Statistical Methods in Particle Physics
Lecture 3: Multivariate Methods

Glen Cowan
Physics Department
Royal Holloway, University of London
g.cowan@rhul.ac.uk
www.pp.rhul.ac.uk/~cowan
Outline

Lecture #1: An introduction to Bayesian statistical methods
Role of probability in data analysis (Frequentist, Bayesian)
A simple fitting problem: Frequentist vs. Bayesian solution
Bayesian computation, Markov Chain Monte Carlo

Lecture #2: Setting limits, making a discovery
Frequentist vs Bayesian approach,
treatment of systematic uncertainties

Lecture #3: Multivariate methods for HEP
Event selection as a statistical test
Neyman-Pearson lemma and likelihood ratio test
Some multivariate classifiers:
   NN, BDT, SVM, ...
Resources on multivariate methods

Books:


Materials from some recent meetings:


Caltech workshop on multivariate analysis, 11 February, 2008 [indico.cern.ch/conferenceDisplay.py?confId=27385](http://indico.cern.ch/conferenceDisplay.py?confId=27385)

Software for multivariate analysis

**TMVA**, Höcker, Stelzer, Tegenfeldt, Voss, Voss, physics/0703039

From [tmva.sourceforge.net](http://tmva.sourceforge.net), also distributed with ROOT

Variety of classifiers

Good manual

**StatPatternRecognition**, I. Narsky, physics/0507143

Further info from [www.hep.caltech.edu/~narsky/spr.html](http://www.hep.caltech.edu/~narsky/spr.html)

Also wide variety of methods, many complementary to **TMVA**

Currently appears project no longer to be supported
A simulated SUSY event in ATLAS

high $p_T$ muons

high $p_T$ jets of hadrons

missing transverse energy
Background events

This event from Standard Model ttbar production also has high $p_T$ jets and muons, and some missing transverse energy.

→ can easily mimic a SUSY event.
A simulated event

PYTHIA Monte Carlo

pp → gluino-gluino
Event selection as a statistical test

For each event we measure a set of numbers: \( \bar{x} = (x_1, \ldots, x_n) \)

\[
x_1 = \text{jet } p_T \\
x_2 = \text{missing energy} \\
x_3 = \text{particle i.d. measure, ...}
\]

\( \bar{x} \) follows some \( n \)-dimensional joint probability density, which depends on the type of event produced, i.e., was it \( pp \rightarrow t\bar{t}, \ pp \rightarrow \tilde{g}\tilde{g}, \ldots \)

E.g. hypotheses \( H_0, H_1, \ldots \)
Often simply “signal”, “background”
Finding an optimal decision boundary

In particle physics usually start by making simple “cuts”:

\[ x_i < c_i \]
\[ x_j < c_j \]

Maybe later try some other type of decision boundary:
Test statistics

The decision boundary is a surface in the $n$-dimensional space of input variables, e.g., $y(\bar{x}) = \text{const}$.

We can treat the $y(x)$ as a scalar test statistic or discriminating function, and try to define this function so that its distribution has the maximum possible separation between the event types:

The decision boundary is now effectively a single cut on $y(x)$, dividing $x$-space into two regions:

- $R_0$ (accept $H_0$)
- $R_1$ (reject $H_0$)
Constructing a test statistic

The Neyman-Pearson lemma states: to obtain the highest background rejection for a given signal efficiency (highest power for a given significance level), choose the acceptance region for signal such that

\[
\frac{p(\bar{x}|s)}{p(\bar{x}|b)} > c
\]

where \(c\) is a constant that determines the signal efficiency.

Equivalently, the optimal discriminating function is given by the likelihood ratio:

\[
y(\bar{x}) = \frac{p(\bar{x}|s)}{p(\bar{x}|b)}
\]

N.B. any monotonic function of this is just as good.
Neyman-Pearson doesn't always help

The problem is that we usually don't have explicit formulae for the pdfs \( p(x|s) \), \( p(x|b) \), so for a given \( x \) we can't evaluate the likelihood ratio.

Instead we have Monte Carlo models for signal and background processes, so we can produce simulated data:

\[
\begin{align*}
\text{generate } & \tilde{x} \sim p(\tilde{x}|s) & \quad \text{generate } & \tilde{x} \sim p(\tilde{x}|b) \\
& \tilde{x}_1, \ldots, \tilde{x}_{N_s} & \quad & \tilde{x}_1, \ldots, \tilde{x}_{N_b}
\end{align*}
\]

“training data”

events of known type

Naive try: enter each \((s,b)\) event into an \( n \)-dimensional histogram, use e.g. \( M \) bins for each of the \( n \) dimensions, total of \( M^n \) cells.

