

Multivariate Analysis

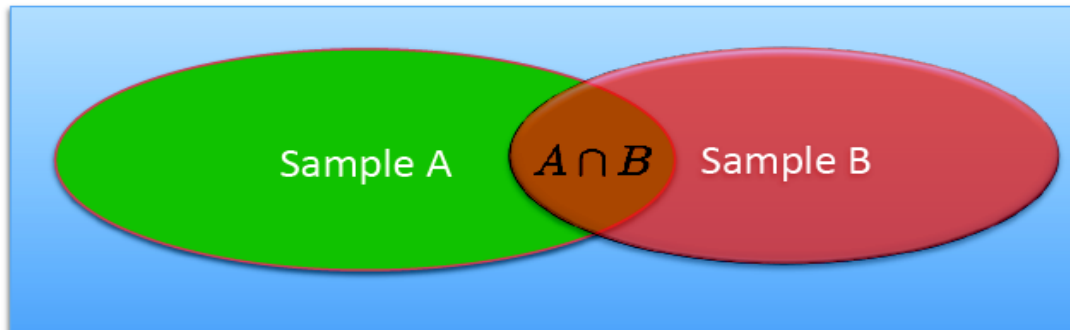
Wykład 1 Introduction

What do we mean by multivariate analysis?

- We start with a data sample of interesting events: U
 - Each event can be described in terms of n dimensions (or n *discriminating variables*) of interest.
- This sample contains more than one class of events: A, B, \dots
- Lets just consider the case of two classes (simple to generalize to more)
- So: $A \subset U$, and $B \subset U$

$$A \cap B \neq \emptyset$$

Note: If the intersection of A and B is null, then the problem is not interesting, and we can easily separate the two classes of interest with a set of cuts.



Q) How can we optimally separate classes A and B in n dimensions?

What do we mean by multivariate analysis?

- Consider the event e_i :
 - $e_i = e_i(\underline{x}) = e_i(x_1, x_2, x_3, \dots, x_n)$, which is the i^{th} event of a dataset U .
 - How do we determine the *Aness* or *Bness* of a given event $e_i = e_i(\underline{x})$?

• We need some way to assign a probability to the hypothesis that event e_i is of class A.

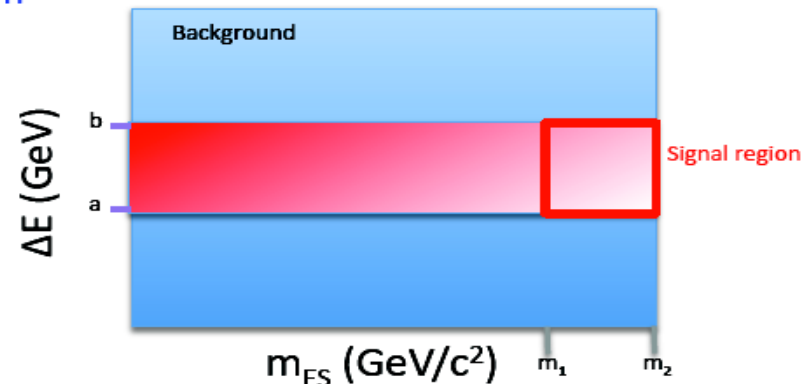
$$P(e_i \in A) \leq 1$$

• The complement is the probability that e_i is in the class B (as we are only considering two classes).

$$\overline{P(e_i \in A)} = P(e_i \in B) \leq 1$$

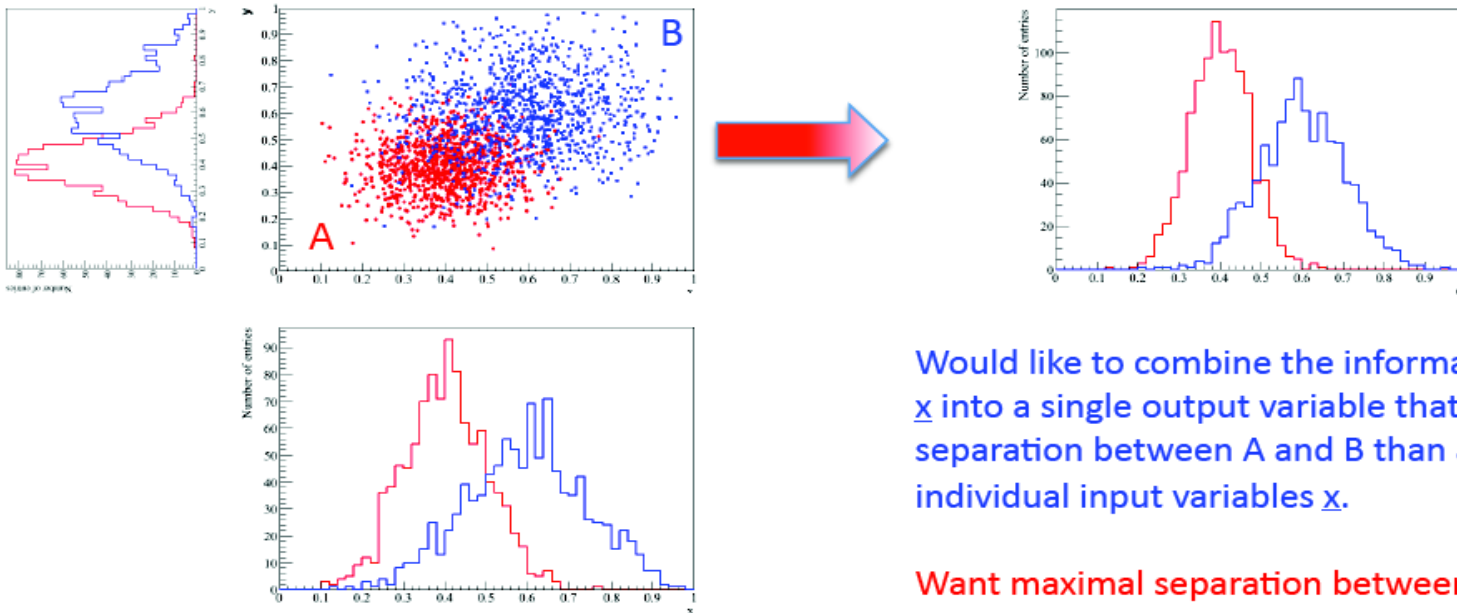
• Most of the time we can't tell for certain if an event e_i is of class A or class B.

Think of the familiar example of a signal region, where we know that the signal purity will be higher in this region, than outside it.



What do we mean by multivariate analysis?

- What are the desirable properties of the classification algorithm?



Would like to combine the information in \underline{x} into a single output variable that has a larger separation between A and B than any of the individual input variables \underline{x} .

Want maximal separation between A and B.

What do we mean by multivariate analysis?

- The MVA we perform requires that we choose an algorithm f to operate on e_i and produce some output O_i :

$$O_i = f(e_i)$$

- What is a good (or optimal) algorithm to use?
 - Separation of classes A and B → Experimental sensitivity:
 - Statistical precision on some measured observable?
 - Precision (including systematic uncertainties) on some measured observable?
 - Understandable algorithm:
 - Is it easy to understand what should have happened when defining the parameters of the function f ?
 - Can we tell when something has gone wrong?
 - Are there well known pathologies with a technique?
 - Ease of use:
 - Toolkits such as TMVA, StatPatternRecognition, TMultiLayerPerceptron, NeuralNetworkObjects (NNO) etc. mean we don't have to code f ourselves.
 - But ... that means we don't have to understand what happens inside the black box, which can be a very bad thing!
 - Should properly consider how training and uncertainties affect our choice of f .

What do we mean by multivariate analysis?

