Computing Methods for Particle Physics

Classification Problems

Overview

- Today: **Classification Problems!**

- Classification methods (supervised machine learning)
  - Decision Trees
  - Naive Bayes Classifier
- Simple example
- Large feature set example with experimental simulation dataset

Classification Problems

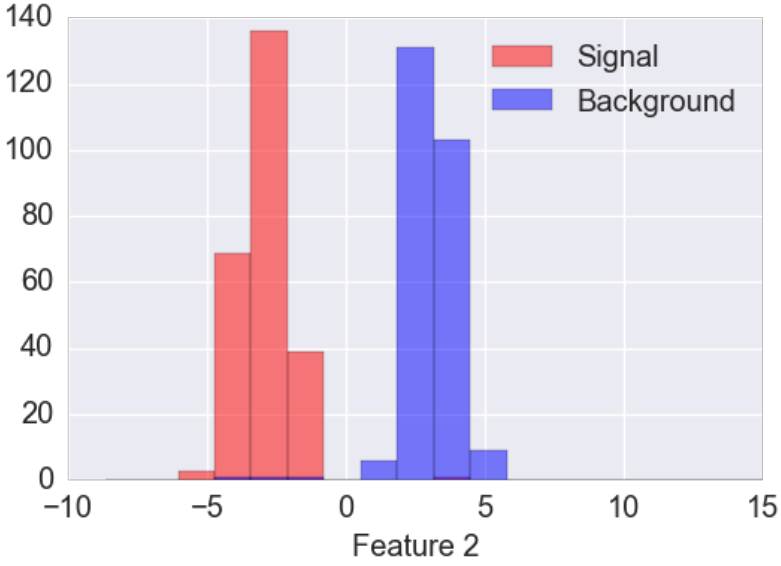
In classification problems we want to use an algorithm that can categorise observations based on a multi-dimensional feature set.

Example: In particle physics we often have feature sets that include reconstructed variables such as: E, p, track/jet/shower properties, etc. We want to be able to categorise each of these events into signal or background categories.

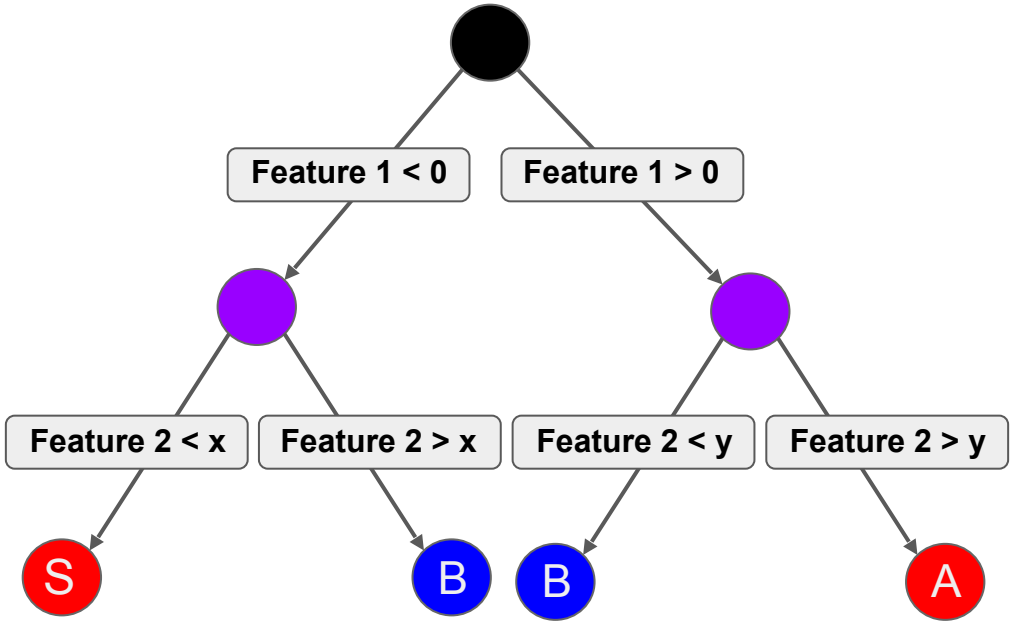
Classification Problems: Boundaries

Maybe it's perfectly clear where to draw a line in a feature of our data to separate signal from background.

Feature 2 > 0? **Background**
Feature 2 < 0? **Signal**



Decision Trees



A decision tree represents a branching **chain of decisions** made based on feature set to classify an observation. This forms a set of hypercubes (squares with two features) in the feature space to distinguish between the classes.

