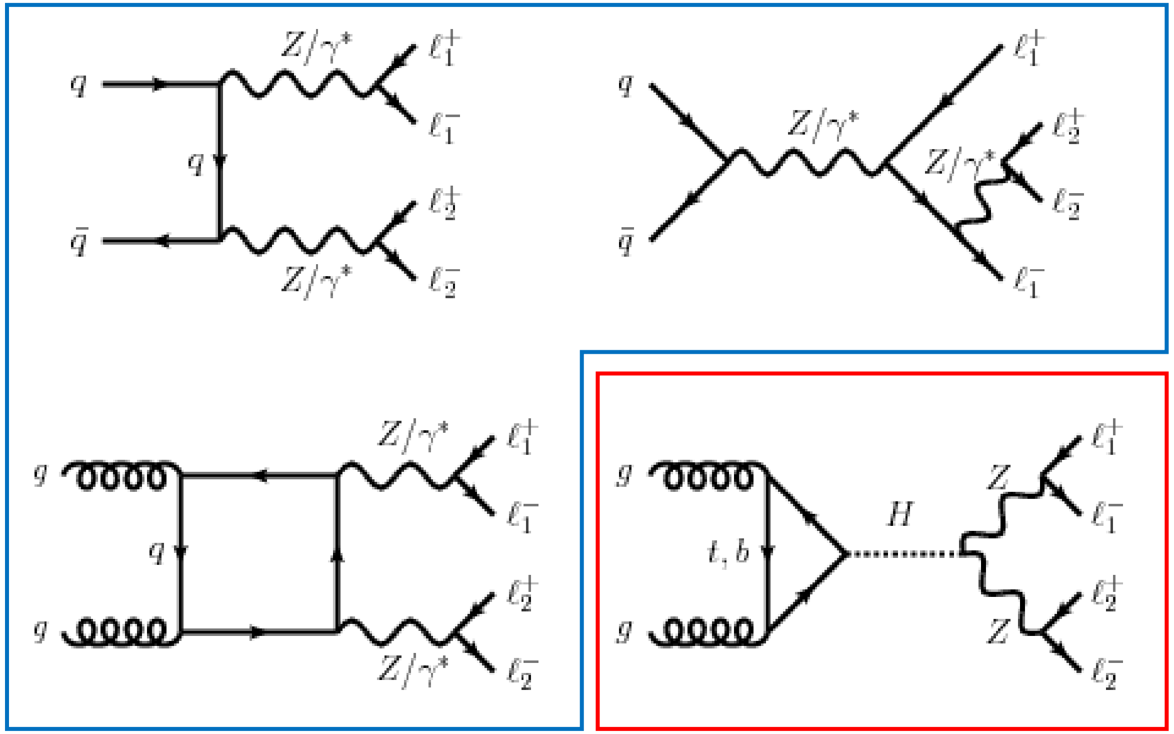
Analysis of Higgs boson decay into four leptons at 8 TeV

Matteo Malucchi

October 2022

$H \rightarrow ZZ^* \rightarrow l_1^+ l_1^- l_2^+ l_2^-$

The datasets used throughout the analysis were reduced NanoAOD files created from CMS Open Data using data from Run 1 for a total of 11.6 fb^{-1} at 8 TeV.



SM ZZ background

SM Higgs signal

Variable	Type	Description
run	*int*	Run number
luminosityBlock	*unsigned int*	Luminosity block number
event	*unsigned long*	Event number
PV_npvs	*int*	Number of primary vertices
PV_x	*float*	Position of the primary vertex in x direction
PV_y	*float*	Position of the primary vertex in y direction
PV_z	*float*	Position of the primary vertex in z direction
nMuon	*unsigned int*	Number of muons in this event
Muon_pt	*float[nMuon]*	Transverse momentum of the muons
Muon_eta	*float[nMuon]*	Pseudorapidity of the muons
Muon_phi	*float[nMuon]*	Azimuth angle of the muons
Muon_mass	*float[nMuon]*	Mass of the muons
Muon_charge	*int[nMuon]*	Charge of the muons (either 1 or -1)
Muon_dxy	*float[nMuon]*	Distance of the muons to the primary vertex in the transverse plane
Muon_dxyErr	*float[nMuon]*	Uncertainty on the distance of the muons to the primary vertex in the transverse plane
Muon_dz	*float[nMuon]*	Distance of the muons to the primary vertex in parallel to the beam pipe
Muon_dzErr	*float[nMuon]*	Uncertainty on the distance of the muons to the primary vertex in parallel to the beam pipe
Muon_pfRelIso03_all	*float[nMuon]*	Muon isolation divided by the transverse momentum in the $R = 0.3$ cone
Muon_pfRelIso04_all	*float[nMuon]*	Muon isolation divided by the transverse momentum in the $R = 0.4$ cone
nElectron	*unsigned int*	Number of electrons in this event
Electron_pt	*float[nElectron]*	Transverse momentum of the electrons
Electron_eta	*float[nElectron]*	Pseudorapidity of the electrons
Electron_phi	*float[nElectron]*	Azimuth angle of the electrons
Electron_mass	*float[nElectron]*	Mass of the electrons
Electron_charge	*int[nElectron]*	Charge of the electrons (either 1 or -1)
Electron_dxy	*float[nElectron]*	Distance of the electrons to the primary vertex in the transverse plane
Electron_dxyErr	*float[nElectron]*	Uncertainty on the distance of the electrons to the primary vertex in the transverse plane
Electron_dz	*float[nElectron]*	Distance of the electrons to the primary vertex in parallel to the beam pipe
Electron_dzErr	*float[nElectron]*	Uncertainty on the distance of the electrons to the primary vertex in parallel to the beam pipe
Electron_pfRelIso03_all	*float[nElectron]*	Electron isolation divided by the transverse momentum in the $R = 0.3$ cone
MET_pt	*float*	Missing transverse energy
MET_phi	*float*	Azimuth angle of the missing transverse energy

