# Statistische Methoden der Datenanalyse

# Wintersemester 2011/2012

# Albert-Ludwigs-Universität Freiburg

Prof. Markus Schumacher

Physikalisches Institut Westbau 2 OG Raum 008

Telefonnummer 07621 203 7612

E-Mail: Markus.Schumacher@physik.uni-freiburg.de

Vorlesung basiert in weiten Teilen auf der von Glen Cowan.

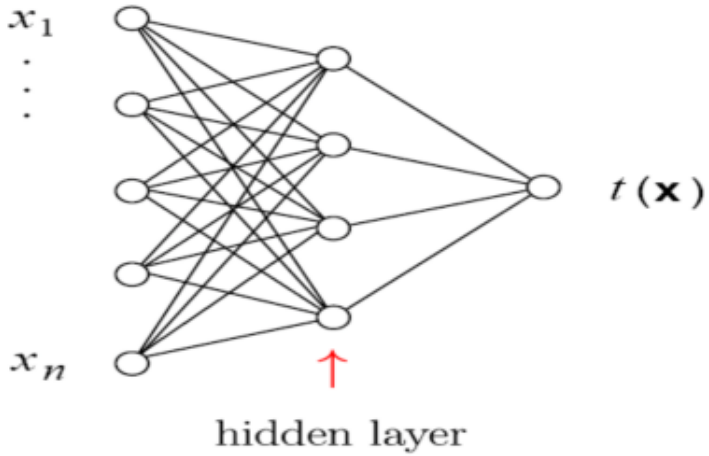
http://terascale.physik.uni-freiburg.de/lehre/ws_1112/statmethoden_ws1112

# Kapitel 8

# Ereignisklassifizierung

# Das Multilagen-Perzeptron

Verallgemeinerung auf mehrlagiges Perzeptron:




Die Werte der "Neuronen" in der versteckten Lage sind:

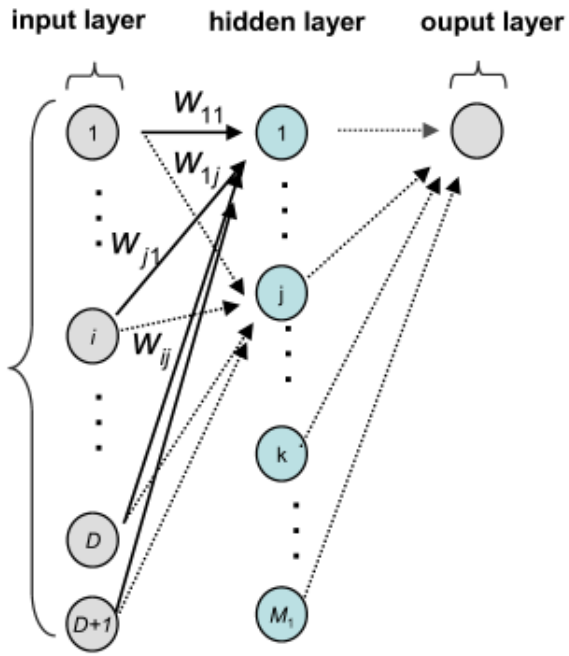
$$h_i(\vec{x}) = s\left(w_{i0} + \sum_{j=1}^{n} w_{ij} x_j\right) ,$$

Der Ausgangswert/Output des Netzwerks ist gegeben durch:

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

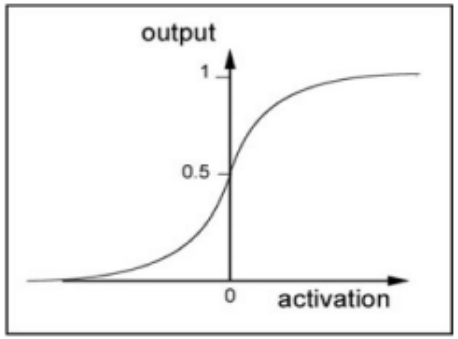
$a_i, w_{ij} =$ Gewichte der Synapsen (Verbinndungsstärken)

# Das Multilagen-Perzeptron




$D_{var}$ discriminating input variables as input + 1 offset

**output:** $$y(x)=\sum_{i}^{M} w_{0i} A\left( w_{i0} + \sum_{j=1}^{D} w_{ij} \cdot x_j \right)$$




"Activation" function e.g. sigmoid:

$$A(x) = \left(1 + e^{-x}\right)^{-1}$$

or tanh

Bestimmung der optimalen Gewichte im Trainnigsprozess

Nur Vorwärtspropagation der Information ("Feed Forward Network")

Es könne auch mehrere versteckte Lagen vewendet werden.
In der Praxis maximal 2. Theorie sagt, dass 1 Lage im Prinzip schon ausreicht.

# Training des Multilagen-Perzeptron

Für jedes Ereignis a kennen wir Observablenwerte und Zielwert des Output

$$\vec{x}_a = (x_1, \ldots, x_n) \qquad t_a = 0, 1$$

Sei ***w*** der Vektor aller anzupassenden Gewichte

Bestimmung der optimalen Gewichte durch Minimierung der Fehlerfunktion:

$$E(\boldsymbol{w}) = \frac{1}{2} \sum_{a=1}^{N} |y(\vec{x}_a, \boldsymbol{w}) - t_a|^2 = \sum_{a=1}^{N} E_a(\boldsymbol{w})$$

Im Prinzip: auf allen Ereignissen auswerten und versuchen einen Multiparameterfit durchzuführen.
Liefert in Praxis kein gutes Ergebnis (sehr viele Nebenminima)

Alternative: Online-Lernen = Update der Gewichte nach Auswertung jedes Ereignisses

# Training des Multilagen-Perzeptron

Zu Beginn: zufällige Zuweisung von Gewichten an Synapsen

Nach Prüfung der Ereignisse werden die Gewichte geändert gemäß:

$$\boldsymbol{w}^{(\tau+1)} = \boldsymbol{w}^{(\tau)} - \eta \nabla E\left(\boldsymbol{w}^{(\tau)}\right)$$

Lernrate $\eta > 0$

Erstaunlich schlechte Leistungsgüte wenn alle Ereignisse berücksichtigt werden.

In Praxis besser: Anpassung der Gewichte nach jedem Ereignis a:

$$\boldsymbol{w}^{(\tau+1)} = \boldsymbol{w}^{(\tau)} - \eta \nabla E_a\left(\boldsymbol{w}^{(\tau)}\right)$$

# Fehlerrückpropagation

Fehlerrückpropagation = Algorithmus zum Auffinden des Gradientenvektors

Die Netzwerkantwort kann geschieben werden als: $y(\boldsymbol{x}) = h(u(\boldsymbol{x}))$

wobei: $u(\vec{x}) = \sum_{j=0} w_{1j}^{(2)} \varphi_j(\vec{x})$ $\qquad \varphi_j(\vec{x}) = h\left(\sum_{k=0} w_{jk}^{(1)} x_k\right)$

Mit $\phi_0 = x_0 = 1$ die Summe erstreckt sich über alle Knoten/Neuronen in der vorherige Lage (+offset w0)

Zunächst Update der Synapsen zwischen versteckter Lage und Ausgang z.B. für Ereignis a:

$$\frac{\partial E_a}{\partial w_{1j}^{(2)}} = (y_a - t_a)\, h'(u(\vec{x}))\, \varphi_j(\vec{x})$$

Dann Update der Synapsen zwischen Einganglage und versteckter Lage (mit Kettenregel)

# Fehlerrückpropagation: Explizit

n Eingangsgrößen, eine versteckte Lage mit m Knoten

Netzausgang:

$$y(\vec{x}) = h\left(w_0 + \sum_{j=1}^{m} w_j \, \phi_j(\vec{x})\right)$$

versteckte Knoten:

$$\phi_j(\vec{x}) = h\left(w_{j0} + \sum_{i=1}^{n} w_{ji} \, x_i\right)$$

Anregungsfkt. und deren Ableitung:

$$h(u) = \frac{1}{1+e^{-u}} \qquad \frac{dh}{du} = \frac{e^{-u}}{(e^{-u}+1)^2} = h(1-h)$$

Update der Gewichte der Synapsen zwischen versteckter Lage und dem Ausgangsknoten

$$\frac{\partial E}{\partial w_j} = \left(y(\vec{x}) - t\right) \frac{dy}{\partial u_1} \frac{\partial u_1}{\partial w_j}$$

$$= \quad '' \quad y(1-y) \; \phi_j(\vec{x})$$

$$w_j^{\tau+1} = w_j^{\tau} - \alpha \, (y-t) \, y(1-y) \, \phi_j(\vec{x})$$

# Fehlerrückpropagation: Explizit (2)

n Eingangsgrößen, eine versteckte Lage mit m Knoten

Netzausgang:

$$y(\vec{x}) = h\left(w_0 + \sum_{j=1}^{m} w_j \, \phi_j(\vec{x})\right)$$

versteckte Knoten:

$$\phi_j(\vec{x}) = h\left(w_{j0} + \sum_{i=1}^{n} w_{ji} \, x_i\right)$$

Anregungsfkt. und deren Ableitung:

$$h(u) = \frac{1}{1+e^{-u}} \qquad \frac{dh}{du} = \frac{e^{-u}}{(e^{-u}+1)^2} = h(1-h)$$

Update der Gewichte der Synapsen zwischen Eingansknoten und versteckter Lage. Neue Gewichte aus dem ersten Schritt.

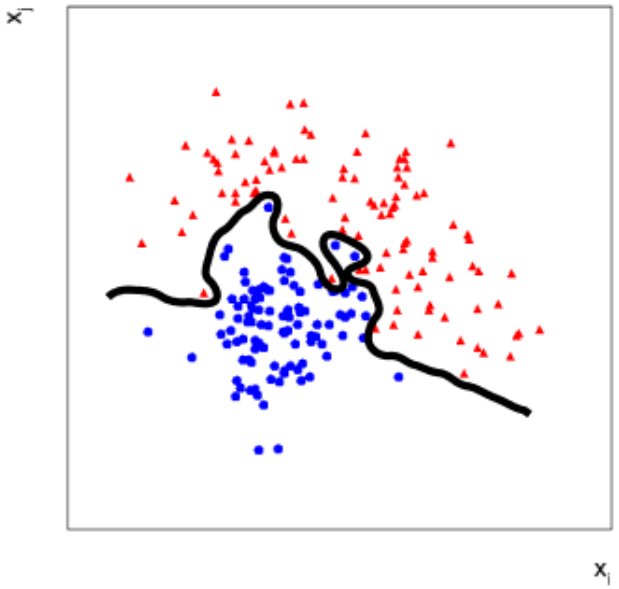
$$\frac{\partial E}{\partial w_{jk}} = (y(\vec{x})-t)\,\frac{\partial y}{\partial u_1}\,\frac{\partial u_1}{\partial \phi_j}\,\frac{\partial \phi_j}{\partial u_2}\,\frac{\partial u_2}{\partial w_{jk}}$$

$$= y(1-y)\, w_j^{\tau+1}\, \phi_j(1-\phi_j)\, x_k$$

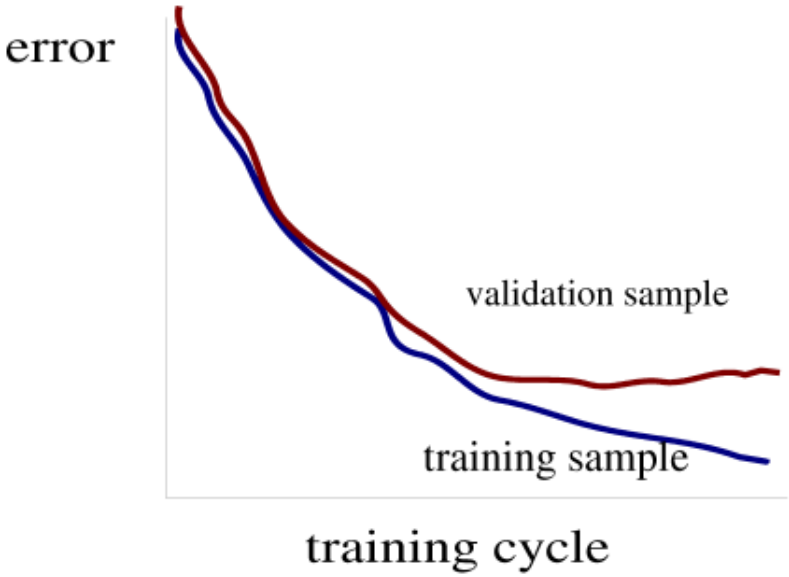
$$w_{jk}^{\tau+1} = w_{jk}^{\tau+1} - \alpha\,(y-t)\,y(1-y)\,w_j^{\tau+1}\,\phi_j(1-\phi_j)\,x_k$$

# Übertraining

Falls Netzwerk zu viele Knoten besitzt, und zu lange trainiert wird, wird sich Entscheidungsgrenze zu sehr an statistische Fluktuationen in Trainigsdatensatz anpassen:




Monitoring des Training:




Überprüfe Fehlerrate mit Hilfe von statistisch unabhängigen Ereignissen, die nicht im Training verwendet werden.