Aside on common interfaces

We will use multiple ML packages:

scikit learn (python based)

TMVA (ROOT based)

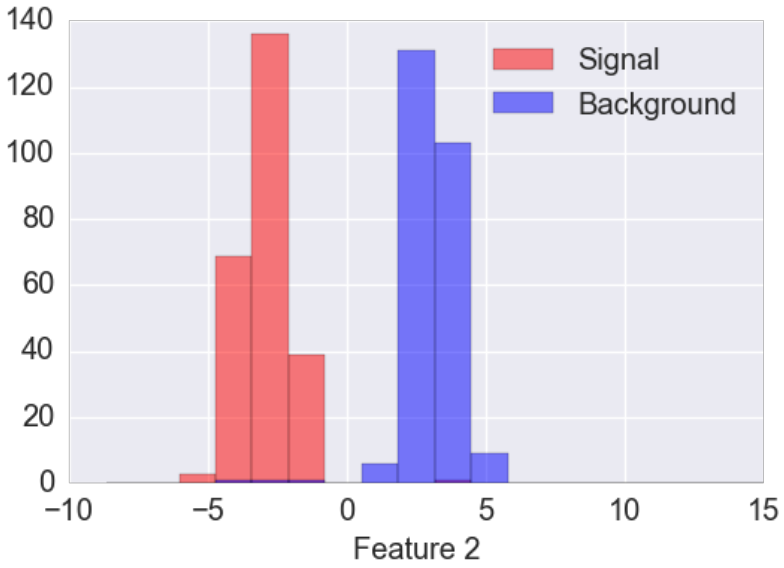
These packages both provide (separate) **common interfaces** to their ML methods. Once you implement one, it's straightforward to evaluate many.

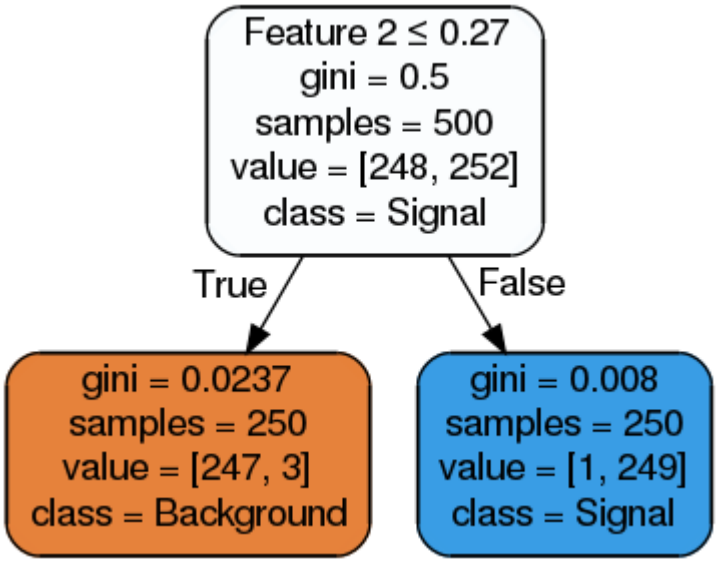
Common approach is to just get **one** algorithm working, then evaluate as **many** as you think are interesting or necessary.

Simple Decision Tree

Example of a simple decision tree.

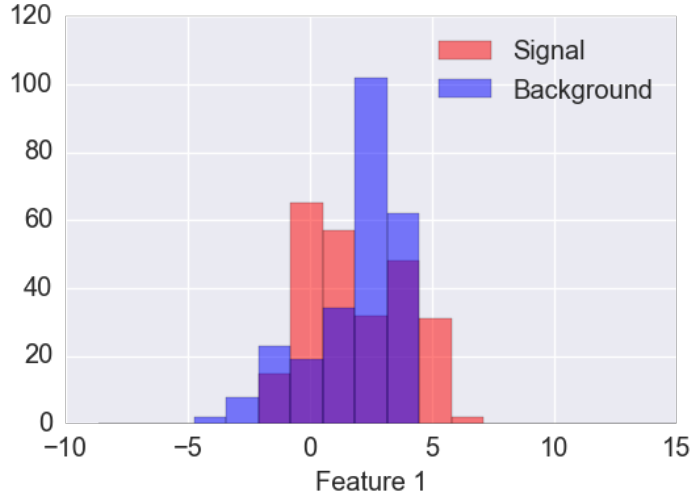
Using our simple dataset:

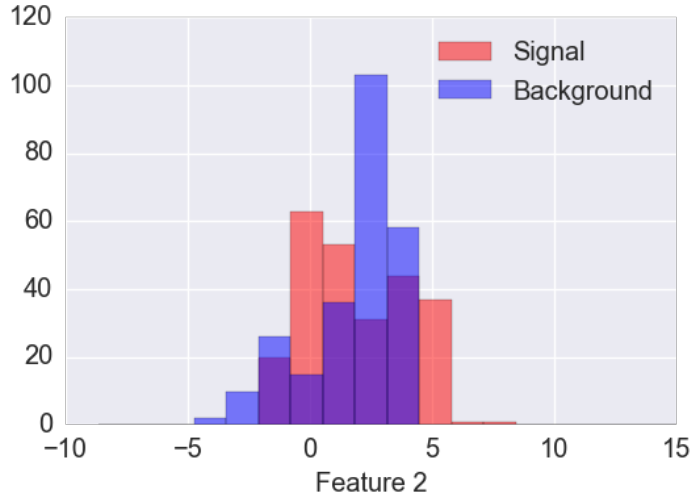




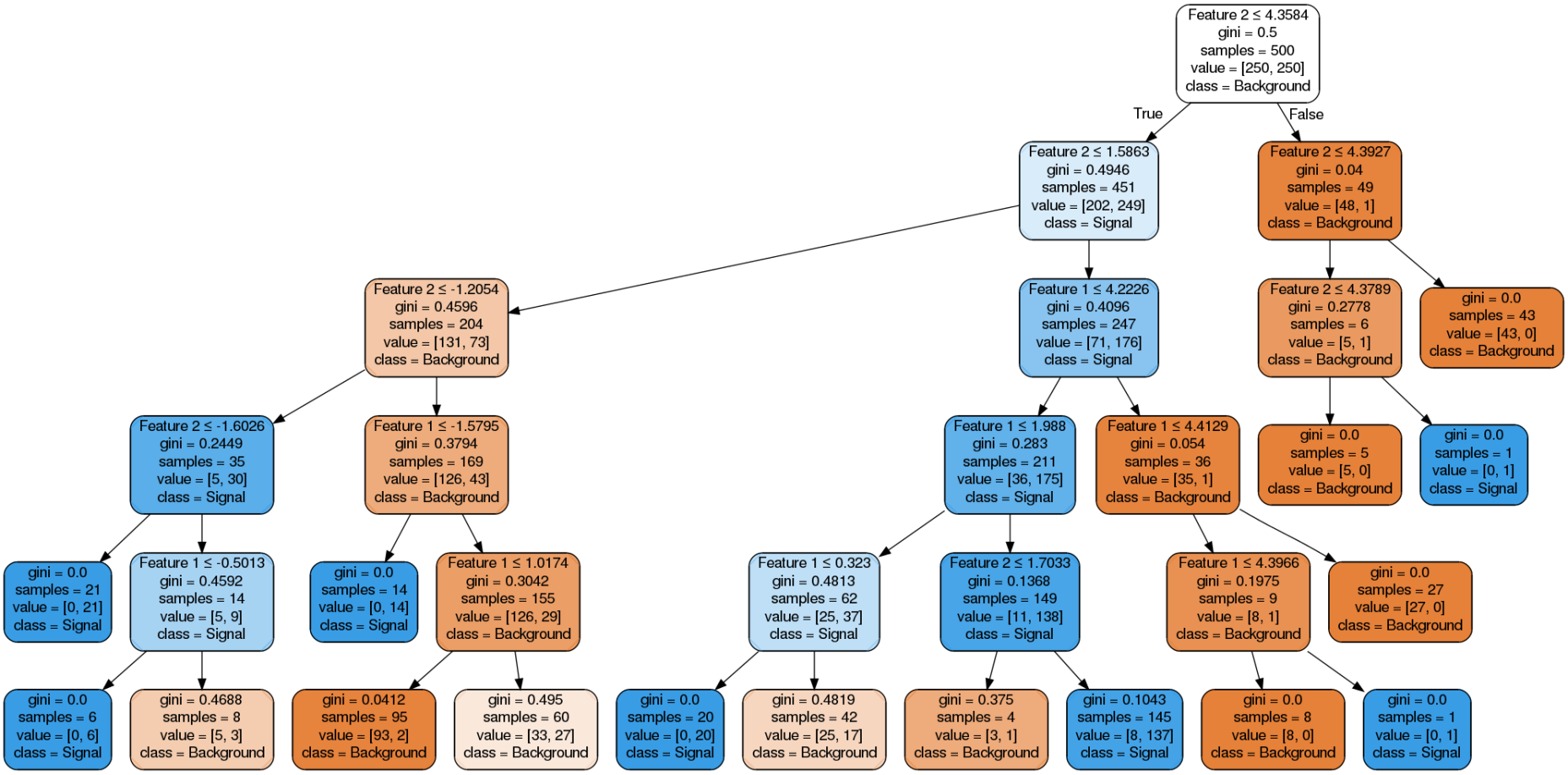
Classification Problems

What do we do if we have unclear boundaries between signal and background in the two features?





A more complicated tree



Boosted Decision Trees

We can train multiple trees using weighted versions of the same dataset to form a **forest** that is more robust, with respect to statistical fluctuations, than an individual tree.

A **Boosted Decision Tree** (BDT) uses a forest of DTs to make a classification.

The **forest** is more stable than individual trees because they are able to see fluctuations in the data samples and thus avoid **overtraining** on the training dataset.

