Physics 509: Multivariate Analysis

Scott Oser
Lecture #17

Sir Ronald Aylmer Fisher

Statement of the problem

Suppose you have data drawn from two or more similar-looking populations. Most simply, you want to classify each data point into “Type A”, “Type B”, etc. Or, if you can't decide on an event-by-event basis, at least you want to estimate the total number of data points from population A vs. population B.

How do you most effectively separate tell the different types apart?

For example, what animal made this print?

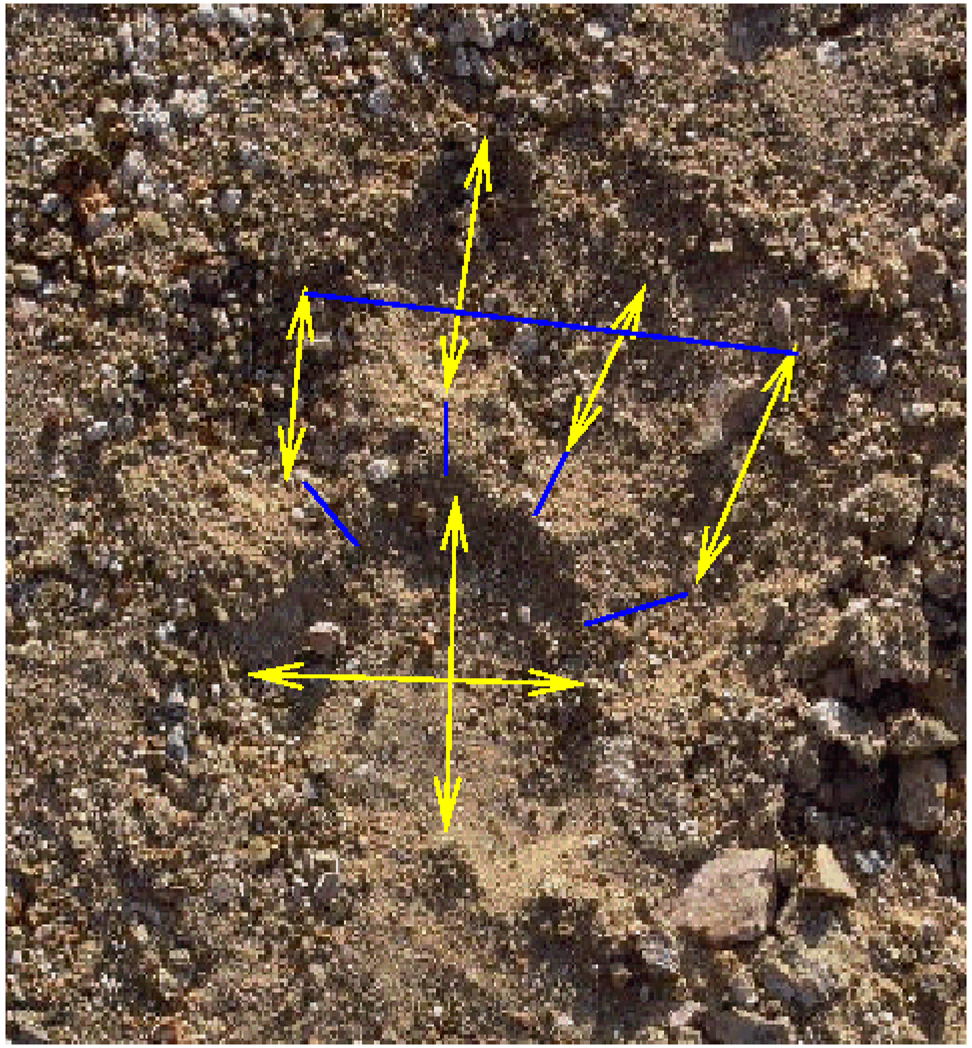
Measure something

Obviously there are lots of things you could measure:

1) How long?
2) How wide?
3) How many toes?
4) Length of the longest toe?
5) Depth of print
6) Length of second-longest toe divided by the area of the palm

You may have lots of data---a multivariate problem

Which quantity or quantities provide the best indication of whether this was a coyote or a mountain lion?



What we want is a test statistic

Some of the measurements will be very helpful in telling what kind of animal made a print (eg. the number of toes ought to at least rule out some of the possibilities, although maybe not with 100% reliability due to noise). Others are likely to be useless---either they're too noisy, or don't provide any meaningful separation between different kinds of animals.

Your enterprising graduate student measures 100 parameters for 1000 different pawprints for several species of animals.

Ideally there should be some way to combine these into a “test statistic” we could use in a hypothesis test. But what combination should we use? Given 100 parameters, there are an infinite number of ways to combine them!

What will most reliably let us distinguish mountain lions from coyotes?

The Neyman-Pearson Lemma

We actually already know how to solve this problem in principle. The Neyman-Pearson lemma says that the most powerful hypothesis test we can do is a test on the likelihood ratio:

$$\frac{P(\vec{t}\,|H_0)}{P(\vec{t}\,|H_1)} > c$$

So it's a simple problem, right? Just use ALL of the available measurements, determine the joint probability distribution for the set of all 100 measured parameters for both hypotheses, and use those PDFs to calculate the likelihood ratio for each paw print.

Problem solved, in principle ...

But in practice ...

The Neyman-Pearson lemma works well when you already know the full joint probability distribution for all of the measurements. But in practice you never do, and you wind up having to estimate these from data.

Let's suppose we have 100 different measurements for each paw print, and let's say we make a coarsely binned PDFs with 10 bins in each variable (i.e. treat each measurement as a discrete variable)

Then to fully specify the PDF we would need to determine 10^{100} values.

Knee-jerk response: send your graduate student out to get more data.

On further reflection: we need to reduce the dimensionality of the data, and develop some kinds of approximations

The linear Fisher discriminant

The Fisher discriminant function is one of the easiest ways to construct a simple test statistic from the data. Let t be a statistic formed from some linear combination of the different measured parameters:

$$t(\vec{x}) = \sum_{i=1}^{n} a_i x_i = \vec{a} \cdot \vec{x}$$

Here the a_i are some coefficients whose value we want to choose so as to maximize the separation between the PDFs $P(t|H_0)$ and $P(t|H_1)$. Start by calculating the mean values and the covariances of the parameters using the available data:

$$\vec{\mu}_k = \int \vec{x} P(\vec{x}|H_k) \; d\vec{x}$$

$$(V_k)_{ij} = \int (x-\mu_k)_i (x-\mu_k)_j P(\vec{x}|H_k) \; d\vec{x}$$

Separation of the linear Fisher discriminant

We can calculate the expectation value and variance of t for each hypothesis:

$$\langle t \rangle_k \equiv \tau_k = \int t\, P(t|H_k)\, dt = \vec{a} \cdot \vec{\mu}_k$$

$$\Sigma_k^2 = \int (t - \tau_k)_j^2 P(t|H_k)\, dt = \vec{a}^T \cdot V_k \cdot \vec{a}$$

We want to increase the separation, so we want to choose the *a*$_i$ so as to maximize $|\tau_0 - \tau_1|$. For example, a useful measure of separation might be

$$J(\vec{a}) = \frac{(\tau_0 - \tau_1)^2}{\Sigma_0^2 + \Sigma_1^2}$$

This function can be maximized as a function of *a*$_{i.}$

Formula for the linear Fisher discriminant

The formula for the coefficients is:

$$\vec{a} \propto (V_0 + V_1)^{-1} \cdot (\vec{\mu}_0 - \vec{\mu}_1)$$

So the procedure is rather simple:

1) Calculate the mean and covariance between the measured parameters using a “training set” of data of each type. For n measured parameters, there are n(n+1)/2 quantities to be determined from the training set.

