Searches for the Higgs boson in the tau tau decay channel

Domenico Riccardi, Viola Floris
GitHub repository link

26 aprile 2022

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

Prima Parte - 3 cfu

Il progetto studia il processo di decadimento $H \to \tau_\mu \tau_h$ utilizzando i dataset prodotti dalla collaborazione CMS e resi disponibili come Open Data[1]. Nella prima parte dello studio si implementa una replica della procedura di selezione degli eventi prodotti nel processo di decadimento dell'Higgs SM nei due leptoni tau che decadono uno leptonicamente ($\tau^\pm \to \mu^\pm \nu_\mu \nu_\tau$) e l'altro adronicamente (prevalentemente $\tau^\pm \to \pi^\pm \pi^0 \nu_\tau$). I canali di produzione dominanti dell'Higgs a LHC sono *Gluon-Gluon fusion* e *Vector Boson fusion*, che corrispondono ad una sezione d'urto complessiva nel nostro canale di segnale di $\sigma(ggH) \sim 19.6\,\text{pb}$ @8 TeV e $\sigma(VBF) \sim 1.55\,\text{pb}$ @8 TeV. I background dominanti per il processo sono: Drell-Yan, eventi prodotti dall'annichilazione $q\bar{q}$, W + jets dovuti all'interazione p-p, $t\bar{t}$ da gluon fusion o annichilazione $q\bar{q}$ e fondi di QCD.

I dati utilizzati sono stati raccolti nella campagna di presa dati del 2012 da CMS (Run2012B-Run2012C) e corrispondono ad una luminosità integrata di $\mathcal{L} \simeq 11.5\,\text{fb}^{-1}$. Le simulazioni MC comprendono sia i due canali di produzione dell'Higgs con successivo decadimento nel nostro canale di segnale, sia parte dei fondi elencati prima. In tabella 1 si raccolgono i processi che sono stati adoperati nell'analisi assieme alla sezione d'urto e al numero di eventi utilizzati per calcolare l'opportuno fattore di scala (peso dell'evento) per confrontare i MC con i dati.

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

Tabella 1: Campioni MC 2012 a 8 TeV per l'analisi con annessa sezione d'urto e numero di eventi generati.

Sono stati scritti codici in ROOT/C++ che implementano la procedura di selezione degli eventi e il plotting delle distribuzioni delle variabili contenute nei ROOT files. Nello specifico, nel

[1] http://opendata.web.cern.ch/record/12350

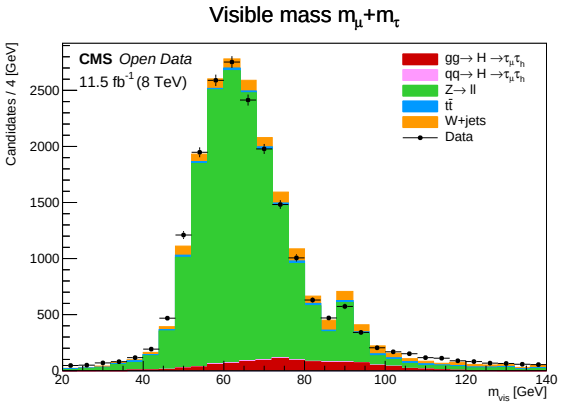

Figura 1: Massa visibile.

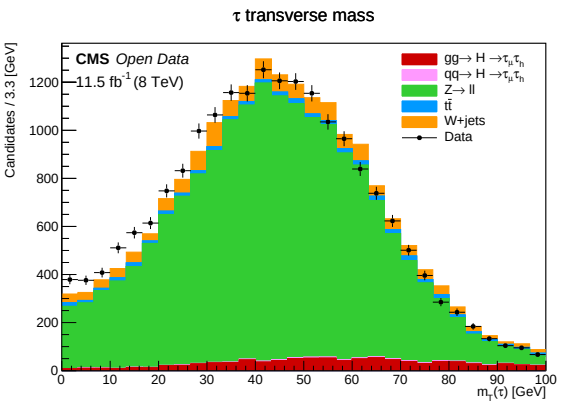

Figura 2: Massa trasversa del τ.

file `analisys.cpp` vengono applicati i tagli esplicitati in tabella 2, costruite le coppie di candidati $H \to \tau_\mu\tau_h$ e calcolate delle nuove variabili che sono state utilizzate nella successiva fase di Machine Learning. Invece, nello script `plotting.cpp` viene definita una procedura per la rappresentazione delle distribuzioni fisiche che produce, ad esempio, quelle riportate nelle figure 1 e 2. È stata anche implementata la parallelizzazione via multithreading con `ROOT::EnableImplicitMT()` e utilizzate le classi `RDataFrame`, `RVec` e altre della libreria Standard (namespace std).

