Searches for the Higgs boson in the tau-tau decay channel

using CMS Open Data

Final Project for CMEPDA exam

05/05/22

Domenico Riccardi[1] & Viola Floris[2]
[1] domenico.riccardi@cern.ch
[2] v.floris1@studenti.unipi.it

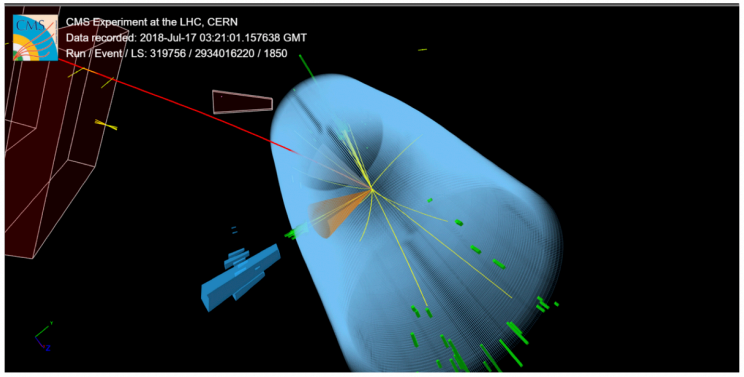
GitHub Repository

https://github.com/DomRic98/Searches-for-the-Higgs-boson-in-the-tau-tau-decay-channel

README.md

build passing | license GPL-3.0 | languages 4 | last commit today | total lines 20k | python 65.8%

Searches for the Higgs boson in the tau-tau decay channel



Event display of a p-p collision recorded at CMS with a candidate Higgs boson decaying into two tau leptons. [https://cds.cern.ch/record/2725256]

Description

Welcome to this repository!

Here, you find all the material produced for the final project for Computing Methods for Experimental Physics and Data Analysis exam at the University of Pisa. Running the Python and C++ scripts that you can find in the folders, you can explore a little part of a more complex analysis on the search of Higgs boson decay into two tau leptons.

Environments 1

github-pages Active

Languages

Python 65.8% C++ 32.1%
Batchfile 1.2% Makefile 0.9%

Struttura del progetto

Contents

1. **Studio dei Dataset e selezione degli eventi (ROOT/C++)** 
    - Fisica del processo
    - Utilizzo di RDataFrame per lo sviluppo del codice di selezione
    - Plotting delle distribuzioni delle variabili fisiche
2. **Implementazione di algoritmi ML per la classificazione sig/bkg (Python)** 
    - Artificial Neural Network
    - Random Forest Classifier
    - Plotting delle predizioni su Dataset di test
    - Studio sulla selezione degli iperparametri delle reti
    - Implementazione di vari unittest e documentazione del codice
3. **Sviluppo di algoritmi per il download dei Dataset in sequenza e in parallelo** 
    - Algoritmi e unittest

Icons by icons8.com

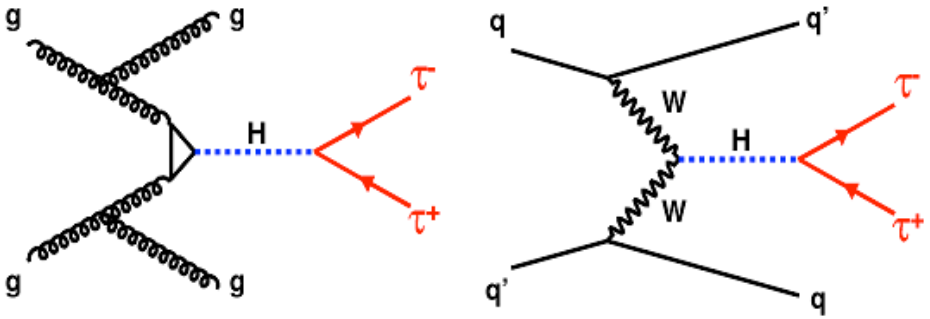
Studio dei Dataset e selezione degli eventi (ROOT/C++)

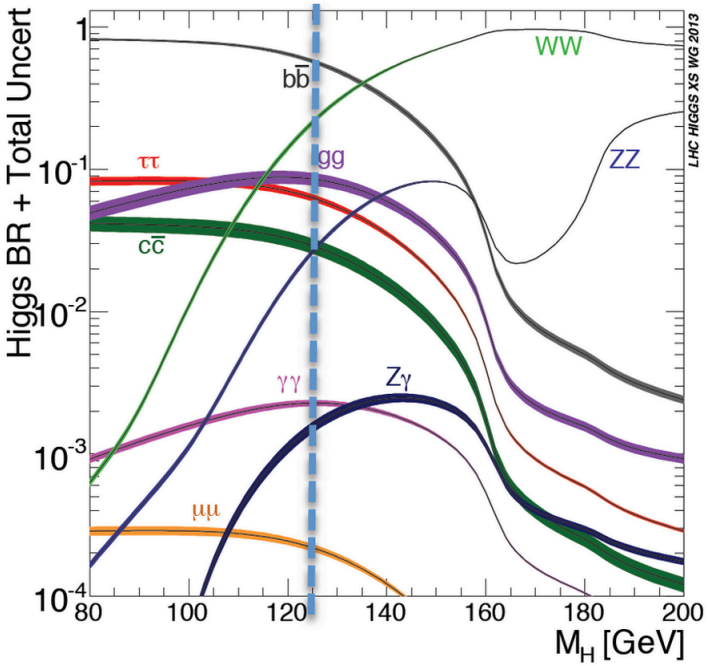
Prima Parte

Fisica del processo

- Obiettivo: Rivelare il decadimento $H \rightarrow \tau_\mu \tau_h$, dove con τ_μ si intende il processo $\tau \rightarrow \mu \nu_\mu \nu_\tau$ e con τ_h il conseguente decadimento in adroni

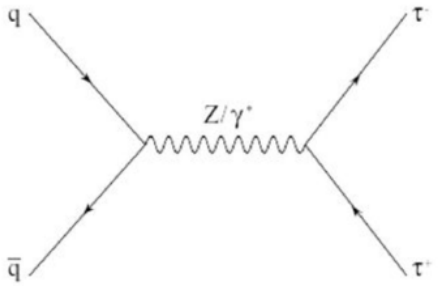
- Produzione dominante dell'Higgs a LHC:
  - Gluon gluon Fusion, $\sigma(ggH) \sim 49\,\text{pb}\,@\,13\text{TeV}$
  - Vector Boson Fusion, $\sigma(VBF) \sim 3.8\,\text{pb}\,@\,13\text{TeV}$





Background e Dataset

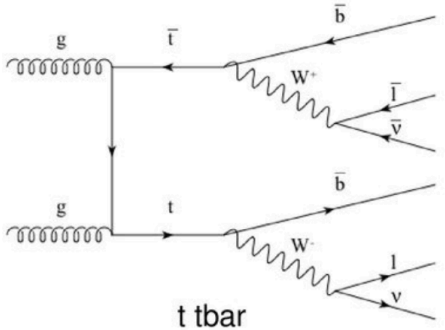
- I fondi principali per il nostro canale di segnale sono:
  - Drell-Yan: $Z/\gamma^* \rightarrow l^+l^-$
  - Produzione di coppie $t\bar{t}$
  - W + jets



Z → tau tau

MC sample (ROOT files)	Cross section [pb]	Number of events
GluGluToHToTauTau	19.6	476963
VBF_HToTauTau	1.55	491653
DYJetsToLL	3503.7	30458871
TTbar	225.2	6423106
W1JetsToLNu	6381.2	29784800
W2JetsToLNu	2039.8	30693853
W3JetsToLNu	612.5	15241144

MC samples (CMS OpenData, 2012) @ 8 TeV
con **sezione d'urto** e **numero di eventi** generati per canale



t tbar

CMS OpenData Dataset semantics

N	Variable	Type	Description
1	**event**	*unsigned long*	Event number
2	**HLT_IsoMu17_eta2p1_LooseIsoPFTau20**	*bool*	Flag indicating whether the event passed the respective trigger
3	**PV_npvs**	*int*	Number of primary vertices
4	**nMuon**	*unsigned int*	Number of muons in this event
5	**Muon_pt**	*float[nMuon]*	Transverse momentum of the muons
6	**Muon_eta**	*float[nMuon]*	Pseudorapidity of the muons
7	**Muon_phi**	*float[nMuon]*	Azimuth angle of the muons
8	**Muon_mass**	*float[nMuon]*	Mass of the muons
9	**Muon_charge**	*int[nMuon]*	Charge of the muons (either 1 or -1)
10	**Muon_tightId**	*bool[nMuon]*	Muon object identification with tight requirements
11	**nTau**	*unsigned int*	Number of taus in this event
12	**Tau_pt**	*float[nTau]*	Transverse momentum of the taus
13	**Tau_eta**	*float[nTau]*	Pseudorapidity of the taus
14	**Tau_phi**	*float[nTau]*	Azimuth angle of the taus
15	**Tau_mass**	*float[nTau]*	Mass of the taus
16	**Tau_charge**	*int[nTau]*	Charge of the taus (either 0, 1 or -1). 0 indicates that the charge could not be determined.
17	**Tau_decayMode**	*int[nTau]*	Hadronic decay mode of the taus
18	**Tau_relIso_all**	*float[nTau]*	Tau isolation divided by the transverse momentum
19	**Tau_idDecayMode**	*bool[nTau]*	Indicates passed decay mode finding of the taus
20	**Tau_idIso[VLoose,Loose,Medium,Tight]**	*bool[nTau]*	Indicates passed tau discriminator with [very loose, loose, medium, tight] requirements
21	**Tau_idAntiEle[Loose,Medium,Tight]**	*bool[nTau]*	Indicates passed anti-electron discriminator with [loose, medium, tight] requirements
22	**Tau_idAntiMu[Loose,Medium,Tight]**	*bool[nTau]*	Indicates passed anti-muon discriminator with [loose, medium, tight] requirements
23	**nJet**	*unsigned int*	Number of jets in this event
24	**Jet_pt**	*float[nJet]*	Transverse momentum of the jets
25	**Jet_eta**	*float[nJet]*	Pseudorapidity of the jets
26	**Jet_phi**	*float[nJet]*	Azimuth angle of the jets
27	**Jet_mass**	*float[nJet]*	Mass of the jets
28	**Jet_puId**	*bool[nJet]*	Jet pile-up identification flag
29	**Jet_btag**	*float[nJet]*	b-Jet identification score
30	**MET_pt**	*float*	Missing transverse energy
31	**MET_phi**	*float*	Azimuth angle of the missing transverse energy

GluGlutoHtoTauTau

Analysis code

- Abilita il multi-threading implicito di ROOT

```cpp
int main()
{
    // Implicit parallelism in ROOT

    ROOT::EnableImplicitMT();
    int poolSize = ROOT::GetImplicitMTPoolSize();
    std::cout << "Pool size used for multi-threading: "
              << poolSize << std::endl;

    std::vector<std::string> ROOT_files = {
        "GluGluToHToTauTau", "VBF_HToTauTau", // Signal MC
        "DYJetsToLL", "TTbar",
        "W1JetsToLNu", "W2JetsToLNu", "W3JetsToLNu",// Bkg MC
        "Run2012B_TauPlusX", "Run2012C_TauPlusX" // Real Data
    };

    auto start = std::chrono::system_clock::now();

    for (const std::string &sample : ROOT_files) {

        std::cout << "****************************************" << std::endl;
        std::cout << "Processing sample: " << sample << std::endl;
        ROOT::RDataFrame d_root("Events", sample + ".root");

        int num_events = *d_root.Count();
        const float integratedLuminosity = 11.467 * 1000.0; //(pb^-1)

        float weight = 1.;
        if(sample.find("Run") != 0)
            weight = (cross_section[sample]/num_events) * integratedLuminosity;
        auto d_MCweight = d_root.Define("weight", [weight](){ return weight;});

        std::cout << "\t Number of events in ROOT file: " << num_events
                  << std::endl;

        auto d_muon = muon_selection(d_MCweight);
        std::cout << "\t Number of events after muon selection: "
                  << *d_muon.Count() << std::endl;

        auto d_tau = tau_selection(d_muon);
        std::cout << "\t Number of events after tau selection: "
                  << *d_tau.Count() << std::endl;

        auto d_h2mutau = h2mutau(d_tau);
        std::cout << "\t Number of events after selecting candidates for the
                    Higgs (H->ττ): " << *d_h2mutau.Count() << std::endl;

        auto d_variables = ML_variables(d_h2mutau);
        auto d_jets = take_jet(d_variables);
        auto d = d_jets.Filter("muon_charge*tau_charge<0");

        // Save the new DataFrame of the events selected in the analysis
        procedure.
        d.Snapshot("Events", sample + "_selected.root", finalVariables);
    }

    auto end = std::chrono::system_clock::now();
    std::chrono::duration<double> elapsed_seconds = end-start;
    std::time_t end_time = std::chrono::system_clock::to_time_t(end);
    std::cout << "Finished computation at "
              << std::ctime(&end_time) << "Total elapsed time: "
              << elapsed_seconds.count() << "s" << std::endl;

}
```

analysis.cpp

Analysis code

- Definisco una nuova colonna del DataFrame: peso dell'evento MC

```
int main()
{
    // Implicit parallelism in ROOT

    ROOT::EnableImplicitMT();
    int poolSize = ROOT::GetImplicitMTPoolSize();
    std::cout << "Pool size used for multi-threading: "
              << poolSize << std::endl;

    std::vector<std::string> ROOT_files = {
        "GluGluToHToTauTau", "VBF_HToTauTau", // Signal MC
        "DYJetsToLL", "TTbar",
        "W1JetsToLNu", "W2JetsToLNu", "W3JetsToLNu",// Bkg MC
        "Run2012B_TauPlusX", "Run2012C_TauPlusX" // Real Data
    };

    auto start = std::chrono::system_clock::now();

    for (const std::string &sample : ROOT_files) {
      std::cout << "****************************************" << std::endl;
      std::cout << "Processing sample: " << sample << std::endl;
      ROOT::RDataFrame d_root("Events", sample + ".root");

      int num_events = *d_root.Count();
      const float integratedLuminosity = 11.467 * 1000.0; //(pb^-1)

      float weight = 1.;
      if(sample.find("Run") != 0)
          weight = (cross_section[sample]/num_events) * integratedLuminosity;
      auto d_MCweight = d_root.Define("weight", [weight](){ return weight;});
          std::cout << "\t Number of events in ROOT file: " << num_events
                    << std::endl;

          auto d_muon = muon_selection(d_MCweight);
          std::cout << "\t Number of events after muon selection: "
                    << *d_muon.Count() << std::endl;

          auto d_tau = tau_selection(d_muon);
          std::cout << "\t Number of events after tau selection: "
                    << *d_tau.Count() << std::endl;

          auto d_h2mutau = h2mutau(d_tau);
          std::cout << "\t Number of events after selecting candidates for the
                      Higgs (H->ττ): " << *d_h2mutau.Count() << std::endl;

          auto d_variables = ML_variables(d_h2mutau);
          auto d_jets = take_jet(d_variables);
          auto d = d_jets.Filter("muon_charge*tau_charge<0");

          // Save the new DataFrame of the events selected in the analysis
          procedure.
          d.Snapshot("Events", sample + "_selected.root", finalVariables);
      }

      auto end = std::chrono::system_clock::now();
      std::chrono::duration<double> elapsed_seconds = end-start;
      std::time_t end_time = std::chrono::system_clock::to_time_t(end);
      std::cout << "Finished computation at "
                << std::ctime(&end_time) << "Total elapsed time: "
                << elapsed_seconds.count() << "s" << std::endl;

}
```

analysis.cpp

Analysis code

- Flusso delle chiamate alle funzioni di selezione & salvataggio del nuovo RDataFrame come ROOT file

```
int main()
{
    // Implicit parallelism in ROOT

    ROOT::EnableImplicitMT();
    int poolSize = ROOT::GetImplicitMTPoolSize();
    std::cout << "Pool size used for multi-threading: "
              << poolSize << std::endl;

    std::vector<std::string> ROOT_files = {
        "GluGluToHToTauTau", "VBF_HToTauTau", // Signal MC
        "DYJetsToLL", "TTbar",
        "W1JetsToLNu", "W2JetsToLNu", "W3JetsToLNu",// Bkg MC
        "Run2012B_TauPlusX", "Run2012C_TauPlusX" // Real Data
    };

    auto start = std::chrono::system_clock::now();

    for (const std::string &sample : ROOT_files) {

        std::cout << "****************************************" << std::endl;
        std::cout << "Processing sample: " << sample << std::endl;
        ROOT::RDataFrame d_root("Events", sample + ".root");

        int num_events = *d_root.Count();
        const float integratedLuminosity = 11.467 * 1000.0; //(pb^-1)

        float weight = 1.;
        if(sample.find("Run") != 0)
            weight = (cross_section[sample]/num_events) * integratedLuminosity;
        auto d_MCweight = d_root.Define("weight", [weight](){ return weight;});

        std::cout << "\t Number of events in ROOT file: " << num_events
                  << std::endl;

        auto d_muon = muon_selection(d_MCweight);
        std::cout << "\t Number of events after muon selection: "
                  << *d_muon.Count() << std::endl;

        auto d_tau = tau_selection(d_muon);
        std::cout << "\t Number of events after tau selection: "
                  << *d_tau.Count() << std::endl;

        auto d_h2mutau = h2mutau(d_tau);
        std::cout << "\t Number of events after selecting candidates for the
                    Higgs (H->ττ): " << *d_h2mutau.Count() << std::endl;

        auto d_variables = ML_variables(d_h2mutau);
        auto d_jets = take_jet(d_variables);
        auto d = d_jets.Filter("muon_charge*tau_charge<0");

        // Save the new DataFrame of the events selected in the analysis
        procedure.
        d.Snapshot("Events", sample + "_selected.root", finalVariables);
    }

    auto end = std::chrono::system_clock::now();
    std::chrono::duration<double> elapsed_seconds = end-start;
    std::time_t end_time = std::chrono::system_clock::to_time_t(end);
    std::cout << "Finished computation at "
              << std::ctime(&end_time) << "Total elapsed time: "
              << elapsed_seconds.count() << "s" << std::endl;
}
```

analysis.cpp

Analysis code

Esempio di output

- Eseguo dal prompt ROOT

```
domenico@MacBook-Pro-di-domenico ROOT_workspace % root -l
root [0] .L analysis.cpp
root [1] main()
```