- The MVA we perform requires that we choose an algorithm f to operate on e_i and produce some output O_i :

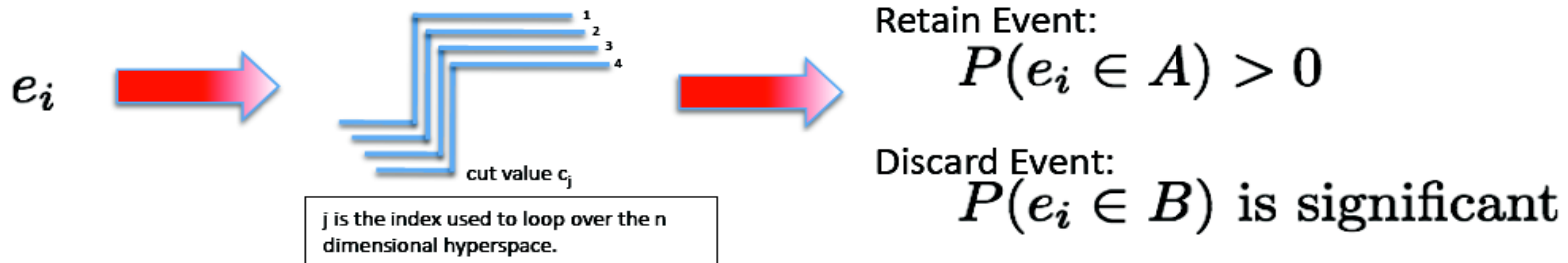
$$O_i = f(e_i)$$

- What is a good (or optimal) algorithm to use?
 - There is no single right answer to this question.
 - Need to understand the issues relevant to the problem we are trying to solve, and make a balanced decision as to what the best choice is.
 - As a rule of thumb – try and keep things simple
 - ... unless there is a significant gain to justify increased complexity).

**The best algorithm to use is the one you understand!
Never use an algorithm you don't ...**

Cutting on variables

- This is a (very) short recap of what you already know...
 - Apply an n-dimensional step function to an event e_i :



- Determine the cut values using some optimization recipe:
 - Significance S of the signal A in the presence of a background B .

$$S = \frac{N_A}{\sqrt{N_A + N_B}}$$

- In general the same steps are required to tune parameters of any Multivariate algorithm:
 - Define variables, algorithm, and recipe used to determine parameters of algorithm.
 - Compute response function of applying algorithm on data.

Fischer's linear discriminant

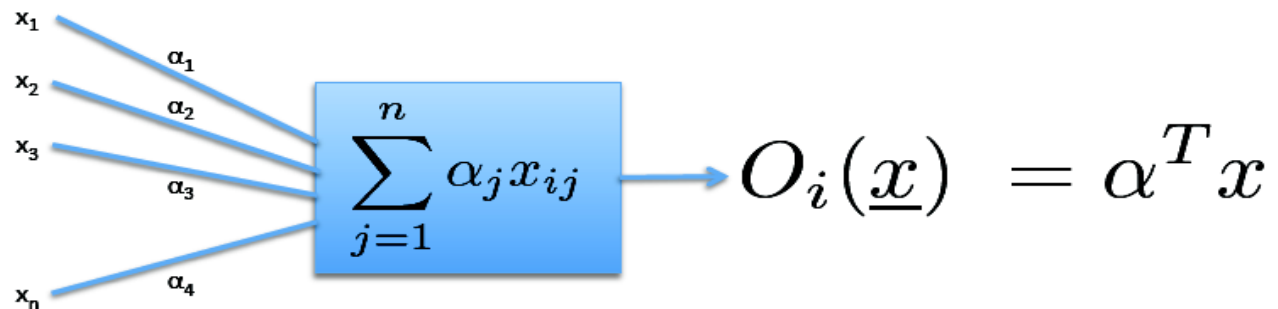
- Consider the case when we have a sample of data U containing a signal class and a background class.
 - We can use quantitative descriptors of the sets of signal and background events to develop a multivariate discriminating variable $O_i(\underline{x})$ for the i^{th} event:

$$O_i(\underline{x}) = \sum_{j=1}^n \alpha_j x_{ij} + \beta$$

- We usually drop β in our discussion as the offset is arbitrary, and set for convenience.
- Again we want to maximize the separation between signal and background.
 - How?
 - ... more to the point, how do we determine the coefficients $\underline{\alpha}$?

Fischer's linear discriminant

- For the i^{th} event



- What do we know about the data (Signal/Background)?
 - $\mu(x)$ and $\sigma(x)$ for each dimension for each type.
- We can write the Fisher mean and sigma of the corresponding signal and background distributions as:

$$M_{S,B} = \alpha^T \mu_{S,B}$$

$$\Sigma_{S,B}^2 = \alpha^T \sigma_{S,B}^2 \alpha$$

(just the vector sum of the scaled mean and variance using the corresponding weights in α)

Fischer's linear discriminant

- To maximize the separation between signal and background we want to
 - maximize $|M_S - M_B|$
 - minimize the variances Σ_S^2 and Σ_B^2

- We can balance these requirements with:

$$J(\alpha) = \frac{[M_S - M_B]^2}{\Sigma_S^2 + \Sigma_B^2}$$

- where:

$$[M_S - M_B]^2 = \sum_{i,j=1}^n \alpha_i \alpha_j (\mu_S - \mu_B)_i (\mu_S - \mu_B)_j = \alpha^T B \alpha$$

$$\Sigma_S^2 + \Sigma_B^2 = \sum_{i,j=1}^n \alpha_i \alpha_j (V_S + V_B)_{ij} = \alpha^T W \alpha$$

Fischer's linear discriminant

- So:

$$J(\alpha) = \frac{\alpha^T B \alpha}{\alpha^T W \alpha}$$

B represents the separation between the classes

W represents the sum of covariances within the classes

- Optimal separation can be found for

$$\frac{\partial J(\alpha)}{\partial \alpha_i} = 0$$

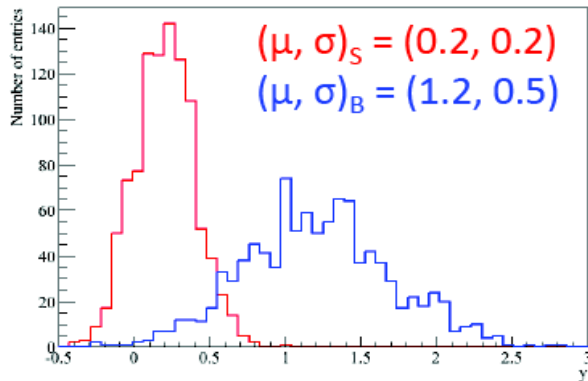
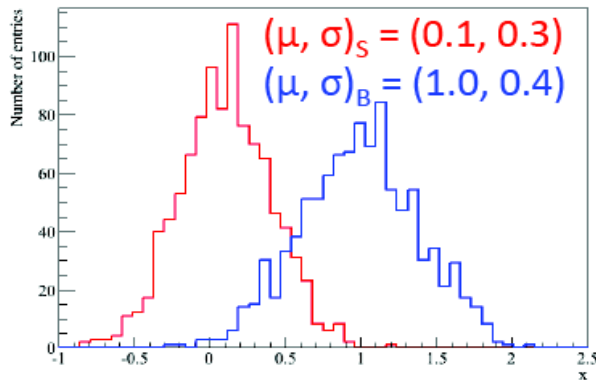
- where $\alpha \propto W^{-1}(\mu_S - \mu_B)$

Need to be able to compute W^{-1} . If W is singular then we can't use the Fisher method.

