StatPatternRecognition: User Guide

Ilya Narsky, Caltech

Also see
http://www.slac.stanford.edu/BFROOT/www/Organization/CollabMtgs/2005/detSep05/Wed4/narsky.pdf

Classifiers in StatPatternRecognition

- binary split (aka “binary stump”, aka “cut”)
- linear and quadratic discriminant analysis
- decision trees
- bump hunting (PRIM, Friedman & Fisher)
- AdaBoost (Freund and Schapire)
- bagging and random forest (Breiman)
- variant of arc-x4 (Breiman)
- multi-class learner (Allwein, Schapire and Singer)
- combiner of classifiers
- interfaces to Stuttgart Neural Network Simulator feedforward backprop Neural Net and radial basis function Neural Net – not for training! Just for reading the saved net configuration and classifying new data

in red – will be covered today

Other tools in StatPatternRecognition

- bootstrap
- validation and cross-validation (choosing optimal parameters for classifiers)
- estimation of data means, covariance matrix and kurtosis
- goodness-of-fit evaluation using decision trees (Friedman, Phystat 2003)
- reading from Ascii or Root
- saving input/output into Hbook or Root
- filtering data (imposing cuts, choosing input variables and classes)

in red – will be covered today

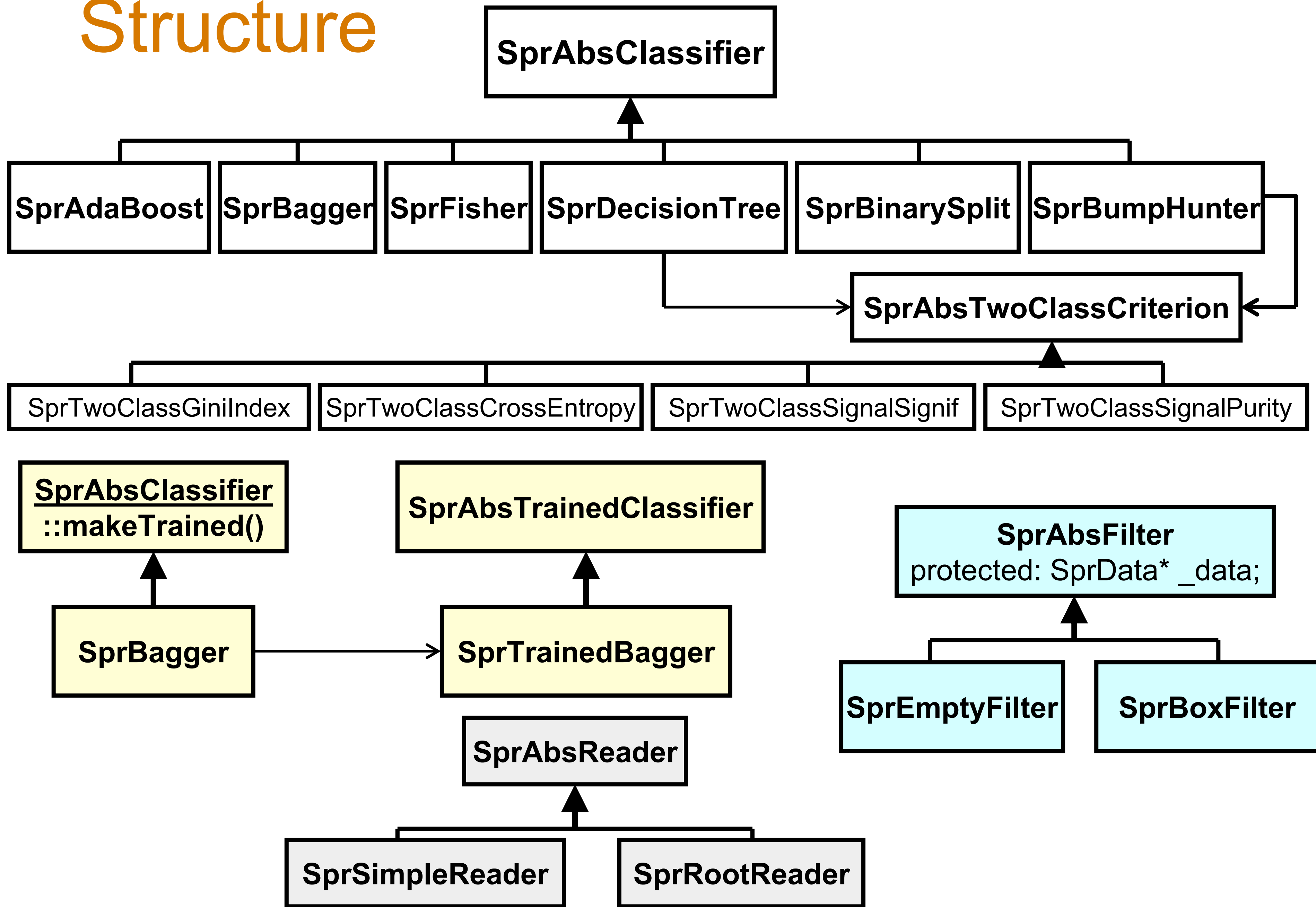
Message for today

- SPR is a package for multivariate classification
  - as in "dozens or hundreds of variables"
  - hardly expect any improvement over, e.g., 5D data classified by a neural net
  - however, if you have 20+ input variables, it is almost guaranteed that SPR will outperform NN
- If you care about classification quality at the level of a few percent:
  - must try both boosted decision trees and random forest (no clear prediction for which one will work better in your particular case)
  - must do a thorough job of selecting classifier parameters by using (cross-)validation
- **Use as many variables as you can and invest time in choosing classifier parameters!**