- Oppure produco e run l'eseguibile con

```
domenico@MacBook-Pro-di-domenico ROOT_workspace % g++ analysis.cpp `root-config --cflags --glibs` -o analysis
domenico@MBP-di-domenico ROOT_workspace % ./analysis
```

```
Pool size used for multi-threading: 4
*****************************************************************************
Processing sample: GluGluToHToTauTau
     Number of events in ROOT file: 476963
     Number of events after muon selection: 32196
     Number of events after tau selection: 8085
     Number of events after selecting candidates for the Higgs (H->ττ): 8073
*****************************************************************************
Processing sample: VBF_HToTauTau
     Number of events in ROOT file: 491653
     Number of events after muon selection: 47233
     Number of events after tau selection: 10493
     Number of events after selecting candidates for the Higgs (H->ττ): 10441
*****************************************************************************
Processing sample: DYJetsToLL
     Number of events in ROOT file: 30458871
     Number of events after muon selection: 4704720
     Number of events after tau selection: 70138
     Number of events after selecting candidates for the Higgs (H->ττ): 66311
*****************************************************************************
Processing sample: TTbar
     Number of events in ROOT file: 6423106
     Number of events after muon selection: 848658
     Number of events after tau selection: 36921
     Number of events after selecting candidates for the Higgs (H->ττ): 35324
*****************************************************************************
Processing sample: W1JetsToLNu
     Number of events in ROOT file: 29784800
     Number of events after muon selection: 1208333
     Number of events after tau selection: 24015
     Number of events after selecting candidates for the Higgs (H->ττ): 23962
*****************************************************************************
Processing sample: W2JetsToLNu
     Number of events in ROOT file: 30693853
     Number of events after muon selection: 2281445
     Number of events after tau selection: 42476
     Number of events after selecting candidates for the Higgs (H->ττ): 42341
*****************************************************************************
Processing sample: W3JetsToLNu
     Number of events in ROOT file: 15241144
     Number of events after muon selection: 1467529
     Number of events after tau selection: 23989
     Number of events after selecting candidates for the Higgs (H->ττ): 23889
*****************************************************************************
Processing sample: Run2012B_TauPlusX
     Number of events in ROOT file: 35647508
     Number of events after muon selection: 9028134
     Number of events after tau selection: 136411
     Number of events after selecting candidates for the Higgs (H->ττ): 133783
*****************************************************************************
Processing sample: Run2012C_TauPlusX
     Number of events in ROOT file: 51303171
     Number of events after muon selection: 12578182
     Number of events after tau selection: 210897
     Number of events after selecting candidates for the Higgs (H->ττ): 206812
Finished computation on Tue Apr 26 16:32:57 2022
Total elapsed time: 3418.47s
```

analysis.cpp

Analysis code

Selection functions

```
// namespace-alias-definition
using RFilter =
      ROOT::RDF::RInterface<ROOT::Detail::RDF::RJittedFilter>;

/*
* Perform a muons selection by selecting the final state with
* at least one good μ candidate.
*/
template <typename T>
RFilter muon_selection(T &d)
{
    auto d_temp = d.Filter("nMuon>0 &
                    HLT_IsoMu17_eta2p1_LooseIsoPFTau20==1”)
                   .Define("goodMuons", "(abs(Muon_eta)<2.1) &
                           (Muon_pt>17) & (Muon_tightId==1)”)
                   .Filter("ROOT::VecOps::Sum(goodMuons)>0");
    return d_temp;
}
```

```
/*
 * Find at least one good tau.
 */

template <typename T>
RFilter tau_selection(T &d)
{
    auto d_temp = d.Filter("nTau>0 &
                      HLT_IsoMu17_eta2p1_LooseIsoPFTau20==1")
                   .Define("goodTaus", "Tau_charge!=0 &
                           abs(Tau_eta)<2.3 & Tau_pt>20 &
                           Tau_idDecayMode==1 &
                           Tau_idIsoTight==1 &
                           Tau_idAntiEleTight==1 &
                           Tau_idAntiMuTight==1")
                   .Filter("ROOT::VecOps::Sum(goodTaus)>0");
    return d_temp;
}
```

analysis.cpp

Analysis code

- Definisco vari constraints sulle variabili del jet

```
/*
 * Select a "good jet" in the sample.
 */
template <typename T>
RFilter take_jet(T &d)
{
    auto first_jet = [] (ROOT::RVec<unsigned long> &goodJets_idx,
                         ROOT::RVec<float> &jet_var)
    {
        auto var = ROOT::VecOps::Take(jet_var, goodJets_idx);
        if (goodJets_idx.size()>0)
            return var[0];
        return -999.f;
    };

    auto second_jet = [] (ROOT::RVec<unsigned long> &goodJets_idx,
                          ROOT::RVec<float> &jet_var)
    {
        auto var = ROOT::VecOps::Take(jet_var, goodJets_idx);
        if (goodJets_idx.size()>1)
            return var[1];
        return -999.f;
    };

    auto compute_mjj = [] (int nGoodJets, ROOT::Math::PtEtaPhiMVector jjp4)
    {
        if (nGoodJets>1) {
            return float(jjp4.M());
        }
        return -999.f;
    };

    auto compute_ptjj = [] (int nGoodJets, ROOT::Math::PtEtaPhiMVector jjp4)
    {
        if (nGoodJets>1) {
            return float(jjp4.Pt());
        }
        return -999.f;
    };

    auto compute_jdeta = [](int nGoodJets, float jeta_1, float jeta_2){
        if (nGoodJets>1) {
            return jeta_1 - jeta_2;
        }
        return -999.f;
    };

    auto d_temp = d.Define("goodJets", "Jet_pt>30 & abs(Jet_eta)<4.7 & Jet_puId==1")
                   .Define("goodJets_idx", "ROOT::VecOps::Nonzero(goodJets)")
                   .Define("jpt_1", first_jet, {"goodJets_idx", "Jet_pt"})
                   .Define("jpt_2", second_jet, {"goodJets_idx", "Jet_pt"})
                   .Define("jeta_1", first_jet, {"goodJets_idx", "Jet_eta"})
                   .Define("jeta_2", second_jet, {"goodJets_idx", "Jet_eta"})
                   .Define("jphi_1", first_jet, {"goodJets_idx", "Jet_phi"})
                   .Define("jphi_2", second_jet, {"goodJets_idx", "Jet_phi"})
                   .Define("jm_1", first_jet, {"goodJets_idx", "Jet_mass"})
                   .Define("jm_2", second_jet, {"goodJets_idx", "Jet_mass"})
                   .Define("jbtag_1", first_jet, {"goodJets_idx", "Jet_btag"})
                   .Define("jbtag_2", second_jet, {"goodJets_idx", "Jet_btag"})
                   .Define("nGoodJets", "int(goodJets_idx.size())")
                   .Define("jp4_1", "ROOT::Math::PtEtaPhiMVector(jpt_1, jeta_1,
                                                                 jphi_1, jm_1)")
                   .Define("jp4_2", "ROOT::Math::PtEtaPhiMVector(jpt_2, jeta_2,
                                                                 jphi_2, jm_2)")
                   .Define("jjp4", “jp4_1+jp4_2”)
                   .Define("jj_m", compute_mjj, {"nGoodJets", "jjp4"})
                   .Define("jj_pt", compute_ptjj, {"nGoodJets", "jjp4"})
                   .Define("jj_delta", compute_jdeta, {"nGoodJets", "jeta_1", "jeta_2"});
    return d_temp;
}
```

analysis.cpp

Analysis code

- Nuove colonne nel DataFrame per il primo e il secondo jet

```
/*
 * Select a "good jet" in the sample.
 */
template <typename T>
RFilter take_jet(T &d)
{
    auto first_jet = [] (ROOT::RVec<unsigned long> &goodJets_idx,
                         ROOT::RVec<float> &jet_var)
    {
        auto var = ROOT::VecOps::Take(jet_var, goodJets_idx);
        if (goodJets_idx.size()>0)
            return var[0];
        return -999.f;
    };

    auto second_jet = [] (ROOT::RVec<unsigned long> &goodJets_idx,
                          ROOT::RVec<float> &jet_var)
    {
        auto var = ROOT::VecOps::Take(jet_var, goodJets_idx);
        if (goodJets_idx.size()>1)
            return var[1];
        return -999.f;
    };

    auto compute_mjj = [] (int nGoodJets, ROOT::Math::PtEtaPhiMVector jjp4)
    {
        if (nGoodJets>1) {
            return float(jjp4.M());
        }
        return -999.f;
    };

    auto compute_ptjj = [] (int nGoodJets, ROOT::Math::PtEtaPhiMVector jjp4)
    {
        if (nGoodJets>1) {
            return float(jjp4.Pt());
        }
        return -999.f;
    };

    auto compute_jdeta = [](int nGoodJets, float jeta_1, float jeta_2){
        if (nGoodJets>1) {
            return jeta_1 - jeta_2;
        }
        return -999.f;
    };

    auto d_temp = d.Define("goodJets", "Jet_pt>30 & abs(Jet_eta)<4.7 & Jet_puId==1")
                   .Define("goodJets_idx", "ROOT::VecOps::Nonzero(goodJets)")
                   .Define("jpt_1", first_jet, {"goodJets_idx", "Jet_pt"})
                   .Define("jpt_2", second_jet, {"goodJets_idx", "Jet_pt"})
                   .Define("jeta_1", first_jet, {"goodJets_idx", "Jet_eta"})
                   .Define("jeta_2", second_jet, {"goodJets_idx", "Jet_eta"})
                   .Define("jphi_1", first_jet, {"goodJets_idx", "Jet_phi"})
                   .Define("jphi_2", second_jet, {"goodJets_idx", "Jet_phi"})
                   .Define("jm_1", first_jet, {"goodJets_idx", "Jet_mass"})
                   .Define("jm_2", second_jet, {"goodJets_idx", "Jet_mass"})
                   .Define("jbtag_1", first_jet, {"goodJets_idx", "Jet_btag"})
                   .Define("jbtag_2", second_jet, {"goodJets_idx", "Jet_btag"})
                   .Define("nGoodJets", "int(goodJets_idx.size())")
                   .Define("jp4_1", "ROOT::Math::PtEtaPhiMVector(jpt_1, jeta_1,
                                                                 jphi_1, jm_1)")
                   .Define("jp4_2", "ROOT::Math::PtEtaPhiMVector(jpt_2, jeta_2,
                                                                 jphi_2, jm_2)")
                   .Define("jjp4", “jp4_1+jp4_2”)
                   .Define("jj_m", compute_mjj, {"nGoodJets", "jjp4"})
                   .Define("jj_pt", compute_ptjj, {"nGoodJets", "jjp4"})
                   .Define("jj_delta", compute_jdeta, {"nGoodJets", "jeta_1", "jeta_2"});
    return d_temp;
}
```

analysis.cpp

Analysis code

- Creo oggetti di tipo 4-vettore con PtEtaPhiMVector() e calcolo nuove variabili

```
/*
 * Select a "good jet" in the sample.
 */
template <typename T>
RFilter take_jet(T &d)
{
    auto first_jet = [] (ROOT::RVec<unsigned long> &goodJets_idx,
                         ROOT::RVec<float> &jet_var)
    {
        auto var = ROOT::VecOps::Take(jet_var, goodJets_idx);
        if (goodJets_idx.size()>0)
            return var[0];
        return -999.f;
    };

    auto second_jet = [] (ROOT::RVec<unsigned long> &goodJets_idx,
                          ROOT::RVec<float> &jet_var)
    {
        auto var = ROOT::VecOps::Take(jet_var, goodJets_idx);
        if (goodJets_idx.size()>1)
            return var[1];
        return -999.f;
    };

    auto compute_mjj = [] (int nGoodJets, ROOT::Math::PtEtaPhiMVector jjp4)
    {
        if (nGoodJets>1) {
            return float(jjp4.M());
        }
        return -999.f;
    };

    auto compute_ptjj = [] (int nGoodJets, ROOT::Math::PtEtaPhiMVector jjp4)
    {
        if (nGoodJets>1) {
            return float(jjp4.Pt());
        }
        return -999.f;
    };

    auto compute_jdeta = [](int nGoodJets, float jeta_1, float jeta_2){
        if (nGoodJets>1) {
            return jeta_1 - jeta_2;
        }
        return -999.f;
    };

    auto d_temp = d.Define("goodJets", "Jet_pt>30 & abs(Jet_eta)<4.7 & Jet_puId==1")
                   .Define("goodJets_idx", "ROOT::VecOps::Nonzero(goodJets)")
                   .Define("jpt_1", first_jet, {"goodJets_idx", "Jet_pt"})
                   .Define("jpt_2", second_jet, {"goodJets_idx", "Jet_pt"})
                   .Define("jeta_1", first_jet, {"goodJets_idx", "Jet_eta"})
                   .Define("jeta_2", second_jet, {"goodJets_idx", "Jet_eta"})
                   .Define("jphi_1", first_jet, {"goodJets_idx", "Jet_phi"})
                   .Define("jphi_2", second_jet, {"goodJets_idx", "Jet_phi"})
                   .Define("jm_1", first_jet, {"goodJets_idx", "Jet_mass"})
                   .Define("jm_2", second_jet, {"goodJets_idx", "Jet_mass"})
                   .Define("jbtag_1", first_jet, {"goodJets_idx", "Jet_btag"})
                   .Define("jbtag_2", second_jet, {"goodJets_idx", "Jet_btag"})
                   .Define("nGoodJets", "int(goodJets_idx.size())")
                   .Define("jp4_1", "ROOT::Math::PtEtaPhiMVector(jpt_1, jeta_1,
                                                                 jphi_1, jm_1)")
                   .Define("jp4_2", "ROOT::Math::PtEtaPhiMVector(jpt_2, jeta_2,
                                                                 jphi_2, jm_2)")
                   .Define("jjp4", "jp4_1+jp4_2")
                   .Define("jj_m", compute_mjj, {"nGoodJets", "jjp4"})
                   .Define("jj_pt", compute_ptjj, {"nGoodJets", "jjp4"})
                   .Define("jj_delta", compute_jdeta, {"nGoodJets", "jeta_1", "jeta_2"});
    return d_temp;
}
```

analysis.cpp

Analysis code

- Trovo la coppia che rappresenta un “buon” candidato Higgs: $H \rightarrow \tau_\mu \tau_h$

```
/*
 * Define a function which constructs a muon-tau pair from the
 * collections of goodMuons and goodTaus.
 */
template <typename T>
RFilter h2mutau(T &d)
{
    auto goodPair = [] (ROOT::RVec<int> &goodMuons,
                        ROOT::RVec<int> &goodTaus,
                        ROOT::RVec<float> &eta1,
                        ROOT::RVec<float> &eta2,
                        ROOT::RVec<float> &phi1,
                        ROOT::RVec<float> &phi2,
                        ROOT::RVec<float> &pt1,
                        ROOT::RVec<float>& iso2,
                        ROOT::RVec<int>& charge1,
                        ROOT::RVec<int>& charge2)
    {
        auto muons_idx = ROOT::VecOps::Nonzero(goodMuons); //idx of μ
        auto taus_idx = ROOT::VecOps::Nonzero(goodTaus);   //idx of τ
        // Find valid pairs based on delta r
        if(taus_idx.size() == 1 & muons_idx.size() == 1){
            // The function DeltaR computes ΔR=√(η1−η2)^2+(ϕ1−ϕ2)^2 of
               the given collections eta1, eta2, phi1 and phi2
            auto DeltaR = ROOT::VecOps::DeltaR(
                                eta1[muons_idx[0]], eta2[taus_idx[0]],
                                phi1[muons_idx[0]], phi2[taus_idx[0]]);
// Select the pair with ΔR > 0.5
            if(DeltaR > 0.5){
                return std::pair<int, int>(muons_idx[0], taus_idx[0]);}
            return std::pair<int, int>(-999, -999);
        }
        else{
            // If there is more than one good muon or good tau, find
               the muon with the maximum pt and the tau with the
               minimum value of isolation.
            auto idx_pt_max = ROOT::VecOps::ArgMax(pt1);
            auto idx_iso_min = ROOT::VecOps::ArgMin(iso2);
            auto DeltaR = ROOT::VecOps::DeltaR(
                                eta1[idx_pt_max], eta2[idx_iso_min],
                                phi1[idx_pt_max], phi2[idx_iso_min]);
            if(DeltaR > 0.5){
                return std::pair<int, int >(idx_pt_max, idx_iso_min);}
            return std::pair<int, int>(-999, -999);}

    };

    auto d_temp = d.Define("h2mutau", goodPair, {"goodMuons",
             "goodTaus", "Muon_eta", "Tau_eta", "Muon_phi", "Tau_phi",
             "Muon_pt", "Tau_relIso_all", "Muon_charge", "Tau_charge"})
                   .Define("idx_muon", “h2mutau.first")
                   .Filter("idx_muon >= 0")
                   .Define("idx_tau", "h2mutau.second");
    return d_temp;
}
```

