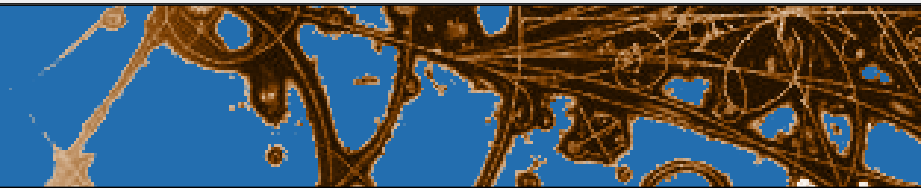


Introduzione all'Analisi Multivariata

III. Esempio di Analisi

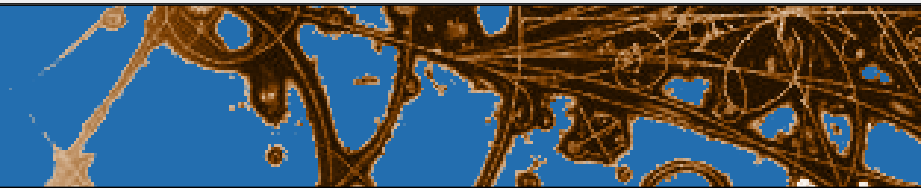
Pietro Biassoni[1,2]

[1]INFN Milano

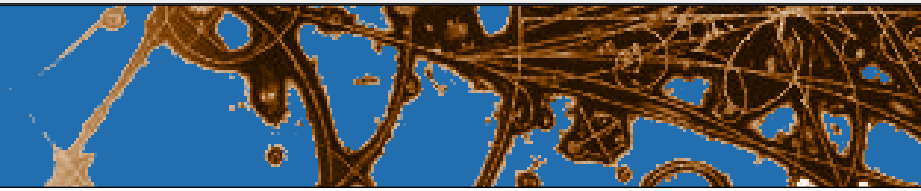
[2]Dipartimento di Fisica
Università degli Studi di Milano

Metodologie di Analisi dei Dati
15 Gennaio 2010

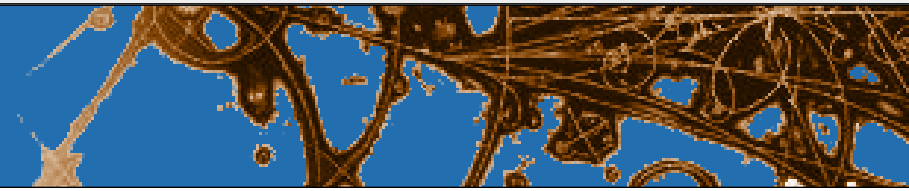
Outline



Outline



Esempio di Analisi

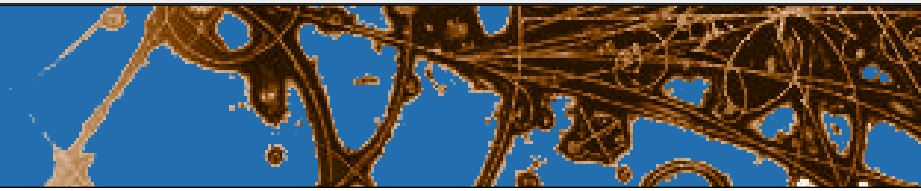


Lo scopo della lezione di oggi è provare ad affrontare un'analisi, mettendo in pratica tutte le nozioni imparate in queste lezioni.
Per prima cosa dobbiamo però imparare ad utilizzare una classe di ROOT estremamente versatile ed utile: `TSelector`

Cosa impareremo a fare:

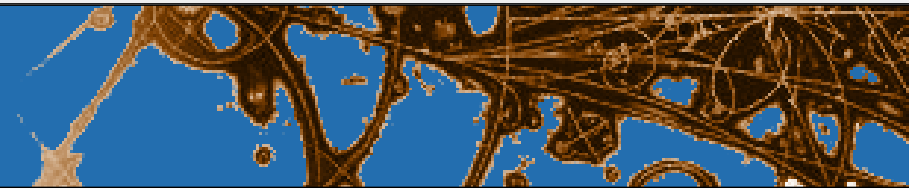
- Creare un selettore
- Processare degli eventi
- Scrivere un file di output

Outline



Le classi

Introduzione



Prima di esplorare come è fatto un selettore, una breve introduzione a cosa sono le classi.
Le classi sono delle strutture di C++ che ci permettono di creare (*istanziare*) oggetti che posseggono certe proprietà e sanno compiere determinate operazioni (*metodi*).

Ad esempio `TTree` è una classe, il comando

```
TTree t;
```

istanzia un oggetto "t" della classe `TTree`. Il comando

```
t.Print()
```

"dice" all'oggetto "t" di utilizzare il metodo `Print` che è definito nella classe `TTree`.