- So we can compute the output of the fisher given target samples of signal and background events.
 - The solution can be determined up to an arbitrary scale.
-

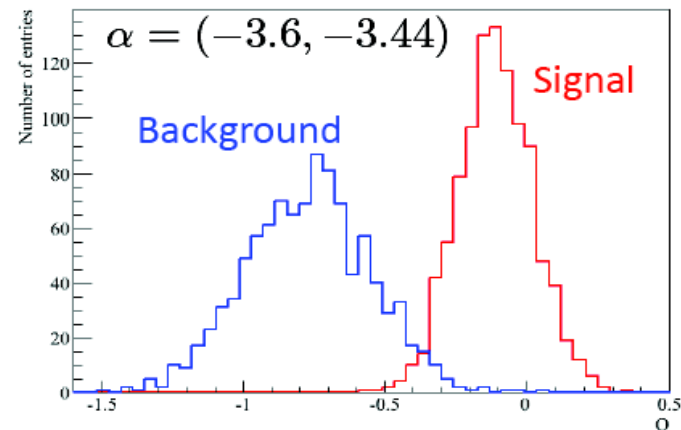
Fischer's linear discriminant

- Let's see an example



$$(\mu_S - \mu_B) = (-0.9, -1.0)$$

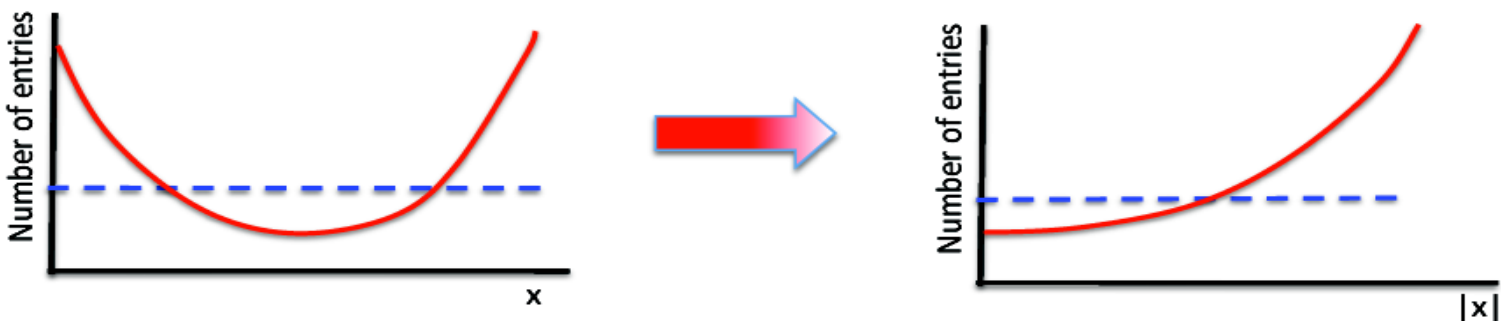
$$W = \begin{pmatrix} 0.25 & 0 \\ 0 & 0.29 \end{pmatrix}$$



(the Fisher discriminant output has signal as positive. Flip the sign in the mean difference term to reverse this behavior).

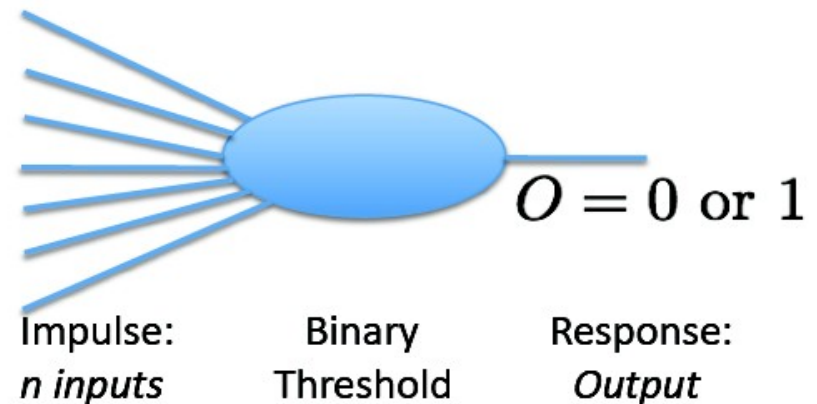
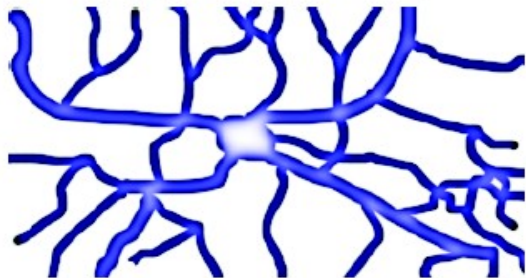
Fischer's linear discriminant

- Consider the variable x .
 - If this is symmetric about the mean value, and both signal and backgrounds have the same mean, then the variable will not contribute to the fisher.
 - So for this case $\alpha_i \approx 0$.
 - If the x distribution has a different shape for signal and background, then you can use $|x|$ as the fisher input instead.



Neural networks

- These are non-linear algorithms:
 - Called Artificial Neural Networks: ANN or just Neural Networks NN.
 - The fundamental building block of a NN is the perceptron (algorithmic analogy of a neuron).



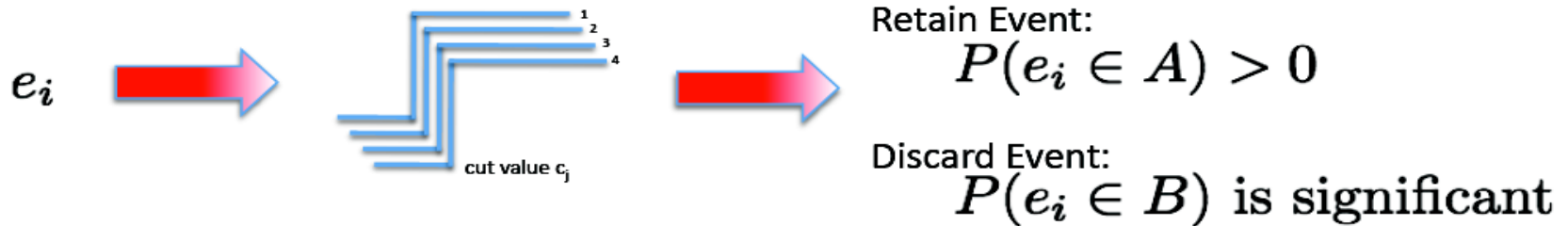
$$y = \underline{w} \cdot \underline{x} + b$$

If $y > 0$, then $O = 1$

y is the definition of a plane in n -dimensional hyperspace.

Neural networks

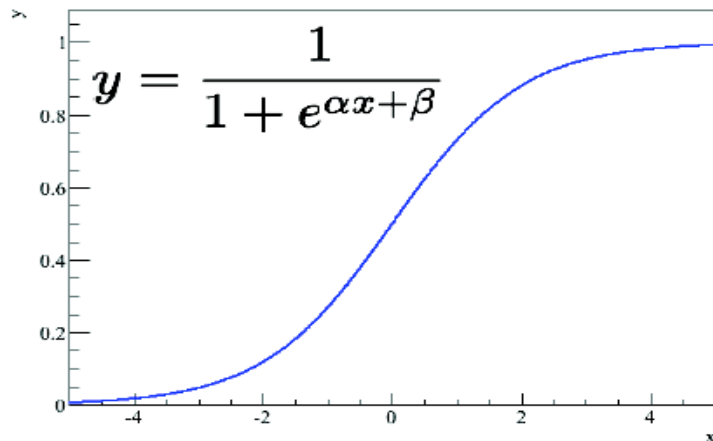
- The perceptron looks like familiar concept...
... recall the logic used when cutting on a set of variables



- The perceptron is doing the same thing for a given \underline{w} and b .
 - It makes an cut in the problem hyperspace.
- Recall that all cut information is encoded in \underline{c} for a cut based analysis.
- This tells us something useful: A *single* perceptron won't help us any more than optimally cutting on the data.
- Can replace the binary decision with any activation function:
$$y = f(\underline{w} \cdot \underline{x} + b)$$