analysis.cpp

Analysis code

- Definisco (e calcolo) le variabili del nuovo RDataFrame che utilizzeremo per il Machine Learning

```
template <typename T>
RFilter ML_variables(T &d)
{
    // Lambda function to calculate the transverse mass between
       lepton and MET.
    auto compute_mt = [](float pt, float phi,
                         float met_pt, float met_phi){
        double dphi = ROOT::VecOps::DeltaPhi(phi, met_phi);
        return std::sqrt(2.0 * pt * met_pt * (1.0 -
                        std::cos(dphi)));
    };

    auto compute_DeltaR = [](float mu_eta, float tau_eta,
                             float mu_phi, float tau_phi){
        return ROOT::VecOps::DeltaR(mu_eta, tau_eta,
                                    mu_phi, tau_phi);
    };

    auto d_temp = d.Define("muon_pt", "Muon_pt[idx_muon]")
                   .Define("muon_eta", "Muon_eta[idx_muon]")
                   .Define("muon_phi", "Muon_phi[idx_muon]")
                   .Define("muon_m", "Muon_mass[idx_muon]")
                   .Define("muon_charge","Muon_charge[idx_muon]")
                   .Define("muon_p4",
                           "ROOT::Math::PtEtaPhiMVector(muon_pt,
                            muon_eta, muon_phi, muon_m)")
                   .Define("muon_iso",
                           "Muon_pfRelIso03_all[idx_muon]")
                   .Define("tau_pt", "Tau_pt[idx_tau]")
                   .Define("tau_eta", "Tau_eta[idx_tau]")
                   .Define("tau_phi", "Tau_phi[idx_tau]")
                   .Define("tau_m", "Tau_mass[idx_tau]")
                   .Define("tau_charge", "Tau_charge[idx_muon]")
                   .Define("tau_iso", "Tau_relIso_all[idx_tau]")
                   .Define("tau_p4", "ROOT::Math::PtEtaPhiMVector
                            (tau_pt, tau_eta, tau_phi, tau_m)")
                   .Define("p4", "muon_p4+tau_p4")
                   .Define("m_vis", "p4.M()")
                   .Define("pt_vis", "p4.Pt()")
                   .Define("mt_mu", compute_mt,
                    {"muon_pt", "muon_phi", "MET_pt", "MET_phi"})
                   .Define("mt_tau", compute_mt,
                    {"tau_pt", "tau_phi", "MET_pt", "MET_phi"})
                   .Define("dRmu_tau", compute_DeltaR,
                    {"muon_eta","tau_eta","muon_phi","tau_phi"});

    return d_temp;
}
```

analysis.cpp

Plotting code

- Main function per la produzione delle distribuzioni delle variabili elencate prima

```
int main()
{
    // Create a new folder to contain the distribution plots produced
    std::string dirName = "Distribution_plots";
    std::stringstream bufH;
    bufH << dirName;
    if (!std::experimental::filesystem::exists(bufH.str()))
    {
        std::experimental::filesystem::create_directories(bufH.str());
    }

    // Define a std::map that contains the pair sample-RDataFrame and a
       std::vector of ROOT files' names
    std::map<std::string, RFilter> samples;
    std::vector<std::string> ROOT_files = {
        "GluGluToHToTauTau_selected",
        "VBF_HToTauTau_selected",
        "DYJetsToLL_selected",
        "TTbar_selected",
        "W*JetsToLNu_selected",
        "Run2012*_TauPlusX_selected"
    };

    // For loop to process the events in ROOT files and to apply final
       cuts to reduce the background (mostly W+jets)
    for (const std::string &channel: ROOT_files) {
        std::cout << "\tProcessing: " << channel << std::endl;
        ROOT::RDataFrame d("Events", channel + ".root");
        auto d_filter = d.Filter("mt_mu<30 & muon_iso<0.1")
                         .Filter("tau_pt>25 & muon_pt>20");
        samples.insert(std::pair<std::string,RFilter>(channel,d_filter));
        std::cout << "\tNumber of events: "
                  << *samples.at(channel).Count() << std::endl;
    }

    // Plot distribution of physical variables by using the plotting
       function
    std::cout << " Plotting distribution" << std::endl;
    plotting(samples, "m_vis", "Visible mass m_{#mu}+m_{#tau}",
                      "m_{vis}", "[GeV]", 30, 20, 140);
    plotting(samples, "pt_vis", "Visible transverse momentum",
                      "p_{T,vis}", "[GeV]", 30, 0, 60);
    plotting(samples, "tau_pt", "#tau transverse momentum", "p_{T
                      (#tau )", "[GeV]", 30, 17, 70);
    plotting(samples, "muon_pt", "#mu transverse momentum", "p_{T}(#mu)",
                      "[GeV]", 30, 17, 70);
    plotting(samples, "muon_eta", "#mu pseudorapidity", "#eta(#mu)",
                      "", 30, -2.1, 2.1);

    // And other 28 variables!
}
```

plotting.cpp

Plotting code

- Creo istogrammi per ciascun sample

```
template <typename T>
void plotting(T samples, std::string variable, const char *title, std::string xlabel_,
std::string unit, int bins, float lim_left, float lim_right) {
    std::cout << "\tPlotting: " << variable << std::endl;

    TCanvas *c = new TCanvas("c","c", 1000, 700);
    gStyle->SetOptStat(0);
    gStyle->SetTitleFontSize(0.05);
    // Define six 1D histograms for each channel
    auto h1 = samples.at("GluGluToHToTauTau_selected")
              .Histo1D({"", "", bins, lim_left, lim_right}, variable, "weight");
    auto h2 = samples.at("VBF_HToTauTau_selected")
              .Histo1D({"", "", bins, lim_left, lim_right}, variable, "weight");
    auto h3 = samples.at("DYJetsToLL_selected")
              .Histo1D({"", "", bins, lim_left, lim_right}, variable, "weight");
    auto h4 = samples.at("TTbar_selected")
              .Histo1D({"", "", bins, lim_left, lim_right}, variable, "weight");
    auto h5 = samples.at("W*JetsToLNu_selected")
              .Histo1D({"", "", bins, lim_left, lim_right}, variable, "weight");
    auto h6 = samples.at("Run2012*_TauPlusX_selected")
              .Histo1D({"", "", bins, lim_left, lim_right}, variable);

    std::vector<decltype(h1)> hist = {h1, h2, h3, h4, h5};
    std::vector<decltype(kRed+1)> color =
                          {kRed+1, kMagenta-9, kGreen-3, kAzure+7, kOrange-3};
    // Assign the colours to each histogram
    for (int i=0; i<hist.size(); i++) {
        hist[i]->SetLineColor(color[i]);
        hist[i]->SetFillColor(color[i]);
    }
    // Create a stacked histrogram with MC simulation samples
    THStack *hs = new THStack("hs", "");
    for (auto &h : hist) {
        hs->Add(h.GetPtr());
    }
    hs->Draw("HISTO");
    // Draw the Data with black points and error bars
    h6->SetMarkerStyle(kFullCircle);
    h6->SetMarkerColor(kBlack);
    h6->SetLineColor(kBlack);
    h6->DrawClone("ESAME");

    // Set the x-y axis labels and calculate the binning
    auto binning = (lim_right - lim_left)/bins;
    std::stringstream ss;
    ss.precision(2);
    ss << binning;
    std::string y_label = "Candidates / " + ss.str() + " " + unit;
    std::string x_label = xlabel_ + " " + unit;
    hs->GetXaxis()->SetTitle(x_label.c_str());
    hs->GetYaxis()->SetTitle(y_label.c_str());
    hs->SetTitle(title);

    // Draw a Tlegend for Real Data and MC samples
    TLegend *legend = new TLegend(0.65, 0.65, 0.98, 0.88);
    legend->AddEntry(h1.GetPtr(),"gg#rightarrow H #rightarrow#tau_{#mu}#tau_{h}","f");
    legend->AddEntry(h2.GetPtr(),"qq#rightarrow H #rightarrow#tau_{#mu}#tau_{h}","f");
    legend->AddEntry(h3.GetPtr(),"Z#rightarrow ll","f");
    legend->AddEntry(h4.GetPtr(),"t#bar{t}", "f");
    legend->AddEntry(h5.GetPtr(),"W+jets", "f");
    legend->AddEntry(h6.GetPtr(), "Data");
    legend->SetBorderSize(0);
    legend->SetFillStyle(0);
    legend->Draw();

    float xposition_lt = hs->GetXaxis()->GetXmax() *0.03 + hs->GetXaxis()->GetXmin();
    TLatex *lt = new TLatex(xposition_lt, hs->GetMaximum()*0.95,
      "#scale[0.8]{#splitline{CMS #bf{#it{Open Data}}}{#bf{11.5 fb^{-1} (8 TeV)}}}");
    lt->Draw("Same");

    // Save the plot in the correct directory
    std::string name = "./Distribution_plots/" + variable + ".pdf";
    c->SaveAs(name.c_str());
    c->Update();
}
```

plotting.cpp

Plotting code

- Assegno i colori ad ogni istogramma e definisco un TStack per contenerli tutti

```
template <typename T>
void plotting(T samples, std::string variable, const char *title, std::string xlabel_,
std::string unit, int bins, float lim_left, float lim_right) {
    std::cout << "\tPlotting: " << variable << std::endl;

    TCanvas *c = new TCanvas("c","c", 1000, 700);
    gStyle->SetOptStat(0);
    gStyle->SetTitleFontSize(0.05);
    // Define six 1D histograms for each channel
    auto h1 = samples.at("GluGluToHToTauTau_selected")
              .Histo1D({"", "", bins, lim_left, lim_right}, variable, "weight");
    auto h2 = samples.at("VBF_HToTauTau_selected")
              .Histo1D({"", "", bins, lim_left, lim_right}, variable, "weight");
    auto h3 = samples.at("DYJetsToLL_selected")
              .Histo1D({"", "", bins, lim_left, lim_right}, variable, "weight");
    auto h4 = samples.at("TTbar_selected")
              .Histo1D({"", "", bins, lim_left, lim_right}, variable, "weight");
    auto h5 = samples.at("W*JetsToLNu_selected")
              .Histo1D({"", "", bins, lim_left, lim_right}, variable, "weight");
    auto h6 = samples.at("Run2012*_TauPlusX_selected")
              .Histo1D({"", "", bins, lim_left, lim_right}, variable);

    std::vector<decltype(h1)> hist = {h1, h2, h3, h4, h5};
    std::vector<decltype(kRed+1)> color =
                          {kRed+1, kMagenta-9, kGreen-3, kAzure+7, kOrange-3};
    // Assign the colours to each histogram
    for (int i=0; i<hist.size(); i++) {
        hist[i]->SetLineColor(color[i]);
        hist[i]->SetFillColor(color[i]);
    }
    // Create a stacked histrogram with MC simulation samples
    THStack *hs = new THStack("hs", "");
    for (auto &h : hist) {
        hs->Add(h.GetPtr());
    }
    hs->Draw("HISTO");
    // Draw the Data with black points and error bars
    h6->SetMarkerStyle(kFullCircle);
    h6->SetMarkerColor(kBlack);
    h6->SetLineColor(kBlack);
    h6->DrawClone("ESAME");

    // Set the x-y axis labels and calculate the binning
    auto binning = (lim_right - lim_left)/bins;
    std::stringstream ss;
    ss.precision(2);
    ss << binning;
    std::string y_label = "Candidates / " + ss.str() + " " + unit;
    std::string x_label = xlabel_ + " " + unit;
    hs->GetXaxis()->SetTitle(x_label.c_str());
    hs->GetYaxis()->SetTitle(y_label.c_str());
    hs->SetTitle(title);

    // Draw a Tlegend for Real Data and MC samples
    TLegend *legend = new TLegend(0.65, 0.65, 0.98, 0.88);
    legend->AddEntry(h1.GetPtr(),"gg#rightarrow H #rightarrow#tau_{#mu}#tau_{h}","f");
    legend->AddEntry(h2.GetPtr(),"qq#rightarrow H #rightarrow#tau_{#mu}#tau_{h}","f");
    legend->AddEntry(h3.GetPtr(),"Z#rightarrow ll","f");
    legend->AddEntry(h4.GetPtr(),"t#bar{t}", "f");
    legend->AddEntry(h5.GetPtr(),"W+jets", "f");
    legend->AddEntry(h6.GetPtr(), "Data");
    legend->SetBorderSize(0);
    legend->SetFillStyle(0);
    legend->Draw();

    float xposition_lt = hs->GetXaxis()->GetXmax() *0.03 + hs->GetXaxis()->GetXmin();
    TLatex *lt = new TLatex(xposition_lt, hs->GetMaximum()*0.95,
      "#scale[0.8]{#splitline{CMS #bf{#it{Open Data}}}{#bf{11.5 fb^{-1} (8 TeV)}}}");
    lt->Draw("Same");

    // Save the plot in the correct directory
    std::string name = "./Distribution_plots/" + variable + ".pdf";
    c->SaveAs(name.c_str());
    c->Update();
}
```

plotting.cpp

Plotting code

- Imposto le label per gli assi e creo la TLegend

```
template <typename T>
void plotting(T samples, std::string variable, const char *title, std::string xlabel_,
std::string unit, int bins, float lim_left, float lim_right) {
    std::cout << "\tPlotting: " << variable << std::endl;

    TCanvas *c = new TCanvas("c","c", 1000, 700);
    gStyle->SetOptStat(0);
    gStyle->SetTitleFontSize(0.05);
    // Define six 1D histograms for each channel
    auto h1 = samples.at("GluGluToHToTauTau_selected")
              .Histo1D({"", "", bins, lim_left, lim_right}, variable, "weight");
    auto h2 = samples.at("VBF_HToTauTau_selected")
              .Histo1D({"", "", bins, lim_left, lim_right}, variable, "weight");
    auto h3 = samples.at("DYJetsToLL_selected")
              .Histo1D({"", "", bins, lim_left, lim_right}, variable, "weight");
    auto h4 = samples.at("TTbar_selected")
              .Histo1D({"", "", bins, lim_left, lim_right}, variable, "weight");
    auto h5 = samples.at("W*JetsToLNu_selected")
              .Histo1D({"", "", bins, lim_left, lim_right}, variable, "weight");
    auto h6 = samples.at("Run2012*_TauPlusX_selected")
              .Histo1D({"", "", bins, lim_left, lim_right}, variable);

    std::vector<decltype(h1)> hist = {h1, h2, h3, h4, h5};
    std::vector<decltype(kRed+1)> color =
                          {kRed+1, kMagenta-9, kGreen-3, kAzure+7, kOrange-3};
    // Assign the colours to each histogram
    for (int i=0; i<hist.size(); i++) {
        hist[i]->SetLineColor(color[i]);
        hist[i]->SetFillColor(color[i]);
    }
    // Create a stacked histrogram with MC simulation samples
    THStack *hs = new THStack("hs", "");
    for (auto &h : hist) {
        hs->Add(h.GetPtr());
    }
    hs->Draw("HISTO");
    // Draw the Data with black points and error bars
    h6->SetMarkerStyle(kFullCircle);
    h6->SetMarkerColor(kBlack);
    h6->SetLineColor(kBlack);
    h6->DrawClone("ESAME");

    // Set the x-y axis labels and calculate the binning
    auto binning = (lim_right - lim_left)/bins;
    std::stringstream ss;
    ss.precision(2);
    ss << binning;
    std::string y_label = "Candidates / " + ss.str() + " " + unit;
    std::string x_label = xlabel_ + " " + unit;
    hs->GetXaxis()->SetTitle(x_label.c_str());
    hs->GetYaxis()->SetTitle(y_label.c_str());
    hs->SetTitle(title);

    // Draw a Tlegend for Real Data and MC samples
    TLegend *legend = new TLegend(0.65, 0.65, 0.98, 0.88);
    legend->AddEntry(h1.GetPtr(),"gg#rightarrow H #rightarrow#tau_{#mu}#tau_{h}","f");
    legend->AddEntry(h2.GetPtr(),"qq#rightarrow H #rightarrow#tau_{#mu}#tau_{h}","f");
    legend->AddEntry(h3.GetPtr(),"Z#rightarrow ll","f");
    legend->AddEntry(h4.GetPtr(),"t#bar{t}", "f");
    legend->AddEntry(h5.GetPtr(),"W+jets", "f");
    legend->AddEntry(h6.GetPtr(), "Data");
    legend->SetBorderSize(0);
    legend->SetFillStyle(0);
    legend->Draw();

    float xposition_lt = hs->GetXaxis()->GetXmax() *0.03 + hs->GetXaxis()->GetXmin();
    TLatex *lt = new TLatex(xposition_lt, hs->GetMaximum()*0.95,
      "#scale[0.8]{#splitline{CMS #bf{#it{Open Data}}}{#bf{11.5 fb^{-1} (8 TeV)}}}");
    lt->Draw("Same");

    // Save the plot in the correct directory
    std::string name = "./Distribution_plots/" + variable + ".pdf";
    c->SaveAs(name.c_str());
    c->Update();
}
```

plotting.cpp

Plotting code

- Salvo tutti i plot nel folder “Distribution Plots”