Le classi

Struttura

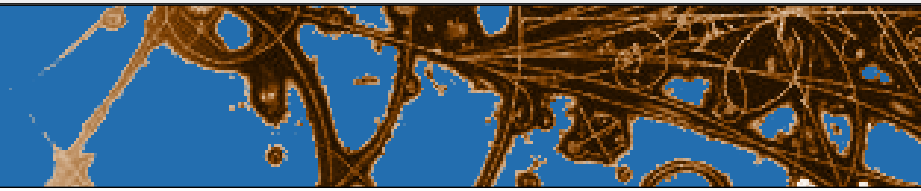
Normalmente (ma non è un obbligo) una classe è divisa in due file: myclass.h (*header*) e myclass.cc.

myclass.h

```
class myclass{
public:
// In ‘‘public’’ sono definiti tutti gli oggetti
// accessibili dall’esterno della classe.
Double_t var1;
void SetOption(const char* option);
...
private:
// In ‘‘public’’ sono definiti tutti gli oggetti
// accessibili solo dall’interno della classe.
Double_t _Integral();
...
}
```

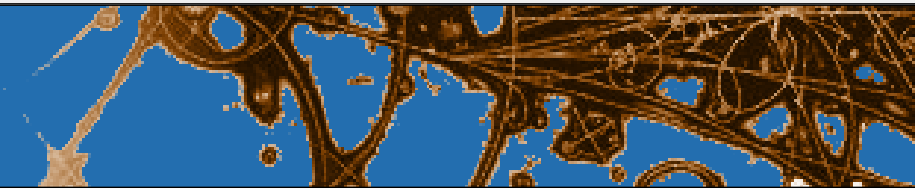
In myclass.cc si trovano i metodi definiti in myclass.h

Outline



TSelector

Introduzione

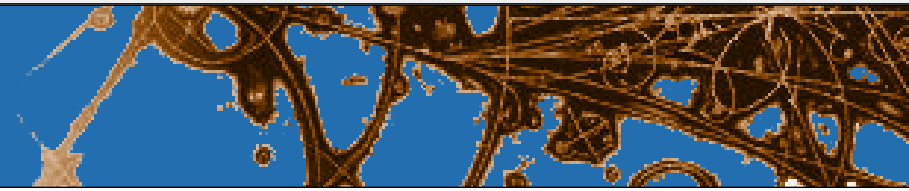


`TSelector` è una classe pensata per processare gli eventi di un `TTree`.

L'idea è che dopo alcune operazioni iniziali, una serie di istruzioni sempre uguale vengano eseguite utilizzando i valori contenuti nel `TTree` *evento per evento*.

Ad esempio voglio riempire un istogramma solo con gli eventi che hanno $var1 > 0.9$.
Il vantaggio rispetto all'utilizzo dei metodi `Draw` dei `TTree` è che le operazioni eseguite da un selettore sono programmate dall'utente e possono essere quindi complesse a piacere.

Creare un TSelector



Per la prima parte dell'esercitazione di oggi utilizzeremo il file `/home/comune/AnalisiDati/mva1_sample.root`
ROOT ha un comando in automatico per creare un selettore

```
> root -l mva1_sample.root
root [1] .ls
TFile** mva1_sample.root
TFile* mva1_sample.root
KEY: TTree bkg;1 Input to ML
KEY: TTree signal;1 Input to ML
root [2] signal->MakeSelector("selector") Info in
<TTreePlayer::MakeClass>:  Files:  selector.h and
selector.C generated from TTree:  signal
```

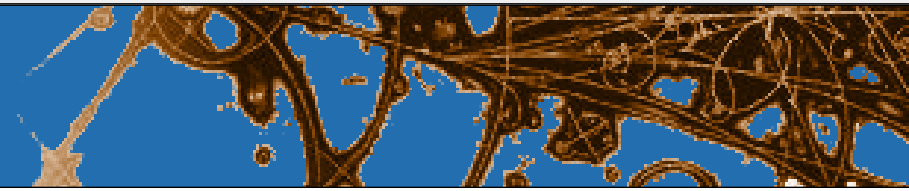
Questo genera nella cartella in cui vi trovate due files: `selector.h` e `selector.C`.

Esercizio

Create un selettore...

TSelector

selector.h



Analizziamo ora `selector.h`.
ROOT legge il `TTree` da cui crea il selettore e definisce una variabile e una `TBranch` per ogni `branch` presente nel `TTree`.

```
// Declaration of leaf types
Double_t var1;
Double_t var2;

// List of branches
TBranch *b_var1; //!
TBranch *b_var2; //!
```

Segue poi la definizione dei metodi (commenteremo quelli di nostro interesse nelle prossime slides).

TSelector

Init

Un metodo molto importante, che è definito in `selector.h`, è `Init` che associa le variabili alle `branch` del `TTree`.

```
void selector::Init(TTree *tree)
{
...
fChain->SetBranchAddress("var1", &var1, &b_var1);
fChain->SetBranchAddress("var2", &var2, &b_var2);
}
```

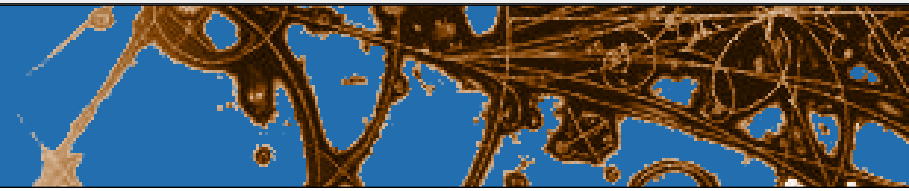
In questo modo, se nel `TTree` `var1` vale 5 per la prima entry del `TTree` i comandi

```
fChain->GetTree()->GetEntry(0);
cout <<var1 <<endl;
```

restituiranno come output "5".

