多変数解析ことはじめ

Junpei Maeda
Kobe University

ATLAS-Japan Software Tutorial 2016

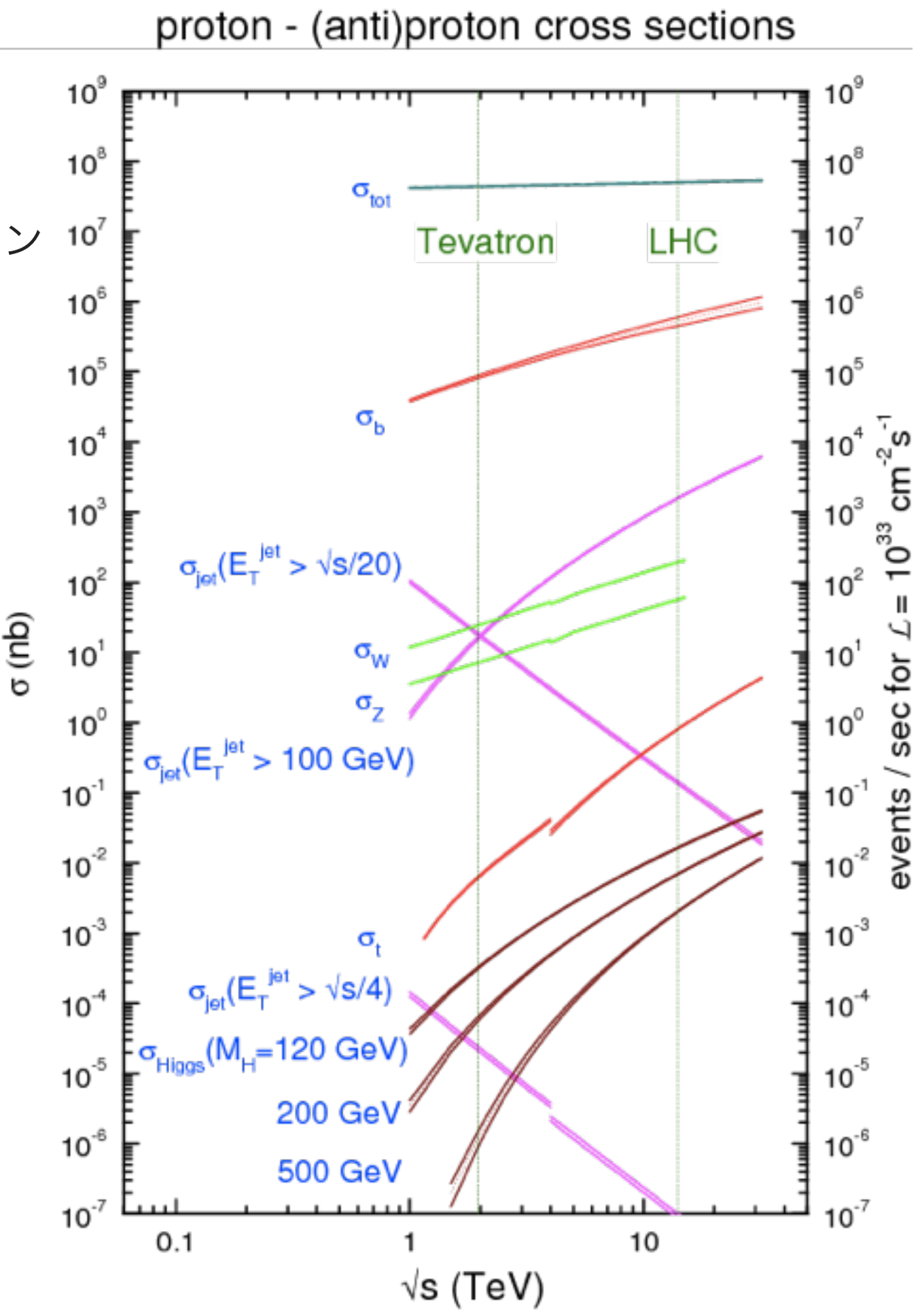
ICEPP, 28 December 2016

何やるの？

- たくさんの事象のなかから興味ある事象のみを選別する
  - トリガー & オフラインイベントセレクション
    e.g.) E_T > xxx GeV, N_{jet} > y
- 一つ（ずつ）の変数でカットをかけるのは限界がある
  - 複数のパラメータを組み合わせて使う

- 多変数解析とは
  - シグナル（欲しいもの）
  - バックグラウンド（いらないもの）
- の二つを効率よく分けるために
  それぞれ特徴をよく表すパラメータ一つ以上を定量化（数値化）して、事象選別を行う解析手法

- ROOTにくっついたものでは*T*MVAというパッケージがある

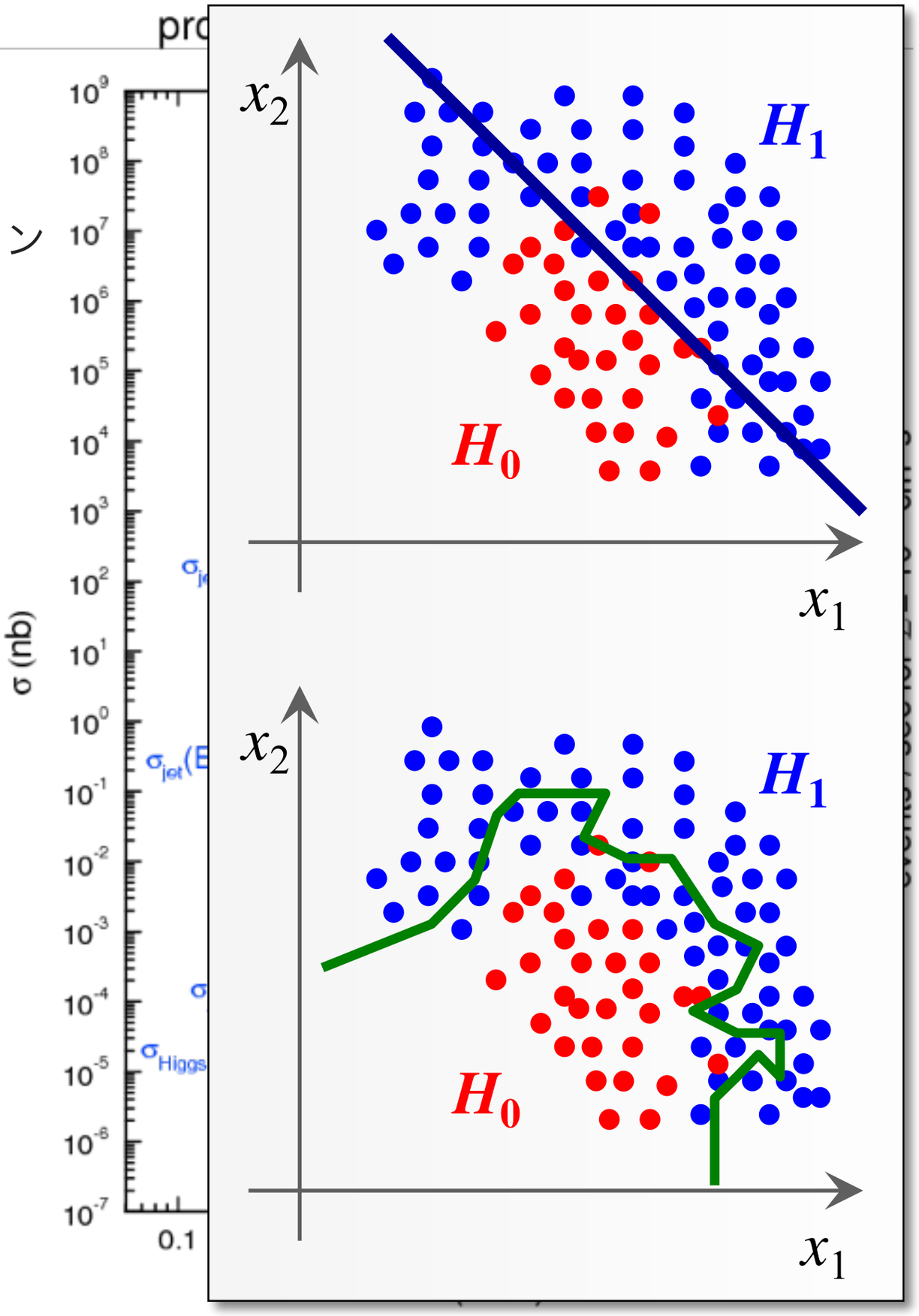


何やるの？

- たくさんの事象のなかから興味ある事象のみを選別する
  - トリガー & オフラインイベントセレクション
    e.g.) E_T > xxx GeV, N_{jet} > y
- 一つ（ずつ）の変数でカットをかけるのは限界がある
  - 複数のパラメータを組み合わせて使う