Neural networks

- The next logical step: the Multi-Layer Perceptron (MLP).
 - Combine layers of perceptrons in a way so as to obtain a refined separation between classes A and B.
 - Modify the output of a perceptron so that it is some function with an output (usually) between 0 and 1:
 - Step function's can be used (as done up until now)
 - Any other suitable function can be adopted for the activation function.
 - The Sigmoid (or logistic) function is often used:



$$y = \frac{1}{1 + e^{\underline{\alpha} \cdot \underline{x} + \beta}}$$

commonly used activation functions include

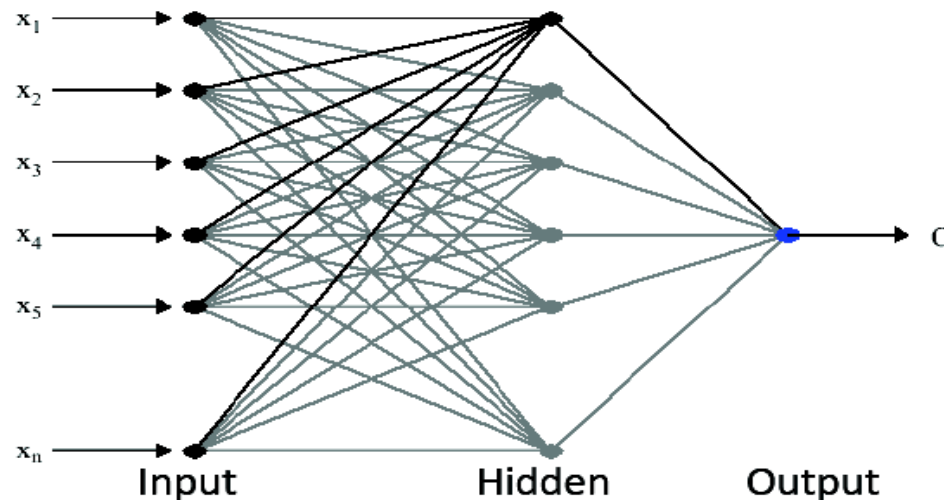
- sigmoid (0, 1)
- tanh (-1, +1)
- step (0, 1)
- radial (0, 1)

Neural networks

- An example of an MLP with
 - n inputs
 - 1 hidden layer of n nodes
 - 1 output

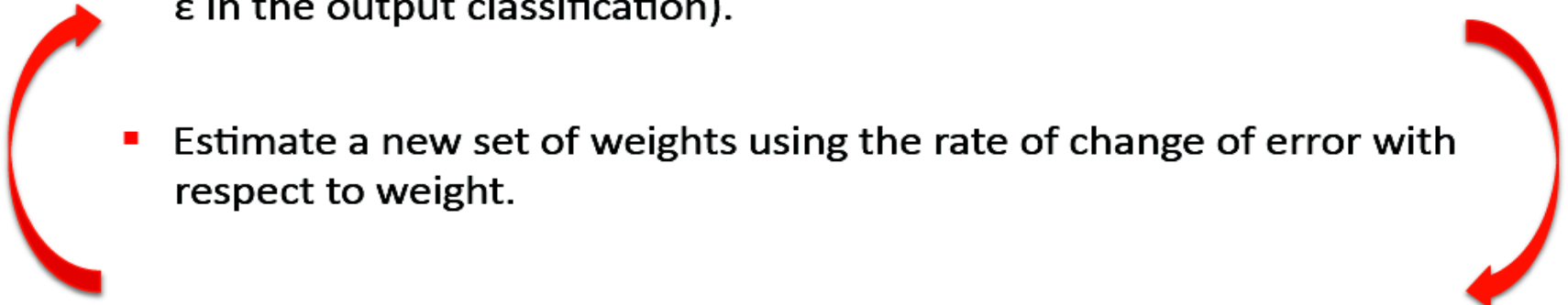
Looks good but...

... what are we supposed to do now?



- Decide on the activation function to use for each node/layer.
- Determine the weights used to evaluate y_i for each node.
- Check that we have not over-trained our network

How do we find weights?

- Start with an initial guess for the weights.
 - Determine how good an estimate this is (using a measure of the error ϵ in the output classification).
 - Estimate a new set of weights using the rate of change of error with respect to weight.
 - re-evaluate the error on the new set of weights.
 - When the result is *stable* and *good enough*, we stop iterating, and have determined the parameters that define the network.
 - Q) Is our solution the global minimum?
- 

How do we find weights?

- Consider for the moment the simple case of a single perceptron:
 - Class A ($t=1$) and Class B ($t=0$) events e_i from a total data sample U are input to the perceptron (supervised learning).
 - We want to train our algorithm, so we know the target type t_i of each event e_i .
 - Sometimes we get the classification wrong, and characterize this by an error ϵ_i :

$$\epsilon_i = \frac{1}{2} (t_i - y_i)^2$$

Diagram illustrating the error formula $\epsilon_i = \frac{1}{2} (t_i - y_i)^2$. Red arrows point from the labels below to the corresponding terms in the equation:

- Error (points to ϵ_i)
- Target type (points to t_i)
- output of perceptron (activation function) (points to y_i)

- So for the whole data sample U , containing N events, we have a total error E :

$$E = \sum_{i=1}^N \epsilon_i = \sum_{i=1}^N \frac{1}{2} (t_i - y_i)^2$$

How do we find weights?

- Consider for the moment the simple case of a single perceptron:
 - Class A ($t=1$) and Class B ($t=0$) events e_i from a total data sample U are input to the perceptron (supervised learning).
 - We want to train our algorithm, so we know the target type t_i of each event e_i .
 - Sometimes we get the classification wrong, and characterize this by an error ϵ_i :

$$\epsilon_i = \frac{1}{2} (t_i - y_i)^2$$

Diagram illustrating the error formula $\epsilon_i = \frac{1}{2} (t_i - y_i)^2$ with labels and arrows:

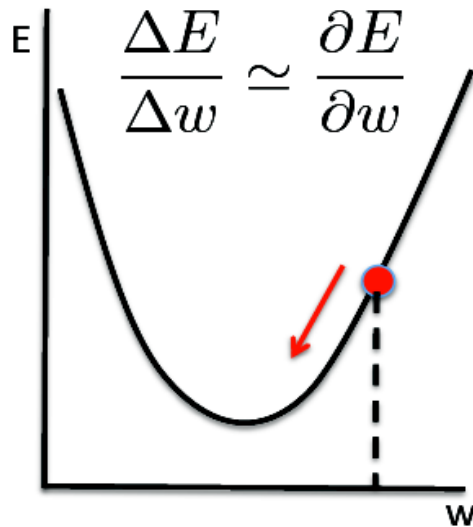
- Red arrow from "Error" to ϵ_i
- Red arrow from "Target type" to t_i
- Red arrow from "output of perceptron (activation function)" to y_i

- So for the whole data sample U , containing N events, we have a total error E :

$$E = \sum_{i=1}^N \epsilon_i = \sum_{i=1}^N \frac{1}{2} (t_i - y_i)^2$$

How do we find weights?

- Now that we have defined an error, we can:
 - Guess a set of weights.
 - Evaluate the error related to using those weights.
- Next we have to estimate the new set of weights:



• Want to try and estimate a new value for w , some small distance from the initial value: $w_0 \rightarrow w_0 + \Delta w$.

• At the same time we want to move toward the minimum, so let

$$\Delta w = -\alpha \frac{\partial E}{\partial w}$$

where α is a small positive parameter (learning rate).

