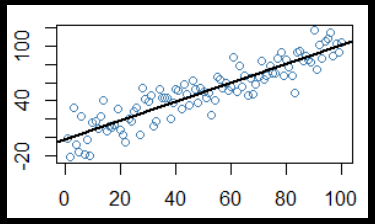
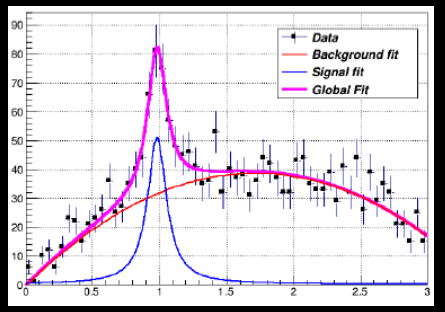
Temas avanzados en física computacional
Análisis de datos





Semestre

2016-I

Clase-6

José Bazo

jbazo@pucp.edu.pe

Contenidos del curso

- ✓ Introducción al análisis de datos y data science
- ✓ Lenguaje de programación R
- ✓ ROOT Data Analysis Framework
- ✓ Manipulación y visualización de datos
- ✓ Modelamiento estadístico

6. Machine Learning
7. TMVA (Toolkit for Multivariate Data Analysis)

6. Machine Learning

Bibliografía

Hastie et al. The Elements of Statistical Learning. 2008

Hoecker et al. Toolkit for Multivariate Data Analysis with ROOT User´s Guide. 2007

Machine learning

Retos del aprendizaje con datos:

- Gran número de observaciones y variables
- Tiempo computacional
- Datos no parejos: mezcla de variables cuantitativas, binarias y categóricas
- Valores faltantes
- Distribuciones de respuesta asimétricas y con colas largas
- Fracción considerable de grandes errores de medida (outliers).
- Variables de predicción medidas con escalas muy diferentes
- Sólo una fracción de variables predictivas son relevantes para el resultado.
- Filtrar características irrelevantes que perjudican rendimiento de métodos.
- Se necesitan modelos interpretables.
- Entender cualitativamente relación entre variables de input y output (no blackbox)

Métodos en TMVA

- **Rectangular Cut Optimisation**

Corte inferior y superior para cada variable (busca intervalos usando árboles binarios).
Recomendable reducir dimensiones antes de optimizar cortes.

- **Fisher and H-Matrix**

Discriminación lineal solo para clasificación.

- **Linear Discriminant (LD)**

Equivalente a Fisher pero incluye regresión lineal

- **Function Discriminant Analysis (FDA)**

Extiende LD con correlaciones moderadamente no lineales ajustadas a los datos.

Métodos en TMVA

- **Likelihood**

Estimación de PDF no paramétrica via suavizamiento de histogramas e interpolación con funciones spline y estimadores de densidad de kernel cuasi-unbinned.
PDF ajustada individualmente para cada variable de input.

Multidimensional probability density estimators (para pocas variables input):

- **PDE-RS** (Probability Density Estimator Range-Search)

Estimador de densidad de probabilidad (PDE) multidimensional.
Búsqueda de intervalo adaptativa, varios métodos de estimación de kernel.

- **PDE-Foam**

Versión rápida de PDE-RS en volúmenes fijos determinados y optimizados durante training: self-adapting phase-space binning

- **k-NN** (k-Nearest Neighbour)

Con poca estadística underperforms ligeramente PDE-RS, pero más rápido para grandes muestras.

Métodos en TMVA

- **BDT** (Boosted Decision Trees)

The BDT implementation has received constant attention over the years of its development. The current version includes additional features like bagging or gradient boosting, and manual or automatic pruning of statistically insignificant nodes.

- **ANN** (Artificial Neural Networks)

Fast feed-forward multilayer perceptron (MLP) algorithms.

- **SVM** (Support Vector Machine)

Bueno para discriminación no lineal e insensible a overtraining.
Optimización directa debido a pocos parámetros (2 en caso de kernel gaussiano).

- **RuleFit**

by J. Friedman, Rule based Learning Ensembles

No en TMVA:
MARS (Multivariate Adaptive Regression Splines) (for high-dimensional problems) by J. Friedman

Decision trees

Decision trees

Método “off-the-shelf”: aplicable sin necesidad de mucho pre-procesamiento o tuning.
-> Decision Trees, **método más popular de aprendizaje**.

Ventajas:

- Rápidos de construir
- Resultados interpretables
- Incorporan mezcla de variables numéricas y categóricas
- Invariantes bajo transformaciones monótonas (scaling) de variables input.
- Inmunes a efectos de outliers.
- Selección interna de características-> resistentes a inclusión de inputs irrelevantes
- Simples: cada paso (node splitting) conlleva solo una optimización unidimensional.
- **Defecto**: imprecisión, poco poder de predicción
- **BDT**: **Boosted Decision Trees** mejoran mucho la precisión, manteniendo casi todas las otras propiedades, salvo velocidad e interpretabilidad.

Comparación de modelos

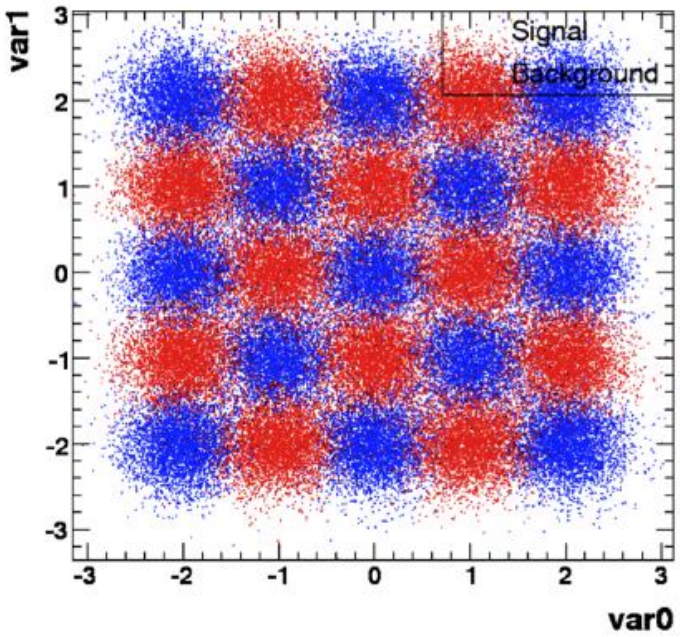
TABLE 10.1. *Some characteristics of different learning methods. Key:* ▲ *= good,* ◆ *=fair, and* ▼ *=poor.*

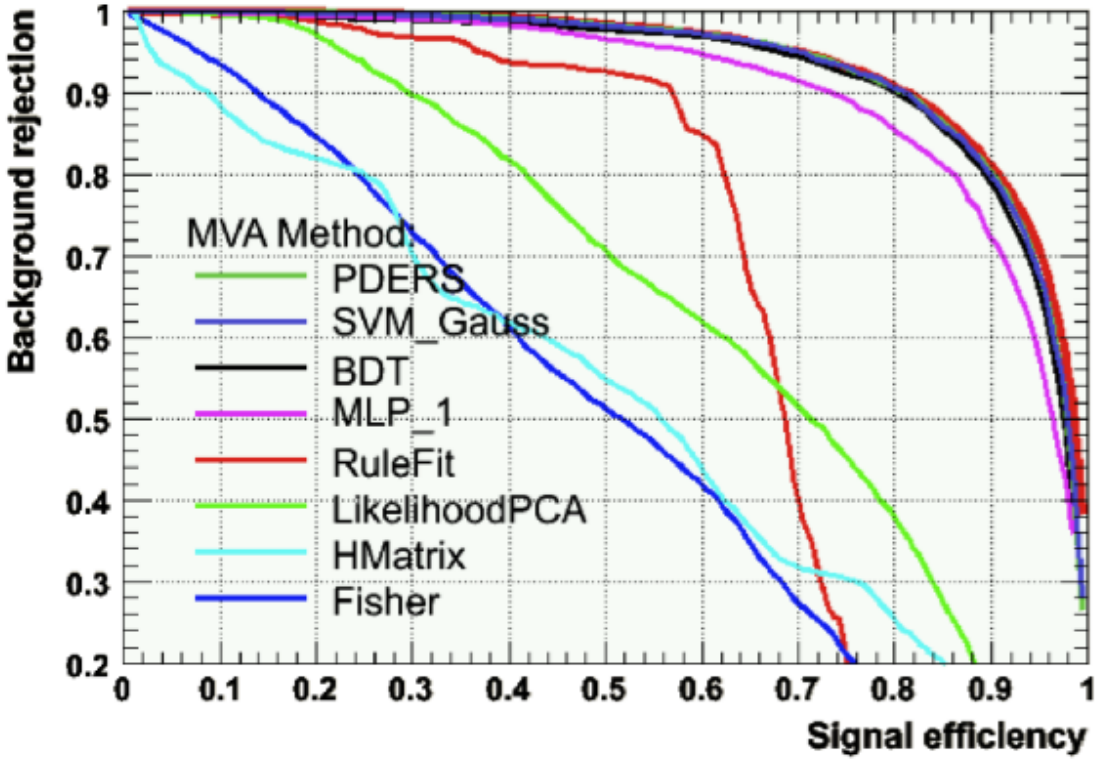
Characteristic	Neural Nets	SVM	Trees	MARS	k-NN, Kernels
Natural handling of data of "mixed" type	▼	▼	▲	▲	▼
Handling of missing values	▼	▼	▲	▲	▲
Robustness to outliers in input space	▼	▼	▲	▼	▲
Insensitive to monotone transformations of inputs	▼	▼	▲	▼	▼
Computational scalability (large N)	▼	▼	▲	▲	▼
Ability to deal with irrelevant inputs	▼	▼	▲	▲	▼
Ability to extract linear combinations of features	▲	▲	▼	▼	◆
Interpretability	▼	▼	◆	▲	▼
Predictive power	▲	▲	▼	◆	▲

Comparación de modelos

	Criteria	MVA Method: Cuts	Likelihood	PDE-RS / k-NN	PDE-Foam	H-Matrix	Fisher / LD	MLP	BDT	Rule-Fit	SVM
Performance	No or linear correlations	⋆	⋆⋆	⋆	⋆	⋆	⋆⋆	⋆⋆	⋆	⋆⋆	⋆
	Nonlinear correlations	○	○	⋆⋆	⋆⋆	○	○	⋆⋆	⋆⋆	⋆⋆	⋆⋆
Speed	Training	○	⋆⋆	⋆⋆	⋆⋆	⋆⋆	⋆⋆	⋆	⋆	⋆	○
	Response	⋆⋆	⋆⋆	○	⋆	⋆⋆	⋆⋆	⋆⋆	⋆	⋆⋆	⋆
Robustness	Overtraining	⋆⋆	⋆	⋆	⋆	⋆⋆	⋆⋆	⋆	⋆[39]	⋆	⋆⋆
	Weak variables	⋆⋆	⋆	○	○	⋆⋆	⋆⋆	⋆	⋆⋆	⋆	⋆
Curse of dimensionality		○	⋆⋆	○	○	⋆⋆	⋆⋆	⋆	⋆	⋆	
Transparency		⋆⋆	⋆⋆	⋆	⋆	⋆⋆	⋆⋆	○	○	○	○

Comparación de modelos



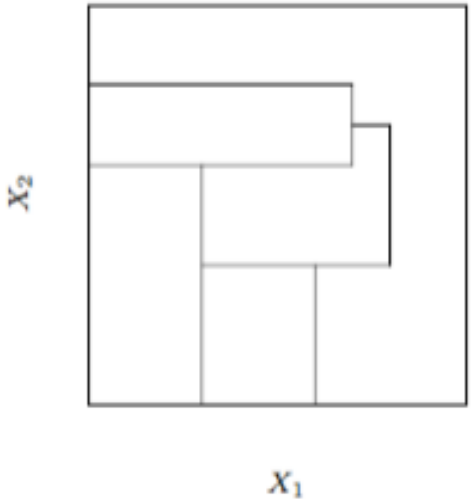


Decision trees (DT)

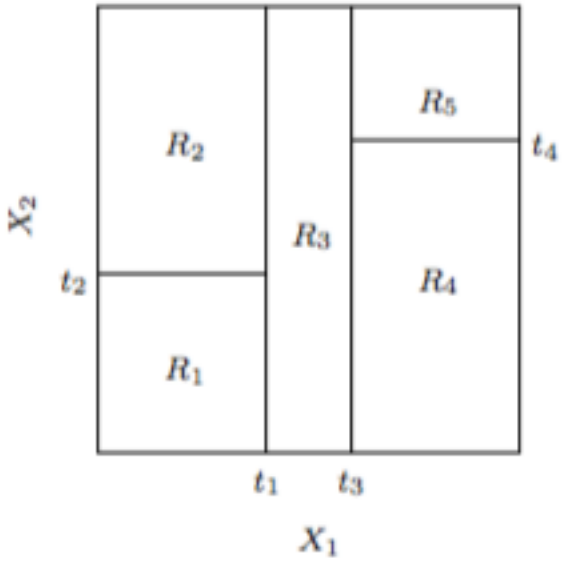
Método:

- Dividir espacio de inputs en regiones (rectángulos)
- ajustar modelo simple (constante) en cada una.