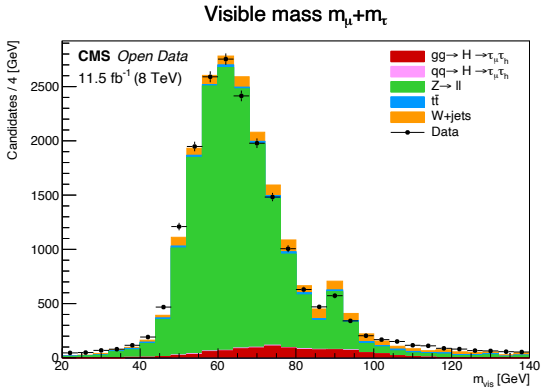
```
template <typename T>
void plotting(T samples, std::string variable, const char *title, std::string xlabel_,
std::string unit, int bins, float lim_left, float lim_right) {
    std::cout << "\tPlotting: " << variable << std::endl;

    TCanvas *c = new TCanvas("c","c", 1000, 700);
    gStyle->SetOptStat(0);
    gStyle->SetTitleFontSize(0.05);
    // Define six 1D histograms for each channel
    auto h1 = samples.at("GluGluToHToTauTau_selected")
              .Histo1D({"", "", bins, lim_left, lim_right}, variable, "weight");
    auto h2 = samples.at("VBF_HToTauTau_selected")
              .Histo1D({"", "", bins, lim_left, lim_right}, variable, "weight");
    auto h3 = samples.at("DYJetsToLL_selected")
              .Histo1D({"", "", bins, lim_left, lim_right}, variable, "weight");
    auto h4 = samples.at("TTbar_selected")
              .Histo1D({"", "", bins, lim_left, lim_right}, variable, "weight");
    auto h5 = samples.at("W*JetsToLNu_selected")
              .Histo1D({"", "", bins, lim_left, lim_right}, variable, "weight");
    auto h6 = samples.at("Run2012*_TauPlusX_selected")
              .Histo1D({"", "", bins, lim_left, lim_right}, variable);

    std::vector<decltype(h1)> hist = {h1, h2, h3, h4, h5};
    std::vector<decltype(kRed+1)> color =
                          {kRed+1, kMagenta-9, kGreen-3, kAzure+7, kOrange-3};
    // Assign the colours to each histogram
    for (int i=0; i<hist.size(); i++) {
        hist[i]->SetLineColor(color[i]);
        hist[i]->SetFillColor(color[i]);
    }
    // Create a stacked histrogram with MC simulation samples
    THStack *hs = new THStack("hs", "");
    for (auto &h : hist) {
        hs->Add(h.GetPtr());
    }
    hs->Draw("HISTO");
    // Draw the Data with black points and error bars
    h6->SetMarkerStyle(kFullCircle);
    h6->SetMarkerColor(kBlack);
    h6->SetLineColor(kBlack);
    h6->DrawClone("ESAME");

    // Set the x-y axis labels and calculate the binning
    auto binning = (lim_right - lim_left)/bins;
    std::stringstream ss;
    ss.precision(2);
    ss << binning;
    std::string y_label = "Candidates / " + ss.str() + " " + unit;
    std::string x_label = xlabel_ + " " + unit;
    hs->GetXaxis()->SetTitle(x_label.c_str());
    hs->GetYaxis()->SetTitle(y_label.c_str());
    hs->SetTitle(title);

    // Draw a Tlegend for Real Data and MC samples
    TLegend *legend = new TLegend(0.65, 0.65, 0.98, 0.88);
    legend->AddEntry(h1.GetPtr(),"gg#rightarrow H #rightarrow#tau_{#mu}#tau_{h}","f");
    legend->AddEntry(h2.GetPtr(),"qq#rightarrow H #rightarrow#tau_{#mu}#tau_{h}","f");
    legend->AddEntry(h3.GetPtr(),"Z#rightarrow ll","f");
    legend->AddEntry(h4.GetPtr(),"t#bar{t}", "f");
    legend->AddEntry(h5.GetPtr(),"W+jets", "f");
    legend->AddEntry(h6.GetPtr(), "Data");
    legend->SetBorderSize(0);
    legend->SetFillStyle(0);
    legend->Draw();

    float xposition_lt = hs->GetXaxis()->GetXmax() *0.03 + hs->GetXaxis()->GetXmin();
    TLatex *lt = new TLatex(xposition_lt, hs->GetMaximum()*0.95,
      "#scale[0.8]{#splitline{CMS #bf{#it{Open Data}}}{#bf{11.5 fb^{-1} (8 TeV)}}}");
    lt->Draw("Same");

    // Save the plot in the correct directory
    std::string name = "./Distribution_plots/" + variable + ".pdf";
    c->SaveAs(name.c_str());
    c->Update();
}
```

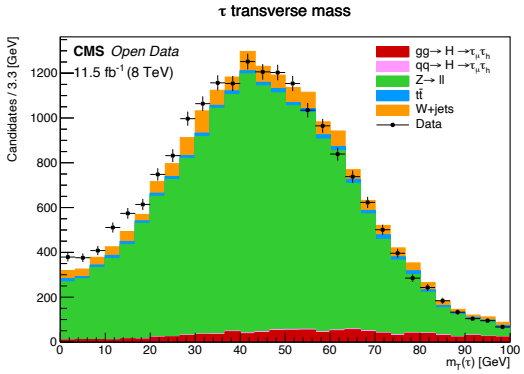
plotting.cpp

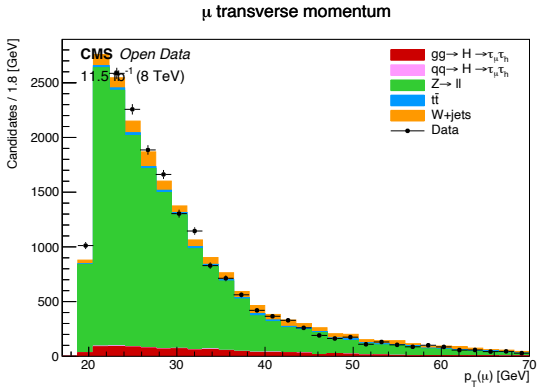
Plotting code

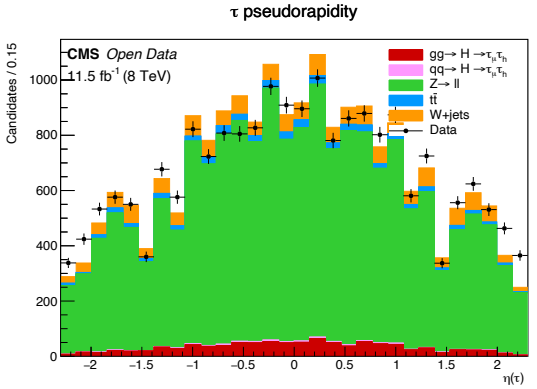
```
domenico@MacBook-Pro-di-domenico ROOT_workspace % ./plotting
    Processing: GluGluToHToTauTau_selected
    Number of events: 2034
    Processing: VBF_HToTauTau_selected
    Number of events: 2514
    Processing: DYJetsToLL_selected
    Number of events: 12927
    Processing: TTbar_selected
    Number of events: 1344
    Processing: W*JetsToLNu_selected
    Number of events: 1410
    Processing: Run2012*_TauPlusX_selected
    Number of events: 20187
******************** Plotting distribution **********************
    Plotting: m_vis
    Plotting: pt_vis
    Plotting: tau_pt
    Plotting: muon_pt
    Plotting: muon_eta
    Plotting: tau_eta
    Plotting: muon_phi
    Plotting: tau_phi
    Plotting: muon_iso
    Plotting: tau_iso
    Plotting: muon_charge
    Plotting: tau_charge
    Plotting: MET_pt
    Plotting: MET_phi
    Plotting: muon_m
    Plotting: tau_m
    Plotting: mt_mu
    Plotting: mt_tau
    Plotting: mt_mu
    Plotting: jpt_1
    Plotting: jpt_2
    Plotting: jeta_1
    Plotting: jeta_2
    Plotting: jphi_1
    Plotting: jphi_2
    Plotting: jm_1
    Plotting: jm_2
    Plotting: jbtag_1
    Plotting: jbtag_2
    Plotting: PV_npvs
    Plotting: nGoodJets
    Plotting: jj_m
    Plotting: jj_pt
    Plotting: jj_delta
```

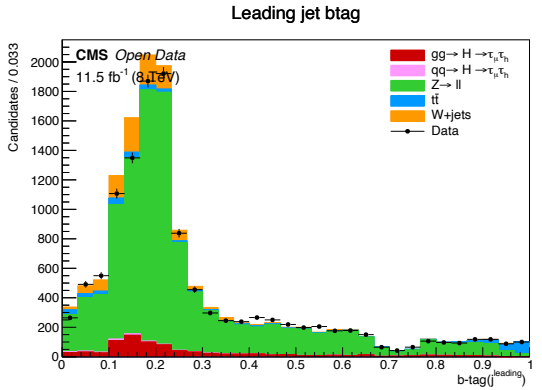
Distribution_plots

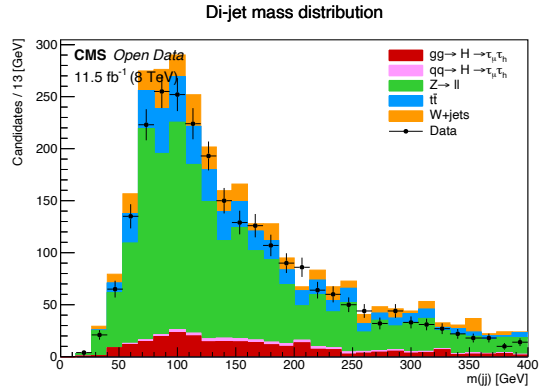












Algoritmi ML per la classificazione segnale/fondo (Python)

Seconda Parte

Machine Learning: classificazione segnale/fondo (1)

Main function

```python
if __name__ == "__main__":
    ML_dict = { ... }

    # Create a new directory contains ML plots
    NAME_DIR = '/ML_plots'
    PATH = os.getcwd()
    if not os.path.exists(PATH + NAME_DIR):
        os.mkdir(PATH + NAME_DIR)
        print(f">>> ..{NAME_DIR} directory has been created at {PATH}")
    else:
        print(f">>> ..{NAME_DIR} folder is already present at {PATH}")

    # Ask to user what signal channel would like to use
    print("Which file would you process? [ggH o VBF]")
    CHOICE = input()
    print(f"****************** {CHOICE} is our signal channel ******************")
    if (CHOICE != "ggH") and (CHOICE != "VBF"):
        print("This CHOICE isn't present")
        sys.exit()

    events = read_root(ML_dict[CHOICE]['complete_name'])
    NUM_VARS = len(ML_dict[CHOICE]['ML_VARS'])
    # Remove undefined variables
    for i in range(NUM_VARS):
        events = events[(events[VARS[i]] > -999)]

    print(f"Number of signal (1) and background events (0):\n{events.event.value_counts()}")
```

```python
ML_dict = {
        'ggH':
            {
                'complete_name': 'GluGluToHToTauTau',
                'ML_VARS': ["nGoodJets", "PV_npvs", "MET_pt", "MET_phi", "pt_vis",
                            "muon_pt", "muon_eta", "muon_phi", "muon_iso",
                            "tau_pt", "tau_eta", "tau_phi"],
                'number_input': 10,
                'N_EPOCHS': 150,
            },
        'VBF':
            {
                'complete_name': 'VBF_HToTauTau',
                'ML_VARS': ["nGoodJets", "PV_npvs", "MET_phi", "m_vis", "pt_vis",
                            "muon_pt", "muon_eta", "muon_phi", "muon_iso",
                            "tau_pt", "tau_eta", "tau_phi", "tau_iso",
                            "jbtag_1", "jj_delta"],
                'number_input': 15,
                'N_EPOCHS': 100
            }
    }
```

ML_Higgs.py

Machine Learning: classificazione segnale/fondo (2)

Read root function

```python
def read_root(file_name):
    """ doc """
    PATH = "ROOT_workspace/"
    df = {}
    for file in MLfiles:
        Events = uproot.open(PATH + file + "_selected.root" + ":Events")
        df[file] = pd.DataFrame(Events.arrays(VARS, library="np"), columns=VARS)
        print(f"\tProcessing: {file}, Number of events: {len(df[file])}")
        if file == file_name:
            df[file]["event"] = 1.
        else:
            df[file]["event"] = 0.
    return pd.concat(df.values(), ignore_index=True)
```

```
***************** ggH is our signal channel *****************
    Processing: GluGluToHToTauTau, Number of events: 4492
    Processing: VBF_HToTauTau, Number of events: 5326
    Processing: DYJetsToLL, Number of events: 35099
    Processing: TTbar, Number of events: 13630
    Processing: W1JetsToLNu, Number of events: 7042
    Processing: W2JetsToLNu, Number of events: 11427
    Processing: W3JetsToLNu, Number of events: 6327
Number of signal (1) and background events (0):
0.0    78839
1.0     4492
Name: event, dtype: int64
```

ML_Higgs.py, Documentation

Machine Learning: classificazione segnale/fondo (3)

Main function

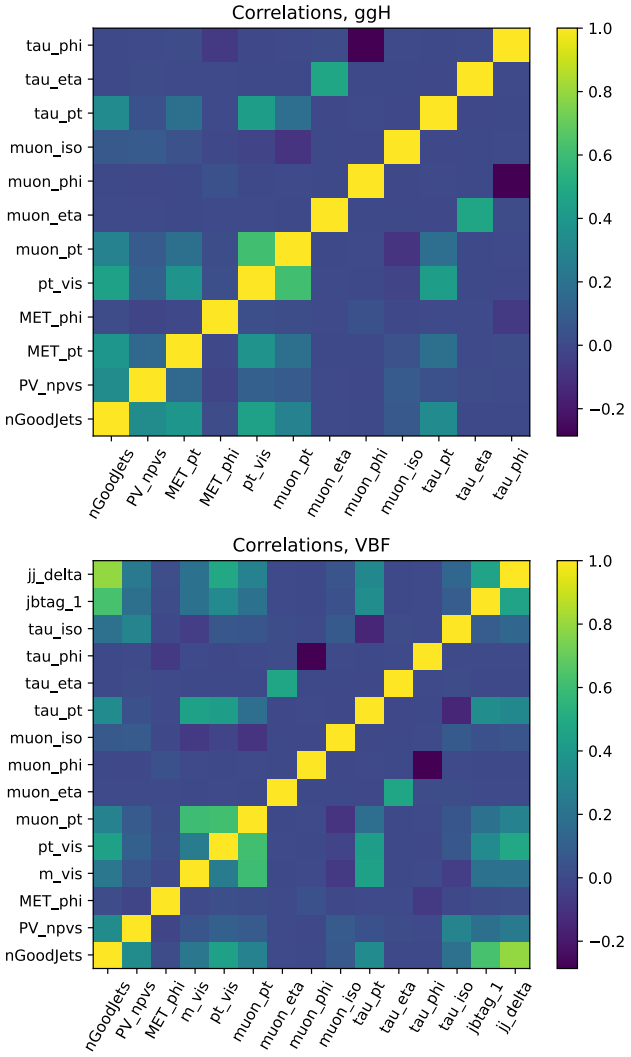
```
    # Renormalization of events
    sig = events["event"] == 1.
    bgk = events["event"] == 0.
    weight_sig_sum = events["weight"][sig].sum(axis=0)
    weight_bkg_sum = events["weight"][bgk].sum(axis=0)
    events["weight"][sig] = events["weight"][sig] / weight_sig_sum
    events["weight"][bgk] = events["weight"][bgk] / weight_bkg_sum

    # Plot correlation matrix of ML variables
    correlations(CHOICE, events[ML_dict[CHOICE]["ML_VARS"]])

    # Generate features' matrix
    features = events.filter(ML_dict[CHOICE]["ML_VARS"])
    X = np.asarray(features.values).astype(np.float32)
    # Generate target vector
    target = events.filter(['event'])
    y = np.asarray(target.values).astype(np.float32)
    # Generate event weights vector
    weight = events.filter(['weight'])
    W = np.asarray(weight.values).astype(np.float32)

    # Split into training and testing dataset
    X_train, X_test, y_train, y_test, W_train, W_test = train_test_split(X, y, W,
                                                                         test_size=0.3, shuffle=True)
    print(f'Number of events for training phase: {len(X_train)}')
    print(f'Number of events for test phase: {len(X_test)}')

    # Perform features scaling with StandardScaler
    SC = StandardScaler()
    X_train = SC.fit_transform(X_train)
    X_test = SC.transform(X_test)
```

ML_Higgs.py

Machine Learning: classificazione segnale/fondo (4)

Correlation function

```python
def correlations(channel_name, data):
    """ doc """
    plt.figure()
    corrmat = data.corr()
    heatmap1 = plt.pcolor(corrmat)  # get heatmap
    plt.colorbar(heatmap1)  # plot colorbar

    plt.title(f'Correlations, {channel_name}')  # set title
    x_variables = corrmat.columns.values  # get variables from data columns

    plt.xticks(np.arange(len(x_variables)) + 0.5, x_variables, rotation=60)  # x-tick for each label
    plt.yticks(np.arange(len(x_variables)) + 0.5, x_variables)  # y-tick for each label
    plt.savefig(f'{NAME_DIR}/correlation_{channel_name}.pdf', format='pdf', bbox_inches='tight')
    plt.clf()
```



Documentation, plot_functions.py, correlation plots

ANN Machine Learning: classificazione segnale/fondo (1)

