Statistical analysis of experimental data
Machine Learning

Aleksander Filip Żarnecki

FACULTY OF PHYSICS
UNIVERSITY OF WARSAW

Lecture 12

January 19, 2023

Machine Learning

1. Artificial Neural Networks
2. Boosting
3. Decision Trees
4. Boosted Decision Trees
5. Homework

Event classification

Problem definition

The problem is similar to the one discussed in lecture 10: we want to discriminate between two model hypothesis H_0 and H_1 based on the collected data D.

Different case - **classification of collected measurements**:

- H_0 - measurement can be attributed to the Standard Model,
- H_1 - measurement is due to BSM contribution,
- D - single measurement ("event" in HEP experiments)

According to Neymann and Pearson, the optimal, "most powerful" method to discriminate between the two hypothesis is to look at likelihood ratio

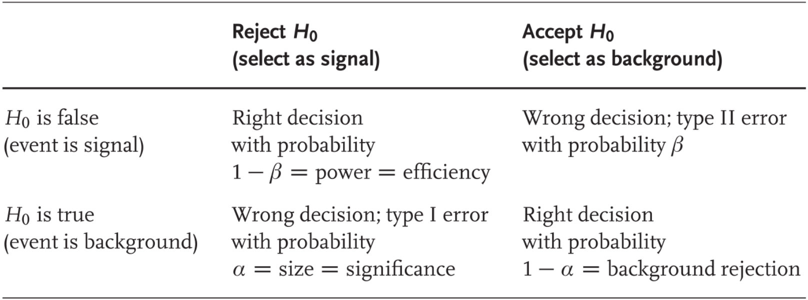
$$Q(D) = \frac{L(D|H_1)}{L(D|H_0)}$$

Event classification

Classification errors

O.Behnke et. al, *Data Analysis in High Enegy Physics*

Selecting the classification cut, two types of error need to be considered



	Reject H_0 **(select as signal)**	**Accept H_0** **(select as background)**
H_0 is false (event is signal)	Right decision with probability $1 - \beta$ = power = efficiency	Wrong decision; type II error with probability β
H_0 is true (event is background)	Wrong decision; type I error with probability α = size = significance	Right decision with probability $1 - \alpha$ = background rejection

Probability of accepting fake

$$\alpha = \int_{y(m)>y_{cut}} dm\ p(m|H_0)$$

Probability of rejecting good

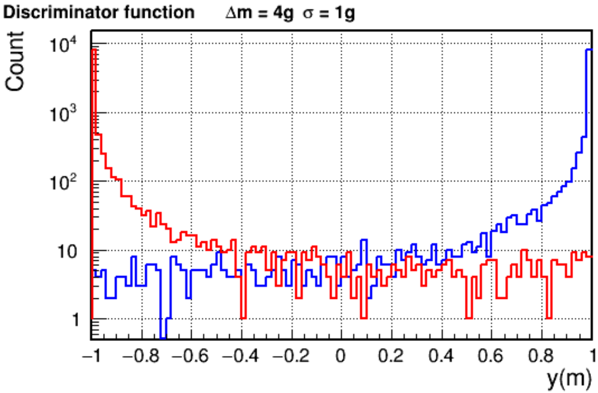
$$\beta = \int_{y(m)<y_{cut}} dm\ p(m|H_1)$$

Event classification

Simple example

Discriminator function distribution

We expect $y \to -1$ for fake coin, $y \to +1$ for good coin

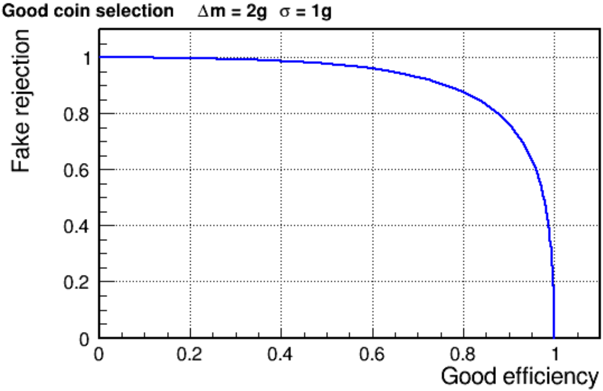


Event classification

ROC curve

For both good and fake coins, efficiency depends on the assumed y_{cut} value. All possible choices on a Receiver-Operating-Characteristic curve:

$1 - \alpha$



$1 - \beta$

In the realistic case, we can not have $\alpha \to 0$ and $\beta \to 0$ at the same time...

Optimal cut value strongly depends on the actual goal of the analysis...

Likelihood classifier

Single measurement (event) often corresponds to a set of observables:

$$\mathbf{x} \;=\; (x_1, x_2, \ldots x_N)$$

If N is large, it is difficult to reconstruct probability density function of $\mathbf{x}$.

We usually start from considering probabilities for single variable:

$$p_k^{(j)}(x_j) \;=\; P(x_j|H_k) \;=\; \int \cdots \int_{i\neq j} dx_i \; P(\mathbf{x}, H_k) \qquad k = 1, 2$$

We can then apply the Bayes' Theorem to single variable distribution:

$$P(H_1|x_j) \;=\; \frac{f_1 \cdot p_1^{(j)}(x_j)}{f_1 \; p_1^{(j)}(x_j) + (1 - f_1) \; p_0^{(j)}(x_j)}$$

Naive Bayes Classifier

Likelihood classifier

Assuming the absence of correlations between the observables,
treating different observables as independent random variables,
multi-deminsional pdf can be calculated as a product of variable pdfs.

Likelihood of hypothesis k for measured event $\mathbf{x}$ is then given by

$$L_k(\mathbf{x}) = L(H_k|\mathbf{x}) = \prod_j P(H_k|x_j)$$

We can then construct the classifier based on the likelihood ratio:

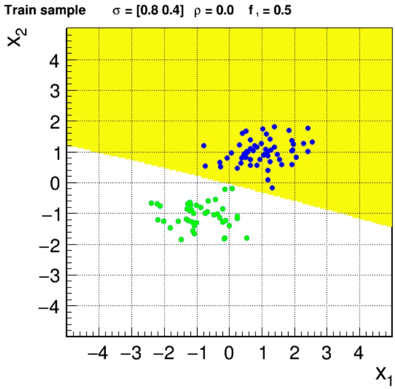
$$\gamma(\mathbf{x}) = \frac{L_1(\mathbf{x})}{L_0(\mathbf{x}) + L_1(\mathbf{x})}$$

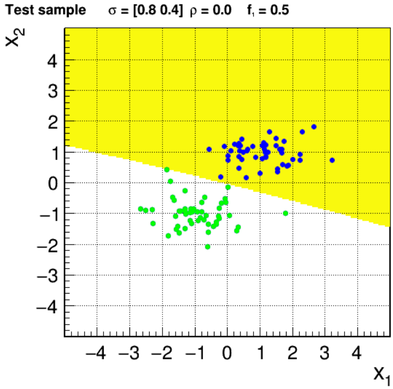
which should be equivalent to the Neyman-Pearson classifier.
Assuming correlations can be neglected and in the limit of large training samples.

Naive Bayes Classifier

Example

Efficient classification can be obtained for uncorrelated variables.



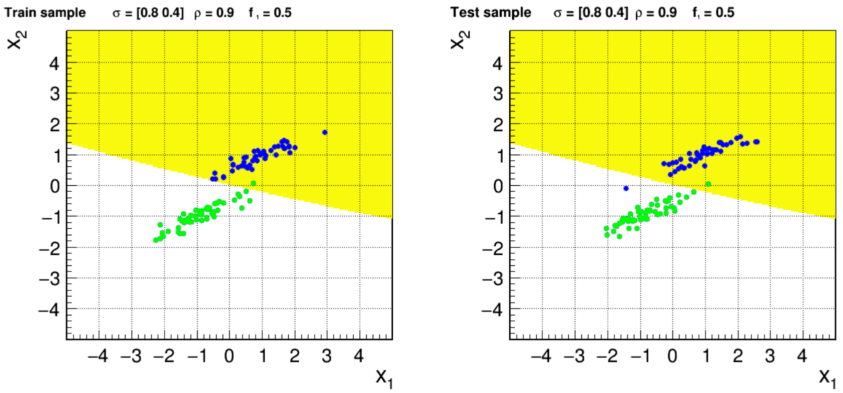


Implementation of the Gaussian Naive Bayes Classifier in **sklearn**.

Naive Bayes Classifier

Example

Efficient classification can be obtained for uncorrelated variables.
However, it is clearly far from optimal in case of correlations!



Implementation of the Gaussian Naive Bayes Classifier in **sklearn**.

Nearest neighbors classifier

Principle

This classifier refers directly to the Neymann and Pearson Lemma.

It is based on the expectation, that the likelihood ratio can be related to the ratio of the expected event densities:

$$Q(\mathbf{x}) \;=\; \frac{L(\mathbf{x}|H_1)}{L(\mathbf{x}|H_0)} = \frac{1}{N_1}\frac{dN_1}{d\mathbf{x}} \left(\frac{1}{N_0}\frac{dN_0}{d\mathbf{x}}\right)^{-1} = \frac{N_0}{N_1}\,\frac{dN_1}{dN_0}$$

where dN_k represent the expected number of events for hypothesis k, in a small variable space volume $d\mathbf{x}$ (in the limit $d\mathbf{x} \to 0$, $N_k \to \infty$)

The idea is to replace the expected event densities dN_k by numbers of events in the actual data (training sample including H_0 and H_1 events):

$$dN_k(\mathbf{x}) \;\to\; n_k(\mathbf{x}) \;=\; \sum_{\mathbf{x}' \in \Delta(\mathbf{x})} \mathbf{x}' \subset H_k$$

The key point is how to define "neighborhood" region $\Delta(\mathbf{x})$ of point $\mathbf{x}$

Nearest neighbors classifier

Principle

Two possible approaches are commonly used.

One can define $\Delta(\mathbf{x})$ by specifying maximum distance d between points:

$$\Delta(\mathbf{x}) \;=\; \{x' \;:\; d(\mathbf{x}',\mathbf{x}) < R_{max}\}$$

However, R_{max} has to be sufficiently large to always accept a sample of training events, also in the regions of lowest probability density...
That is why this approach is not very efficient...

This problem is “solved” in the “k nearest neighbors” (kNN) classification.
We sort training events **x**’ according to their distance from the test point **x** and take the closest k points:

$$\Delta(\mathbf{x}) \;=\; \{x' \;:\; d(\mathbf{x}',\mathbf{x}) < R(\mathbf{x})\} \quad \text{and} \quad R(\mathbf{x}) \;:\; \sum_{\mathbf{x}'} 1 = k$$

Where we still need to define the distance measure $d(\mathbf{x}',\mathbf{x})$...

k Nearest Neighbors

To take different variable scales int account, one could redefine considered set of variables to span the same numerical range and use Euclidean metric. This however will neglects possible correlations.

In the general case, distance measure properly reflecting properties of the data set should be used. Frequent choice:

$$d^2(\mathbf{x}', \mathbf{x}) \;=\; (\mathbf{x}' - \mathbf{x})^\intercal \, \mathbb{C}_{\mathbf{x}}^{-1} \, (\mathbf{x}' - \mathbf{x}) \;=\; \sum_{jk} (x'_j - x_j) \, (\mathbb{C}_{\mathbf{x}}^{-1})_{jk} \, (x'_k - x_k)$$

where $\mathbb{C}_{\mathbf{x}}$ is the covariance matrix of the measurement.

This is so-called "Mahalanobis distance" measure.

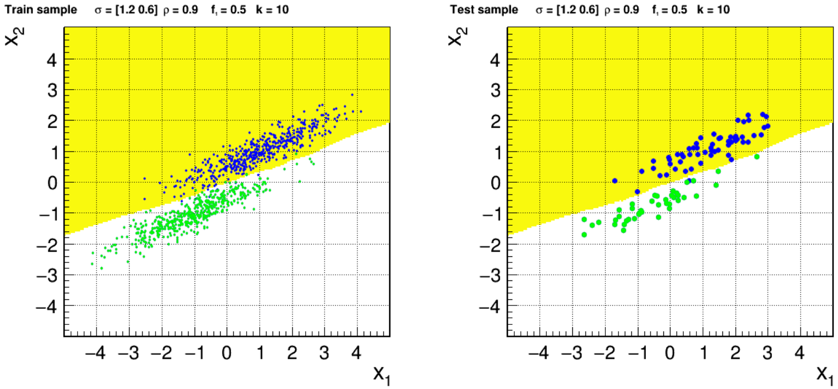
Similar to the calculation of the χ^2 value between two points

Nearest neighbors classifier

*k*NN example

separation of 2D Gaussian distributions

Two-dimensional data set with large correlation between variables
With Mahalanobis distance measure (including correlations)

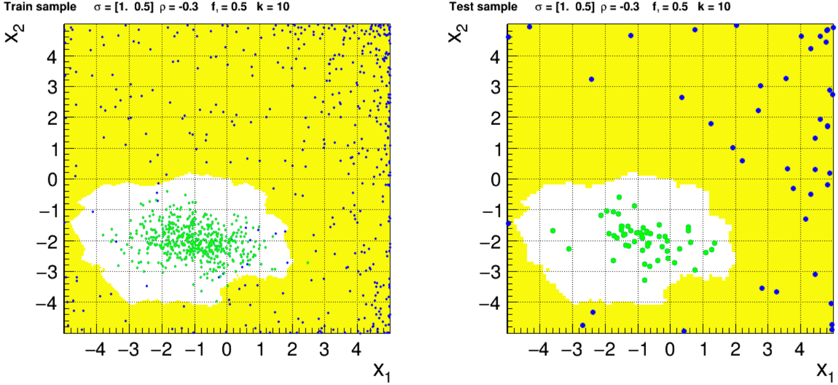


Implementation of the *k* Nearest Neighbors classifier in **sklearn**.

Nearest neighbors classifier

k Nearest Neighbors

While Naive Bayes and Fisher Linear Classifiers are based on modeling the likelihood distribution, nearest neighbors classifier is very general, can be used in (almost) any case. Example application:



Unfortunately, it is also very slow and requires large training samples...

Fisher Linear Discriminant

Linear discriminant

(Behnke)

Classifier based on the linear combination of input variables:

$$F(\mathbf{x};\mathbf{w}) \;=\; w_0 + \sum_{j=1}^{N} w_j\, x_j \;=\; w_0 + \mathbf{w}\cdot\mathbf{x}$$

Resulting decision boundaries, $F(\mathbf{x}) = F_{cut}$, are hyperplanes in N dim.

Weight vector **w** defines the direction, on which all events are projected.
Projection "reduces" the N variable problem to single variable $F(\mathbf{x})$.

If we assume Gaussian variable distributions, we can look at the direction which maximizes the relative distance between the two hypothesis in F:

$$D(\mathbf{w}) \;=\; \frac{(h_1 - h_0)^2}{\sigma_1^2 + \sigma_0^2}$$

h_k and σ_k^2 are the expected values and variances of $F(\mathbf{x})$ for hypothesis k.

Fisher Linear Discriminant

Linear discriminant

(Behnke)

Classifier based on the linear combination of input variables:

$$F(\mathbf{x};\mathbf{w}) \;=\; w_0 + \sum_{j=1}^{N} w_j\, x_j \;=\; w_0 + \mathbf{w}\cdot\mathbf{x}$$

Resulting decision boundaries, $F(\mathbf{x}) = F_{cut}$, are hyperplanes in N dim.

Weight vector **w** defines the direction, on which all events are projected.
Projection "reduces" the N variable problem to single variable $F(\mathbf{x})$.