Ejemplo: Problema de regresión con respuesta continua Y e inputs X_1 y X_2 con valores en el intervalo unitario.



División general compleja

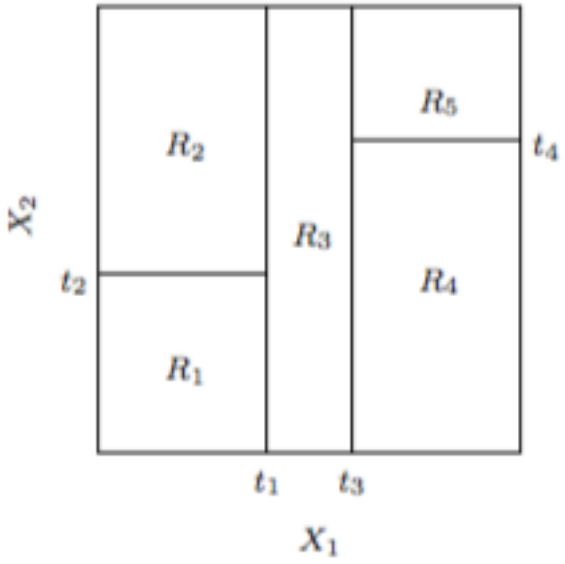


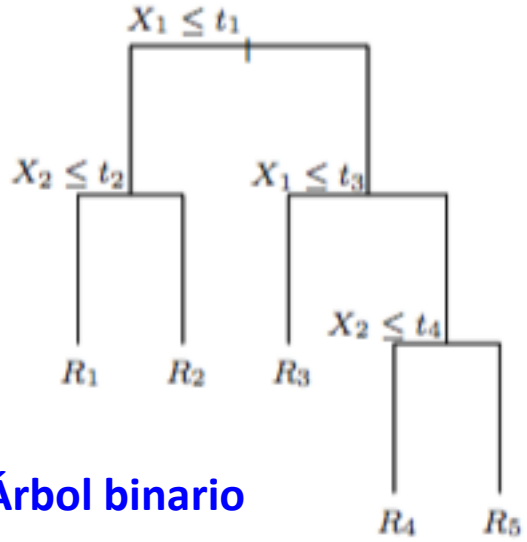
Partición obtenida por división binaria recursiva

Decision trees

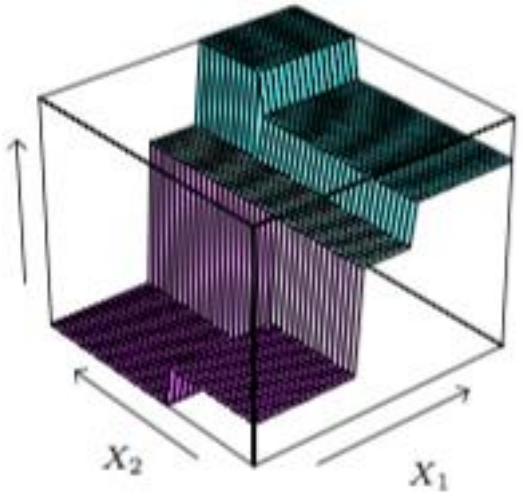
Modelo de regresión $$\hat{f}(X) = \sum_{m=1}^{5} c_m I\{(X_1, X_2) \in R_m\}.$$

Constante c_m en cada región R_m





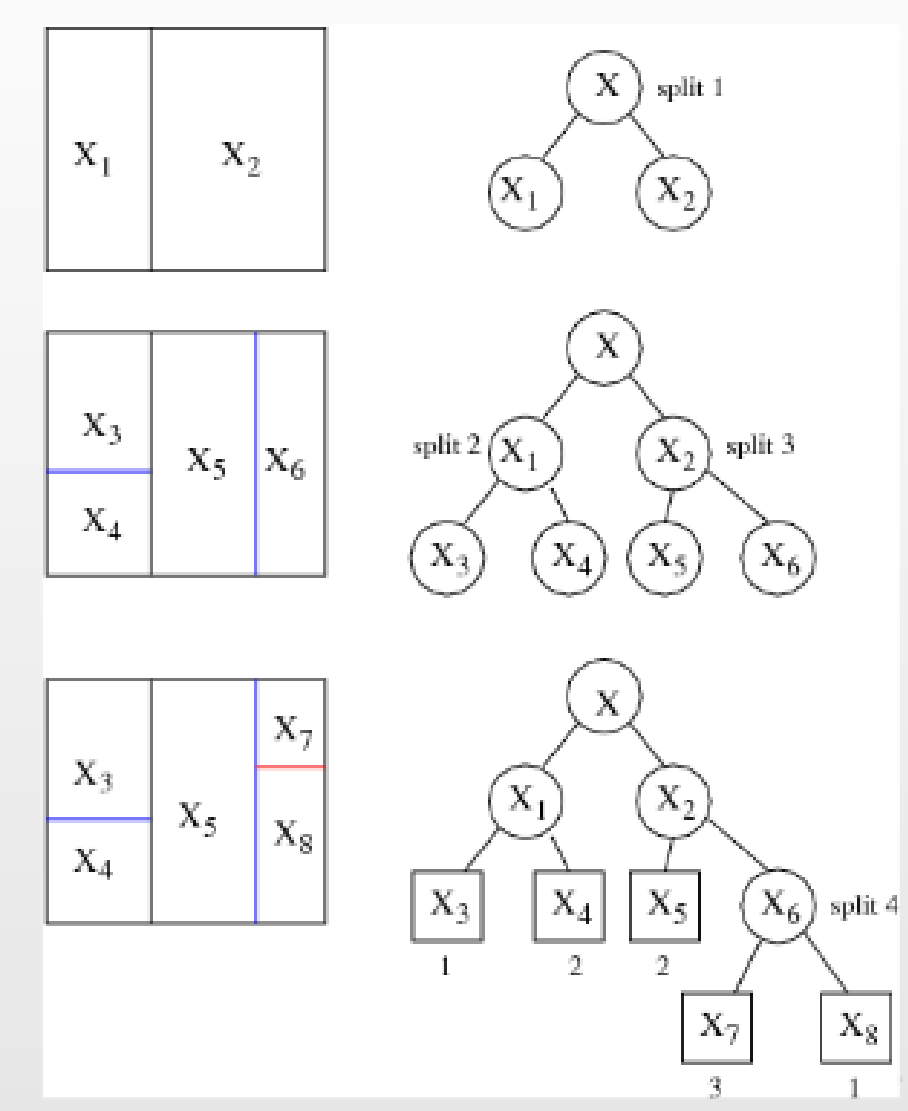
Árbol binario



Superficie de regresión
(predicción)

Nodos terminales u hojas
corresponden a las regiones R_m

Decision trees



- Dividir espacio en dos regiones
- Modelar respuesta con media de Y en cada región.
- Se escoge la variable y punto de corte que dé el mejor ajuste.
- Repetir proceso con otras divisiones hasta detenerse por alguna regla.

Decision trees

¿Cómo desarrollar un **árbol de regresión**?

Ejemplo: Datos consisten en *p* inputs y un output para cada N observaciones. (x_i,y_i) con i=1,2,…N, $x_i=(x_{i1}, x_{i2}, \ldots x_{ip})$

Algoritmo debe decidir sobre:

- puntos de división para cada variable
- topología del árbol.

Si se divide en ***m* regiones**, R_m y se modela la respuesta en cada región con una constante $\mathbf{c_m}$:

$$f(x) = \sum_{m=1}^{M} c_m I(x \in R_m).$$

Criterio de minimización: suma de cuadrados $\sum(y_i - f(x_i))^2$

el mejor c_m es el promedio de y_i en la región R_m: $\hat{c}_m = \text{ave}(y_i | x_i \in R_m).$

Decision trees

Se utiliza el siguiente **algoritmo**:
Se considera la variable j de división y el punto s y se define el par:

$$R_1(j,s) = \{X | X_j \leq s\} \text{ and } R_2(j,s) = \{X | X_j > s\}.$$

Se buscan j y s de tal manera que sean solución de:

$$\min_{j,\, s} \left[\min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2 \right]$$

Para cualquier j y s la minimización interior tiene solución:

$$\hat{c}_1 = \text{ave}(y_i | x_i \in R_1(j,s)) \text{ and } \hat{c}_2 = \text{ave}(y_i | x_i \in R_2(j,s))$$

Para cada variable x_j se puede hallar el mejor s rápidamente y luego se prueba con todos los j para hallar el mejor par (j,s)

Decision trees

Con la mejor división (j,s) cada región se vuelva a dividir en 2 regiones correspondientes. Todo este proceso se repite en las nuevas regiones.

¿Cuánto debe crecer el árbol?

- Árbol muy grande -> probablemente da overfit
- Árbol muy pequeño -> no captura estructura importante

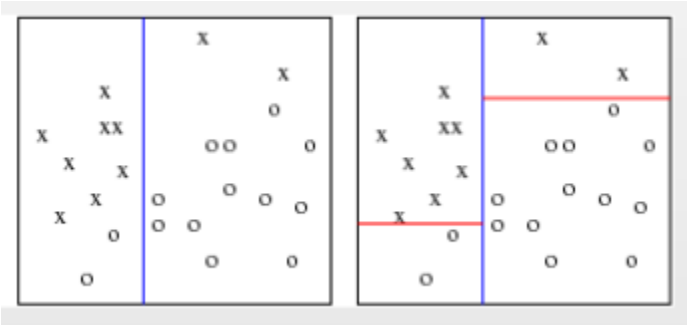
Tree size = tuning parameter

Gobierna complejidad del modelo. Tamaño óptimo escogido para datos

Estrategia: parar división cuando:

- se alcanza un tamaño mínimo del nodo (eg. 5 eventos)
- categoría es pura (solo resultados iguales).

Luego el árbol grande T_0 se reduce usando técnica costo-complejidad



Decision trees

Definir **subtree** $T \subset T_0$

obtenido al colapsar algunos nodos internos (no terminales): **pruning**

Identificamos nodos terminales con índice m y región R_m.