Status of SPR

- All statements and examples in the present talk are based on tag V02-03-04
  - I recommend updating to latest V-tag regularly
- Updates since September meeting:
  - support for reading input data from Root (thanks to Jan Strube)
  - a couple of new classifiers implemented (a hybrid of boosting and random forest, arc-x4)
  - SprTopdownTree, a faster implementation of decision tree. Random forest has become much faster!
  - computation of quadratic and exponential loss for cross-validation
  - goodness-of-fit method added (although not tested on physics)
  - treatment of multi-class problems
    - multi-class learner implemented
    - easy selection of input classes for two-class methods

Structure



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.
- Use this algorithm if you want to find a rectangular signal region
- Optimization criteria for the algorithm:
  - plugged through an abstract interface

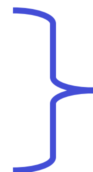
Optimization criteria – I

```
SprBumpHunter(SprAbsFilter* data,
              const SprAbsTwoClassCriterion* crit,
              int nbump,       // number of bumps
              int nmin,        // min number of events per bump
              double apeel);   // peel parameter
```

```
class SprAbsTwoClassCriterion {
public:
/*
    Return figure of merit.
    wcor0 - correctly classified weighted fraction of background
    wmis0 - misclassified weighted fraction of background
    wcor1 - correctly classified weighted fraction of signal
    wmis1 - misclassified weighted fraction of signal
  */
  virtual double fom(double wcor0, double wmis0,
                     double wcor1, double wmis1) const = 0;
}
```

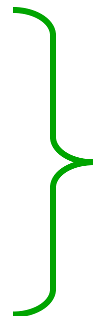
Optimization criteria – II

SprTwoClassGiniIndex
SprTwoClassIDFraction
SprTwoClassCrossEntropy



"technical" (for decision tree optimization)

SprTwoClassSignalSignif
SprTwoClassBKDiscovery
SprTwoClassTaggerEff
SprTwoClassPurity
SprTwoClassUniformPriorUL90



for physics

Bump hunter in SPR

Ready-to-go executable: SprBumpHunterApp

- **`SprBumpHunterApp -b 1 -n 100 -x 0.1 -c 2 -w 0.001 -f bump.spr -t validation.pat -o training.hbook -p validation.hbook training.pat`**

Find one bump with at least 100 events by optimizing the signal significance (-c 2) using peel parameter 0.1. Signal will be weighted down by 0.001. The found bump will be saved into bump.spr. Signal significance will be evaluated for validation data. Training and validation data will be saved into training.hbook and validation.hbook.

```
Found box 0
==============================================
Validation FOM=0.270037
Content of the signal region:   W0=911  W1=8.187   N0=911  N1=8187
==============================================
```

- **`cat bump.spr`**

```
Bumps: 1
-------------------------------------------------------
Bump      0    Size 2      FOM=0.263075   W0=957        W1=8.173      N0=957        N1=8173
Variable                    x0    Limits        -6.97225        6.8124
Variable                    x1    Limits          -6.373       -3.55095
```

Optimization of the bump hunter parameters

- Only one degree of freedom – peel parameter
- Choose the optimal peel parameter by maximizing the selected figure of merit on validation data
- Test bump hunter performance on an independent test sample

Choice of classifier parameters

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

Training error seriously underestimates true model error.

Validation error somewhat underestimates true model error.

Test error is an unbiased estimate of the true model error.

In principle, if the validation set is fairly large and the FOM estimate is based on statistically large subsets, the test error should be close to the validation error. Hence, the test stage can be omitted.

Use with caution!!!

What if you have barely enough data for training?

- Resampling of training data:
  - cross-validation
  - bootstrap
- Basic idea: make many test samples by resampling the original training set
- Do not use cross-validation for big samples: slow and unnecessary
- More info:
  - Hastie, Tibshirani and Friedman “The Elements of Statistical Learning”; chapter 7 “Model Assessment and Selection”