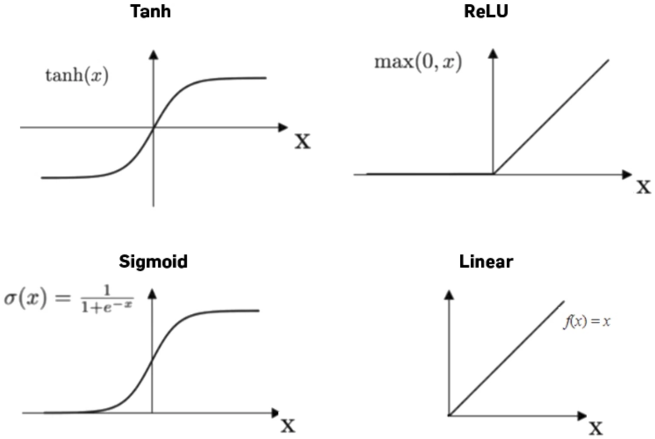
However, the problem can be also solved without looking at the global properties, by minimizing the "loss function". Possible choice, "distance":

$$L(\mathbf{w}) \;=\; \sum_{\text{events } i} \left[t^{(i)} - y(F(\mathbf{x}^{(i)};\mathbf{w})) \right]^2$$

where y is the "activation function", $t^{(i)}$ is true class of event $\mathbf{x}^{(i)}$.

Iterative procedure

Activation function



Source: Artificial Intelligence Wiki

Iterative procedure

Perceptron Learning **"Learning on errors"**

One can consider the iterative procedure of adjusting the weights:

$$\mathbf{w}^{(n+1)} = \mathbf{w}^{(n)} - \eta \sum_{i} \left(y^{(i)} - t^{(i)} \right) \cdot \mathbf{x}^{(i)}$$

where η is the learning rate parameter.

Events which are incorrectly classified contribute most to loss function.
They also have largest impact in the weigh adjustment procedure...

This approach was first proposed by M. Rosenblatt in 1958.

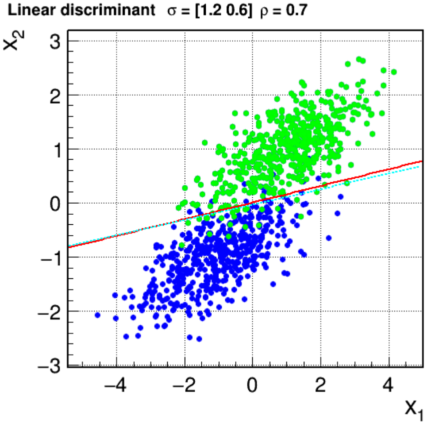
Weight correction can be applied on event by event basis (starting from the beginning when event loop completed) or calculating global correction for the whole sample.

Surprisingly, with proper choice of η this procedure works, results in classification optimization, even without referring to the loss function...

Perceptron Learning example

Example results for linear discriminant, starting from random weights:

N=1000



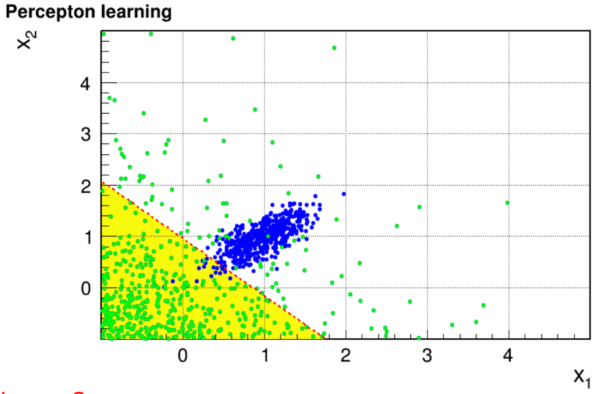
Iterative procedure (dashed cyan) compared with Fisher discriminant (solid red)

Machine Learning

1. Artificial Neural Networks
2. Boosting
3. Decision Trees
4. Boosted Decision Trees
5. Homework

Linear discriminant — Single percepton training

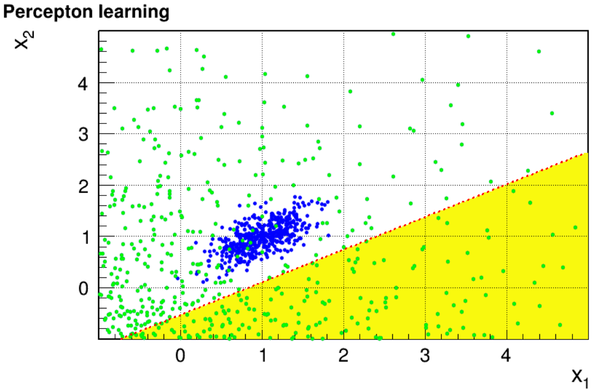
Linear discriminant is quite effective for separation of two Gaussian samples, but clearly not optimal for more complicated cases



Can we do better?

Linear discriminant Single percepton training

Linear discriminant is quite effective for separation of two Gaussian samples, but clearly not optimal for more complicated cases

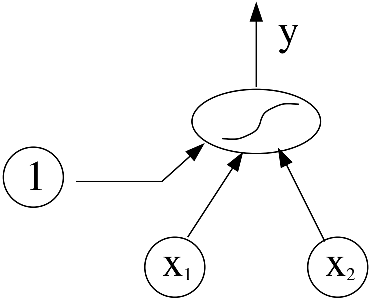


Can we do better?

Artificial Neural Networks

Single percepton

We can present the data flow in as a simple diagram:



Classification is based on the output y of the activation function.

Activation function is calculated for a linear combination of three inputs:

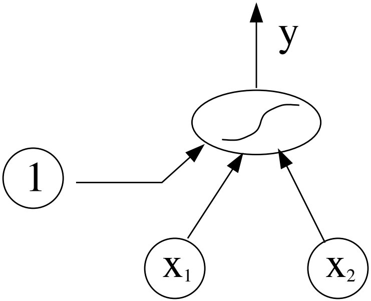
- two input variables, x_1 and x_2
- constant offset (1)

Input weights can be found in the iterative "learning procedure"

$$\mathbf{w}^{(n+1)} = \mathbf{w}^{(n)} - \eta \sum_{events} \left(y^{(i)} - t^{(i)} \right) \cdot \mathbf{x}^{(i)}$$

Single percepton

We can present the data flow in as a simple diagram:



Classification is based on the output *y* of the activation function.

Activation function is calculated for a linear combination of three inputs:

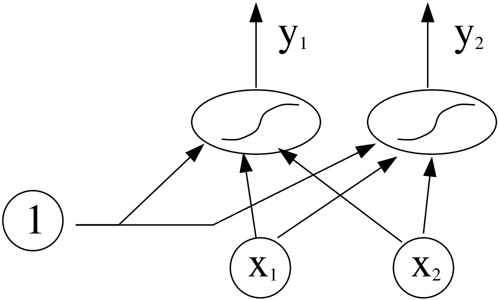
- two input variables, x_1 and x_2
- constant offset (1)

Input weights can be found in the iterative "learning procedure"

But single linear combination always results in a linear decision boundary...

Two perceptons

We can try to train two independent classifiers:



If starting from random initial weights, training results could be different...

But how to combine them?

Two percepton layers

It seems quite natural to add additional percepton to combine the two...

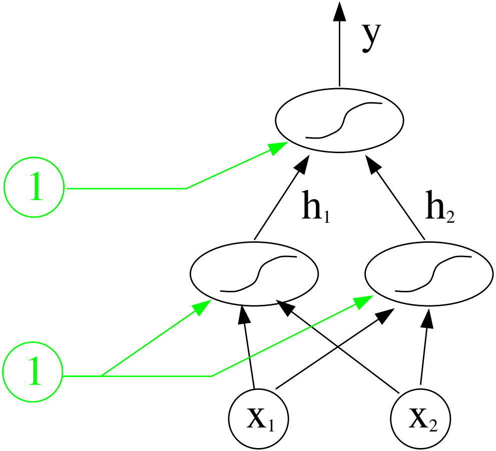
Output-layer neuron:

$$y \;=\; f\left(w_0^{(1)} + \sum_{j=1}^{2} w_j^{(1)}\, h_j\right)$$

Hidden-layer neuron:

$$h_j \;=\; f\left(w_{j,0}^{(2)} + \sum_{k=1}^{2} w_{j,k}^{(2)}\, x_k\right)$$

⇒ nine independent weights
one for each arrow



Learning rules

Miroslav Kubat, *An Introduction to Machine Learning*

Backpropagation of Errors: contribution of event i to the weight-adjusting procedure is proportional to the classification error:

$$\delta_i^{(1)} \quad = \quad (y_i - t_i)$$

Learning rules

Miroslav Kubat, *An Introduction to Machine Learning*

Backpropagation of Errors: contribution of event i to the weight-adjusting procedure is proportional to the classification error:

$$\delta_i^{(1)} = (y_i - t_i)\,(1 - y_i)\,(1 + y_i)$$

Additional factor reduces impact of “well classified” events, $y \to \pm 1$
$\Rightarrow$ we focus on those where classification was “weak”, $y_i \sim 0$.

Learning rules

Miroslav Kubat, *An Introduction to Machine Learning*

Backpropagation of Errors: contribution of event *i* to the weight-adjusting procedure is proportional to the classification error:

$$\delta_i^{(1)} = (y_i - t_i)\,(1 - y_i)\,(1 + y_i)$$

Additional factor reduces impact of "well classified" events, $y \to \pm 1$
$\Rightarrow$ we focus on those where classification was "weak", $y_i \sim 0$.

For the output layer neurons, we can apply procedure similar to the percepton learning:

$$\mathbf{w}^{(1)(n+1)} = \mathbf{w}^{(1)(n)} - \eta \sum_i \delta_i^{(1)} \cdot \mathbf{h}_i$$

where $\mathbf{h}_i$ is the vector of hidden layer results + offset

Learning rules

For hidden layer, we need to define the corresponding "error" for each node j. We "back propagate" it for each event from the output node:

$$\delta^{(2)}_{j,i} \;=\; w^{(1)}_j \, \delta^{(1)}_i \, (1 - h_{j,i}) \, (1 + h_{j,i})$$

where we include weight $w^{(1)}_j$ connecting given node to output neuron. Again, we suppress impact of events with "strong opinion".

Learning rules

For hidden layer, we need to define the corresponding “error” for each node j. We “back propagate” it for each event from the output node:

$$\delta_{j,i}^{(2)} \;=\; w_j^{(1)}\,\delta_i^{(1)}\,(1-h_{j,i})\,(1+h_{j,i})$$

where we include weight $w_j^{(1)}$ connecting given node to output neuron. Again, we suppress impact of events with “strong opinion”.

Weight update rule for hidden layer neurons:

$$\mathbf{w}_j^{(2)(n+1)} \;=\; \mathbf{w}_j^{(2)(n)} \;-\; \eta\,\sum_i \delta_{j,i}^{(2)}\cdot\mathbf{x}_i$$

Learning rules

For hidden layer, we need to define the corresponding "error" for each node j. We "back propagate" it for each event from the output node:

$$\delta^{(2)}_{j,i} \;=\; w^{(1)}_j \, \delta^{(1)}_i \, (1 - h_{j,i}) \, (1 + h_{j,i})$$

where we include weight $w^{(1)}_j$ connecting given node to output neuron. Again, we suppress impact of events with "strong opinion".

Weight update rule for hidden layer neurons:

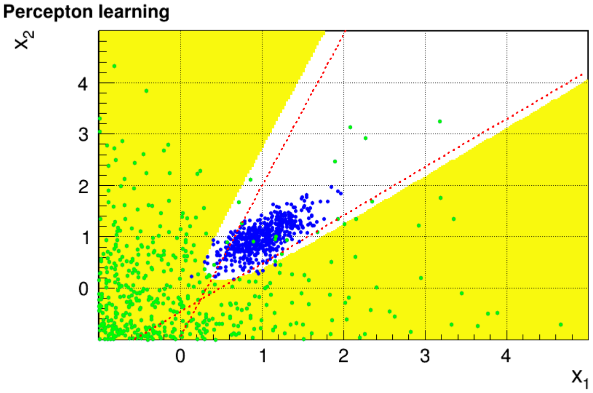
$$\mathbf{w}^{(2)(n+1)}_j \;=\; \mathbf{w}^{(2)(n)}_j \;-\; \eta \sum_i \delta^{(2)}_{j,i} \cdot \mathbf{x}_i$$

Iterative procedure, starting from random weights:

- calculate y_i for train sample events $\Rightarrow$ extract $\delta^{(1)}_i$ and $\delta^{(2)}_{j,i}$
- update $\mathbf{w}^{(1)}$ and $\mathbf{w}^{(2)}_j$, decrease η, repeat from the beginning

Simplest case

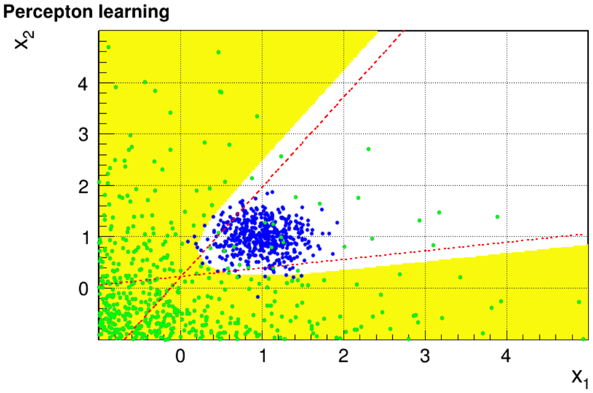
Simplest network: one hidden layer with two preceptons...
Visible improvement in efficiency and flexibility of classification!



Correlation for signal sample $\rho = 0.7$

Simplest case

Simplest network: one hidden layer with two preceptons...
Visible improvement in efficiency and flexibility of classification!

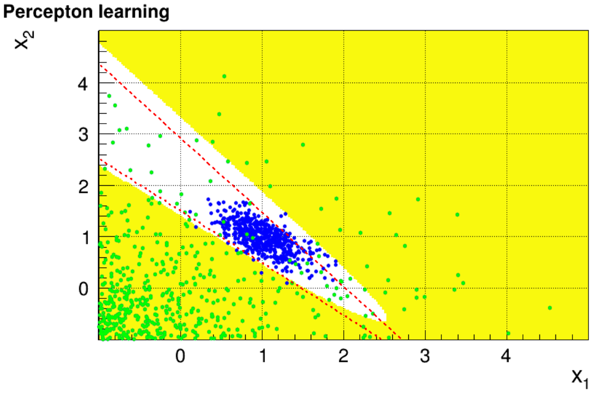


Correlation for signal sample $\rho = 0$

Artificial Neural Networks

Simplest case

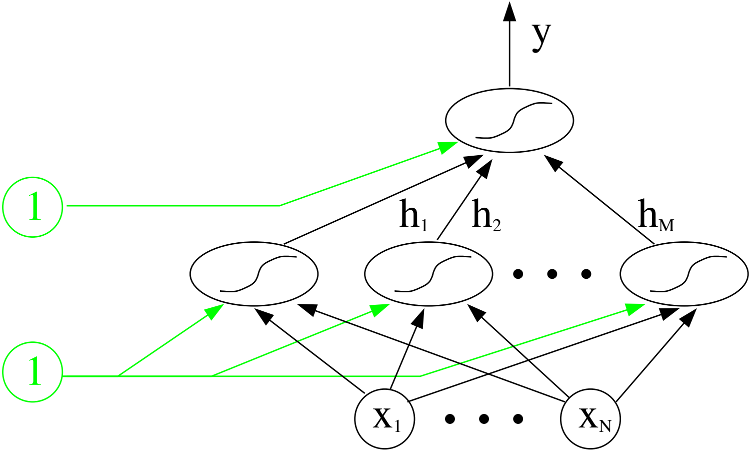
Simplest network: one hidden layer with two preceptons...
Visible improvement in efficiency and flexibility of classification!



Correlation for signal sample $\rho = -0.7$

More complex case

We can have arbitrary number of neurons in hidden layer...

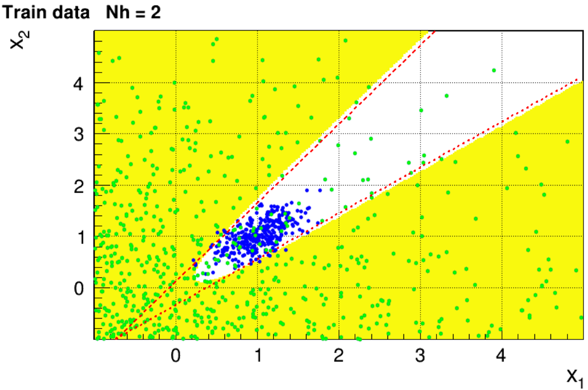


as well as more input variables...