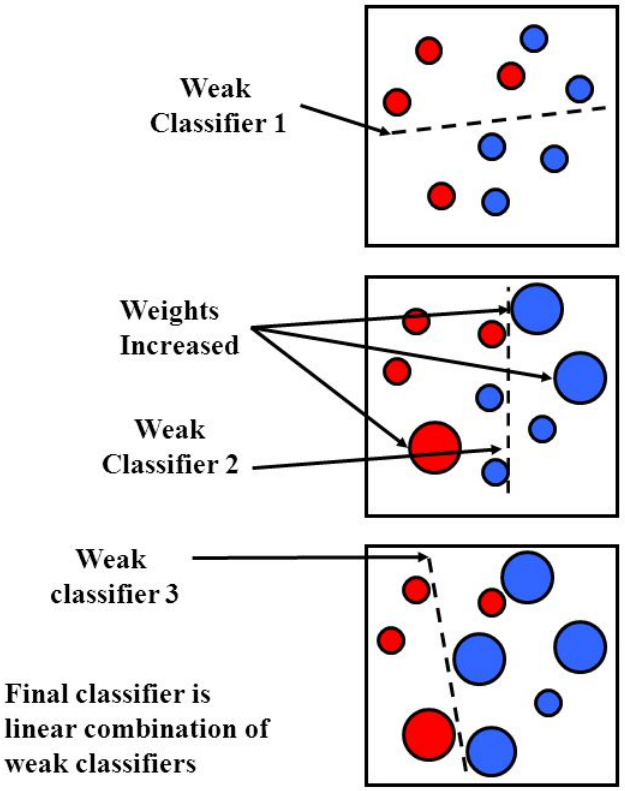
BDT Example

We will revisit the previous example with a version of a BDT that is formed by a meta-classifier algorithm known as **AdaBoost**. This is an iterative approach that repeatedly trains an ensemble of **weak classifiers** while emphasizing cases where the classification failed in previous sets. The final classification of an event is decided by a weighted average across the ensemble.

Note: AdaBoost can be applied to any classifier (not just decision trees!).

This is known to perform better than a single classifier because it is able to use the forest of classifiers to better account for the boundary cases. We will see a complex classification topology for our example.

The example is here.



http://www.vision.rwth-aachen.de/course/8/

Naive Bayes

We can’t just look at one classifier, what would we compare it too?!

Another common choice, related to the previous lecture, is the **Naive Bayes Classifier**. This uses the Bayes theorem to make a classification decision under the assumption that all the features are independent (hence, “naive”). It reduces the problem of finding an n-dimensional pdf to one of finding n, 1-d pdfs.

It’s a very simple algorithm, that is fast because of its naive assumptions. Especially optimal at high-dimension datasets.

Back to our dataset...

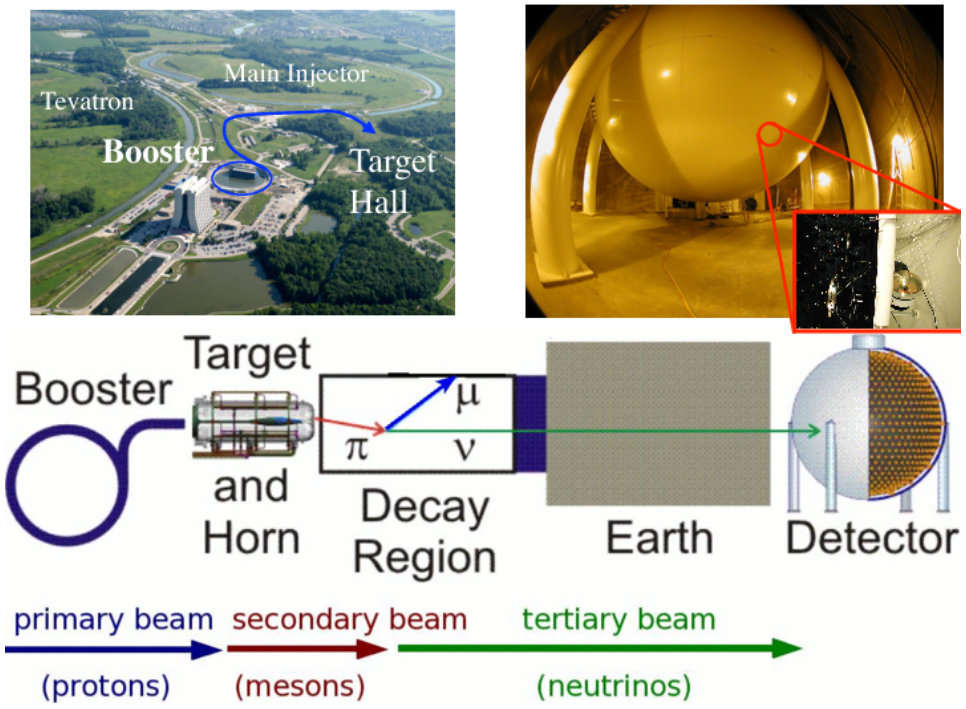
In the first lecture we looked at a dataset from the **MiniBooNE** experiment.

Let's look at this dataset again, but with **multiple** classification methods in mind.

- Decision Tree
- Boosted Decision Tree
- Naive Bayes (aka Likelihood)

Reminder: you can find the dataset here.

