Table of Contents

BTag Analysis package

Complete:

The `RecoBTag/Analysis` package

An analysis tool is provided in the package **`RecoBTag/Analysis`**. It is meant to produce most of the plots needed to debug and evaluate the performance of the tagging algorithms. It is an EDAnalyzer named **`BTagPerformanceAnalyzer`**. As such, the only thing needed to get it running on a sample is to put in the path. Note that both the JetTag and the Extended TagInfo produced by the tagger needs to be in the event! It produces most of the plots which were done for the Physics TDR. These are put into a root file, and eps and/or ps files can be produced.

An example configuration file can be found in RecoBTag/Analysis/test/bTagAnalysisExample.cfg or RecoBTag/Analysis/test/test.cfg. Default .cfi include files suitable for the various algorithms are placed in RecoBTag/Analysis/data.

Plots produced

Root file

All the plots produced are stored in a root file. This file is independent of the datafiles, and contains only the plots. In this file, the plots are organized in folders named according to the category of the plot and the (pt, eta) bin **`Category_eta_pt`**:

- Algorithm independent plots (folder name starting **`JetTag`**) based on JetTag object: jet and parton properties, true jet flavour, discriminator, efficiencies vs. discriminator cut, mistag rate vs. b-efficiency
- Algorithm specific plots (folder name starting e.g. **`TrackCounting`**) based on TagInfo object: plots of the variables used to compute the discriminator, e.g impact parameters. Some ExtendedInfo (e.g. TrackCounting) allow one to recompute the discriminator with different parameters, and these are also be shown here
- Differential plot (**`MisidForBEff`**): mistag rate at constant b-efficiency

In addition to the bins specified in the configuration, there will be plots for the pt or eta "slice" and for the global space (named **`GLOBAL`**). The list of folders might therefore look like this. In that example, two bins in eta (0.0, 1.4, 2.4) and two bins in pt (30., 50., 80.) were specified. The **`eta_pt`** naming in the folder name will be:

- **`GLOBAL`**: Whole eta/pt space
- **`PT_30-50`**: pt "slice", all eta
- **`ETA_0-1.4`**: eta "slice", all pt
- **`ETA_1.4-2.4__PT_30-50`**: eta/pt bin

The histograms in the folders have a similar naming convention. Their names also indicate which jet flavour is being studied (individual flavours = "D", "U","S","C","B" & "G" (gluon), combined flavours ("DUS","DUSG") and jet flavour not identified = "NI"). Key histograms include **`"FlavEffVsBeff_DUS"`** etc. showing tagging efficiency of one flavour (e.g. DUS) vs. b tagging efficiency' and **`"discrCutCond"`** showing the fraction of jets of a given flavour passing specified disriminator cut.

PS file

One PS file will be produced per root directory, with a name similar to the name of the directory (e.g. JetTagPlots__GLOBAL.ps).

EPS file

One EPS file will be produced per plot and bin, with a name containing the name of the quantity shown, and the bin it refers to, as shown above (e.g. discr__GLOBAL.eps for the discriminator).

Parameters

- In CMSSW 1.6.X, the module label of the JetTag collection(s) to use is specified in a string-vector:

```
vstring jetTagModuleLabel = { "jetProbabilityJetTags" }
```

Since the JetTag has a reference to the TagInfo it was computed from, additional plots with data from the TagInfo can also be produced. This is done via the bool useTagInfo and the string tagInfo, specifying the class name (excluding the final "TagInfo" part of its name) of the TagInfo objects. If the TagInfo plots are requested, all JetTag collections analyzed in the same analysis have to be be produced by TagInfos of the same type. If the TagInfo plots are not requested, the JetTag collections can be from different TagInfos.

```
bool useTagInfo = true
string tagInfo = "TrackIP"
```

- The name of the root file in which the histograms will be saved is given by **`rootfile`**. The output to eps and/or ps files of the plots is controlled by the boolean **`produceEps`** and **`producePs`**, and the files will be prepended with a string (which can also point to a directory):

```
string rootfile = "output.root"
bool produceEps = true
bool producePs = true
string psBaseName = ""
string epsBaseName= "anotherDirectory/"
```

- To reduce the number of plots, the **`bool allHistograms`** can be set to false (or true for the full set of histograms!):

- Eta and pt bins/ranges for the distributions:

```
vdouble etaRanges = { 0.0,  1.4, 2.4 }
vdouble ptRanges = { 30.0, 80.0, 120.0 ,400.0 }
```

- Jet selection requirements:

```
// reconstructed jet eta
double etaMin     = 0.0
double etaMax     = 2.4
// reconstructed jet pt
double ptRecJetMin = 30.0
double ptRecJetMax = 400.0
// parton pt
double ptPartonMin = 0.0
double ptPartonMax = 99999.0
```