Número de **nodos terminales** de T = |T|

Se definen:

$$N_m = \#\{x_i \in R_m\},$$

$$\hat{c}_m = \frac{1}{N_m} \sum_{x_i \in R_m} y_i,$$

$$Q_m(T) = \frac{1}{N_m} \sum_{x_i \in R_m} (y_i - \hat{c}_m)^2,$$ *squared-error node impurity measure*

Cost complexity criterion:

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha |T|.$$

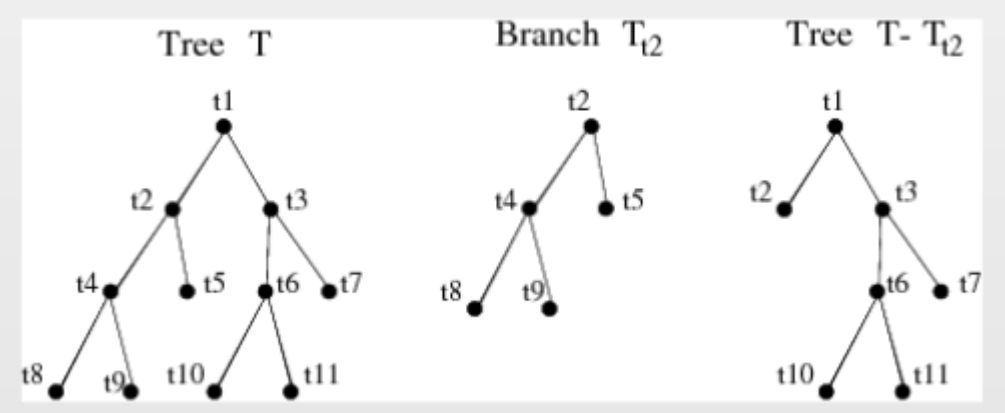
α≥0 número real "**complexity parameter**"

Encontrar para cada α el $T_\alpha \subseteq T_0$ que minimiza $C_\alpha(T)$.

α gobierna balance entre tamaño del árbol y bondad del ajuste.

Decision trees

Pruning a branch T_t de un árbol T consiste en eliminar de T todos los descendientes de t: cortar todo T_t excepto su nodo raíz. Resultado: $T-T_t$.



Meta de pruning: remover nodos estadísticamente insignificantes reduciendo overtraining del árbol .

Decision trees

Valores grandes de α -> árboles pequeños y viceversa
α = 0 -> solución es árbol completo

¿Cómo elegir α?

Para cada α existe un único subtree T_α más pequeño que minimiza $C_\alpha(T)$.

Para encontrar T_α se usa ***weakest link pruning***:

Colapsar sucesivamente nodos internos que producen el más pequeño incremento por nodo de $\Sigma_m N_m Q_m(T)$ y continuar hasta producir el árbol de un solo nodo (raíz)

Da secuencia finita de subtrees que contiene a T_α

Tree	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}	T_{12}	T_{13}
$\lvert\tilde{T}_k\rvert$	71	63	58	40	34	19	10	9	7	6	5	2	1

Se estima α via 10-fold cross-validation de manera que minimiza $C_\alpha(T)$

Más detalles

Decision trees

¿Cómo desarrollar un **árbol de clasificación**?

Ejemplo: Resultados de clasificación 1, 2, . . . ,K

Cambios en algoritmo:

- Criterio de división de nodos
- Pruning

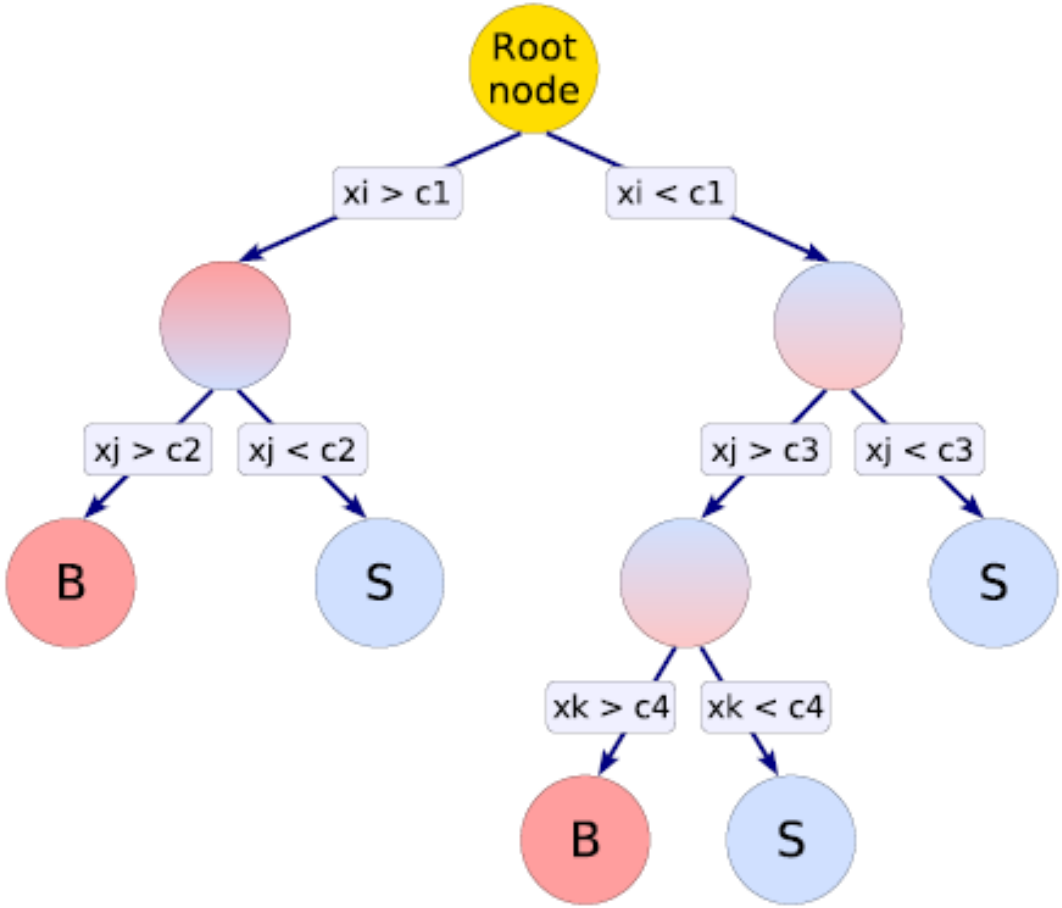
Se define la **proporción de observaciones** de clase k en el nodo m

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k)$$

Se clasifican las observaciones en el nodo m a la clase k por mayoría

$$k(m) = \arg\max_k \hat{p}_{mk}$$

Métodos de Boosting



Decision trees

Medidas de impureza del nodo $Q_m(T)$

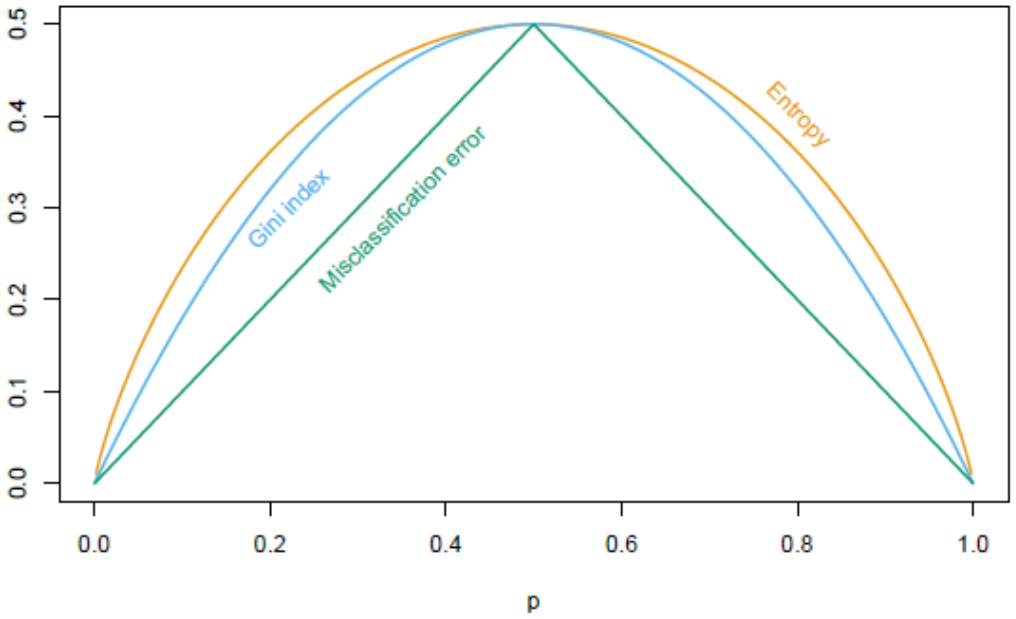
Misclassification error: $\frac{1}{N_m}\sum_{i\in R_m} I(y_i \neq k(m)) = 1 - \hat{p}_{mk(m)}.$

Gini index: $\sum_{k\neq k'} \hat{p}_{mk}\hat{p}_{mk'} = \sum_{k=1}^{K} \hat{p}_{mk}(1-\hat{p}_{mk}).$

Cross-entropy or deviance: $-\sum_{k=1}^{K} \hat{p}_{mk}\log\hat{p}_{mk}.$

} diferenciables

$Q_m(T)$



Clasificación de 2 categorías:

$1 - \max(p, 1-p)$

$2p(1-p)$

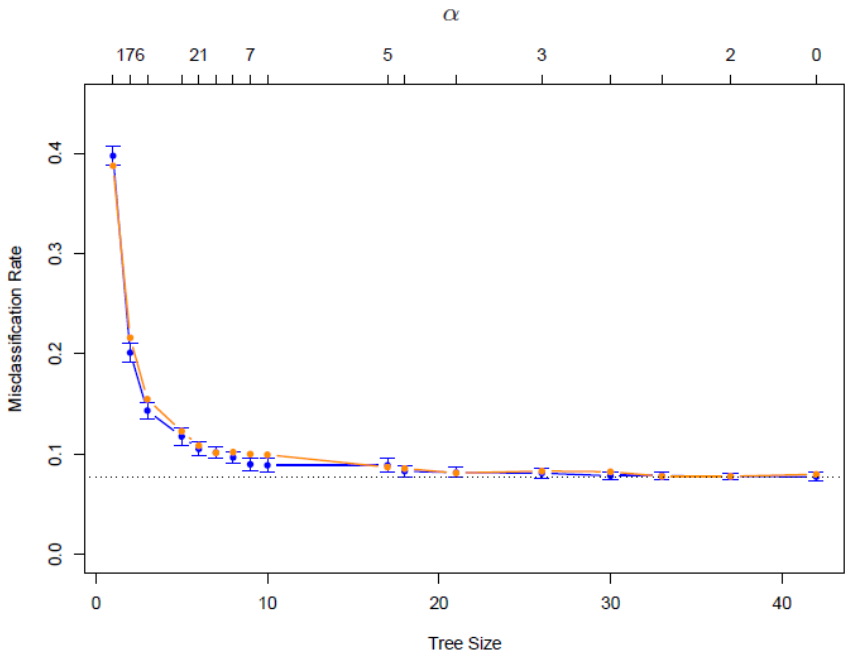
$-p\log p - (1-p)\log(1-p)$

Cross-entropy normalizada para que pase por (0.5,0.5)

Decision trees

Usualmente:

- División de nodos: Gini index o cross-entropy
- Cost-complexity pruning: misclassification rate