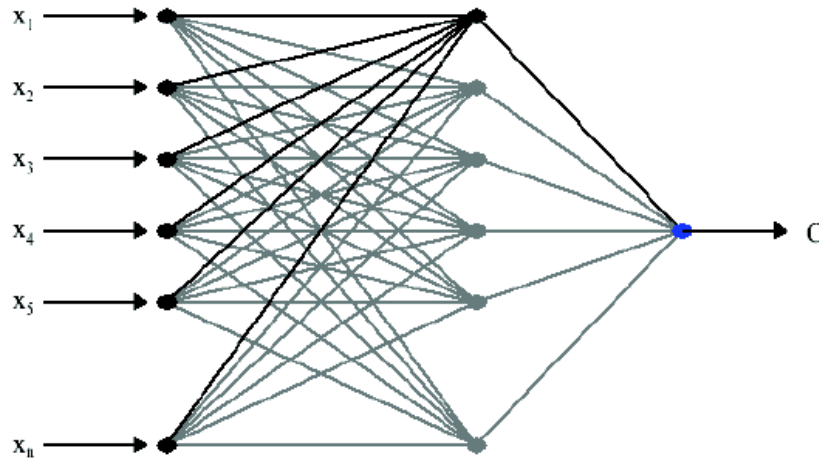
Hence:

$$\Delta E = -\alpha \left(\frac{\partial E}{\partial w} \right)^2$$

which is always negative.

Back propagation

- This is a supervised learning method that we can use to determine the weights (or parameters) of the activation functions used in our MLP.
 - This is a generalization of the delta rule.



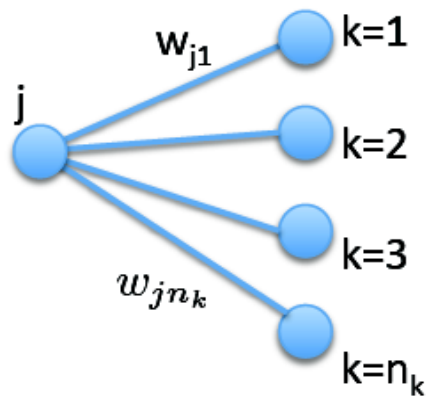
There are many nodes, hence many many weight parameters to determine when training an MLP.

This is a complicated problem akin to a multi-dimensional fit with many free parameters.

Need to consider the error contribution from each node in each layer.

Back propagation

- Apportioning blame (or contribution to the network)
 - Consider a node j , with some number of connections to the next layer of the MLP: k



x_{input} is the input value to the node j .
- Either input variable to MLP, or
output of a previous layer.

w_{jk} is the weight of the connection.

- We need to evaluate the error from each assignment of target value using the activation functions of each node.

Back propagation

- So we need to take into account error contributions from:

- Hidden layer nodes.
- Output layer nodes.



$$\Delta w_{jk} = \alpha x_{\text{input}} \Delta t \frac{\partial E}{\partial w}$$

Δt depends on the type of node.
 α is the learning rate.
 x_{input} is input to node j.

when training an MLP. These are given by:

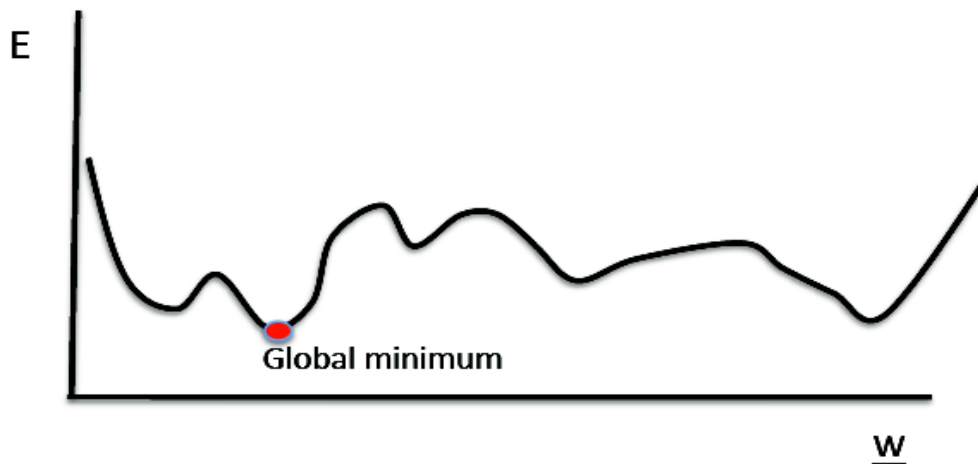
$$\text{Hidden node: } \Delta t = \sum_{k \text{ nodes}} (t_j - y_j) w_{jk}$$

$$\text{Output node: } \Delta t = t_j - y_j$$

- Again: this is a generalization of the delta rule, so the optimization algorithm works on error minimization/gradient descent.

Training MLP

- This is a multi-parameter problem.
- There are many minima, and we want to converge on the global minimum, not on a local one.



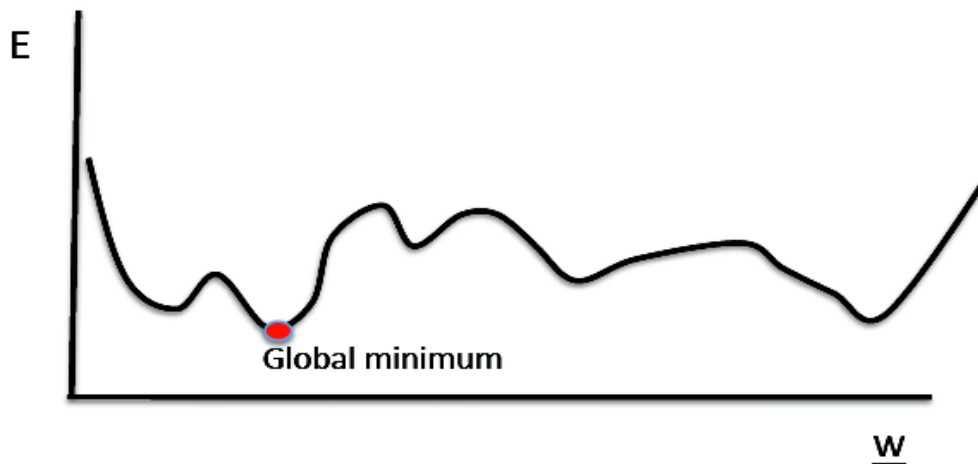
There are many nodes, hence many many weight parameters to determine when training an MLP.

This is a complicated problem akin to a multi-dimensional ML fit with many free parameters.

- Determining the global minimum can be non-trivial.

Training MLP

- This is a multi-parameter problem.
- There are many minima, and we want to converge on the global minimum, not on a local one.



There are many nodes, hence many many weight parameters to determine when training an MLP.

This is a complicated problem akin to a multi-dimensional ML fit with many free parameters.

- Determining the global minimum can be non-trivial.

Training MLP

- In order to train the MLP we need two samples of data:
 - Sample A, which is a data-set containing M entries of class A events.
 - Sample B, which is a data-set containing M entries of class B events.

You don't have to use equal numbers of events for both classes, however not doing so will affect the convergence of your network. You are advised to keep to using equal number of events in samples A and B.
- How do we know when training has finished?
 - Just compare the error against some anticipated threshold?
 - Just compare the error gradient against some anticipated threshold?
 - Compare the error obtained against a validation sample.

Training MLP

- Why use a validation sample?
 - Provides a statistically independent reference point.
 - Solution should be more robust than using all data for training.
 - Con: Slower convergence on the optimal solution.
 - Con: Corresponding limits on the ability to train a complicated net.
 - Pro: You've not fine-tuned your algorithm on a single statistically limited sample [You will always want more statistics].
 - Pro: If the training and validation samples perform the same with a set of weights, then you can have faith in the network configuration when applying it to data.

How much data do we need to train MLP?

- As a rule of thumb, the number of events scales with the complexity of the network as follows:

M = sample size

W = number of weight parameters

N = number of nodes

ϵ = error threshold

If there is a single hidden layer, to avoid failing to train a net properly you want to make sure that the training sample size M satisfies:

$$M > O\left(\frac{W}{\epsilon}\right)$$

If your sample doesn't satisfy this then you run a high risk of misclassification of events.