Artificial Neural Networks

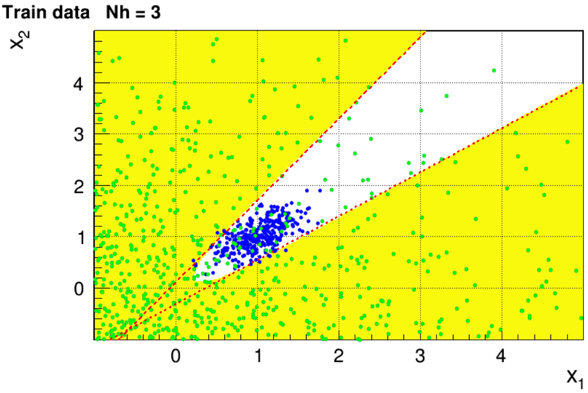
More complex case

Classification improves with the number of nodes in the hidden layer.
Learning takes a little bit longer, but we can gain a lot...



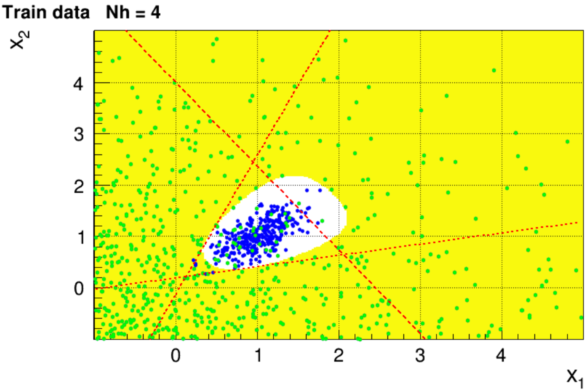
More complex case

Classification improves with the number of nodes in the hidden layer.
Learning takes a little bit longer, but we can gain a lot...



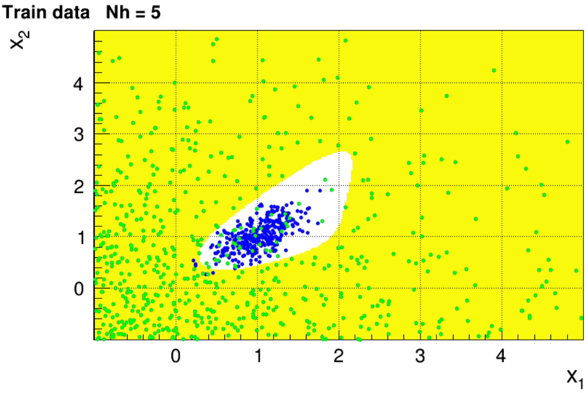
More complex case

Classification improves with the number of nodes in the hidden layer.
Learning takes a little bit longer, but we can gain a lot...



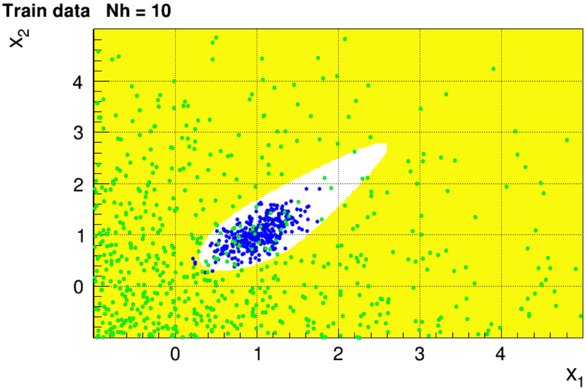
More complex case

Classification improves with the number of nodes in the hidden layer.
Learning takes a little bit longer, but we can gain a lot...



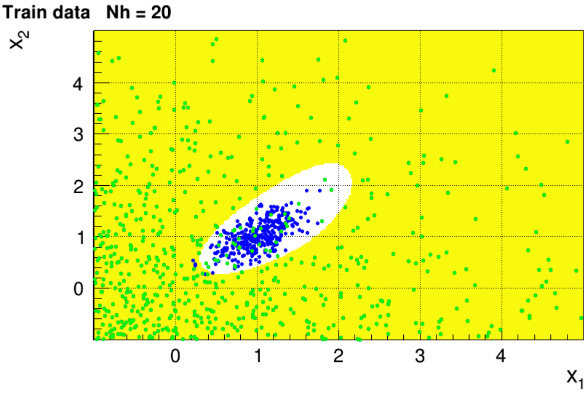
More complex case

Classification improves with the number of nodes in the hidden layer.
Learning takes a little bit longer, but we can gain a lot...



More complex case

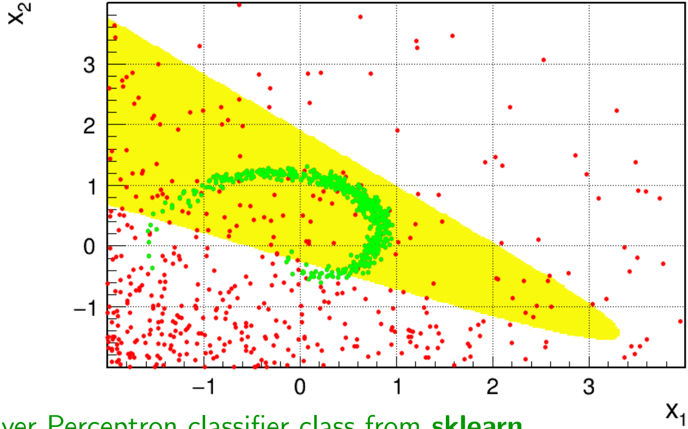
Classification improves with the number of nodes in the hidden layer.
Learning takes a little bit longer, but we can gain a lot...



Multilayer example

One hidden layer only - limited shape flexibility

Neural network from sklearn, hidden neurons: 2, test sample

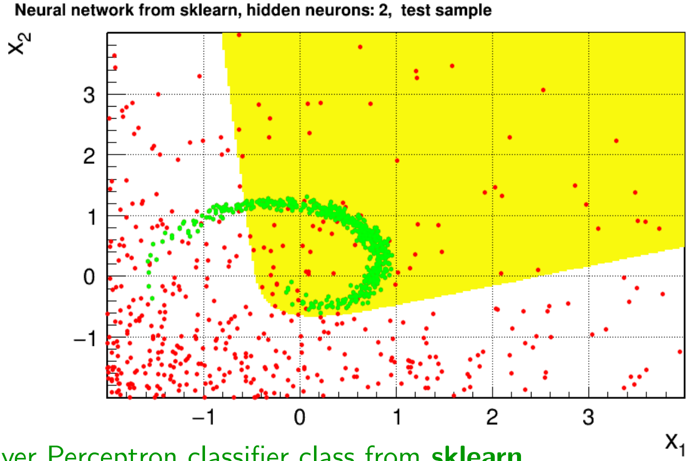


Multi-layer Perceptron classifier class from **sklearn**

Artificial Neural Networks

Multilayer example

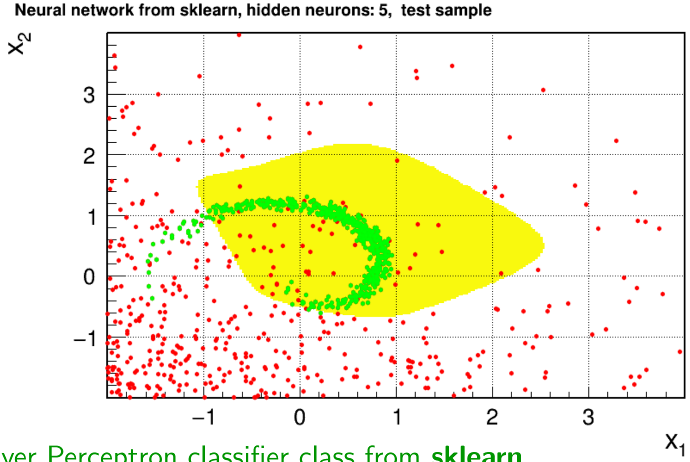
One hidden layer only - limited shape flexibility



Multi-layer Perceptron classifier class from **sklearn**

Multilayer example

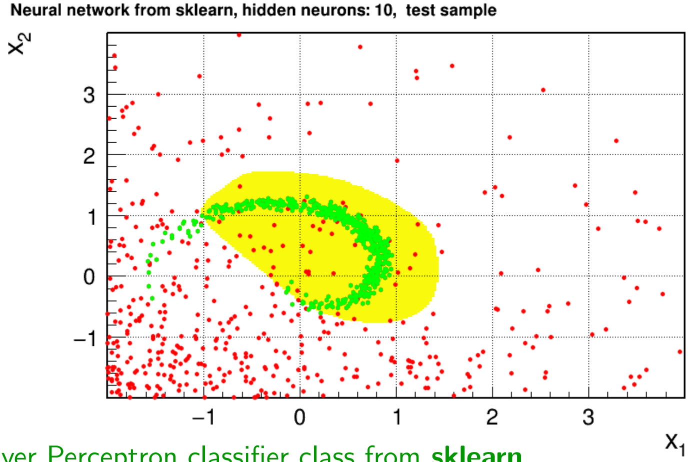
One hidden layer only - limited shape flexibility



Multi-layer Perceptron classifier class from **sklearn**

Multilayer example

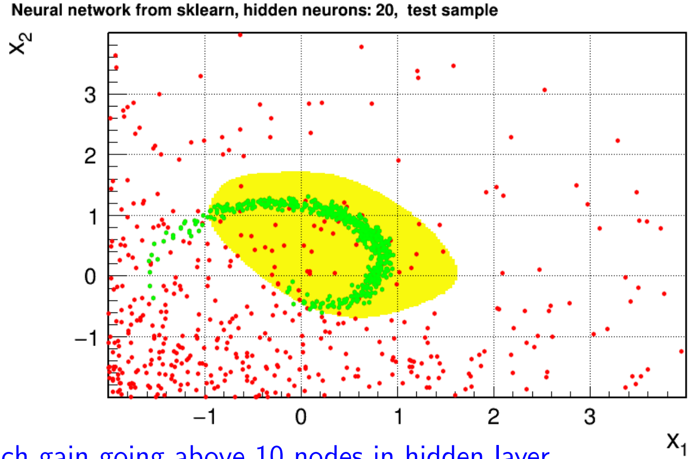
One hidden layer only - limited shape flexibility



Multi-layer Perceptron classifier class from **sklearn**

Multilayer example

One hidden layer only - limited shape flexibility

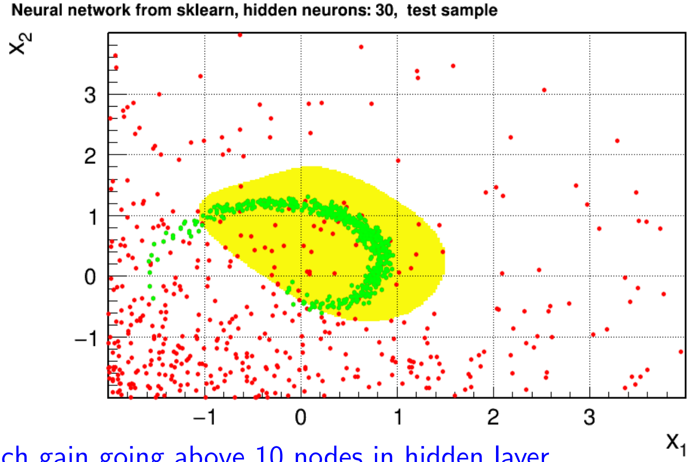


Not much gain going above 10 nodes in hidden layer...

Artificial Neural Networks

Multilayer example

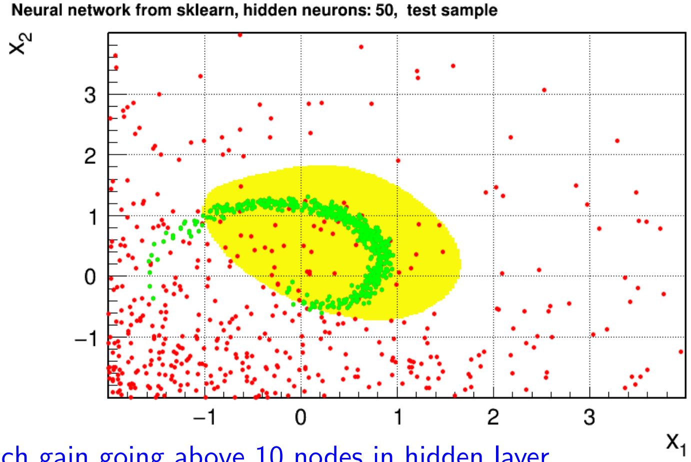
One hidden layer only - limited shape flexibility



Not much gain going above 10 nodes in hidden layer...

Multilayer example

One hidden layer only - limited shape flexibility

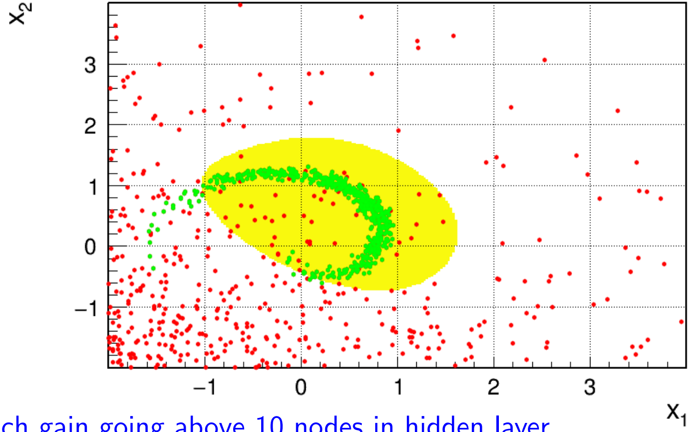


Not much gain going above 10 nodes in hidden layer...

Multilayer example

One hidden layer only - limited shape flexibility

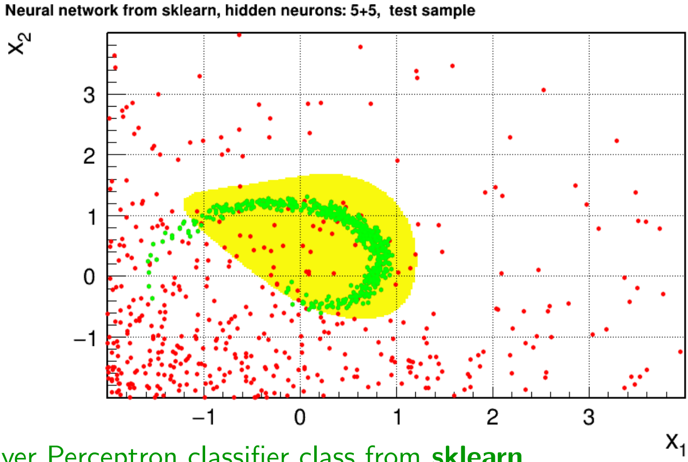
Neural network from sklearn, hidden neurons: 100, test sample



Not much gain going above 10 nodes in hidden layer...