Main function, ANN

```python
print("\n******************** ARTIFICIAL NEURAL NETWORK ******************")
print(f"\tML variables ({NUM_VARS} features) are:")
for var in ML_dict[CHOICE]['ML_VARS']:
    print(f"\t{var}")

# Create Artificial Neural Network (with 3 hidden layers)
input_layer = Input(shape=(NUM_VARS,), name='input')
hidden = Dense(NUM_VARS * ML_dict[CHOICE]['number_input'], name='hidden1',
               activation='selu')(input_layer)
hidden = Dropout(rate=0.1)(hidden)
hidden = Dense(NUM_VARS * 2, name='hidden2', activation='selu')(hidden)
hidden = Dropout(rate=0.5)(hidden)
hidden = Dense(NUM_VARS, name='hidden3', activation='selu')(hidden)
hidden = Dropout(rate=0.1)(hidden)
output_layer = Dense(1, name='output', activation='sigmoid')(hidden)
# Initialising ANN model
ANN = Model(inputs=input_layer, outputs=output_layer, name='ANN')
ANN.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'],
            weighted_metrics=['accuracy'])  # Compiling ANN
ANN.summary()  # Print on screen the model summary

# Fit the ANN model
HISTORY = ANN.fit(X_train, y_train, sample_weight=W_train, batch_size=20,
                  epochs=ML_dict[CHOICE]["N_EPOCHS"], verbose=1, validation_split=0.3)

plotting_loss(CHOICE, HISTORY.history['loss'], HISTORY.history['val_loss'])
plotting_accuracy(CHOICE, HISTORY.history['accuracy'], HISTORY.history['val_accuracy'])
```

ggH

```
Model: "ANN"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input (InputLayer)          [(None, 12)]              0

 hidden1 (Dense)             (None, 120)               1560

 dropout (Dropout)           (None, 120)               0

 hidden2 (Dense)             (None, 24)                2904

 dropout_1 (Dropout)         (None, 24)                0

 hidden3 (Dense)             (None, 12)                300

 dropout_2 (Dropout)         (None, 12)                0

 output (Dense)              (None, 1)                 13

=================================================================
Total params: 4,777
Trainable params: 4,777
Non-trainable params: 0
_________________________________________________________________
```


ML_Higgs.py

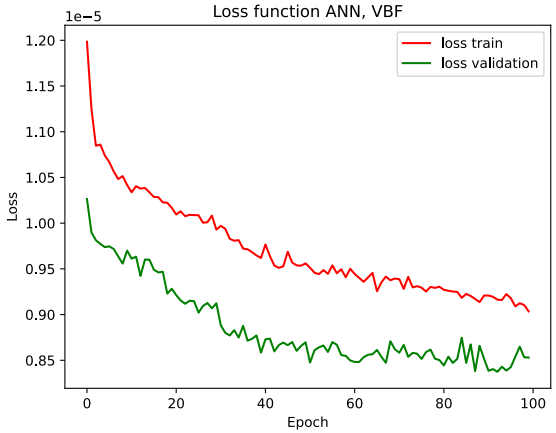
ANN Machine Learning: classificazione segnale/fondo (2)

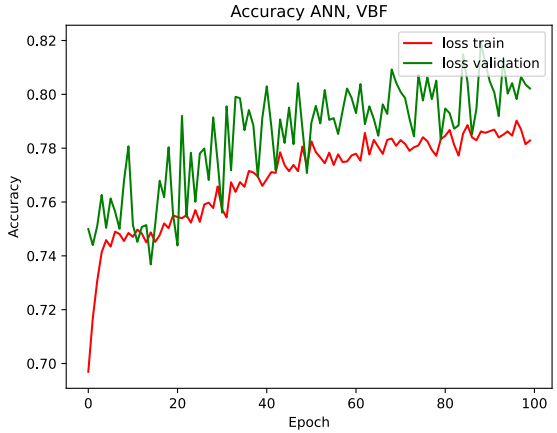
Loss and accuracy plotting functions

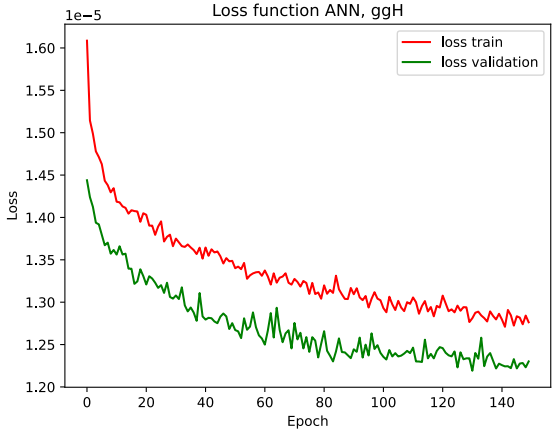
```python
def plotting_loss(channel_name, loss, val_loss):
    """ doc """
    plt.figure()
    plt.plot(loss, label='loss train', color='red')
    plt.plot(val_loss, label='loss validation', color='green')
    plt.title(f'Loss function ANN, {channel_name}')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend(loc='upper right')
    plt.savefig(f'{NAME_DIR}/Loss_ANN_{channel_name}.pdf',
                format='pdf')
    plt.clf()

def plotting_accuracy(channel_name, accuracy, val_accuracy):
    """ doc """
    plt.figure()
    plt.plot(accuracy, label='loss train', color='red')
    plt.plot(val_accuracy, label='loss validation', color='green')
    plt.title(f'Accuracy ANN, {channel_name}')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend(loc='upper right')
    plt.savefig(f'{NAME_DIR}/Accuracy_ANN_{channel_name}.pdf',
                format='pdf')
    plt.clf()
```

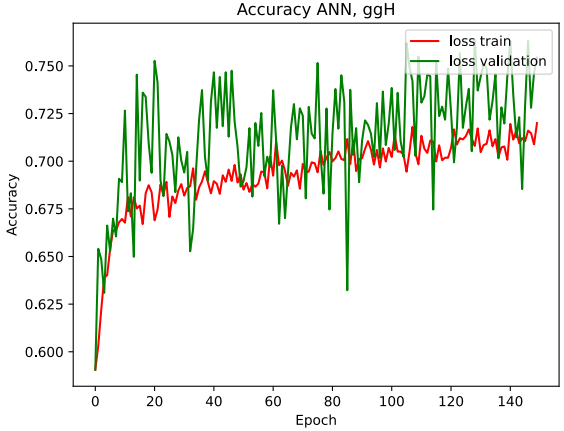

ML_Higgs.py, Documentation, plot_functions.py

ANN Machine Learning: classificazione segnale/fondo (3)







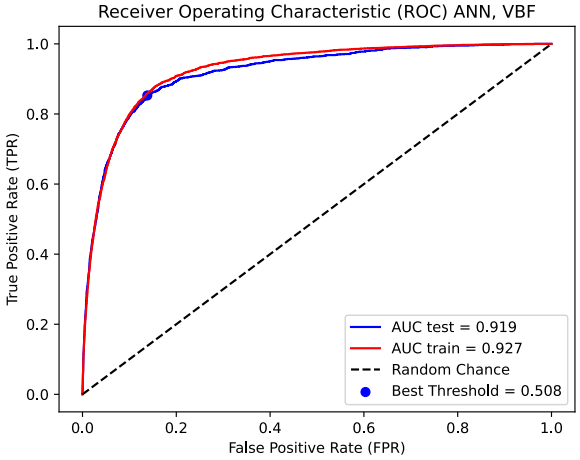


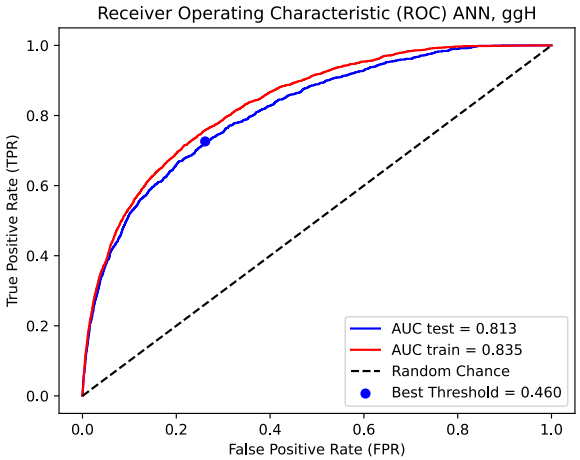
ML_Higgs.py, plot_functions.py

ANN Machine Learning: classificazione segnale/fondo (4)

Main function, ROC curves

```
    # Get ANN model event predictions
    y_prediction_test = ANN.predict(X_test)
    y_prediction_train = ANN.predict(X_train)

    # Get False Positive Rate (FPR), True Positive Rate (TPR) and Thresholds/Cut
    # on the ANN's score
    fpr_test, tpr_test, thresholds_test = roc_curve(y_true=y_test,
                                                    y_score=y_prediction_test,
                                                    sample_weight=W_test)
    fpr_train, tpr_train, thresholds_train = roc_curve(y_true=y_train,
                                                       y_score=y_prediction_train,
                                                       sample_weight=W_train)

    # Optimal Threshold for ROC Curve with g-mean
    index_g = g_mean_threshold(tpr_test, fpr_test, thresholds_test)


    # Plot the ANN ROC curve on the test and training datasets
    # with plotting_ROC function implemented in plot_function.py script
    roc_auc_test = auc(fpr_test, tpr_test)
    roc_auc_train = auc(fpr_train, tpr_train)
    plotting_ROC(f'ANN_{CHOICE}', fpr_test, tpr_test, fpr_train, tpr_train,
                 thresholds_test, index_g, roc_auc_test, roc_auc_train)
```

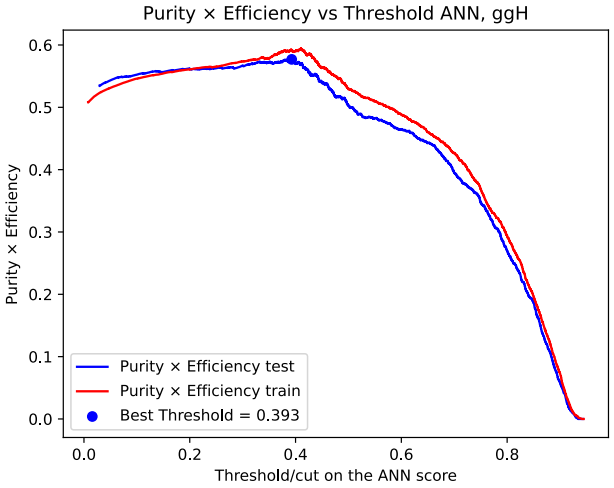


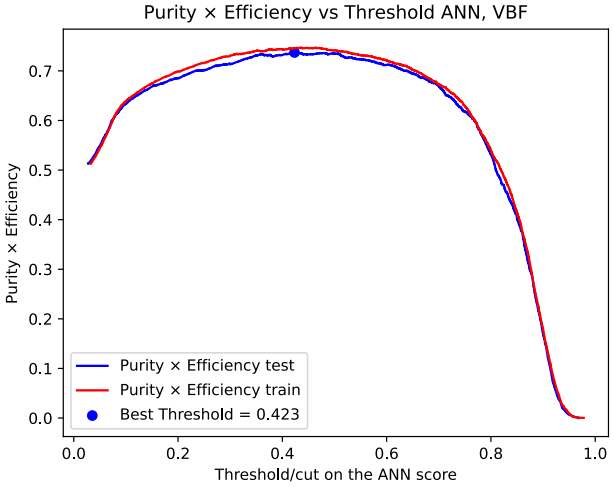


ML_Higgs.py, plot_functions.py, ROC curves plots

ANN Machine Learning: classificazione segnale/fondo (5)

Main function, P-E curves






```python
# Get precision (purity) and recall (signal efficiency) with precision_recall_curve method.
purity_test, recall_test, t_test = precision_recall_curve(y_true=y_test,
                                                          probas_pred=y_prediction_test,
                                                          sample_weight=W_test)
purity_train, recall_train, t_train = precision_recall_curve(y_true=y_train,
                                                             probas_pred=y_prediction_train,
                                                             sample_weight=W_train)

# Optimal threshold for Precision-Recall Curve (PR) with simplify f-score
index_f = fscore_threshold(purity_test[:-1], recall_test[:-1], t_test)

plotting_purity_vs_efficiency(f'ANN_{CHOICE}', t_test, t_train, recall_test,
                              recall_train, purity_test, purity_train, index_f)
```


ML_Higgs.py, plot_functions.py

ANN Machine Learning: classificazione segnale/fondo (6)

Plotting function, ROC and P-E

```python
def plotting_ROC(channel_name, fpr_test, tpr_test, fpr_train, tpr_train, thresholds,
                 ix, roc_auc_test, roc_auc_train):
    """ doc """
    names = channel_name.split("_")
    plt.figure()
    plt.plot(fpr_test, tpr_test, color='blue', label=f'AUC test = {roc_auc_test:.3f}')
    plt.plot(fpr_train, tpr_train, color='red', label=f'AUC train = {roc_auc_train:.3f}')
    plt.plot([0, 1], [0, 1], linestyle='--', color='k', label='Random Chance')
    plt.scatter(fpr_test[ix], tpr_test[ix], marker='o', color='blue',
                label=f'Best Threshold = {thresholds[ix]:.3f}')
    plt.xlabel('False Positive Rate (FPR)')
    plt.ylabel('True Positive Rate (TPR)')
    plt.title(f'Receiver Operating Characteristic (ROC) {names[0]}, {names[1]}')
    plt.legend(loc='lower right')
    plt.savefig(f'{NAME_DIR}/ROC_{channel_name}.pdf', format='pdf')
    plt.clf()

def plotting_purity_vs_efficiency(channel_name, t_test, t_train, recall_test, recall_train,
                                  purity_test, purity_train, ix):
    """ doc """
    names = channel_name.split("_")
    plt.figure()
    plt.plot(t_test, recall_test[:-1] * purity_test[:-1], color='blue',
             label=r'Purity $\times$ Efficiency test')
    plt.plot(t_train, recall_train[:-1] * purity_train[:-1], color='red',
             label=r'Purity $\times$ Efficiency train')
    plt.scatter(t_test[ix], recall_test[ix] * purity_test[ix], marker='o', color='blue',
                label=f'Best Threshold = {t_test[ix]:.3f}')
    plt.xlabel(f'Threshold/cut on the {names[0]} score')
    plt.ylabel(r'Purity $\times$ Efficiency')
    plt.title(fr'Purity $\times$ Efficiency vs Threshold {names[0]}, {names[1]}')
    plt.legend(loc='lower left')
    plt.savefig(f'{NAME_DIR}/metrics_PR_{channel_name}.pdf', format='pdf')
    plt.clf()
```


Documentation, plot_functions.py

ANN Machine Learning: classificazione segnale/fondo (7)

Main function, g-mean and simply f-score

```python
def g_mean_threshold(tpr, fpr, thresholds):
    """ doc """
    g_means = np.sqrt(tpr * (1 - fpr))
    index = np.argmax(g_means)  # locate the index of the largest g-mean
    print(f'Best Threshold={thresholds[index]:.3f}, \
    G-Mean={g_means[index]:.3f}')
    return index

def fscore_threshold(purity, recall, thresholds):
    """ doc """
    fscore = purity[:-1] * recall[:-1]
    index = np.argmax(fscore) # locate the index of the largest f-score
    print(f'Best Threshold={thresholds[index]:.3f}, \
    F-Score simplify={fscore[index]:.3f}')
    return index
```


ML_Higgs.py, Documentation

ANN Machine Learning: classificazione segnale/fondo (8)

Main function, confusion matrix and res.

```python
    # Threshold selection according to signal channel
    cut_dnn = 0.
    if CHOICE == "ggH":
        cut_dnn = thresholds_test[index_g]
    else:
        cut_dnn = t_test[index_f]

    # Plotting output score distribution
    plotting_output_score(f'ANN_{CHOICE}', y_train, y_prediction_train, y_test,
                          y_prediction_test, cut_dnn)

    # Transform predictions into an array of 0, 1
    filter_test_sig = y_prediction_test >= cut_dnn  # classify as signal
    filter_test_bkg = y_prediction_test < cut_dnn  # classify as background
    y_prediction_test[filter_test_sig] = 1
    y_prediction_test[filter_test_bkg] = 0

    # Metrics values for the ANN algorithm having fixed an ANN score threshold
    accuracy = accuracy_score(y_test[:, 0], y_prediction_test[:, 0], sample_weight=W_test[:, 0])
    precision = precision_score(y_test[:, 0], y_prediction_test[:, 0], sample_weight=W_test[:, 0])
    recall = recall_score(y_test[:, 0], y_prediction_test[:, 0], sample_weight=W_test[:, 0])
    print(f'Threshold on the ANN output : {cut_dnn:.3f}')
    print(f'ANN Test Accuracy: {accuracy:.3f}')
    print(f'ANN Test Precision/Purity: {precision:.3f}')
    print(f'ANN Test Sensitivity/Recall/TPR/Signal Efficiency: {recall:.3f}')
    plotting_confusion_matrix(f'ANN_{CHOICE}', y_test[:, 0], y_prediction_test[:, 0], W_test[:, 0])
```

ggH

```
...
Epoch 150/150
2042/2042 [==============================] - 5s 3ms/step -
loss: 1.2763e-05 - accuracy: 0.7200 - weighted_accuracy: 0.7248 -
val_loss: 1.2301e-05 - val_accuracy: 0.7544 - val_weighted_accuracy: 0.7356

Best Threshold=0.460, G-Mean=0.732
Best Threshold=0.393, F-Score simplify=0.577
Threshold on the ANN output : 0.460
ANN Test Accuracy: 0.732
ANN Test Precision/Purity: 0.741
ANN Test Sensitivity/Recall/TPR/Signal Efficiency: 0.726
```