- Range of the disciminator:

```
double discriminatorStart = -10.1
double discriminatorEnd   = 10.1
```

- The value of constant b-efficiency at which to compute the differential plots, misstag rate versus pt and eta (reasonable values: lifetime based: 0.5 soft lepton : 0.05):

```
double effBConst = 0.5
```

- Binning and range for the misid. vs. efficiency plots: for all lifetime based algos this should be (100, 0.005 , 1.005). For the soft lepton tags the upper bound should (roughly) correspond to the probability to find a lepton within the jet:

```
int32   nBinEffPur  = 100
double startEffPur = 0.005
double endEffPur   = 1.005
```

- The jet identification is to be configured via a PSet to be called **`jetIdParameters`**:

```
PSet jetIdParameters = {
  bool fillPartons = true
  bool fillHeavyHadrons = false
  bool fillLeptons =  false
  double coneSizeToAssociate = 0.3
  bool physicsDefinition = false
  bool rejectBCSplitting = false
  vstring vetoFlavour = {  }
}
```

  - **`physicsDefinition`**: Choice of the physics definition (**`true`**), or algorithmic definition (**`false`**)
  - Flavours can be vetoed by adding a string in the **`vetoFlavour`** string vector: "B", "C", "UDS", "G"

Job splitting

It may be needed to run on different data samples or to split a dataset into several jobs (e.g. when using CRAB). The final histograms will have to be the combination of the different samples/jobs, and the performance plots (efficiency vs. mistag, etc) have to be calculated from the final disciminator distributions. Two possibilities exist:

- Update an existing root file: The existing root file will be read at the beginning of the job, and the events will be added to the various histograms. The performance histograms will be recomputed at the end of the job from the final disciminator distributions. The parameters which control this are the boolean **`update`** and the string **`inputfile`** specifying the name of the input root file. The final plots will be placed in the root file specified by the string **`rootfile`**, as above:

```
bool update = true
string inputfile = input_file.root"
```

- Addition of existing root files: The jobs can be run, and the root files merged when all are available. The root executable **`hadd`** is to be used

to add all the histograms: `hadd final.root file1.root file2.root file3.root ...` This will add all the distributions, but can not update correctly the performance plots. To produce the correct final performance plots from the final discriminator distributions, re-run the performance analyzer, giving as input file the root file produced by **`hadd`**, but without running on any event. Use an EmptySource to avoid accessing any CMSSW datafile:

```
source = EmptySource {
   untracked int32  maxEvents = 0
}
```

```
(..)
  bool update = true
  string inputfile = final.root"
```

Please note that in 15X and following this has to change, since the parameter **`maxEvents`** is no more in the source. The previos code has to become

```
  source = EmptySource {}

  untracked PSet maxEvents = {untracked int32 input =0}

(..)
  bool update = true
  string inputfile = final.root"
```

Tags for CMSSW 1.6.X and CSA07

To process CSA07 data with CSA1.6.X, the following tags have to be used:

- Tag V00-03-08 of /RecoBTag/Analysis/
- Branch B160_V00-04-06 of /CMS.PhysicsTools/JetMCAlgos
- Tag V01-01-02 of /RecoBTag/MCTools

```
cvs co -r V00-03-08 RecoBTag/Analysis
cvs co -r B160_V00-04-06 PhysicsTools/JetMCAlgos
cvs co -r V01-01-02 RecoBTag/MCTools
```

This includes the code to use the flavour from the JetMC tools developed by Attilio (which uses the GenParticleCandidate collection).

Tags for CMSSW 1.6.X (non-CSA07)

The latest tags to be used with CMSSW 1.6.X are:

- Tag V00-03-08 of /RecoBTag/Analysis/
- Tag V01-01-02 of /RecoBTag/MCTools

```
cvs co -r V00-03-08 RecoBTag/Analysis
cvs co -r V01-01-02 RecoBTag/MCTools
```

Tags for CMSSW 2.1.X

Please refer to this page. The validation is now done in the framework of DQM.

In case you still want to use RecoBTag/Analysis and do not have HepMC, you have to insert the following in your pyton config file:

```
process.load("CMS.PhysicsTools.JetMCAlgos.CaloJetsMCFlavour_cfi")

process.bTagAnalysis.fastMC = True
process.bTagAnalysis.jetMCSrc = 'IC5byValAlgo'

process.path = cms.Path(
       process.myPartons *
       process.iterativeCone5Flavour *
etc...
```

Review Status

Editor/Reviewer and date	Comments
Main.speer - 30 Oct 2006	page author (Thomas Speer)
JennyWilliams - 28 Feb 2007	moved into SWGuide

Responsible: Main.speer
Last reviewed by: Reviewer

This topic: CMSPublic > SWGuideBTagAnalysis
Topic revision: r22 - 2011-12-06 - KatiLassilaPerini