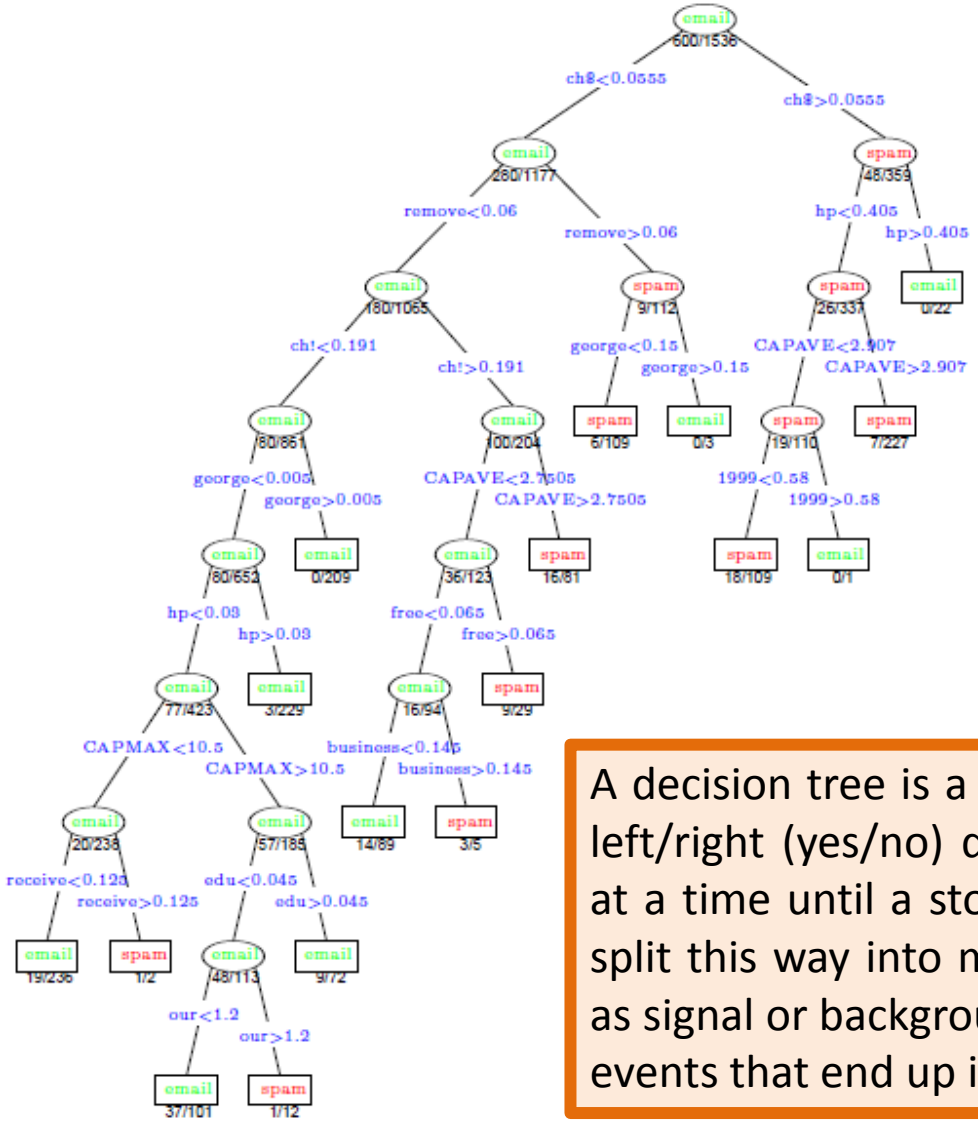
Implementación de árboles: algoritmo:

CART (**C**lassification **A**nd **R**egression **T**ree)

Otro algoritmo: **C5.0** (Quinlan)

Ranking de variables input: contar cuántas veces se usa la variable para dividir nodos del DT, pesando cada división con la separación ganada cuadrada y número de eventos en el nodo.

Decision trees



Pruned tree.

Número bajo nodo terminal es misclassification rate

A decision tree is a binary tree structured classifier. Repeated left/right (yes/no) decisions are taken on one single variable at a time until a stop criterion is fulfilled. The phase space is split this way into many regions that are eventually classified as signal or background, depending on the majority of training events that end up in the final leaf node.

Boosted Decision Trees (BDT)

- Boosting de un DT extiende concepto de un árbol a varios formando un bosque (**forest**).
- Árboles creados del mismo conjunto de training **repesando eventos**.
- Se combinan en un **único clasificador: promedio pesado** de DT individuales.

- Estabiliza respuesta de DTs con respecto a fluctuaciones de muestra y optimiza resultado.
- Algoritmos de boosting trabajan mejor con clasificadores débiles: limitar profundidad del árbol (más que con pruning -> no es necesario)

Algoritmo: **AdaBoost** Freund and Schapire (1997)

Ejemplo: Problema de 2 clases, con variable output $Y \in \{-1, 1\}$
vector de variables input X y clasificador G(X)

Discrete AdaBoost

Friedman et al. (2000)

Tasa de error en muestra de training:

$$\overline{\text{err}} = \frac{1}{N}\sum_{i=1}^{N} I(y_i \neq G(x_i)),$$

Si fuera todo el intervalo real {-1,1}:
Real AdaBoost

Error esperado en predicciones futuras: $\mathrm{E}_{XY} I(Y \neq G(X))$

Boosted Decision Trees (BDT)

Clasificador débil: uno cuya tasa de error es ligeramente mejor que adivinar.

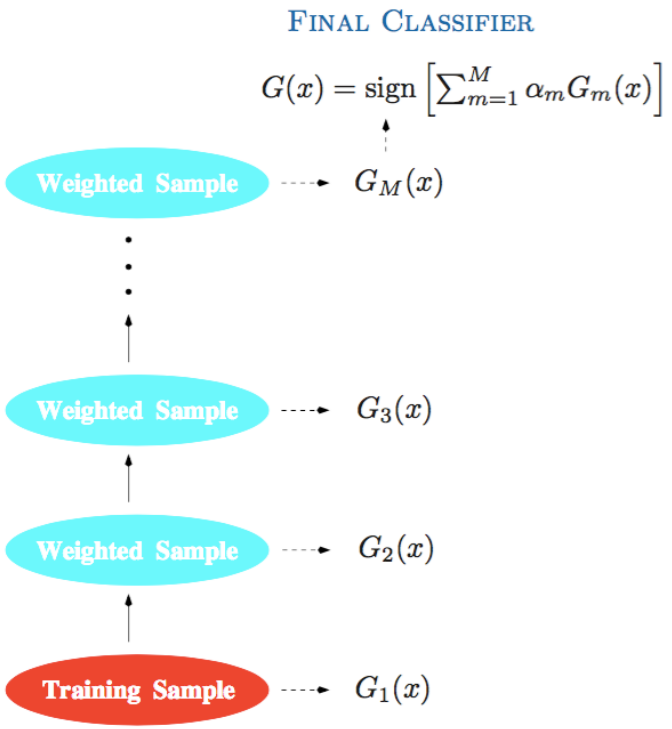
Boosting: aplicar secuencialmente algoritmo de clasificación débil a versiones siempre modificadas de datos produciendo secuencia de clasificadores débiles

$$G_m(x), m = 1, 2, \ldots, M.$$

Predicciones individuales se combinan por mayoría pesada dando predicción final:

$$G(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m G_m(x)\right)$$

Dan mayor peso a clasificadores más precisos.



Boosted Decision Trees (BDT)

Generación de muestras modificadas: cada evento por peso w_i

Inicialmente $w_i=1/N$ entrena primeros datos sin alterar

Para cada iteración posterior los pesos individuales son modificados

en el paso m, a las observaciones mal clasificadas por $G_{m-1}(x)$ se les aumenta el peso w_i, mientras que se disminuye si la clasificación fue correcta.

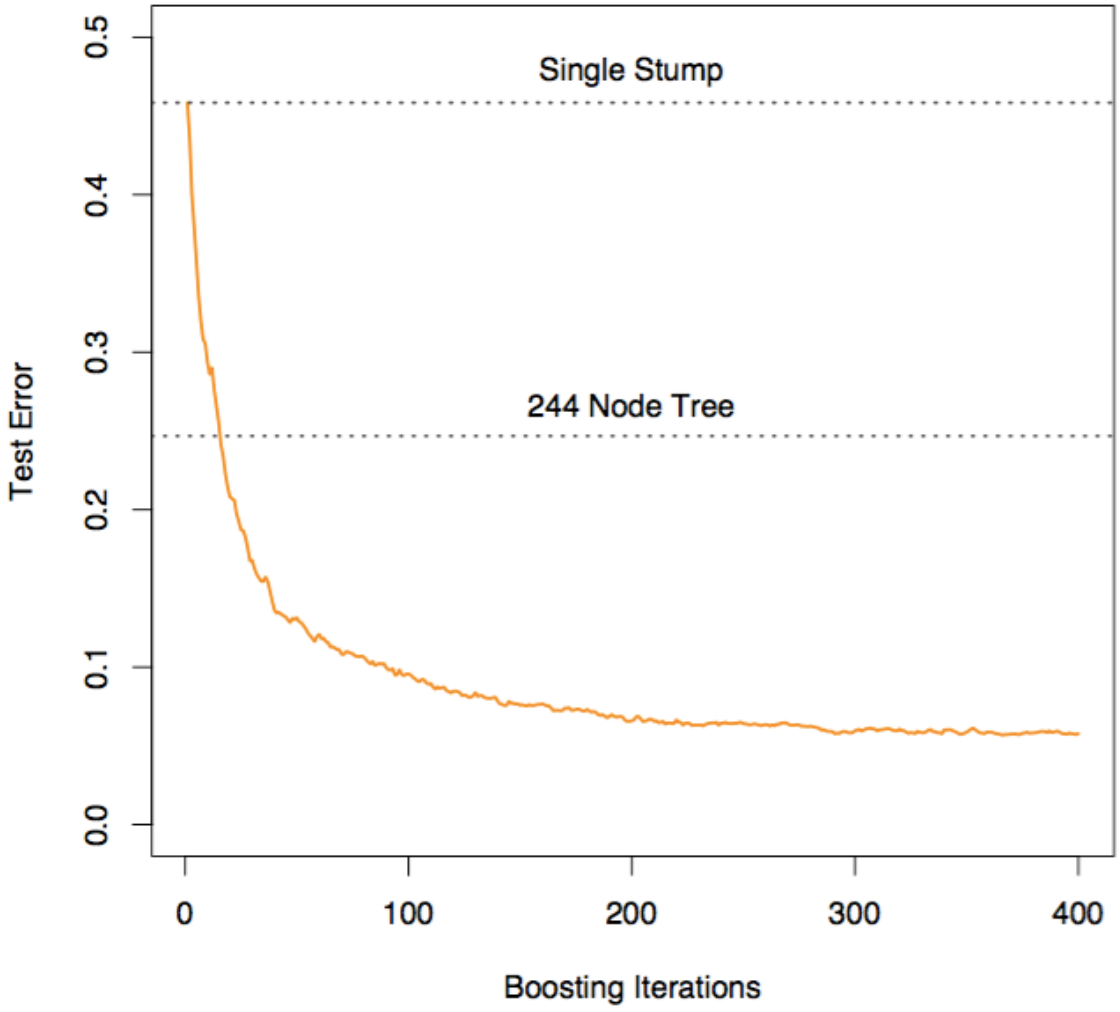
Cada clasificador posterior se debe concentrar más en las observaciones problemáticas de clasificar

Boosted Decision Trees (BDT)

Algorithm 10.1 *AdaBoost.M1.*

1. Initialize the observation weights $w_i = 1/N,\ i = 1, 2, \ldots, N$.
2. For $m = 1$ to M:
   (a) Fit a classifier $G_m(x)$ to the training data using weights w_i.
   (b) Compute
   $$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}.$$
   (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$.
   (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))],\ i = 1, 2, \ldots, N$.
3. Output $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

Boosted Decision Trees (BDT)



Usando TMVA

Para versiones anteriores se puede descargar e instalar.
Versiones actuales ya incluyen TMVA

```
download TMVA "tar-ball" from https://sourceforge.net/projects/tmva/files/
~> tar -zxvf TMVA-v4.2.0.tgz
~> cd TMVA-v4.2.0
~/TMVA-v4.2.0> make
~/TMVA-v4.2.0> cd test
~/TMVA-v4.2.0/test> source setup.sh #for c-shell family: source setup.csh
```

Su uso se divide en:

- **Training Stage: TMVA Factory** — Ejemplo: tutorials/tmva/TMVAClassification.C
- **Application Stage: TMVA Reader** — Ejemplo: tutorials/tmva/TMVAClassificationApplication .C

En otros casos se encuentran en tmva/test/

TMVA: Training Stage

Cargar librerías:

TMVA::Tools::Instance();

Crear archivo que contendrá resultados:

TFile* outputFile = TFile::Open(”TMVAout.root”, "RECREATE");

Declarar TMVA Factory (realiza el análisis) con opciones de configuración:

