13
boosted decision trees	1.33	2.17	4.12	6.20
Random Forest	0.92	1.49	3.01	4.79

Future of (or a wish list for) pattern recognition in HEP

- More input variables: from a few to a few hundred
- Community better educated about classification algorithms
- A common HEP software repository
- More unified software tools – either adopted or developed within HEP

Summary

- Lots of classifiers out there. Need to know a bit about learning machines to choose the right one for the problem at hand.
- StatPatternRecognition is a convenient and well-tested tool. Hopefully will gain some recognition in the HEP community.