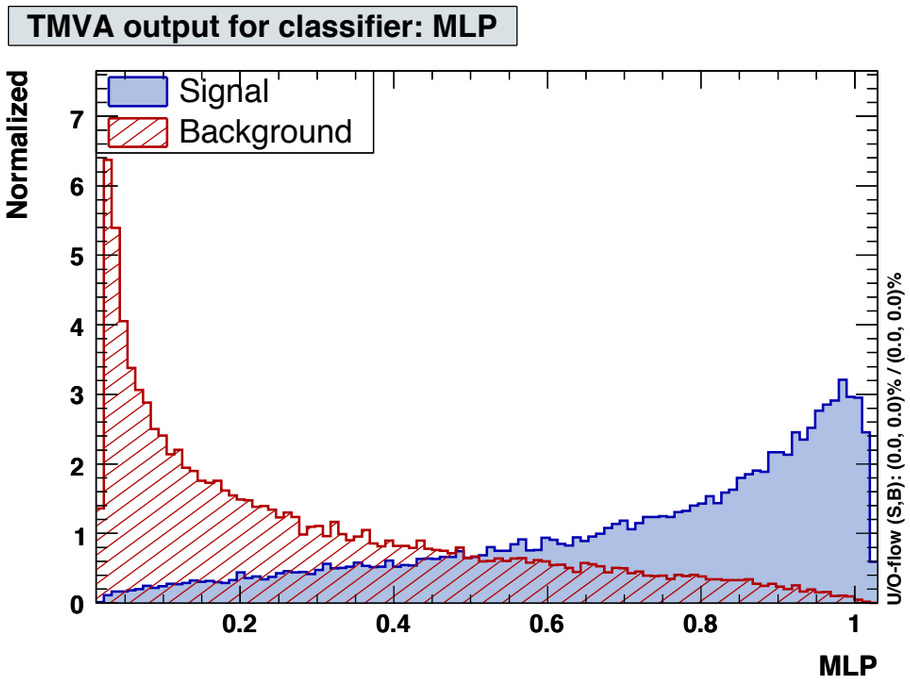
- 多変数解析とは
  - シグナル（欲しいもの）
  - バックグラウンド（いらないもの）
- の二つを効率よく分けるために
  それぞれ特徴をよく表すパラメータ一つ以上を定量化（数値化）して、
  事象選別を行う解析手法

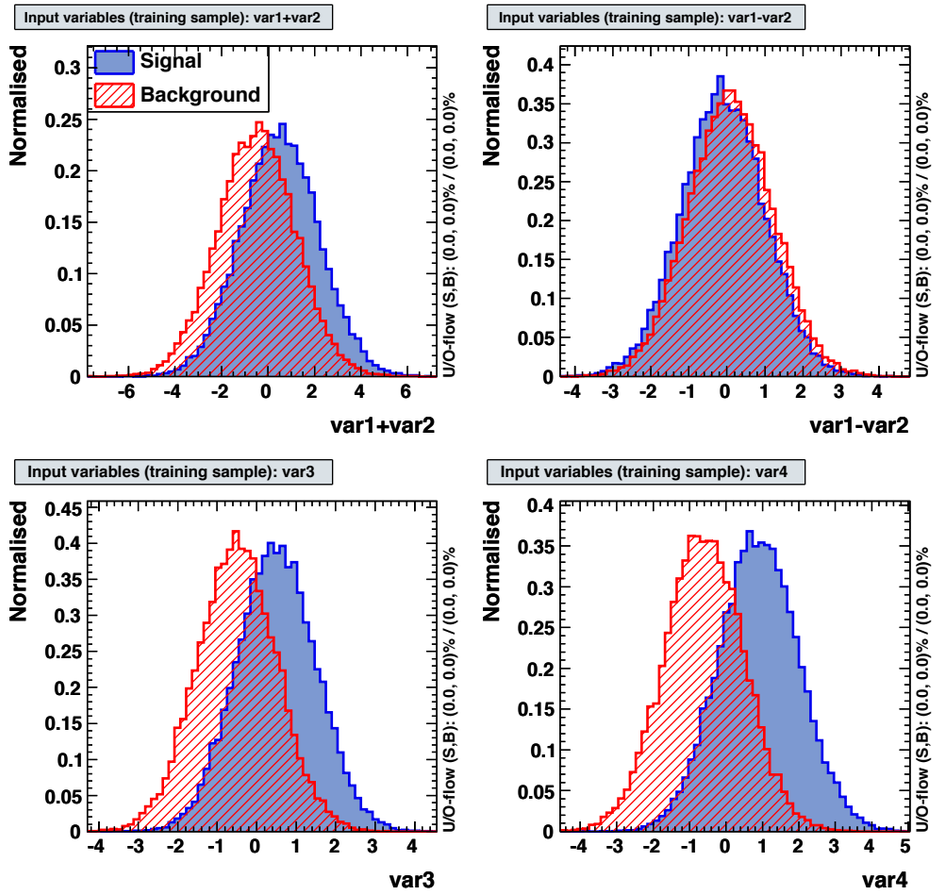
- ROOTにくっついたものでは*T*MVAというパッケージがある



TMVAとは…

- 多変数解析の色々な手法を提供する
  - 複数のinput informationを
    一つの数値としてoutputする





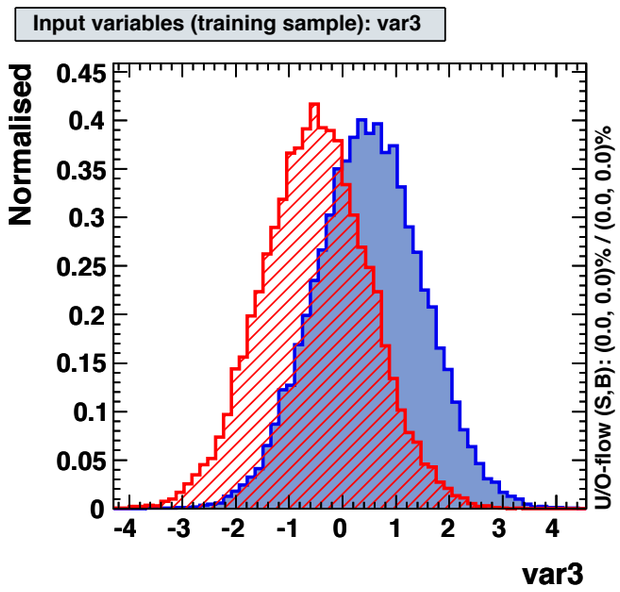
- こちらがあらかじめ教える（トレーニングする）必要がある
- それ以外に…

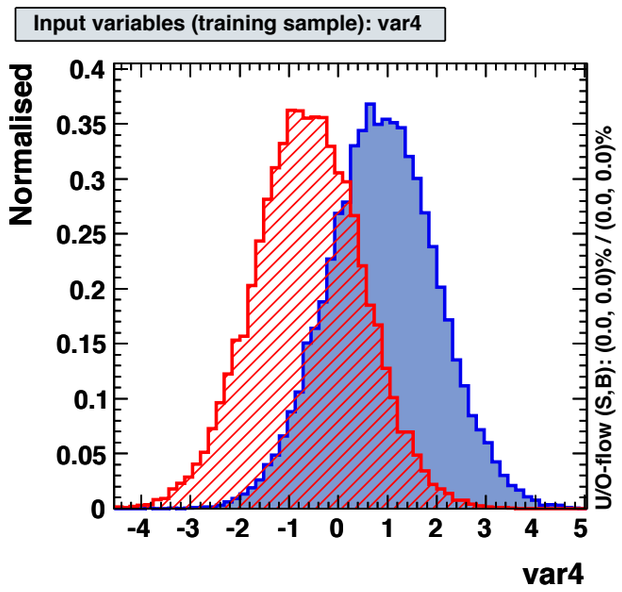
– a common interface for all MVA techniques
– a common interface for classification and regression
– easy training and testing of all methods on the same datasets
  - consistent evaluation and comparison
  - common data preprocessing

