Table of Contents

Instructions to work on Zprime analysis

Introduction

The following instructions help you to setup a working area with the Zprime analysis in cmsgridui1.pg.infn.it machine. The analysis to search for a heavier gauge boson in tau channel done in the following steps:

```
1. AOD (from DAS) -> 2. patTuple -> 3. NTuple -> 4. Analysis -> 5. Plots, limits, discovery etc.
```

For details of the instructions from step1 to step2 can be found in the following twiki:
https://twiki.cern.ch/twiki/bin/viewauth/CMS/HighMassDitau2012

We will start from step2, where we have already a patTuple. The following instructions will guide you to perform the production of the nTuple (step3), and analyze them (step4). So our assumption is that we have already a patTuple already available in the d'Cache of Perugia.

Setting up CMSSW environment

If you have never used CMSSW in cmsgridui1, first you have to do the following setup:

```
export SCRAM_ARCH=slc5_amd64_gcc434; # for CMSSW_4_2_X
export SCRAM_ARCH=slc5_amd64_gcc462; # for CMSSW_5_2_4 in cmsgridui1 ( installed in /opt/exp_soft
export CLHEP_BASE_DIR=/afs/cern.ch/cms/sw/slc5_ia32_gcc434/external/clhep/2.0.4.2/;
source /opt/exp_soft/cms/cmsset_default.sh;
```

Get a CMSSW release area

Then you have to setup CVSROOT, so that we can download codes from CVS in cmsgridui1:

```
export CVSROOT=:pserver:anonymous@cmssw.cvs.cern.ch:/local/reps/CMSSW
```

While in FNAL you have to do:

```
cmscvsroot CMSSW
cvs login
Password: 98passwd
```

To see the available CMSSW versions in cmsgridui1 and/or FNAL, use:

```
scram list
```

Take one of the, like CMSSW_5_3_3, and do:

```
cmsrel CMSSW_5_3_3
cd CMSSW_5_3_3/src/
cmsenv
```

At this point we need the code, that we have to run to get our nTuple. We need two modules ZprimeAnalysis, for accessing all the necessary information for our analysis and PrimaryVertexProducer for re-fitting the vertex with and without two leading Pt leptons.

```
cvs co -dZprimeAnalysis UserCode/ASaha/ZprimeAnalysis
mv ZprimeAnalysis SusyCAF
mkdir SUSYBSMAnalysis
mv SusyCAF/ SUSYBSMAnalysis/
cvs co -r V01-06-05 RecoVertex/PrimaryVertexProducer
scram b -j 8
```

You have to be a little patient, as it may take some time, and if you see the following message:

```
gmake[1]: Entering directory `/uscms_data/d2/anirban/CMSSW_5_3_3'
@@@@ Refreshing Plugins:edmPluginRefresh
>> Creating project symlinks
>> Done python_symlink
>> Compiling python modules cfipython/slc5_amd64_gcc462
>> Compiling python modules python
>> Compiling python modules src/RecoVertex/PrimaryVertexProducer/python
>> Compiling python modules src/SUSYBSMAnalysis/SusyCAF/python
>> All python modules compiled
>> Pluging of all type refreshed.
gmake[1]: Leaving directory `/uscms_data/d2/anirban/CMSSW_5_3_3'
```

In case you are familiar to look at the structure of the code in browser, you can have a look into the following area of my CVS area:

http://cmssw.cvs.cern.ch/cgi-bin/cmssw.cgi/UserCode/ASaha/ZprimeAnalysis/

Now at this point, the code is successfully build, now we need some patTuple to run our code.

```
cd SUSYBSMAnalysis/SusyCAF/test/
rm -rf * , as we will not need any files
cp /uscms_data/d2/anirban/CMSSW_5_3_2_patch4/src/SUSYBSMAnalysis/SusyCAF/test/globaltag.py .
vi globaltag.py
cmsRun globaltag.py , to test on Z' mass = 1 TeV patTuple, before sending the CRAB jobs
```

```
cp /uscms_data/d2/anirban/CMSSW_5_3_3/src/SUSYBSMAnalysis/SusyCAF/test/crab.cfg .
```

Change the globaltag.py and crab.cfg files to include the data set you want to run on.