If the network is more complicated then you should try to ensure that:

$$M > O\left[\frac{W}{\epsilon} \log(N/\epsilon)\right]$$

Baum & Haussler, Neural Comp. 1 151-160 (1989)

How much data do we need to train MLP?

- Example:

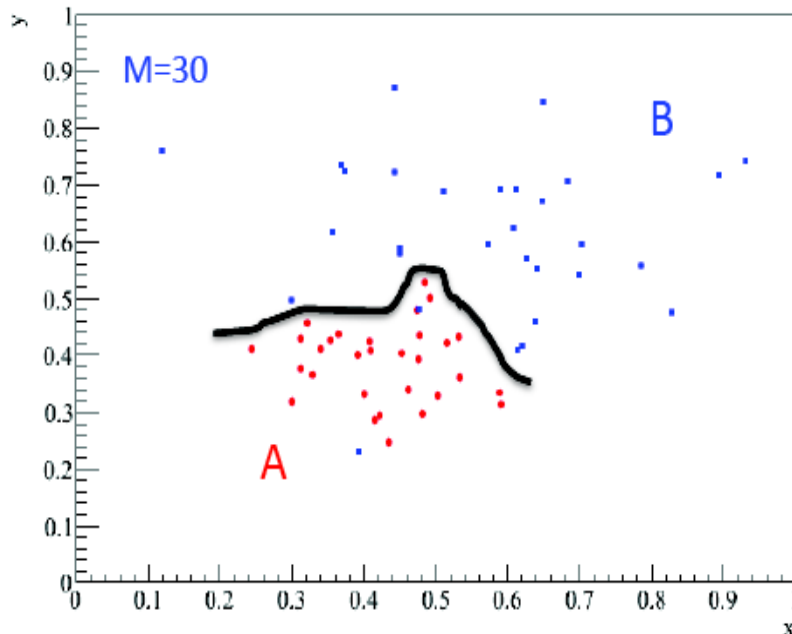
- an MLP with 1 hidden layer of 10 nodes, 10 inputs and 1 output node (so... $W=(10+1)\times 10$), and the misclassification error level you want to achieve is 0.1:

$$M > O\left(\frac{W}{\epsilon}\right)$$

- You want more than 1100 training events to have a reasonable chance of obtaining an optimal separation of signal and background.
- This doesn't mean that you get a properly trained net – you have to do some more checks to ensure that!

Validating the result

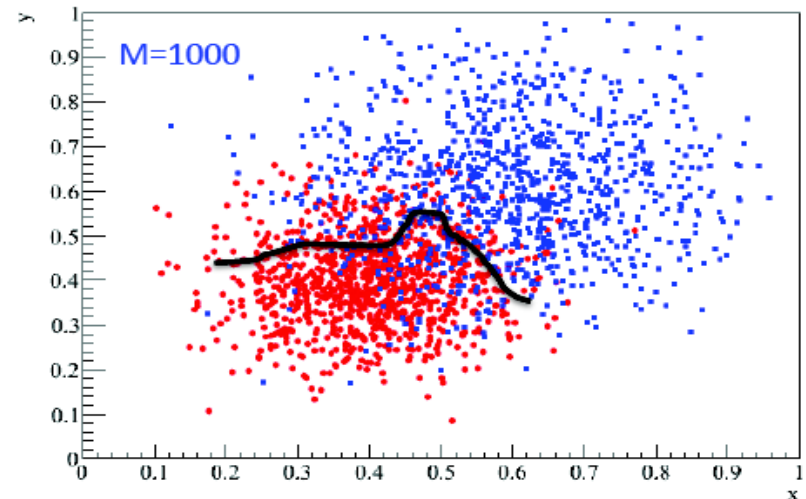
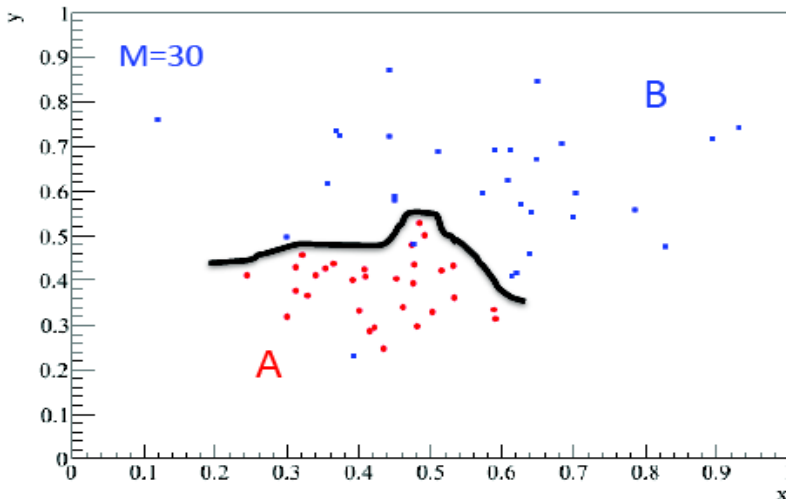
- Overtraining occurs when you have obtained weights that are tailored to your specific sample of A and B events, rather than being a true representation of the optimal discrimination between the classes.



Is the line a reasonable boundary to use as a cut between A and B?

Validating the result

- Overtraining occurs when you have obtained weights that are tailored to your specific sample of A and B events, rather than being a true representation of the optimal discrimination between the classes.

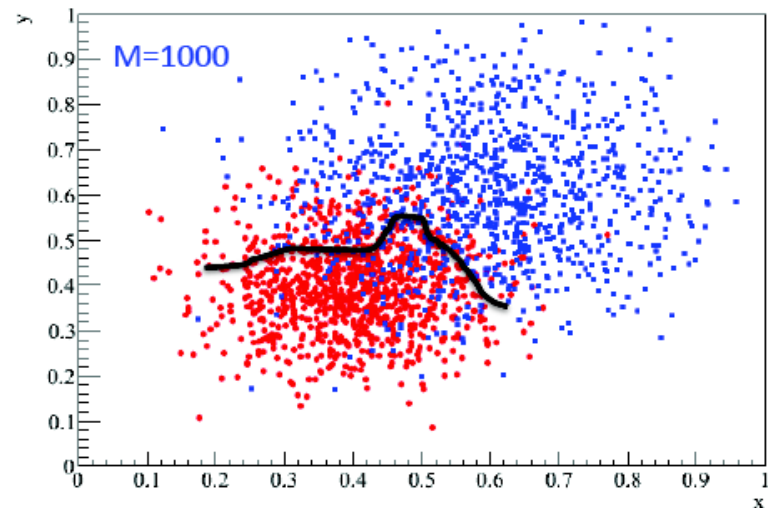
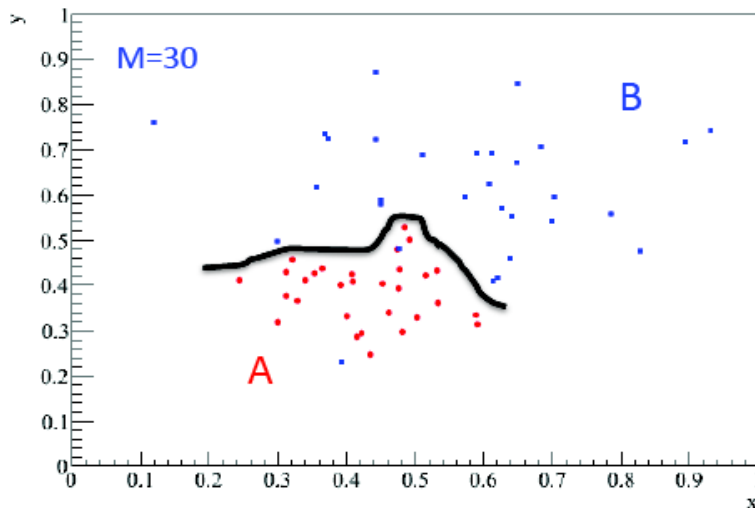


The boundary has been tailored to the initial sample of statistics and (in this case) is not the best choice of boundary for a separate sample.