2) Calculate the set of coefficients using the above formula.

Fisher discriminant for some simple Gaussian variables

In this example, we measure 6 parameters. Suppose they follow 6 uncorrelated Gaussians with unit RMS:

	Mean for A	Mean for B
x1	0	1
x2	0	0
x3	0	1
x4	0	0
x5	0	0
x6	0	0

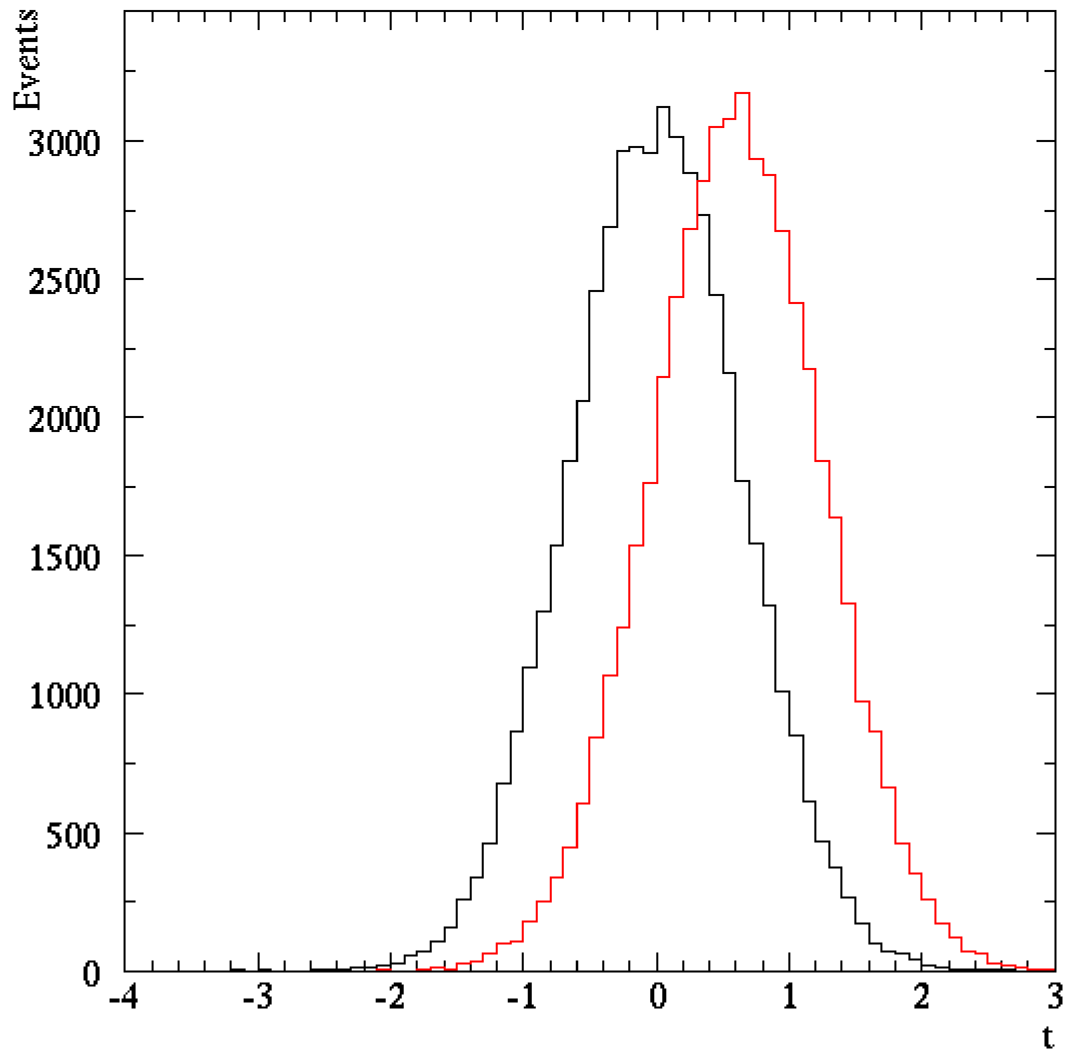
It's clear in this case that all of the separation will come from parameters x_1 and x_3.

Calculating the Fisher coefficients, using 100,000 events of each type to calculate the coefficients:

a_1=+0.4970
a_2=+0.0004
a_3=+0.5002
a_4=+0.0006
a_5= -0.0001
a_6= -0.0005

I get the same result using 10,000 events, and almost the same using 1,000.