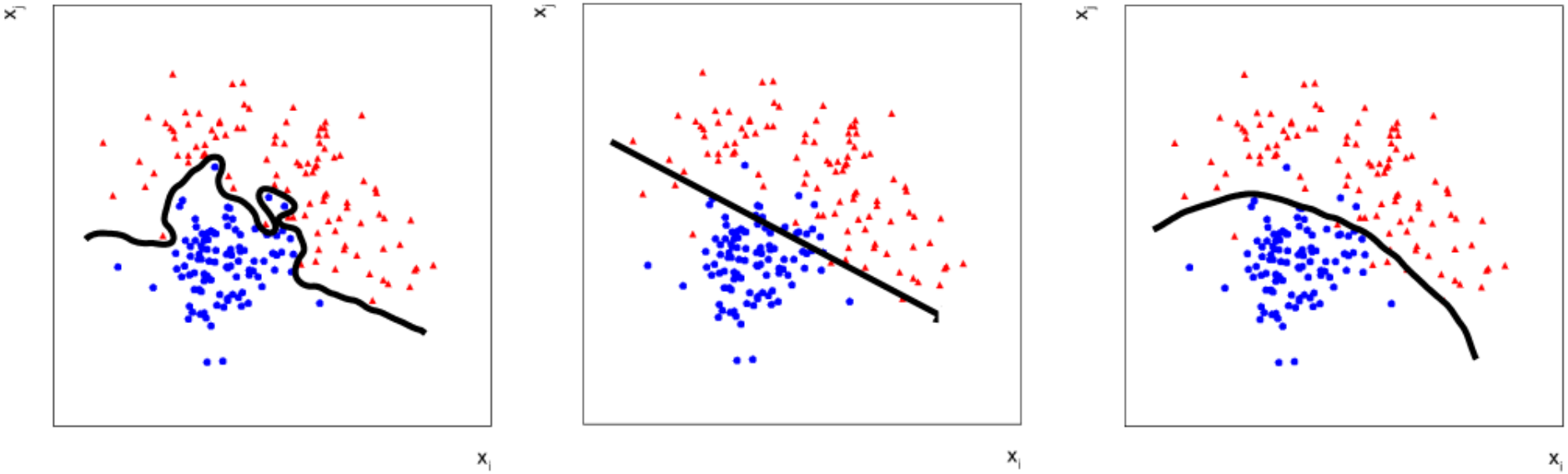
Beende Training, wenn beide Kurven sich voneinander trennen oder die Fehlerrate im Validierungssample wieder steigt.

# “Trade Off” zwischen “Bias” und “Varianz”

Für endliche Anzahl an Trainigsereignissen und steigender Anzahl an Neuronen haben Gewichte große stat.Fehler (→ großer Varianz, Übertraining)

Für zu kleine Anzahl von Neuronen kann das Netwerk nicht ausreichend die Nichtlinearitäten berücksichtigen (→ Bias)

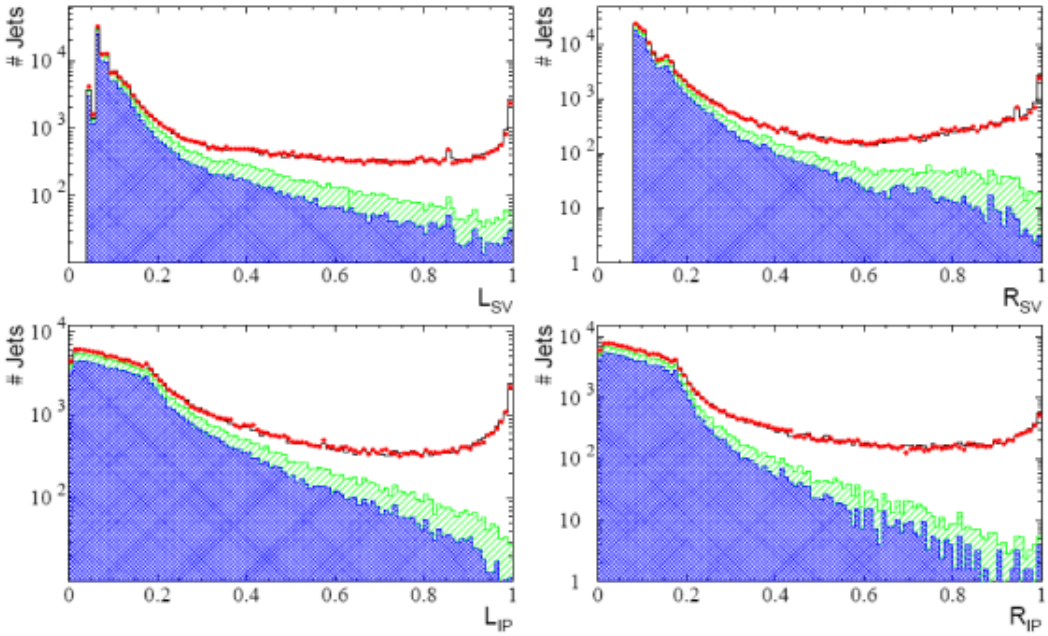
Bias und Varianz sind wiederrum gegenläufige Kriterien.



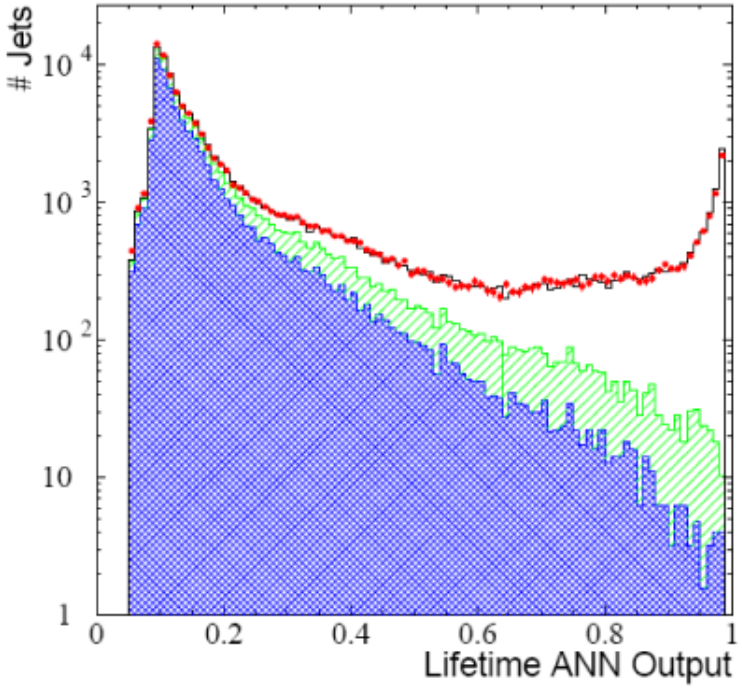

high variance  high bias  good trade-off

# “Beispiel von ANN bei LEP2 (OPAL)

ANN input variables




ANN output



# Struktur des Netzwerkes

**Theorem**: An MLP with a single hidden layer having a sufficiently large number of nodes can approximate arbitrarily well the Bayes optimal decision boundary. K. Weierstrass-Theorem

Holds for any continuous non-polynomial activation function

Leshno, Lin, Pinkus and Schocken (1993), *Neural Networks* **6**, 861—867

In practice often choose a single hidden layer and try increasing the the number of nodes until no further improvement in performance is found.

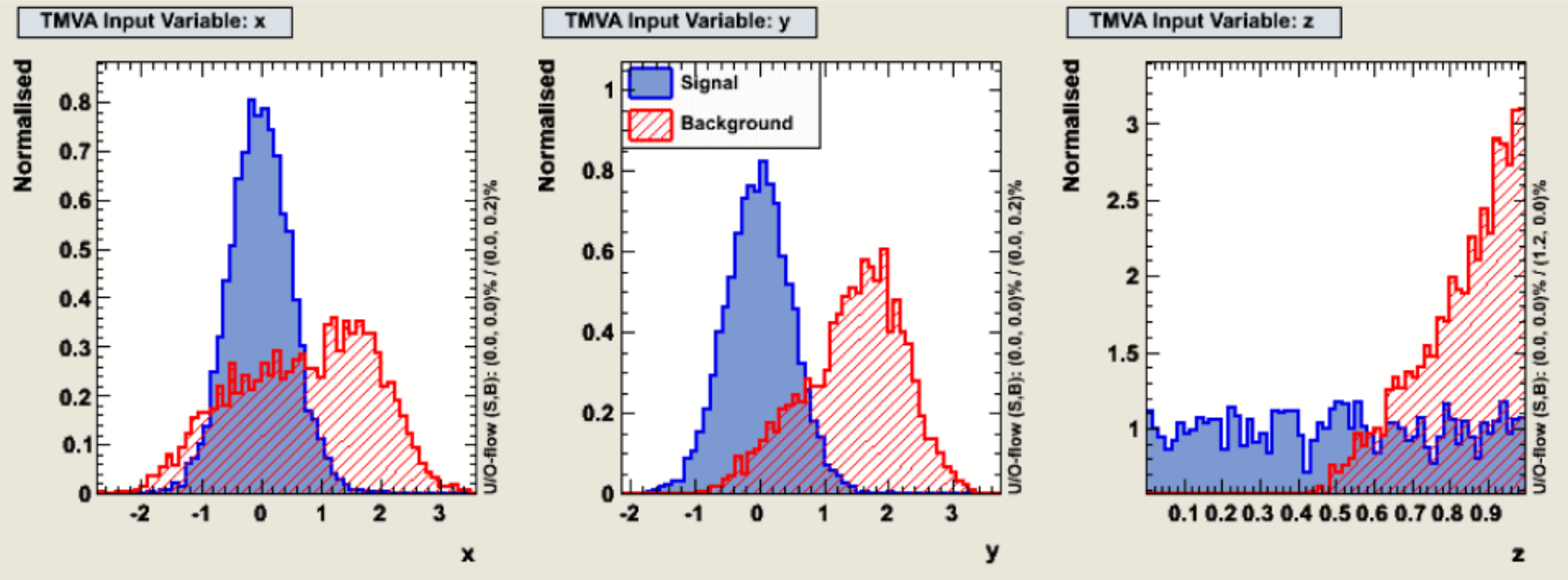
# Struktur des Netzwerkes

"Relatively little is known concerning the advantages and disadvantages of using a single hidden layer with many units (neurons) over many hidden layers with fewer units. The mathematics and approximation theory of the MLP model with more than one hidden layer is not well understood."