*** The change you must do is to run on the data you have to comment out the "process.susycafgen" line since there is no GEN level information in Data ***

Then set up the CRAB environment in FNAL:

```
setenv SCRAM_ARCH slc5_amd64_gcc462
cmsenv
source /uscmst1/prod/grid/CRAB/crab.csh
source /uscmst1/prod/grid/gLite_SL5.csh
```

While the similar command in cmsgridui1 are:

```
source /afs/cern.ch/cms/LCG/LCG-2/UI/cms_ui_env.sh
cmsenv
source /afs/cern.ch/cms/ccs/wm/scripts/Crab/crab.sh
```

When the CRAB jobs are DONE, check with:

```
crab -status -c <Task_name>
```

When all jobs are Done, do:

```
crab -getoutput 1-last_job_no -c <Task_name> , again do
crab -status -c <Task_name> shows the summary of the jobs,
crab:  ExitCodes Summary
 >>>>>>>>> 30 Jobs with Wrapper Exit Code : 0
         List of jobs: 1-30
It means all the jobs are done correctly, then do:
crab -report  -c <Task_name>
Will show you how many events are there in the Ntuples.
```

At this point the Ntuple is ready to be analyzed, so all the steps up to step 3 is DONE.

==========

Now put the following patTuple which is already in d'Cache of Perugia:

```
'dcap://gridse3.pg.infn.it:22125/pnfs/pg.infn.it/data//cms/store/user/romeo/DYToEE_M20_CT10/skimP
```

Put it within i following place:

```
fileNames = cms.untracked.vstring( INPUT FILE )
```

It will create out nTuple, to be ready to read by the framework, described below.

Download the code from SVN

Go to your area in data06 of cmsgridui1 machine, like mine is /data06/users/anirban/

```
svn co svn+ssh://USER@svn.cern.ch/reps/zpmdi-tau nameDir
```

asks two times the password for lxplus. You can also view the class structure in your browser:

https://svnweb.cern.ch/cern/wsvn/zpmdi-tau

Configure the path and set up

```
cd nameDir
./configure.py
```

asks for different options, put 1, CMS PG

[1] PG CMSGRIDUI

It will create a file called setup.sh and setup.csh

Depending on your shell use:

```
source setup.sh  ( for bash shell and .csh for tcshrc shell )
```

Compile the code

```
cd framework
```

Now, go to a CMS release and do cmsenv:

```
e.g.,
cd ~/CMSSW_4_2_8/src/;cmsenv; cd -
```

It will set all the environment right, otherwise you may have problem with complation.

```
make -f Debug ( in framework directory )
and then
cd ../Zprime
make -f Debug
```

At this point all the codes are fully compiled.

Test run of the compiled code (interactively)

Now to run on a Monte-Carlo patTuple, you need to go to the folder called scripts:

```
cd scripts/
```

Open the python script p1_mutau_all_mcs.py to see, what are the input files it will run on:

```
vi p1_mutau_all_mcs.py
allMCs = [DYToTauTau_M_20_TuneZ2star_8TeV_pythia6_tauola]
```

The "allMCs" section shows you the input root files you want to run on, the list of those files are in :

```
python/DYToTauTau_M_20_TuneZ2star_8TeV_pythia6_tauola.py
```

Here you can see after the list of files we have :

```
        CrossSection = 142.77, # It is "NLO cross-section from theoretical calculation times the
    Events = 148191, #The number of events the samples have in Ntuples
)
```

It returns you the weight for 100 inverse pb, you can also check in the following class:

```
vi src/common/ZpmSelection.cc
w=ev.GetEventWeight();
```

Since it returns you the weight for 100 inverse pb, do not forget to re-weight your MCs with the amount of data you have:

```
######IMORTANT##############
#Re-weight this factor correctly to the ipbs you have
reweight(allMCs,522.)
```

To run the python script interactively, you have to do:

```
./p1_mutau_all_mcs.py
```

Batch job submission

It is also possible to run them on batch mode, which is much faster:

```
 ./XXXX.py -b -n NUM,
```

where NUM is the number of root files you want to analyze in each job.

For an example, if there are 158 input root files, and you put NUM = 10,

```
./p1.py -b -n 10
```

it will create 16 jobs over 158 files.

-- AnirbanSaha - 04-Jun-2012

Modify lepton selection

A standard selection is implemented in the framework for these objects: CommonMuons, CommonElectrons, CommonTaus.