MiniBooNE Overview



Key Machine Learning Concepts

Data Cleaning
Feature Normalisation
Training & Testing Samples
ROC Curves

Data Cleaning

Data is rarely perfect, even if it is simulated! Common problems with data include:

- Column headers interpreted as rows
- Data not categorised properly (implicit classification labels)

Approaches to **data cleaning**:

- Substitution: replace bad features with sensible values
  - *Example: you know that the simulation returns -3 for a feature if the output of an algorithm was undefined.*
- Deleting observations

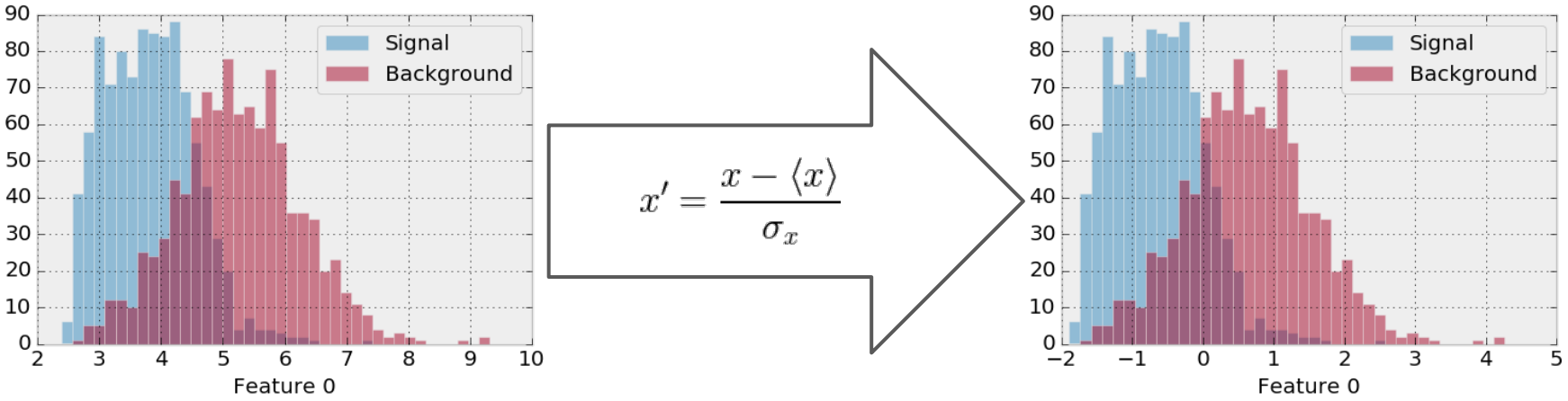
In this MiniBooNE dataset:

- No classification **labels**; we add them based on data ordering
- No **headers**; that is ok for our needs, but usually want to know feature names
- Some entries have -999.0; a typical, **default** value; delete them

Feature Standardisation & Normalisation

Many ML algorithms rely on the distance between features as a benchmark when, for example, computing gradients. If two features naturally have different scales this can lead to different gradients and effect the relative importance of different features.

- Many ML algorithms perform better with **normalised** or **standardised** features.
- **Decision trees** are not sensitive to normalisations since they can learn the scale
  - BUT if we want to compare to an algorithm that is sensitive it may be useful!
- **Standardisation**: <x>=0, σ=1
- **Normalisation**: min(x)=-1 or 0, max(x)=1



Training & Testing Samples

It is important to split simulated datasets into training and testing samples:

Training: used to train your chosen ML method

Testing: used to test the performance of your chosen ML method

Why not just use all the data for training?

This would bias your assessment of the ML results.

The method needs to be evaluated on data it has not seen before.

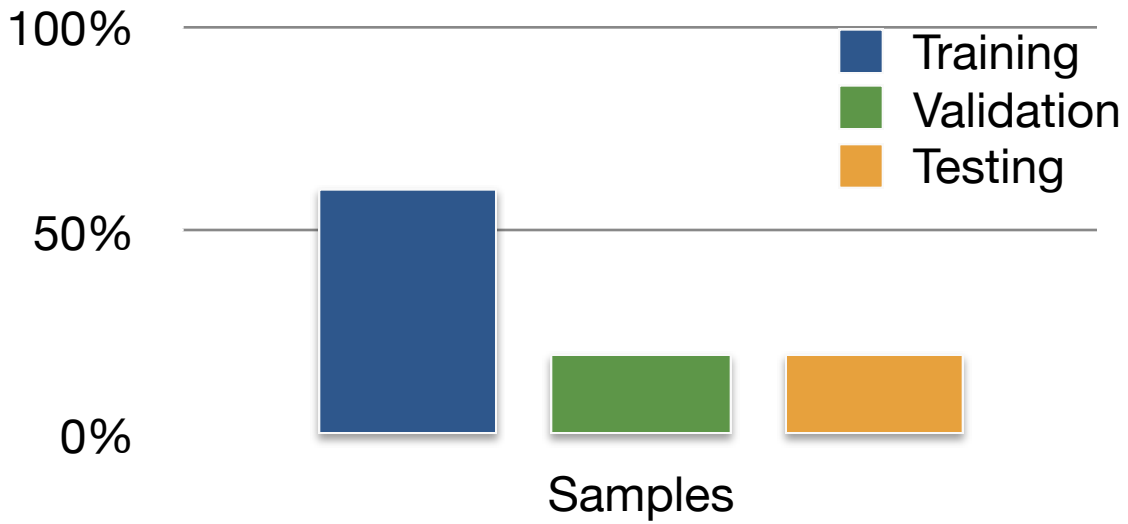
Overtraining: your ML method has learned your training sample **too** well!
Worse performance on your testing sample than on the training sample.