"Nonetheless there seems to be reason to conjecture that the two hidden layer model may be significantly more promising than the single hidden layer model, ..."

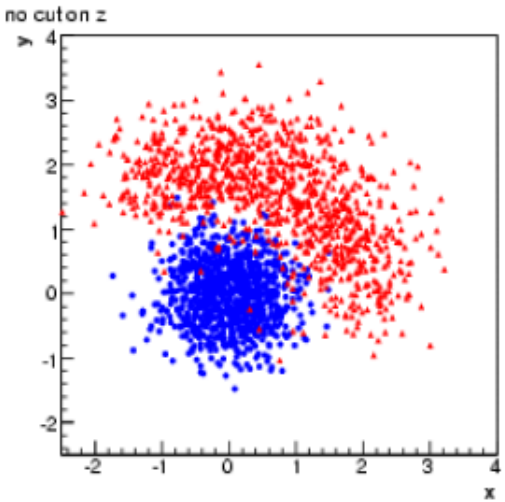
A. Pinkus, *Approximation theory of the MLP model in neural networks*, Acta Numerica (1999), pp. 143—195.

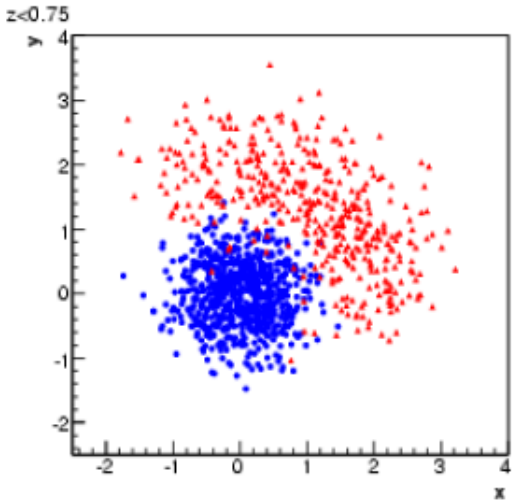
# Ein vergleichendes Beispiel



# Ein vergleichendes Beispiel

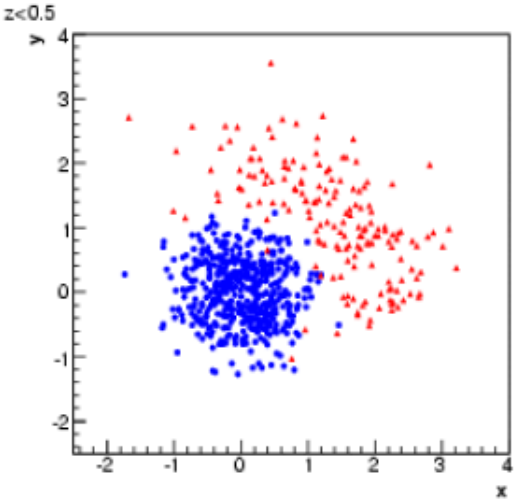
no cut on $z$

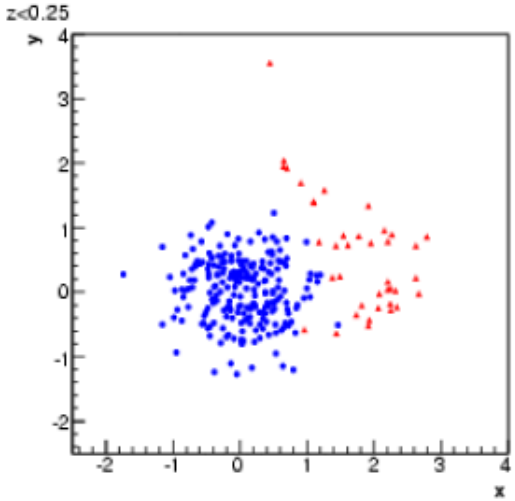



$z < 0.75$

$z < 0.5$

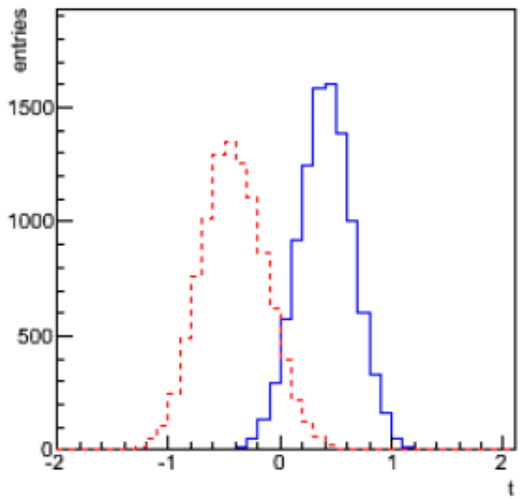


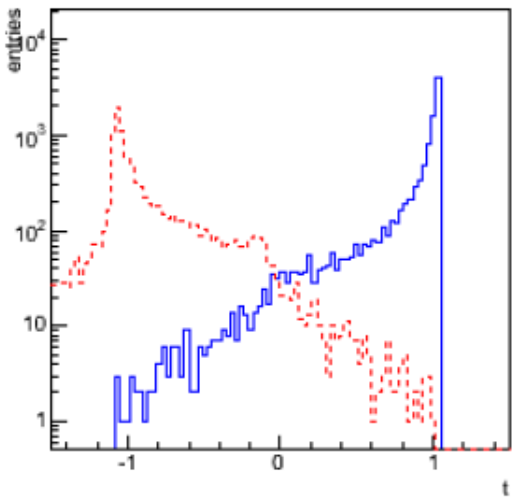




$z < 0.25$

# Ein vergleichendes Beispiel

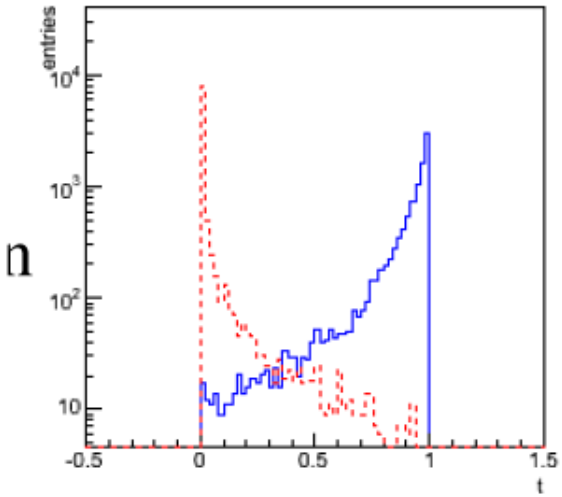
Fisher

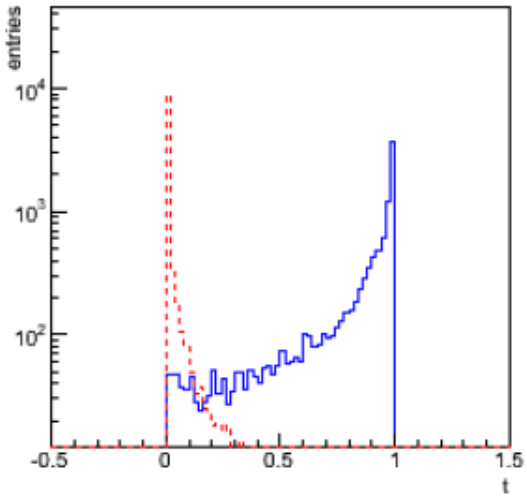



ANN

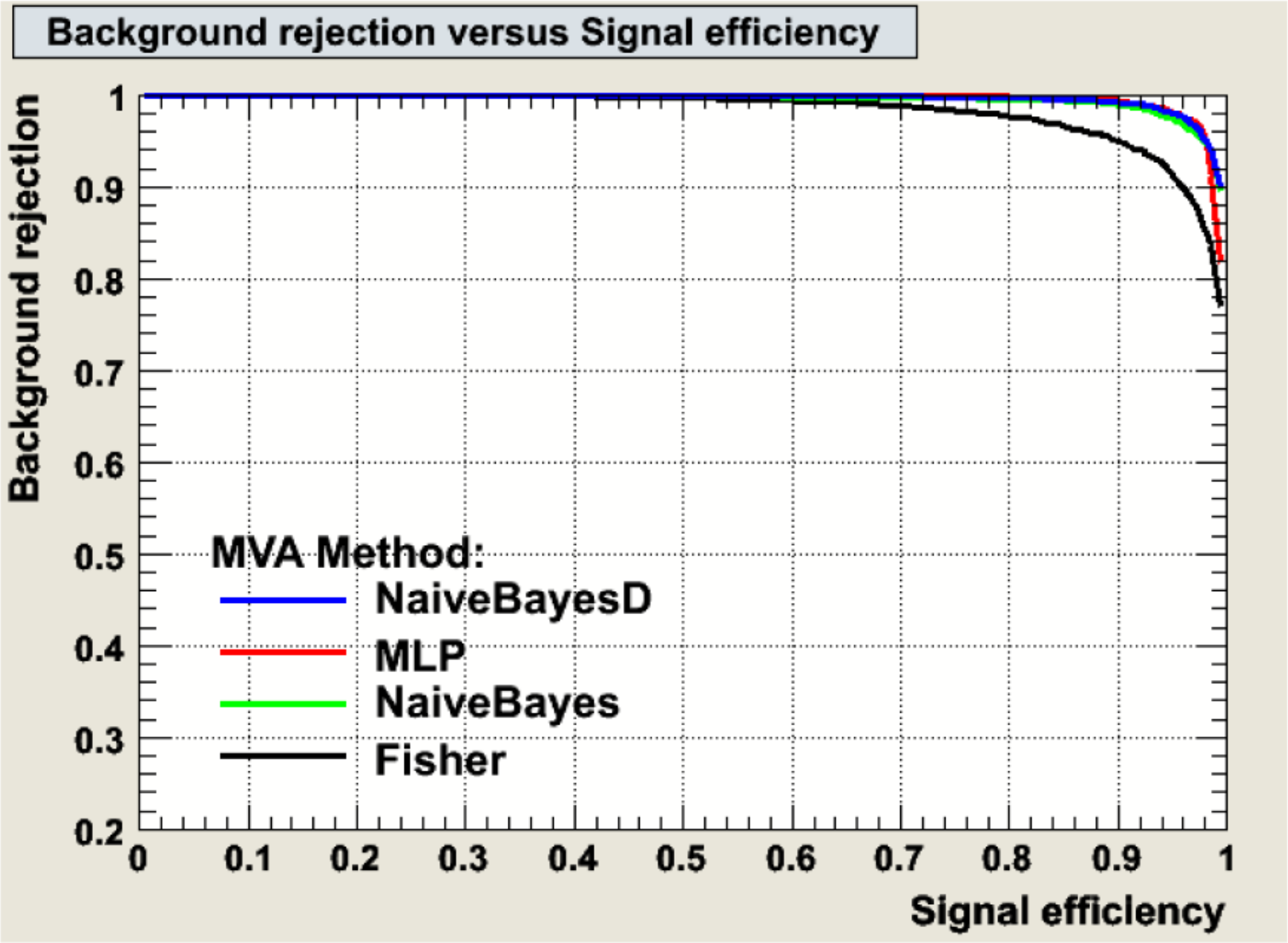
Likelihood ohne Dekorrelation



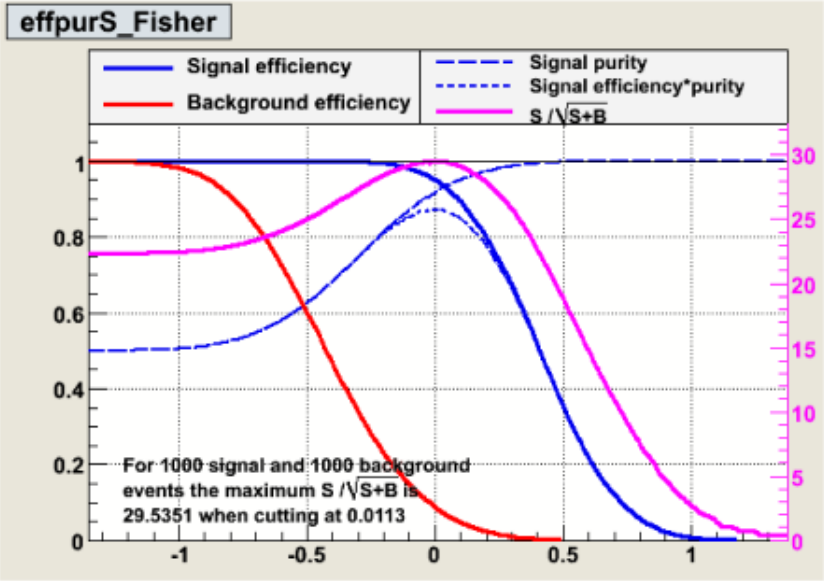



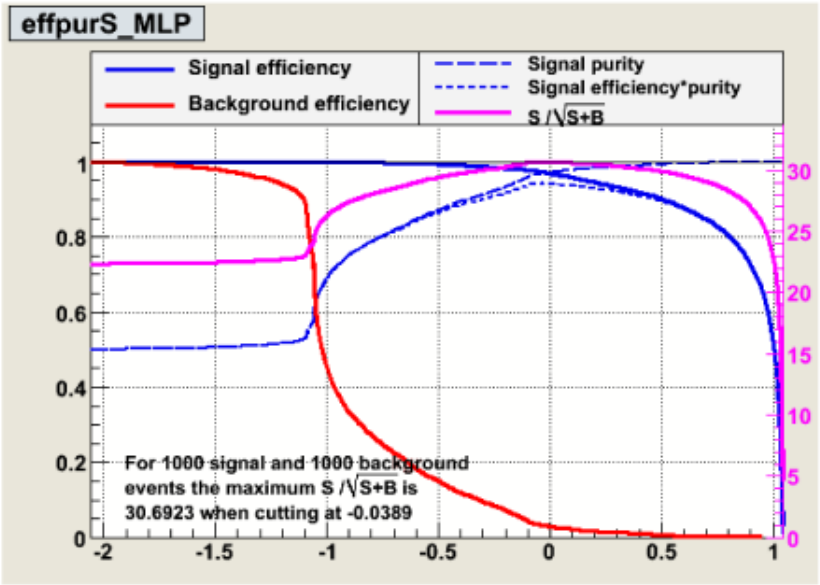

Likelihood mit Dekorrelation

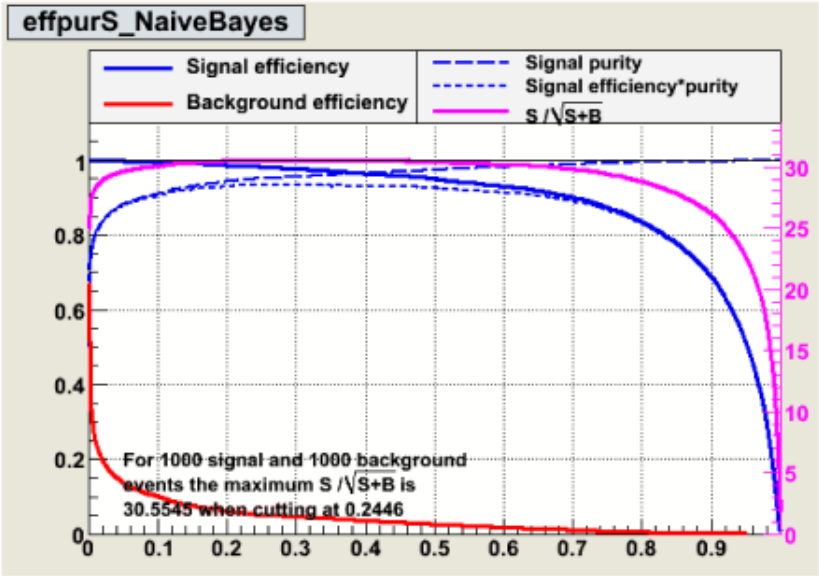
# Ein vergleichendes Beispiel

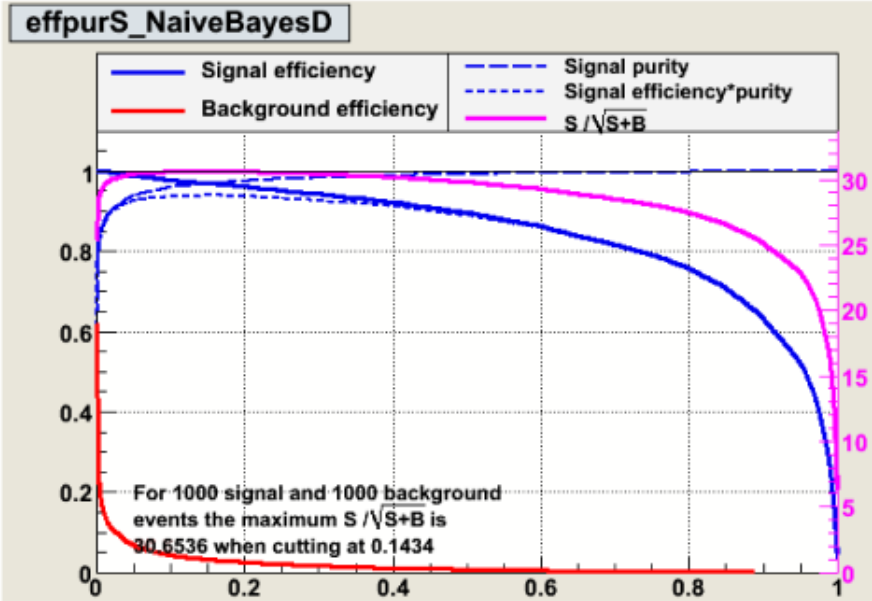


# Ein vergleichendes Beispiel



# Entscheidungsbäume

Idee: zerlege den n-dimensionalen Observablenraum in viele kleine n-dimensionale Quader, die jeweils als Akzeptanz- oder Verwerfungsregion für Signal klassifiziert werden