TMVA::Factory *factory = new TMVA::Factory("TMVAClassification",outputFile,"**V**:!Silent:**Color**:**Transformations**=I:**Draw ProgressBar**:**AnalysisType**=Classification");

```
--- Factory                  : Parsing option string:
--- Factory                  : ... "V:!Silent:Color:Transformations=I:DrawProgressBar:AnalysisType=Classification"
--- Factory                  : The following options are set:
--- Factory                  : - By User:
--- Factory                  :     V: "True" [Verbose flag]
--- Factory                  :     Color: "True" [Flag for coloured screen output (default: True, if in batch mode: False)]
--- Factory                  :     Transformations: "I" [List of transformations to test; formatting example: "Transformations=I;D;P;U;G,D", for identit
y, decorrelation, PCA, Uniform and Gaussianisation followed by decorrelation transformations]
--- Factory                  :     Silent: "False" [Batch mode: boolean silent flag inhibiting any output from TMVA after the creation of the factory cl
ass object (default: False)]
--- Factory                  :     DrawProgressBar: "True" [Draw progress bar to display training, testing and evaluation schedule (default: True)]
--- Factory                  :     AnalysisType: "Classification" [Set the analysis type (Classification, Regression, Multiclass, Auto) (default: Auto)]
--- Factory                  : - Default:
--- Factory                  :     <none>
--- Factory                  : You are running ROOT Version: 6.04/14, Feb 3, 2016
--- Factory                  :
--- Factory                  : _/_/_/_/_/ _|      _|  _|      _|    _|_|
--- Factory                  :    _/      _|_|  _|_|  _|      _|  _|    _|
--- Factory                  :   _/       _|  _|  _|  _|      _|  _|_|_|_|
--- Factory                  :  _/        _|      _|    _|  _|    _|    _|
--- Factory                  : _/         _|      _|      _|      _|    _|
--- Factory                  :
--- Factory                  : ___________TMVA Version 4.2.1, Feb 5, 2015
--- Factory                  :
```

TMVA: Training Stage

Configuration options reference for class: *Factory*				
Option	**Array**	**Default value**	**Predefined values**	**Description**
V	No	False	–	Verbose flag
Color	No	True	–	Flag for coloured screen output (default: True, if in batch mode: False)
Transformations	No		–	List of transformations to test; formatting example: Transformations=I;D;P;U;G,D, for identity, decorrelation, PCA, Uniform and Gaussianisation followed by decorrelation transformations
Silent	No	False	–	Batch mode: boolean silent flag inhibiting any output from TMVA after the creation of the factory class object (default: False)
DrawProgressBar	No	True	–	Draw progress bar to display training, testing and evaluation schedule (default: True)
AnalysisType	No	Auto	Classification, Regression, Multiclass, Auto	Set the analysis type (Classification, Regression, Multiclass, Auto) (default: Auto)

Más opciones aquí

TMVA: Training Stage

Problema de clasificación:

link

- Se necesitan dos muestras: señal y fondo (o dos modelos).
- Seleccionar muestras con cortes para aumentar la pureza o usar MCTruth.
- Cortes aplicados en variables deben ser iguales para datos y MC.

```
Double_t signalWeight = 1.0;
Double_t backgroundWeight=1.0;
```

Archivo se encuentra en tutorial/tmva/data
o en: http://root.cern.ch/files/tmva_class_example.root

```
TFile* toy= new TFile(" toy_sigbkg_categ_offset.root");

 TTree* sigTree = (TTree*)(toy->Get("TreeS"));
 TTree* bkgTree = (TTree*)(toy->Get("TreeB"));

 factory->AddSignalTree(sigTree,signalWeight);
 factory->AddBackgroundTree(bkgTree,backgroundWeight);

 Para pesos individuales (variable debe existir en tree):
 factory->SetSignalWeightExpression("weight1");
 factory->SetBackgroundWeightExpression("weight2");
```

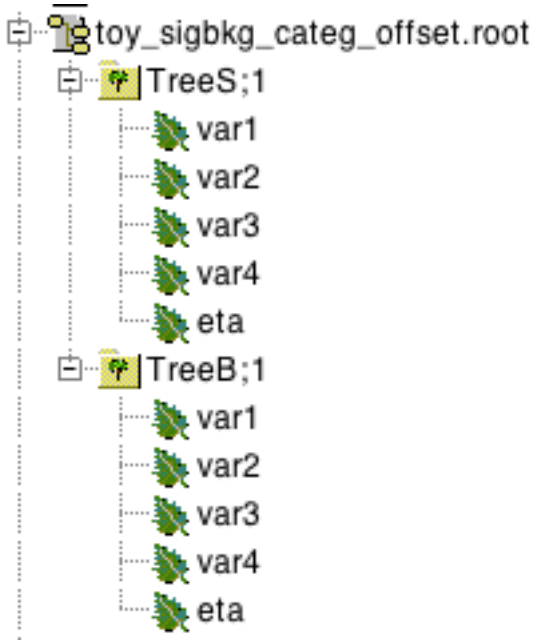
TMVA: Training Stage

Definir lista de variables para entrenar (ya definidas u operaciones con originales)

Tip: para variable ~0 se puede tener una mejor separación si se usa su log

```
factory->AddVariable( "myvar1:= var1+var2",”NewVar1”,”units”, 'F' );
factory->AddVariable( "myvar2:= var1-var2",”NewVar2”,”units”, 'F' );
factory->AddVariable( "var3", "Var3", "units", 'F' );
factory->AddVariable( "var4", "Var4", "units", 'F' );
```

Tipo de variable: float/double ('F')
integer/boolean ('I')



Se pueden añadir variables no usadas en la BDT pero que se almacenan en el tree de output:

```
factory->AddSpectator( "spec1 := var1*2",  "Spec1", "units", 'F' );
```

TMVA: Training Stage

Se pueden usar cortes en las muestras:

```
TCut mycuts = ""; // TCut mycuts = "abs(var1)<0.5 && abs(var2-0.5)<1";
TCut mycutb = ""; // TCut mycutb = "abs(var1)<0.5";
```

Preparar factory para entrenamiento.

```
factory->PrepareTrainingAndTestTree(mycuts,mycutb,"random");
```

Default: dividir muestra en dos: mitad training, mitad test (no overtraining)

modo de dividir aleatorio

Para especificar número:

```
factory->PrepareTrainingAndTestTree( mycuts, mycutb,
"nTrain_Signal=1000:nTrain_Background=1000:SplitMode=Random:NormMode=NumEvents:!V" );
```

TMVA: Training Stage

Para usar trees diferentes para entrenamiento y prueba usar:
factory->AddSignalTree(signalTrainingTree, signalTrainWeight, **"Training"**);
factory->AddSignalTree(signalTestTree, signalTestWeight, **"Test"**);

Definir método(s) multivariado y opciones

usando BDT

factory->**BookMethod**(TMVA::Types::kBDT,"BDT","NTrees=400:MaxDepth=2");

Opciones:

- Número de decision trees
- Máxima profundidad de cada tree.

Más es mejor, pero no hay mucha estadística peligro de overtraing: compromiso.

Boosted Decision Trees:

"BDT", Adaptive Boost
"BDTG", Gradient Boost
"BDTB", uses Bagging
"BDTD", decorrelation + Adaptive Boost
"BDTF", allow usage of fisher discriminant for node splitting

TMVA: Training Stage

Configuration options reference for MVA method: *BDT*				
Option	**Array**	**Default value**	**Predefined values**	**Description**
V	No	False	–	Verbose output (short form of VerbosityLevel below - overrides the latter one)
VerbosityLevel	No	Default	Default, Debug, Verbose, Info, Warning, Error, Fatal	Verbosity level
VarTransform	No	None	–	List of variable transformations performed before training, e.g., D_Background,P_Signal,G,N_AllClasses for: Decorrelation, PCA-transformation, Gaussianisation, Normalisation, each for the given class of events ('AllClasses' denotes all events of all classes, if no class indication is given, 'All' is assumed)
H	No	False	–	Print method-specific help message
CreateMVAPdfs	No	False	–	Create PDFs for classifier outputs (signal and background)
IgnoreNegWeightsInTraining	No	False	–	Events with negative weights are ignored in the training (but are included for testing and performance evaluation)
NTrees	No	800	–	Number of trees in the forest
MaxDepth	No	3	–	Max depth of the decision tree allowed
MinNodeSize	No	5%	–	Minimum percentage of training events required in a leaf node (default: Classification: 5%, Regression: 0.2%)
nCuts	No	20	–	Number of grid points in variable range used in finding optimal cut in node splitting
BoostType	No	AdaBoost	AdaBoost, RealAdaBoost, Bagging, AdaBoostR2, Grad	Boosting type for the trees in the forest

TMVA: Training Stage

AdaBoostR2Loss	No	Quadratic	Linear, Quadratic, Exponential	Type of Loss function in AdaBoostR2
UseBaggedBoost	No	False	–	Use only a random subsample of all events for growing the trees in each iteration.
Shrinkage	No	1	–	Learning rate for GradBoost algorithm
AdaBoostBeta	No	0.5	–	Learning rate for AdaBoost algorithm
UseRandomisedTrees	No	False	–	Determine at each node splitting the cut variable only as the best out of a random subset of variables (like in RandomForests)
UseNvars	No	2	–	Size of the subset of variables used with RandomisedTree option
UsePoissonNvars	No	True	–	Interpret UseNvars not as fixed number but as mean of a Possion distribution in each split with RandomisedTree option
BaggedSampleFraction	No	0.6	–	Relative size of bagged event sample to original size of the data sample (used whenever bagging is used (i.e. UseBaggedGrad, Bagging,)
UseYesNoLeaf	No	True	–	Use Sig or Bkg categories, or the purity=S/(S+B) as classification of the leaf node -> Real-AdaBoost
NegWeightTreatment	No	InverseBoostNegWeights	InverseBoostNegWeights, IgnoreNegWeightsInTraining, PairNegWeightsGlobal, Pray	How to treat events with negative weights in the BDT training (particular the boosting) : IgnoreInTraining; Boost With inverse boostweight; Pair events with negative and positive weights in traning sample and *annihilate* them (experimental!)
NodePurityLimit	No	0.5	–	In boosting/pruning, nodes with purity > NodePurityLimit are signal; background otherwise.

TMVA: Training Stage

SeparationType	No	GiniIndex	CrossEntropy, GiniIndex, GiniIndexWithLaplace, MisClassificationError, SDivSqrtSPlusB, RegressionVariance	Separation criterion for node splitting
DoBoostMonitor	No	False	–	Create control plot with ROC integral vs tree number
UseFisherCuts	No	False	–	Use multivariate splits using the Fisher criterion
MinLinCorrForFisher	No	0.8	–	The minimum linear correlation between two variables demanded for use in Fisher criterion in node splitting
UseExclusiveVars	No	False	–	Variables already used in fisher criterion are not anymore analysed individually for node splitting
DoPreselection	No	False	–	and and apply automatic pre-selection for 100% efficient signal (bkg) cuts prior to training
RenormByClass	No	False	–	Individually re-normalize each event class to the original size after boosting
SigToBkgFraction	No	1	–	Sig to Bkg ratio used in Training (similar to NodePurityLimit, which cannot be used in real adaboost
PruneMethod	No	NoPruning	NoPruning, ExpectedError, CostComplexity	Note: for BDTs use small trees (e.g.MaxDepth=3) and NoPruning: Pruning: Method used for pruning (removal) of statistically insignificant branches
PruneStrength	No	0	–	Pruning strength
PruningValFraction	No	0.5	–	Fraction of events to use for optimizing automatic pruning.
GradBaggingFraction	No	0.6	–	deprecated: Use *BaggedSampleFraction* instead: Defines the fraction of events to be used in each iteration, e.g. when UseBaggedGrad=kTRUE.
UseNTrainEvents	No	0	–	deprecated: Use *BaggedSampleFraction* instead: Number of randomly picked training events used in randomised (and bagged) trees

TMVA: Training Stage

Finalmente se entrena y prueba la BDT:
factory->**TrainAllMethods**();
factory->**TestAllMethods**();