VBF

```
...
Epoch 100/100
2042/2042 [==============================] - 13s 6ms/step -
loss: 9.0350e-06 - accuracy: 0.7829 - weighted_accuracy: 0.8481 -
val_loss: 8.5298e-06 - val_accuracy: 0.8022 - val_weighted_accuracy: 0.8594

Best Threshold=0.508, G-Mean=0.857
Best Threshold=0.423, F-Score simplify=0.737
Threshold on the ANN output : 0.423
ANN Test Accuracy: 0.854
ANN Test Precision/Purity: 0.840
ANN Test Sensitivity/Recall/TPR/Signal Efficiency: 0.877
```


ML_Higgs.py

ANN Machine Learning: classificazione segnale/fondo (9)

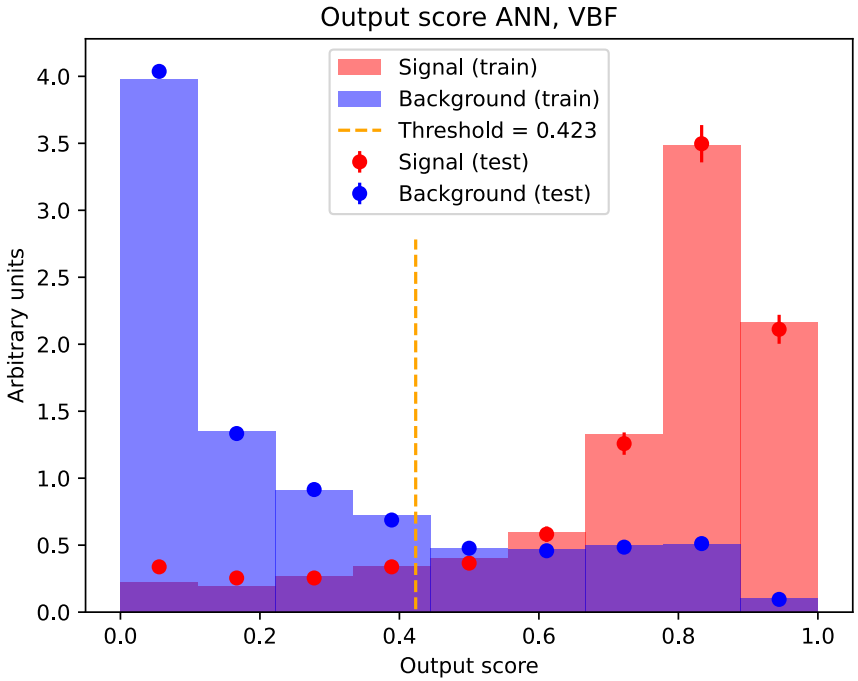
Plotting function, output-score

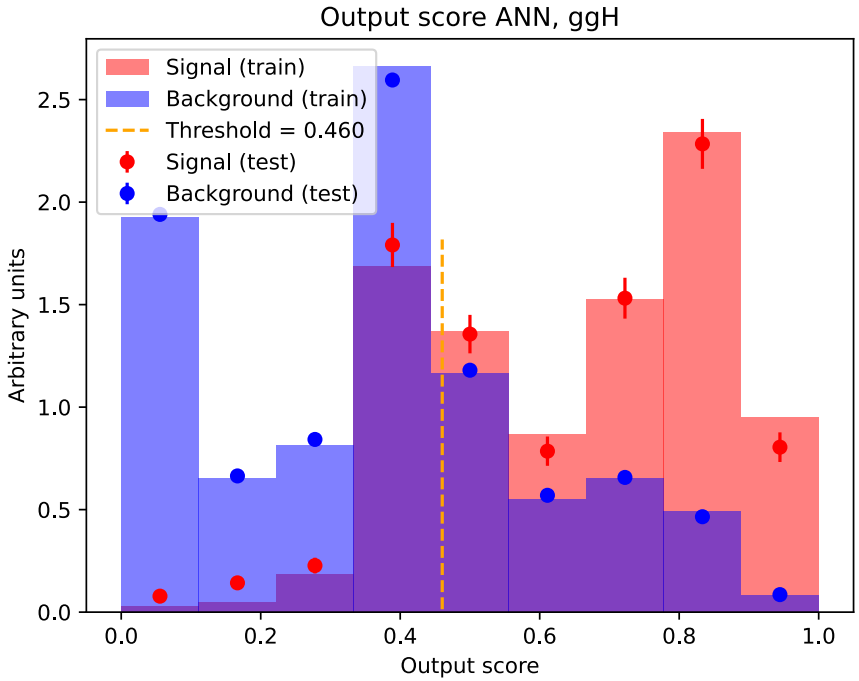
```python
def plotting_output_score(channel_name, y_train, y_prediction_train, y_test, y_prediction_test, cut_dnn):
    """ doc """
    names = channel_name.split("_")
    plt.figure()
    xx = np.linspace(0.0, 1.0, 10)
    plt.hist(y_prediction_train[y_train == 1], bins=xx, density=1, label='Signal (train)',
             alpha=0.5, color='red')
    plt.hist(y_prediction_train[y_train == 0], bins=xx, density=1, label='Background (train)',
             alpha=0.5, color='blue')

    hist_test_sig, bin_edges = np.histogram(y_prediction_test[y_test == 1], bins=xx, density=True)
    hist_test_bgk, _ = np.histogram(y_prediction_test[y_test == 0], bins=xx, density=True)

    center = (bin_edges[:-1] + bin_edges[1:]) / 2  # bin centres

    scale = len(y_prediction_test[y_test == 1]) / sum(hist_test_sig)
    err_sig = np.sqrt(hist_test_sig * scale) / scale
    plt.errorbar(center, hist_test_sig, yerr=err_sig, label='Signal (test)', c='red', fmt='o')

    scale = len(y_prediction_test[y_test == 0]) / sum(hist_test_bgk)
    err_bgk = np.sqrt(hist_test_bgk * scale) / scale
    plt.errorbar(center, hist_test_bgk, yerr=err_bgk, label='Background (test)', c='blue', fmt='o')
    plt.axvline(x=cut_dnn, ymin=0.005, ymax=0.65, color='orange', label=f'Threshold = {cut_dnn:.3f}',
                linestyle='--')
    plt.title(f'Output score {names[0]}, {names[1]}')
    plt.xlabel('Output score')
    plt.ylabel('Arbitrary units')
    plt.legend()
    plt.savefig(f'{NAME_DIR}/{channel_name}_output_score.pdf', format='pdf')
    plt.clf()
```


Documentation, plot_functions.py

ANN Machine Learning: classificazione segnale/fondo (10)

Output score ANN



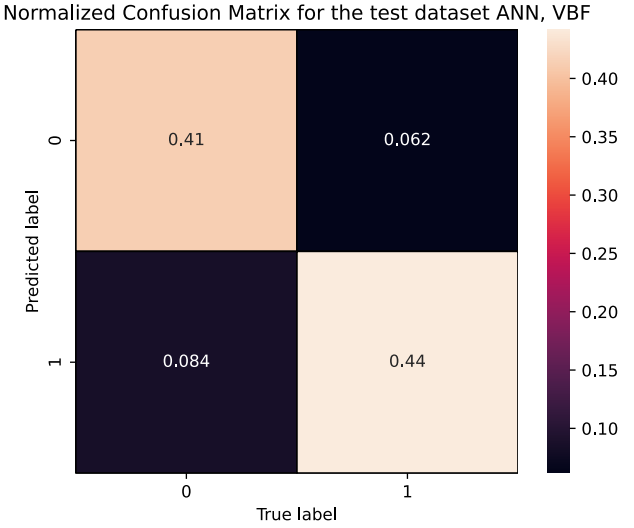


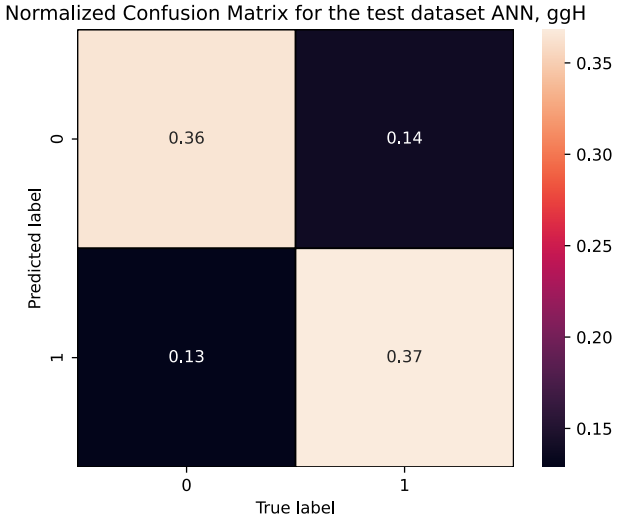
Output score plots

ANN Machine Learning: classificazione segnale/fondo (11)

Plotting function, confusion matrix

```python
def plotting_confusion_matrix(channel_name, y_test, y_prediction_test, w_test):
    """ doc """
    names = channel_name.split("_")
    plt.figure()
    mat_test = confusion_matrix(y_test, y_prediction_test, sample_weight=w_test, normalize='all')
    sns.heatmap(mat_test.T, square=True, annot=True, cbar=True, linewidths=1, linecolor='black')
    plt.xlabel('True label')
    plt.ylabel('Predicted label')
    plt.title(f'Normalized Confusion Matrix for the test dataset {names[0]}, {names[1]}')
    plt.savefig(f'{NAME_DIR}/confusion_matrix_{channel_name}.pdf', format='pdf')
    plt.clf()
```





Documentation,

plot_functions.py

RF Machine Learning: classificazione segnale/fondo (1)

Main function, model e Hyperparameter tuning

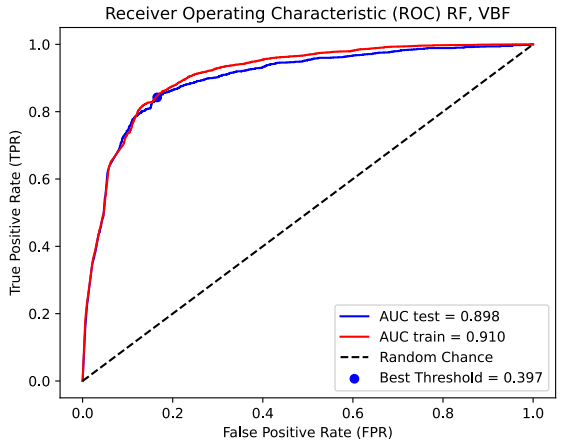
```
print("************************************ RANDOM FOREST ************************************")

    # Uncomment the following lines to perform the tuning of hyper-parameters of the Random Forest Classifier
    # classifier = RandomForestClassifier(random_state=7, verbose=1)
    # grid_param = {
        # 'criterion': ['gini', 'entropy'],
        # 'n_estimators': [300, 500],
        # 'bootstrap': [True, False],
        # 'max_depth': [5, 7],
        # 'max_features': [10, None],
    # }
    # tuner_RF = GridSearchCV(estimator=classifier, param_grid=grid_param, scoring='accuracy', cv=3, n_jobs=-1)
    # tuner_RF.fit(X_train, np.ravel(y_train))
    # print(f"Best parameters: {tuner_RF.best_params_}")
    # print(f"Best metrics score {tuner_RF.best_score_}")

    # Create a RF classifier
    RFC = RandomForestClassifier(n_estimators=500, criterion='gini', verbose=1, max_depth=5,
                                 max_features=None, bootstrap=True, n_jobs=-1)

    # Train the model on the training dataset
    randomforest = RFC.fit(X_train, np.ravel(y_train), np.ravel(W_train))

    # Get RF model event predictions on the test and training dataset.
    # Return a vector of 0s and 1s (the two labels of the classification problem)
    y_prediction_test_rf = RFC.predict(X_test)
    y_prediction_train_rf = RFC.predict(X_train)
```

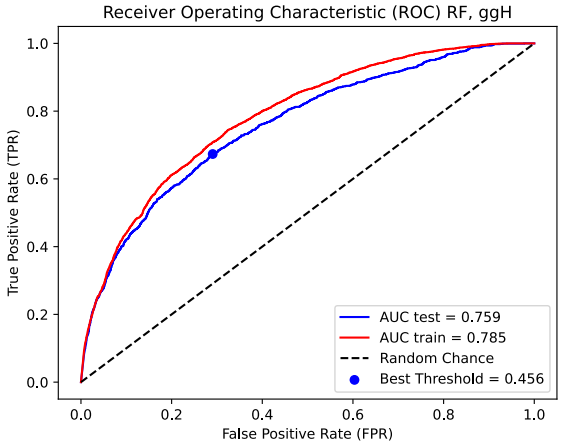

ML_Higgs.py

RF Machine Learning: classificazione segnale/fondo (2)

Main function, ROC curves

```
    # Method predict_proba return a 2D vector.
    # The first column is the probability of the event being 'bkg'
    # and the second is the probability of it being 'sig'
    y_prediction_test_prob = randomforest.predict_proba(X_test)
    y_prediction_train_prob = randomforest.predict_proba(X_train)

    print(f"Accuracy of the RF model: {accuracy_score(y_test, y_prediction_test_rf):.3f} (test dataset)")
    print(f"Accuracy of the RF model: {accuracy_score(y_train, y_prediction_train_rf):.3f} (train dataset)")

    fpr_test_rf, tpr_test_rf, thresholds_test_rf = roc_curve(y_true=y_test,
                                                             y_score=y_prediction_test_prob[:, -1],
                                                             sample_weight=W_test)
    fpr_train_rf, tpr_train_rf, thresholds_train_rf = roc_curve(y_true=y_train,
                                                                y_score=y_prediction_train_prob[:, -1],
                                                                sample_weight=W_train)

    # Calculate area under the ROC curve
    roc_auc_test_rf = auc(fpr_test_rf, tpr_test_rf)
    roc_auc_train_rf = auc(fpr_train_rf, tpr_train_rf)

    index_g_RF = g_mean_threshold(tpr_test_rf, fpr_test_rf, thresholds_test_rf)
    plotting_ROC(f'RF_{CHOICE}', fpr_test_rf, tpr_test_rf, fpr_train_rf, tpr_train_rf,
                 thresholds_test_rf, index_g_RF, roc_auc_test_rf, roc_auc_train_rf)

```

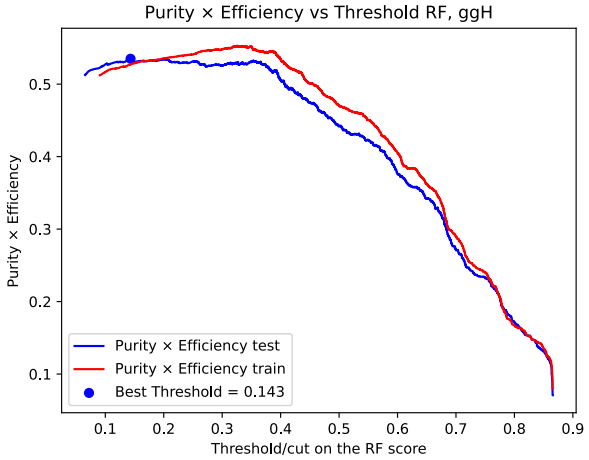


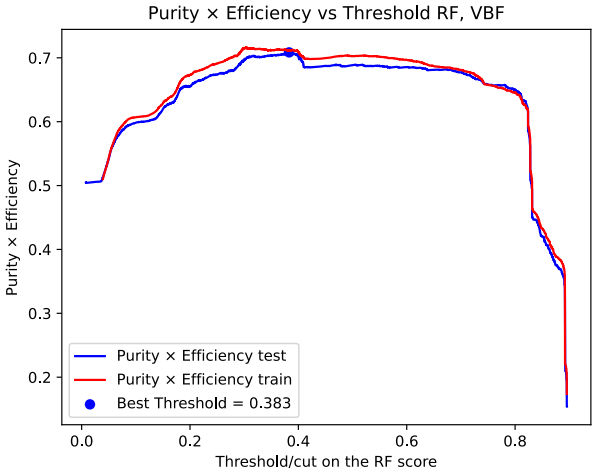


ML_Higgs.py, plotting ROC

RF Machine Learning: classificazione segnale/fondo (3)