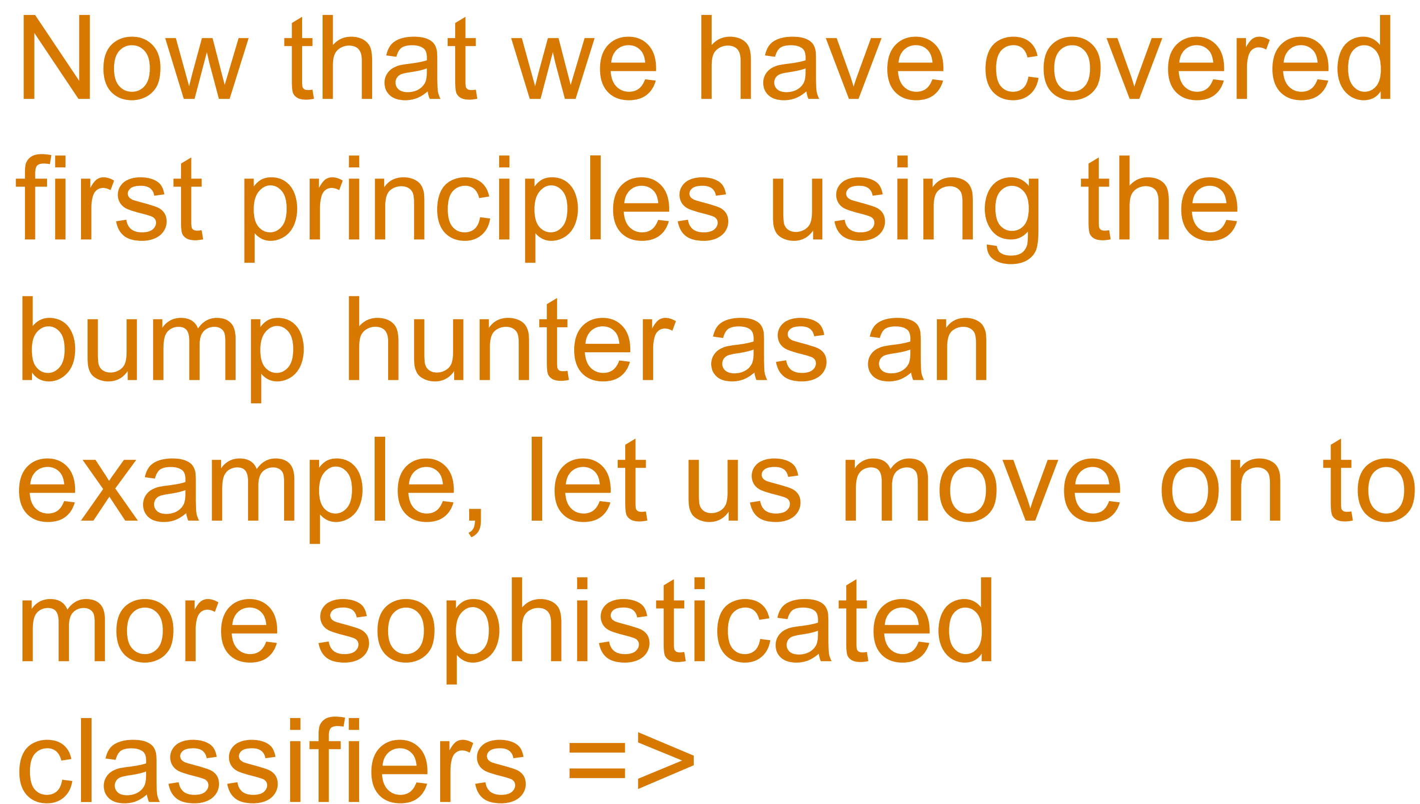
in red – will be covered today

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

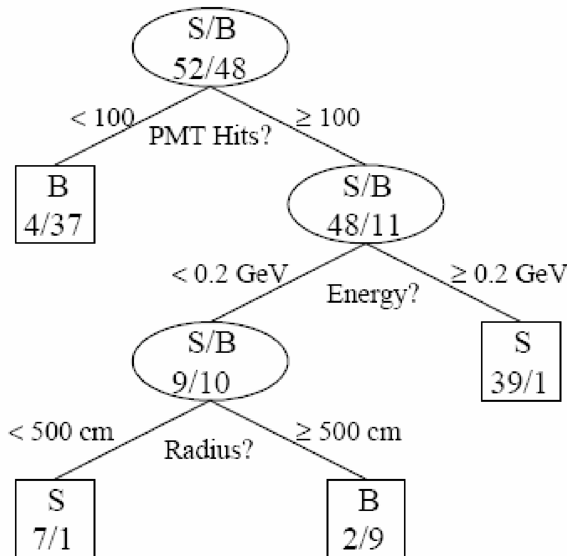
$$m(i):\ \{1,2,\ldots,N\} \mapsto \{1,2,\ldots,M\} \qquad R_{CV} = \frac{1}{N}\sum_{i=1}^{N} L\left(y_i, f_{-m(i)}(x_i)\right)$$

- 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
- I typically choose M=5-10


Now that we have covered first principles using the bump hunter as an example, let us move on to more sophisticated classifiers =>

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

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

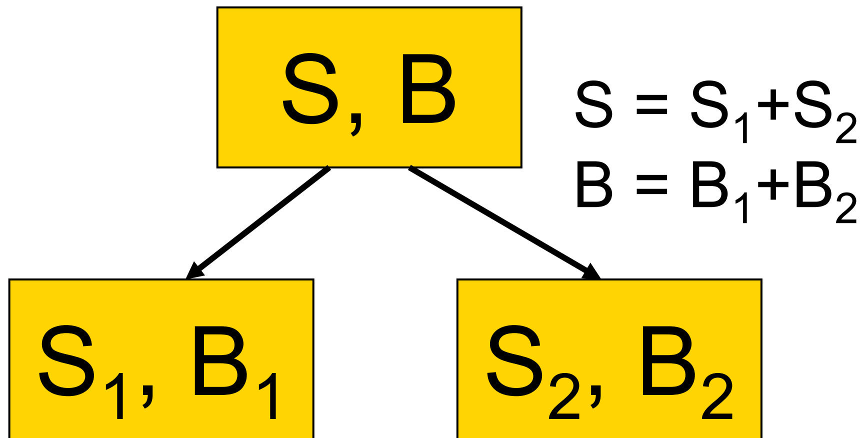
Stopping criteria:

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

Decision tree output:

- **discrete scoring: 1 if an event falls into a signal node, 0 otherwise**
- **continuous scoring: for example, signal purity, s/(s+b)**

Decision trees in StatPatternRecognition



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).