Prinzip: multiple Ketten von konsekutiven Schnitte (seit langem bekannt ausserhalb Physik)

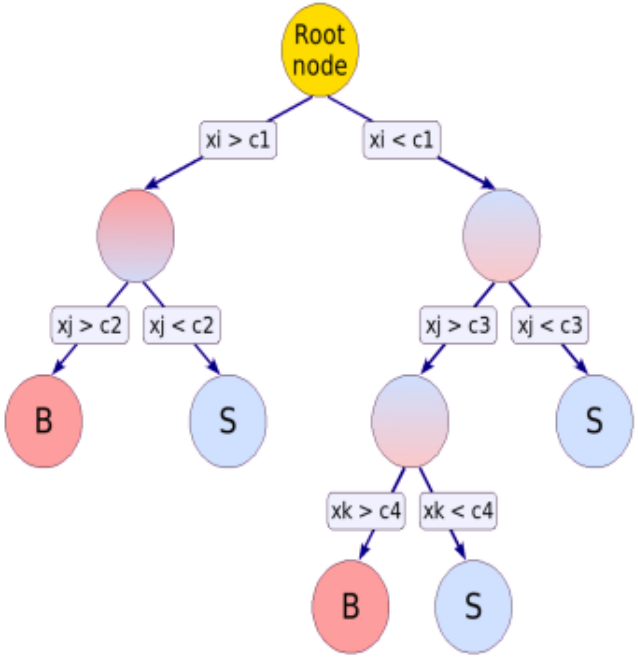
Vorteile:
- einfache Interpretation, Visualisierung
- unabhängig von montonen Transformationen der Observablen
- Variablen ohne Trennkraft werden ignoriert, keine negativer Einfluss

Nachteil:
- sehr sensitiv auf statistischen Fluktuationen im Trainigssample

→ Verstärkte Entscheidungsbäume (Boosted Decision Trees) (1996)
- kombiniere einen ganz “Wald “von Entscheidungsbäumen
- Stabilitätsproblem (Anfälligkeit gegen stat. Fluktuationen) behoben (erste Anwedung in HEP 2005 MiniBooNe Experiment)

einfach, aber nicht optimal aus theoretischer Sichtweise

# Entscheidungsbäume




Eine Observable wird gewählt.
Ein Schnitt auf diese spaltet den
Datensatz in zwei Untermengen (Knoten)

Für jeden Knoten wird die Prozedur wiederholt.
In den unterschiedlichen Ästen kann
die nächste Observable unterschiedlich sein.

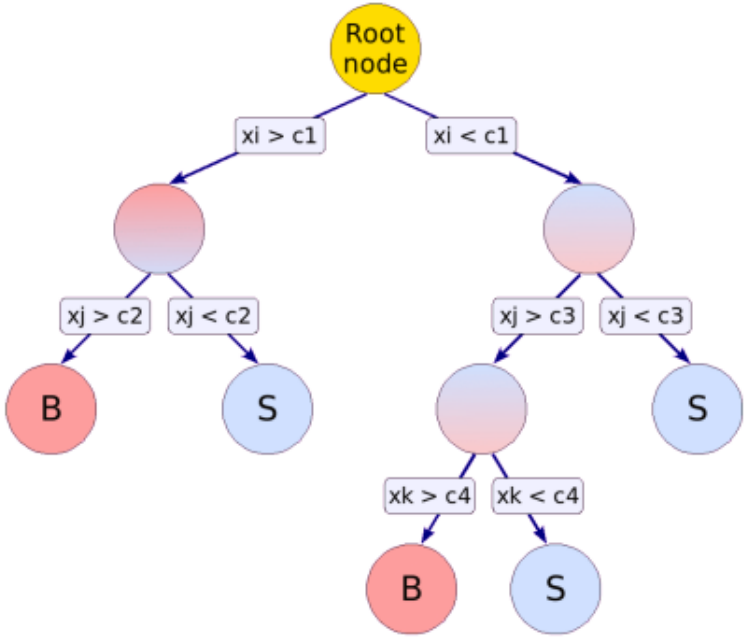
Observable können mehrmals auftauchen.

Aufspaltung endet gemäß Abruchkrriterium.

Endknoten = Blätter werden als Signal +1
oder Untergrund -1 klassifiziert.

Fragen: Auswahl der Observablen? Bestimmung des Schnittwertes?
Abruchkrtierum für Wachstum?
Klassifitierungskriterium für Blätter?