Set up and run the analysis

run_analysis.py

```python
if args_global.download != "":
    download_dataset.download(args_global, logger_global)

if args_global.skim:
    skim.skim(args_global, logger_global)

if args_global.ml:
    ml_training.ml_training(args_global, logger_global)
    ml_evaluation.ml_evaluation(args_global, logger_global)
    ml_selection.ml_selection(args_global, logger_global)
    ml_histo.ml_histo(args_global, logger_global)
    ml_plot.ml_plot(args_global, logger_global)
```

```python
if args_global.graphPlots:
    make_histo.make_histo(args_global, logger_global)
    make_plot.make_plot(args_global, logger_global)

if args_global.invariantMassFit:
    fit_mass.fit_mass(args_global, logger_global)
```

```
Analysis Tool

options:
  -h, --help            show this help message and exit
  -l LOGLEVEL, --logLevel LOGLEVEL
                        integer representing the level of the logger: DEBUG=10, INFO = 20, WARNING = 30, ERROR = 40
  -d [DOWNLOAD], --download [DOWNLOAD]
                        enables the download of input data. If not specified otherwise the files are saved in the 'Input/' directory
  -b [BASEPATH], --basePath [BASEPATH]
                        base path where to find the input data. If enabled it automatically gets the input data from EOS unless a local directory is specified
  -o OUTPUT, --output OUTPUT
                        name of the output directory
  -c [CLEAROUTPUT], --clearOutput [CLEAROUTPUT]
                        name of output folder to be deleted. If not specified otherwise the 'Output/' directory is deleted
  -q, --skim            disables the skimming step
  -m, --ml              disables machine learning algorithm
  -g, --graphPlots      disables the graphing of the distribution plots
  -i, --invariantMassFit
                        disables fit of the Higgs mass
  -s SAMPLE, --sample SAMPLE
                        string with comma separated list of samples to analyse: Run2012B_DoubleElectron, Run2012B_DoubleMuParked, Run2012C_DoubleElectron,
                        Run2012C_DoubleMuParked, SMHiggsToZZTo4L, ZZTo2e2mu, ZZTo4e, ZZTo4mu
  -f FINALSTATE, --finalState FINALSTATE
                        comma separated list of the final states to analyse: FourMuons,FourElectrons,TwoMuonsTwoElectrons
  -e, --typeOfParallel  parallel type for the downloads: default is multi-thread, if activated is multi-process
  -p, --parallel        disables running in parallel
  -n NWORKERS, --nWorkers NWORKERS
                        number of workers for multi-threading
  -r [RANGE], --range [RANGE]
                        number of events on which the analysis is ran over (does not work in parallel)
  -a MLVARIABLES, --MLVariables MLVARIABLES
                        name of the set of variables to be used in the ML algorithm defined 'Analysis/Definitions/variables_ml_def.py': tot, angles, higgs
  -v VARIABLEDISTRIBUTION, --variableDistribution VARIABLEDISTRIBUTION
                        string with comma separated list of the variables to plot. The complete list is defined in 'Analysis/Definitions/variables_def.py'
  -t TYPEDISTRIBUTION, --typeDistribution TYPEDISTRIBUTION
                        comma separated list of the type of distributions to plot: data, background, signal, sig_bkg_normalized, total
```

set_up.py

```python
# Check if typeDistribution is valid
try:
    args.typeDistribution = check_val(logger, args.typeDistribution,
        ["all", "data", "background", "signal", "sig_bkg_normalized", "total"], "typeDistribution")
except AttributeError:
    pass
```

```python
def check_val(log, input, correct_list, name):

    try:
        if not any(correct_in in input.split(",") for correct_in in correct_list):
            list_str=', '.join(correct_list)
            raise argparse.ArgumentTypeError(f"{name} {input} is invalid: it must be either {list_str}")
    except argparse.ArgumentTypeError as arg_err:
        log.exception("%s \n>>>%s is set to %s \n", arg_err, name, correct_list[0], stack_info=True)
        return correct_list[0]
    else:
        return input
```

```python
# Create the directory to save the outputs if doesn't already exist
try:
    create_dir(logger, args.output, False)
except AttributeError:
    pass
```

```python
def create_dir(log, dir, ignore):

    try:
        os.makedirs(dir)
        log.debug("Directory %s/ Created", dir)
    except FileExistsError:
        log.debug("The directory %s/ already exists", dir)
    finally:
        if ignore == True:
            file_path=os.path.join(dir, ".gitignore")
            if os.path.exists(file_path):
                pass
            else:
                # create .gitignore
                with open(file_path, 'w') as fp:
                    fp.write("# Ignore everything in this directory \n*")
                    fp.write("\n# Except this file \n!.gitignore")
```

Download the datasets

download_dataset.py

```python
def download(args, logger):

    logger.info(">>> Executing %s \n", os.path.basename(__file__))
    t= perf_counter()

    if args.parallel:
        logger.info(">>> Executing in parallel \n")
        parallel_list = []
        #Loop over the various samples
        for sample_name, number in SAMPLES_DOWNLOAD.items():

            # Check if the sample is one of those requested by the user
            if sample_name not in args.sample and args.sample != "all":
                continue

            logger.info(">>> Process sample: %s \n", sample_name)
            file_name=os.path.join(args.download, f"{sample_name}.root")

            if args.typeOfParallel == "thread":
                logger.info(">>> Executing multi-threading \n")
                parallel_list.append(thr.Thread(target=get_file_parallel, args=(logger, number, sample_name, file_name)))
            elif args.typeOfParallel == "process":
                logger.info(">>> Executing multi-processing \n")
                parallel_list.append(mp.Process(target=get_file_parallel, args=(logger, number, sample_name, file_name)))

        #start parallel
        for parallel_elem in parallel_list:
            parallel_elem.start()
        #join parallel
        for parallel_elem in parallel_list:
            parallel_elem.join()

    else:
        logger.info(">>> Executing not in parallel \n")
        #Loop over the various samples
        for sample_name, number in SAMPLES_DOWNLOAD.items():

            # Check if the sample is one of those requested by the user
            if sample_name not in args.sample and args.sample != "all":
                continue

            logger.info(">>> Process sample: %s \n", sample_name)
            file_name=os.path.join(args.download, f"{sample_name}.root")
            get_file(logger, number, sample_name, file_name)
```

```python
class MyProgressBar():

    def __init__(self):

        self.pbar = None

    def __call__(self, block_num, block_size, total_size):

        if not self.pbar:
            self.pbar=progressbar.ProgressBar(maxval=total_size)
            self.pbar.start()

        downloaded = block_num * block_size
        if downloaded < total_size:
            self.pbar.update(downloaded)
        else:
            self.pbar.finish()


def count(func):

    @wraps(func)
    def counted(*args):
        counted.call_count += 1
        return func(*args)
    counted.call_count = 0
    return counted
```

```python
@count_func
def get_file(log, num, sample, file):

    try:
        urllib.request.urlretrieve(
            f"http://opendata.cern.ch/record/{num}/files/{sample}.root", file, MyProgressBar())
    except urllib.error.HTTPError as http_err:
        log.exception("File %s.root can't be found %s",
                        sample, http_err, stack_info=True)
    except urllib.error.ContentTooShortError:
        log.error("Network conditions is not good. Reloading for %d time file %s.root",
                        get_file.call_count, sample)
        if get_file.call_count == 7:
            log.exception(
                "ERROR: Download of %s.root has failed due to bad network conditions!\n", sample)
            get_file.call_count=0
            return
        get_file(log, num, sample, file)
    else:
        get_file.call_count=0
```

```python
def get_file_parallel(log, num, sample, file):

    count = 1
    while count <= 6:
        try:
            urllib.request.urlretrieve(
                f"http://opendata.cern.ch/record/{num}/files/{sample}.root", file)
        except urllib.error.HTTPError as http_err:
            log.exception("File %s.root can't be found %s",
                            sample, http_err, stack_info=True)
            return
        except urllib.error.ContentTooShortError:
            log.error(
                "Network conditions is not good. Reloading for %d time file %s.root",
                count, sample)
            count += 1
        else:
            return

    log.exception("ERROR: Download of %s.root has failed due to bad network conditions!\n", sample)
```