Fisher discriminant for some simple Gaussian variables: graphical result



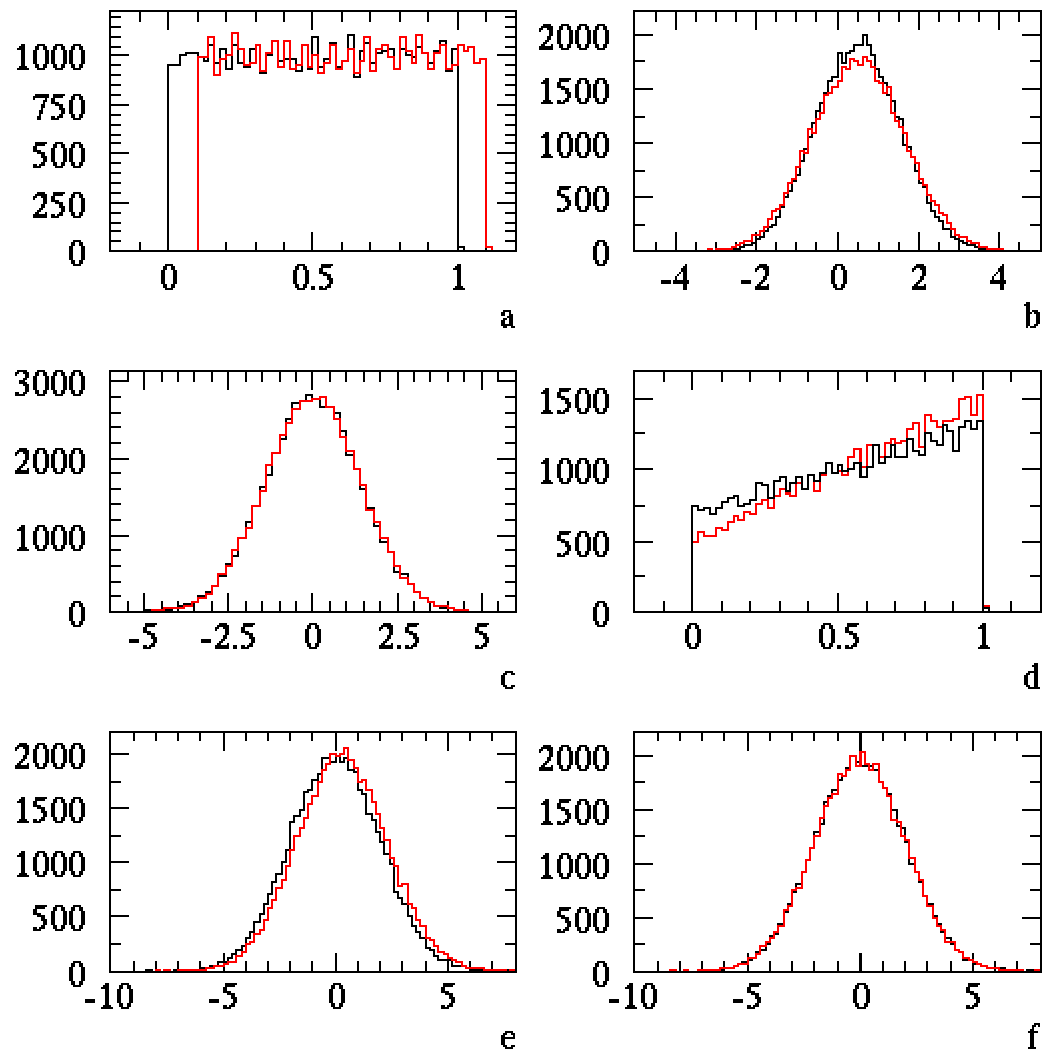
Using the calculated coefficients, the value of t was calculated for each data point and plotted for events of type B (red) and type A (black).

Multidimensional Gaussian distributions in which A and B have identical covariance matrices are a special case: in this case the Fisher discriminant is just as powerful as the full likelihood ratio!

What does a Fisher discriminant actually do?

If you want an intuitive way to think about the Fisher discriminant, it basically tries to draw a rotated coordinate system through the data so that the separation between the data is maximized when projected onto a single axis.

A more complicated data set



The plots to the left show the 1D profile plots for two more complicated data sets. It's clear that variable a and d provide some separation power between the two signals, although others don't look powerful.

Fisher discriminant coefficients reflect this:

$a_1 = \ 0.5728$

$a_2 = -0.0113$

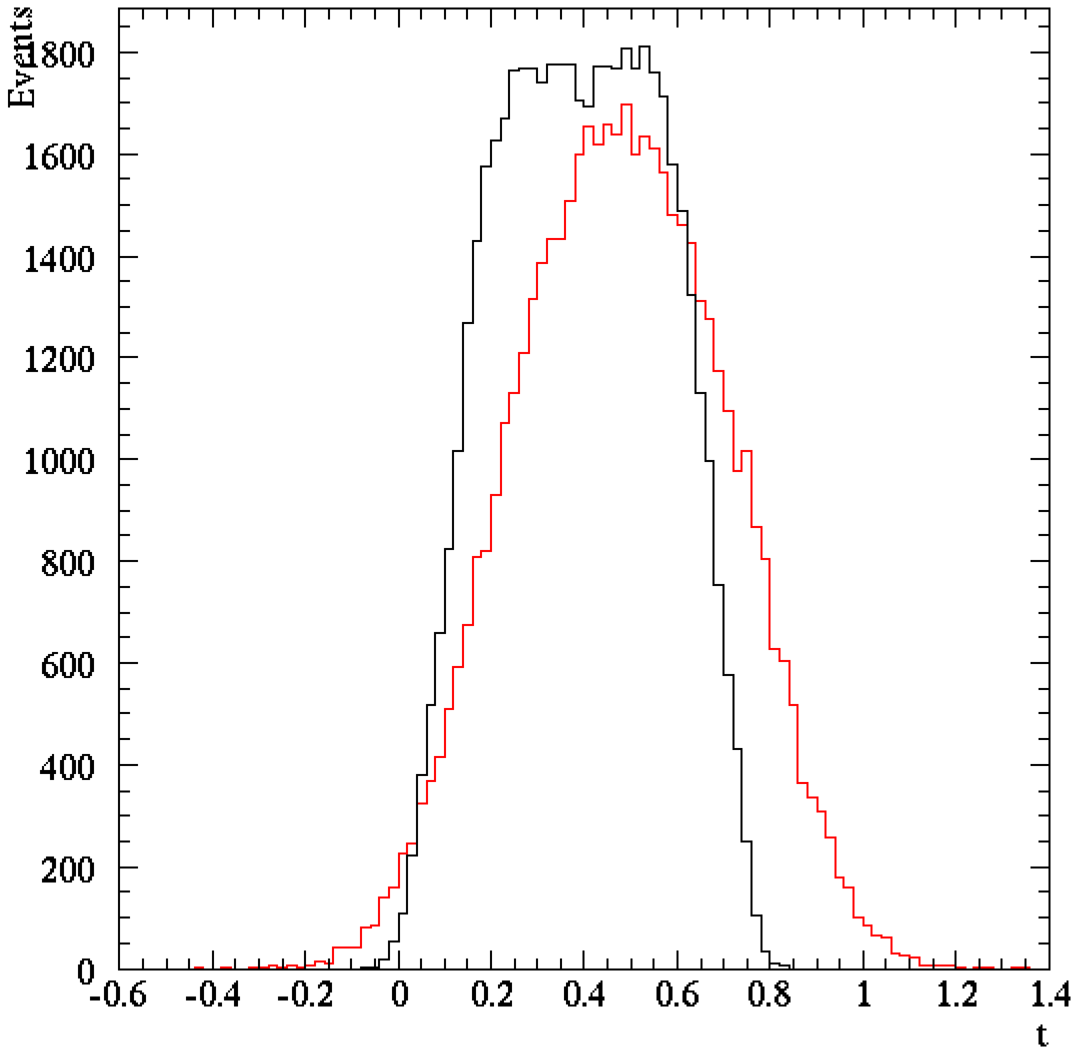
$a_3 = \ 0.0003$

$a_4 = \ 0.1964$

$a_5 = \ 0.0509$

$a_6 = -0.0257$

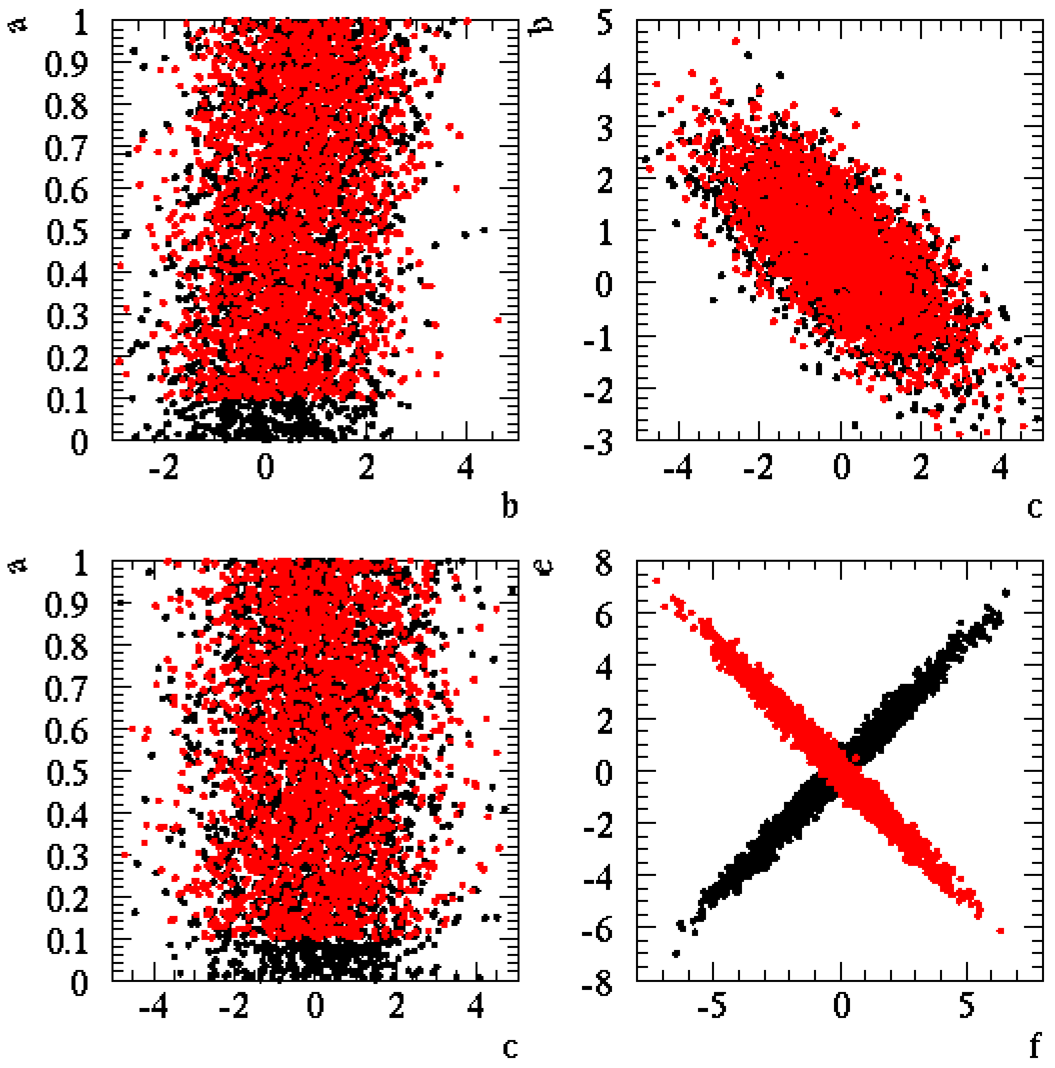
Fisher discriminants for the more complicated data set



The plot to the left shows the separation between the two kinds of events for the linear Fisher discriminant.

Some separation is evident, but it's not that powerful, seemingly. The distributions still overlap considerably.

A closer look at this data set



Scatterplots of:

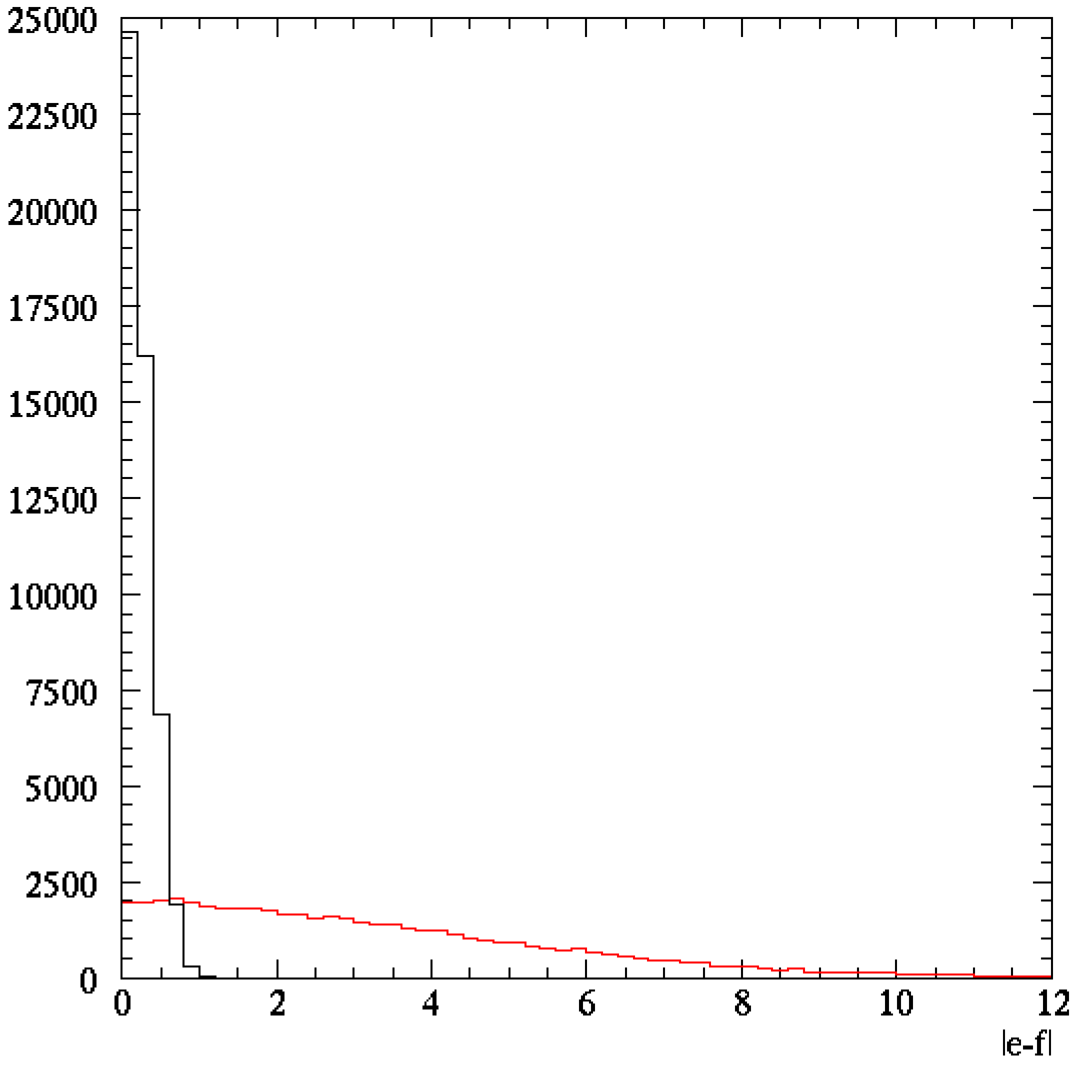
a vs. b
b vs. c
a vs. c
e vs. f

Notice that the scatterplots provide some interesting insight---a cut on |e-f| in particular should provide good separation between a and b.