Function tamplates	Selections
`muon_selection`	$p_T^\mu > 17\,\text{GeV}$, $\lvert\eta^\mu\rvert < 2.1$, `Muon_tightId`=1, `nMuon` > 0
	`HLT_IsoMu17_eta2p1_LooseIsoPFTau20`=1
`tau_selection`	$p_T^{\tau_h} > 20\,\text{GeV}$, $\lvert\eta^{\tau_h}\rvert < 2.3$, $q^{\tau_h} \neq 0$, `nTau` > 0
	`Tau_idIsoTight`=1, `Tau_idAntiEleTight`=1, `Tau_idAntiMuTight`=1
`take_jet`	$p_T^{jet} > 30\,\text{GeV}$, $\lvert\eta^{jet}\rvert < 4.7$, `Jet_puId`=1

Tabella 2: Constraints implementati dalle funzioni contenute nel codice d'analisi.

Algoritmi ML per la classificazione segnale/fondo (Python)

Seconda Parte - 6 cfu

I file `*_selected.root` contenenti i candidati Higgs prodotti nella prima parte del progetto, rappresentano l'input dello script `ML_Higgs.py` nel quale sono implementati due algoritmi di Machine Learning, un'*Artificial Neural Network* (ANN) e un *Random Forest* (RF), che si occupano di effettuare la classificazione binaria segnale/fondo. Il canale di segnale, VBF o ggH, viene scelto esplicitamente dall'utente all'inizio dell'esecuzione del programma.

Per l'integrazione dei ROOT files con l'ambiente Python si è utilizzata la libreria Uproot che, leggendo questi ultimi, permette di produrre dei Pandas Dataframes (funzione `read_root`). La scelta delle features $(x_1, \ldots, x_n)$ per gli algoritmi è stata fatta studiando la correlazione tra le variabili, scegliendo le coppie con $\text{corr}(x_i, x_j) \lesssim 0.8$, e ciò ha consentito di selezionare solo le features più rilevanti in modo da ridurre la possibilità di overfitting e incrementare l'accuracy (figure 3 e 4). Il vettore target, invece, viene costruito assegnando 1 agli eventi di segnale, 0 a quelli di fondo (*true label*).

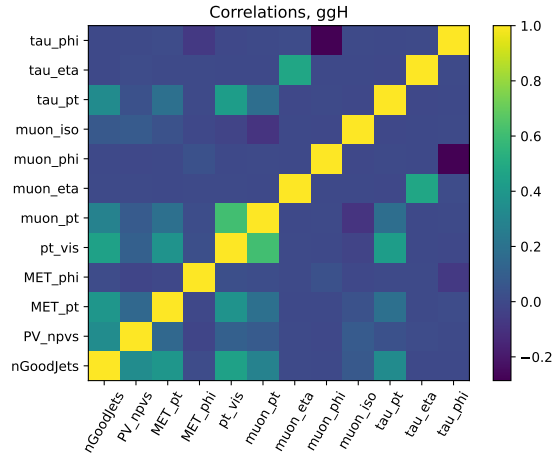

Figura 3: Correlation plot, canale ggH.

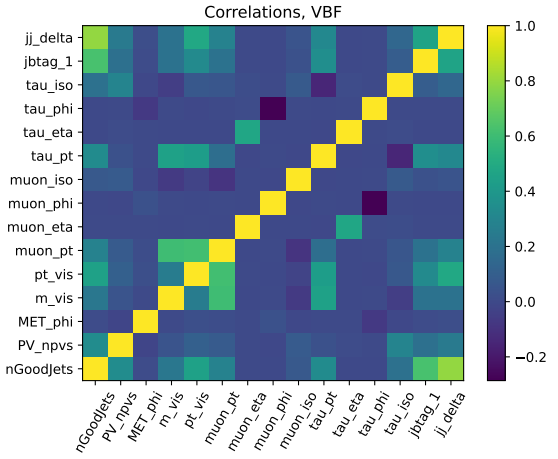

Figura 4: Correlation plot, canale VBF.