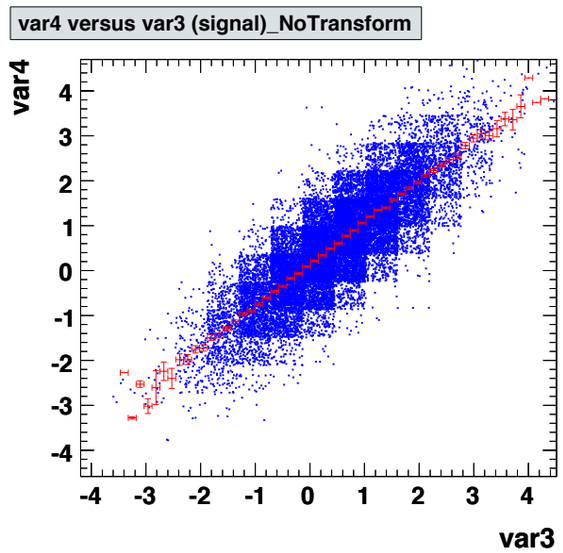
– a complete user analysis framework and examples
– embedding in ROOT
– creation of standalone C++ classes (ROOT independent)
– an understandable Users Guide

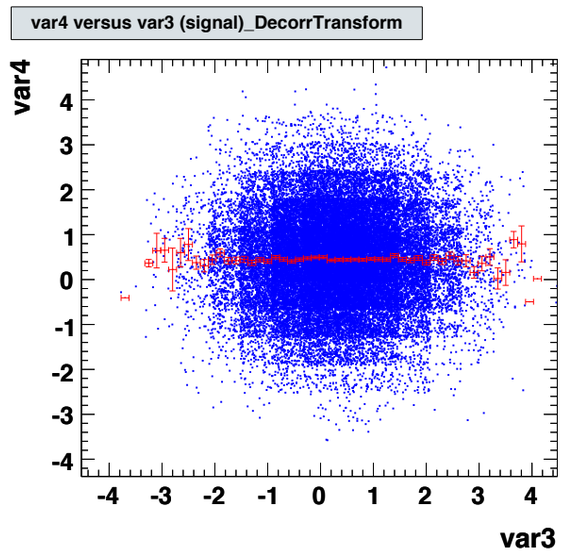
Correlation of input variables

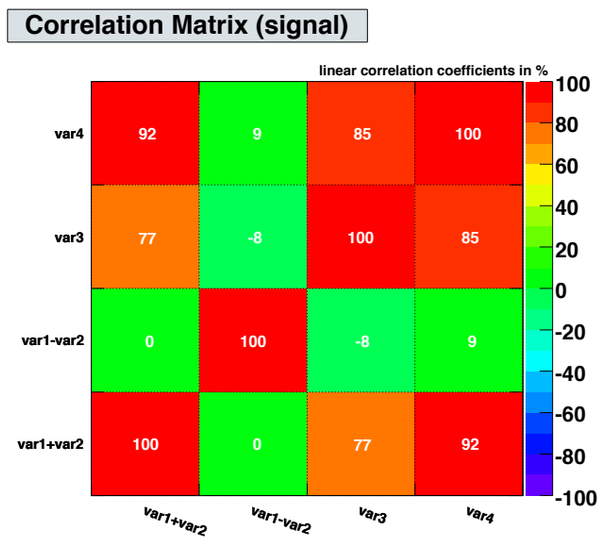
- 使う変数間の相関を知ることは大事
  - 一般的に相関無い方が多変数解析は扱いやすい。
    e.g.) 変数x, yに対してy = axという相関になっていればxだけと(x, y)を使うのは原理的に同じ
  - 多変数解析はサンプルを用意したら後は数遊びな点もあるので
    Decorrelationという手法も存在する
    - とある条件（originalがガウス分布とか）が
      満たされているときに使用可能（今回は割愛します）

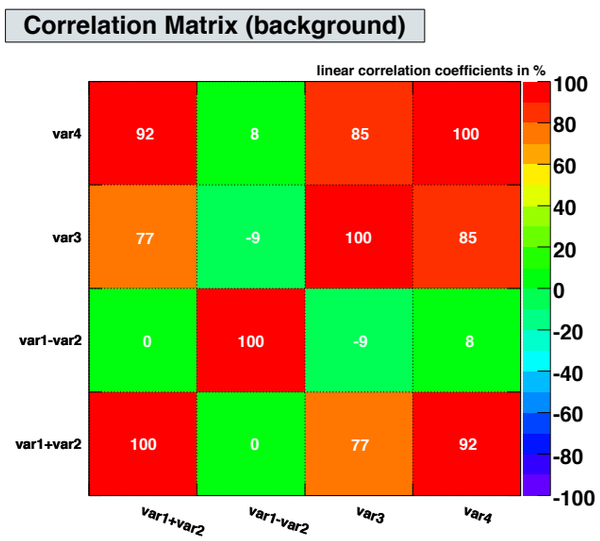








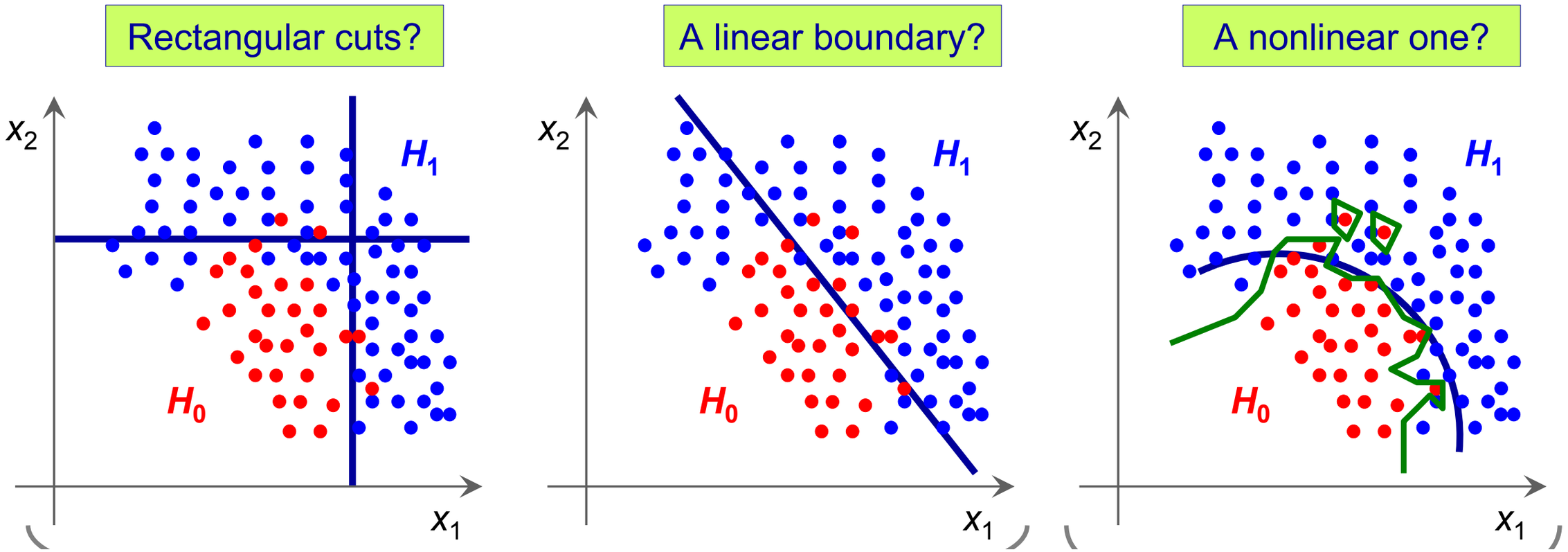




Classifiers

Classifiers: 実装されているアルゴリズム

- Rectangular cut optimization
- Projective and Multidimensional likelihood estimator
- k-Nearest Neighbor algorithm
- Fisher and H-Matrix discriminants
- Function discriminant
- Artificial Neural Network
- Boosted/Bagged decision trees
- RuleFit
- Support Vector Machine
- …



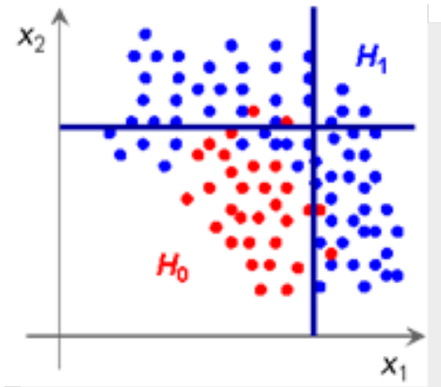
Summary of Classifiers and their Properties