Skim the dataset

skim.py

```python
#Enamble multi-threading if range is not active
if args.parallel and args.range == 0:
    ROOT.ROOT.EnableImplicitMT(args.nWorkers)
    thread_size = ROOT.ROOT.GetThreadPoolSize()
    logger.info(">>> Thread pool size for parallel processing: %s", thread_size)
```

```python
try:
    rdf2 = skim_tools.event_selection(rdf, final_state)
    rdf3 = skim_tools.four_vec(rdf2, final_state)
    rdf4 = skim_tools.order_four_vec(rdf3, final_state)
except RuntimeError as run_time_err:
    logger.exception("Sample %s ERROR: %s ", sample_name, run_time_err,  stack_info=True)
    continue

rdf5 = skim_tools.def_mass_pt_eta_phi(rdf4)
rdf6 = skim_tools.four_vec_boost(rdf5)
rdf7 = skim_tools.def_angles(rdf6)
rdf_final = skim_tools.add_event_weight(rdf7, WEIGHTS[sample_name])
```

skim_functions.h

```cpp
RVec<float> sipDef(VecF dxy, VecF dz, VecF sigma_dxy, VecF sigma_dz){
    auto ip=sqrt(dxy*dxy + dz*dz);
    auto sigma_ip=sqrt((sigma_dxy)*(sigma_dxy) + (sigma_dz)*(sigma_dz));
    auto sip=(ip/sigma_ip);
    return sip;
};


bool ptCuts(VecF mu_pt, VecF el_pt){
    if (Max(mu_pt)>20 && Min(mu_pt)>10) return true;
    if (Max(el_pt)>20 && Min(el_pt)>10) return true;
    return false;
};
```

skim_tools.py

```python
def event_selection(rdf, final_state):

    if final_state == "FourMuons":
        return rdf.Filter("nMuon==4",
                          "Four muons")\
                  .Filter("Sum(Muon_charge==1)==2 && Sum(Muon_charge==-1)==2",
                          "Two positive and two negative muons")\
                  .Filter("All(abs(Muon_pfRelIso04_all)<0.40)",
                          "Good isolation of the muons")\
                  .Filter("All(Muon_pt>5) && All(abs(Muon_eta)<2.4)",
                          "Good muon kinematics")\
                  .Define("Muon_3d_sip",
                          "sipDef(Muon_dxy, Muon_dz, Muon_dxyErr, Muon_dzErr)")\
                  .Filter("All(Muon_3d_sip<4) && All(abs(Muon_dxy)<0.5) && All(abs(Muon_dz)<1.0)",
                          "Muons originate from the same primary vertex")

    if final_state == "FourElectrons":
        return rdf.Filter("nElectron==4",
                          "Four electrons")\
                  .Filter("Sum(Electron_charge==1)==2 && Sum(Electron_charge==-1)==2",
                          "Two positive and two negative electrons")\
                  .Filter("All(abs(Electron_pfRelIso03_all)<0.40)",
                          "Good isolation of the electrons")\
                  .Filter("All(Electron_pt>7) && All(abs(Electron_eta)<2.5)",
                          "Good electron kinematics")\
                  .Define("Electron_3d_sip",
                          "sipDef(Electron_dxy, Electron_dz, Electron_dxyErr, Electron_dzErr)")\
                  .Filter("All(Electron_3d_sip<4) && All(abs(Electron_dxy)<0.5) \
                          && All(abs(Electron_dz)<1.0)",
                          "Electrons originate from the same primary vertex")
```

```python
if final_state == "TwoMuonsTwoElectrons":
    return rdf.Filter("nMuon==2 && nElectron==2",
                      "Two muons and two electrons")\
              .Filter("Sum(Electron_charge)==0 && Sum(Muon_charge)==0",
                      "Two opposite charged electron and muon pairs")\
              .Filter("All(abs(Electron_eta)<2.5) && All(abs(Muon_eta)<2.4)",
                      "Eta cuts")\
              .Filter("All(abs(Muon_pfRelIso04_all)<0.40) && \
                      All(abs(Electron_pfRelIso03_all)<0.40)",
                      "Require good isolation")\
              .Filter("ptCuts(Muon_pt, Electron_pt)", "Pt cuts")\
              .Define("Muon_dr",
                      "ROOT::VecOps::DeltaR(Muon_eta[0], Muon_eta[1], \
                       Muon_phi[0], Muon_phi[1])")\
              .Define("Electron_dr",
                      "ROOT::VecOps::DeltaR(Electron_eta[0], Electron_eta[1], \
                      Electron_phi[0], Electron_phi[1])")\
              .Filter("Muon_dr>0.02 && Electron_dr>0.02",
                      "Delta R cuts")\
              .Define("Muon_3d_sip",
                      "sipDef(Muon_dxy, Muon_dz, Muon_dxyErr, Muon_dzErr)")\
              .Filter("All(Muon_3d_sip<4) && All(abs(Muon_dxy)<0.5) && All(abs(Muon_dz)<1.0)",
                      "Muons originate from the same primary vertex")\
              .Define("Electron_3d_sip",
                      "sipDef(Electron_dxy, Electron_dz, Electron_dxyErr, Electron_dzErr)")\
              .Filter("All(Electron_3d_sip<4) && All(abs(Electron_dxy)<0.5) \
                      && All(abs(Electron_dz)<1.0)",
                      "Electrons originate from the same primary vertex")\
```

test_skim.py

class `Test.test_skim.TestSkim`(**args, **kwargs*) [source]

`test_boost()` [source]

Test the boost of the fourvectors in a given frame.

`test_cos_theta()` [source]

Test the definition of cos(theta_star), cos(theta1) and cos(theta2).

`test_cross()` [source]

Test the normalized cross product between two vectors.

`test_deltar()` [source]

Test the angular separation of particles building the Z systems.

`test_lep1()` [source]

Test the selection of the lepton/anti-lepton belonging to the heaviest boson Z1 in case of leptons of different kinds.

`test_lep2()` [source]

Test the selection of the lepton/anti-lepton belonging to the lighter boson Z2 in case of leptons of different kinds.

`test_lep_fourvec()` [source]

Test the reconstruction of the lepton fourvectors.

`test_order_idx_z()` [source]

Test the order of the Z fourvectors so that the first Z is the heaviest one.

`test_phi()` [source]

Test the definition of Phi and Phi1.

`test_pt_cuts()` [source]

Test the lepton pt cuts that require that in at least one of the lepton couples the highest energy particle has Pt > 20 GeV while the other one Pt > 10 GeV.

`test_sip()` [source]

Test the definition of the significance of the impact parameter sip as the ratio between the impact parameter at the point of closest approach to the vertex and its uncertainty.

`test_split_lep_samekind()` [source]

Test the order of the leptons in the case of 4 leptons of the same kind.

`test_theta()` [source]

Test the definition of theta_star, theta1 and theta2.

`test_z_fourvec_2mu2el()` [source]

Test the reconstruction of the two Z fourvectors in the case of leptons of different kind and their ascending distance to Z mass organization.

`test_z_fourvec_samekind()` [source]

Test the reconstruction of the two Z fourvectors in the case of leptons of the same kind and their ascending distance to Z mass organization.

`test_z_heavy()` [source]

Test the selection of the heavier Z.

`test_z_idx_samekind()` [source]

Test the reconstruction of the same kind lepton pair whose invariant mass is closest to the Z mass.

`test_z_light()` [source]

Test the selection of the lighter Z.