Se evalúa y compara desempeño de métodos multivariados configurados
factory->**EvaluateAllMethods**();

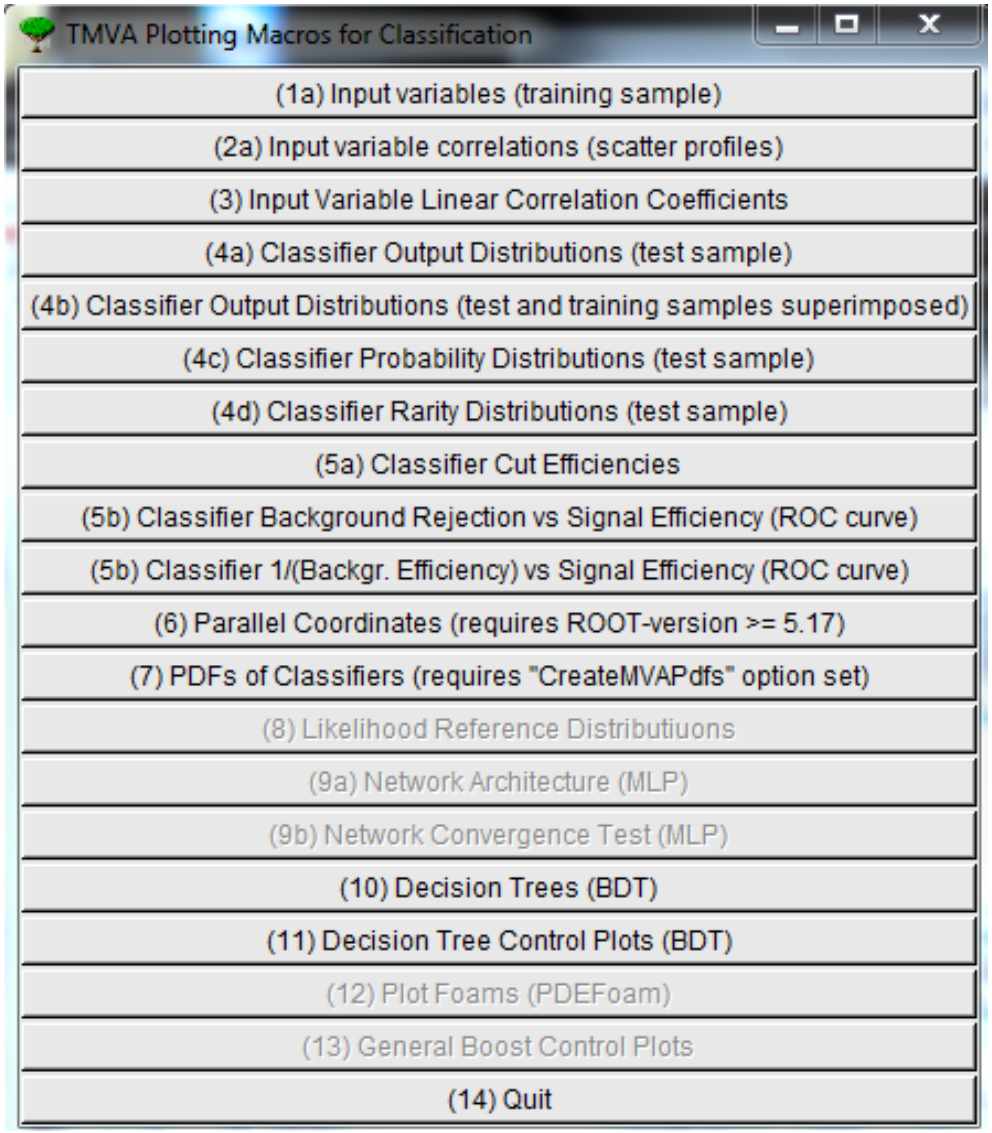
outputFile->Close();
delete factory;

Output importante de TMVA usados para aplicar BDT a datos (creados en carpeta weights):
TMVAClassification.weights.xml
TMVAClassification.class.C

Se pueden ver resultados del entrenamiento con:
TMVA::TMVAGui(outfileName);

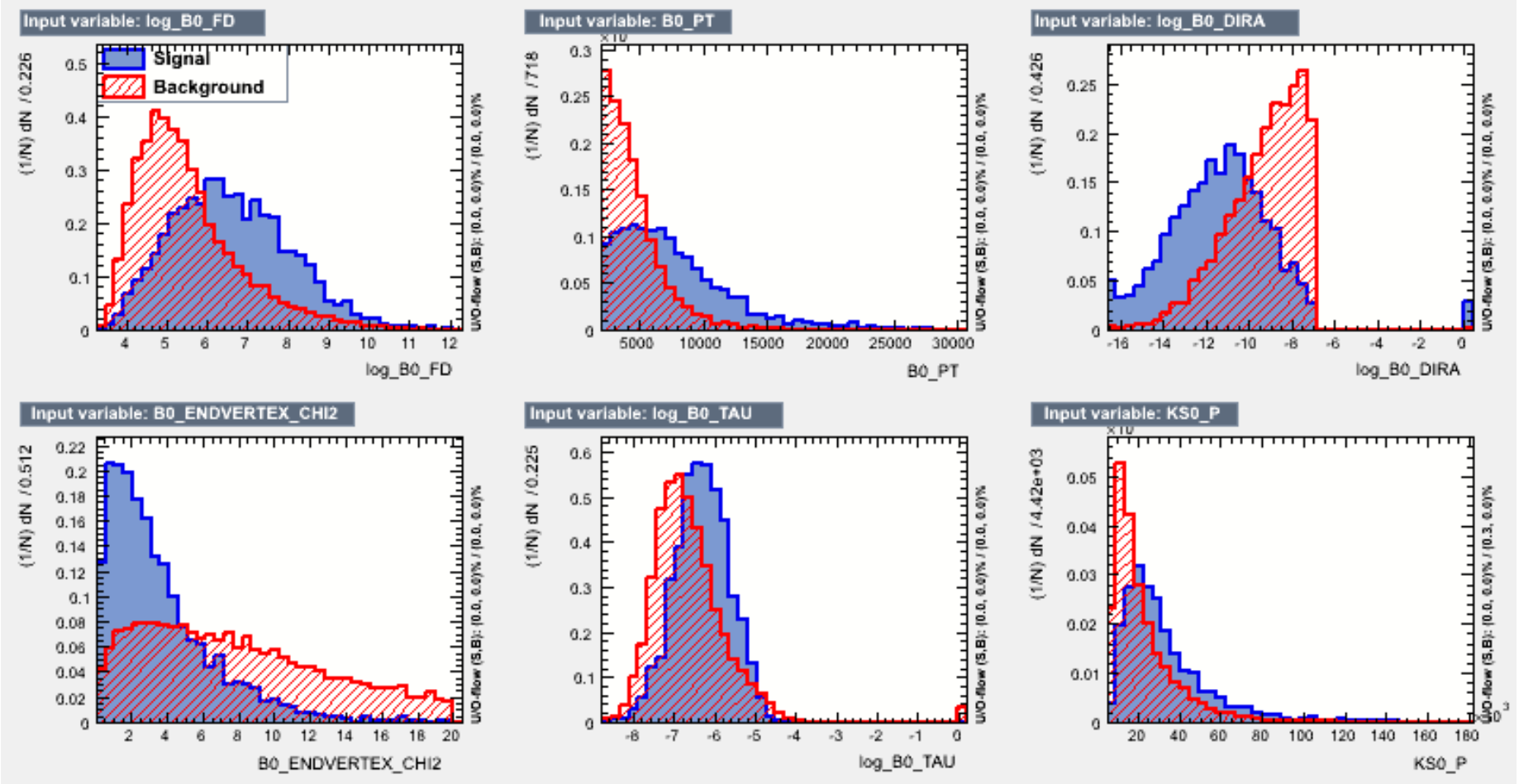
Para versiones anteriores (output: TMVA.root):
$ROOTSYS/test/TMVAGui.C