\( n \) is potentially large \( \rightarrow \) prohibitively large number of cells to populate, can't generate enough training data.
Two distinct event selection problems

In some cases, the event types in question are both known to exist.

Example: separation of different particle types (electron vs muon)
Use the selected sample for further study.

In other cases, the null hypothesis $H_0$ means "Standard Model" events,
and the alternative $H_1$ means "events of a type whose existence is
not yet established" (to do so is the goal of the analysis).

Many subtle issues here, mainly related to the heavy burden
of proof required to establish presence of a new phenomenon.

Typically require $p$-value of background-only hypothesis
below $\sim 10^{-7}$ (a 5 sigma effect) to claim discovery of
"New Physics".
Using classifier output for discovery

**Diagram:**

- **Left:**
  - **Signal Distribution:** $f(y)$
  - **Background Distribution:**
  - Normalized to unity
  - **Search Region:** $y$

- **Right:**
  - **Signal Distribution:** $N(y)$
  - **Background Distribution:**
  - Normalized to expected number of events
  - **Search Region:** $y_{cut}$
  - **Excess Question:** $y$

**Text:**

Discovery = number of events found in search region incompatible with background-only hypothesis.

$p$-value of background-only hypothesis can depend crucially distribution $f(y|b)$ in the "search region".

G. Cowan

SUSSP65, St Andrews, 16-29 August 2009 / Statistical Methods 3
Some “standard” multivariate methods

Place cuts on individual variables
Simple, intuitive, in general not optimal

Linear discriminant (e.g. Fisher)
Simple, optimal if the event types are Gaussian distributed with equal covariance, otherwise not optimal.

Probability Density Estimation based methods
Try to estimate $p(x|s), p(x|b)$ then use $y(\mathbf{x}) = \hat{p}(x|s)/\hat{p}(x|b)$.
In principle best, difficult to estimate $p(x)$ for high dimension.

Neural networks
Can produce arbitrary decision boundary (in principle optimal), but can be difficult to train, result non-intuitive.
Example of a "cut-based" study

In the 1990s, the CDF experiment at Fermilab (Chicago) measured the number of hadron jets produced in proton-antiproton collisions as a function of their momentum perpendicular to the beam direction:

Prediction low relative to data for very high transverse momentum.
High $p_T$ jets = quark substructure?

Although the data agree remarkably well with the Standard Model (QCD) prediction overall, the excess at high $p_T$ appears significant:

The fact that the variable is "understandable" leads directly to a plausible explanation for the discrepancy, namely, that quarks could possess an internal substructure.

Would not have been the case if the variable plotted was a complicated combination of many inputs.
High $p_T$ jets from parton model uncertainty

Furthermore the physical understanding of the variable led one to a more plausible explanation, namely, an uncertain modeling of the quark (and gluon) momentum distributions inside the proton.

When model adjusted, discrepancy largely disappears:

Can be regarded as a "success" of the cut-based approach. Physical understanding of output variable led to solution of apparent discrepancy.
Linear decision boundaries

A linear decision boundary is only optimal when both classes follow multivariate Gaussians with equal covariances and different means.

For some other cases a linear boundary is almost useless.
Nonlinear transformation of inputs

We can try to find a transformation, \( x_1, \ldots, x_n \rightarrow \varphi_1(\vec{x}), \ldots, \varphi_m(\vec{x}) \) so that the transformed “feature space” variables can be separated better by a linear boundary:

\[
\varphi_1 = \tan^{-1}(x_2/x_1)
\]

\[
\varphi_2 = \sqrt{x_1^2 + x_2^2}
\]

Here, guess fixed basis functions (no free parameters)
Neural networks in particle physics

For many years, the only "advanced" classifier used in particle physics.

Usually use single hidden layer, logistic sigmoid activation function:

\[ h_i(\vec{x}) = s \left( w_{i0} + \sum_{j=1}^{n} w_{ij} x_j \right), \]

\[ t(\vec{x}) = s \left( a_0 + \sum_{i=1}^{n} a_i h_i(\vec{x}) \right). \]

\[ s(u) \equiv (1 - e^{-u})^{-1}. \]
Neural network example from LEP II

Signal: \( e^+e^- \rightarrow W^+W^- \) (often 4 well separated hadron jets)

Background: \( e^+e^- \rightarrow qqgg \) (4 less well separated hadron jets)

← input variables based on jet structure, event shape, ...
none by itself gives much separation.

Neural network output:

(Garrido, Juste and Martinez, ALEPH 96-144)
Some issues with neural networks

In the example with WW events, goal was to select these events so as to study properties of the W boson.