# Entscheidungsbäume




Auswahl der Observablen?
Bestimmung des Schnittwertes?

wähle beide, so dass
Separationsgewinn maximiert wird
(Details gleich)

Abbruchkritierium für Wachstum?
- Minimalanzahl von Ereignissen pro Blatt
- maximale Anzahl von Knoten
- maximale Tiefe
- jede weitere Aufspaltung ergibt keinen Wert über geforderten minimalen Separationsgewinn

Klassifizierungskriterium für Blätter?
- Mehrheit der Ereignisse
- vorgegegebener Minimalwert für S/B um als Signalblatt zu gelten

# Auffinden des besten individuellen Schnitts

Die Trennkraft in einem Knoten (Auswahl an Observable und Schnittwert) can z.B durch den *Gini-Koeffizienten* , bewertet werden, der sich aus der Reinheit (für Signal oder Untergrund) berechnet wird gemäß:

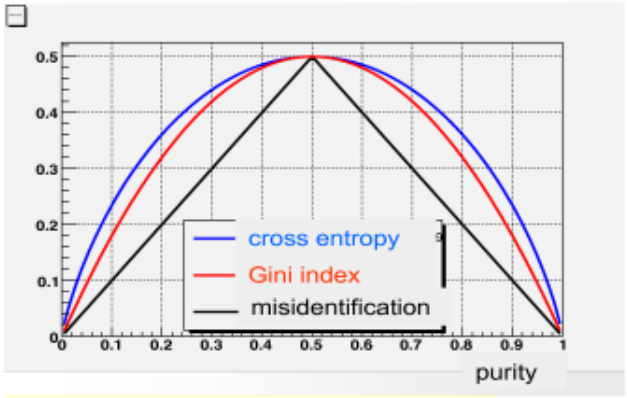
$$G = p(1-p)$$

Für einen Schnitt der eine Menge von Ereignissen a in zwei Untermengen b und c atiufteilt, kann die Verbesserung der Trennkraft durch die Änderung in den gewichteten Gini-Koeeffizienten quanitfiziert werden:

$$\Delta = W_a G_a - W_b G_b - W_c G_c \quad \text{wobei} \quad W_a = \sum_{i \in a} w_i$$

Wähle die Observable und den Schnittwert der *D*; maximiert.

Eine andere Variante ist an Stelle
des Gini-Koeffizienten
die Missidenfizierungsrate 1-max(p,1-p)
oder die “Cross entropy” –(pln p + (1-p) ln (1-p))
als Güteindex zu verwenden:



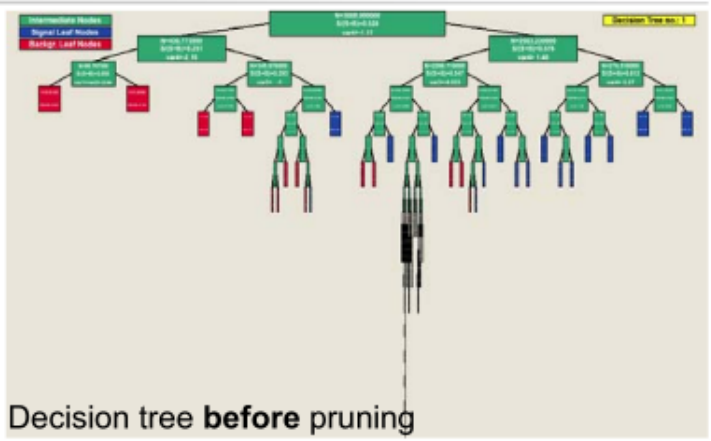
# Ausasten/ Baum zurückschneiden

Stop des Baumwachstums: bis alle Blätter 100% Signal oder Untegrund sind
→ offensichtliches Übertraining
vorher Anhalten: eventuell Trennkraft verschenkt.

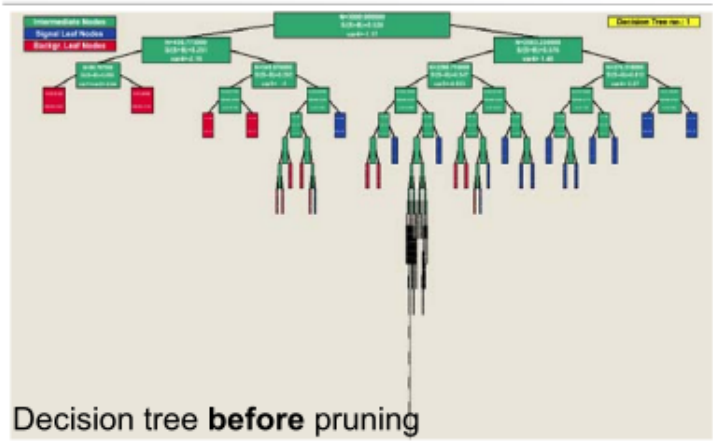
bis zum Ende wachsen lassen, dann zurückschneiden ("pruning")
zu jeden Unterbaum T wird
"Cost Complexity Function" berechnet

$$C(T,\alpha) = \sum_{\substack{\text{leafs} \\ \text{of T}}} \sum_{\substack{\text{events} \\ \text{in leaf}}} |y(x) - y(C)| + \alpha N_{\text{leaf nodes}}$$

schneide Unterbaum mit kleinstem C(T,a) raus
iteriere bis Trennkraft in Testsample und Trainigssample gleich



Decision tree **before** pruning



Decision tree **before** pruning

Ausasten für einzelnen Baum.
Nicht notwendigerweise optimal für einen Wald von Bäumen

# Verstärkung / “Boosting”

“Boosting” ist eine allgemeine Methode ein Menge von “Klassifizierern” zu erzeugen, die man in einem neuen “Klassifizerer” kombinieren kann,um größere Stabilität und kleiner Fehlerrate zu erhalten als mit einem einzelnen.

Oft wird sie auf Entscheidungsbäume angwendet, aber kann auch auf andere Klassifizierungsmethoden angewendet werden

Annahme: wir haben ein Trainnigssample T mit N Ereignissen mit

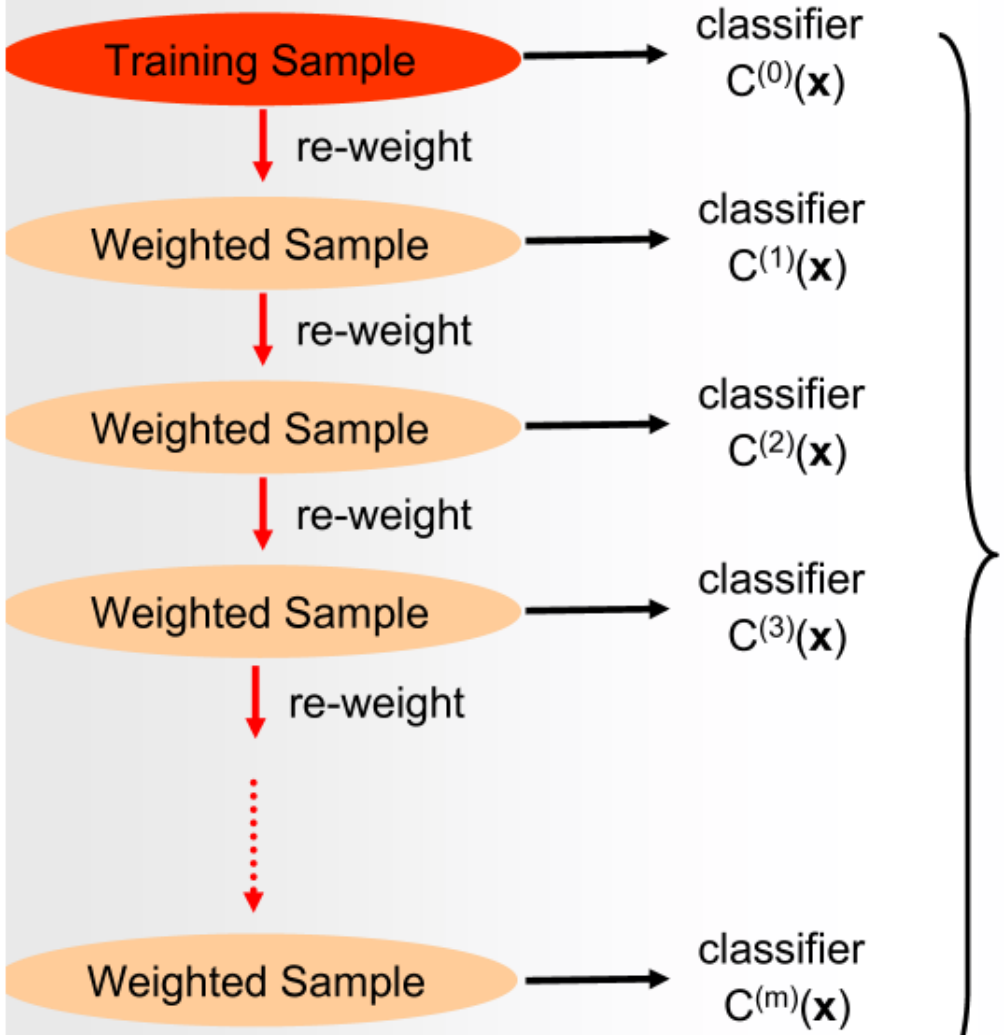
$\boldsymbol{X}_1,\ldots,\boldsymbol{X}_N$ Ereignisvektoren (jedes $\boldsymbol{X}$ ist eine Vektor)

$Y_1,\ldots Y_n$, Klassifizierungswerte 1 für Signal, -1 für untergrund

$w_1,\ldots,w_n$ Ereignisgewichte

Nun definieren wir eine Regel, wie wir aus diesem Sample ein Ensemble von Trainnigssample $T_1$, $T_2$, …. erzeugen, bestimmen aus jedem einen Klassifizierer und bilden den gewichteten Mittelwert aus diesen.

# Verstärkung / “Boosting”




$$y(x) = \sum_{i}^{N_{Classifier}} w_i C^{(i)}(x)$$

gewichtet über viele Klassifizierer $C^{(i)}$

# “AdaBoost”

Ein erfolgreicher Algorithmus ist AdaBoost (Freund & Shapie, 1997)

Start: intialisiere das Traningssample $T_1$ unter Verwendung der Originalwerte für

$\boldsymbol{X}_1,\ldots,\boldsymbol{X}_N$ Ereignisvektoren (jedes $\boldsymbol{X}$ ist eine Vektor)
$Y_1,\ldots,Y_N$ Klassifizierungswerte 1 für Signal, -1 für untergrund
$w_1,\ldots,w_N$ mit selben Ereignisgewichten = 1/N

$$\sum_{i=1}^{N} w_i^{(1)} = 1$$

Trainiere den Klassifizierer $f_1(\boldsymbol{x})$ (z.B. einen Entscheidungsbaum) unter Verwendung der Geiwchte $w_i(1)$. Nun definieren wir eine Regel, wie wir aus diesem Sample ein Ensemble von Trainnigssample $T_1$, $T_2$, …. erzeugen, bestimmen aus jedem einen Klassifizierer und bilden den gewichteten Mittelwert aus diesen.

$$f(\mathbf{x}) = \begin{cases} 1 & \mathbf{x} \text{ in signal region,} \\ -1 & \mathbf{x} \text{ in background region.} \end{cases}$$

# “AdaBoost”: Update der Ereignisgewichte

Kreiere k+1-ten Traningssample $T_{k+1}$ aus dem in der k-ten Stufe $T_k$ durch Umgewichtugn der Ereinisse gemäß:

$$w_i^{(k+1)} = w_i^{(k)} \frac{e^{-\alpha_k f_k(\boldsymbol{x}_i) y_i / 2}}{Z_k}$$

$i$ = event index

$k$ = training sample index

$Z_k$: Normierungsfaktor, so dass Gewichtssumme = 1

Gewicht für Ereignis i wird erhöht im k+1-ten Trainingssample, wenn es im k-ten Trainnigssample falsch klassifiziert wurde.

Idee: nächster Entscheidungsbaum soll mehr Aufmerksamkeit auf die Ereignisse verwenden die inkorrekt klassifiziert wurden.

# “AdaBoost”

In jedem Schritt wird der Klassifiziere $f_k(x)$ so definiert, dass er die Fehlerrate minimiert:

$$\varepsilon_k = \sum_{i=1}^{N} w_i^{(k)} I(y_i f_k(\boldsymbol{x_i}) \leqslant 0)$$

Wobei I(X)=1, wenn X wahr ist und sonst 0.