TMVA: Training Stage

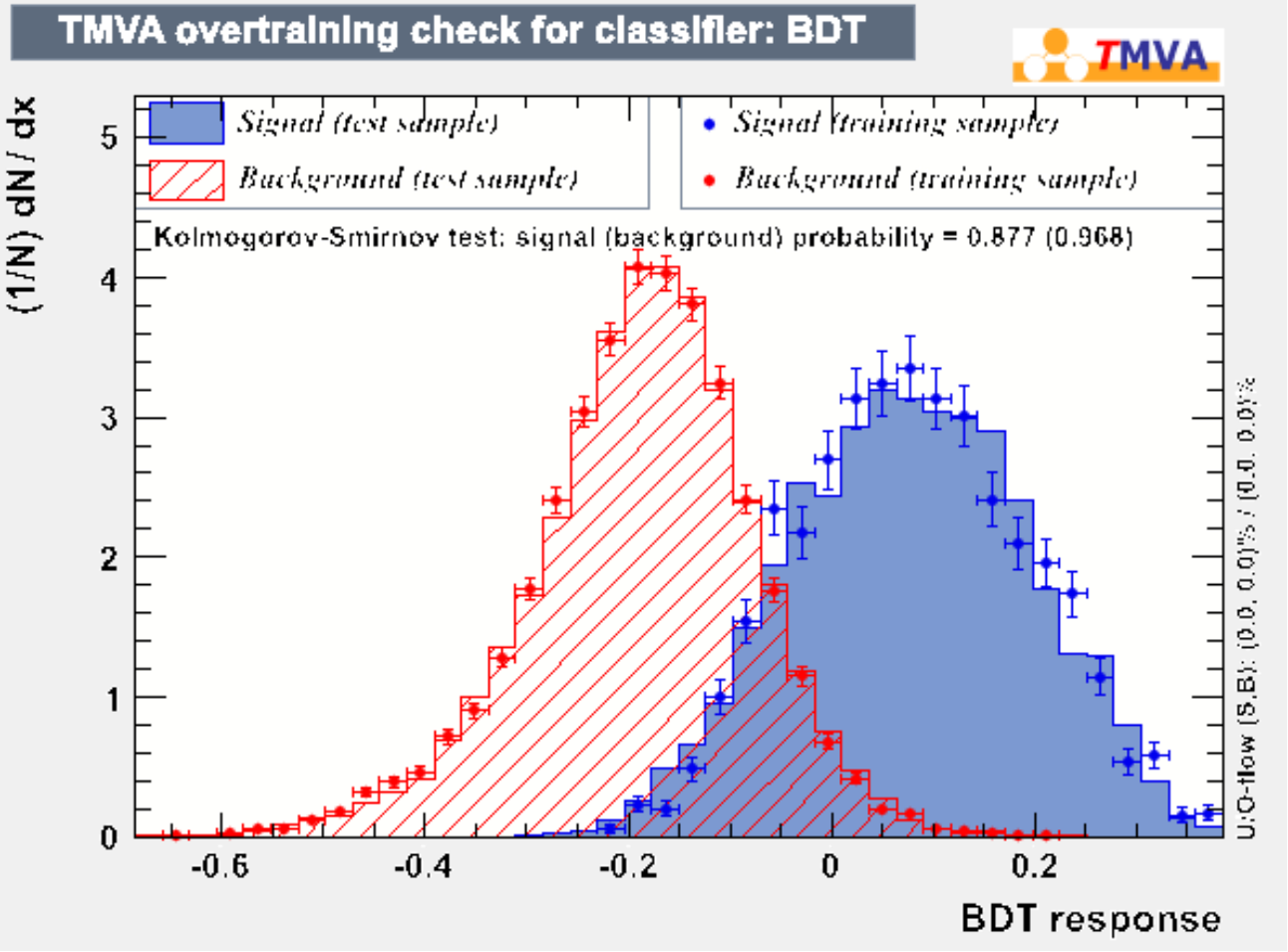


$ROOTSYS/test/TMVAGui.C

TMVA: Training Stage



TMVA: Training Stage

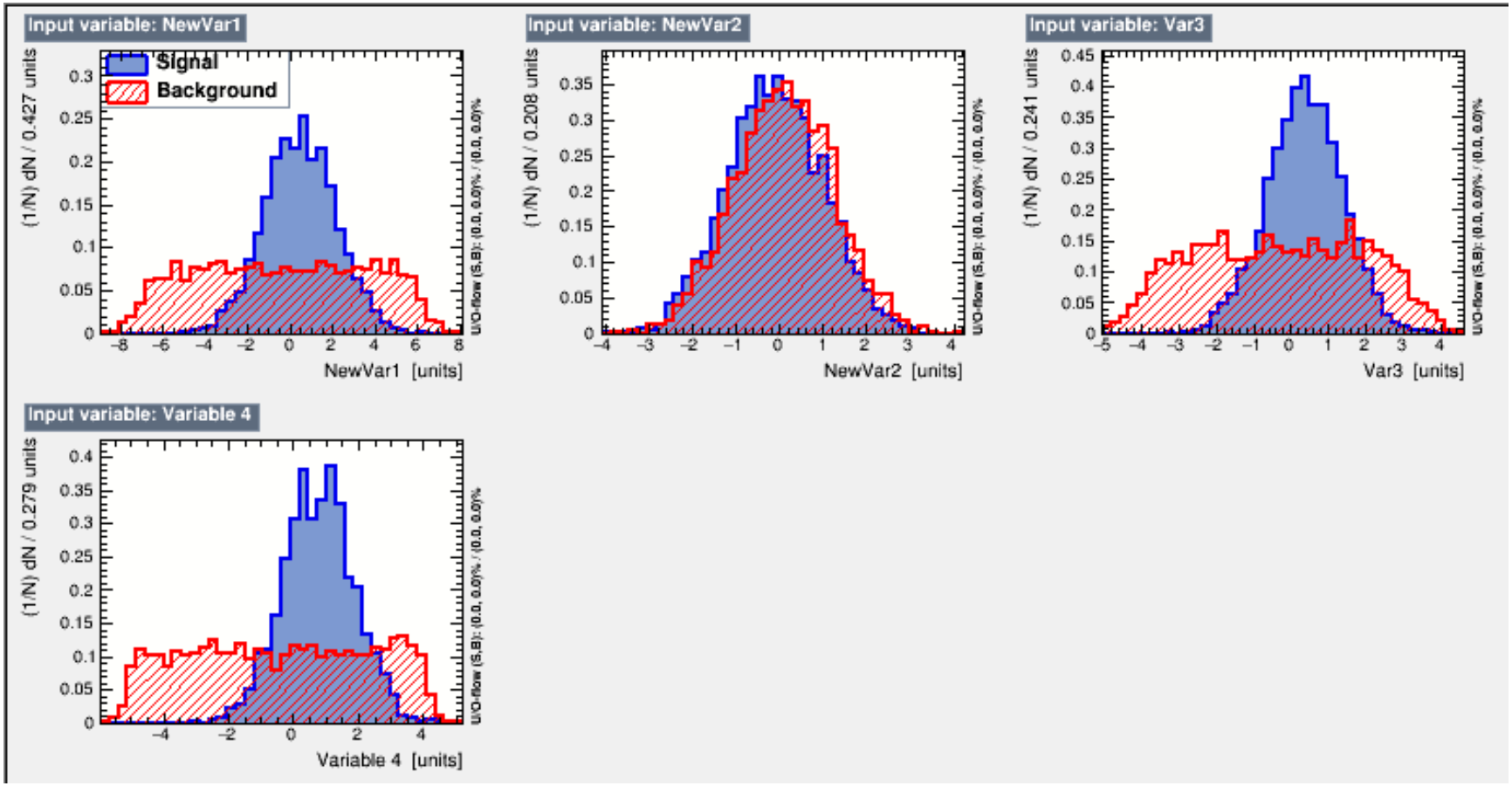


No hay overtraining

Respuesta BDT de muestras de test y training comparadas. Si no coinciden, hay overtraining -> no hay eventos suficientes para entrenamiento y BDT no es eficiente.

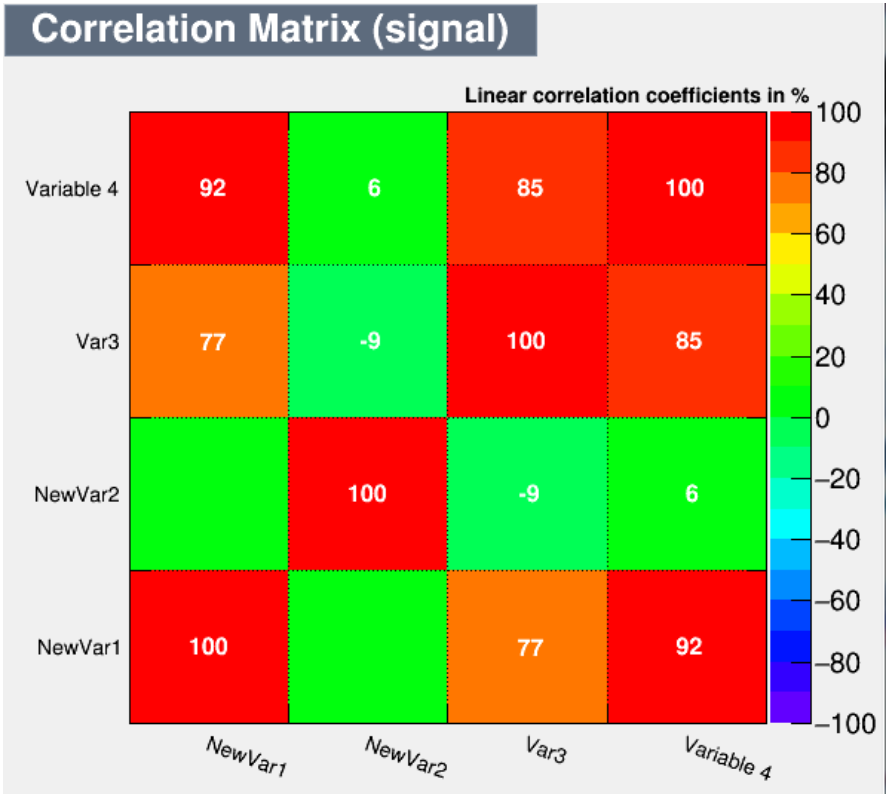
TMVA: Training Stage

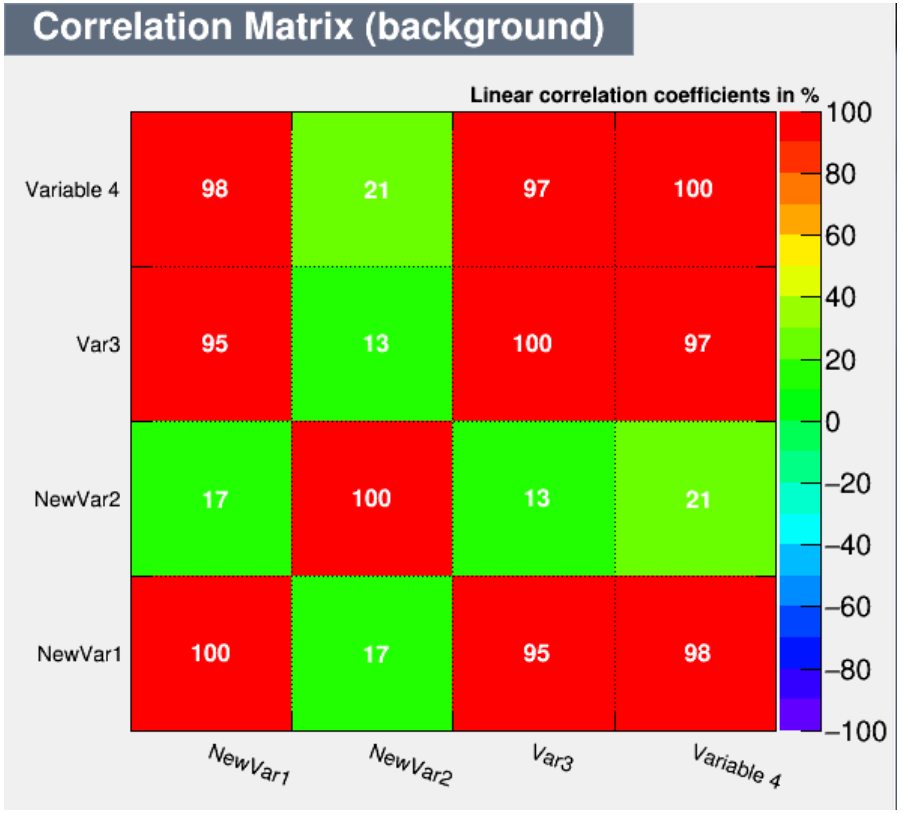
(1a) Input variables (training sample)



TMVA: Training Stage

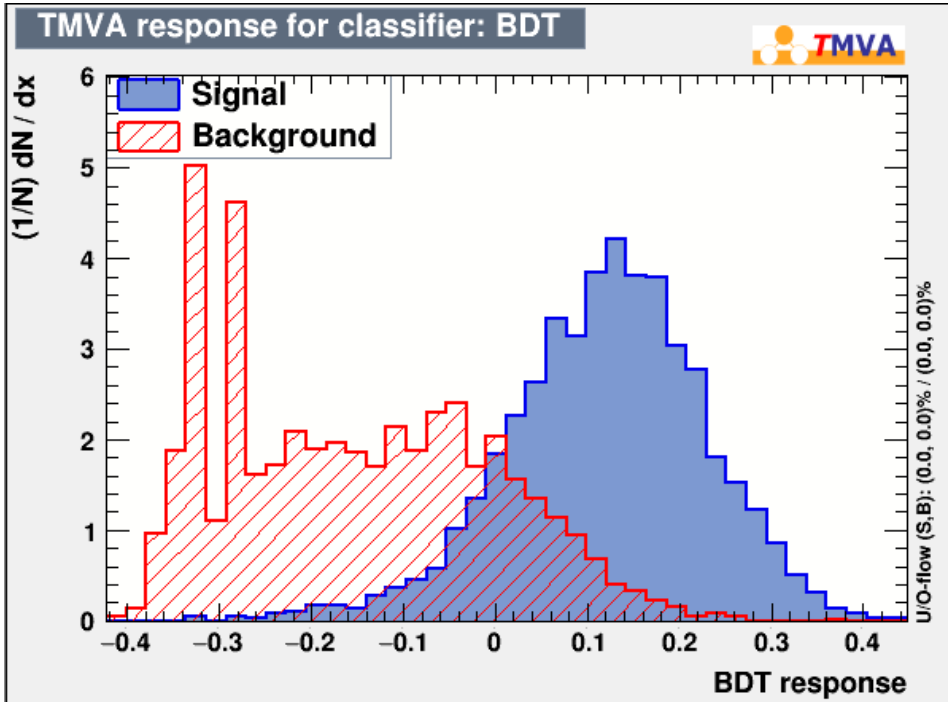
(3) Input Variable Linear Correlation Coefficients