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 tree in SPR

Ready-to-go executable: SprDecisionTreeApp

- **SprDecisionTreeApp –n 100 –c 2 –m –w 0.1 –f tree.spr –t validation.pat –o training.hbook –p validation.hbook training.pat**

Build a decision tree with at least 100 events per leaf by optimizing the signal significance (-c 2). Signal will be weighted down by 0.1. The found leaf nodes will be merged (-m). Signal significance will be evaluated for validation data. Training and validation data will be saved into training.hbook and validation.hbook.

```
Optimization criterion set to Signal significance S/sqrt(S+B)
Decision tree initialized with minimal number of events per node 100
Included 5 nodes in category 1 with overall FOM=6.48006    W1=198.4 W0=739    N1=1984 N0=739
===================================
Validation FOM=5.78557
===================================
```

```
SprDecisionTree(SprAbsFilter* data,
                const SprAbsTwoClassCriterion* crit,
                int nmin, bool doMerge, bool discrete,
                SprIntegerBootstrap* bootstrap=0);
```

How does the tree merge nodes?

- Merging algorithm:
  - sort all nodes by signal purity in descending order => vector of N nodes
  - start with the node with the highest purity and add nodes from the sorted list sequentially; at each step compute the overall FOM => vector of N FOM's
  - choose the element with the largest FOM and use the corresponding combination of nodes
- This algorithm only makes sense for asymmetric FOM's:
  - SprTwoClassSignalSignif, SprTwoClassBKDiscovery, SprTwoClassTaggerEff, SprTwoClassPurity, SprTwoClassUniformPriorUL90
  - use –m (doMerge=true) option only for these!!!

Optimization of the decision tree parameters

- Only one degree of freedom – minimal number of events per leaf node
- Choose the optimal leaf size by maximizing the selected figure of merit on validation data
- Test decision tree performance on an independent test sample

Cross-validation:

```
➢ SprDecisionTreeApp -c 2 -w 0.1 -m -x 5 -q
"2,5,10,20,50,100,200,500" training.pat

Cross-validated FOMs:
Node size=       2        FOM=    1.95209
Node size=       5        FOM=    2.36133
Node size=      10        FOM=    2.59742
Node size=      20        FOM=    2.55526
Node size=      50        FOM=    2.55019
Node size=     100        FOM=     2.5308
Node size=     200        FOM=     2.5293
Node size=     500        FOM=    2.27798
```

← I would choose this one (Node size= 20)

Output of a decision tree

➢ `cat tree.spr`

```
Trained Decision Tree: 5      signal nodes.    Overall FOM=6.48006    W0=739         W1=198.4
N0=739          N1=1984

-------------------------------------------------------
-------------------------------------------------------
Signal nodes:
-------------------------------------------------------
Node        0     Size 8         FOM=5.35067     W0=161          W1=83.7         N0=161          N1=837
Variable            LeadingBTaggedJetPt     Limits            -1.0045           2.8145
Variable           LeadingUntaggedJetPt     Limits           -0.33205           4.1695
Variable                    Pt_Jet1Jet2     Limits    -1.79769e+308           2.0915
Variable           InvariantMass_AllJets    Limits             0.3405           5.6125
Variable                 DeltaRJet1Jet2     Limits    -1.79769e+308            2.724
Variable  Cos_BTaggedJetAllJets_AllJets     Limits            -1.7245     1.79769e+308
Variable                 BTaggedTopMass     Limits            -0.9932           2.3885
Variable                      QTimesEta     Limits            -2.3785     1.79769e+308
-------------------------------------------------------
Node        1     Size 11        FOM=1.13555     W0=31           W1=7            N0=31           N1=70
........
```

- For each node, shows only variables on which cuts are imposed, e.g., for Node 0 only 8 out of 11 variables are shown.
- Gives a full list of leaf nodes in the order of decreasing signal purity.

Bump hunter vs decision tree

Bump Hunter	Decision Tree
finds one rectangular region that optimizes a certain FOM	finds a set of rectangular regions with a globally optimized FOM
cross-validation not available	can cross-validate
a conservative and slow algorithm (shrinkage/expansion); the speed is controlled by the peel parameter	a greedy and fast algorithm; the speed and size of the tree is controlled by the minimal number of events per leaf node

AdaBoost – I

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% *and* the requested number of training cycles is not exceeded

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

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

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

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

$$\text{weight of classifier K}: \beta_K = \frac{1}{2}\log\left(\frac{1-\varepsilon_K}{\varepsilon_K}\right)$$

To classify new data: weighted vote of all classifiers

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

AdaBoost – II

- Formally, 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:

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

- Margin theory: test error keeps decreasing even after the training error turns zero

Boosting in SPR

Ready-to-go executables: SprAdaBoostBinarySplitApp and SprAdaBoostDecisionTreeApp

- `SprAdaBoostDecisionTreeApp -n 100 -l 1000 -f adatree.spr -t validation.pat -d 5 training.pat`

Boost 100 decision trees with at least 1000 events per leaf and save the trained classifier configuration into adatree.spr. Display exponential loss for validation data every 5 trees. By default, Gini index will be used for tree construction (I don't recommend changing optimization FOM for boosted trees ever).

- `SprAdaBoostDecisionTreeApp -n 0 -r adatree.spr -o test.hbook -s test.pat`

Read AdaBoost configuration from adatree.spr and apply it to test data. Save test data and AdaBoost output into test.hbook

- `SprAdaBoostBinarySplitApp -n 100 -f adasplit.spr training.pat`
- `SprAdaBoostBinarySplitApp -n 0 -r adasplit.spr -o test.hbook -s test.pat`

Train and test boosted binary splits with 100 splits per dimension.