Multilayer example

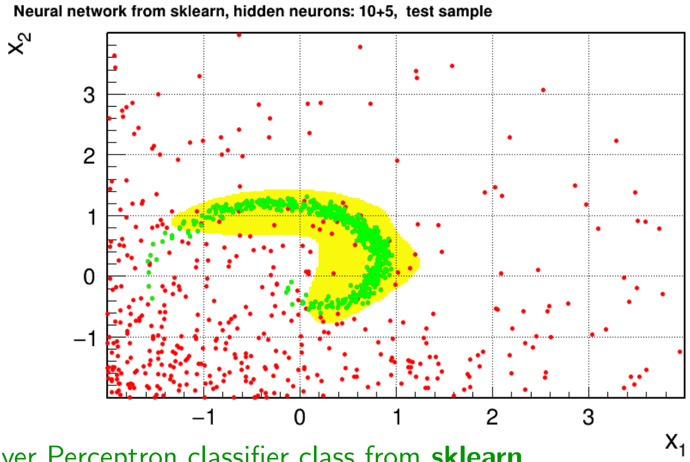
Two hidden layer - more shape flexibility



Multi-layer Perceptron classifier class from **sklearn**

Multilayer example

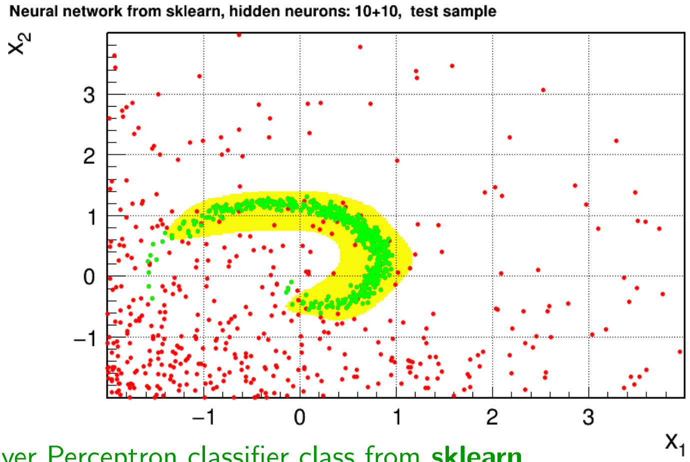
Two hidden layer - more shape flexibility



Multi-layer Perceptron classifier class from **sklearn**

Multilayer example

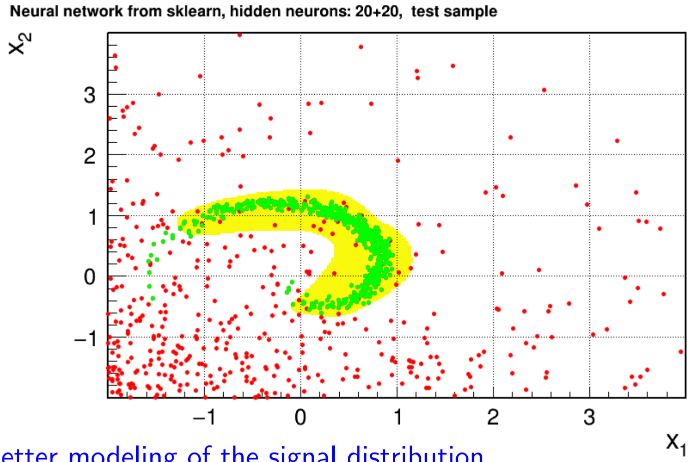
Two hidden layer - more shape flexibility



Multi-layer Perceptron classifier class from **sklearn**

Multilayer example

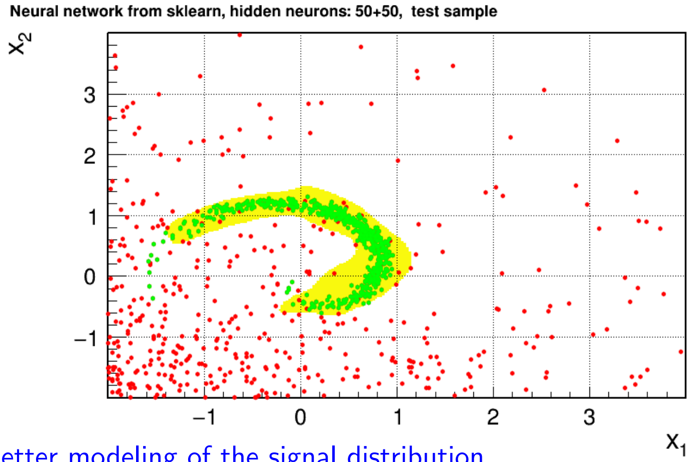
Two hidden layer - more shape flexibility



Much better modeling of the signal distribution...

Multilayer example

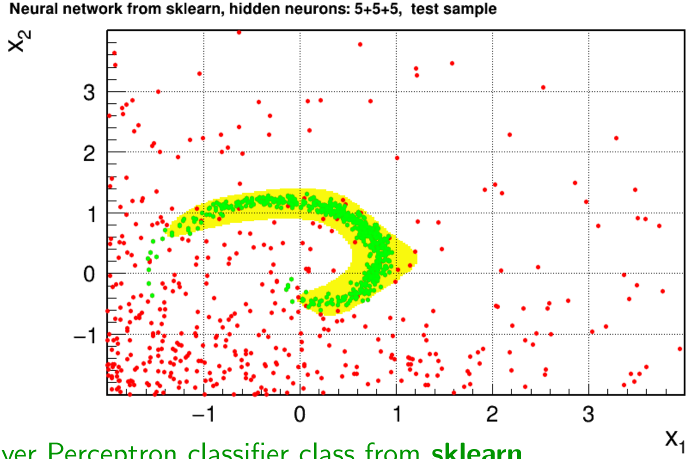
Two hidden layer - more shape flexibility



Much better modeling of the signal distribution...

Multilayer example

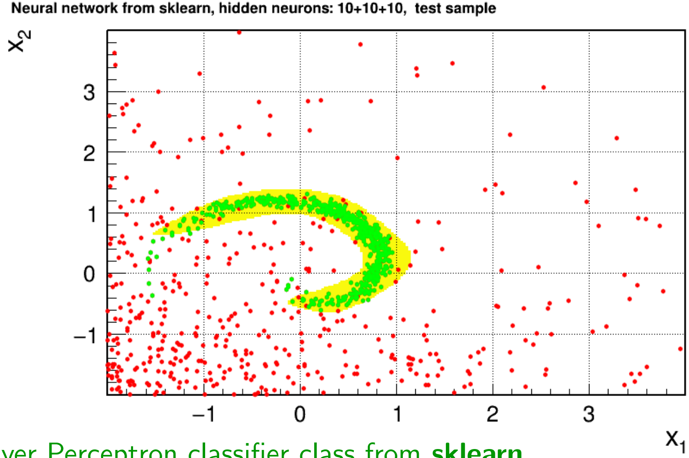
Three hidden layer - more details can be included



Multi-layer Perceptron classifier class from **sklearn**

Multilayer example

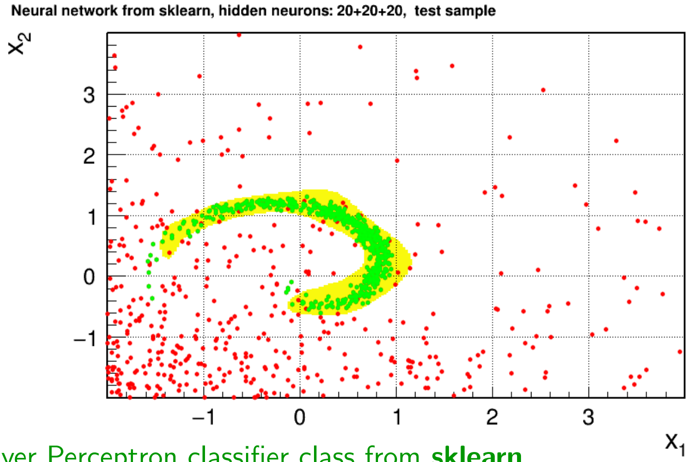
Three hidden layer - more details can be included



Multi-layer Perceptron classifier class from **sklearn**

Multilayer example

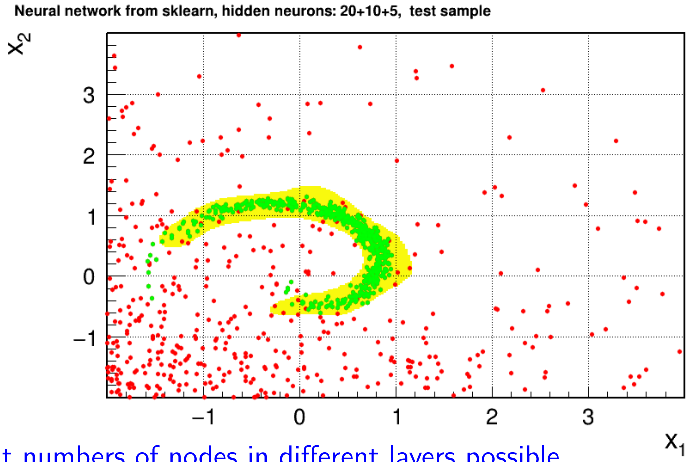
Three hidden layer - more details can be included



Multi-layer Perceptron classifier class from **sklearn**

Multilayer example

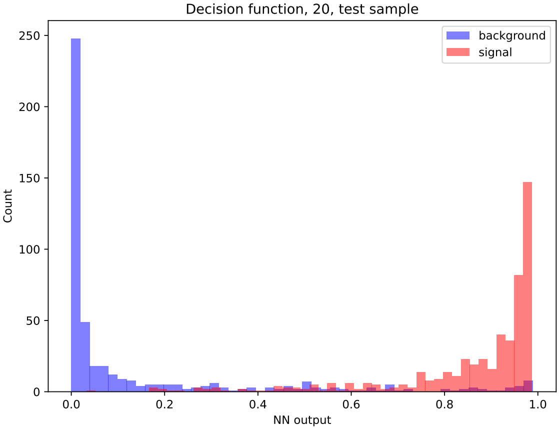
Three hidden layer - more details can be included



Different numbers of nodes in different layers possible...

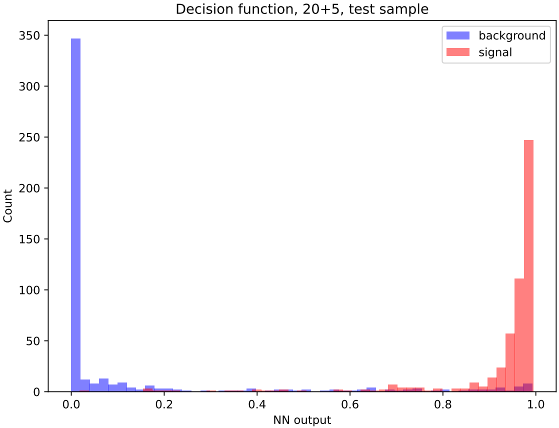
Comparison of the output discriminator function distribution

Single hidden layer with 20 neurons



Comparison of the output discriminator function distribution

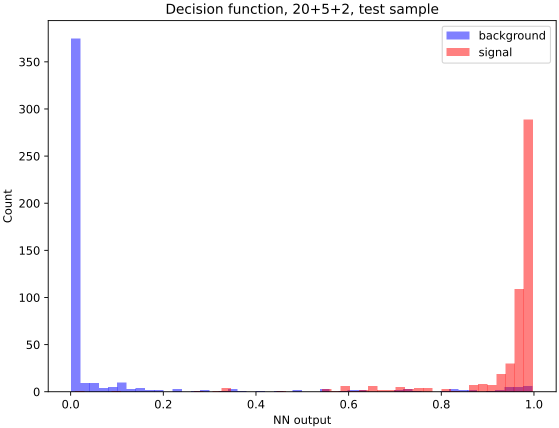
Two hidden layers, with 20 and 5 neurons



Comparison of the output discriminator function distribution

Three hidden layers, with 20, 5 and 2 neurons

Clear improvement of the event classification



sklearn tips...

https://scikit-learn.org/

Multi-layer perceptron is sensitive to variable scales.

It is highly recommended to scale input data, so each variable has the same range (eg. $[-1, +1]$) or same mean and variance (eg. $\mu = 0$ and $\sigma = 1$). Both training and test samples need to be scaled in the same way!

Artificial Neural Networks

sklearn tips...

https://scikit-learn.org/

Multi-layer perceptron is sensitive to variable scales.

It is highly recommended to scale input data, so each variable has the same range (eg. $[-1, +1]$) or same mean and variance (eg. $\mu = 0$ and $\sigma = 1$). Both training and test samples need to be scaled in the same way!

Different, more advanced learning algorithms are implemented in **sklearn**, one can choose between them with 'solver' parameter.

- 'lbfgs' converges faster and with better solutions on small datasets.
- For relatively large datasets, 'adam' is very robust. It usually converges quickly and gives pretty good performance.
- 'sgd' can perform best if learning rate is correctly tuned.

Machine Learning

1. Artificial Neural Networks
2. Boosting
3. Decision Trees
4. Boosted Decision Trees
5. Homework

Boosting

Ensemble methods

It is relatively easy (in most cases) to design a classification algorithm which will result in the classification efficiency (fraction of correct classifications) slightly above 50% (random classification level).

Such classifiers are called "weak classifiers"

Boosting

Ensemble methods

It is relatively easy (in most cases) to design a classification algorithm which will result in the classification efficiency (fraction of correct classifications) slightly above 50% (random classification level).

Such classifiers are called "weak classifiers"

It is much more difficult (in most realistic cases) to design a single classifier, which will result in efficiency close to 100% (error-less classification).

Such classifiers are called "strong classifiers"

Boosting

Ensemble methods

It is relatively easy (in most cases) to design a classification algorithm which will result in the classification efficiency (fraction of correct classifications) slightly above 50% (random classification level).

Such classifiers are called "weak classifiers"

It is much more difficult (in most realistic cases) to design a single classifier, which will result in efficiency close to 100% (error-less classification).

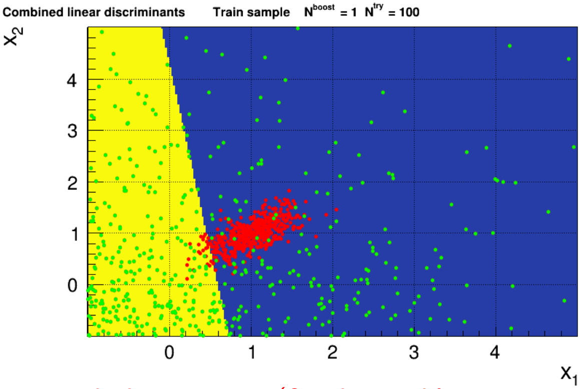
Such classifiers are called "strong classifiers"

However, it turns out that one can build a strong classifier from many weak classifiers!

This is the underlying principle in many machine learning techniques...

Example weak discriminant

Generate $N_{try} = 100$ random linear discriminants. Select the one with the highest efficiency (highest number of properly classified events).



This is clearly a weak discriminant (for this problem; $\varepsilon \sim 50 - 60\%$)

Ensemble methods

https://scikit-learn.org/

Two families of ensemble methods are usually distinguished:

- In averaging methods, the driving principle is to build several estimators independently and then to average their predictions. On average, the combined estimator is usually better than any of the single base estimator because its variance is reduced.

Boosting

Ensemble methods

https://scikit-learn.org/

Two families of ensemble methods are usually distinguished:

- In averaging methods, the driving principle is to build several estimators independently and then to average their predictions. On average, the combined estimator is usually better than any of the single base estimator because its variance is reduced.
- In boosting methods, base estimators are built sequentially and one tries to reduce the bias of the combined estimator. The motivation is to combine several weak models to produce a powerful ensemble.

Boosting

Ensemble methods

https://scikit-learn.org/

Two families of ensemble methods are usually distinguished:

- In averaging methods, the driving principle is to build several estimators independently and then to average their predictions. On average, the combined estimator is usually better than any of the single base estimator because its variance is reduced.
- In boosting methods, base estimators are built sequentially and one tries to reduce the bias of the combined estimator. The motivation is to combine several weak models to produce a powerful ensemble.

The two methods can also be combined...