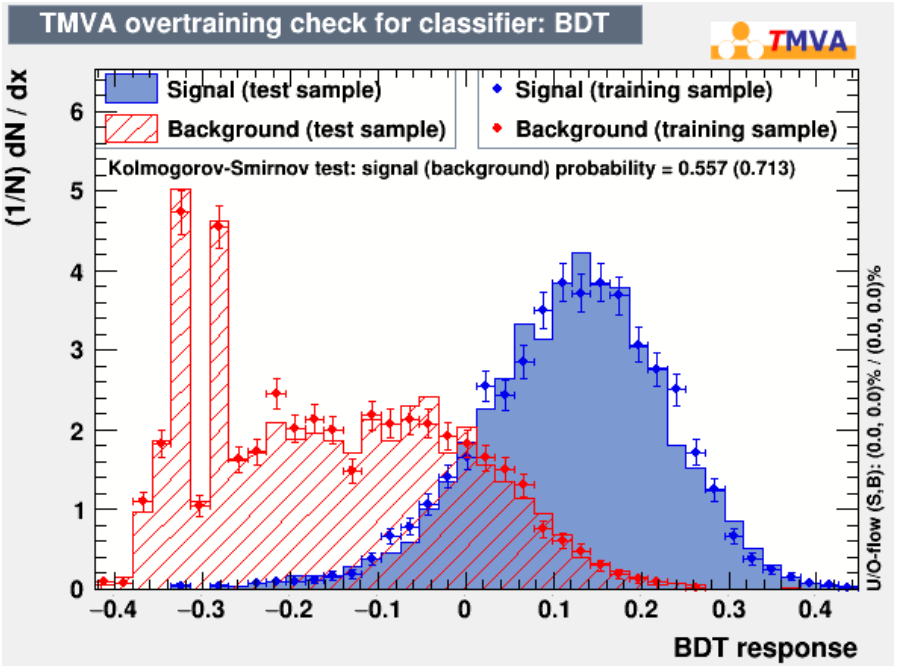


TMVA: Training Stage

(4a) Classifier Output Distributions (test sample)

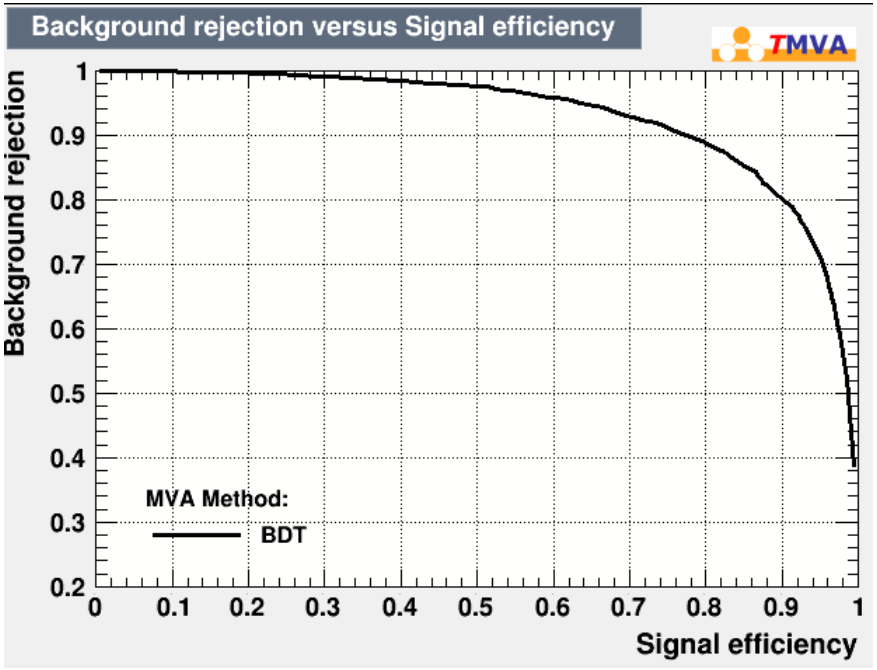


(4b) Classifier Output Distributions (test and training samples superimposed)

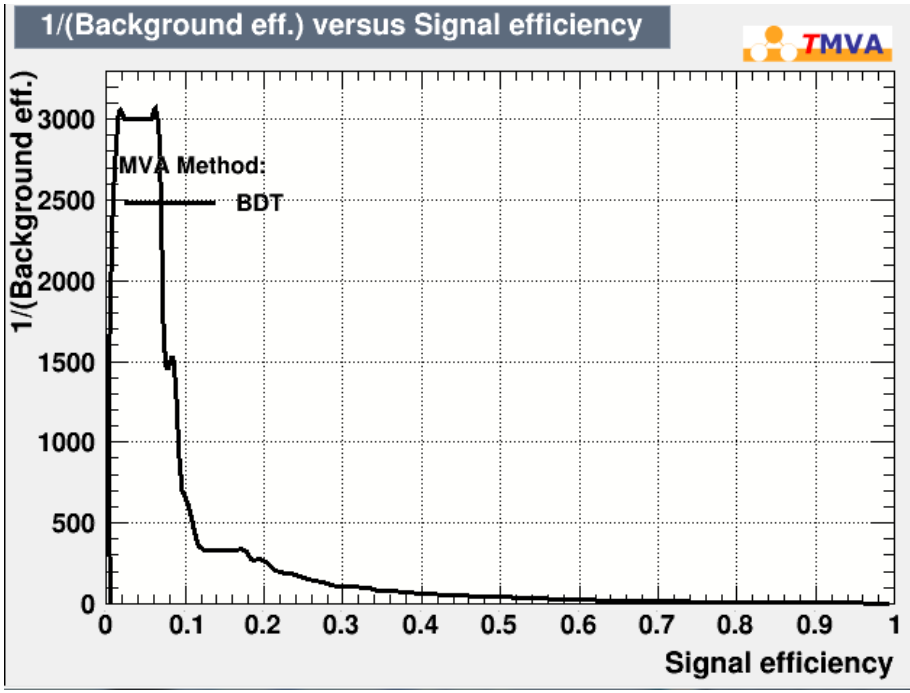


TMVA: Training Stage

(5b) Classifier Background Rejection vs Signal Efficiency (ROC curve)

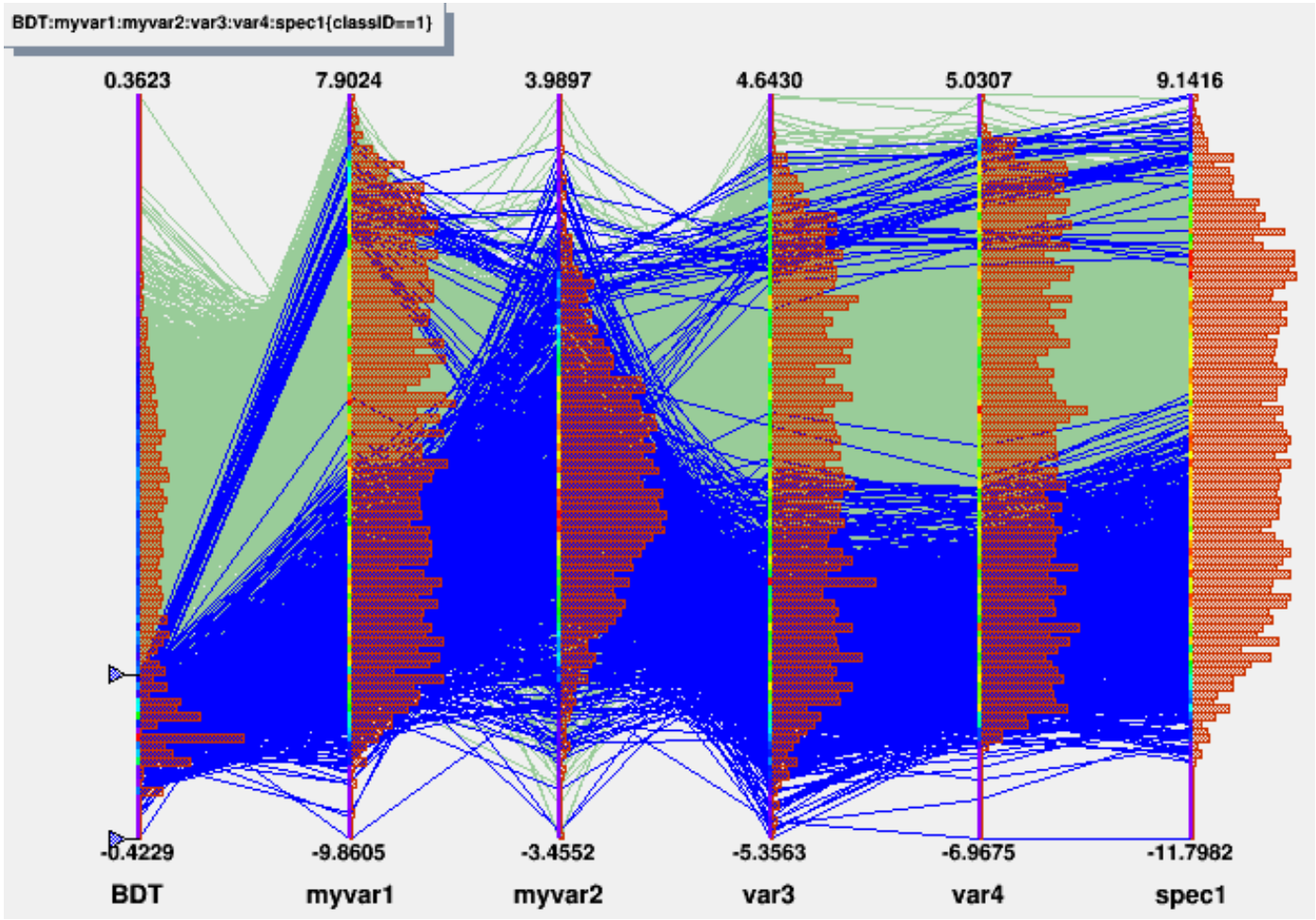


(5b) Classifier 1/(Backgr. Efficiency) vs Signal Efficiency (ROC curve)



TMVA: Training Stage

(6) Parallel Coordinates (requires ROOT-version >= 5.17)



TMVA: Training Stage

Optimizar configuración usando eventos de training (implementado en versiones recientes)

factory->**OptimizeAllMethods**("ROCIntegral","FitGA");

TMVA: Application Stage

Usar BDT, obtenida del entrenamiento con señal y fondo separados, en datos reales (mezcla de ambos)

Definir archivo de datos reales:

```
TFile* data = new TFile("real_data.root");
TTree* dataTree = (TTree*)(data->Get("tree"));
```

Crear archivo de salida con respuesta de BDT a cada evento:

```
TFile *target = new TFile("real_data-tmva_output.root","RECREATE" );
TTree *tree = new TTree("tree",“bdt tree");
```

Cargar librería y llamar a clase Reader:

```
TMVA::Tools::Instance();
TMVA::Reader *reader = new TMVA::Reader( "V:Color:!Silent" );
```

TMVA: Application Stage

Definir variables usadas en training y asignarlas a un arreglo.
Nombre y tipo iguales a archivo de pesos y definidas en el mismo orden

```
Float_t var[4];

reader->AddVariable(“NewVar1",&var[0]);
reader->AddVariable(“NewVar2",&var[1]);
reader->AddVariable(“var3",&var[2]);
reader->AddVariable(“var4",&var[3]);
```

Definir método usado e incluir archivo de pesos:

```
reader->BookMVA("BDT method", "weights/TMVAClassification.weights.xml");
```

TMVA: Application Stage

Definir variables en muestra de datos correspondientes a variables usadas en BDT.

```
Float_t userVar[4];

dataTree->SetBranchAddress(“var1",&userVar[0]);
dataTree->SetBranchAddress(“var2",&userVar[1]);
dataTree->SetBranchAddress(“var3”,&userVar[2]);
dataTree->SetBranchAddress(“var4”,&userVar[3]);
```

Añadir branch para salvar respuesta de BDT de cada evento:

```
Float_t BDT_response;
tree->Branch("BDT_response",&BDT_response);
```

Para agregar variables al archivo de salida:

```
Float_t var5;
dataTree->SetBranchAddress(“var5",&var5);
tree->Branch(“var5",&var5);
```

TMVA: Application Stage

Hacer un loop sobre eventos de muestra de datos y evaluar BDT.
Aquí se realizan operaciones matemáticas con variables.

```
for (Long64_t ievt=0; ievt<dataTree->GetEntries();ievt++) {

   if (ievt%100000 == 0) std::cout << "--- ... Processing event: " << ievt <<std::endl;
   dataTree->GetEntry(ievt);

   var[0]=userVar[0]+userVar[1] ;
   var[1]=userVar[0]-userVar[1] ;
   var[2]=userVar[2];
   var[3]=userVar[3];

   BDT_response=reader->EvaluateMVA("BDT method");

   tree->Fill();
  }

tree->Write();
```