Criteria		Classifiers								
		Cuts	Likeli-hood	PDERS / k-NN	H-Matrix	Fisher	MLP	BDT	RuleFit	SVM
Perfor-mance	no / linear correlations	😐	☺	☺	😐	☺	☺	😐	☺	☺
	nonlinear correlations	😐	☹	☺	☹	☹	☺	☺	😐	☺
Speed	Training	☹	☺	☺	☺	☺	😐	☹	😐	☹
	Response	☺	☺	☹/😐	☺	☺	☺	😐	😐	😐
Robust-ness	Overtraining	☺	😐	😐	☺	☺	☹	☹	😐	😐
	Weak input variables	☺	☺	☹	☺	☺	😐	😐	😐	😐
Curse of dimensionality		☹	☺	☹	☺	☺	😐	☺	😐	😐
Transparency		☺	☺	😐	☺	☺	☹	☹	☹	☹

The properties of the Function discriminant (FDA) depend on the chosen function A

Classifier: Projective Likelihood Estimator (Naive Bayes)

- それぞれのパラメータ (input variables)の PDF (probability density function) を 掛け合わせる。



Likelihood ratio for event i_{event} ← $R_L(i_{event})$

PDFs ← p_k^{sig}, p_k^{bkg}

discriminant variables ← $x_k(i_{event})$

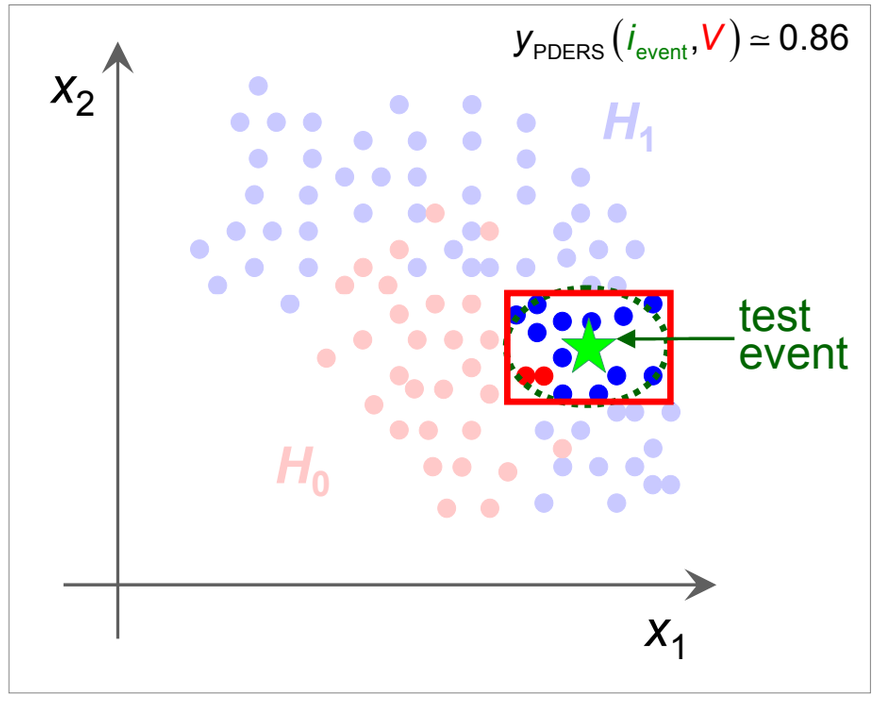
$$R_L(i_{event}) = \frac{\prod_{k\in\{\text{variables}\}} p_k^{sig}(x_k(i_{event}))}{\prod_{k\in\{\text{variables}\}} p_k^{sig}(x_k(i_{event})) + \prod_{k\in\{\text{variables}\}} p_k^{bkg}(x_k(i_{event}))}$$

- 前提として各inputは独立と想定
  - 相関が無い時にもっとも力を発揮する
  - 線形相関のときはdecorrelationという手法もつかえる (今日は割愛)

Performance		Speed		Robustness		Curse of Dim.	Transparency
No/linear correlations	Nonlinear correlations	Training	Response	Overtraining	Weak input vars		
Good	**Bad**	**Good**	**Bad**	**Fair**	**Good**	**Good**	**Good**

classifier: Multidimensional PDE approach

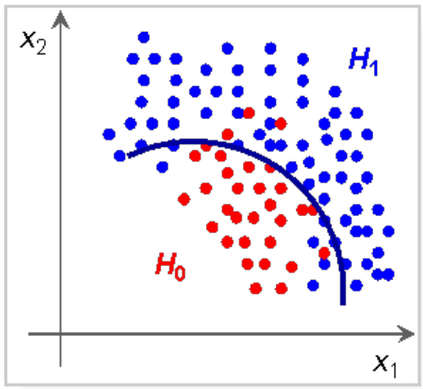
- PDE Range-Search:
  - 見たいイベント「付近」のトレーニングサンプルの
    シグナルとバックグラウンドの数を見て、そのシグナル
    密度を評価する手法
    - T. Carli et al., NIM A501 (2003) 576.
    - used for ZEUS hadronic tau identification
  - Binary tree searchを使って高速に数はカウントしている

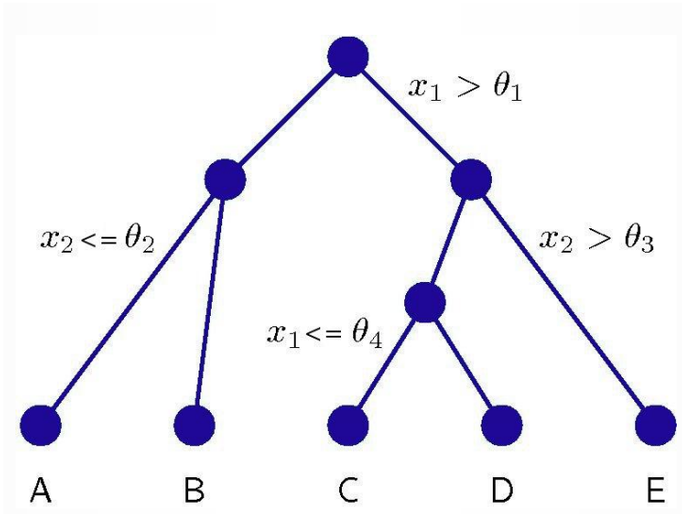


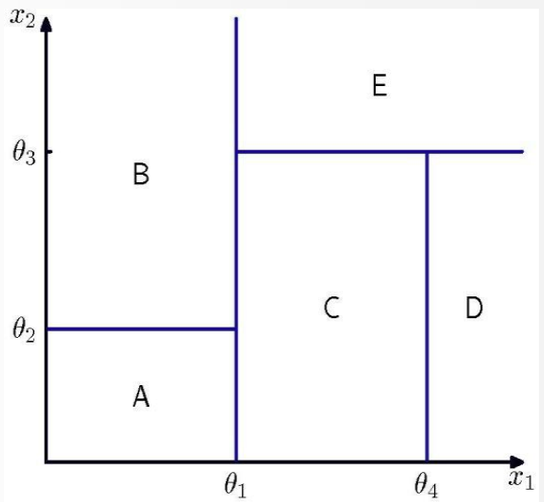
Performance		Speed		Robustness		Curse of Dim.	Transparency
No/linear correlations	Nonlinear correlations	Training	Response	Overtraining	Weak input vars		
Good	**Good**	**Good**	**Fair/Bad**	**Fair**	**Bad**	**Bad**	**Fair**

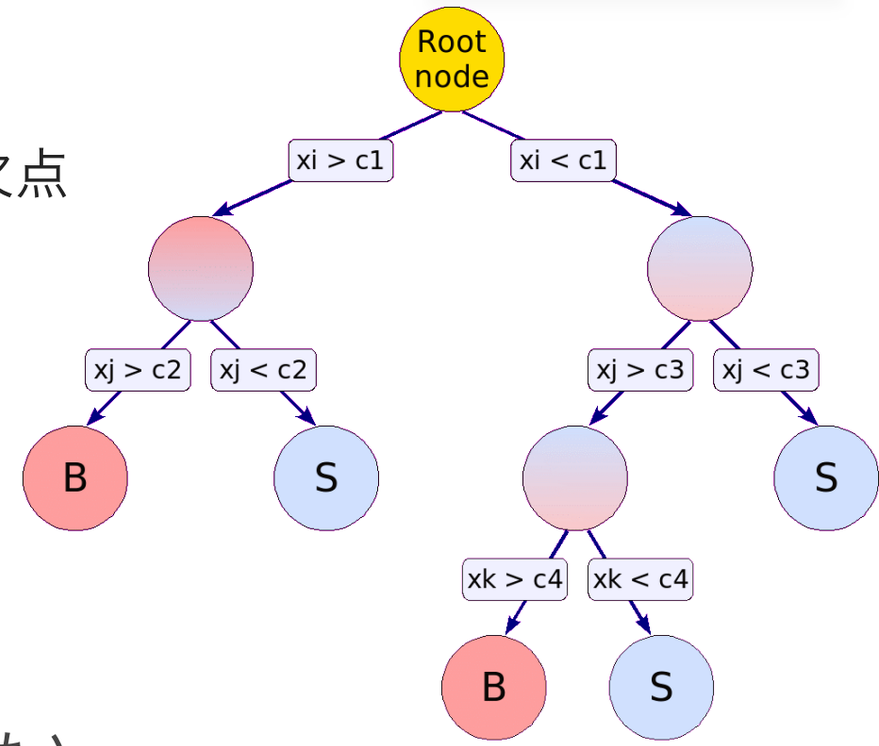
Classifier: Boosted Decision Trees (BDT)