Notice that you get little separation using e or f alone.

If there are hundreds of variables to choose from , how would you ever find the magic scatter plot? Usually it doesn't even exist!

A fantastic separating variable

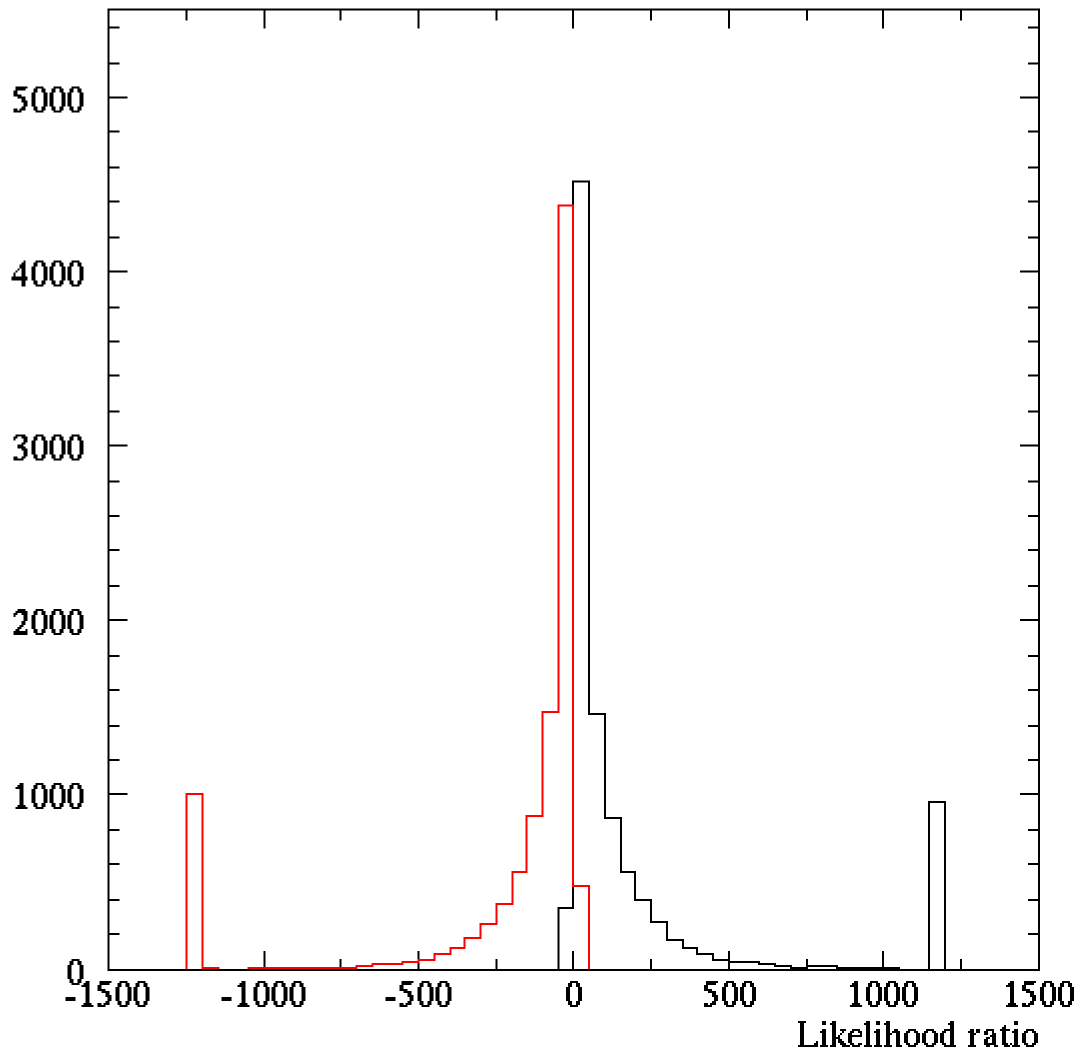


The variable | *e* - *f* | provides much better separation than the linear Fisher discriminant formed from all of the variables. Why?

1) A Fisher discriminant uses only linear variables. A variable like | *e* - *f* | is non-linear, and can be more powerful.

2) A Fisher discriminant is better at handling shifts than differences in widths---not well suited for resolving two Gaussians with same mean but different widths.

Best-case scenario: full multi-dim likelihood ratio



To see how good I could do in the best possible case, I cheated, and used my knowledge of what the actual multi-dimensional PDF to calculate the likelihood ratio using all of the available information.

Separation is nearly perfect!

In real life, there's no way I would really know what the true multi-dimensional PDF was given a reasonable number of statistics. This really is cheating.

Uncorrelated likelihood analysis

In reality I don't know the full PDF a priori. I can instead try to approximate it based on the data, taking into account that I have limited statistics for both classes of events with which to “train” the method.

Simplest approximation is to ignore correlations:

$$P(x_{1,}x_{2,}\ldots x_6) \approx P_{uncorr} = \prod_{i=1}^{6} f_i(x_i)$$

We estimate the different factors by making 1D plots of the data, and making binned PDFs:

Form a likelihood ratio:

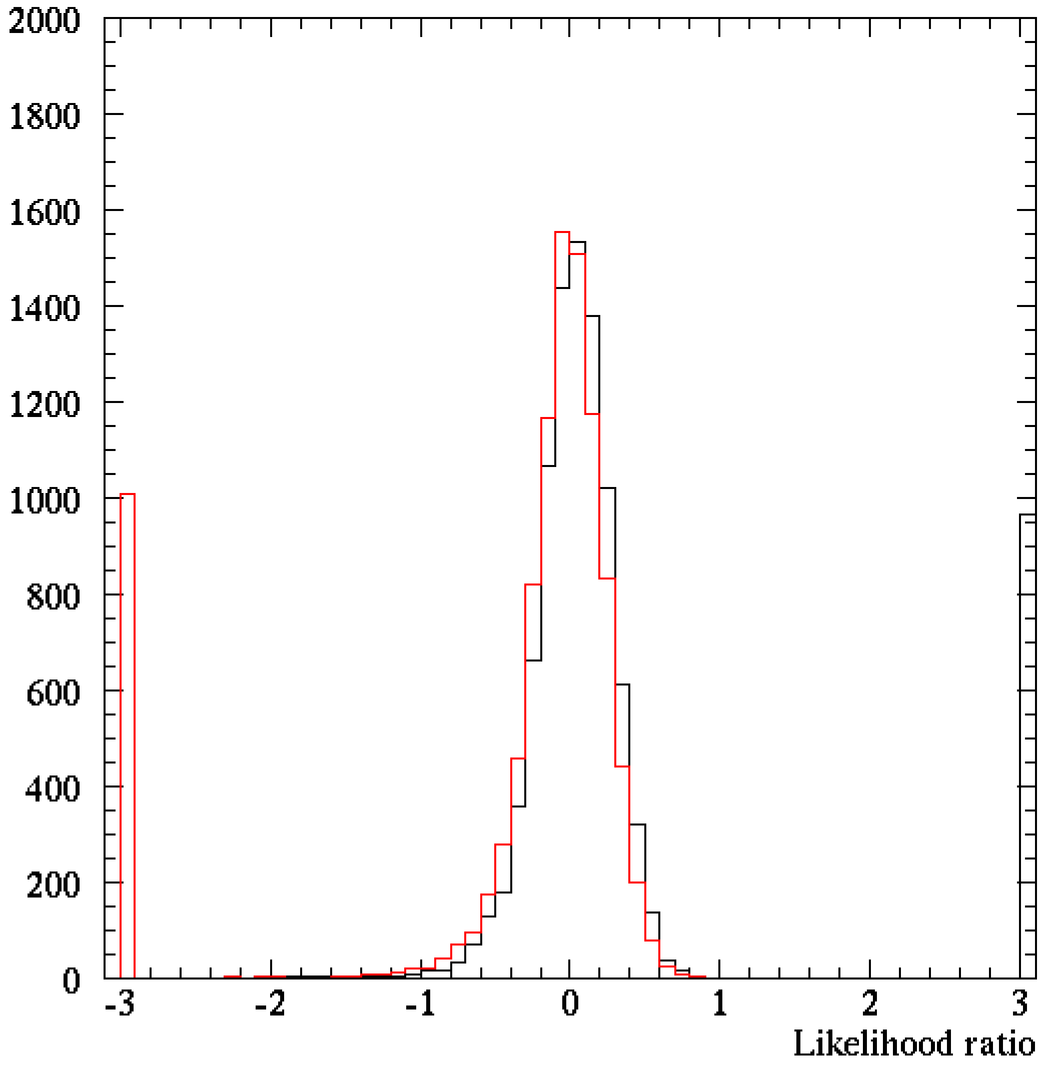
$$\ln\left(\frac{P_{uncorr,A}(\vec{x})}{P_{uncorr,B}(\vec{x})}\right) = \sum_{i=1}^{6} \ln f_{i,A}(x_i) - \sum_{i=1}^{6} \ln f_{i,B}(x_i)$$

Uncorrelated likelihood analysis: results

This plot shows the likelihood ratio found from assuming that the PDF factorizes, ignoring correlations.

It cleanly separates events lying on the extremes of the first distribution, but is pretty crappy overall.

Problem: the factorization assumption ignores correlations, but for these data almost all of the useful information comes from looking at how one parameter varies with another.



Gaussian approximation in likelihood analysis

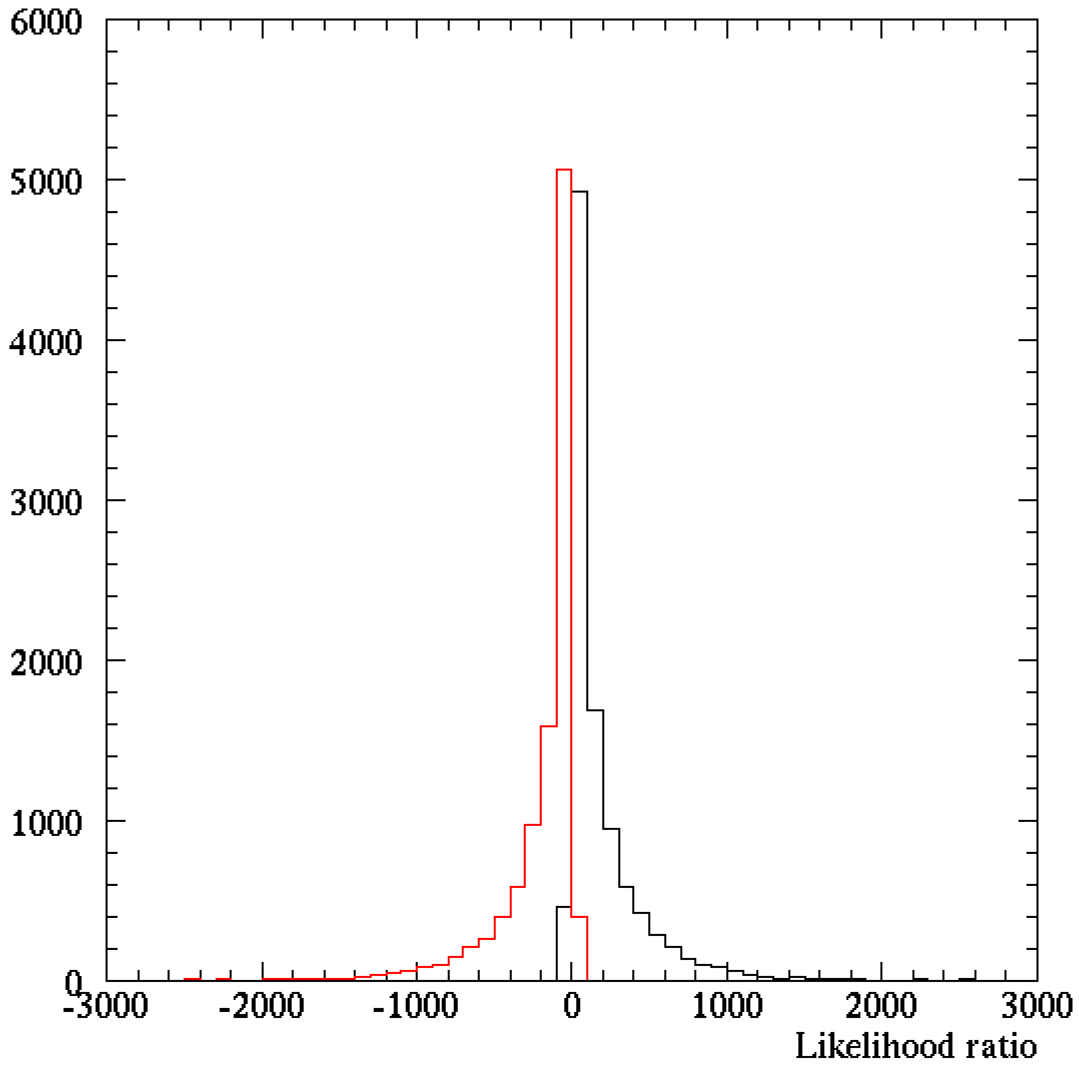
Try to include correlations by using Gaussian approximation for PDFs. Calculate the means and covariance matrix between all 6 parameters for both kinds of events.