Needed to avoid using input variables correlated to the properties we eventually wanted to study (not trivial).

In principle a single hidden layer with an sufficiently large number of nodes can approximate arbitrarily well the optimal test variable (likelihood ratio).

Usually start with relatively small number of nodes and increase until misclassification rate on validation data sample ceases to decrease.

Often MC training data is cheap -- problems with getting stuck in local minima, overtraining, etc., less important than concerns of systematic differences between the training data and Nature, and concerns about the ease of interpretation of the output.
Overtraining

If decision boundary is too flexible it will conform too closely to the training points → overtraining.

Monitor by applying classifier to independent test sample.
Particle i.d. in MiniBooNE

Detector is a 12-m diameter tank of mineral oil exposed to a beam of neutrinos and viewed by 1520 photomultiplier tubes:

Search for $\nu_\mu$ to $\nu_e$ oscillations required particle i.d. using information from the PMTs.

H.J. Yang, MiniBooNE PID, DNP06
Decision trees

Out of all the input variables, find the one for which with a single cut gives best improvement in signal purity:

\[
P = \frac{\sum_{\text{signal}} w_i}{\sum_{\text{signal}} w_i + \sum_{\text{background}} w_i}
\]

where \( w_i \) is the weight of the \( i \)th event.

Resulting nodes classified as either signal/background.

Iterate until stop criterion reached based on e.g. purity or minimum number of events in a node.

The set of cuts defines the decision boundary.

Example by MiniBooNE experiment, B. Roe et al., NIM 543 (2005) 577
Decision trees (2)

The terminal nodes (leaves) are classified as signal or background depending on majority vote (or e.g. signal fraction greater than a specified threshold).

This classifies every point in input-variable space as either signal or background, a decision tree classifier, with the discriminant function

\[ f(x) = 1 \text{ if } x \in \text{signal region}, -1 \text{ otherwise} \]

Decision trees tend to be very sensitive to statistical fluctuations in the training sample.

Methods such as boosting can be used to stabilize the tree.
Boosting

Boosting is a general method of creating a set of classifiers which can be combined to achieve a new classifier that is more stable and has a smaller error than any individual one.

Often applied to decision trees but, can be applied to any classifier.

Suppose we have a training sample $T$ consisting of $N$ events with

- $x_1, \ldots, x_N$ event data vectors (each $x$ multivariate)
- $y_1, \ldots, y_N$ true class labels, +1 for signal, -1 for background
- $w_1, \ldots, w_N$ event weights

Now define a rule to create from this an ensemble of training samples $T_1, T_2, \ldots$, derive a classifier from each and average them.
AdaBoost

A successful boosting algorithm is AdaBoost (Freund & Schapire, 1997).

First initialize the training sample $T_1$ using the original

- $x_1, \ldots, x_N$ event data vectors
- $y_1, \ldots, y_N$ true class labels (+1 or -1)
- $w^{(1)}_1, \ldots, w^{(1)}_N$ event weights

with the weights equal and normalized such that $\sum_{i=1}^{N} w^{(1)}_i = 1$.

Train the classifier $f_1(x)$ (e.g. a decision tree) using the weights $w^{(1)}$ so as to minimize the classification error rate,

$$\varepsilon_1 = \sum_{i=1}^{N} w^{(1)}_i I(y_i f_1(x_i) \leqslant 0),$$

where $I(X) = 1$ if $X$ is true and is zero otherwise.
Updating the event weights (AdaBoost)

Assign a score to the $k$th classifier based on its error rate:

$$\alpha_k = \ln \frac{1 - \varepsilon_k}{\varepsilon_k}$$

Define the training sample for step $k+1$ from that of $k$ by updating the event weights according to

$$w_{i}^{(k+1)} = w_{i}^{(k)} e^{-\alpha_k f_k(x_i) y_i / 2}$$

Normalize so that $\sum_i w_i^{(k+1)} = 1$

$$i = \text{event index} \quad k = \text{training sample index}$$

Iterate $K$ times, final classifier is

$$y(x) = \sum_{k=1}^{K} \alpha_k f_k(x, T_k)$$
BDT example from MiniBooNE

~200 input variables for each event ($\nu$ interaction producing e, $\mu$ or $\pi$).

Each individual tree is relatively weak, with a misclassification error rate $\sim 0.4 - 0.45$

B. Roe et al., NIM 543 (2005) 577
Monitoring overtraining

From MiniBooNE example:
Performance stable after a few hundred trees.
Comparison of boosting algorithms

A number of boosting algorithms on the market; differ in the update rule for the weights.
Boosted decision tree summary

Advantage of boosted decision tree is it can handle a large number of inputs. Those that provide little/no separation are rarely used as tree splitters are effectively ignored.