Die Lernrate ergibt sich aus der Fehlerrate: $\alpha_k = \ln \frac{1-\varepsilon_k}{\varepsilon_k}$

Der finale verstärkte Klassifizierer ist definiert als:

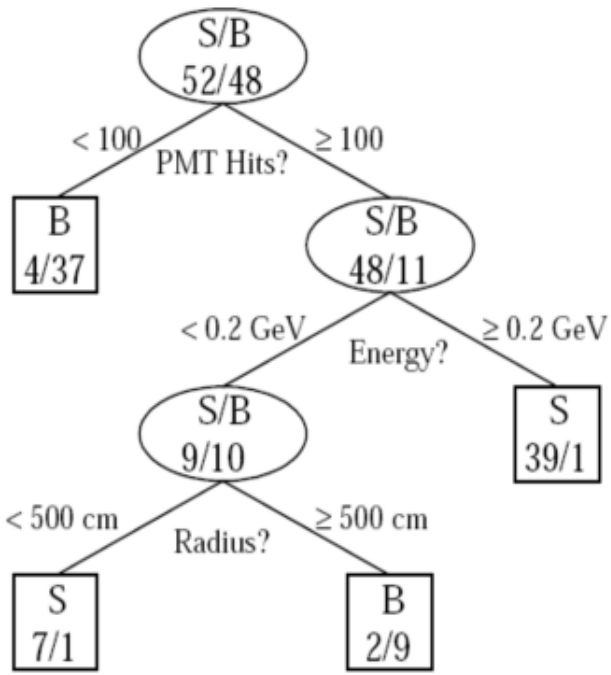
$$f(\boldsymbol{x}) = \sum_{k=1}^{K} \alpha_k f_k(\boldsymbol{x}, T_k)$$

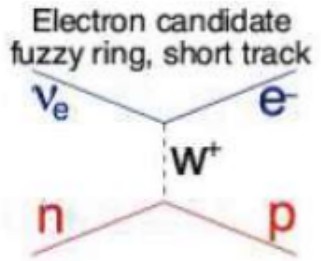
Es lägt sich zeigen, dass für die Fehlerrate dieses Klassifizierers gilt:

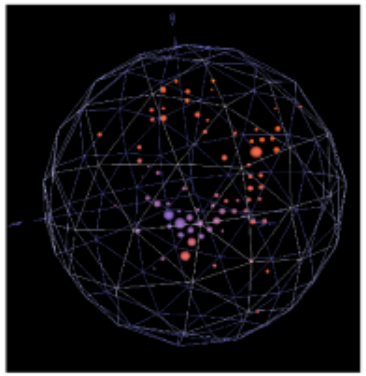
$$\varepsilon \leqslant \prod_{k=1}^{K} 2\sqrt{\varepsilon_k(1-\varepsilon_k)}$$

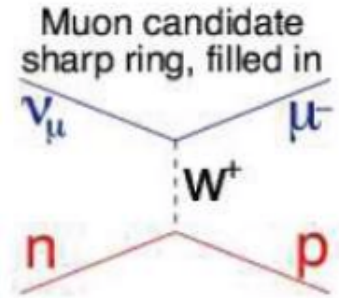
# BDT: ein Beispiel

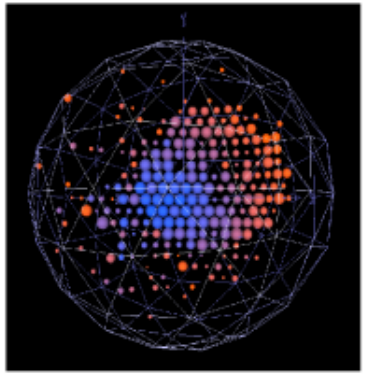
Beispiel des
MiniBooNe-Experimentes

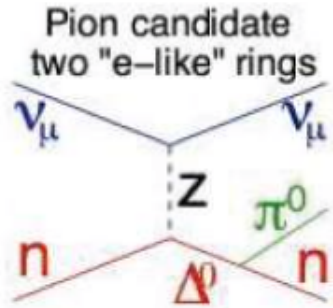


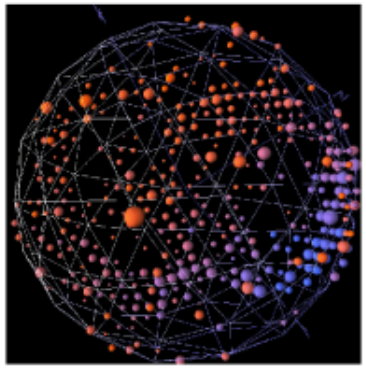





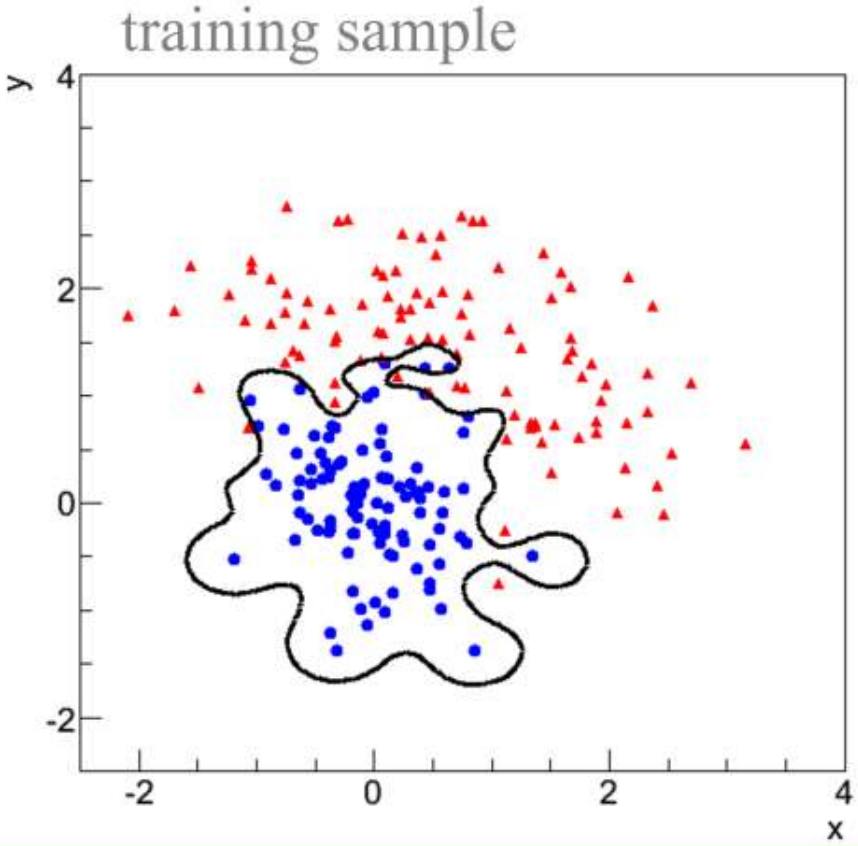


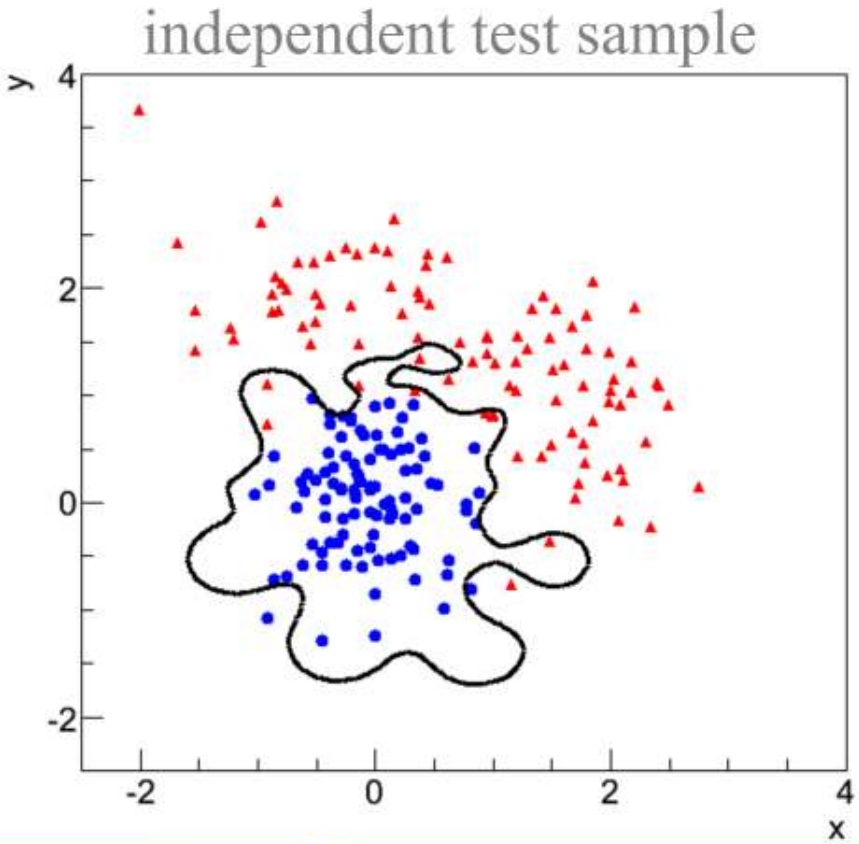



H.J. Yang, MiniBooNE PID, DNP06

# Übertraining

Wenn die Entscheidungsgrenze zu flexible ist, wird sie sich zu eng an alle Ereignisse im Trainingssample anschmiegen → Übertraining
Überprüfe die Qualität der Trennkraft durch Anwendung auf statistisch unabhängiges Ereignistestsample.