Define the decay angles

Let us consider, of all the possible combinations of same-flavor opposite-sign lepton pairs, the one that has mass closest to $m_Z = 91.2$ GeV. The *on-shell* Z boson associated with it is referred to as Z_{close}. The other *off-shell* Z boson is reconstructed using the remaining lepton pair and is called Z_{far}. Subsequently, the reconstructed Z boson with highest mass is called Z_1 and the another one Z_2. The three-momentum of the Z_i boson is called $\mathbf{q}_i$, while $\mathbf{q}_{i1}$ and $\mathbf{q}_{i2}$ indicate respectively the three-momenta of the lepton and anti-lepton associated with Z_i. Indicating as superscript the rest frame in which the three-momenta are taken, the definitions of the angles are:

- $\theta^* \in [0, \pi]$ is the production angle of the leading Z in the four-lepton rest frame

$$\cos\theta^* = \frac{q_{1z}^{4l}}{|\mathbf{q}_1^{4l}|}$$

where q_{1z}^{4l} is the z component of the three-momentum of Z_1 in the four-leptons rest frame

- $\Phi_1 \in [-\pi, \pi]$ is the angle between the decay plane of the leading lepton pair and a plane defined by Z_1 in the four-lepton rest frame and the positive direction of the collision axis $\hat{n}_z = (0, 0, 1)$

$$\Phi_1 = \frac{\mathbf{q}_1^{4l} \cdot (\hat{n}_1 \times \hat{n}_{coll})}{|\mathbf{q}_1^{4l} \cdot (\hat{n}_1 \times \hat{n}_{coll})|} \times \arccos(\hat{n}_1 \cdot \hat{n}_{coll})$$

where the normal vectors to the two planes are defined as

$$\hat{n}_1 = \frac{\mathbf{q}_{11}^{4l} \times \mathbf{q}_{12}^{4l}}{|\mathbf{q}_{11}^{4l} \times \mathbf{q}_{12}^{4l}|}, \qquad \hat{n}_{coll} = \frac{\hat{n}_z \times \mathbf{q}_1^{4l}}{|\hat{n}_z \times \mathbf{q}_1^{4l}|}$$

- $\Phi \in [-\pi, \pi]$ is the angle between the decay planes of the two lepton pairs in the four-leptons rest frame

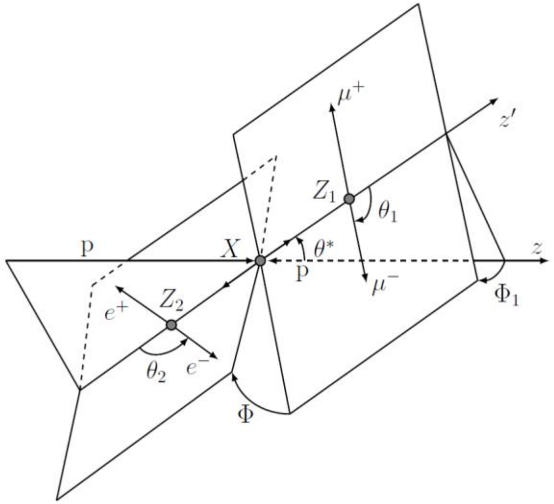
$$\Phi = \frac{\mathbf{q}_1^{4l} \cdot (\hat{n}_1 \times \hat{n}_2)}{|\mathbf{q}_1^{4l} \cdot (\hat{n}_1 \times \hat{n}_2)|} \times \arccos(-\hat{n}_1 \cdot \hat{n}_2)$$

where

$$\hat{n}_2 = \frac{\mathbf{q}_{21}^{4l} \times \mathbf{q}_{22}^{4l}}{|\mathbf{q}_{21}^{4l} \times \mathbf{q}_{22}^{4l}|}$$

- $\theta_1 \in [0, \pi]$ and $\theta_2 \in [0, \pi]$ are the angles between final-state negatively charged leptons and the direction of flight of their respective Z bosons

$$\cos\theta_1 = -\frac{\mathbf{q}_2^{Z_1} \cdot \mathbf{q}_{11}^{Z_1}}{|\mathbf{q}_2^{Z_1}||\mathbf{q}_{11}^{Z_1}|}, \qquad \cos\theta_2 = -\frac{\mathbf{q}_1^{Z_2} \cdot \mathbf{q}_{21}^{Z_2}}{|\mathbf{q}_1^{Z_2}||\mathbf{q}_{21}^{Z_2}|}$$



skim_tools.py

`Analysis.Skimming.skim_tools.four_vec(rdf, final_state)` [source]

Reconstruct fourvector for leptons, Z and Higgs candidates.

- **Parameters:**
  - **rdf** (*ROOT.RDataFrame*) – Input RDataFrame
  - **final_state** (*str*) – Final state to be analysed
- **Raises:** **RuntimeError** – Raised when an unknown final state is passed
- **Returns:** Output RDataFrame
- **Return type:** ROOT.RDataFrame

`Analysis.Skimming.skim_tools.order_four_vec(rdf, final_state)` [source]

Put the four vectors in order according to the following criteria:

- Z1 = heavier boson candidate
- Z2 = lighter boson candidate
- Lep11 = lepton belonging to the heavier boson Z1
- Lep12 = anti-lepton belonging to the heavier boson Z1
- Lep21 = lepton belonging to the lighter boson Z2
- Lep22 = anti-lepton belonging to the lighter boson Z2

- **Parameters:**
  - **rdf** (*ROOT.RDataFrame*) – Input RDataFrame
  - **final_state** (*str*) – Final state to be analysed
- **Raises:** **RuntimeError** – Raised when an unknown final state is passed
- **Returns:** Output RDataFrame
- **Return type:** ROOT.RDataFrame

`Analysis.Skimming.skim_tools.def_mass_pt_eta_phi(rdf)` [source]

Define Mass, Pt, Eta and Phi of Higgs boson and Z candidates.

- **Parameters:** **rdf** (*ROOT.RDataFrame*) – Input RDataFrame
- **Returns:** Output RDataFrame
- **Return type:** ROOT.RDataFrame

`Analysis.Skimming.skim_tools.four_vec_boost(rdf)` [source]

Boost the various fourvectors in different frames.

- **Parameters:** **rdf** (*ROOT.RDataFrame*) – Input RDataFrame
- **Returns:** Output RDataFrame
- **Return type:** ROOT.RDataFrame

`Analysis.Skimming.skim_tools.def_angles(rdf)` [source]

Define the five angles that allow discrimination between signal and background.

- **Parameters:** **rdf** (*ROOT.RDataFrame*) – Input RDataFrame
- **Returns:** Output RDataFrame
- **Return type:** ROOT.RDataFrame

`Analysis.Skimming.skim_tools.add_event_weight(rdf, weight)` [source]

Add weights for the normalisation of the simulated samples in the histograms.