- Decision Tree: カットを繰り返してシグナルとバックグラウンドを分けていく。最終的に一番下のノード (leaf)がシグナル or バックグラウンドに別れる。
  - 長い期間HEP以外でも使われていた手法
  - 実際のカット値の繰り返しを表示しやすい
  - Separationの弱い変数は無視される
  - トレーニングサンプルの統計的ふらつきにSensitiveなのが欠点







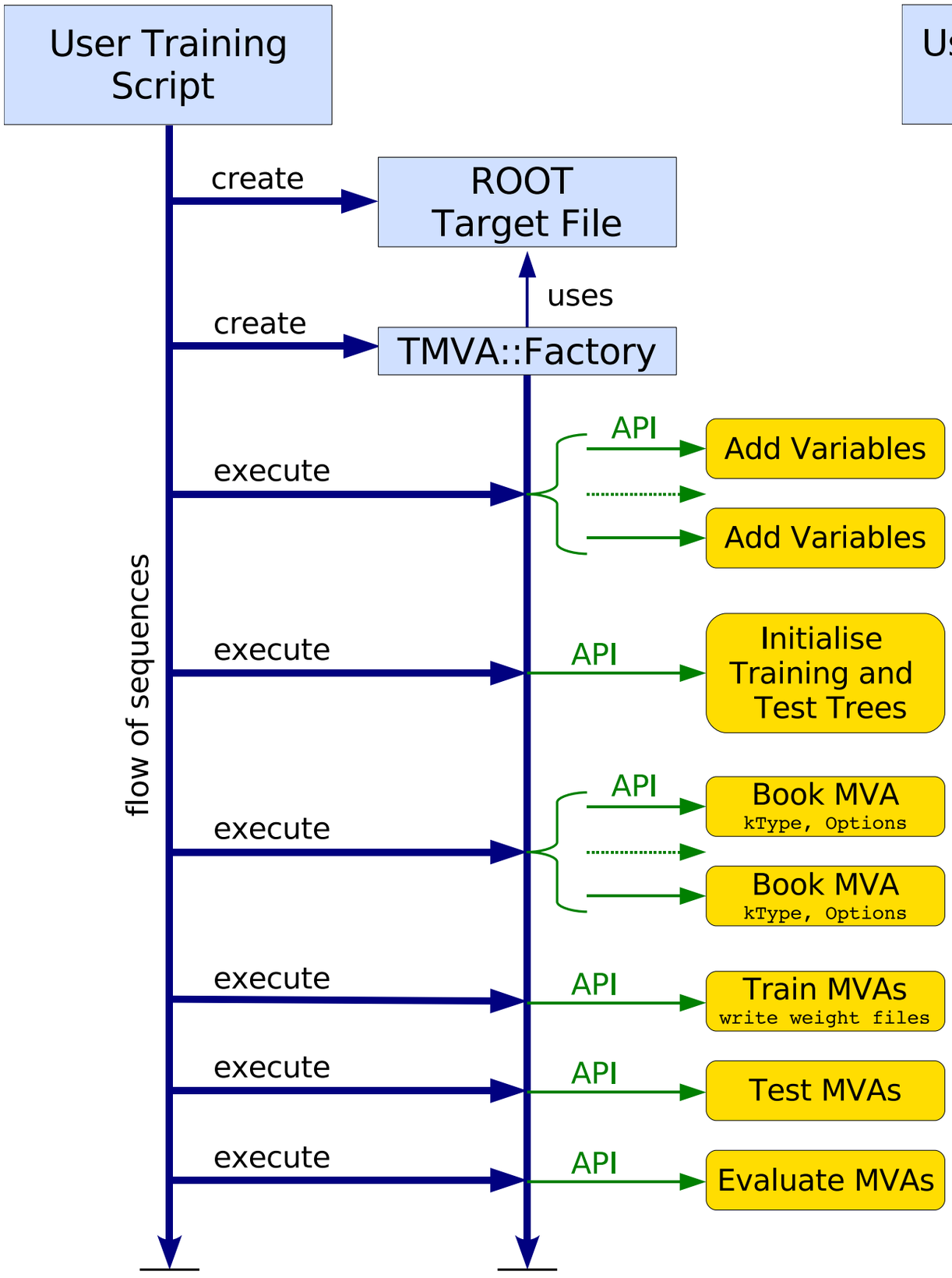


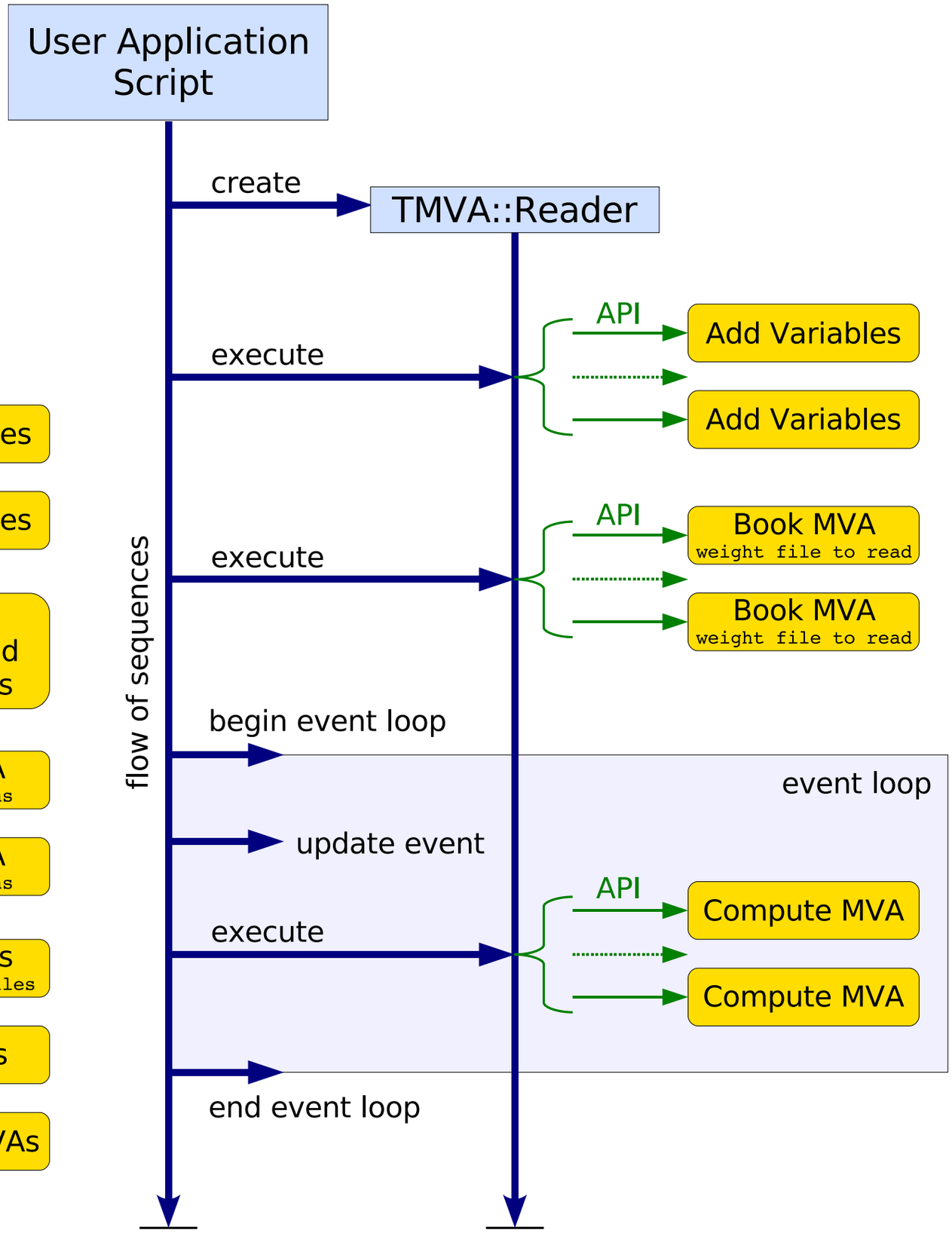
- Boosted Decision Trees (MiniBooNEで有名になった):
  - Decision treeを拡張したもので、Decision treeを同じサンプルに対して異なるevent weightを適用したもの