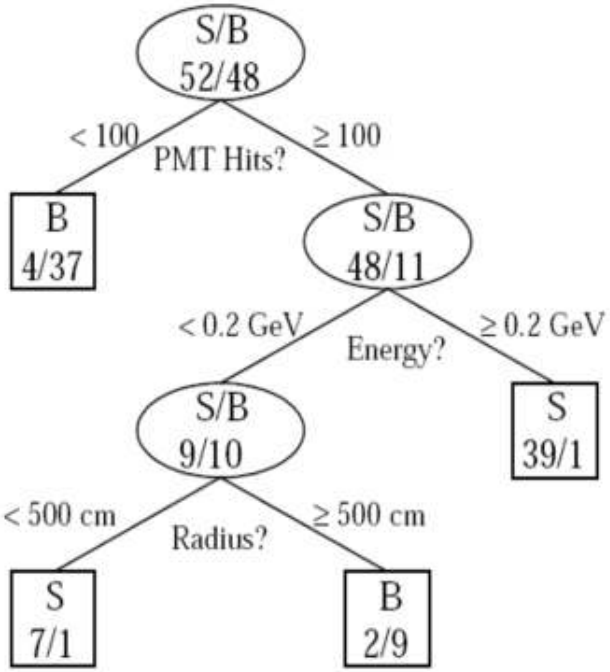


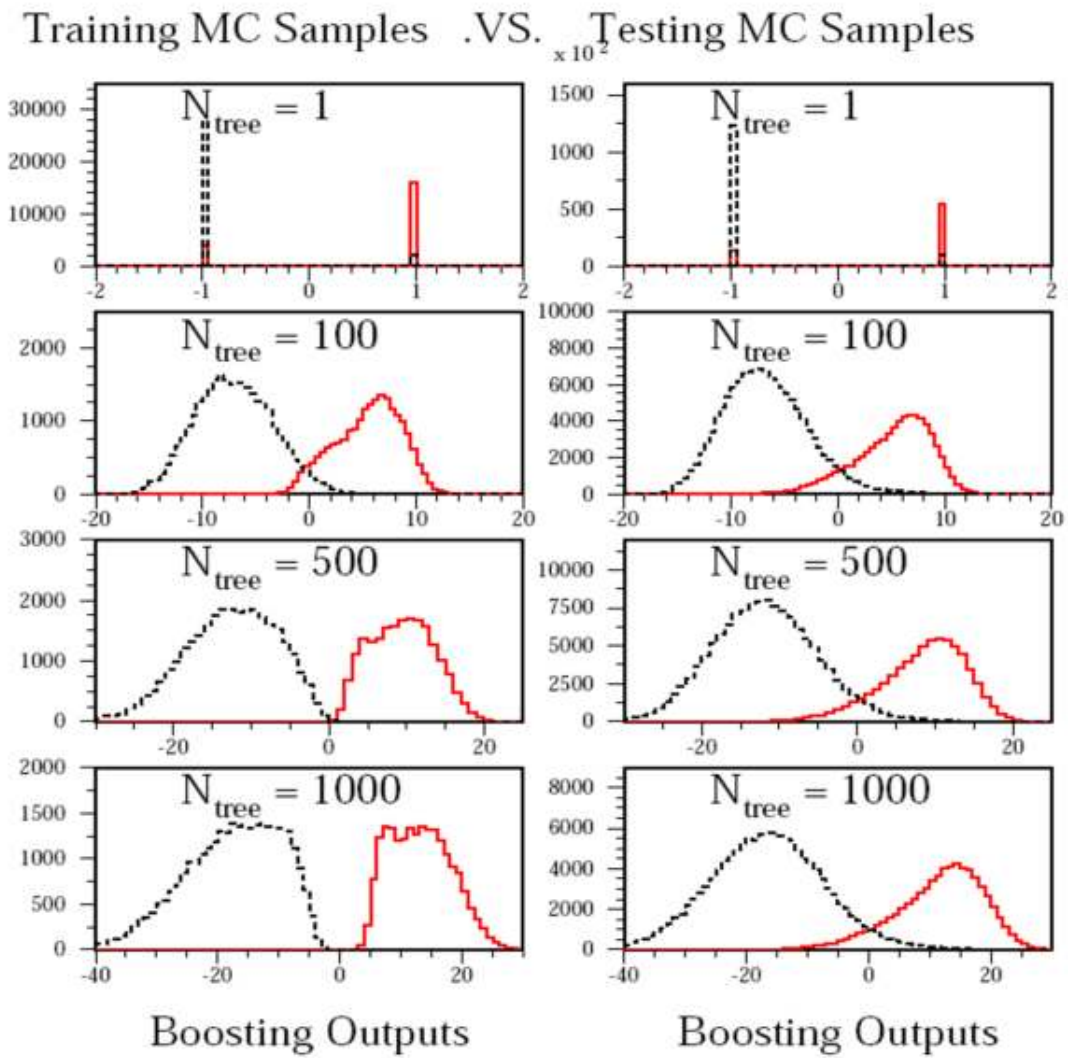



# Überwachung des Trainingprozesses

Beispiel des MiniBooNe-Experimentes

Trennkraft stabil nach ein paar hundert Entscheidungsbäumen.

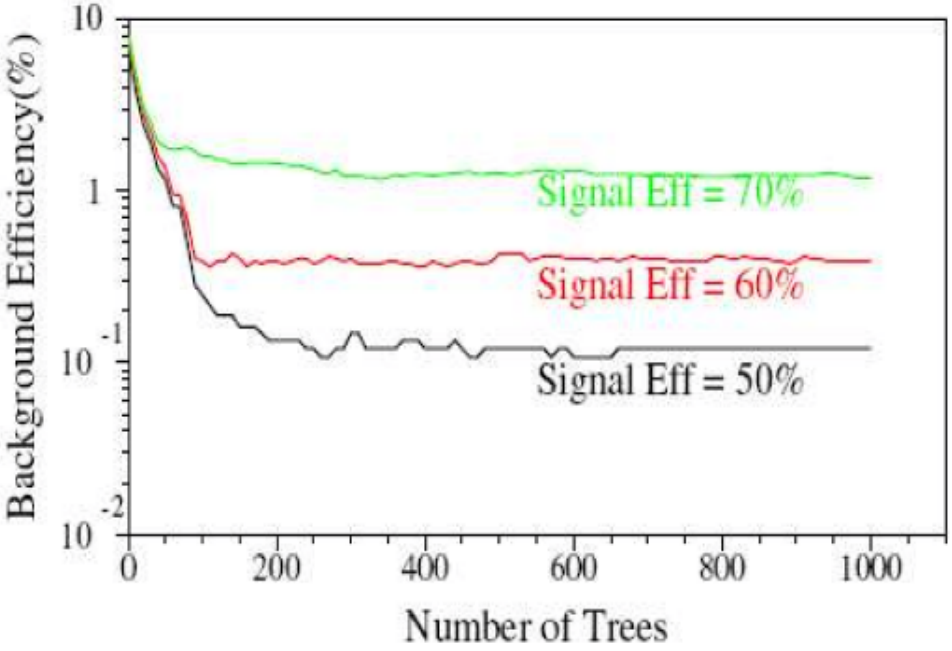


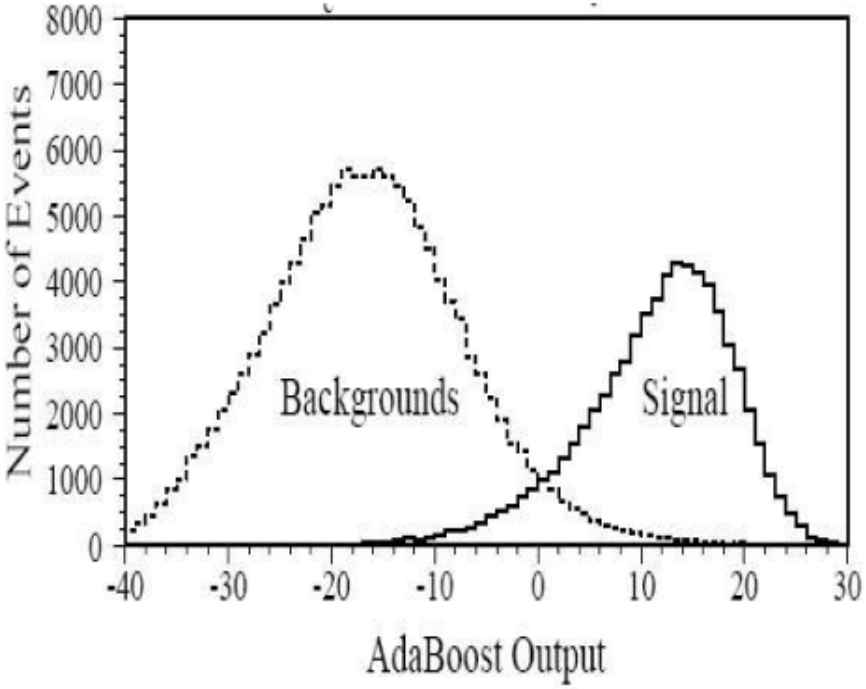




B. Roe et al., NIM 543 (2005) 577

# Übertraining

Wenn die Entscheidungsgrenze zu flexible ist, wird sie sich zu eng an alle Ereignisse im Trainingssample anschmiegen → Übertraining
Überprüfe die Qualität der Trennkraft durch Anwendung auf statistisch unabhängiges Ereignistestsample.

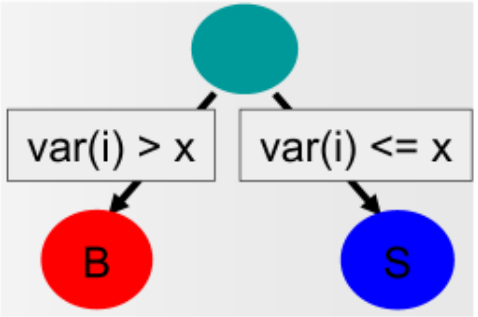


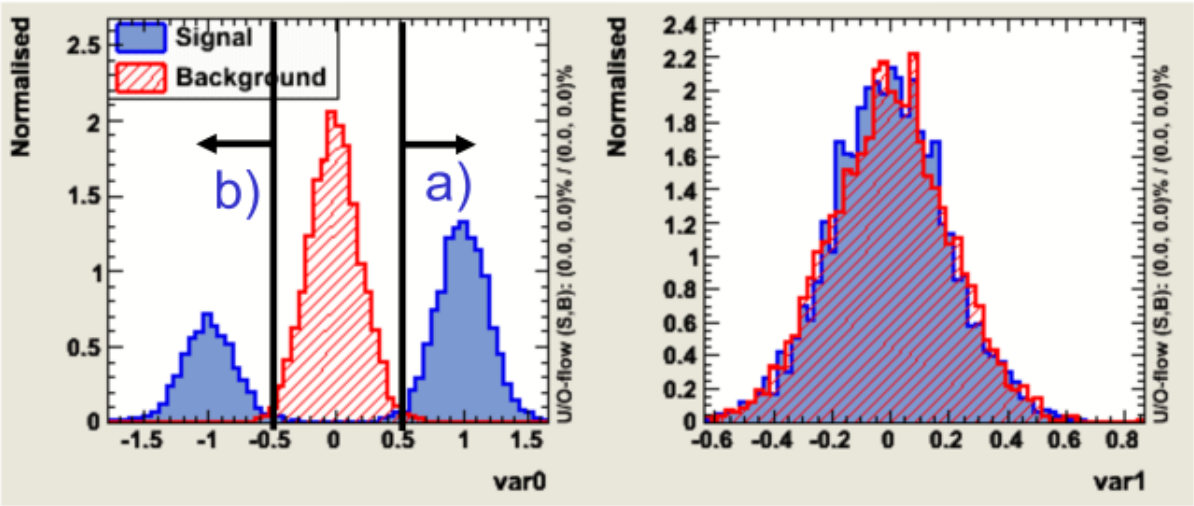



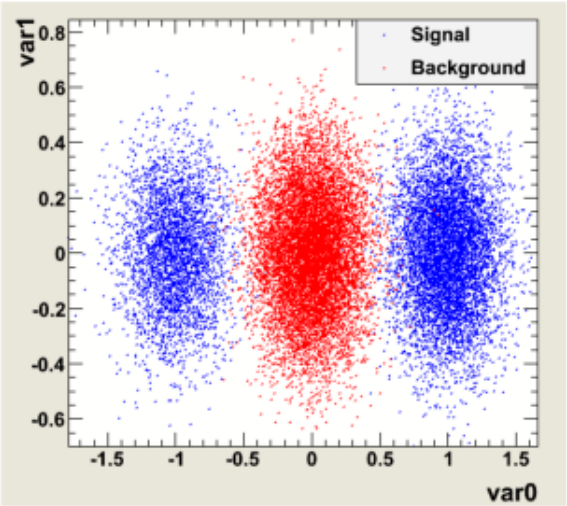
# AdaBoost: Eine naives Beispiel

Zwei Variablen: in der einen 3 Peaks

Einfacher Entscheidungsbaum findet 1 Schnittwert




2 sinnvolle Schnitte: a) Var0 > 0.5 → $\varepsilon_{signal}$=66% $\varepsilon_{bkg} \approx 0\%$ Fehlerrate 16.5%

oder

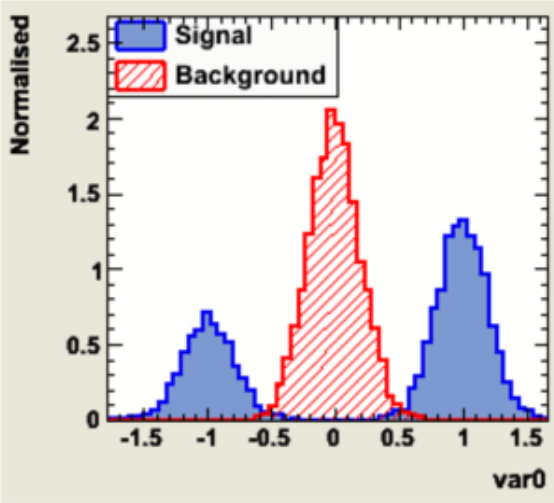
b) Var0 < -0.5 → $\varepsilon_{signal}$=33% $\varepsilon_{bkg} \approx 0\%$ Fehlerrate 33%