Boosting

Procedure

Let as assume that we have a sample of events $\mathbf{x}_i$ with true categories t_i.
All events have the same initial weight $w_i^{(1)} = 1/N$

Boosting

Procedure

Let as assume that we have a sample of events $\mathbf{x}_i$ with true categories t_i.
All events have the same initial weight $w_i^{(1)} = 1/N$

The iterative procedure looks like follows. In step j:

1. train classifier C_j using our input data $\mathbf{x}_i$ with weights $w_i^{(j)}$

Boosting

Procedure

Let as assume that we have a sample of events $\mathbf{x}_i$ with true categories t_i.
All events have the same initial weight $w_i^{(1)} = 1/N$

The iterative procedure looks like follows. In step j:

1. train classifier C_j using our input data $\mathbf{x}_i$ with weights $w_i^{(j)}$
2. calculate classifier response: $y_i^{(j)} = C_j(\mathbf{x}_i)$

Boosting

Procedure

Let as assume that we have a sample of events $\mathbf{x}_i$ with true categories t_i.
All events have the same initial weight $w_i^{(1)} = 1/N$

The iterative procedure looks like follows. In step j:

1. train classifier C_j using our input data $\mathbf{x}_i$ with weights $w_i^{(j)}$
2. calculate classifier response: $y_i^{(j)} = C_j(\mathbf{x}_i)$
3. calculate classifier error rate: $\varepsilon_j = \sum w_i^{(j)} \cdot (y_i^{(j)} \; != t_i) \; / \sum w_i^{(j)}$

Boosting

Procedure

Let as assume that we have a sample of events $\mathbf{x}_i$ with true categories t_i.
All events have the same initial weight $w_i^{(1)} = 1/N$

The iterative procedure looks like follows. In step j:

1. train classifier C_j using our input data $\mathbf{x}_i$ with weights $w_i^{(j)}$
2. calculate classifier response: $y_i^{(j)} = C_j(\mathbf{x}_i)$
3. calculate classifier error rate: $\varepsilon_j = \sum w_i^{(j)} \cdot (y_i^{(j)} \, != t_i) \, / \sum w_i^{(j)}$
4. calculate classifier weight: $a_j = \log\left(\frac{1-\varepsilon_j}{\varepsilon_j}\right)$

Procedure

Let as assume that we have a sample of events $\mathbf{x}_i$ with true categories t_i.
All events have the same initial weight $w_i^{(1)} = 1/N$

The iterative procedure looks like follows. In step j:

1. train classifier C_j using our input data $\mathbf{x}_i$ with weights $w_i^{(j)}$
2. calculate classifier response: $y_i^{(j)} = C_j(\mathbf{x}_i)$
3. calculate classifier error rate: $\varepsilon_j = \sum w_i^{(j)} \cdot (y_i^{(j)} \, != t_i) \, / \sum w_i^{(j)}$
4. calculate classifier weight: $a_j = \log \left(\frac{1-\varepsilon_j}{\varepsilon_j} \right)$
5. modify event weights:

$$\begin{aligned} w_i^{(j+1)} &= w_i^{(j)} \cdot \exp(a_j) & \text{for } y_i^{(j)} \neq t_i, \\ w_i^{(j+1)} &= w_i^{(j)} & \text{for } y_i^{(j)} = t_i. \end{aligned}$$

Scale all weights to get $\sum w_i^{(j+1)} = 1$

Boosting

Procedure

Let as assume that we have a sample of events $\mathbf{x}_i$ with true categories t_i.
All events have the same initial weight $w_i^{(1)} = 1/N$

The iterative procedure looks like follows. In step j:

1. train classifier C_j using our input data $\mathbf{x}_i$ with weights $w_i^{(j)}$
2. calculate classifier response: $y_i^{(j)} = C_j(\mathbf{x}_i)$
3. calculate classifier error rate: $\varepsilon_j = \sum w_i^{(j)} \cdot (y_i^{(j)} \, != t_i) \, / \sum w_i^{(j)}$
4. calculate classifier weight: $a_j = \log\left(\frac{1-\varepsilon_j}{\varepsilon_j}\right)$
5. modify event weights:

   or $\quad w_i^{(j+1)} = w_i^{(j)} \cdot \exp(-\alpha \, y_i^{(j)} \, t_i \, a_j)$.

Scale all weights to get $\sum w_i^{(j+1)} = 1$

Boosting

Procedure

(Behnke)

By reweighting events, we force subsequent classifiers to focus on events (i.e. value ranges) where classification was poor.

New classifiers are still “weak”,
but they properly classify different classes of events.

We get a sequence of classifiers focusing on different variable regions.

Boosting

Procedure

(Behnke)

By reweighting events, we force subsequent classifiers to focus on events (i.e. value ranges) where classification was poor.

New classifiers are still "weak",
but they properly classify different classes of events.

We get a sequence of classifiers focusing on different variable regions.

We can get much stronger classifier by combining their outputs

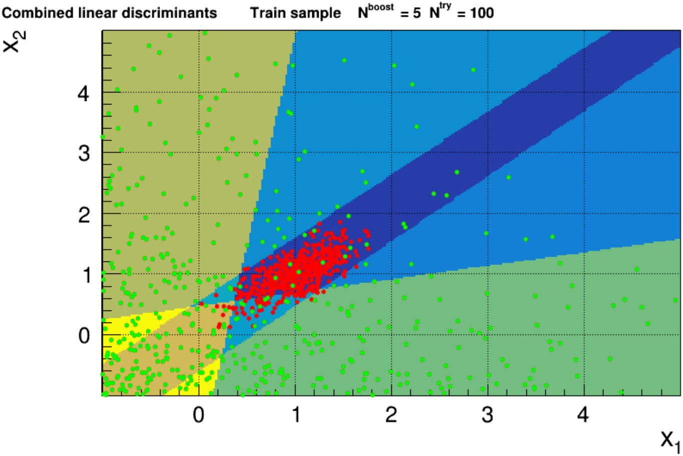
$$C_{Boost}(\mathbf{x}) \quad = \quad \frac{1}{M} \sum_j a_j \; C_j(\mathbf{x})$$

where M is the total number of classifiers in the collection.

This procedure is referred to as "adaptive boost" (AdaBoost)

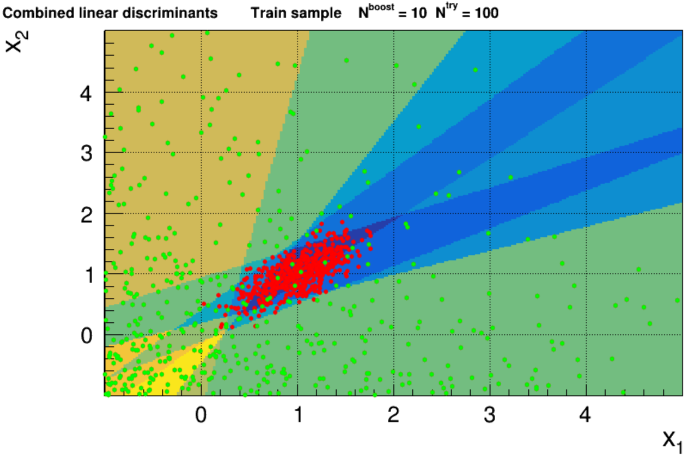
Classifier boosting

Example of weak classifier (linear discriminant) boosting



Classifier boosting

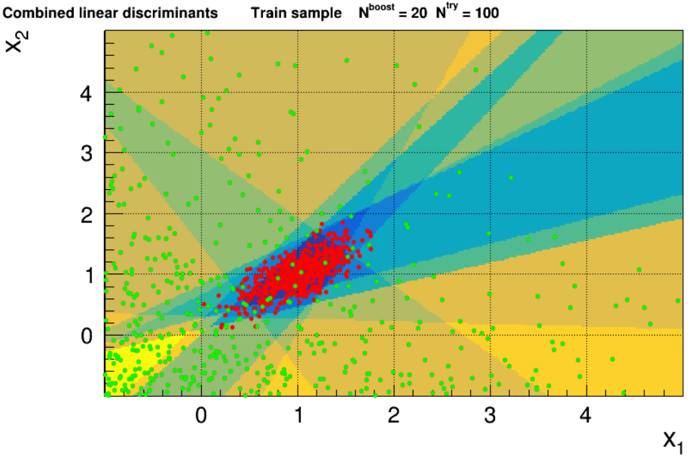
Example of weak classifier (linear discriminant) boosting



Boosting

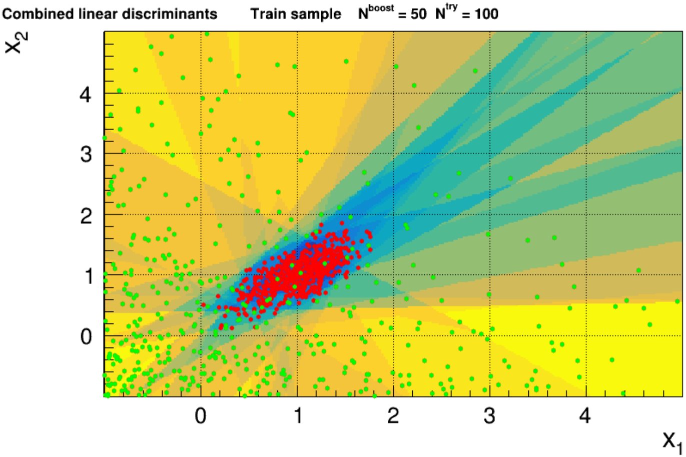
Classifier boosting

Example of weak classifier (linear discriminant) boosting



Classifier boosting

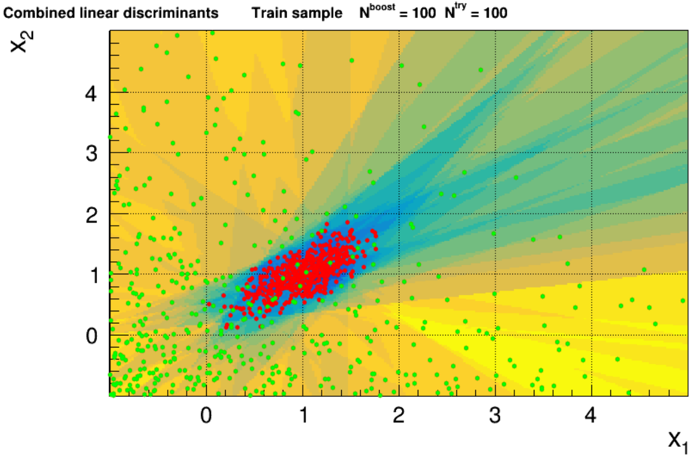
Example of weak classifier (linear discriminant) boosting



Boosting

Classifier boosting

Example of weak classifier (linear discriminant) boosting

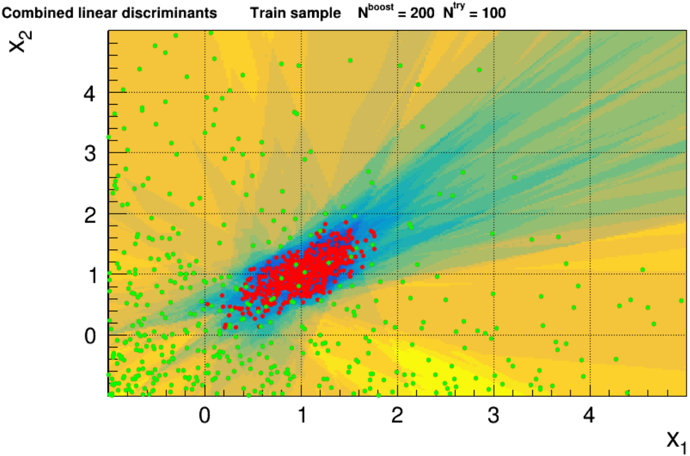


Final performance improves significantly ⇒ “strong classifier” obtained

Boosting

Classifier boosting

Example of weak classifier (linear discriminant) boosting

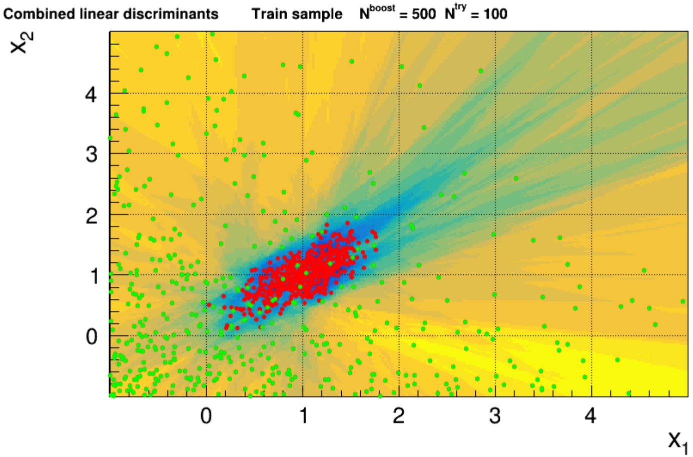


Final performance improves significantly ⇒ “strong classifier” obtained

Boosting

Classifier boosting

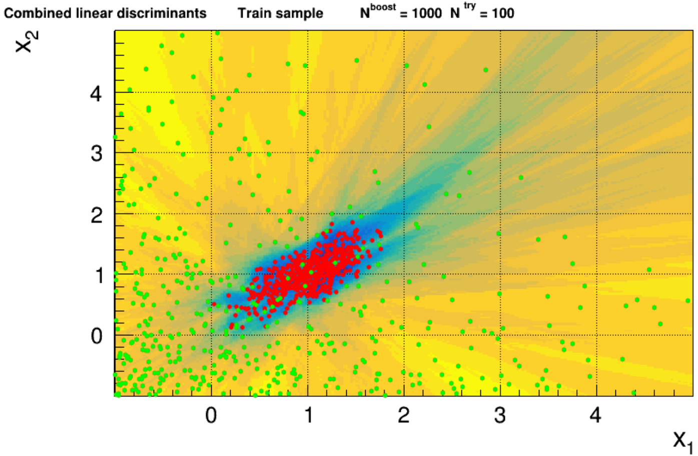
Example of weak classifier (linear discriminant) boosting



Final performance improves significantly ⇒ "strong classifier" obtained

Classifier boosting

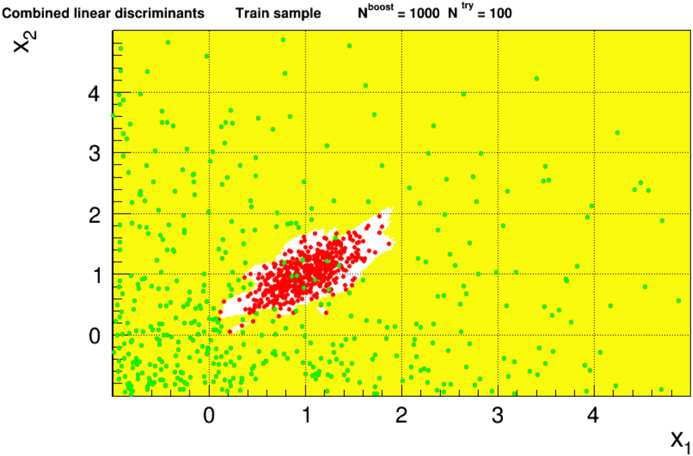
Example of weak classifier (linear discriminant) boosting



Final performance improves significantly ⇒ “strong classifier” obtained

Classifier boosting

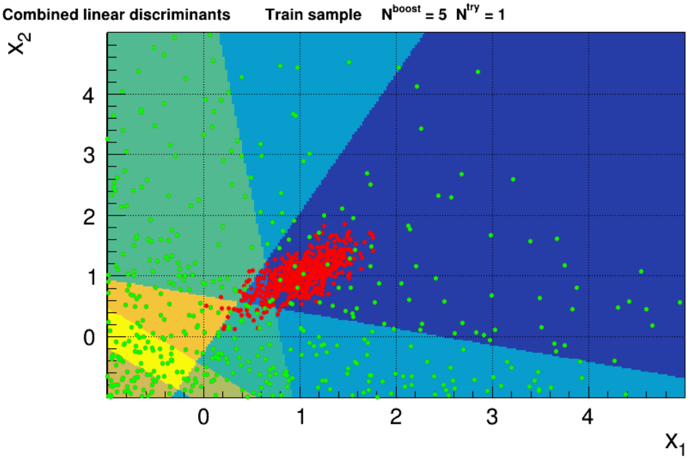
Example of weak classifier (linear discriminant) boosting