This illustrates the need to have sufficient data to train. It highlights the issue of statistical fluctuations in data. **Don't tune on features of a specific data set!**

Validating the result

- Overtraining occurs when you have obtained weights that are tailored to your specific sample of A and B events, rather than being a true representation of the optimal discrimination between the classes.



A solution is to use a statistically independent sample to check the result of the training, and to stop training only when the the training and reference samples give the same performance (within tolerance).

Validating the result

- It is important that we have sufficient data to use in training:
 - Makes sure that the result is sensible.
 - Means that we can use an equal amount of data as a reference to compare against.
 - This can be a tough constraint as we often resort to MLPs when we want to extract every last bit of information from the data, and usually don't have events to spare!
- Similarly make sure that you don't over-train your MLP. How do you know if you are converging on a general feature of the data, or just a specific feature of your dataset?
 - Use a validation sample!
- The temptation is to use all data to train.
 - Don't do it as you can't guarantee the result is sensible!

Validating the result

- It is important that we have sufficient data to use in training:
 - Makes sure that the result is sensible.
 - Means that we can use an equal amount of data as a reference to compare against.
 - This can be a tough constraint as we often resort to MLPs when we want to extract every last bit of information from the data, and usually don't have events to spare!
- Similarly make sure that you don't over-train your MLP. How do you know if you are converging on a general feature of the data, or just a specific feature of your dataset?
 - Use a validation sample!
- The temptation is to use all data to train.
 - Don't do it as you can't guarantee the result is sensible!

Decision Trees

- Apply the initial rule to all data:

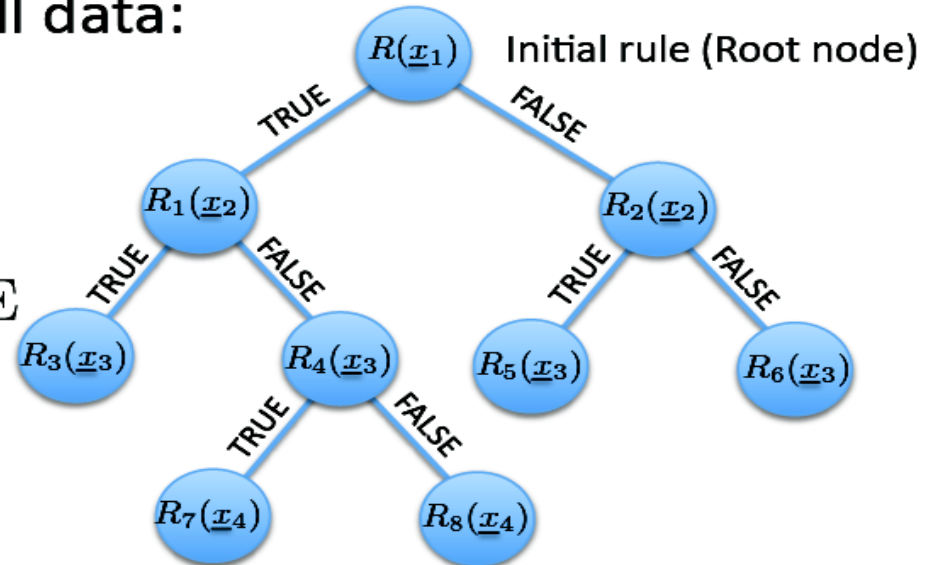
- Divide into two classes
(with a binary output)

$$\begin{aligned} R(\underline{x}_1) &= \underline{x} > \underline{x}_i \text{ TRUE} \\ &= \underline{x} < \underline{x}_i \text{ FALSE} \end{aligned}$$

- Each successive layer divides the data further into Signal (class A)/ Background (class B) classifications.

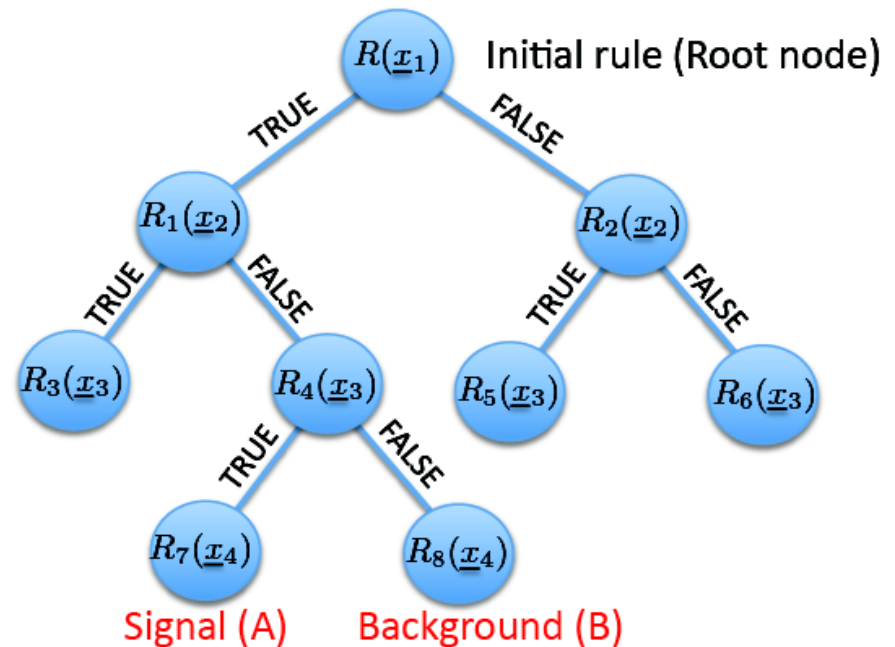
- The classification for a set of cut values will have a classification error.

- So just as with a NN one can vary the cut values x_i in order to train the tree.



Decision Trees

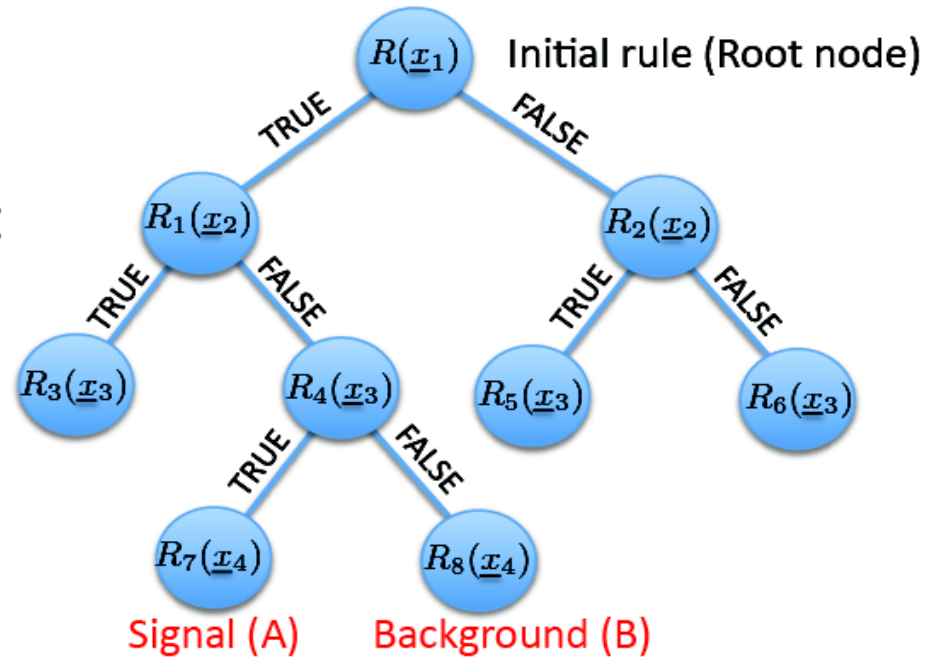
- What is happening?
- Each nodes uses the sub-set of discriminating variables that give the best separation between classes.