Performance		Speed		Robustness		Curse of Dim.	Transparency
No/linear correlations	Nonlinear correlations	Training	Response	Overtraining	Weak input vars		
Fair	**Good**	**Bad**	**Fair**	**Bad**	**Fair**	**Good**	**Bad**

TMVAを使ってみる

TMVA workflow





Training Dataを用意する

- *T*MVAは
  - Data input format: ROOT TTree (or ASCII)
  - 変数を用いたセレクションや、計算したものをinput variableに出来る
  - サンプルやイベントごとにweightをかけることも出来る
  - 自動的にトレーニングサンプルとテストサンプルに分けてくれる
- 今回は時間無いのでROOTがオフィシャルに用意しているtoy sampleと（ちょっと今回用に変更した）コードを用いて遊んでみます。

```
$ setupATLAS
$ lsetup root
$ cp ~junpei/TMVAtut2016 ~/TMVAtut2016
$ cd ~/TMVAtut2016
$ gmake
```

- ちょっとサンプル見てみましょう。例えば

```
$ root -l tmva_class_example.root
root [] .ls
root [] TreeS->Print();
root [] TreeS->Draw("var1");
```

トレーニング＆評価するプログラムを動かす

- 本来はプログラムを書く、色々なプロットを見る、評価する、修正する、を繰り返す必要があります
- 今日は出来上がったサンプルプログラムを動かすだけにします。

```
$ ./TMVAClassification | tee logfile
```

- コードはTMVAClassification.Cに書いてあります。
  上からざっと読んでみましょう。
- また出力されたlogfileも見ていきましょう。
  - 一般的なログファイルに加えて、各Classifierの説明
  - 変数のSeparationの強さ、登録したClassifierの今回の場合のランキング

トレーニング＆評価するプログラムを動かす

- 本来はプログラムを書く、色々なプロットを見る、評価する、修正する、を繰り返す必要があります
- 今日は出来上がったサンプルプログラムを動かすだけにします。

```
$ ./TMVAClassification | tee logfile
```

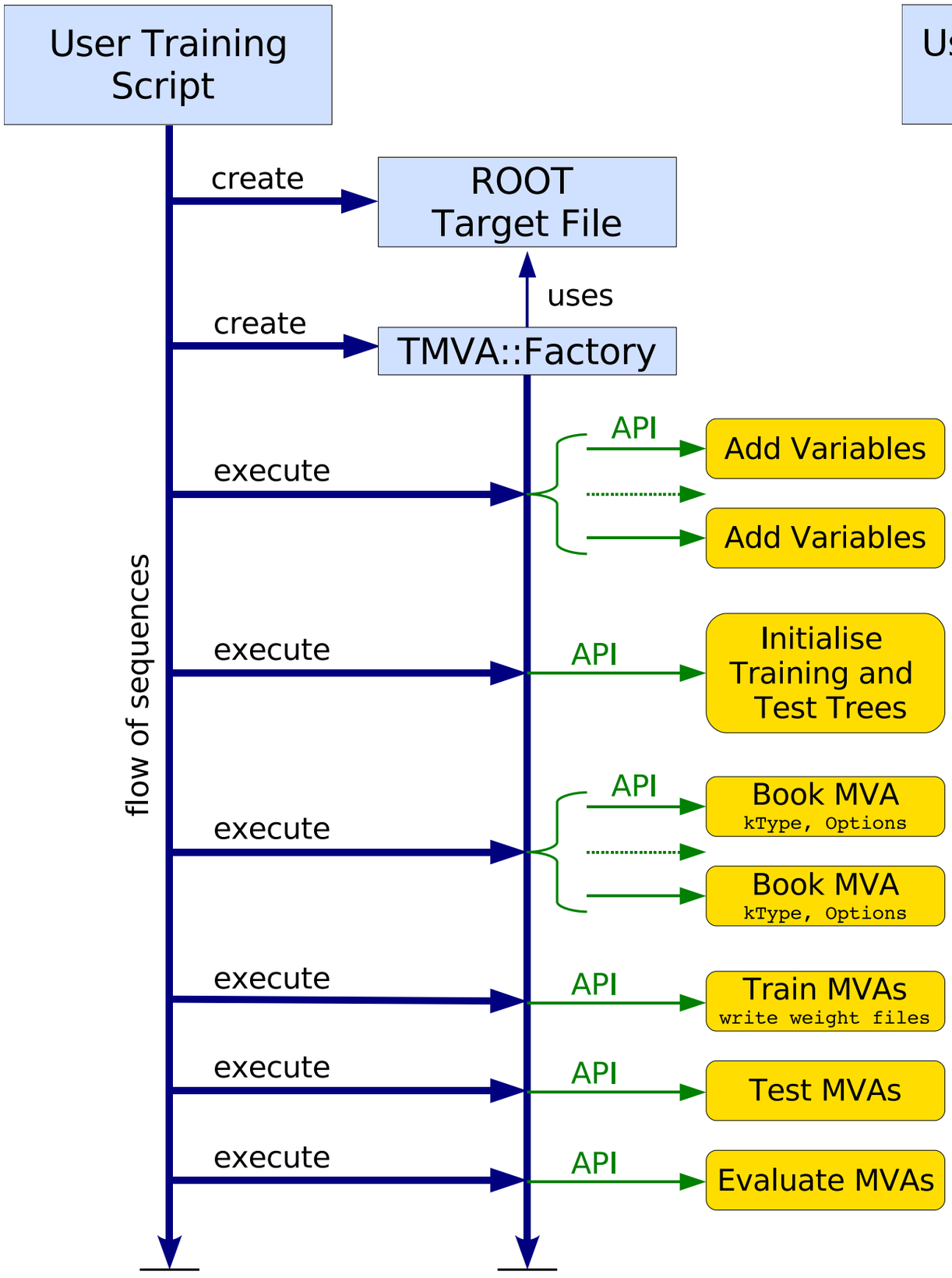
- コードはTMVAClassification.Cに書いてあります。
- ちょっとだけ読んでみましょう