How do we test the **performance**?

Note: the training set is often further divided into **validation** and a **test** sample where the validation sample is used to tune the ML method's parameters and the test is used to evaluate the final performance of the method. This avoids biasing your measurement of the performance of the classifier.

Training, Validation, & Testing Samples

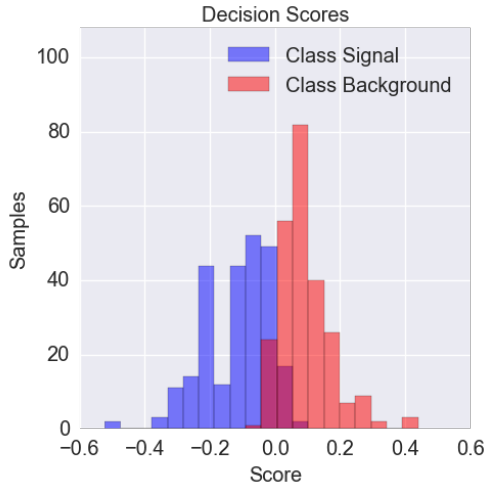
A common split is:



Steps:

1. **Train** your algorithms using the training dataset. Multiple algorithms and/or multiple versions of algorithms with different tunings.
2. **Validate** that your algorithms are performing as expected, decide on an algorithm, or possible repeat the training stage with updated parameters or methods.
3. **Test** your chosen algorithms performance on a dataset that both your chosen algorithm and you yourself have not seen before.

ROC Curves



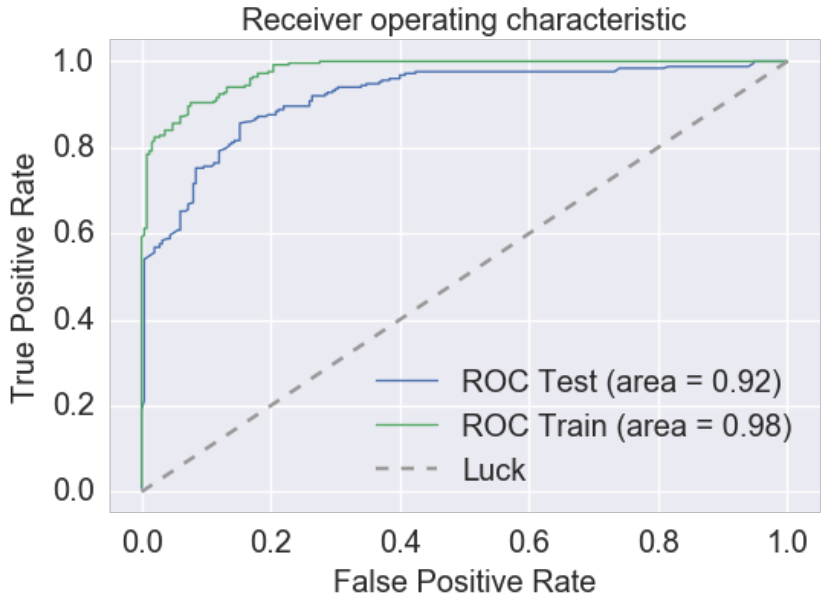
Receiver-Operator Characteristic (ROC) curves grade the performance of a classifier.

Examples:

0/100 false positives; 19/100 true positives
→ 19% signal efficiency, 100% signal purity
10/100 false positives; 76/100 true positives
→ 76% signal efficiency, 90% signal purity

More **area** under the curve is better.
Diagonal is random guessing (luck).

Note: There are different ways of formatting these axes, so be sure to read the axes before interpreting ROC curves.



MiniBooNE Classification Example

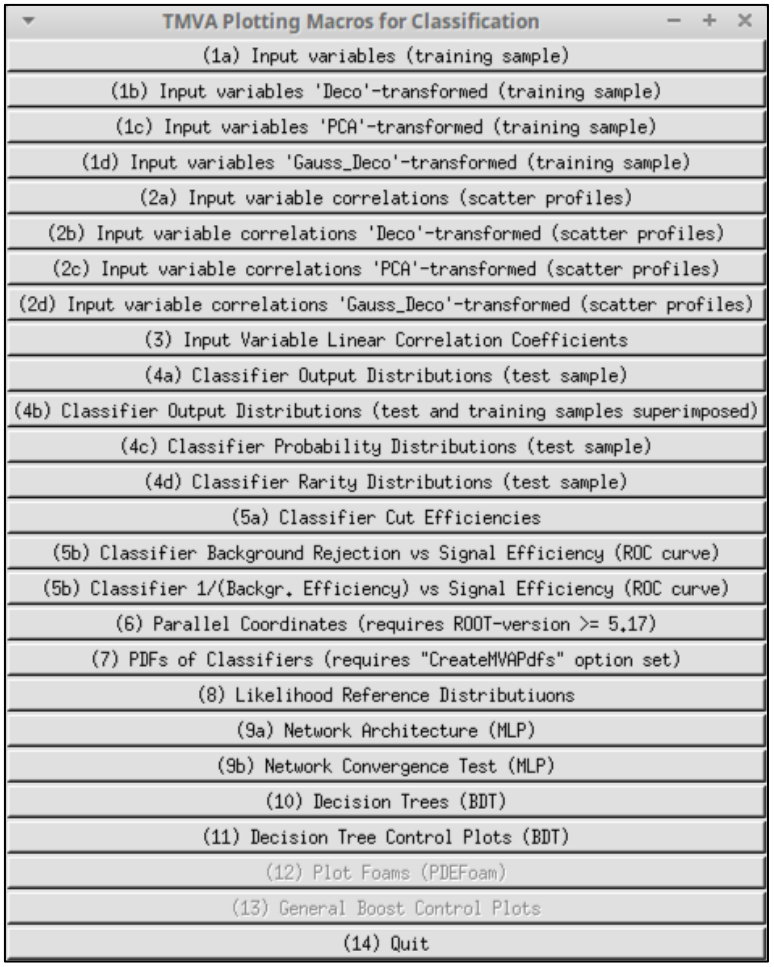
A scikit-learn example, using all 50 MiniBooNE features, is here.