Metodi di TSelector

Begin e Terminate



Spostiamoci ora in `selector.C`.
I metodi `Begin` e `Terminate` vengono eseguiti *una sola volta* all'inizio e alla fine del processo.
Questi sono i luoghi adatti, ad esempio, a inizializzare istogrammi, `TTree` etc., o a disegnare i risultati.

```
TH1F *h;

void selector::Begin(TTree *tree){
TString option = GetOption();
h = new TH1F("h","h",100,0,1);
}
...
void selector::Terminate(TTree *tree){
h->Draw();
}
```

Metodi di TSelector

Process

Il metodo `Process` contiene le istruzioni che devono essere eseguite evento per evento.
La prima istruzione del metodo deve essere

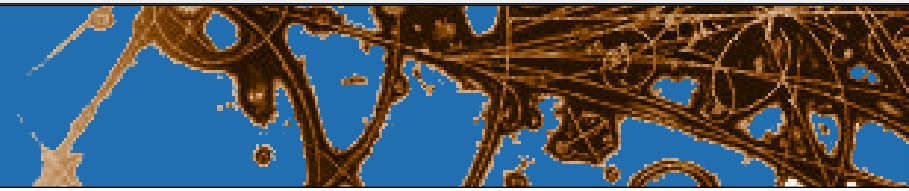
```
fChain->GetTree()->GetEntry(entry);
```

che serve a ottenere dal `TTree` i valori delle variabili per l'evento.
Seguono quindi le altre istruzioni.

Esempio

```
void selector::Process(Long64_t entry){
fChain->GetTree()->GetEntry(entry);
// Se var1<0.9 passo all'evento successivo
if(var1<0.9) return kFALSE;
..
// Riempio l'istogramma
h->Fill(var1);
return kTRUE;
}
```

Eseguire un TSelector



Una volta creato il selettore, per processare un `TTree` le istruzioni sono:

```
> root -l mva1_sample.root
root[0] signal->Process("selector.C+")
```

Il "+" dopo ".C" dice a ROOT che si vuole compilare il codice prima di eseguirlo, questo è utile poichè

- si evitano errori dovuti al codice;
- il programma viene eseguito più velocemente.

Esercizio

Create un selettore che disegni un istogramma di $var1$ per gli eventi che abbiano $var2 < 0.9$.

TTree in output a un TSelector

Ora vogliamo scrivere gli eventi che superano la selezione operata in `Process` su di un `TTree`.
In `Begin` dobbiamo aprire un `TFile`, inizializzare un `TTree` e associare i suoi `TBranch`.

```
*f = new TFile("output.root","RECREATE");
*t = new TTree("outTree","outTree");
t->Branch("var1",&var1,"var1/D");
```

In `Process` prima della fine dobbiamo riempire il `TTree`

```
t->Fill();
```

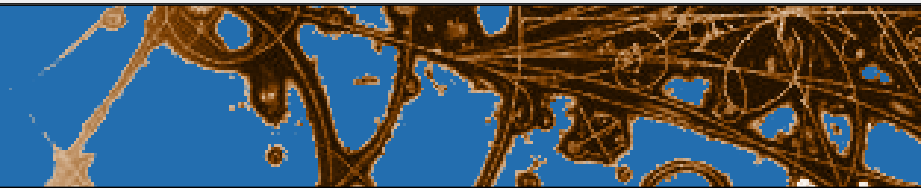
In `Terminate` dobbiamo scrivere il `TTree` e chiudere il file

```
t->Write();
f->Close();
```

Esercizio

Modificate il selettore in modo che scriva un file di output con tutte le variabili presenti nel `TTree` di input.

Outline



La mia prima analisi...

Materiale

Avremo ora a disposizione dei dati reali (gli stessi già usati per il ML fit) raccolti dall'esperimento *BABAR*.

Il file che serve per questa esercitazione si trova in
`/home/comune/AnalisiDati/mva3_sample.root`
Nel file sono presenti tre `TTree` (signal,bkg,data), che contengono due serie di variabili identificate dai prefissi:

- `cut_` Variabili su cui ottimizzare dei tagli;
- `fit_` Variabili da usare nel fit finale;

La mia prima analisi...

Cosa bisogna fare

- Bisogna utilizzare le variabili `cut_` per ottimizzare una serie di tagli. ATTENZIONE! Il segnale e il fondo non sono normalizzati, il rapporto nei campioni e segnale:fondo=10:1.
- Bisogna utilizzare le variabili `fit_` per (ri)fare il fit a ML e (ri)trovare i risultati dell'analisi.
- Attenzione! Se c'è "tanto" segnale e il fondo è facile da discriminare, può essere utile fare una ottimizzazione più "fisica", scegliendo di avere una più alta efficienza sul segnale.