ZPrimeCombine
Interface to the Higgs Combine Tool for the $Z^{'} \rightarrow \ell\ell$ search

User Manual

Jan-Frederik Schulte

Version 0.1, 2017/13/01

Contents

1 Introduction

The ZPrimeCombine package, to be found at TODO: Move to gitlab and link here, provides an interface between the experimental results of the $Z^{'} \rightarrow \ell\ell$ analysis and the Higgs Combine Tool. This tool, referred to simply as "combine" going further, is in turn an interface to the underlying statistical tools provided by RooStats. This document aims to summarize the functionality of the the tool and give instructions how to use it to derive limits and significances for the analysis.

2 Setup

The current implementation in the package is based on version `v6.3.0` of combine. CMSSW_7_4_7 is used to set the environment, but this is only to ensure a consistent version of ROOT, combine does not rely on CMSSW itself. Combine is installed using the following commands

```
export SCRAM_ARCH=slc6_amd64_gcc491
cmsrel CMSSW_7_4_7
cd CMSSW_7_4_7/src
cmsenv
git clone https://github.com/cms-analysis/HiggsAnalysis-CombinedLimit.git HiggsAnalysis/CombinedLimit
cd HiggsAnalysis/CombinedLimit
git fetch origin
git checkout v6.3.0
scramv1 b clean; scramv1 b
```

Detailed documentation of combine can be found on this Twiki page.

To finish the setup, just clone the ZPrimeCombine repository. At the moment, there are no version tags, so simply use the master branch. The framework is python based, so no need for compiling.

3 Usage

The central entry point to the framework is the script `runInterpretation.py`. It steers both the creation of the inputs given to combine as well as the execution of it, either locally or via batch/grid jobs. Let's have a look at its functionality:

```
Steering tool for Zprime -> ll analysis interpretation in combine

optional arguments:
  -h, --help            show this help message and exit
  -r, --redo            recreate datacards and workspaces for this
                        configuration
  -w, --write           create datacards and workspaces for this configuration
  -b, --binned          use binned dataset
  -s, --submit          submit jobs to cluster/GRID
  --signif              run significance instead of limits
  --LEE                 run significance on BG only toys to estimate LEE
  --frequentist         use frequentist CLs limits
  --hybrid              use frequenstist-bayesian hybrid methods
  -e, --expected        expected limits
  -i, --inject          inject signal
  --crab                submit to crab
  -c CONFIG, --config CONFIG
                        name of the congiguration to use
  -t TAG, --tag TAG     tag to label output
  -m MASS, --mass MASS  mass point
```

In the following the use of these options for different purposes is described. The most important parameter is -c, which tells the framework, which configuration file to use for steering. It is the only argument which is mandatory to give, and the name of the configuration will be used to tag all inputs and results. The configuration files themselves are discussed in the next section. Another universal option is the -t option which can used to tag the in- and output of the tool.