Multivariate Analyses in ROOT

ROOT has an extensive toolkit for performing multivariate analyses, the **TMVA**.

You can define your dataset, classes, testing/ training samples and then have access to many classification algorithms. It performs training and testing and even has a **GUI** for analysing the results.

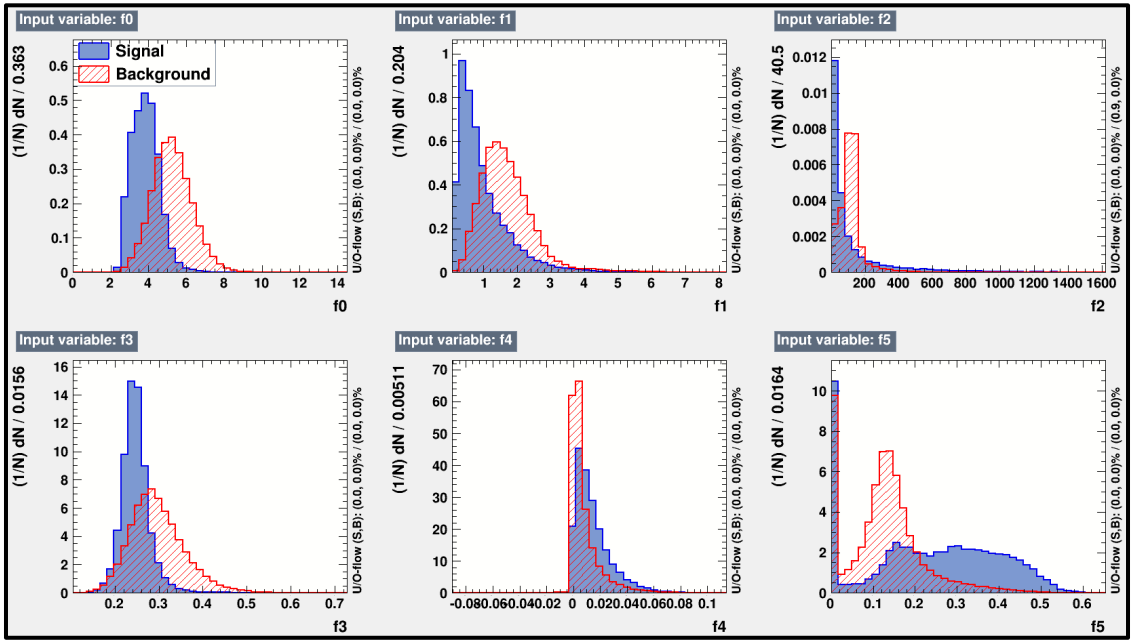
There are very useful TMVA **example scripts** that you can modify to train the full range of MVA methods. These are located in $ROOTSYS/tmva/ test if you have root installed and setup in your environment. See PP cluster machines.