Boosted splits vs boosted trees

Boosted splits	Boosted trees
Applies decision splits on input variables sequentially: split 1 on variable 1, split 2 on variable 2 etc; in the end goes back to variable 1 and starts over.	Tree finds an optimal decision split by choosing among all input variables.
Splits are applied to the whole training set.	Splits are applied recursively to build the tree structure.
Training is very fast and robust, typically at the expense of less-than-perfect predictive power.	Training is slower but the predictive power is usually superior to that of boosted splits.

Optimization of AdaBoost parameters

- Boosted binary splits are very robust. Validation stage can be omitted. Choose at least several dozen splits per input dimension.
- Boosted trees have two degrees of freedom
  - minimal leaf size => needs to be found by (cross-)validation
  - number of trees built => the more, the better (can become CPU-expensive for large samples with many input variables)

Cross-validation:

```
➢ SprAdaBoostDecisionTreeApp –n 20 –g 1 –x 5 –q
"100,200,500,1000,2000,3000" training.pat
```

("-g 1" displays quadratic loss; "-g 2" will display exponential loss)

```
Cross-validated FOMs:
Node size=      100         FOM=  0.212612
Node size=      200         FOM=   0.19664
Node size=      500         FOM=  0.191809
Node size=     1000         FOM=  0.185834
Node size=     2000         FOM=  0.210297
Node size=     3000         FOM=-1.79769e+308
```

$$L_{\text{qua}}(y, f(x)) = (y - f(x))^2$$

$$L_{\text{exp}}(y, f(x)) = \exp(-yf(x))$$

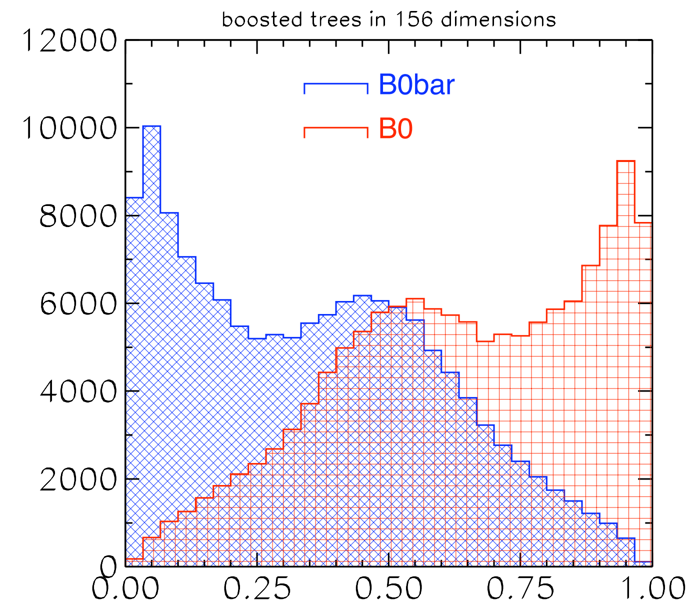
Things to remember

- Boosted trees typically benefit from large leaves. Choose leaf size ~5-50% of the number of events in the smallest class.
- If the leaf size if poorly chosen, AdaBoost performance degrades as you build more and more trees. You can monitor this by looking at validation data.
- Several hundred trees are more than enough for all practical problems I looked at. You can keep adding more trees, but AdaBoost error won't go down significantly.
- Boosted splits are robust and incredibly fast. However, typically give a worse model error than boosted trees or random forest.

How to look at AdaBoost output

- The mysterious “-s” option
  - forces AdaBoost to display output in the range (-infty,+infty)   (“standard” AdaBoost)
  - by default AdaBoost output is in range [0,1]
- Allows to use probabilistic properties of AdaBoost

```
➢ paw
➢ h/fil 1 test.hbook 4096
➢ n/pl 1.(1/(1+exp(-2*ada))) i=0
➢ n/pl 1.(1/(1+exp(-2*ada))) i=1 ! ! ! s
```



How to read saved AdaBoost configuration

- "-r" option for AdaBoost executables
  - stands for "resume", not "read"
  - after reading the saved configuration, you can resume training…

```
> SprAdaBoostDecisionTreeApp -n 100 -l 1000 -r adatree_n100.spr -f adatree_n200.spr training.pat
```

  - …or just classify events

```
> SprAdaBoostDecisionTreeApp -n 0 -r adatree_n100.spr -o test.hbook test.pat
```

- in C++

```
SprAdaBoostTopdownTreeReader reader;
bool readStatus = reader.read("my_input_file");
SprTrainedAdaBoost* trained = reader.makeTrained();
vector<double> input_point = …;
double r = trained->response(input_point);
```

OR

```
SprAdaBoost ada(…,ncycles=100,…);
reader.setTrainable(&ada);
bool trainStatus = ada.train();
```

Random Forest (Breiman, 2001)

- "[Random Forest] is unexcelled in accuracy among current algorithms." – Leo Breiman, http://www.stat.berkeley.edu/users/breiman/RandomForests/
- Random forest = bagging + random selection of input variables for each decision split
- Bagging = bootstrapping of training points
  - draw N points out of sample of size N => one bootstrap replica
  - build many decision trees on bootstrap replicas of the training sample and classify new data by the majority vote
- Random selection of input variables
  - for each decision split, randomly select d out of D variables
  - d is set by the user

Random Forest in SPR

Ready-to-go executable: SprBaggerDecisionTreeApp

```
➢ SprBaggerDecisionTreeApp -n 100 -l 10 -s 6 -g 1 -f
bagtree.spr -t validation.pat -d 5 training.pat
```

Build 100 decision trees with at least 10 events per leaf and save the trained classifier configuration into bagtree.spr. Display quadratic loss (-g 1) for validation data every 5 trees. Randomly select 6 variables to be considered for each decision split.

```
➢ SprBaggerDecisionTreeApp -n 0 -r bagtree.spr -o test.hbook
test.pat
```

Read random forest configuration from bagtree.spr and apply it to test data. Save test data and random forest output into test.hbook

Bagger interface is very similar to that of AdaBoost. Use similar syntax for saving classifier configuration, reading it back from file, resuming training, cross-validation etc.