Final performance improves significantly ⇒ “strong classifier” obtained

Classifier boosting

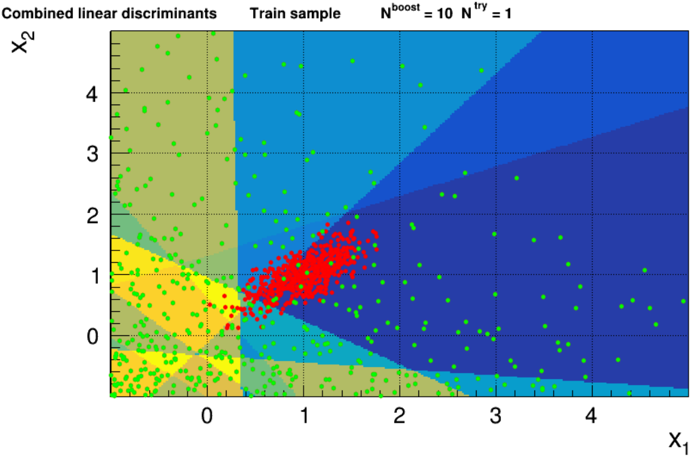
Surprisingly, the procedure works also for the completely random (not optimized in any way) classifiers used as “building blocks”



Boosting

Classifier boosting

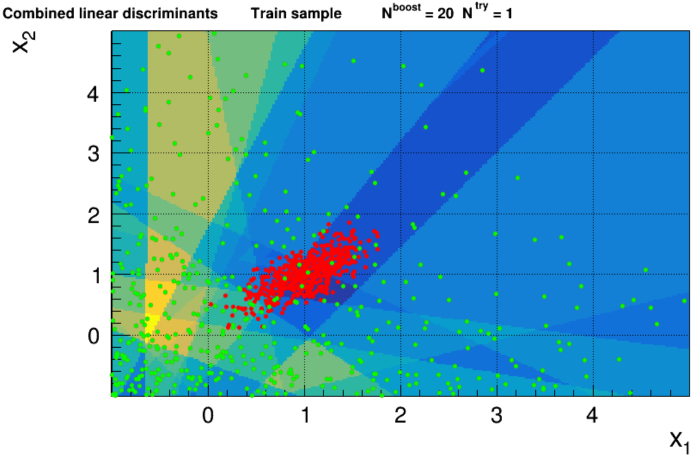
Surprisingly, the procedure works also for the completely random (not optimized in any way) classifiers used as "building blocks"



Boosting

Classifier boosting

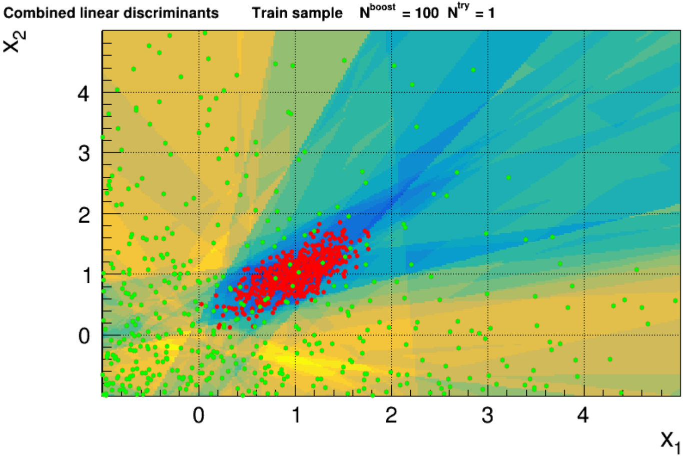
Surprisingly, the procedure works also for the completely random (not optimized in any way) classifiers used as "building blocks"



Boosting

Classifier boosting

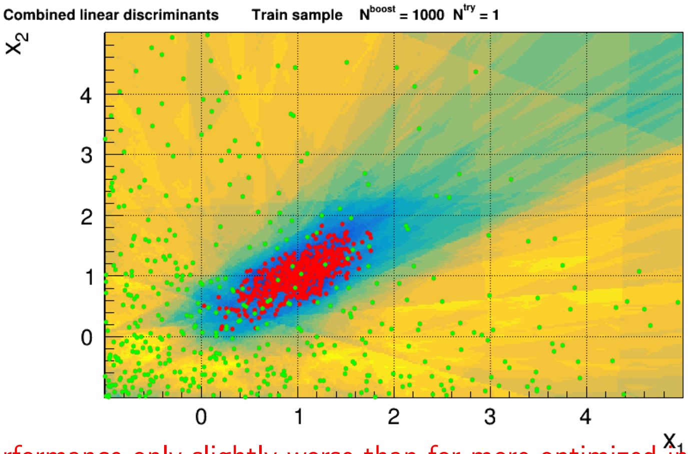
Surprisingly, the procedure works also for the completely random (not optimized in any way) classifiers used as "building blocks"



Boosting

Classifier boosting

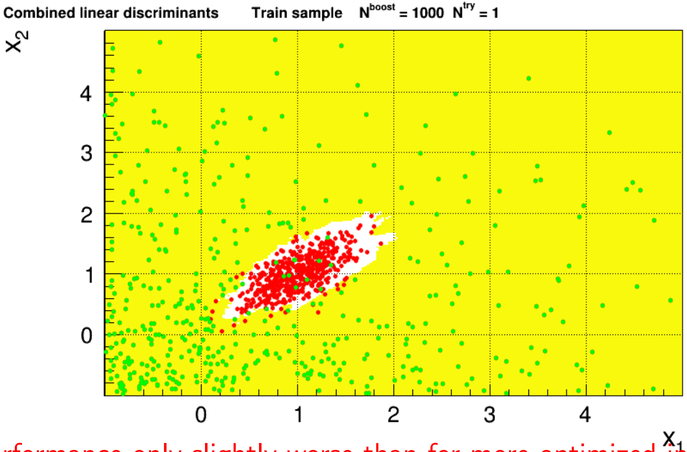
Surprisingly, the procedure works also for the completely random (not optimized in any way) classifiers used as "building blocks"



Final performance only slightly worse than for more optimized input...

Classifier boosting

Surprisingly, the procedure works also for the completely random (not optimized in any way) classifiers used as "building blocks"

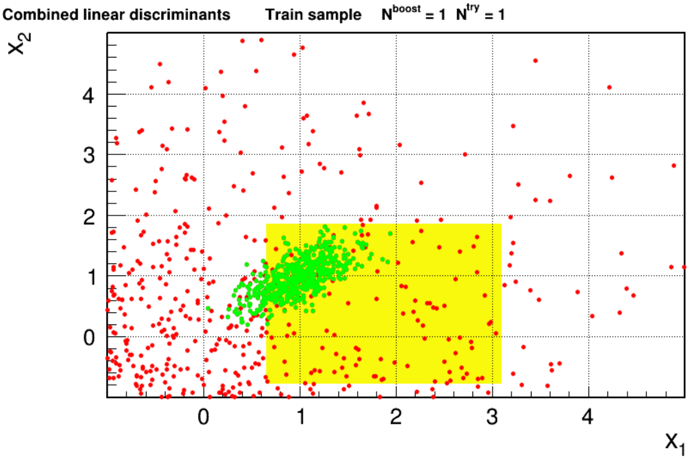


Final performance only slightly worse than for more optimized input...

Boosting

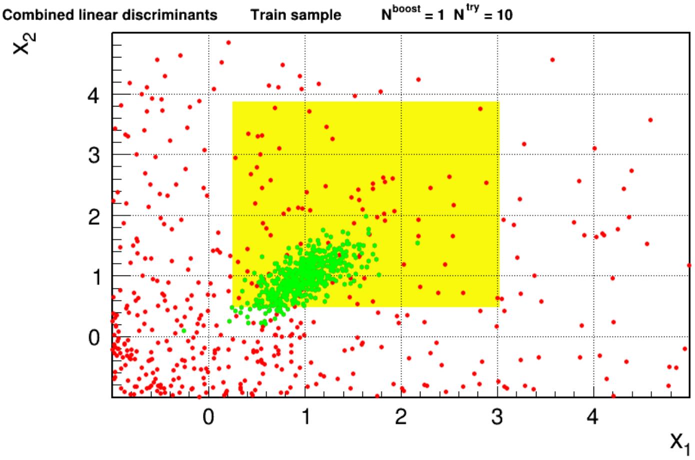
Box cut classifier

Random box cut based on two random points in the parameter space:



Box cut classifier

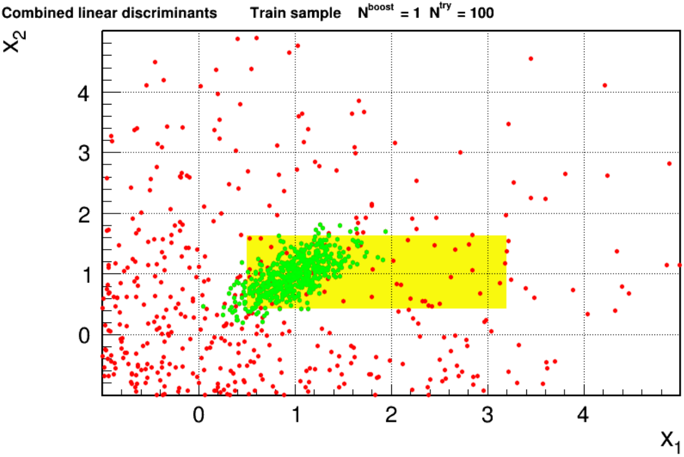
Box cut with highest efficiency selected out of 10 random box cuts



Boosting

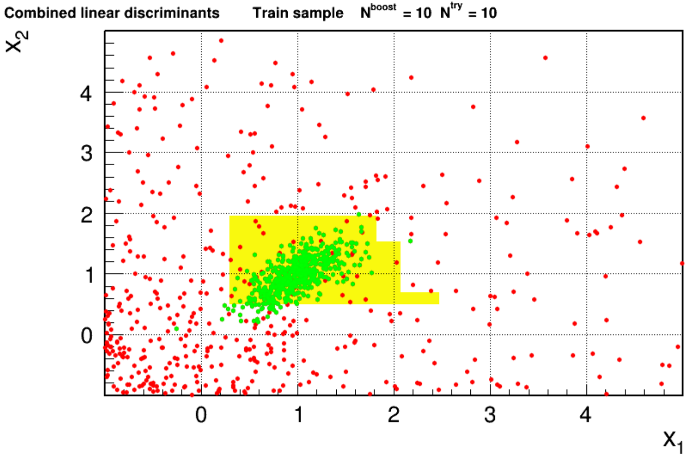
Box cut classifier

Box cut with highest efficiency selected out of 100 random box cuts



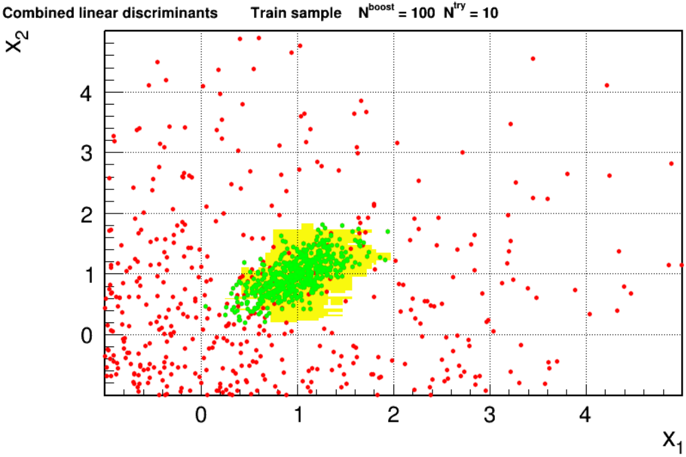
Box classifier boosting

Example of weak classifier (best box cut out of 10 random) boosting



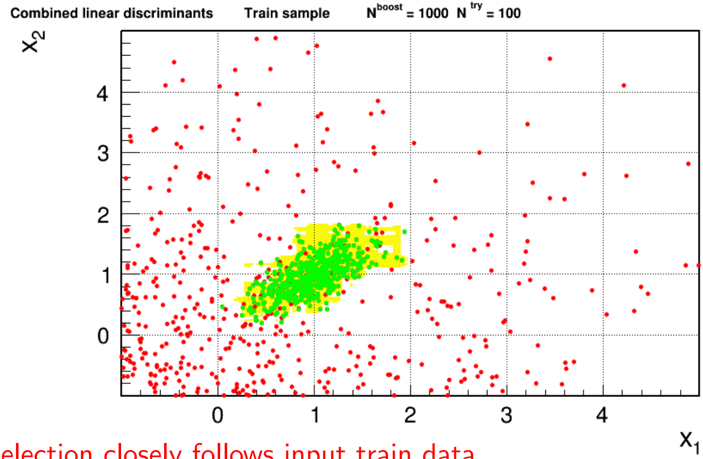
Box classifier boosting

Example of weak classifier (best box cut out of 10 random) boosting



Box classifier boosting

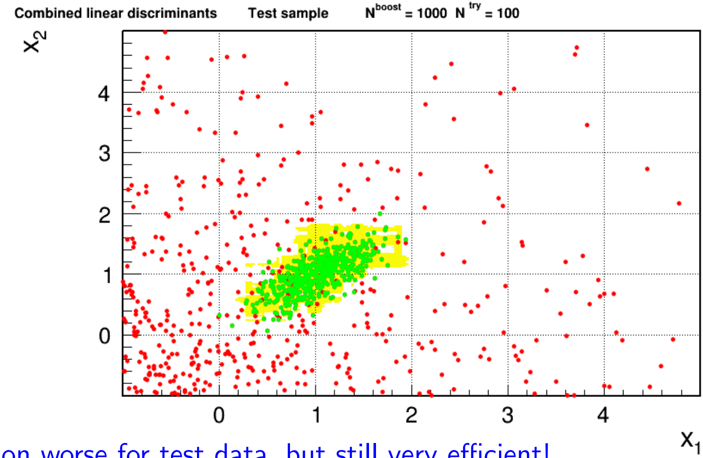
Example of weak classifier (best box cut out of 100 random) boosting



Final selection closely follows input train data...

Box classifier boosting

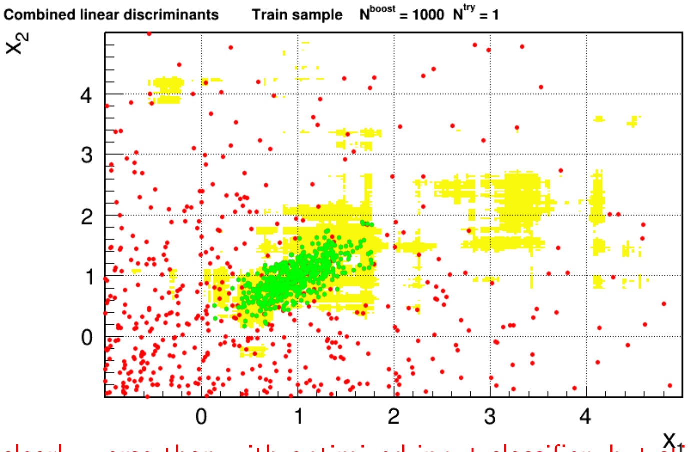
Example of weak classifier (best box cut out of 100 random) boosting



Selection worse for test data, but still very efficient!

Box classifier boosting

Event random box cut (without selection) can get boosted

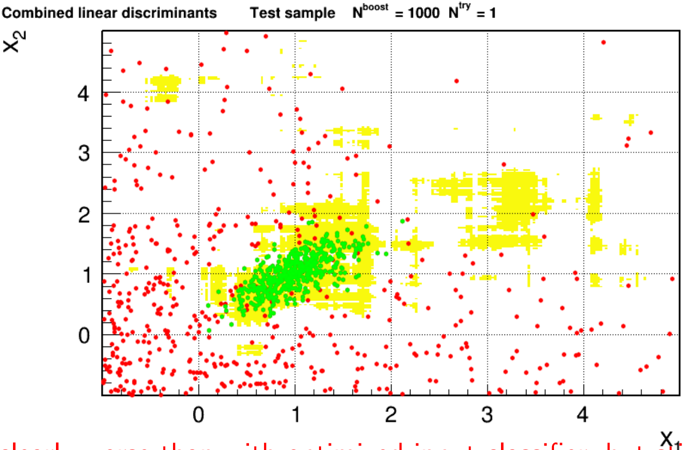


Results clearly worse than with optimized input classifier, but still useful...