Easy to deal with inputs of mixed types (real, integer, categorical...).

If a tree has only a few leaves it is easy to visualize (but rarely use only a single tree).

There are a number of boosting algorithms, which differ primarily in the rule for updating the weights (ε-Boost, LogitBoost,...)

Other ways of combining weaker classifiers: Bagging (Bootstrap-Aggregating), generates the ensemble of classifiers by random sampling with replacement from the full training sample.
Single top quark production (CDF/D0)

Top quark discovered in pairs, but SM predicts single top production.

Use many inputs based on jet properties, particle i.d., ...

Pair-produced tops are now a background process.

signal (blue + green)
Different classifiers for single top

Also Naive Bayes and various approximations to likelihood ratio,....

Final combined result is statistically significant (>5σ level) but not easy to understand classifier outputs.
Support Vector Machines

Map input variables into high dimensional feature space: $x \rightarrow \phi$

Maximize distance between separating hyperplanes (margin) subject to constraints allowing for some misclassification.

Final classifier only depends on scalar products of $\phi(x)$:

$$y(x) = \text{sign} \left( \sum_i \alpha_i y_i \vec{\phi}(x) \cdot \vec{\phi}(x_i) + b \right)$$

So only need kernel

$$K(x, x') = \vec{\phi}(x) \cdot \vec{\phi}(x')$$
Support Vector Machines

Support Vector Machines (SVMs) are an example of a kernel-based classifier, which exploits a nonlinear mapping of the input variables onto a higher dimensional feature space.

The SVM finds a linear decision boundary in the higher dimensional space. But thanks to the “kernel trick” one does not every have to write down explicitly the feature space transformation.

Some references for kernel methods and SVMs:

- The books mentioned in [www.pp.rhul.ac.uk/~cowan/mainz_lectures.html](http://www.pp.rhul.ac.uk/~cowan/mainz_lectures.html)
- The TMVA manual (!)
Linear SVMs

Consider a training data set consisting of

\[ x_1, \ldots, x_N \quad \text{event data vectors} \]

\[ y_1, \ldots, y_N \quad \text{true class labels (+1 or -1)} \]

Suppose the classes can be separated by a hyperplane defined by a normal vector \( w \) and scalar offset \( b \) (the "bias"). We have

\[ x_i \cdot w + b \geq +1 \quad \text{for all } y_i = +1 \]

\[ x_i \cdot w + b \leq -1 \quad \text{for all } y_i = -1 \]

or equivalently

\[ y_i(x_i \cdot w + b) - 1 \geq 0 \quad \text{for all } i \]

Bishop Ch. 7
Margin and support vectors

The distance between the hyperplanes defined by $y(x) = x \cdot w + b = +1$ and $y(x) = -1$ is called the margin, which is:

$$\text{margin} = \frac{2}{||w||}$$

If the training data are perfectly separated then this means there are no points inside the margin.

Suppose there are points on the margin (this is equivalent to defining the scale of $w$). These points are called support vectors.
Linear SVM classifier

We can define the classifier using

\[ f(x) = \text{sign}(x \cdot w + b) \]

which is +1 for points on one side of the hyperplane and −1 on the other.

The best classifier should have a large margin, so to maximize

\[ \text{margin} = \frac{2}{||w||} \]

we can minimize \( ||w||^2 \) subject to the constraints

\[ y_i(x_i \cdot w + b) - 1 \geq 0 \quad \text{for all } i \]
Lagrangian formulation

This constrained minimization problem can be reformulated using a Lagrangian

\[ L = \frac{1}{2} \| w \|^2 - \sum_{i=1}^{N} \alpha_i (y_i (x_i \cdot w + b) - 1) \]

positive Lagrange multipliers \( \alpha_i \)

We need to minimize \( L \) with respect to \( w \) and \( b \) and maximize with respect to \( \alpha_i \).

There is an \( \alpha_i \) for every training point. Those that lie on the margin (the support vectors) have \( \alpha_i > 0 \), all others have \( \alpha_i = 0 \). The solution can be written

\[ w = \sum_{i} \alpha_i y_i x_i \]  

(sum only contains support vectors)
Dual formulation

The classifier function is thus

\[ f(x) = \text{sign}(x \cdot w + b) = \text{sign} \left( \sum_i \alpha_i y_i x \cdot x_i + b \right) \]

It can be shown that one finds the same solution \( a \) by minimizing the dual Lagrangian

\[ L_D = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i \cdot x_j \]

So this means that both the classifier function and the Lagrangian only involve dot products of vectors in the input variable space.
Nonseparable data

If the training data points cannot be separated by a hyperplane, one can redefine the constraints by adding slack variables $\xi_i$:

$$y_i(x_i \cdot w + b) + \xi_i - 1 \geq 0 \text{ with } \xi_i \geq 0 \text{ for all } i$$

Thus the training point $x_i$ is allowed to be up to a distance $\xi_i$ on the wrong side of the margin, and $\xi_i = 0$ at or on the right side.