Main function, P-E curves



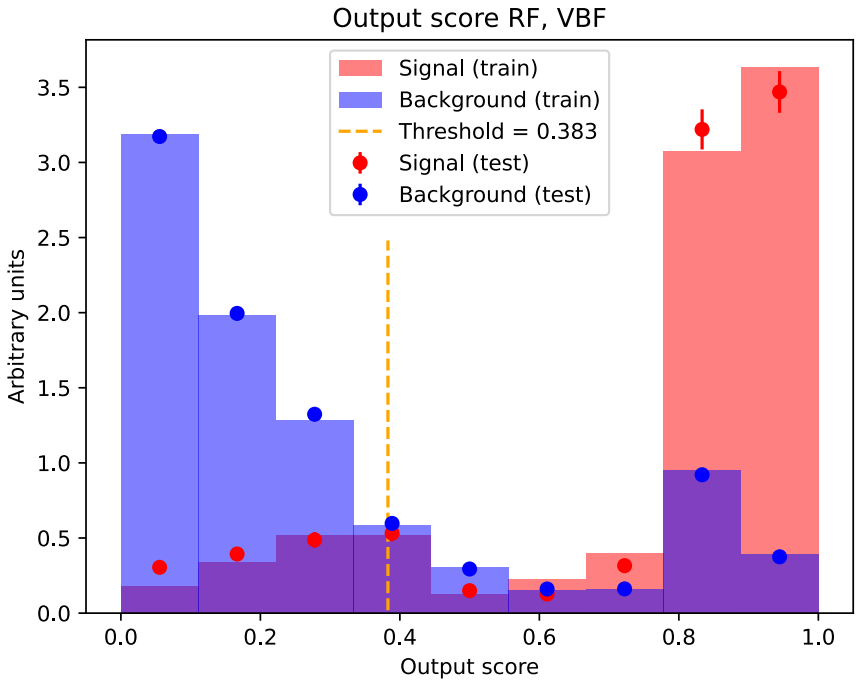


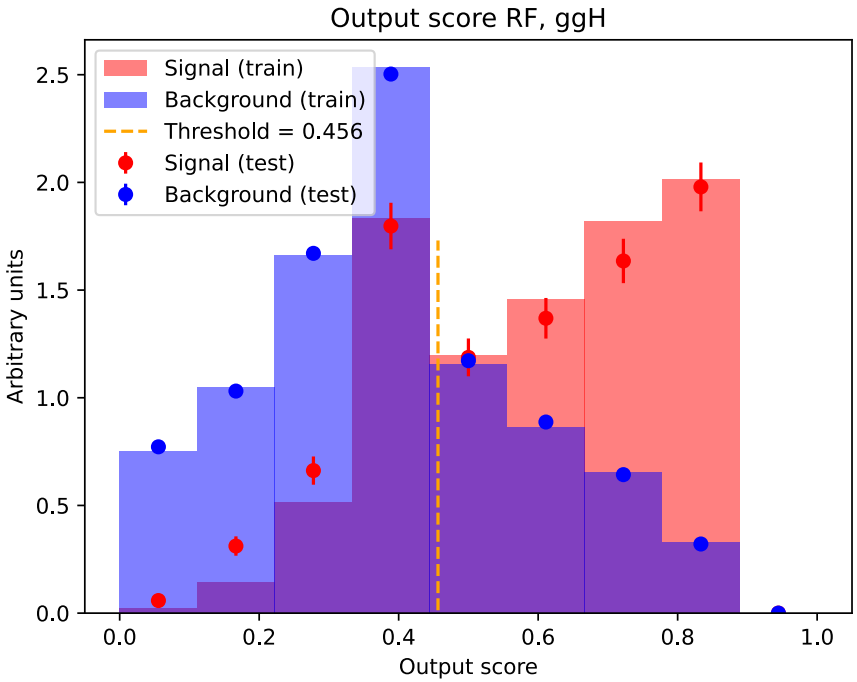

```python
# Get precision and recall
purity_test_rf, recall_test_rf, t_test_rf =
precision_recall_curve(y_true=y_test, probas_pred=y_prediction_test_prob[:, -1], sample_weight=W_test)

purity_train_rf, recall_train_rf, t_train_rf =
precision_recall_curve(y_true=y_train, probas_pred=y_prediction_train_prob[:, -1], sample_weight=W_train)

index_f_RF = fscore_threshold(purity_test_rf, recall_test_rf, t_test_rf)

# Threshold selection according to signal channel
cut_RF = 0.
if CHOICE == "ggH":
    cut_RF = thresholds_test_rf[index_g_RF]
else:
    cut_RF = t_test_rf[index_f_RF]

plotting_purity_vs_efficiency(f'RF_{CHOICE}', t_test_rf, t_train_rf, recall_test_rf,
                              recall_train_rf, purity_test_rf, purity_train_rf, index_f_RF)
plotting_output_score(f'RF_{CHOICE}', y_train[:, 0], y_prediction_train_prob[:, 1],
                      y_test[:, 0], y_prediction_test_prob[:, 1], cut_RF)
```


ML_Higgs.py, P-E curves plot, P-E plotting

RF Machine Learning: classificazione segnale/fondo (4)

Output score





Output score plots

RF Machine Learning: classificazione segnale/fondo (5)

Main function, results and confusion matrix

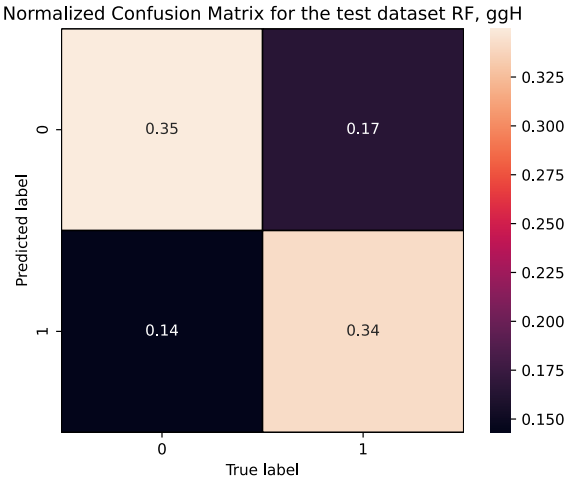
```python
# Transform predictions into an array of 0, 1
filter_test_sig_rf = y_prediction_test_prob[:, -1] >= cut_RF  # classify as signal
filter_test_bkg_rf = y_prediction_test_prob[:, -1] < cut_RF  # classify as background
y_prediction_test_rf[filter_test_sig_rf] = 1
y_prediction_test_rf[filter_test_bkg_rf] = 0

# Metrics values for the ANN algorithm having fixed an ANN score threshold
accuracy_rf = accuracy_score(y_test[:, 0], y_prediction_test_rf, sample_weight=W_test[:, 0])
precision_rf = precision_score(y_test[:, 0], y_prediction_test_rf, sample_weight=W_test[:, 0
recall_rf = recall_score(y_test[:, 0], y_prediction_test_rf, sample_weight=W_test[:, 0])
print(f'Threshold on the RF output : {cut_RF:.3f}')
print(f'RF Test Accuracy: {accuracy_rf:.3f}')
print(f'RF Test Precision/Purity: {precision_rf:.3f}')
print(f'RF Test Sensitivity/Recall/TPR/Signal Efficiency: {recall_rf:.3f}')
plotting_confusion_matrix(f'RF_{CHOICE}', y_test[:, 0], y_prediction_test_rf, W_test[:, 0])
```

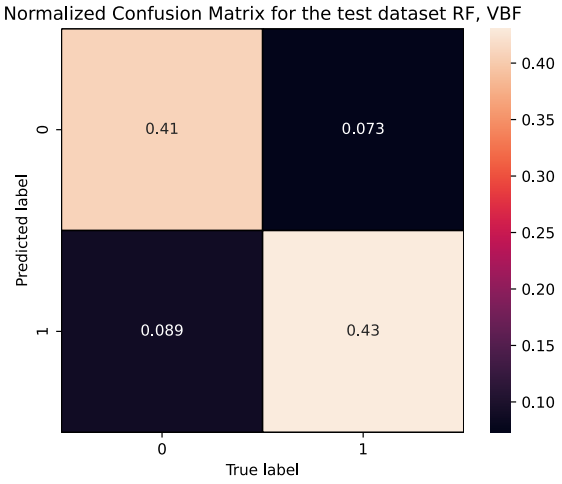
ggH

```
...
[Parallel(n_jobs=4)]: Done 500 out of 500 | elapsed:    3.2s finished

Accuracy of the RF model: 0.724 (test dataset)
Accuracy of the RF model: 0.728 (train dataset)
Best Threshold=0.456, G-Mean=0.691
Best Threshold=0.143, F-Score simplify=0.535
Threshold on the RF output : 0.456
RF Test Accuracy: 0.691
RF Test Precision/Purity: 0.704
RF Test Sensitivity/Recall/TPR/Signal Efficiency: 0.673
```

VBF

```
...
[Parallel(n_jobs=4)]: Done 500 out of 500 | elapsed:    2.9s finished

Accuracy of the RF model: 0.806 (test dataset)
Accuracy of the RF model: 0.804 (train dataset)
Best Threshold=0.397, G-Mean=0.838
Best Threshold=0.383, F-Score simplify=0.708
Threshold on the RF output : 0.383
RF Test Accuracy: 0.838
RF Test Precision/Purity: 0.829
RF Test Sensitivity/Recall/TPR/Signal Efficiency: 0.855
```





ML_Higgs.py,

confusion plots

Machine Learning: classificazione segnale/fondo

Main function & plotting function, variables plots

```
    X_test = SC.inverse_transform(X_test)

    # Plotting of ML variables with models predictions
    for s in ML_dict[CHOICE]['ML_VARS']:
        index_var = ML_dict[CHOICE]['ML_VARS'].index(s)
        plotting_physical_variables(CHOICE, s, X_test[:, index_var], y_test[:, 0], y_prediction_test,
                                    y_prediction_test_rf)
```

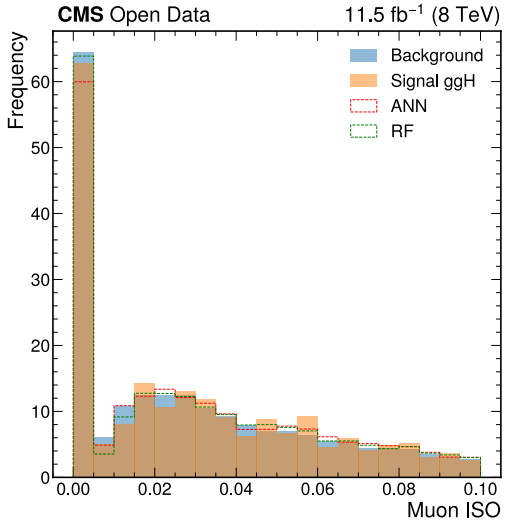
```
def plotting_physical_variables(CHOICE, s, feature, y_test, y_prediction_test, y_prediction_test_rf):
    """ doc """
    plt.style.use(hep.style.CMS)  # or ATLAS/LHCb2
    fig = plt.figure()
    plt.hist(feature[y_test == 0], label='Background', density=1,
             bins=ML_infoplot.plot[s]['bins'], alpha=0.5, linestyle='-')
    plt.hist(feature[y_test == 1], label=f'Signal {CHOICE}', density=1,
             bins=ML_infoplot.plot[s]['bins'], alpha=0.5, linestyle='-')
    plt.hist(feature[y_prediction_test[:, 0] == 1], histtype='step', label='ANN', density=1,
             bins=ML_infoplot.plot[s]['bins'], color='red', linestyle='--')
    plt.hist(feature[y_prediction_test_rf == 1], histtype='step', label='RF', density=1,
             bins=ML_infoplot.plot[s]['bins'], color='green', linestyle='--')
    plt.ylabel(ML_infoplot.plot[s]['ylabel'])
    plt.xlabel(ML_infoplot.plot[s]['xlabel'])
    plt.legend()
    plt.text(0.01, 1.02, r'$\bf{CMS}$ Open Data', transform=plt.gca().transAxes)
    plt.text(0.65, 1.02, r'11.5 $fb^{-1}$ (8 TeV)', transform=plt.gca().transAxes)
    plt.savefig(f'{NAME_DIR}/{s}_{CHOICE}.pdf', format='pdf')
    plt.close(fig)
```

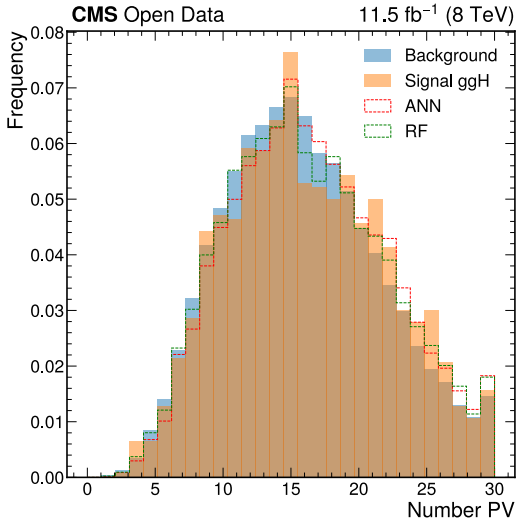
ML_Higgs.py,

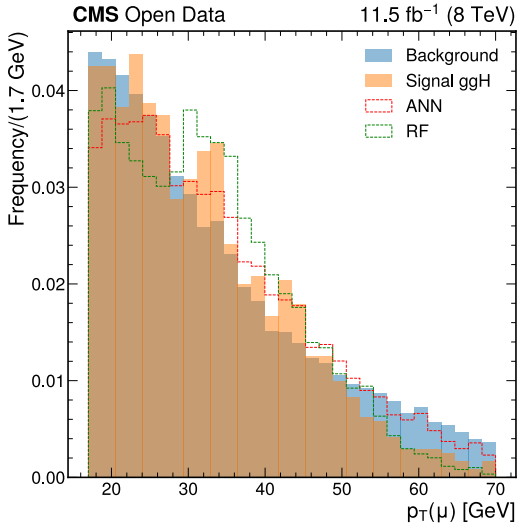
plot_functions.py,

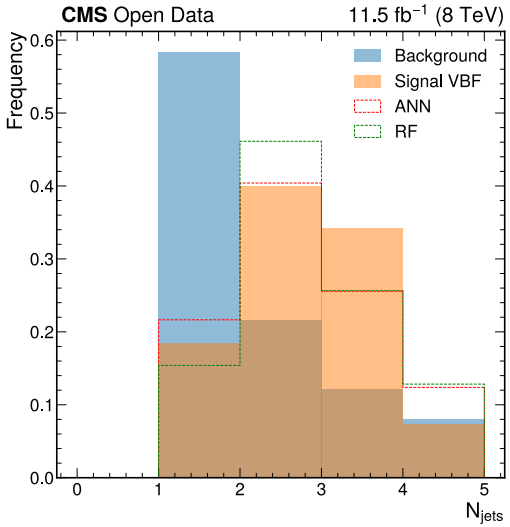
Documentation

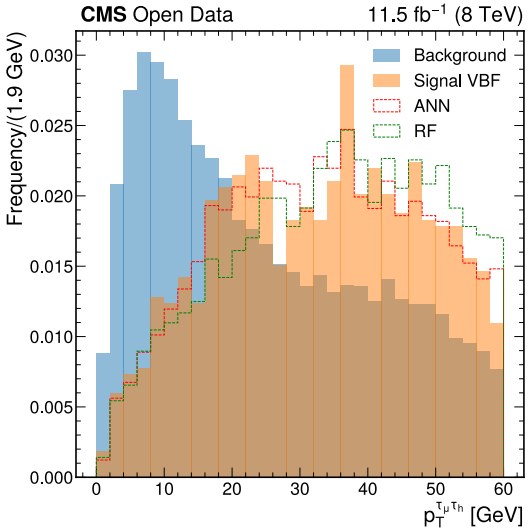
Machine Learning: classificazione segnale/fondo

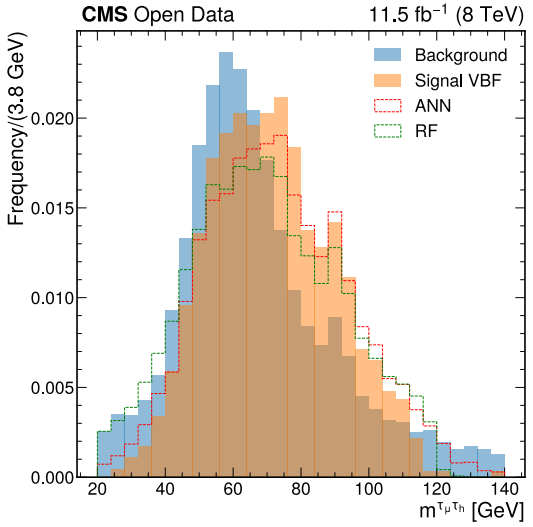












Variables plots

Machine Learning: Unittest

```python
import unittest
import os
from ML_Higgs import read_root

class TestMLfiles(unittest.TestCase):
    def setUp(self):
        self.root_files_list = ["GluGluToHToTauTau", "VBF_HToTauTau", "DYJetsToLL",
                                "TTbar", "W1JetsToLNu", "W2JetsToLNu", "W3JetsToLNu"]
        self.signal_ggH = 4492
        self.signal_VBF = 5326
        self.bkg_TTbar = 13630

        self.plots_list = ["nGoodJets_VBF", "PV_npvs_ggH", "MET_phi_ggH", "pt_vis_VBF",
                           "muon_pt_VBF", "muon_eta_ggH", "muon_phi_VBF", "muon_iso_VBF",
                           "tau_pt_ggH", "tau_eta_VBF", "tau_phi_ggH"]

    def test_files_exist(self):
        """ doc """
        print("TEST 1 - Called...")
        PATH = "ROOT_workspace/"
        results = []
        for file in self.root_files_list:
            results.append(os.path.isfile(PATH + file + "_selected.root"))
        self.assertTrue(all(results), "There aren't all ROOT files")

    def test_read_root_ggH(self):
        """ doc """
        print("TEST 2 - Called...")
        dataframe = read_root("GluGluToHToTauTau")
        value_counts = dataframe['event'].value_counts().to_numpy()
        self.assertEqual(value_counts[1], self.signal_ggH, "Must be 4492")
```