Boosting

Box classifier boosting

Event random box cut (without selection) can get boosted



Results clearly worse than with optimized input classifier, but still useful...

Machine Learning

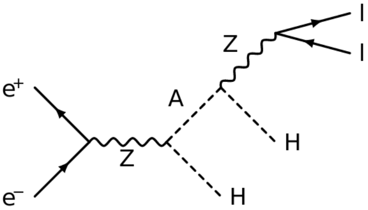
1. Artificial Neural Networks
2. Boosting
3. Decision Trees
4. Boosted Decision Trees
5. Homework

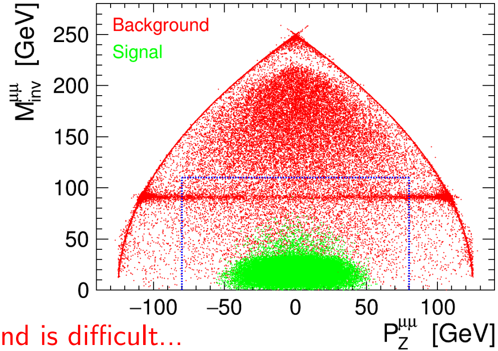
Decision Trees

Principle

It is quite a common approach in data selection to apply cuts on variables considered in the analysis.
We can profit from our understanding of the processes considered...

IDM scalar pair-production
with di-lepton signature



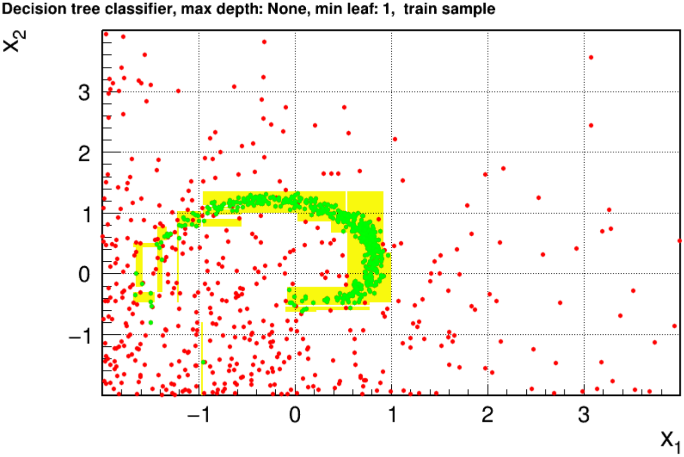


However, tuning the cuts by hand is difficult...

Decision Trees

Example

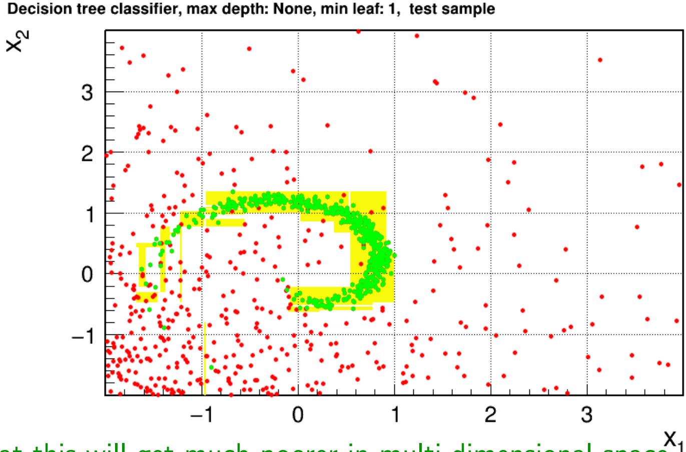
We can write down the cuts that will perfectly classify our training sample:



Decision Trees

Example

But on test sample results will be worse!   Efficiency ∼93%

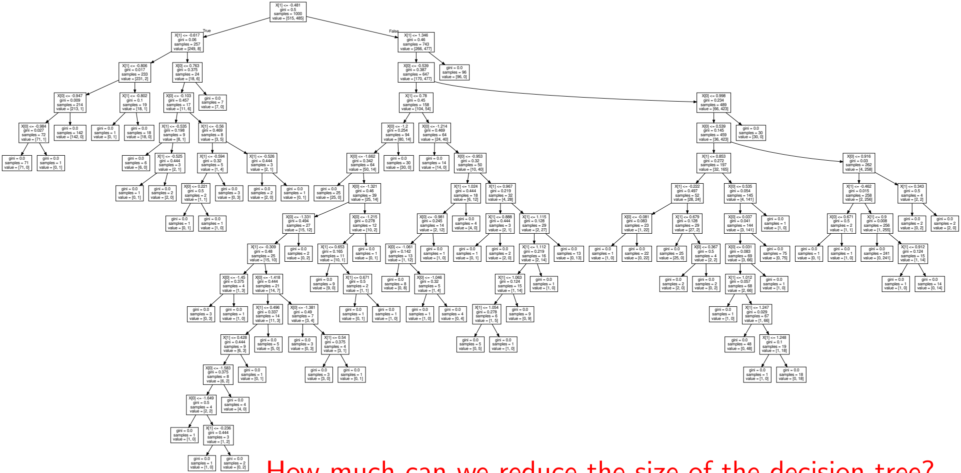


Note that this will get much poorer in multi-dimensional space...

Decision Trees

Example

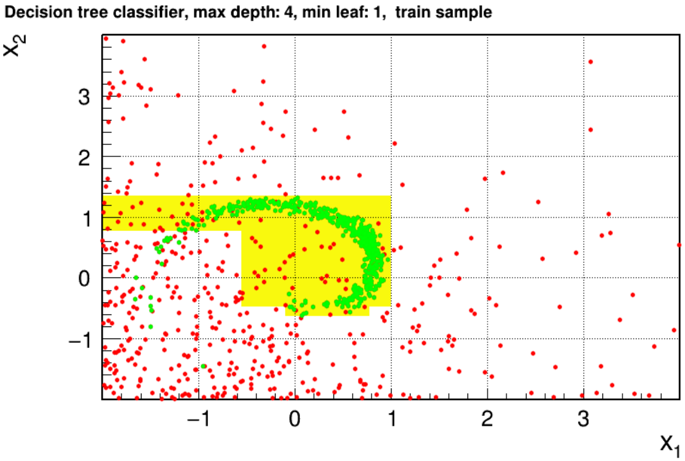
The tree for full sample classification very complicated already in 2-D...


X[1] <= -0.481 gini = 0.5 samples = 1000 value = [515, 485]
True
X[1] <= -0.617 gini = 0.06 samples = 257 value = [249, 8]
False
X[1] <= 1.346 gini = 0.46 samples = 743 value = [266, 477]
X[1] <= -0.806 gini = 0.017 samples = 233 value = [231, 2]
X[0] <= 0.763 gini = 0.375 samples = 24 value = [18, 6]
X[0] <= -0.539 gini = 0.387 samples = 647 value = [170, 477]
gini = 0.0 samples = 96 value = [96, 0]
X[0] <= -0.947 gini = 0.009 samples = 214 value = [213, 1]
X[1] <= -0.802 gini = 0.1 samples = 19 value = [18, 1]
X[0] <= -0.103 gini = 0.457 samples = 17 value = [11, 6]
gini = 0.0 samples = 7 value = [7, 0]
X[1] <= 0.78 gini = 0.45 samples = 158 value = [104, 54]
X[0] <= 0.998 gini = 0.234 samples = 489 value = [66, 423]
X[0] <= -0.984 gini = 0.027 samples = 72 value = [71, 1]
gini = 0.0 samples = 142 value = [142, 0]
gini = 0.0 samples = 1 value = [0, 1]
gini = 0.0 samples = 18 value = [18, 0]
X[1] <= -0.535 gini = 0.198 samples = 9 value = [8, 1]
X[1] <= -0.56 gini = 0.469 samples = 8 value = [3, 5]
X[0] <= -1.2 gini = 0.254 samples = 94 value = [80, 14]
X[0] <= -1.214 gini = 0.469 samples = 64 value = [24, 40]
X[0] <= 0.539 gini = 0.145 samples = 459 value = [36, 423]
gini = 0.0 samples = 30 value = [30, 0]
gini = 0.0 samples = 71 value = [71, 0]
gini = 0.0 samples = 1 value = [0, 1]
gini = 0.0 samples = 6 value = [6, 0]
X[1] <= -0.525 gini = 0.444 samples = 3 value = [2, 1]
X[1] <= -0.594 gini = 0.32 samples = 5 value = [1, 4]
X[1] <= -0.526 gini = 0.444 samples = 3 value = [2, 1]
X[0] <= -1.662 gini = 0.342 samples = 64 value = [50, 14]
gini = 0.0 samples = 30 value = [30, 0]
gini = 0.0 samples = 14 value = [14, 0]
X[0] <= -0.953 gini = 0.32 samples = 50 value = [10, 40]
X[1] <= 0.853 gini = 0.272 samples = 197 value = [32, 165]
X[0] <= 0.916 gini = 0.03 samples = 262 value = [4, 258]
gini = 0.0 samples = 1 value = [0, 1]
gini = 0.0 samples = 2 value = [2, 0]
X[0] <= 0.221 gini = 0.5 samples = 2 value = [1, 1]
gini = 0.0 samples = 3 value = [0, 3]
gini = 0.0 samples = 2 value = [2, 0]
gini = 0.0 samples = 1 value = [0, 1]
gini = 0.0 samples = 25 value = [25, 0]
X[0] <= -1.321 gini = 0.46 samples = 39 value = [25, 14]
X[1] <= 1.024 gini = 0.444 samples = 18 value = [6, 12]
X[1] <= 0.967 gini = 0.219 samples = 32 value = [4, 28]
X[1] <= -0.222 gini = 0.497 samples = 52 value = [28, 24]
X[0] <= 0.535 gini = 0.054 samples = 145 value = [4, 141]
X[1] <= -0.462 gini = 0.015 samples = 258 value = [2, 256]
X[1] <= 0.343 gini = 0.5 samples = 4 value = [2, 2]
gini = 0.0 samples = 1 value = [0, 1]
gini = 0.0 samples = 1 value = [1, 0]
X[0] <= -1.331 gini = 0.494 samples = 27 value = [15, 12]
X[0] <= -1.215 gini = 0.278 samples = 12 value = [10, 2]
X[0] <= -0.981 gini = 0.245 samples = 14 value = [2, 12]
gini = 0.0 samples = 4 value = [4, 0]
X[1] <= 0.888 gini = 0.444 samples = 3 value = [2, 1]
X[1] <= 1.115 gini = 0.128 samples = 29 value = [2, 27]
X[0] <= -0.081 gini = 0.083 samples = 23 value = [1, 22]
X[1] <= 0.679 gini = 0.128 samples = 29 value = [27, 2]
X[0] <= 0.037 gini = 0.041 samples = 144 value = [3, 141]
gini = 0.0 samples = 1 value = [1, 0]
X[0] <= 0.671 gini = 0.5 samples = 2 value = [1, 1]
X[1] <= 0.9 gini = 0.008 samples = 256 value = [1, 255]
gini = 0.0 samples = 2 value = [0, 2]
gini = 0.0 samples = 2 value = [2, 0]
X[1] <= -0.309 gini = 0.48 samples = 25 value = [15, 10]
gini = 0.0 samples = 2 value = [0, 2]
X[1] <= 0.653 gini = 0.165 samples = 11 value = [10, 1]
gini = 0.0 samples = 1 value = [0, 1]
X[0] <= -1.061 gini = 0.142 samples = 13 value = [1, 12]
gini = 0.0 samples = 1 value = [1, 0]
gini = 0.0 samples = 1 value = [0, 1]
gini = 0.0 samples = 2 value = [2, 0]
X[1] <= 1.112 gini = 0.219 samples = 16 value = [2, 14]
gini = 0.0 samples = 13 value = [0, 13]
gini = 0.0 samples = 1 value = [1, 0]
gini = 0.0 samples = 22 value = [0, 22]
gini = 0.0 samples = 25 value = [25, 0]
X[0] <= 0.367 gini = 0.5 samples = 4 value = [2, 2]
X[0] <= 0.031 gini = 0.083 samples = 69 value = [3, 66]
gini = 0.0 samples = 75 value = [0, 75]
gini = 0.0 samples = 1 value = [0, 1]
gini = 0.0 samples = 1 value = [1, 0]
gini = 0.0 samples = 241 value = [0, 241]
X[1] <= 0.912 gini = 0.124 samples = 15 value = [1, 14]
X[0] <= -1.45 gini = 0.375 samples = 4 value = [1, 3]
X[0] <= -1.418 gini = 0.444 samples = 21 value = [14, 7]
gini = 0.0 samples = 9 value = [9, 0]
X[1] <= 0.671 gini = 0.5 samples = 2 value = [1, 1]
gini = 0.0 samples = 8 value = [0, 8]
X[0] <= -1.046 gini = 0.32 samples = 5 value = [1, 4]
X[1] <= 1.063 gini = 0.124 samples = 15 value = [1, 14]
gini = 0.0 samples = 1 value = [1, 0]
gini = 0.0 samples = 2 value = [2, 0]
gini = 0.0 samples = 2 value = [0, 2]
X[1] <= 1.012 gini = 0.057 samples = 68 value = [2, 66]
gini = 0.0 samples = 1 value = [1, 0]
gini = 0.0 samples = 1 value = [1, 0]
gini = 0.0 samples = 14 value = [0, 14]
gini = 0.0 samples = 3 value = [0, 3]
gini = 0.0 samples = 1 value = [1, 0]
X[1] <= 0.496 gini = 0.337 samples = 14 value = [11, 3]
X[0] <= -1.381 gini = 0.49 samples = 7 value = [3, 4]
gini = 0.0 samples = 1 value = [0, 1]
gini = 0.0 samples = 1 value = [1, 0]
gini = 0.0 samples = 1 value = [1, 0]
gini = 0.0 samples = 4 value = [0, 4]
X[1] <= 1.054 gini = 0.278 samples = 6 value = [1, 5]
gini = 0.0 samples = 9 value = [0, 9]
gini = 0.0 samples = 1 value = [1, 0]
X[1] <= 1.247 gini = 0.029 samples = 67 value = [1, 66]
X[1] <= 0.428 gini = 0.444 samples = 9 value = [6, 3]
gini = 0.0 samples = 5 value = [5, 0]
gini = 0.0 samples = 3 value = [0, 3]
X[1] <= 0.54 gini = 0.375 samples = 4 value = [3, 1]
gini = 0.0 samples = 5 value = [0, 5]
gini = 0.0 samples = 1 value = [1, 0]
gini = 0.0 samples = 48 value = [0, 48]
X[1] <= 1.248 gini = 0.1 samples = 19 value = [1, 18]
X[0] <= -1.583 gini = 0.375 samples = 8 value = [6, 2]
gini = 0.0 samples = 1 value = [0, 1]
gini = 0.0 samples = 3 value = [3, 0]
gini = 0.0 samples = 1 value = [0, 1]
gini = 0.0 samples = 1 value = [1, 0]
gini = 0.0 samples = 18 value = [0, 18]
X[0] <= -1.649 gini = 0.5 samples = 4 value = [2, 2]
gini = 0.0 samples = 4 value = [4, 0]
gini = 0.0 samples = 1 value = [1, 0]
X[1] <= -0.236 gini = 0.444 samples = 3 value = [1, 2]
gini = 0.0 samples = 1 value = [1, 0]
gini = 0.0 samples = 2 value = [0, 2]

How much can we reduce the size of the decision tree?