- **Parameters:** **rdf** (*ROOT.RDataFrame*) – Input RDataFrame
- **Returns:** Output RDataFrame
- **Return type:** ROOT.RDataFrame

DNN variables and model

ml_training.py

```
# Setup TMVA
ROOT.TMVA.Tools.Instance()
ROOT.TMVA.PyMethodBase.PyInitialize()
```

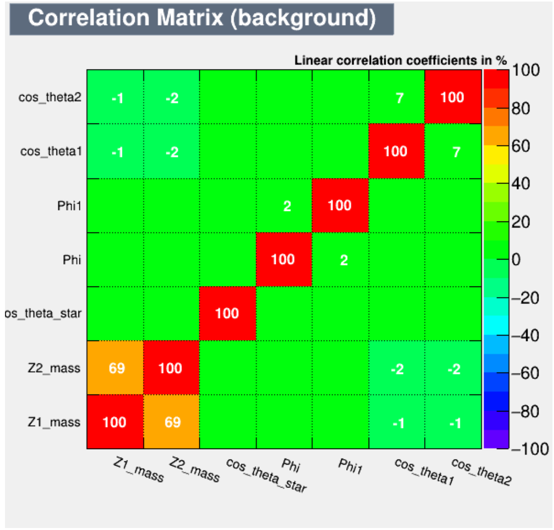
```
factory = ROOT.TMVA.Factory("TMVAClassification", output,
                            "!V:!Silent:Color:DrawProgressBar:Transformations=D,G:AnalysisType=Classification")
```

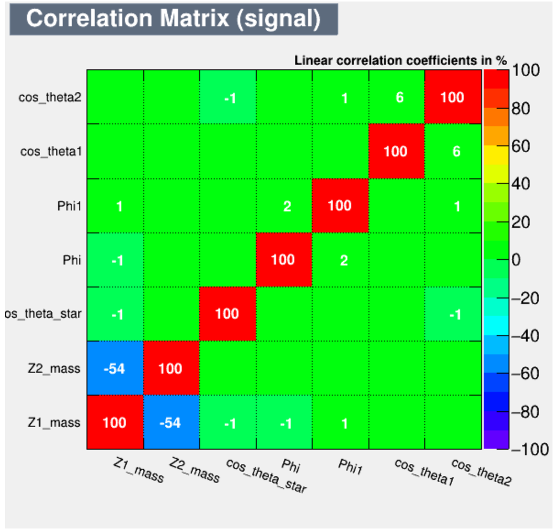
```
# Directory where the weights are saved
dataloader = ROOT.TMVA.DataLoader("dataset")
for variable in variables:
    dataloader.AddVariable(variable)
    logger.debug(variable)
```

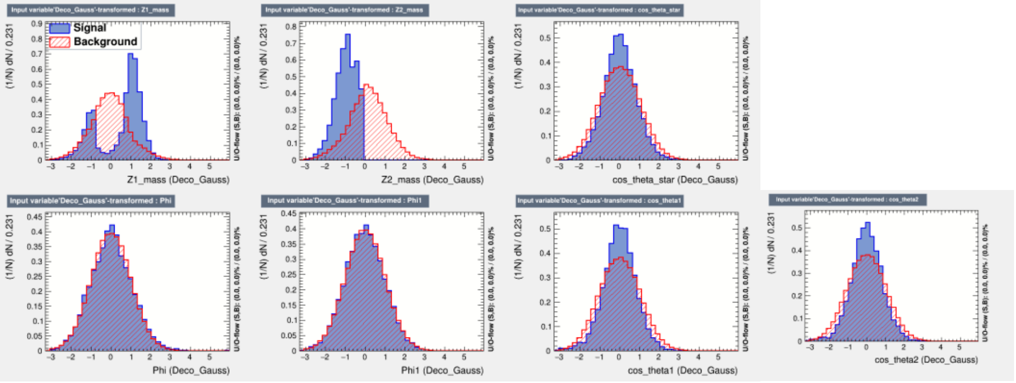
```
dataloader.AddSignalTree(signal_chain, 1.0)
dataloader.AddBackgroundTree(bkg_chain, 1.0)
```

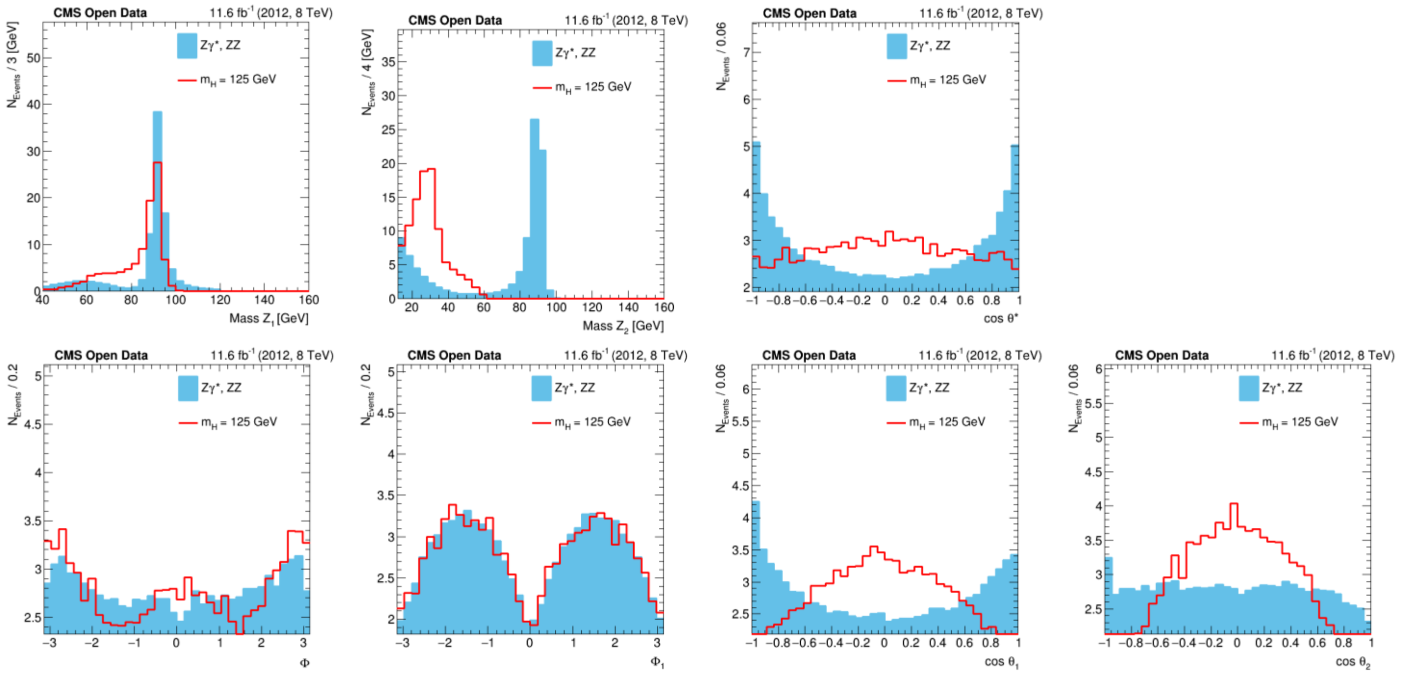
```
dataloader.PrepareTrainingAndTestTree(ROOT.TCut(""),"SplitMode=Random:NormMode=NumEvents:!V")
```

```
# Define model
model = Sequential()
model.add(Dense(12, activation="relu", input_dim=len(variables)))
model.add(Dense(12, activation="relu"))
model.add(Dense(12, activation="relu"))
model.add(Dense(2, activation="softmax"))


# Set loss and optimizer
model.compile(loss="categorical_crossentropy",
              optimizer="adam", metrics=["accuracy", ], weighted_metrics=[])

# Store model to file
model_path = os.path.join(dir_name,"DNNmodel.h5")
model.save(model_path)
model.summary()
```