- Some variables may be used by more than one node.
- Other variables may never be used.

Decision Trees

- What is happening?
- The bottom of a tree just looks like a sub-sample of events subjected to a cut based analysis.



- There are many bottom levels to the tree:
 - ... so there are many signal / background regions defined by the algorithm.

Decision trees

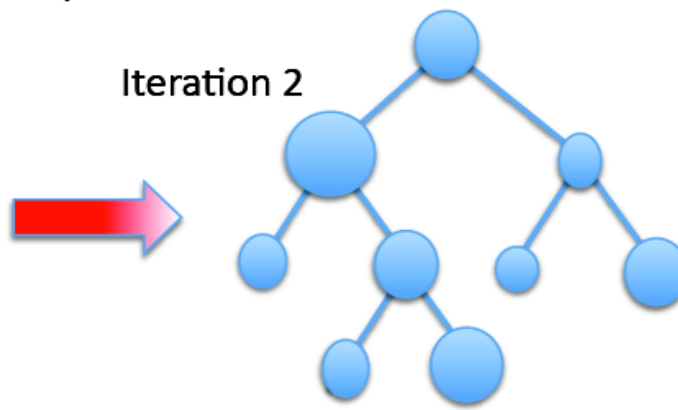
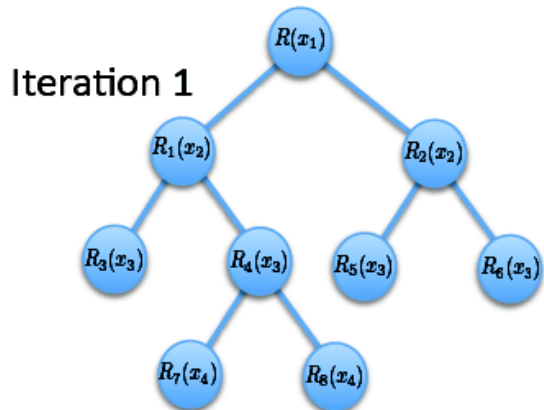
- Binary Decision Tree has the following pros/cons:
- Pros:
 - Easy to understand and interpret.
 - More flexibility in the algorithm when trying to separate classes of events.
 - Able to obtain better separation between classes of events than a simple cut-based approach.
- Cons:
 - Instability with respect to statistical fluctuations in the training sample.
- It is possible to improve upon the binary decision tree algorithm to try and overcome the instability or susceptibility of overtraining.

Boosted Decision Trees

- At each stage in training there may be some misclassification of events (error rate).
 - Assign a greater event weight α to mis-classified events in the next training iteration.

$$\alpha = \frac{1 - \epsilon}{\epsilon} \quad \epsilon = \text{error rate}$$

- Re-weight whole sample so that the sum of weights remains the same, then iterate.



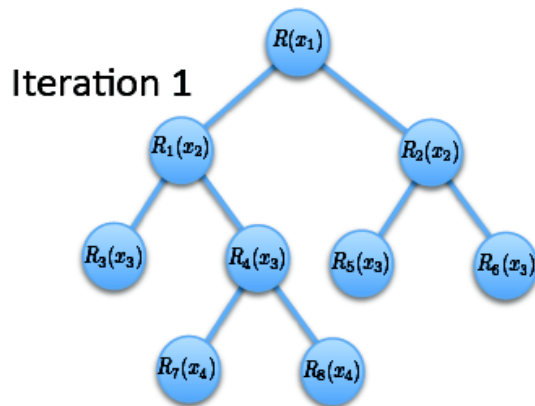
By re-weighting misclassified events by α the aim is to reduce the error rate of the trained tree, compared with an un-boosted algorithm.

Boosted Decision Trees

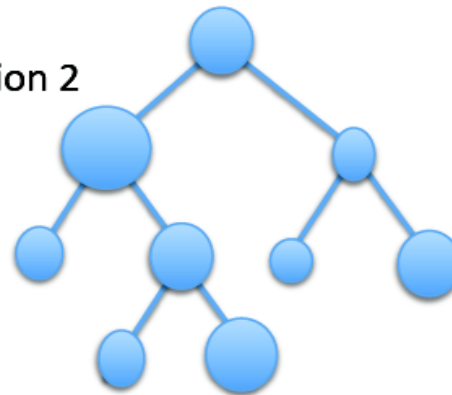
- At each stage in training there may be some misclassification of events (error rate).
 - Assign a greater event weight α to mis-classified events in the next training iteration.

$$\alpha = \frac{1 - \epsilon}{\epsilon} \quad \epsilon = \text{error rate}$$

- Re-weight whole sample so that the sum of weights remains the same, then iterate.



Iteration 2



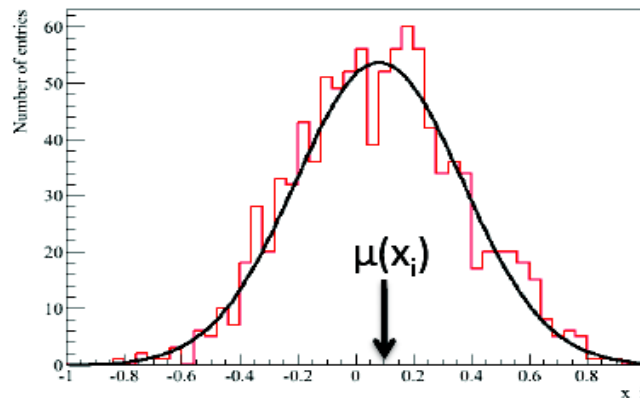
The resulting Boosted Decision Tree tends to be more stable than a normal Decision Tree.

Bagged Decision Trees

- Aim: To improve the stability of a Decision Tree algorithm.
- Solution: Sample the training data used to determine the solution.
- Take the average solution of a number of re-sampled solutions.
 - This re-sampling removes the problem of fine tuning on statistical fluctuations.

Like choosing the mean value of a cut at each level.

Again, the results tend to be more stable than just using a decision tree.



Forests

- A given decision tree may not be stable, so instead we can grow a forest.
 - For a forest, the classification of an event e_i in sample A or B is determined as the dominant result of all of the tree classifications for that event.

e.g. In a forest of 100 trees, if there are 80 classifications of type A, and 20 of type B, the event e_i is considered to be of type A.
- Trees in a forest use a common training sample, and are typically boosted.

Randomised Trees / Prunning

- **Randomized Trees:**
 - A finesse of the previous algorithms is to grow trees using a random sub-set of variables at each node.
- **Pruning:**
 - Some nodes may be insignificant in obtaining the final tree.
 - It is best to train the tree, then remove the insignificant nodes after the fact.
 - Pruning starts at the bottom and works upward.

Decision Trees

- The ranking of an input variable in a decision tree is derived from:
 - The number of times that it is used in one of the nodes of the tree.
 - Weighting based on the square-class-separation .
- Some final remarks:
 - Insensitive to weak discriminating variables.
 - Simple to understand (compared to a NN for example)
 - Best theoretical performance for a decision tree is inferior to neural networks.

Summary

- We have discussed the problem of optimally separating different classes of events using several algorithms:
 - Cutting on the n-dimensional hyperspace
 - Using a Fisher discriminant
 - Using an artificial neural network in the form of an MLP.
- The techniques used in each case have been reviewed, with some of the technical details on how to determine the cut parameters or weights.
- We have looked at some modern MVA techniques to extend our understanding beyond cutting on data, Fisher discriminants, and MLPs.