Example

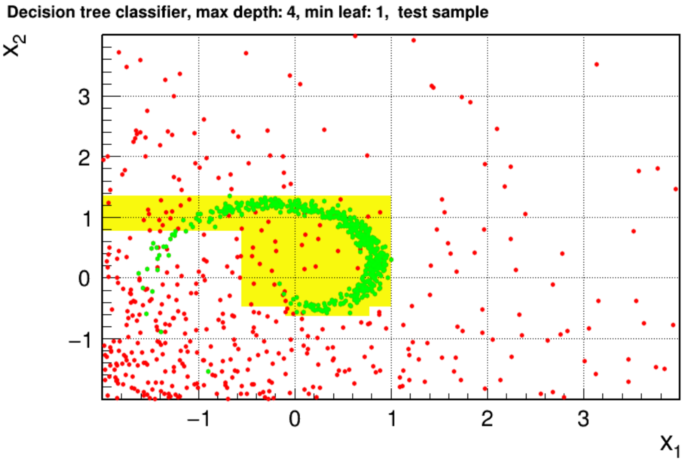
Good performance (efficiency above 90%) already for 4 cut levels!



Decision Trees

Example

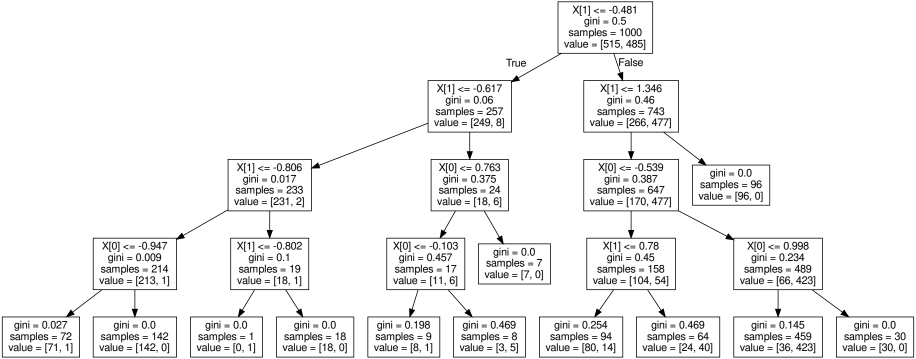
Good performance (efficiency above 90%) already for 4 cut levels!



Decision Trees

Example

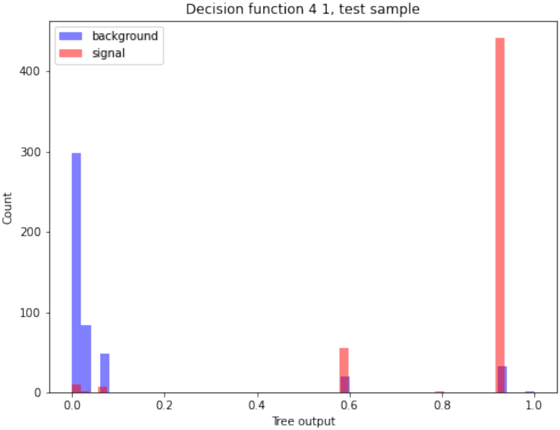
Good performance (efficiency above 90%) already for 4 cut levels!



Decision Trees

Example

Good performance (efficiency above 90%) already for 4 cut levels!



Machine Learning

1. Artificial Neural Networks
2. Boosting
3. Decision Trees
4. Boosted Decision Trees
5. Homework

Boosted Decision Trees

For their good performance, decision trees are "natural candidates" for use in boosting procedure, to get even better classifiers.

Boosted Decision Trees (BDT) algorithms are widely used in particle physics, mainly for their flexibility and stability.

Boosted Decision Trees

For their good performance, decision trees are “natural candidates” for use in boosting procedure, to get even better classifiers.

Boosted Decision Trees (BDT) algorithms are widely used in particle physics, mainly for their flexibility and stability.

Many different algorithms exist, both concerning tree generation and training, and boosting procedure.

Wide range of options implemented in **sklearn** library.

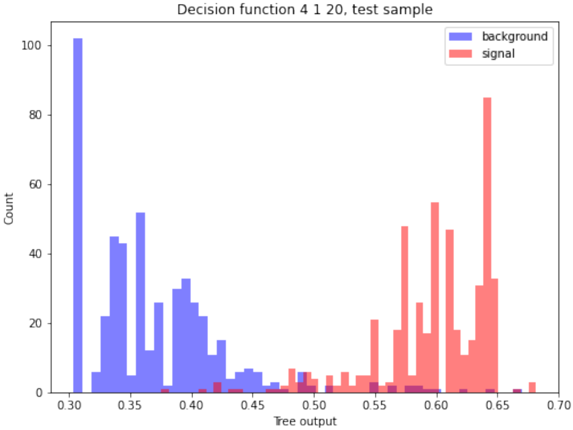
TMVA (Multi Variate Analysis) package for **root** widely used in particle physics community.
More advanced tuning options (⇒ better performance?), but more complicated to use. Based on root, is well integrated into data processing and analysis framework...

Boosted Decision Trees

BDT Example

Good performance (efficiency ∼ 95%) already with 20 trees.

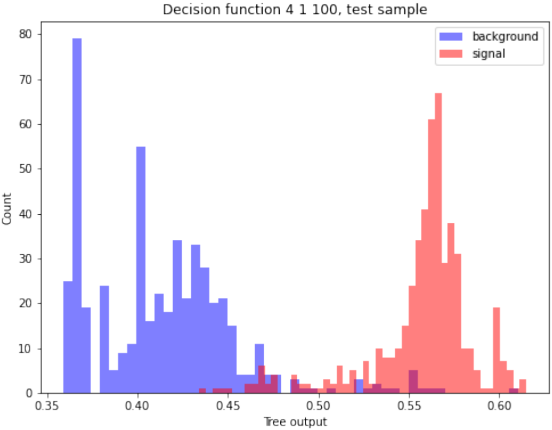
20 trees



BDT Example

Good performance (efficiency $\sim$ 95%) already with 20 trees.

100 trees

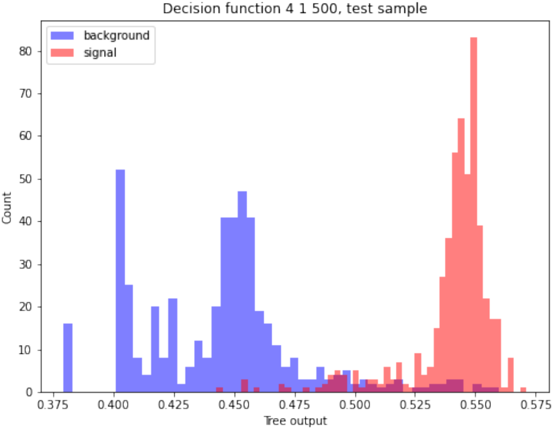


Boosted Decision Trees

BDT Example

Good performance (efficiency $\sim$ 95%) already with 20 trees.

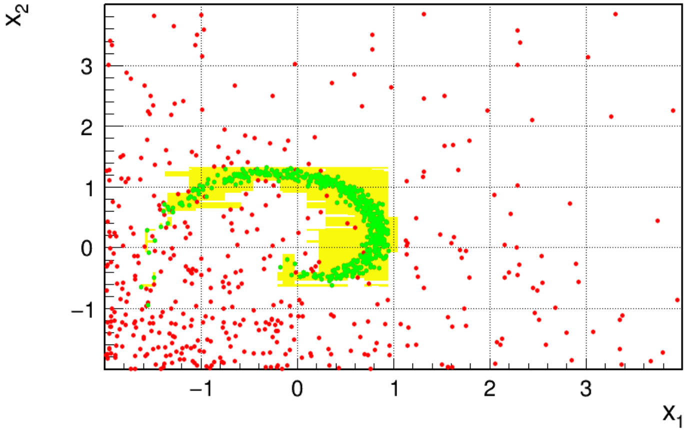
500 trees



BDT Example

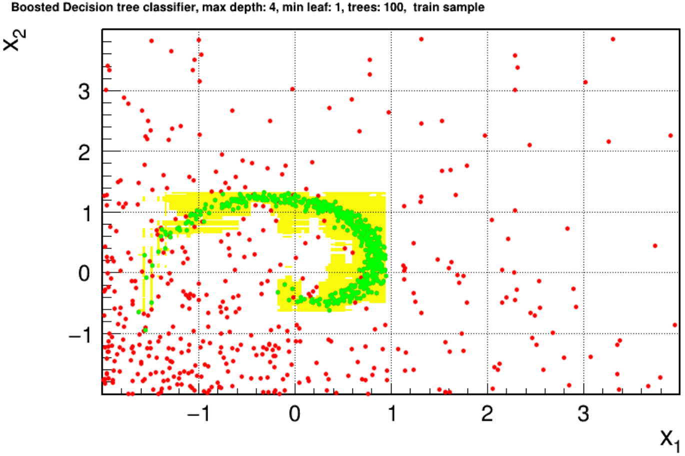
Classification “follows” training sample better and better, with increasing number of trees... 20 trees

Boosted Decision tree classifier, max depth: 4, min leaf: 1, trees: 20, train sample



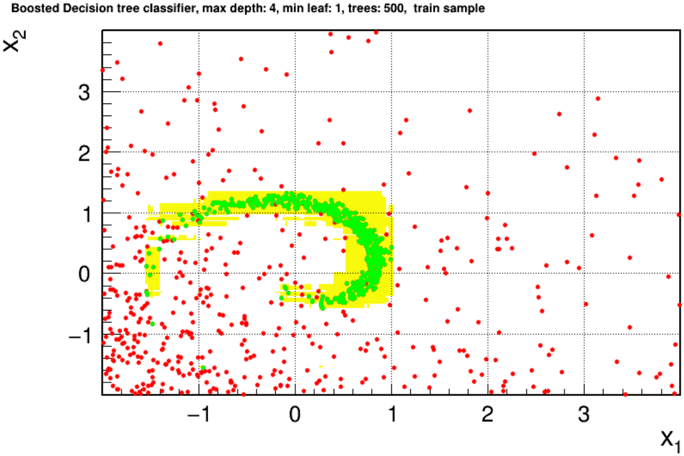
BDT Example

Classification "follows" training sample better and better, with increasing number of trees... 100 trees



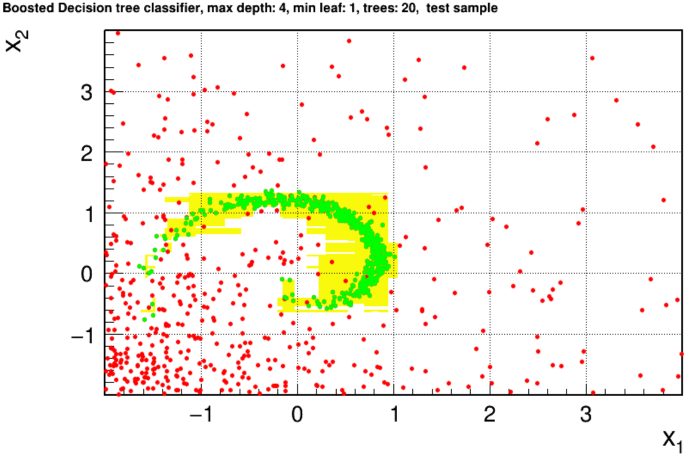
BDT Example

Classification "follows" training sample better and better, with increasing number of trees... 500 trees ⇒ 100% training efficiency



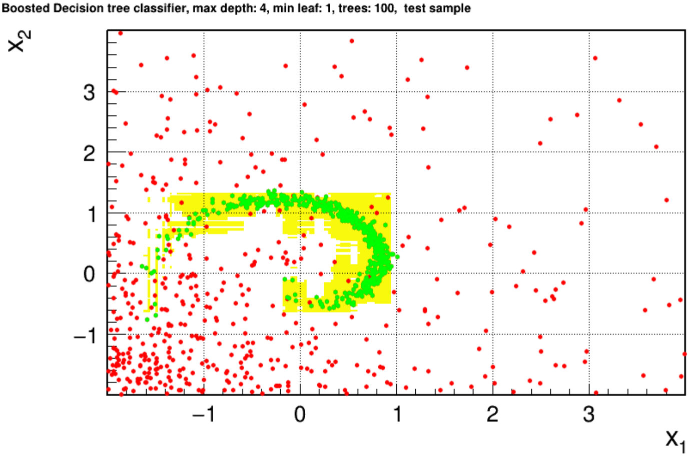
BDT Example

But results "saturate" at some point (at efficiency ∼ 95%) for independent test sample. 20 trees



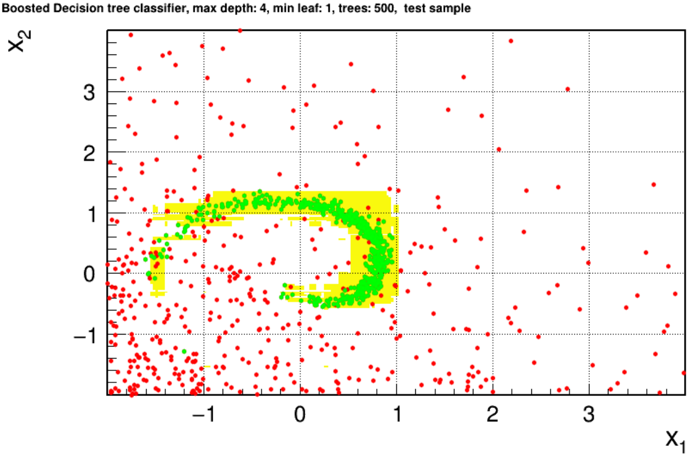
BDT Example

But results “saturate” at some point (at efficiency ∼ 95%) for independent test sample. 100 trees



BDT Example

But results "saturate" at some point (at efficiency ∼ 95%) for independent test sample. 500 trees

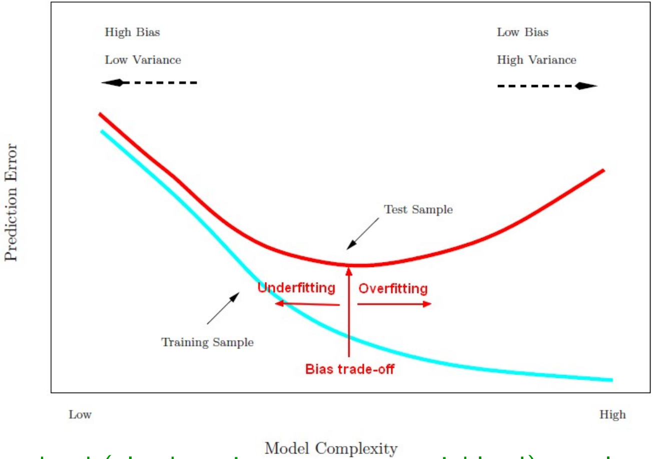


Boosted Decision Trees

Overtraining

source: datacadamia.com

Is a common problem in all Machine Learning methods

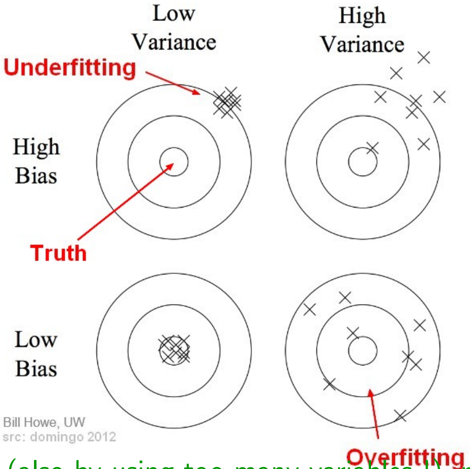


If we try too hard (also by using too many variables !), result can get

Boosted Decision Trees

Overtraining source: datacadamia.com

Is a common problem in all Machine Learning methods



Bill Howe, UW
src: domingo 2012

If we try too hard (also by using too many variables !), result can get

Machine Learning

1 Artificial Neural Networks

2 Boosting

3 Decision Trees

4 Boosted Decision Trees

5 Homework

Homework Solutions to be uploaded by February 2.

Three samples of events $\mathbf{x} = (x_1, x_2, x_3, x_4)$ were prepared:

- training signal sample
- training background sample
- test sample with signal and background events for the analysis

⇒ to be downloaded from lecture web page

Use one of the presented approaches to obtain event classification for the considered event samples:

- draw ROC curve for the obtained classifier
- extract the fraction of the signal events in the test sample
- discuss how the precision of the result depends on the selection cut

Numbers of selected signal and background events have to be corrected for classification efficiency and errors...