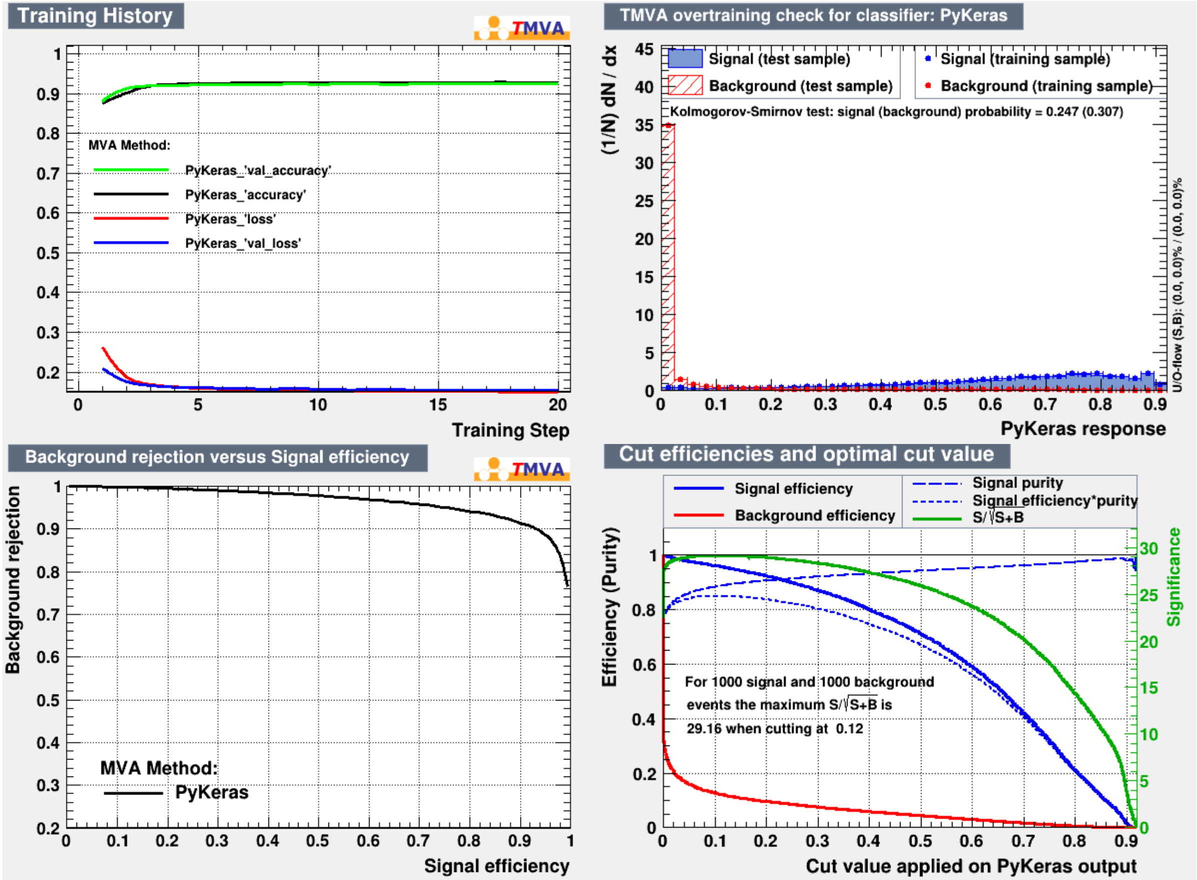




DNN training

ml_training.py

```python
# Book methods
method = factory.BookMethod(dataloader, ROOT.TMVA.Types.kPyKeras, "PyKeras",
                   f"H:!V:VarTransform=D,G:FilenameModel={model_path}:NumEpochs=20:BatchSize=128")

# Run training, test and evaluation
factory.TrainAllMethods()
factory.TestAllMethods()
factory.EvaluateAllMethods()
```



ml_evaluation.py

```python
discr_array = array("f", [-999])
branch = new_tree.Branch("Discriminant", discr_array, "Discriminant/F")
```

```python
n_entries = tree.GetEntries()
for i in range(n_entries):
    new_tree.GetEntry(i)

    if args.MLVariables == "tot":
        discr_array[0] = reader.EvaluateMVA([new_tree.Z1_mass,
                                             new_tree.Z2_mass, new_tree.cos_theta_star,
                                             new_tree.Phi, new_tree.Phi1, new_tree.cos_theta1,
                                             new_tree.cos_theta2], "PyKeras")
```

ml_selection.py

```python
rdf_final = rdf.Filter(f"Discriminant>{cut[0]}",
                       "Select only events with above threshold discriminat")
```

Histograms e plots

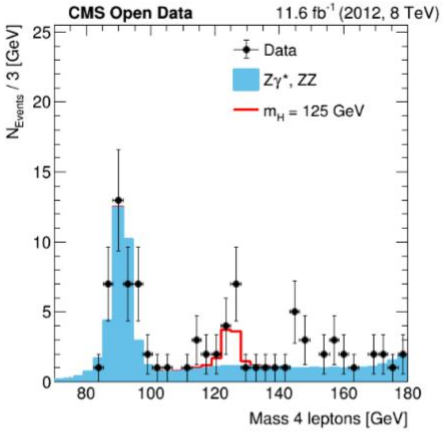
make_histo.py

```
histos[variable] = histogramming_functions.book_histogram_1d\
                        (rdf, variable, var_dict[variable])
histogramming_functions.write_histogram(histos[variable],
                        f"{sample_name}_{final_state}_{variable}_{selection}")
```

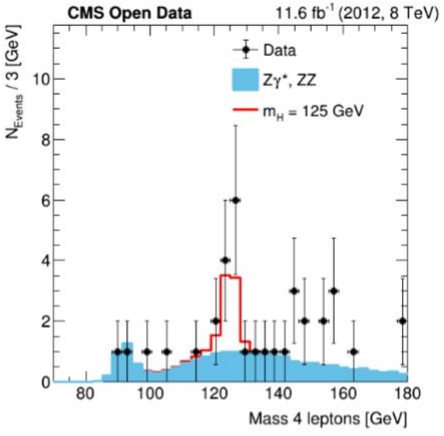
histogramming_functions.py

```
def write_histogram(histo, name):

    histo.SetName(name)
    histo.Write()
```

```
def book_histogram_1d(rdf, variable, range_):

    return rdf.Histo1D(ROOT.ROOT.RDF.TH1DModel(variable, variable,\
                        range_[0], range_[1], range_[2]),\
                        variable, "Weight")
```