Anyway it is possible to change each selection parameter using following commands in the configuration file (e.g. p1_alfredo_test.py):

For muons following parameters are avaible:

```
conf.Common.Muons.PtCut = double
conf.Common.Muons.EtaCut = double
conf.Common.Muons.GlobalMuon= bool
conf.Common.Muons.NormChi2= double
conf.Common.Muons.ChamberHit= int
conf.Common.Muons.ImpactParameter= double
conf.Common.Muons.MatchedStations= int
conf.Common.Muons.TrackerLayers= int
conf.Common.Muons.ValidPixelHits= int
conf.Common.Muons.rel_iso_mu=double
```

While for electrons:

```
conf.Common.Electrons.PtCut = double
conf.Common.Electrons.EtaCut = double
conf.Common.Electrons.IsEBElePAT=bool
conf.Common.Electrons.EtEle=double
conf.Common.Electrons.EtaSCEle=double
conf.Common.Electrons.DeltaEtain=double
conf.Common.Electrons.DeltaPhiin=double
conf.Common.Electrons.HsuEEle=double
conf.Common.Electrons.ScE25Max55=double
conf.Common.Electrons.ScE15Ele55=double
conf.Common.Electrons.TrackPtIsoEle=double
conf.Common.Electrons.LostHitEle=int
conf.Common.Electrons.IsEcalDrivenElePAT=bool
```

Finally for Taus:

```
conf.Common.Taus.PtCut=double
conf.Common.Taus.EtaCut=double
conf.Common.Taus.IsTauAgainstMuon=bool
conf.Common.Taus.IsTauAgainstElectronTight=bool
conf.Common.Taus.IsTauByDecay=bool
conf.Common.Taus.IsTauIdByMediumCombinedIsolationDBSumPtCorr=bool
```

For information about parameters you can look at:

https://twiki.cern.ch/twiki/bin/view/CMS/HEEPElectronID

https://twiki.cern.ch/twiki/bin/view/CMSPublic/SWGuideMuonId

BDT analysis for ee/mumu channel

In order to analyze data with BDT, training weight files are needed (for info about BDT: http://tmva.sourceforge.net/). In the framework it's possible to train our BDT creating weight files and run the analysis at the same times, once we have files for training. Input file for training has to be construct with the same structure as /data06/users/ciangottini/TestZ/Zprime/scripts/training_tautau.root

An example is shown in /data06/users/ciangottini/TestZ/Zprime/scripts/EEMuMu_ch.py

```
Selection=PSet(

TrainFile=False,       #False if we already have file with the structure for the training, True w

TrainSampleEle=False,            #True will train BDT for ee events
    SignalTrainingEle = "training_tautau.root",    # List of training and test file
    BkgTrainingEle="train_EE_200-800.root",
    SignalTestEle ="training_tautau.root",
    BkgTestEle="training_EE.root",
    OutWeightEle="/data06/users/ciangottini/TestZ/Zprime/scripts/weights/MyBDT_BDTall.weights.xml
    OutFileEle="test.root", #Results of training
    LeptonTypeEle="ele",

   TrainSampleMuon=False,     #True will train BDT for mumu events
   SignalTrainingMu = "training_tautau.root",
    BkgTrainingMu="training_MuMu.root",
    SignalTestMu ="training_tautau.root",
    BkgTestMu="training_MuMu.root",
    OutFileMu="testMu.root",
    LeptonTypeMu="muon",
   OutWeightMuon="./weights/MyBDTMu_BDTall.weights.xml" #name and location of the weight file for
   # OutWeightMuon="./MyBDTMu_BDTall.weights.xml"

   )
```

When training is done, the BDT analysis proceeds and in the output file there are 6 histos (example /data06/users/ciangottini/TestZ//scripts/Sel_DYtoEE_M-800.root):

BDT response -> TH1F x2 (electrons and muons)

M_inv -> TH1F x2 (electrons and muons)

BDT response vs M_inv -> TH2F x2 (electrons and muons)

Roofit macro for ee/mumu channel

To make the same procedure descripted in slide 13 and 14 https://indico.cern.ch/getFile.py/access?contribId=1&resId=0&materialId=slides&confId=209394 following file could be usefull : /data06/users/ciangottini/TestZ/Zprime/scripts/ShapeEE.cc

You'll find inside comments step by step.

-- DiegoCiangottini - 09-Jul-2012

This topic: Sandbox > PGZprimeAnalysis
Topic revision: r22 - 2013-09-10 - AnirbanSaha