Optimization of Random Forest

- Random forest has 3 degrees of freedom
  - minimal leaf size => needs to be found by (cross-)validation
  - number of randomly drawn input variables for each decision split => needs to be found by (cross-)validation
  - number of trees built => again, the more the better

Cross-validation:

```
➢ SprBaggerDecisionTreeApp –n 10 –s 6 –g 1 –x 5 –q
"1,2,5,10,20,50,100" training.pat
```

(randomly select 6 variables per split and display quadratic loss)

```
Cross-validated FOMs:
Node size=       1      FOM=   0.17846
Node size=       2      FOM=  0.175674
Node size=       5      FOM=  0.172535
Node size=      10      FOM=  0.173444
Node size=      20      FOM=  0.174015
Node size=      50      FOM=  0.176392
Node size=     100      FOM=  0.178914
```

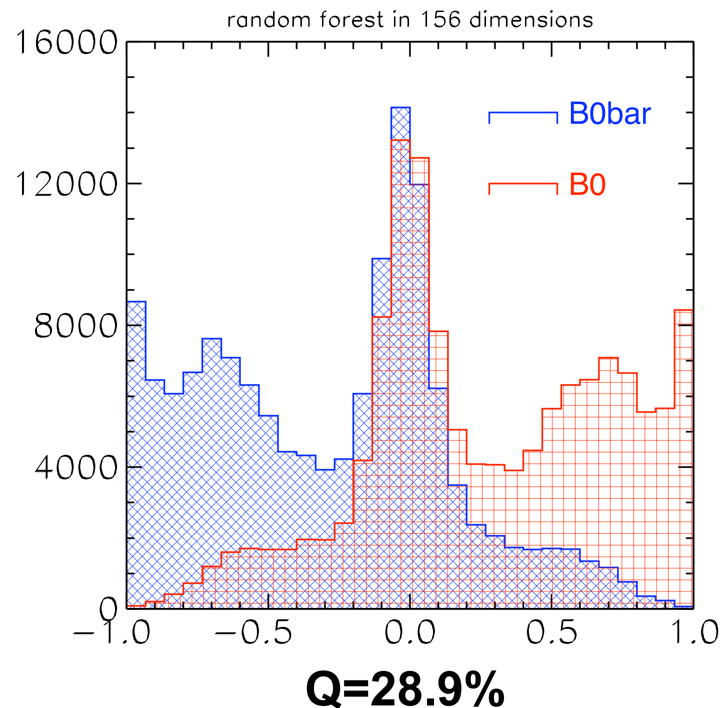
How many input variables to choose at random?

- In my experience, best performance is obtained when "-s" command-line option (nFeaturesToSample) is chosen in the range [D/2,D] (D = dimensionality)
  - Note that sampling is with replacement, so if you set nFeaturesToSample=D, on average only 63% of input variables will be used for each decision split
- If you choose nFeaturesToSample small, random forest will train faster…
- …but you may not save any CPU cycles because you may need to build more trees to obtain comparable performance.
- To compare performance, I recommend using: nFeaturesToSample*nTreesToBuild = const

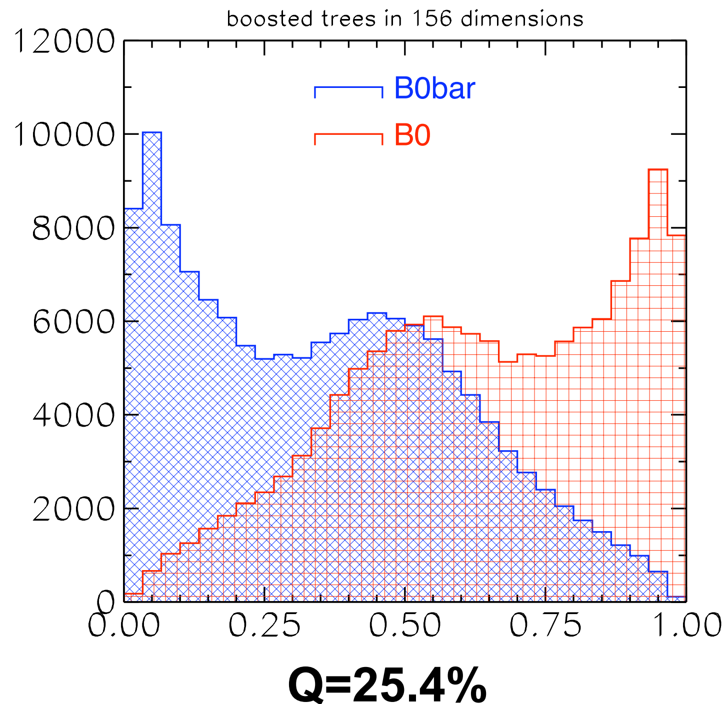
Things to remember

- Random forest typically favors very small leaves and huge trees. Choose leaf size ~0.1% of the number of events in the smallest class.
- Training time is comparable with that for boosted trees: RF trees are bigger but no CPU is spent on reweighting events.
- Classification of new data is generally slower by RF than by boosted trees. But it is far below the level when you need to worry.
- Unlike boosted trees, you can try to "random-forest" asymmetric FOM's such as, e.g., signal significance (-c command line option). For some problems this can produce a significant improvement.