$$\Delta\chi^2 = (\vec{x}-\vec{\mu}_A)^T \cdot \boldsymbol{V}_{\boldsymbol{A}}^{-1} \cdot (\vec{x}-\vec{\mu}_A) - (\vec{x}-\vec{\mu}_B)^T \cdot \boldsymbol{V}_{\boldsymbol{B}}^{-1} \cdot (\vec{x}-\vec{\mu}_B)$$

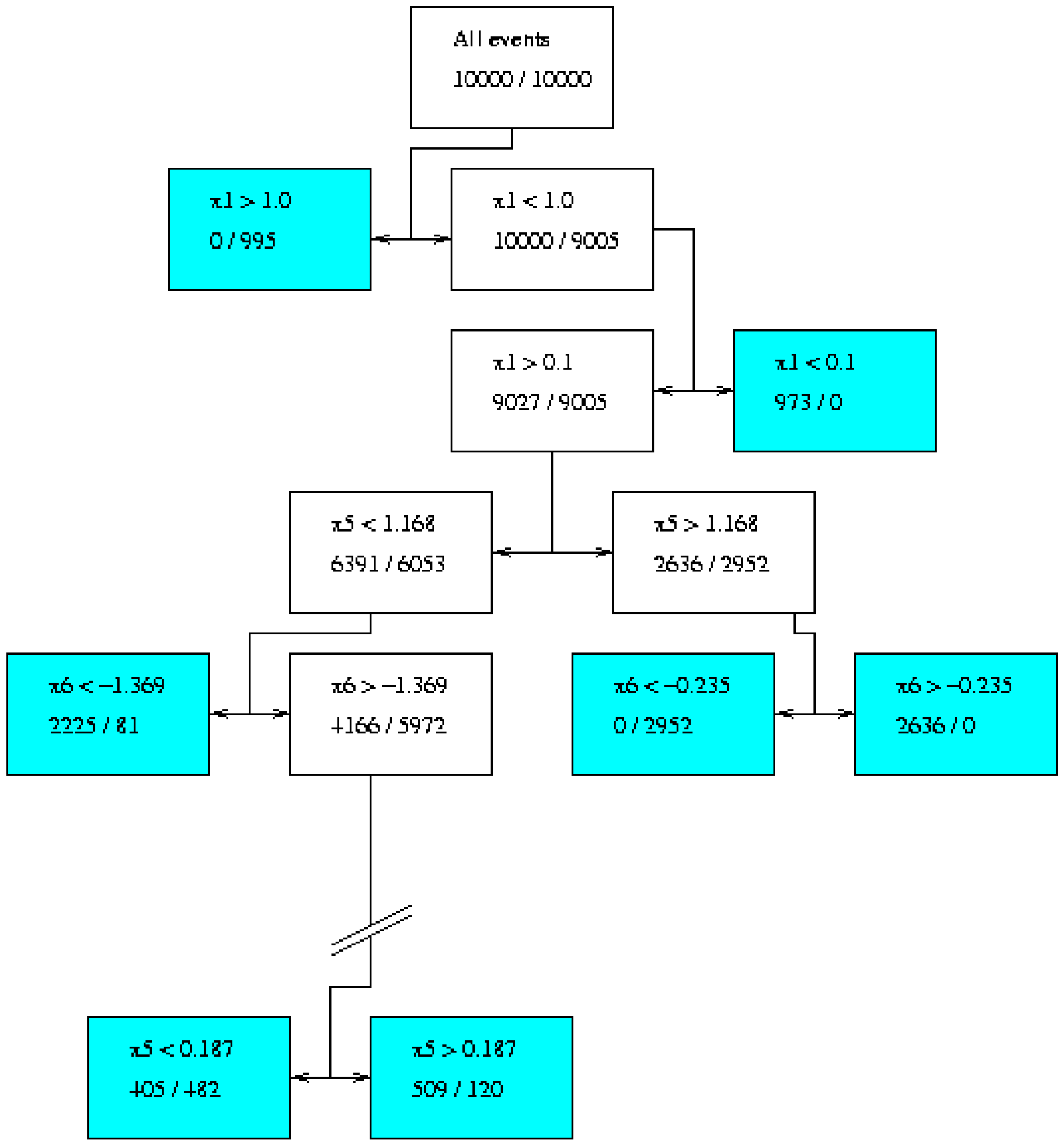
You do need to measure the covariances for all parameters.

Almost as good as full likelihood ratio for this data:
True LR error rate: 4.19%
Gaussian approx: 4.29%

This is atypically good!