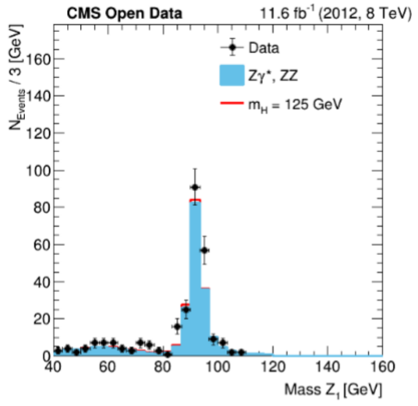
No selection



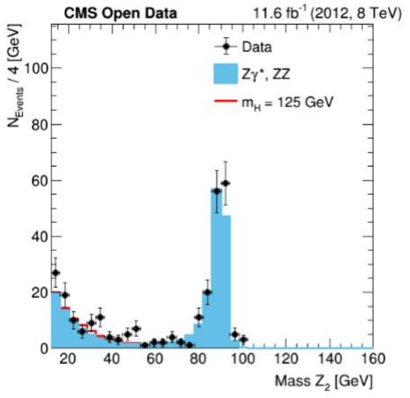
DNN selection

plotting_functions.py

```
def combine_final_states(dict_comb):

    dict_comb["Combined"] = dict_comb["FourMuons"].Clone()
    dict_comb["Combined"].Add(dict_comb["FourElectrons"])
    dict_comb["Combined"].Add(dict_comb["TwoMuonsTwoElectrons"])
```



No selection



No selection

make_plot.py

```
# Get histograms for the signal
signals = {}

# Check if the sample to plot is one of those requested by the user
if "SMHiggsToZZTo4L" in args.sample or args.sample == "all":
    for final_state in ["FourMuons", "FourElectrons", "TwoMuonsTwoElectrons"]:
        histo_name = f"SMHiggsToZZTo4L_{final_state}_{variable}_{selection}"
        try:
            signals[final_state] = plotting_functions.get_histogram(infile, histo_name)
        except RuntimeError as run_time_err:
            logger.debug("ERROR:  %s ", run_time_err,  stack_info=True)

    try:
        plotting_functions.combine_final_states(signals)
    except KeyError:
        logger.debug("ERROR: Failed to create the signal histogram of the combined final states",
                        stack_info=True)

# Get the normalized histograms for the signal
signals_norm = {}
for final_state, signal_histo in signals.items():
    histo = signal_histo.Clone()
    histo.Scale(1/histo.Integral()*100)
    signals_norm[final_state] = histo
```

```
elif input_type == "total":

    # Add the background to the signal
    # in order to compare it with the data.
    signals[final_state].Add(backgrounds[final_state])
    for input_key in inputs:
        input_histo=inputs_dict[input_key][final_state]
        plotting_functions.input_style(input_key, input_histo)
        legend=plotting_functions.add_legend(legend, input_key, input_histo)

    plotting_functions.add_title(signals[final_state], var_dict[variable])
    signals[final_state].SetMaximum(
                        max(backgrounds[final_state].GetMaximum(),
                        data[final_state].GetMaximum()) * 1.4)
    signals[final_state].Draw("HIST")
    backgrounds[final_state].Draw("HIST SAME")
    data[final_state].Draw("E1P SAME")
    legend.Draw()
```

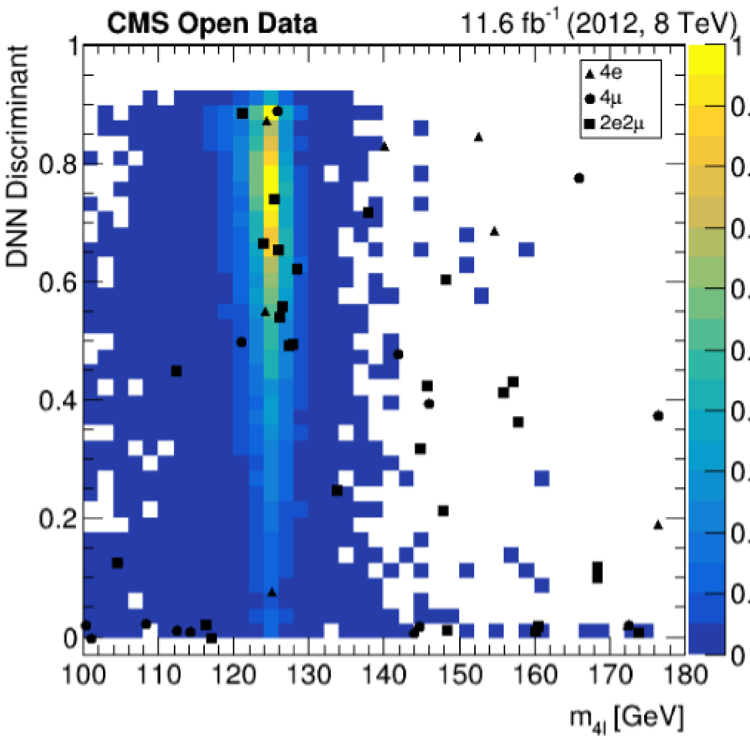
Higgs mass VS DNN Discriminant

ml_histo.py

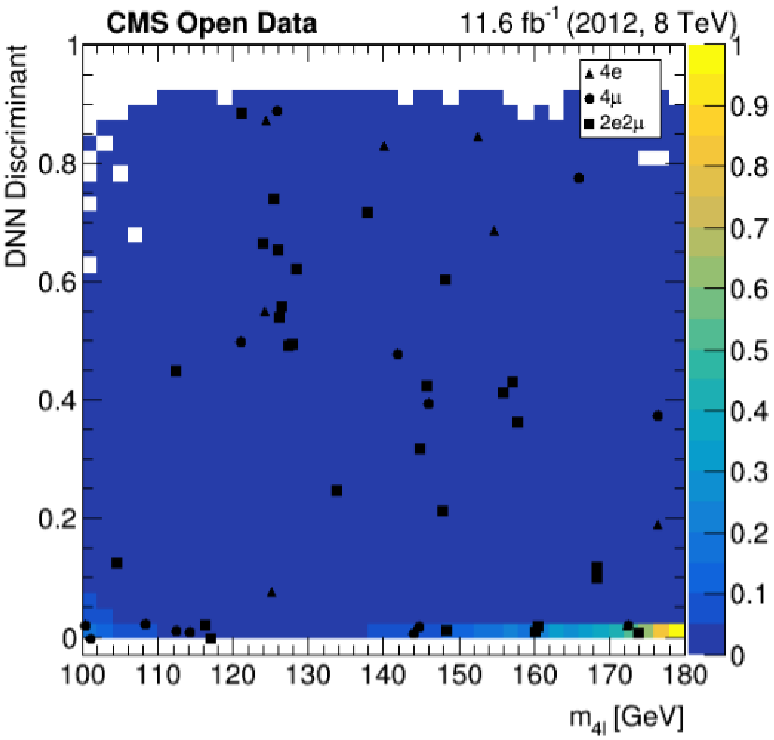
```
rdfs = {
    "signal" : sig_rdf,
    "background" : bkg_rdf,
    "data_el" : data_el_rdf,
    "data_mu" : data_mu_rdf,
    "data_elmu" :  data_elmu_rdf
}

histos = {}
variables = ["Higgs_mass", "Discriminant"]
ranges_x = [40, 100., 180.]
ranges_y = [40, -0.03, 1]
for dataset, rdf in rdfs.items():
    logger.info(">>> Process sample: %s", dataset)
    try:
        histos[dataset] = histogramming_functions.book_histogram_2d(dataset,
                                                rdf, variables, ranges_x, ranges_y)
        histogramming_functions.write_histogram(histos[dataset], dataset)
    except TypeError:
        logger.debug("Dataset %s is empty", dataset)
```