Das Training eines einzelnen Entscheidungbaums findet Schnitt a)

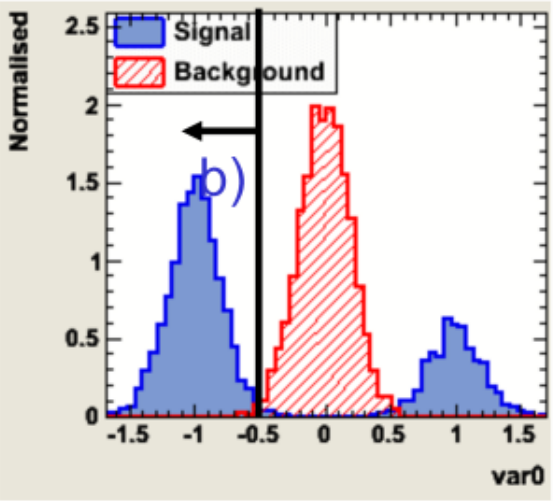
# AdaBoost: Eine naives Beispiel

Vor “Wachsen” des neuen Baumes werden Ereignisse umgewichtet

Der nächste Baum sieht folgende Verteilung in der Variable Var0

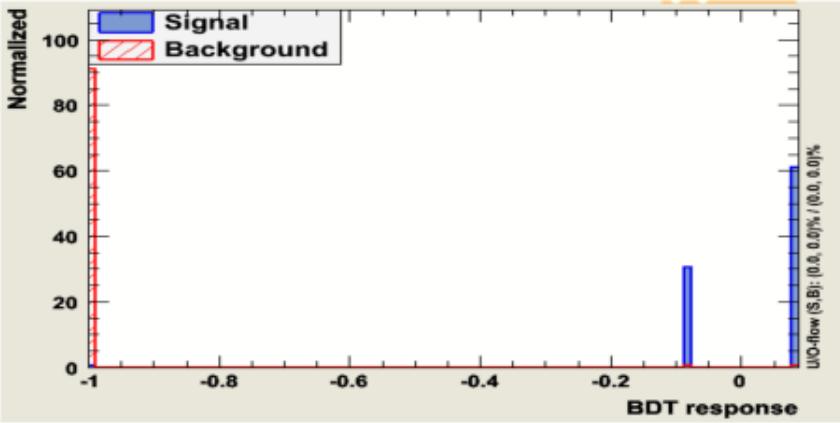



Umgewichtung



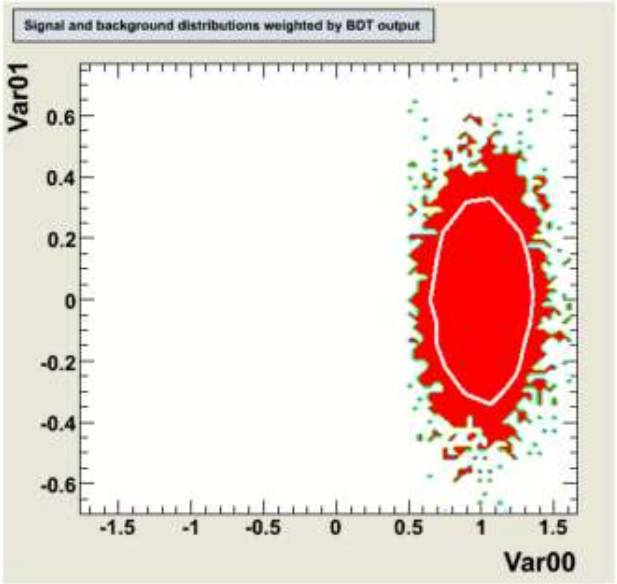

nun wird Schnitt
b) gewählt and
Var0 < -0.5

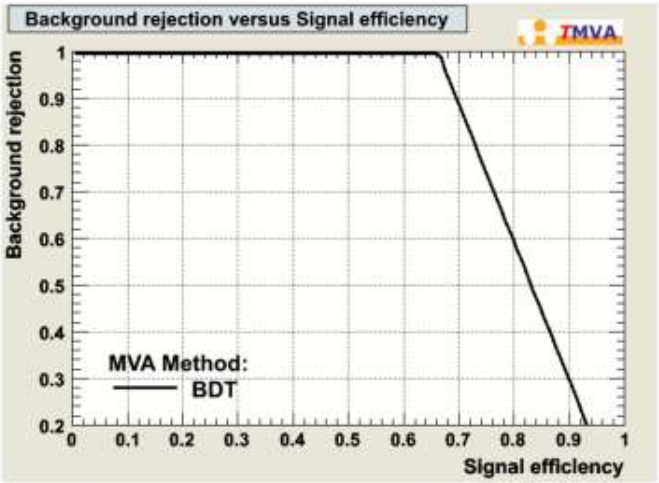
Der kombinierte Klassifizierer
aus Baum 1 und Baum 2
liefert hier optimale Trennkraft



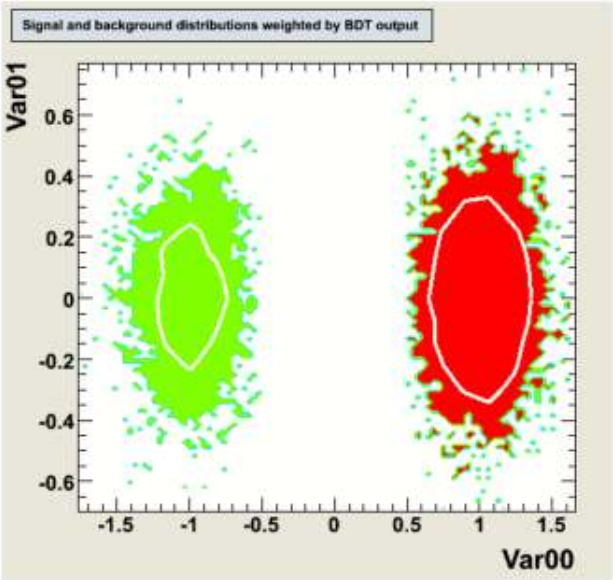
# AdaBoost: Eine naives Beispiel

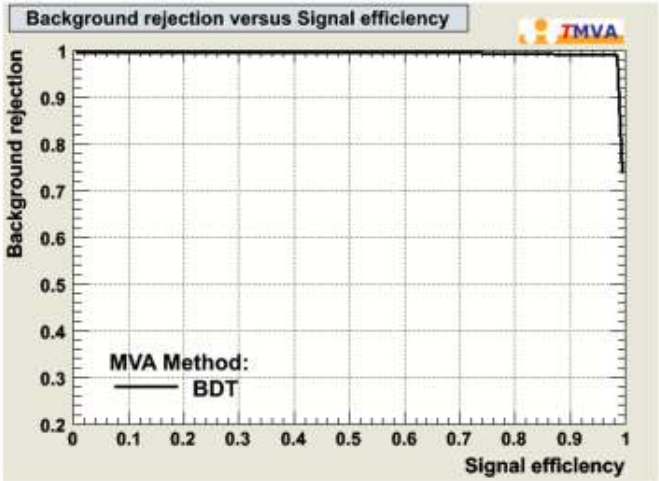
## Nur ein Baum



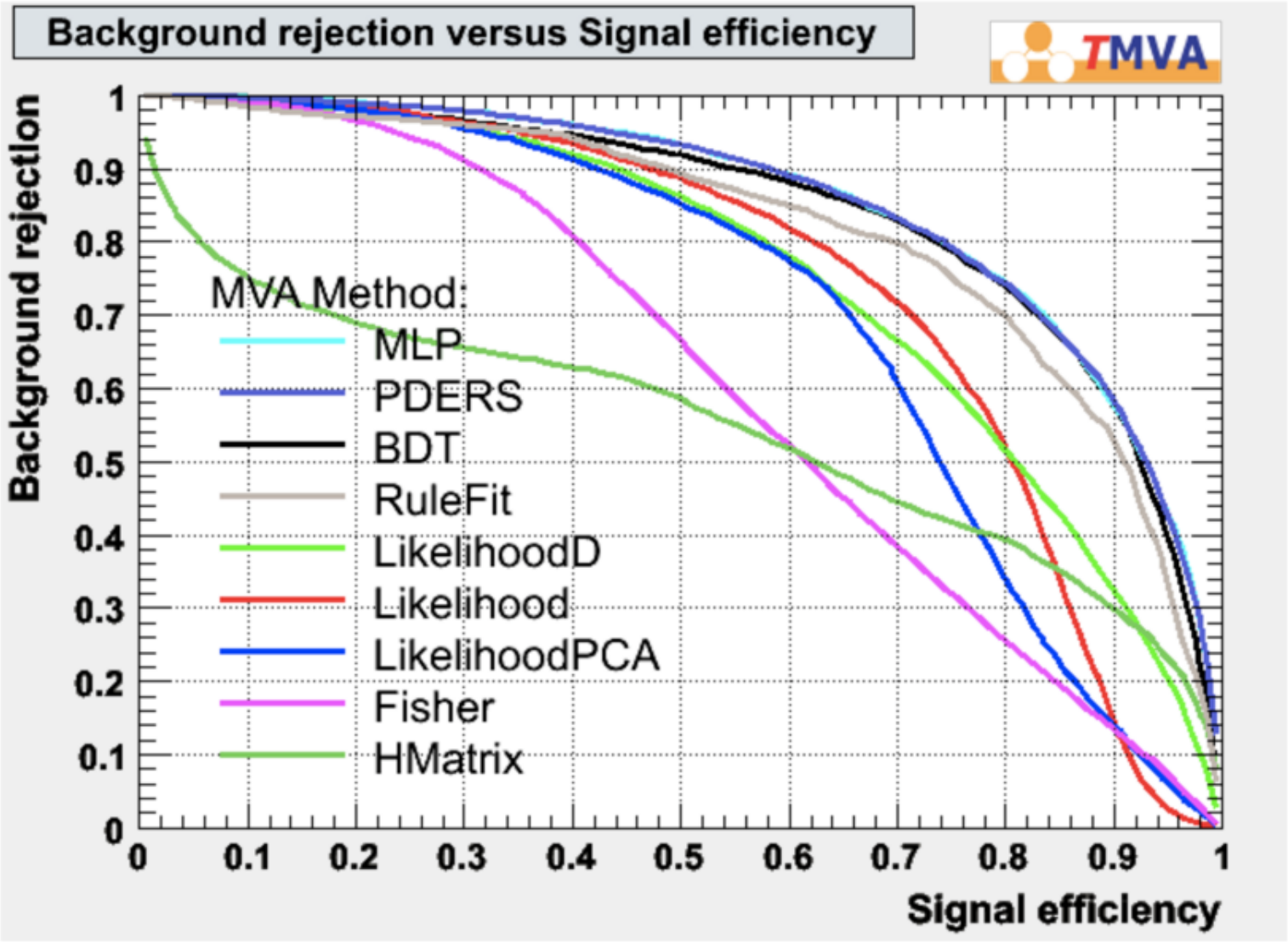




## Kombination von 2 Bäumen



# Vergleich multivariater Methoden




Welche die beste ist hängt vom Problem ab.