Boosted trees or random forest?



Q=28.9%



Q=25.4%

There is no generic answer to this question. Try both!!!

Multi-class problems

- Two popular strategies for K classes:
  - one against one: build K*(K-1)/2 classifiers
  - one against all: build K classifiers
- User can easily implement these strategies "by hand":
  - "- y" command line option for most executables
  - SprAbsFilter::irreversibleFilterByClass(const char* classes)

Example: 3 input classes (0, 1, and 2)

➢ **`SprXXXApp –y "1,2" …`**

(separate class 1 from class 2)

➢ **`SprXXXApp –y ".,2" …`**

(separate class 2 from all others)

By convention, first class in the list is treated as background and second class as signal. For symmetric optimization FOM's, it does not matter; for asymmetric FOM's, it does.

Multi-class learner

Allwein, Schapire and Singer, 2000

- Reduce multi-class problem to a set of two-class problems using an indicator matrix
- For example, a problem with 4 classes:

$$Y_{\text{ONE-VS-ALL}} = \begin{pmatrix} 1 & -1 & -1 & -1 \\ -1 & 1 & -1 & -1 \\ -1 & -1 & 1 & -1 \\ -1 & -1 & -1 & 1 \end{pmatrix} \quad Y_{\text{ONE-VS-ONE}} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & 1 & 0 \\ 0 & -1 & 0 & -1 & 0 & 1 \\ 0 & 0 & -1 & 0 & -1 & -1 \end{pmatrix}$$

- Indicator matrix = K*L matrix for K classes and L binary classifiers

Classification by the multi-class algorithm

- Compute user-defined loss (either quadratic or exponential) for each row of the indicator matrix
  - e.g., compute average quadratic error

$$E_k = \frac{1}{L}\sum_{l=1}^{L}\left(Y_{kl} - f_l(x)\right)^2; \quad k = 1,\ldots,K$$

  - …and assign event X to the class which gives the minimal quadratic error

Multi-class learner in SPR

Ready-to-go executable: SprMultiClassBoostedSplitApp

- **`SprMultiClassBoostedSplitApp -e 1 -y "0,1,2,3,4" -g 1 -f multi.spr -n 100 -t gauss4_uniform_2d_valid.pat gauss4_uniform_2d_train.pat`**

Use the one-vs-all training template (-e 1) to separate the 5 classes from each other and save the trained classifier configuration into multi.spr. Each binary classifier is boosted binary splits with 100 splits per input dimension. After training is completed, display average quadratic loss (-g 1) for validation data.

- **`SprMultiClassBoostedSplitApp -y "0,1,2,3,4" -g 1 -r multi.spr -o multi.hbook gauss4_uniform_2d_valid.pat`**

Read the classifier configuration from multi.spr and apply it to validation data. Save validation data and classifier output into multi.hbook. Quadratic loss for each class will be saved, as well as the overall classification (integer class label).

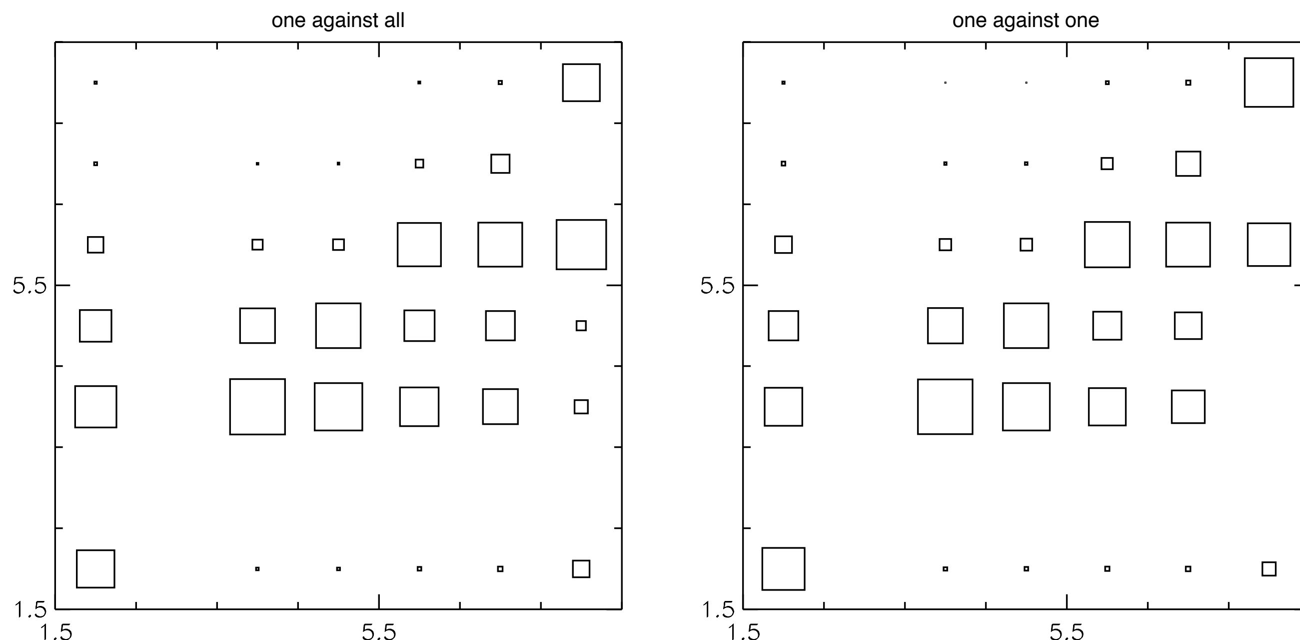
Multi-class learning algorithm is implemented using boosted splits because these are very fast and robust. Training boosted decision trees or random forest requires fine-tuning of the classifier parameters, which can be hardly done for all binary problems at once without loss of accuracy.