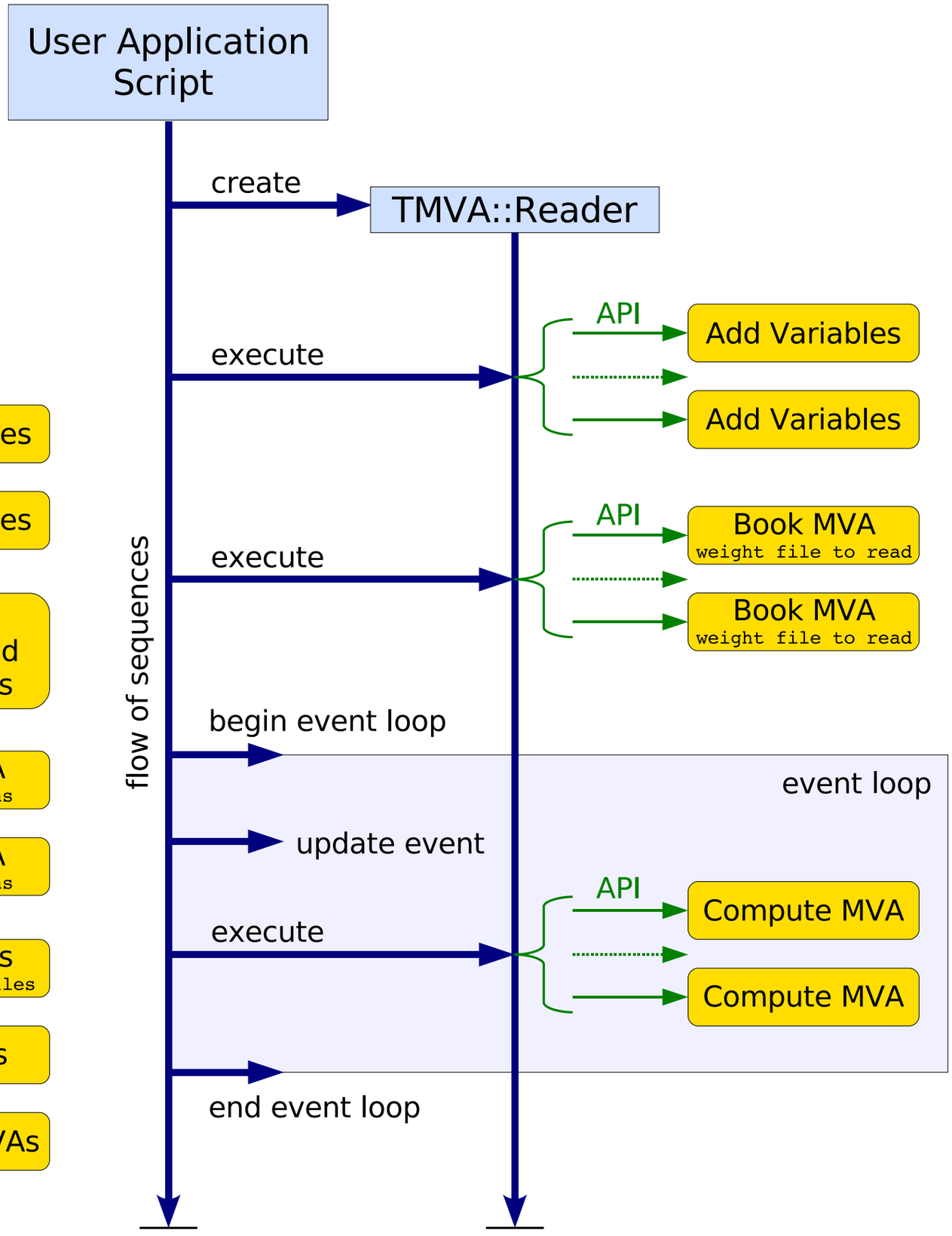
```
--- Factory                  : Evaluation results ranked by best signal efficiency and purity (area)
--- Factory                  : ------------------------------------------------------------------------------
--- Factory                  : MVA                       Signal efficiency at bkg eff.(error):       | Sepa-    Signifi-
--- Factory                  : Method:                   @B=0.01    @B=0.10    @B=0.30    ROC-integ. | ration:  cance:
--- Factory                  : ------------------------------------------------------------------------------
--- Factory                  : BDT                  : 0.289(10)  0.738(09)  0.925(05)    0.913    | 0.529    1.382
--- Factory                  : KNN                  : 0.309(10)  0.642(10)  0.884(07)    0.887    | 0.464    1.286
--- Factory                  : PDERS                : 0.212(09)  0.590(10)  0.829(08)    0.855    | 0.384    1.100
--- Factory                  : Cuts                 : 0.115(07)  0.453(11)  0.736(09)    0.795    | --       --
--- Factory                  : Likelihood           : 0.078(05)  0.373(10)  0.689(10)    0.756    | 0.198    0.520
--- Factory                  : ------------------------------------------------------------------------------
--- Factory                  :
--- Factory                  : Testing efficiency compared to training efficiency (overtraining check)
--- Factory                  : ------------------------------------------------------------------------------
--- Factory                  : MVA                       Signal efficiency: from test sample (from training sample)
--- Factory                  : Method:                   @B=0.01             @B=0.10             @B=0.30
--- Factory                  : ------------------------------------------------------------------------------
--- Factory                  : BDT                  : 0.289 (0.329)      0.738 (0.735)      0.925 (0.924)
--- Factory                  : KNN                  : 0.309 (0.329)      0.642 (0.707)      0.884 (0.886)
--- Factory                  : PDERS                : 0.212 (0.186)      0.590 (0.554)      0.829 (0.813)
--- Factory                  : Cuts                 : 0.115 (0.130)      0.453 (0.462)      0.736 (0.748)
--- Factory                  : Likelihood           : 0.078 (0.081)      0.373 (0.377)      0.689 (0.661)
--- Factory                  : ------------------------------------------------------------------------------
```

ランキング

Overtraining のチェック

TMVA workflow





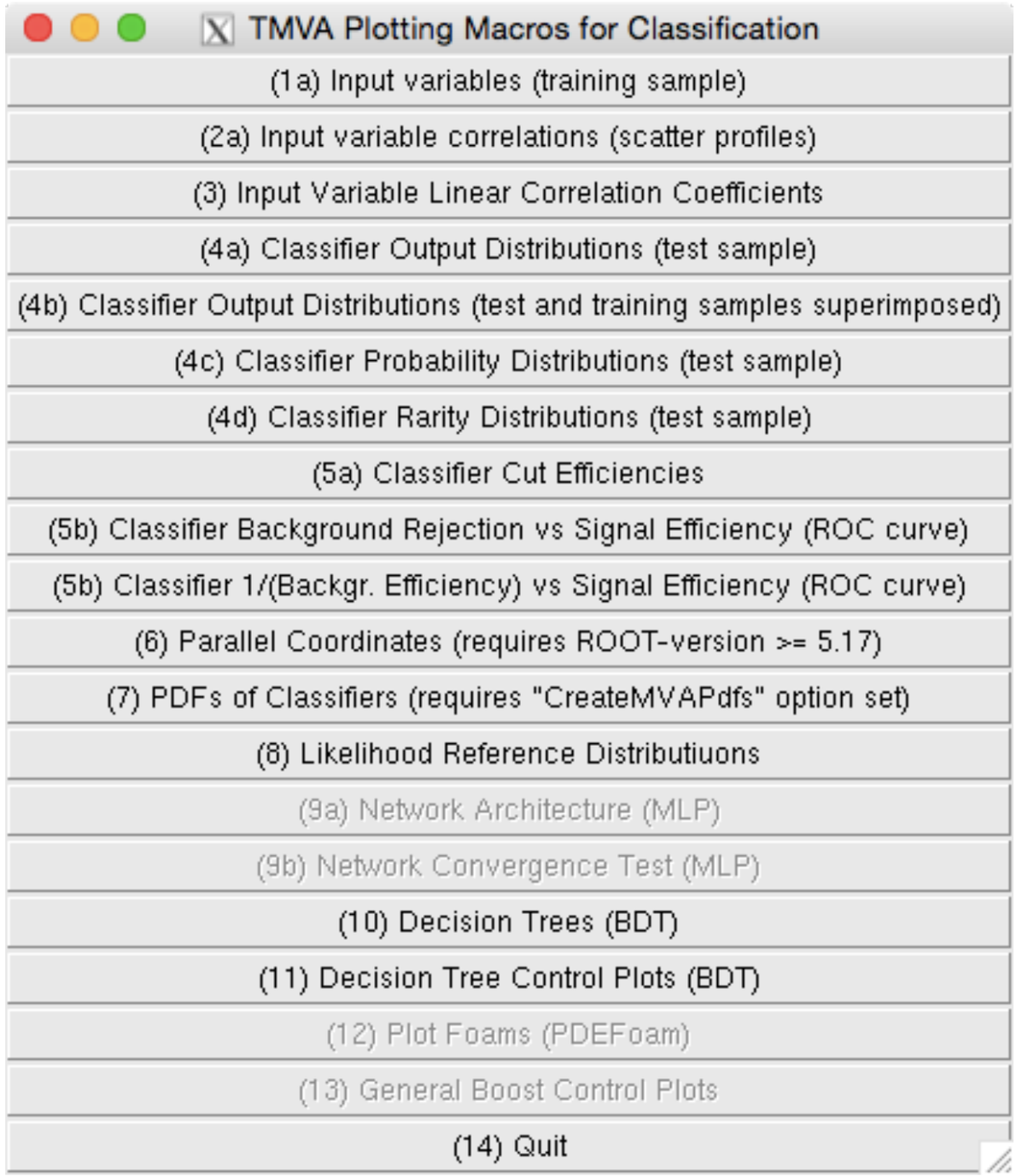
それぞれの変数の分布や相関を見てみる

- トレーニングが終わったらプロットはざっとGUIを使ってみることができます。

```
$ root -l
root [] TMVA::TMVAGui("TMVA.root");
```

- 現れたGUIのボタンを押すと各プロットが出てきます。
  - 最終的な「きれいな」プロットは自分でマクロを書いて、TMVA.rootからヒストグラムを開いて作るようにしましょう。

- チュートリアルでは順番にプロットを表示してこれが何かを説明していきます。



トレーニングした後、実際に利用するには

- 今回はこちらで用意したものを使います。
  - 難しく書いてありますが、基本とROOTのTreeの解析がわかっていれば自分のコードにも組み込めるはずです。

```
$ ./TMVAClassificationApplication
```