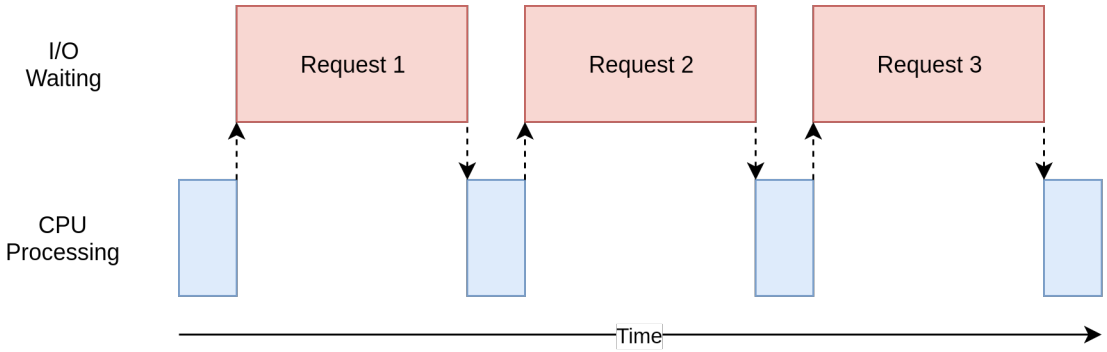
```
% python ML_test.py -b
test_files_exist (__main__.TestMLfiles)
Check on the existence of the file comparing with those in the root files list. ... ok
test_plots (__main__.TestMLfiles)
Check on the names of the saved plots comparing with those in plots list. ... ok
test_read_root_TTbar (__main__.TestMLfiles)
Check on the number of background events, e.g. in TTbar channel. ... ok
test_read_root_VBF (__main__.TestMLfiles)
Check on the number of signal events in the VBF channel. ... ok
test_read_root_ggH (__main__.TestMLfiles)
Check on the number of signal events in the ggH channel. ... ok
----------------------------------------------------------------------
Ran 5 tests in 13.381s
OK
```

```python
    def test_read_root_VBF(self):
        """ doc """
        print("TEST 3 - Called...")
        dataframe = read_root("VBF_HToTauTau")
        value_counts = dataframe['event'].value_counts().to_numpy()
        self.assertEqual(value_counts[1], self.signal_VBF, "Must be 5326")

    def test_read_root_TTbar(self):
        """ doc """
        print("TEST 4 - Called...")
        dataframe = read_root("TTbar")
        value_counts = dataframe['event'].value_counts().to_numpy()
        self.assertEqual(value_counts[1], self.bkg_TTbar, "Must be 13630")

    def test_plots(self):
        """ doc """
        print("TEST 5 - Called...")
        for file in self.plots_list:
            self.assertTrue(os.path.exists(f"ML_plots/{file}.pdf"),
                            f"{file} not found!")

if __name__ == '__main__':
    unittest.main(verbosity=2)
```

ML_test.py

Download dataset

Sequential




```
% python download_sequential.py
The VBF_HToTauTau.root download is assigned to the function call: 1
The 'download' function runtime is: 28.9870s
The GluGluToHToTauTau.root download is assigned to the function call: 2
The 'download' function runtime is: 22.4422s
The 'sequential' function runtime is: 51.4292s
Downloaded files: ['VBF_HToTauTau.root', 'GluGluToHToTauTau.root']
```

```python
# import libraries
import time
import requests
import functools

def timer(func):
    """ doc """
    @functools.wraps(func)
    def wrapper_timer(*args, **kwargs):
        start_time = time.perf_counter()
        value = func(*args, **kwargs)
        end_time = time.perf_counter()
        run_time = end_time - start_time
        print(f"The '{func.__name__}' function runtime is: {run_time:.4f}s")
        return value

    return wrapper_timer

@timer
def download(url, call):
    """ doc """
    response = requests.get(url, stream=True)
    file_name = url.split('/')[-1]
    print(f"The {file_name} download is assigned to the function call: {call}")
    with open(file_name, "wb") as file:
        for data in response.iter_content(1024):
            file.write(data)
    return file_name

@timer
def sequential():
    """ doc """
    urls = [
        "https://root.cern/files/HiggsTauTauReduced/VBF_HToTauTau.root",
        "https://root.cern/files/HiggsTauTauReduced/GluGluToHToTauTau.root"
    ]
    files = []
    call = 0
    for url in urls:
        call += 1
        files.append(download(url, call))
    return files


if __name__ == "__main__":
    done = sequential()
    print(f"Downloaded files: {done}")
```

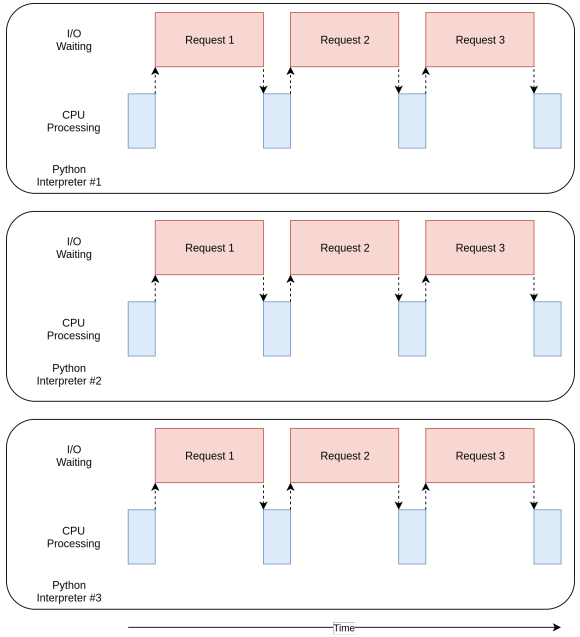
download_sequential.py

Download dataset

Parallel & Threading

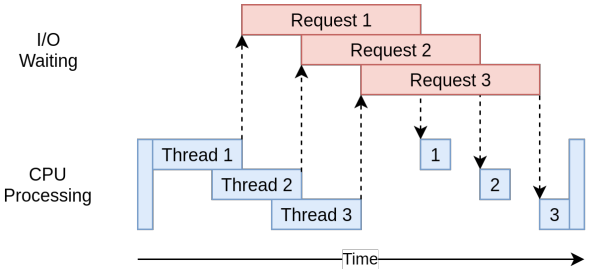
```python
import multiprocessing
...
def parallel():
    """ doc """
    urls = [
        "https://root.cern/files/HiggsTauTauReduced/VBF_HToTauTau.root",
        "https://root.cern/files/HiggsTauTauReduced/GluGluToHToTauTau.root"
    ]
    start_time = time.time()
    print(f"ID parent process: {os.getpid()}")
    with multiprocessing.Pool() as pool:
        pool.map(download, urls)
    run_time = time.time() - start_time
    print(f"Downloaded {len(urls)} in {run_time:.4f} seconds")
    return "Done"
```

```
% python download_parallel.py
ID parent process: 2949
The VBF_HToTauTau.root download is assigned to the process with ID: 2951
The GluGluToHToTauTau.root download is assigned to the process with ID:
2953
The 'download' function runtime is: 30.7368s
The 'download' function runtime is: 34.8277s
Downloaded 2 in 35.5345 seconds
Done
```




```python
import threading
...
def threads():
    """ doc """
    urls = [
        "https://root.cern/files/HiggsTauTauReduced/VBF_HToTauTau.root",
        "https://root.cern/files/HiggsTauTauReduced/GluGluToHToTauTau.root"
    ]
    NUM_THREAD = len(urls)
    start_time = time.time()
    # define a list of threads
    threads = [threading.Thread(target=download, args=(urls[x],)) for x in range(NUM_THREAD)]
    # start threads
    for thread in threads:
        thread.start()
    # join threads
    for thread in threads:
        thread.join()
    run_time = time.time() - start_time
    print(f"Downloaded {NUM_THREAD} in {run_time:.4f} seconds")
    return "Done"
```

```
% python download_threads.py
The VBF_HToTauTau.root download is assigned to the thread: Thread-1
The GluGluToHToTauTau.root download is assigned to the thread: Thread-2
The 'download' function (Thread-2) start time is: 0.5794s, end time is:
37.7494
The 'download' function (Thread-1) start time is: 0.5730s, end time is:
40.1196
Downloaded 2 in 39.5469 seconds
Done
```



download_parallel.py, download_threads.py

Download dataset

Unittest

```python
import os
import unittest
from download_sequential import download

class TestDownload(unittest.TestCase):
    def setUp(self):
        self.url = "https://root.cern/files/HiggsTauTauReduced/"
        self.files_list = ["VBF_HToTauTau.root", "GluGluToHToTauTau.root"]
        self.files_dim = [24184554, 20460281]  # bytes

    def test_download_files(self):
        """ doc """
        print("TEST 1 - Called...")
        result = []
        dim_result = []
        call = 0
        for file in self.files_list:
            call += 1
            result.append(download(self.url + file, call))
            dim_result.append(os.stat(file).st_size)
        self.assertEqual(result, self.files_list, "The files aren't the same")
        self.assertEqual(dim_result, self.files_dim, "The dimensions aren't the same")

if __name__ == '__main__':
    unittest.main(verbosity=2)
```

```
% python download_test.py -b
test_download_files (__main__.TestDownload)
Check on the name of the downloaded files compared to the expected one.
... ok
----------------------------------------------------------------------
Ran 1 test in 55.490s

OK
```

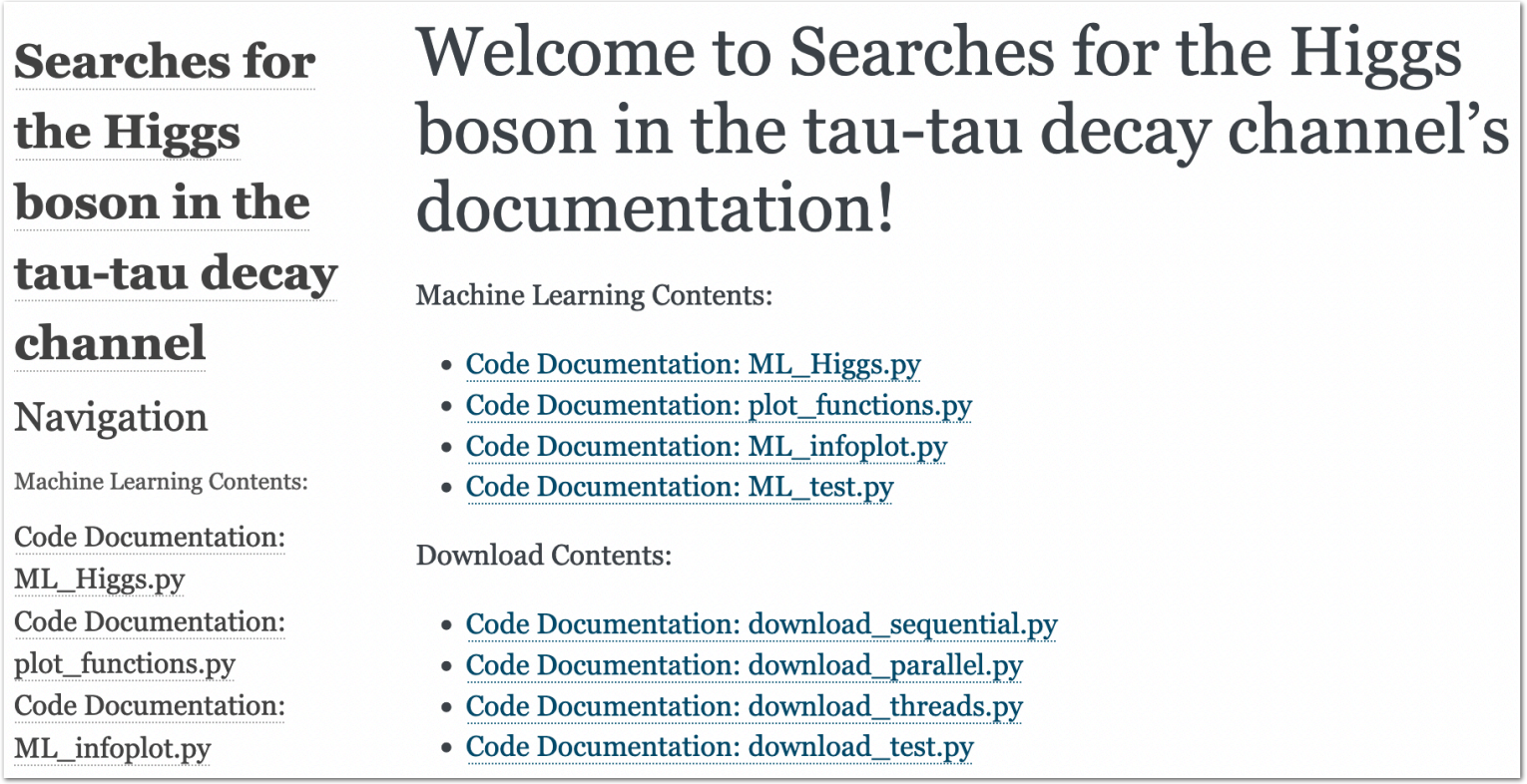
download_test.py

Documentation - Sphinx website

https://domric98.github.io/Searches-for-the-Higgs-boson-in-the-tau-tau-decay-channel/index.html

Files prodotti per la documentazione:

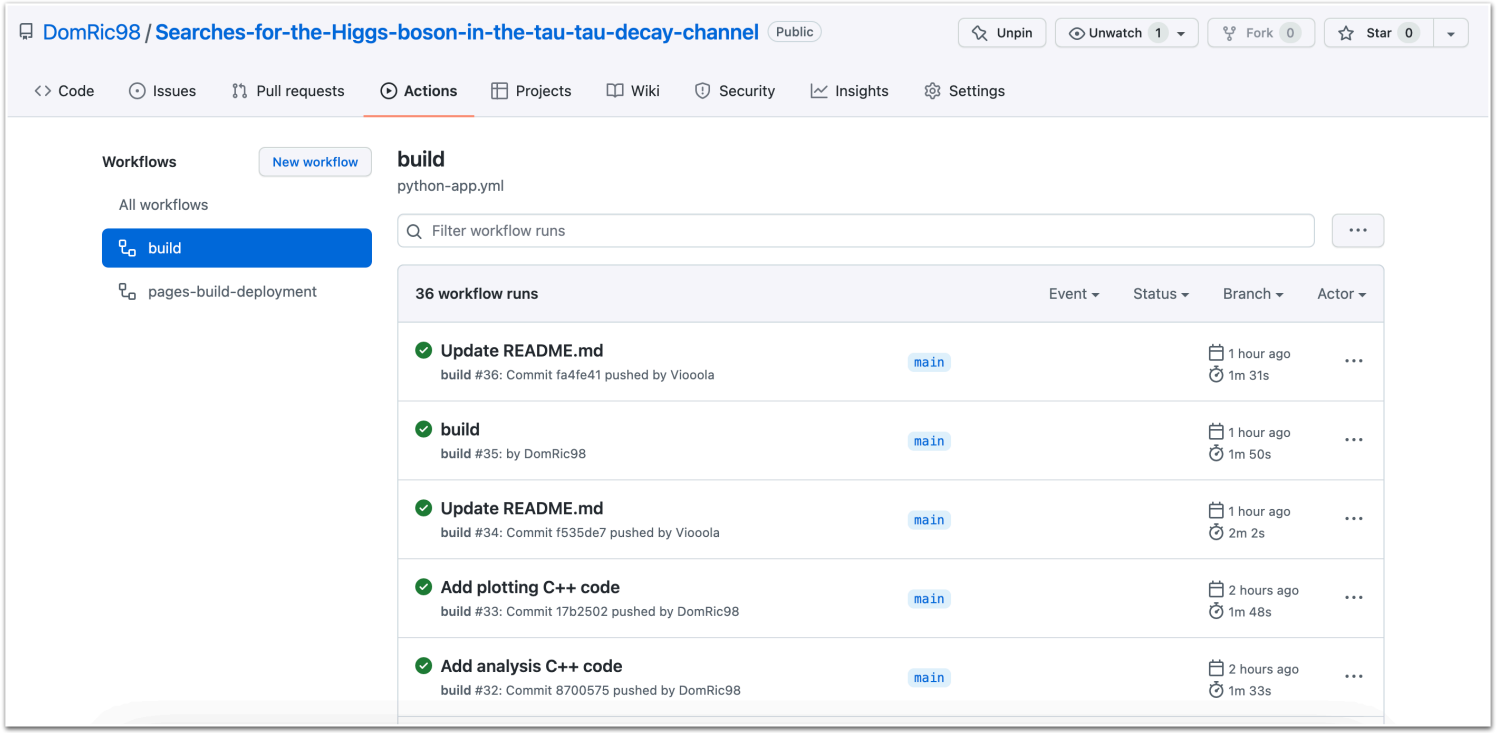
- conf.py
- index.rst
- ML_Higgs.rst
- plot_function.rst
- Infoplot.rst
- ML_test.rst
- download_*.rst



Workflow

On GitHub

```yaml
# This workflow will install Python dependencies, run tests and lint
with a single version of Python
# For more information see: https://help.github.com/actions/language-
and-framework-guides/using-python-with-github-actions

name: build

on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

permissions:
  contents: read

jobs:
  build:

    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v3
    - name: Set up Python 3.10
      uses: actions/setup-python@v3
      with:
        python-version: "3.10"
    - name: Install dependencies
      run: |
        python -m pip install --upgrade pip
        pip install flake8 pytest
        if [ -f requirements.txt ]; then pip install -r
requirements.txt; fi
    - name: Lint with flake8
      run: |
        # stop the build if there are Python syntax errors or undefined
names
        flake8 . --count --select=E9,F63,F7,F82 --show-source --
statistics
        # exit-zero treats all errors as warnings. The GitHub editor is
127 chars wide
        flake8 . --count --exit-zero --max-complexity=10 --max-line-
length=127 --statistics
    - name: Test with pytest
      run: |
        pytest
```



File di configurazione del “build workflow”

python-app.yml