For an error to occur we have $\xi_i > 1$, so

$$\sum_i \xi_i$$

is an upper bound on the number of training errors.
Cost function for nonseparable case

To limit the magnitudes of the $\xi_i$ we can define the error function that we minimize to determine $w$ to be

$$E(w) = \frac{1}{2} \|w\|^2 + C \left( \sum_i \xi_i \right)^k$$

where $C$ is a cost parameter we must choose that limits the amount of misclassification. It turns out that for $k=1$ or 2 this is a quadratic programming problem and furthermore for $k=1$ it corresponds to minimizing the same dual Lagrangian

$$L_D = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j$$

where the constraints on the $\alpha_i$ become $0 \leq \alpha_i \leq C$. 
Nonlinear SVM

So far we have only reformulated a way to determine a linear classifier, which we know is useful only in limited circumstances.

But the important extension to nonlinear classifiers comes from first transforming the input variables to feature space:

\[ \tilde{\phi}(x) = (\varphi_1(x), \ldots, \varphi_m(x)) \]

These will behave just as our new “input variables”. Everything about the mathematical formulation of the SVM will look the same as before except with \( \phi(x) \) appearing in the place of \( x \).
Only dot products

Recall the SVM problem was formulated entirely in terms of dot products of the input variables, e.g., the classifier is

\[ f(x) = \text{sign} \left( \sum_i \alpha_i y_i x \cdot x_i + b \right) \]

so in the feature space this becomes

\[ f(x) = \text{sign} \left( \sum_i \alpha_i y_i \tilde{\phi}(x) \cdot \tilde{\phi}(x_i) + b \right) \]
The Kernel trick

How do the dot products help? It turns on that a broad class of kernel functions can be written in the form:

$$K(x, x') = \phi(x) \cdot \phi(x')$$

Functions having this property must satisfy Mercer's condition

$$\int K(x, x') g(x) g(x') \, dx \, dx' \geq 0$$

for any function $g$ where $\int g^2(x) \, dx$ is finite.

So we don't even need to find explicitly the feature space transformation $\phi(x)$, we only need a kernel.
Finding kernels

There are a number of techniques for finding kernels, e.g., constructing new ones from known ones according to certain rules (cf. Bishop Ch 6).

Frequently used kernels to construct classifiers are e.g.

\[ K(x, x') = (x \cdot x' + \theta)^p \]  

polynomial

\[ K(x, x') = \exp \left( \frac{-\|x - x'\|^2}{2\sigma^2} \right) \]  

Gaussian

\[ K(x, x') = \tanh(\kappa(x \cdot x') + \theta) \]  

sigmoidal
Using an SVM

To use an SVM the user must as a minimum choose

- a kernel function (e.g. Gaussian)
- any free parameters in the kernel (e.g. the $\sigma$ of the Gaussian)
- a cost parameter $C$ (plays role of regularization parameter)

The training is relatively straightforward because, in contrast to neural networks, the function to be minimized has a single global minimum.

Furthermore evaluating the classifier only requires that one retain and sum over the support vectors, a relatively small number of points.

The advantages/disadvantages and rationale behind the choices above is not always clear to the particle physicist -- help needed here.
SVM in particle physics

SVMs are very popular in the Machine Learning community but have yet to find wide application in HEP. Here is an early example from a CDF top quark analysis (A. Vaiciulis, contribution to PHYSTAT02).
Summary on multivariate methods

Particle physics has used several multivariate methods for many years:
  linear (Fisher) discriminant
  neural networks
  naive Bayes

and has in the last several years started to use a few more
  $k$-nearest neighbour
  boosted decision trees
  support vector machines

The emphasis is often on controlling systematic uncertainties between
the modeled training data and Nature to avoid false discovery.

Although many classifier outputs are "black boxes", a discovery
at $5\sigma$ significance with a sophisticated (opaque) method will win the
competition if backed up by, say, $4\sigma$ evidence from a cut-based method.
Extra slides
Decision boundary flexibility

The decision boundary will be defined by some free parameters that we adjust using training data (of known type) to achieve the best separation between the event types.

Goal is to determine the boundary using a finite amount of training data so as to best separate between the event types for an unseen data sample.