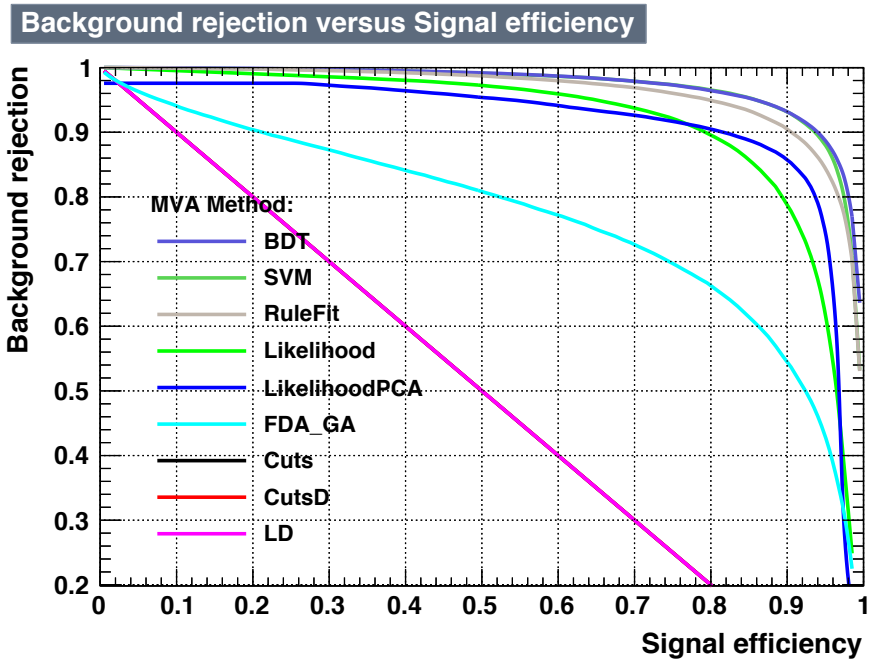
TMVA Example

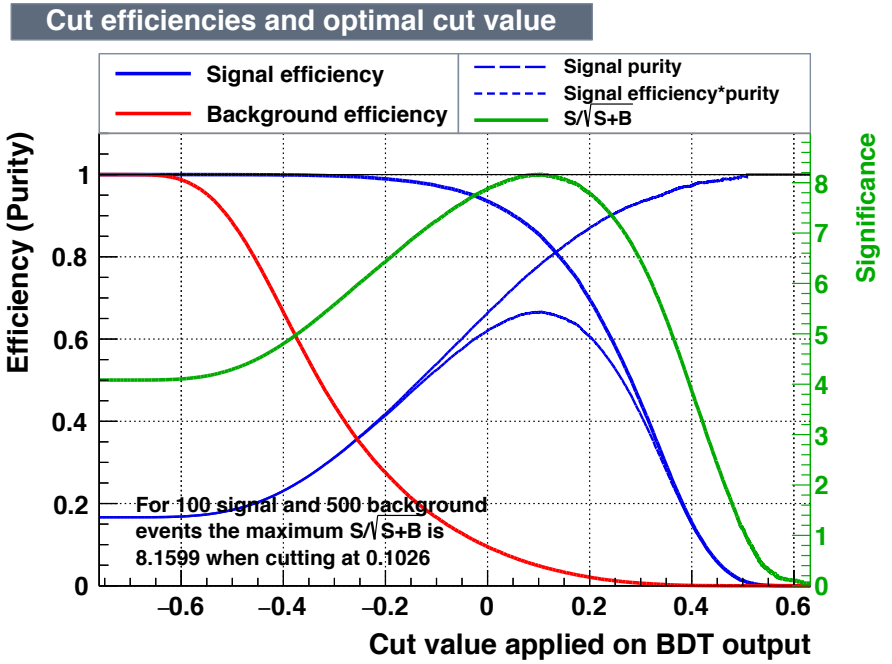
I've modified the TMVAClassification.C example to work with the MiniBooNE dataset. We won't go through the details of the ROOT macro, but you are encouraged to work through it to become familiar with the TMVA framework.

Refer to the TMVA Handbook for more details. I particularly recommend "Chapter 11: Which MVA method should I use for my problem?"



TMVA Example





Summary

We discussed two methods for classifying data.

Decision Trees & Boosted Decision Trees

Naive Bayes

We saw simple and complex **examples** of each method.

We saw a shotgun approach using **TMVA** in ROOT.

Next: extend these ML methods to include **neural networks**.

Suggested Exercises & Further Reading

Exercises:

1. **Different dataset**: Review the TMVA Classification example and pick a different dataset (maybe you have one in mind) to become familiar with the mechanics of TMVA.
2. **Different algorithm**: Pick another ML classification algorithm, research it, and implement it in either Python or using TMVA.
    - Once you understand one algorithm other cases are easier to understand.

Further reading:

- G. Cowan lecture on multivariate methods
- Scikit learn user's guide
- TMVA user's guide

Naive Bayes - Formalism

Bayes:

$$P(\text{Class} \mid x_1, \ldots, x_n) = \frac{P(\text{Class})P(x_1, \ldots x_n \mid \text{Class})}{P(x_1, \ldots, x_n)}$$

Naive independence assumption:

$$P(x_i|\text{Class}, x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) = P(x_i|\text{Class})$$

Allows us to turn conditional probability into product of independent probabilities

$$P(\text{Class} \mid x_1, \ldots, x_n) = \frac{P(\text{Class}) \prod_{i=1}^{n} P(x_i \mid \text{Class})}{P(x_1, \ldots, x_n)}$$

$$P(\text{Class} \mid x_1, \ldots, x_n) \propto P(\text{Class}) \prod_{i=1}^{n} P(x_i \mid \text{Class})$$

We can use a Gaussian likelihood to form the Gaussian Naive Bayes Classifier:

$$P(x_i \mid \text{Class}) = \frac{1}{\sqrt{2\pi\sigma^2_{\text{Class}}}} \exp\left(-\frac{(x_i - \mu_{\text{Class}})^2}{2\sigma^2_{\text{Class}}}\right)$$