Example: $B^+ \rightarrow K^{*+}\nu\nu$ analysis

Disclaimer: a purely hypothetical and silly example of multi-class separation:

- **2 – signal**
- **4 – B+B-**
- **5 – B0B0bar**
- **6 – ccbar**
- **7 – uds**
- **8 – tau+tau-**

Classified category vs true category for test data



Choosing input variables

- Compute correlation between an input variable and the class label
  - SprExploratoryAnalysisApp
- Decision trees can count decision splits on each input variable => discard variables with few hits
  - "-i" option of SprAdaBoostDecisionTreeApp and SprBaggerDecisionTreeApp
  - is useful only if a large fraction of input variables is considered for each split
- Once you decided that some variables are useless, can exclude them from the input list
  - "-z" option for most executables
  - SprAbsReader::chooseAllBut(const set<string>& vars)

Example: for muon PID, the training data are split into bins of momentum and polar angle, but these two variables are not used for classification:

➢ **SprBaggerDecisionTreeApp -z "p,theta"** …

Missing values

- If a certain variable cannot be measured, it is represented by an unphysical value
  - e.g., measure momentum of a lepton candidate in each event: 0 GeV < p < 5 GeV; if no lepton candidate, assign -1
  - by convention adopted in SPR, all missing values should be encoded as values **below** the physical range (e.g., -1 but not 10 in the example above)
- Two strategies to deal with missing values:
  - do nothing (recommended choice)
  - replace missing values with medians of marginal distributions

Why is it ok to do nothing about missing values?

- Decision trees are robust and they work equally well with discrete, continuous and mixed variables
- Abundance of missing values reduces CPU time used by decision trees
  - sorting an array takes $O(N*\log(N))$
  - moving missing values to the beginning of the array takes $O(M)$, and sorting the rest of the array takes $O((N-M)*\log(N-M))$
  - for $N \approx M$ sorting takes $O(N)$ instead of $O(N*\log(N))$
- Adding an extra input variable which is not measured in most events is (mostly) harmless.

Application of SPR to physics analysis

- Search for $B \to K^{(*)} \nu\nu$
  - 60 input variables
  - a significant improvement over rectangular cuts by boosted splits
  - see my presentations at Leptonic AWG meetings
- Muon PID
  - 17 input variables
  - a significant improvement over the neural net by random forest
  - see presentation by Kevin Flood in the PID session
- B0/B0bar tagging
  - 156 input variables
  - my best attempt is currently at Q=28.9% (random forest)
  - worse than official Q=30% but better than Q=25% which I had in September

Benchmarks

- Against R
  - 90k training set in 4D (electron B0/B0bar tagging data)

SprBaggerDecisionTreeApp: 164.360u 5.170s 3:00.55 93.8%

R randomForest: 708.970u 940.830s 31:13.56 88.0%

  - Q from SPR better by ~10%
- Against m-boost (Byron Roe's C package)
  - boosted decision tree results from SPR and m-boost are in very good agreement for 50D data (training set ~50k) with similar tree configurations
  - speed? (under investigation)

What the future holds

- Major interfaces and command-line options are frozen
- The only anticipated development is adding a method for automatic input variable selection (forward selection, backward elimination)
- The package could benefit from aggressive use and bug reports

Invitation/solicitation – I

- So far, all bugs have been found by me; none reported by users.
- The core of the package is fairly well debugged…
- …but flexibility comes with a price – exotic combinations of options are not tested as thoroughly as others
- For example, SprAdaBoostDecisionTreeApp has 24 command-line options. I don’t have manpower to test all possible combinations.
- For example, yesterday I discovered that “-w” option of the executable reweights events in class 1 instead of reweighting events in the 2nd class specified by the “-y” input class list. Such bugs are easy to catch and fix.
- If you use the package, by all means – try to push it to the limit. Use all options. Report any glimpse of problem.

Invitation/solicitation – II

- Planning to submit a NIM paper about the package
- The more practical examples, the better!
- If you applied one of the advanced classifiers (boosted decision trees, random forest or multi-class learner) to your analysis and obtained a result which is significantly better (≥10%) than whatever you had before, feel free to send me your piece for inclusion in the paper.
- Special thanks to a first serious user of the multi-class learning algorithm.

Anti-challenge

- If you are using a neural net, I bet that I can improve your FOM (whatever that might be) by

$$\left(1+\chi_{MISSING}\right)\cdot\left(50-49\exp\left(-\frac{D-20}{100}\right)\right)\%$$

$\chi_{MISSING}$ fraction of missing values in the data

$D \geq 20$ dimensionality

- Bet conditions: you provide your data in ascii format recognized by SPR