3.1 Input creation

The first task of the framework is to create the datacards and workspaces used as inputs for combine. To provide the framework with the necessary information, the user has to provide two different types of inputs. All experimental information is located in the input/ directory. Here, for each channel of the analysis, there is one channelConfig_channelName.py file. For example, for the barrel-barrel category in the dimuon channel for the ICHEP 2016 result, it looks like the example shown below. Important things are commented throughout.

```
import ROOT,sys
ROOT.gROOT.SetBatch(True)
ROOT.gErrorIgnoreLevel = 1
from ROOT import *

nBkg = -1 # If set to minus 1, the overall background normalization is just set ↩
    to the total number of observed events.

dataFile = "input/dimuon_13TeV_2016_ICHEPDataset_BB.txt" # list of masses of all ↩
    observed events over the minimal mass threshold

def provideSignalScaling(mass): # This function provides the scaling from the ↩
    observed number of events to the cross section ratio
    nz   =  21152
    nsig_scale = 1017.9903604773663       # prescale/eff_z (123.685828798/0.1215)↩
        -->derives the lumi
    eff = signalEff(mass)
    result = (nsig_scale*nz*eff)
    return result

def signalEff(mass): # provides the signal efficiency

    trig_a = 0.9878
    trig_b = -7.8162E-08
    trig_c = 0.

    eff_a     = 1.54
    eff_b     = -6.72E3
    eff_c     = 4.69E3
    eff_d      = −6.08E−5
    return  (eff_a+eff_b/(mass+eff_c)+mass*eff_d)*(trig_a + trig_b*mass + trig_c*↩
        mass*mass)



def signalEffUncert(mass): # gives the signal efficiency uncertainty

    effDown = 1.+(0.03**2 + 0.01**2)**0.5

    return [1./effDown,1.01]



def provideUncertainties(mass): # returns the different uncertainties

    result = {}

    result["sigEff"] = signalEffUncert(mass)
    result["massScale"] = 0.01
    result ["bkgUncert"] = 1.4

    return result



def getResolution(mass): # returns the mass resolution

    return 1.9E−02 + 2.4E−05*mass −2.4E−09*mass*mass


def loadBackgroundShape(ws): # creates the background PDF and adds it to the ↩
    workspace

    bkg_a = RooRealVar('bkg_a_dimuon_BB','bkg_a_dimuon_BB',28.51)
    bkg_b = RooRealVar('bkg_b_dimuon_BB','bkg_b_dimuon_BB',−3.614E−4)
    bkg_c = RooRealVar('bkg_c_dimuon_BB','bkg_c_dimuon_BB',−1.470E−7)
    bkg_d = RooRealVar('bkg_d_dimuon_BB','bkg_d_dimuon_BB',6.885E−12)
    bkg_e = RooRealVar('bkg_e_dimuon_BB','bkg_e_dimuon_BB',−4.196)
    bkg_a.setConstant()
    bkg_b.setConstant()
    bkg_c.setConstant()
    bkg_d.setConstant()
    bkg_e.setConstant()
    getattr(ws,'import')(bkg_a,ROOT.RooCmdArg())
    getattr(ws,'import')(bkg_b,ROOT.RooCmdArg())
    getattr(ws,'import')(bkg_c,ROOT.RooCmdArg())
    getattr(ws,'import')(bkg_d,ROOT.RooCmdArg())
    getattr(ws,'import')(bkg_e,ROOT.RooCmdArg())

    # background systematics
    bkg_syst_a = RooRealVar('bkg_syst_a','bkg_syst_a',1.0)
    bkg_syst_b = RooRealVar('bkg_syst_b','bkg_syst_b',0.000)
    #bkg_syst_b = RooRealVar('bkg_syst_b','bkg_syst_b',−0.00016666666666)
    bkg_syst_a.setConstant()
    bkg_syst_b.setConstant()
    getattr(ws,'import')(bkg_syst_a,ROOT.RooCmdArg())
    getattr(ws,'import')(bkg_syst_b,ROOT.RooCmdArg())

    # background shape
    ws.factory("ZPrimeMuonBkgPdf::bkgpdf_dimuon_BB(mass_dimuon_BB, ↩
        bkg_a_dimuon_BB, bkg_b_dimuon_BB, bkg_c_dimuon_BB,bkg_d_dimuon_BB,↩
        bkg_e_dimuon_BB,bkg_syst_a,bkg_syst_b)")
    ws.factory("ZPrimeMuonBkgPdf::bkgpdf_fullRange(massFullRange, bkg_a_dimuon_BB↩
        , bkg_b_dimuon_BB, bkg_c_dimuon_BB,bkg_d_dimuon_BB,bkg_e_dimuon_BB,↩
        bkg_syst_a,bkg_syst_b)")

    return ws
```

For each channel of the analysis (i.e. for each subcategory of the dielectron and dimuon channels), one such config has to be provided. The other input to the tool is located in the `cfgs/` directory. Here, the `scanConguration_ConfigName.py` files contain all information needed to steer the actual interpretation, setting the channels to be considered, the mass range to be scanned, and similar features. Given here is the example for the combination of the three dimuon subcategories for the ICHEP 2016 dataset.

```
leptons = "mumu" # lepton combination. Needed for correct labelling in limit ↩
    plots
systematics = ["sigEff","bkgUncert","massScale"] # list of uncertainties to be ↩
    considered
correlate = False # are the uncertainties correlated between channels?
masses = [[5,400,1000], [10,1000,2000], [20,2000,4500]] # mass ranges for the ↩
    observed limit. For example 5 GeV steps between 400 and 1000 GeV, etc.
massesExp = [[100,400,600,1000,1,500000], [100,600,1000,500,2,500000], ↩
    [250,1000,1500,100,10,50000], [250,2000,4600,100,10,500000]] #mass ranges for↩
     the expected limit. After the mass binning, the latter three integers steer ↩
    the job creation for CRAB, i.e 100 GeV steps between 400 and 600 with 1000 ↩
    jobs, 1 toy per job and 500k steps in the Markov Chain

libraries = ["ZPrimeMuonBkgPdf_cxx.so","PowFunc_cxx.so"] # Libraries for PDFs to ↩
    be given to combine

channels = ["dimuon_BB","dimuon_BEpos","dimuon_BEneg"] # channels to be combined
numInt = 500000 # number of iterations in the MarkovChain
numToys = 10 # number of toys for observed limits
exptToys = 10 # number of toys for expected limits
width = 0.006 # assumed width of the resonance
submitTo = "Purdue" # computing resource for batch jobs. Right now only Purdue is↩
     explicitly supported, but this should work on all clusters with qsub.

CB = False  # use crystal ball in signal shape?
signalInjection = {"mass":2000,"width":0.006,"nEvents":10,"CB":True} # parameters↩
     for signal injection
```