Decision tree

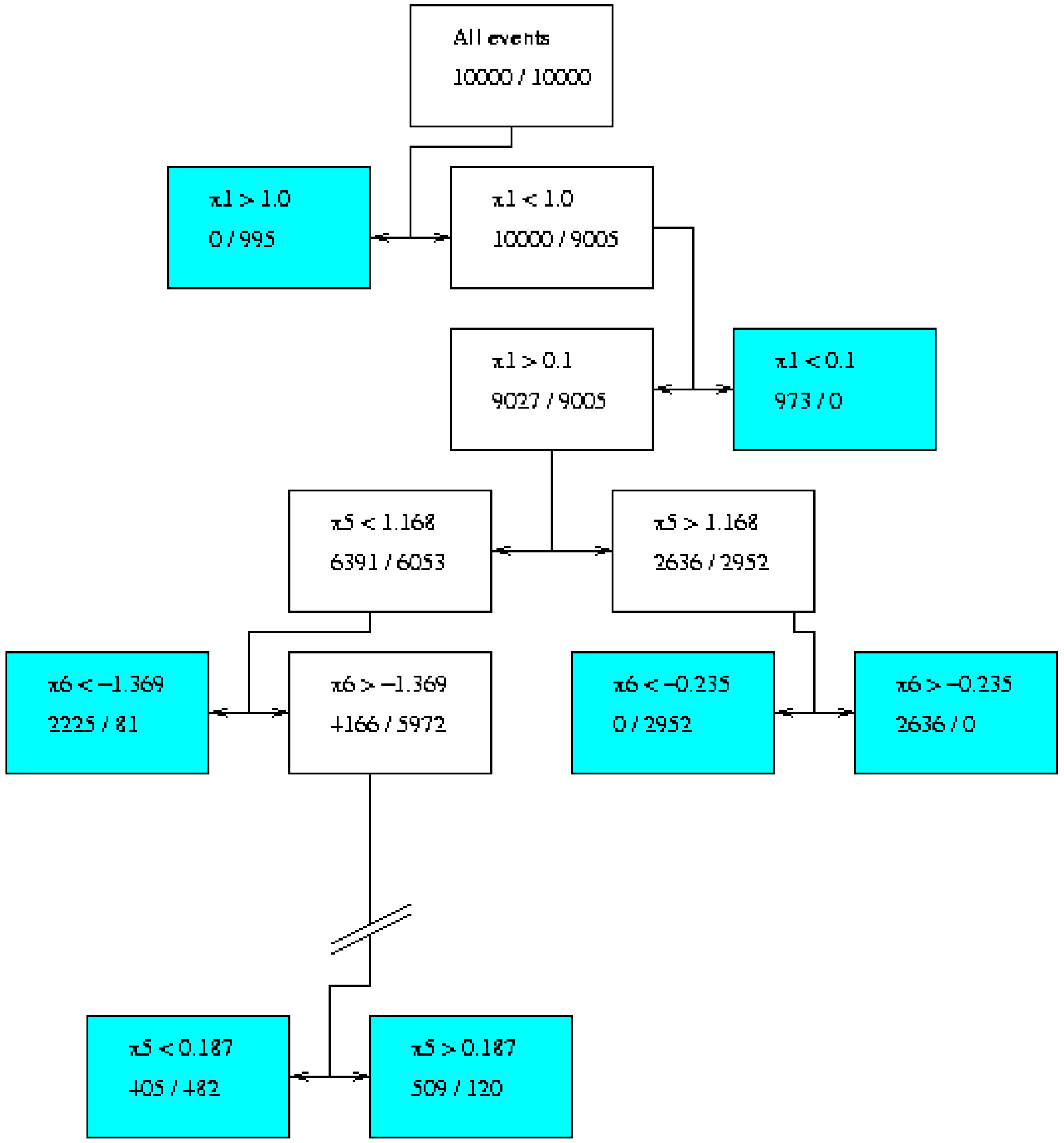


A “decision tree” is a hierarchical series of 1D cuts. Each cut attempts to maximize the signal separation.

Each cut results in a “branch” on the tree. You continue to add more branches until you get perfect separation in a node or the number of events left in a node is small. The end of a branch is a “leaf” (shown in blue).

Each leaf is either mostly signal or mostly background. Classify new events by seeing which leaf they fall in, and whether that leaf is majority signal or majority background.

How do you build a decision tree?



You definitely don't want to develop the tree by hand---hard to optimize and time-consuming!

Fortunately there exist automated procedures for creating decision trees.

How do you build a decision tree?

You start with a training set consisting of equal numbers of signal and background events. The decision tree will train itself on these events.

Take whatever node you want to split. Search through all possible 1D cuts to find the cut that gives the best signal separation as measured by the "information gain" resulting from the cut:

$$\Delta(\text{information}) = \frac{\text{passed}}{\text{total}}\left\{\left(\frac{\text{pass}_A}{\text{pass}_A+\text{pass}_B}\right)\ln\left(\frac{\text{pass}_A}{\text{pass}_A+\text{pass}_B}\right)\right\} + \frac{\text{failed}}{\text{total}}\left\{\left(\frac{\text{fail}_A}{\text{fail}_A+\text{fail}_B}\right)\ln\left(\frac{\text{fail}_A}{\text{fail}_A+\text{fail}_B}\right)\right\}$$

The log terms reflect the information entropy change due to the differing fractions of A events and B events that pass the prospective cut.

After you apply a cut, you get new nodes. If the node is pure signal or pure background, or if the number of events in the node is small, call it a leaf and don't split it further. Continue to try to split all nodes until convergence.

Decision tree results

I used an automated decision tree builder I wrote myself to my canonical example. I trained it on 10,000 signal and 10,000 background events, then applied it to a separate set of 20,000 events.

I could vary how “finely meshed” to make the tree---in other words, what's the smallest node that I will try to subdivide.

I then measure what fraction of signal events and background events are misclassified as a function of the smallest node size.

Split threshold	# of leaves	Fraction Wrong
3000	13	16.9%
2000	15	15.0%
1000	20	9.1%
500	36	5.1%
200	54	4.6%
100	101	4.6%
50	185	4.8%
35	242	5.0%
Theoretical limit		4.2%

Very good performance---however, note that the error rate starts to get worse after a while. Why?

Over-training and over-generalization

Decision trees are a form of *automated machine learning.* The algorithm uses the training data to try and figure out what distinguishes between the two kinds of events. It's really generalizing.

But it can over-generalize. Imagine the following reasoning:
1) All pug dogs I've seen have snub noses, but dalmatians don't.
2) Pug dogs are small and stocky, but dalmatians aren't.
3) Pugs are tan with black coloring, but dalmatians aren't.

Give the program an all-black pug. It will apply the last rule, notice that the animal is not tan with black coloring, and conclude that it must be a dalmatian.

The program over-generalizes! This means that it is tuning cuts to accidental random features in the data rather than truly significant differences.

Lesson: don't over-train any learning system. Don't overthink yourself.

Preventing over-training

How do you know you have over-trained? First of all, pay close attention to how quickly the training converges on test data---once it's started to plateau, don't get greedy and try too hard to squeeze out more performance.

If possible, keep some training data in reserve, and test your trained decision tree on it. Test the error rate as a function of the number of leaves you allow.

Consider pruning your decision tree if it gives evidence of being over-trained.

Read the existing literature---lots written on machine learning!

What does a decision tree really do?

We've already seen that the ideal way to do signal classification is a likelihood ratio using the full-dimensional PDF. But if you don't know the PDF and have to estimate it from the data, you need ridiculous amounts of statistics to make it work.

If you could use very coarse binning in your PDF, you could estimate the PDF's shape with much less data. But of course you will lose information by binning---your bins might hide significant features in the data the provide powerful signal separation.

Decision trees are an automated procedure for dividing your full N-dimensional parameter space into a small number of bins, while optimizing the bin boundaries in N-dimensions so as to isolate those features of the data that provide the most signal separation.

Each bin is a leaf.

An advertisement: alternative approaches

1) Boosted decision trees: a modification of decision trees that uses misclassified events to “retrain” the tree to learn from its mistakes.

2) Artificial neural networks: uses computational model of neuron functioning to build networks that learn much like people. Once trained, tends to look like a “black box”.

3) Non-linear discriminants: generalizations of linear Fisher discriminants

4) Many others!

Multivariate analysis is a Very Complicated Business, and no one approach is suitable to all problems.

See http://tmva.sourceforge.net for some free ROOT-based implementations of most of these algorithms.