ml_plot.py

```
for type_dataset in ["signal", "background"]:

    canvas = ROOT.TCanvas("", "", 600, 600)
    legend = ROOT.TLegend(0.75, 0.8, 0.85, 0.9)
    try:
        plotting_functions.add_title(histos[type_dataset])
        histos[type_dataset].Scale(1/histos[type_dataset].GetMaximum())
        histos[type_dataset].Draw("COLZ")
    except KeyError:
        logger.debug("ERROR: Failed to create the %s histogram",
                            type_dataset, stack_info=True)
        continue

    try:
        histos["data_el"].Draw("SAME P")
    except KeyError:
        logger.debug("ERROR: Failed to create the data histogram of the FourElectrons final state in the %s TH2D",
                            type_dataset, stack_info=True)

    try:
        histos["data_mu"].Draw("SAME P")
    except KeyError:
        logger.debug("ERROR: Failed to create the data histogram of the FourMuons final state in the %s TH2D",
                            type_dataset, stack_info=True)

    try:
        histos["data_elmu"].Draw("SAME P")
    except KeyError:
        logger.debug("ERROR: Failed to create the data histogram of the TwoMuonsTwoElectrons final state in the %s TH2D",
                            type_dataset, stack_info=True)
```



Distribution for the signal

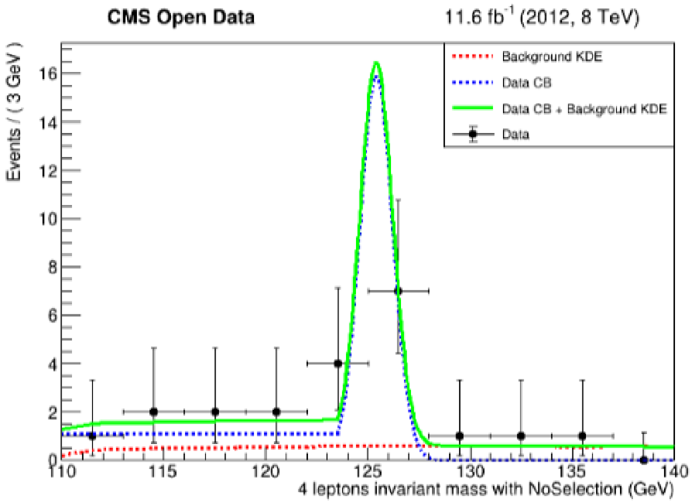


Distribution for the background

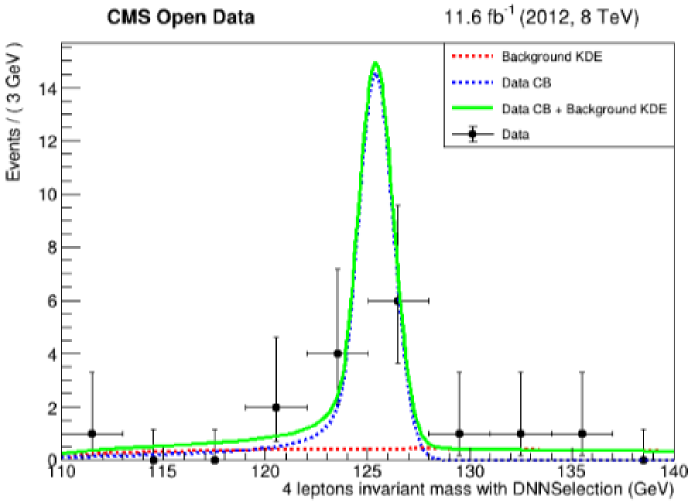
Higgs mass fit

fit_mass.py

```python
m4l = ROOT.RooRealVar("Higgs_mass",f"4 leptons invariant mass with {selection}", 110, 140,"GeV")
weight = ROOT.RooRealVar("Weight","Weight", 0, 1,"GeV")

sig = ROOT.RooDataSet("signal", "", sig_chain, ROOT.RooArgSet(m4l, weight) )
bkg = ROOT.RooDataSet("background", "", bkg_chain, ROOT.RooArgSet(m4l, weight) )
data = ROOT.RooDataSet("data", "", data_chain, ROOT.RooArgSet(m4l, weight) )

# Calculate signal fraction
sig_frac_count = sig.sumEntries()/(sig.sumEntries()+bkg.sumEntries())
bkg_frac_count = bkg.sumEntries()/(sig.sumEntries()+bkg.sumEntries())

# KDE for background. In this configuration the input data
# is mirrored over the right boundary to minimize edge effects in distribution
# that do not fall to zero towards the right edge
bkg_kde = ROOT.RooKeysPdf("bkg_kde", "bkg_kde", m4l, bkg, ROOT.RooKeysPdf.MirrorRight)

# Parameters and model for the fit of the simulated signal samples
meanHiggs_sig = ROOT.RooRealVar("meanHiggs_sig", "The mean of the Higgs CB for the signal", 125, 115, 135, "GeV")
sigmaHiggs = ROOT.RooRealVar("sigmaHiggs", "The width of Higgs CB", 5, 0., 20, "GeV")
alphaHiggs = ROOT.RooRealVar("alphaHiggs", "The tail of Higgs CB", 1.5, -5, 5)
nHiggs = ROOT.RooRealVar("nHiggs", "The normalization of Higgs CB", 1.5, 0, 10)
CBHiggs_sig = ROOT.RooCBShape("CBHiggs_sig","The Higgs Crystall Ball for the signal",
                              m4l, meanHiggs_sig, sigmaHiggs, alphaHiggs, nHiggs)

# Unbinned ML fit to signal
fitHiggs = CBHiggs_sig.fitTo(sig, ROOT.RooFit.Save(True), ROOT.RooFit.AsymptoticError(True))
fitHiggs.Print("v")

# Parameters and model for data fit
meanHiggs_data = ROOT.RooRealVar("m_{H}", "The mean of the Higgs CB for the data", 125, 115, 135, "GeV")
CBHiggs_data = ROOT.RooCBShape("CBHiggs_data","The Higgs Crystall Ball for the data",
                               m4l, meanHiggs_data, sigmaHiggs, alphaHiggs, nHiggs)

# Signal and background fractions
sig_frac= ROOT.RooRealVar("sigfrac", "signal fraction", sig_frac_count)
bkg_frac= ROOT.RooRealVar("bkg_coeff", "bkg fraction", bkg_frac_count)

# Total PDF of data and background
totPDF = ROOT.RooAddPdf("totPDF", "Higgs_data+bkg", ROOT.RooArgList(CBHiggs_data, bkg_kde), ROOT.RooArgList(sig_frac,bkg_frac))

# Unbinned ML fit to data
fitdata = totPDF.fitTo(data, ROOT.RooFit.Save(True))
```



Higgs mass fit without DNN selection



Higgs mass fit with DNN selection

	Without DNN selection	With DNN selection
Higgs mass from MC [GeV]	124.934 ± 0.016	124.938 ± 0.015
Higgs mass from data [GeV]	125.40 ± 0.26	125.39 ± 0.30