Using this input, the framework will create first the datacards for the single channels and afterwards combined datacards. For local running, this can be triggered by running with the `-w` or `-r` options. In the first case, the datacards are produced and the program is exited without performing any statistical procedures. In the latter case, the datacards are reproduced on the fly before performing statistical interpretations. If a local batch system is used, the input will be created inside the individual jobs to increase performance. When tasks are submitted to CRAB, the input is created locally.

3.2 Running statistical procedures

If not called with the `-w` option (which will only write datacards, see above), the default behaviour of `runInterpretation.py` is to calculate observed limits using the Bayesian approach. For this, the mass binning and the configuration of the algorithm given in the scan configuration is given. There are numerous command line options to modify the statistical methods used

- -e switches the limit calculation to expected limits
- `--signif` switches to calculation of p-Values using the ProfileLikelihoodCalculator

- `--frequentist` uses frequentist calculations for limits or p-Values
- `--hybrid` uses Frequentist-Bayesian Hybrid method for the p-Values

Apart from these fundamental options, there are several further modifications that can be made

3.2.1 Binned limits

The `--binned` option triggers the use of binned instead of unbinned datasets. For this purpose, binned templates are generated from the background and signal PDFs. The binning is hardcoded within the `createInputs.py` script. The advantage of this approach is a large improvement in speed, the disadvantage is a very long time needed to generate the templates in the first place.

3.2.2 Single mass points

To run a single mass point instead of the full mass scan, the option `-m mass` can be used.

3.2.3 Signal injection and Look Elsewhere Effect

For performance studies, pseudo-data can be generated in which the statistical interpretation is then performed. When run with the `--inject` option, pseudo background and signal events are generated according to the respective PDFs. The background is normalized to the yield observed in data in each channel. The signal parameters used for the injection are taken from the scan configuration. The signal events are distributed between the sub-channels according to the signal efficiencies.

To account for the look elsewhere effect, the `--LEE` option can be used. Many background only datasets will be generated and p-Value scans will be performed. The tool `readPValueToys.py` can be used to harvest the large number of resulting result cards.

3.2.4 Job submission

As the calculations used for the statistical interpretations, parallelization is unavoidable. The framework supports two options for it, submission to local batch systems and CRAB. The `-s` option triggers submission to batch system. At the moment, only the Purdue system is supported. However, the job configurations can be easily used for any qsub system and should be adaptable to others system as well.

Less specific and giving access to much more computing resources is submission via crab. At the moment, only expected and observed Bayesian limits are supported. On the upside, submission is very easy, just run the tool with the `--crab` option. A valid GRID certificate is required.

3.3 Output processing

The output of the combine tool are root files which contain the resulting limit or p-Value as entries in a ROOT tree. The script `createLimitCard.py` is available to convert these files into simple ascii files. This tool takes a variety of arguments, very similar to the main `runInterpretation.py` script:

```
optional arguments:
  -h, --help            show this help message and exit
  -c CONFIG, --config CONFIG
                        configuration name (default: )
  --input INPUT         folder with input root files (default: )
  -t TAG, --tag TAG     tag (default: )
  --exp                 write expected limits (default: False)
  --signif              write pValues (default: False)
  --injected            injected (default: False)
  --binned              binned (default: False)
  --frequentist         use results from frequentist limits (default: False)
  --hybrid              use results from hybrid significance calculations
                        (default: False)
```