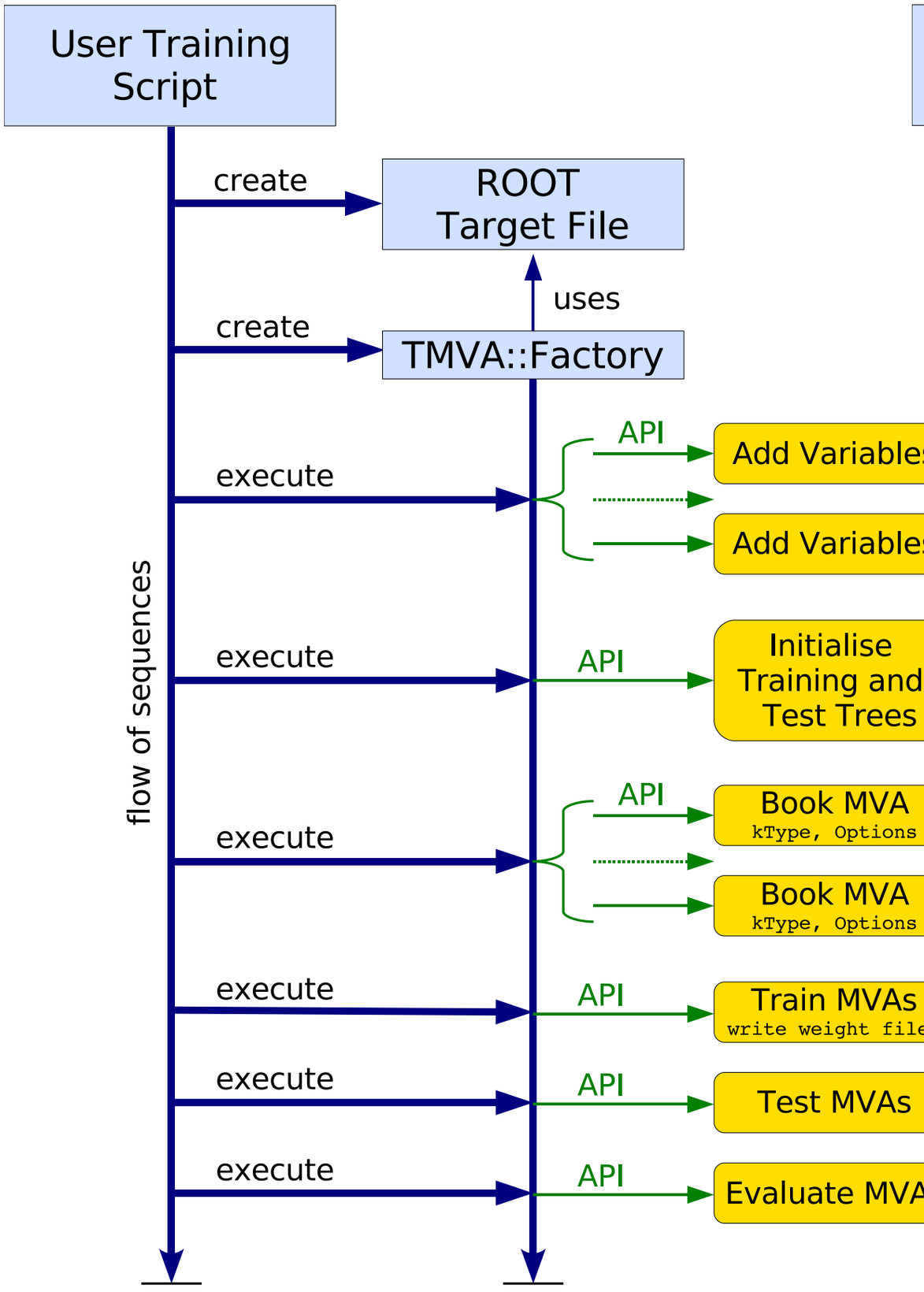
- コードを読んでみましょう。

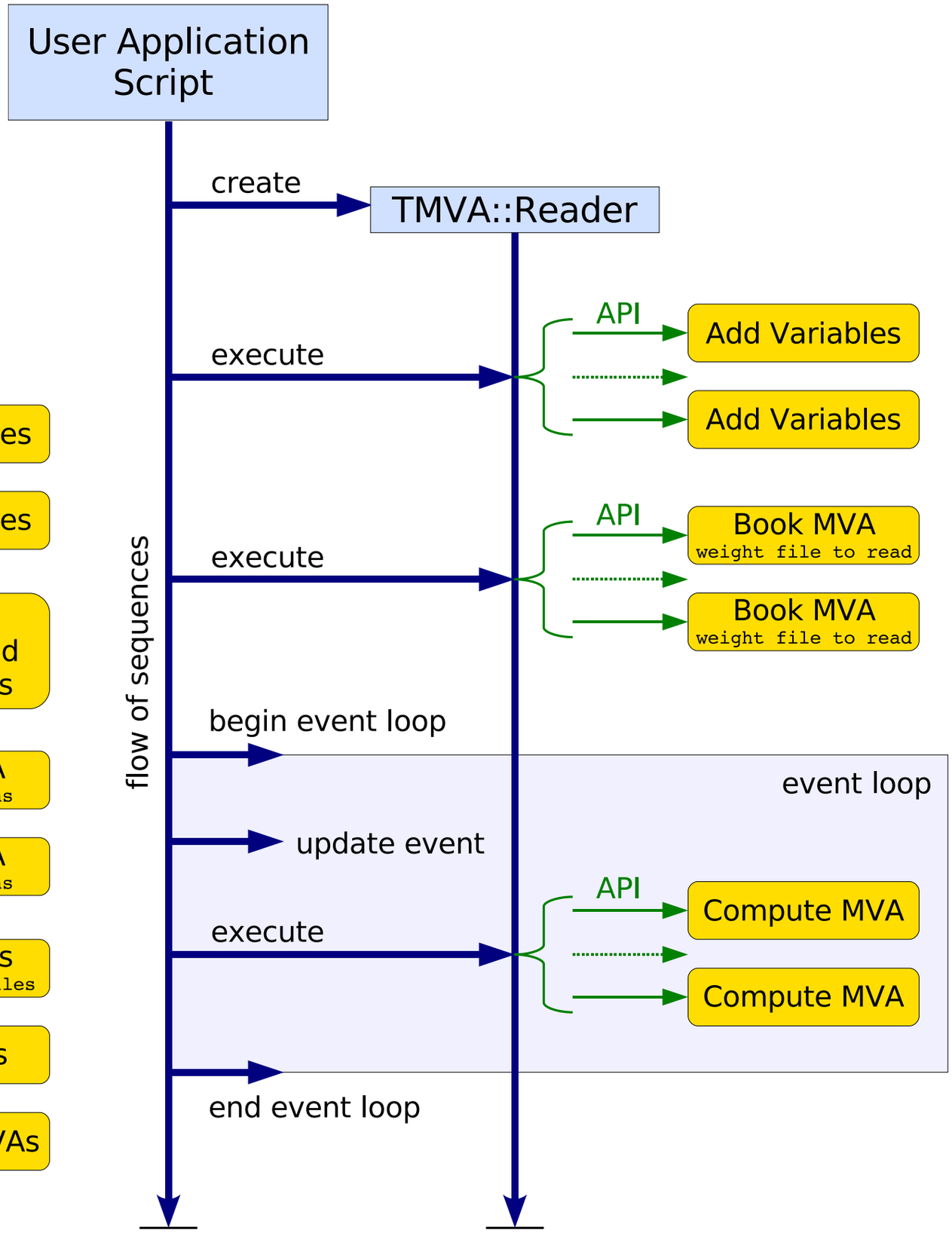
```
$ less TMVAClassificationApplication.C
```

- このプログラムではClassifierのOutputをヒストグラムに詰めています。
  - 読んだTreeが先程のSignal treeなので、Signal分布になるはずです・・・

```
$ root -l TMVApp.root
root [] .ls
root [] MVA_BDT->Draw();
```

TMVA workflow





まとめ&参考文献

- 今日は多変数解析の概略とTMVAのさわりだけ紹介しました。
  - 実際には目的があって、実際のサンプルがあって始めて試行錯誤するもの。
  - 上手く行けば解析の感度を上げられる可能性あり

- 本家ホームページ
  http://tmva.sourceforge.net/
- Users manual
  http://tmva.sourceforge.net/docu/TMVAUsersGuide.pdf
- これまで行われたtutorialや紹介トーク
  http://tmva.sourceforge.net/talks.shtml
- メーリングリスト
  https://sourceforge.net/p/tmva/mailman/
- TWiki tutorial
  https://twiki.cern.ch/twiki/bin/view/TMVA/WebHome