Dovendo lavorare su dataset particolarmente sbilanciati, abbiamo effettuato una rinormalizzazione degli eventi in modo tale che il classificatore si concentrasse maggiormente su quelli di segnale, indipendentemente dal numero di eventi e pesi assoluti per ogni campione. Inoltre, essendo i range di variazione delle features molto diversi tra loro, è stata applicata una standardizzazione sulle variabili in modo che la rete potesse lavorarci equamente.

La scelta degli iperparametri per l'ANN è stata effettuata esplorando una serie di possibilità manualmente, ottenendo i migliori risultati con:

- Numero di hidden (Dense) layers: 3
- Dropuot (dopo ogni hidden layer): 0.1, 0.5, 0.1
- Activation func. (per hidden layer): selu, selu, selu, sigmoid
- Input shape: (N_VARS,)
- Numero di neuroni per layer - ggH(VBF): N_VARS×10(15), N_VARS×2, N_VARS
- Optimizer: adam
- Loss: binary crossentropy
- Metric: accuracy
- Numero di epoche: 150 (ggH), 100 (VBF)

Mentre per il RF è stata implementata una proceduta di *hyperparametern tuning*, che restituisce:

- Numero di estimatori: 500
- Profondità: 5
- Criterion: gini
- Numero max di features: N_VARS
- Bootstrap: True

Avendo nel nostro problema di classificazione uno squilibrio nelle due classi, la scelta di una default threshold a 0.5 porta a prestazioni scadenti. Dunque l'approccio che abbiamo deciso di utilizzare è quello di calcolare delle optimal thresholds che permettono di convertire le probabilità (output score dell'algoritmo) in delle label nitide. Utilizzando la media geometrica come metrica per cercare un equilibrio tra sensitività e specificità, ovvero un bilanciamento tra rate di veri (tpr) e falsi (fpr) positivi, è stata trovata una prima threshold che la massimizza. Formalmente

$$g\_mean = \sqrt{tpr(1-fpr)} = \sqrt{sensitivita' \times specificita'} \tag{1}$$

dove la coppia (tpr, fpr) che produce max(g_mean) è stata indicata sulle ROC curve, figura 5. Alternativamente è possibile definire una soglia che si traduce nel miglior equilibrio tra precision e recall ottenuta massimizzandone il prodotto (proporzionale alla f-score metric). Calcolando la precision-recall curve, infatti, viene definita per ogni coppia una threshold e valutando questa metrica semplificata

$$f\_score\ (simpl.) = precision \times recall \tag{2}$$

individuiamo la corrispondente soglia ottimale, indicata in figura 6.

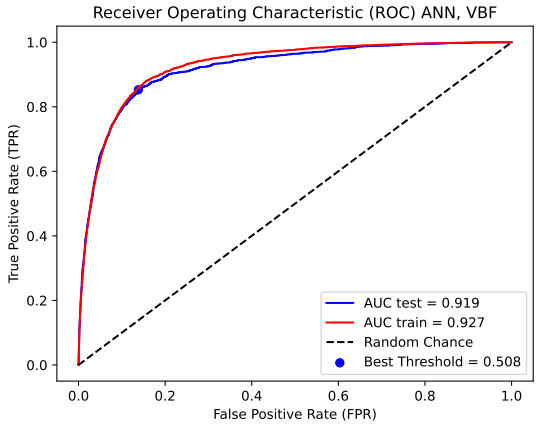

Figura 5: ROC curve per l'ANN, canale VBF.

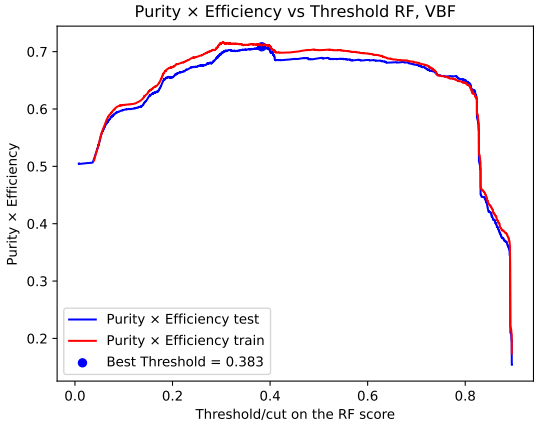

Figura 6: Precision(Purity) - Recall(Efficiency) curve per RF, canale VBF.

Abbiamo prodotto i grafici di confronto delle distribuzioni degli output score per i sample di training e test come check sull'overtraining per ogni canale di segnale, figure 7 e 8, dove è indicata anche la soglia scelta con uno dei metodi descritti prima.

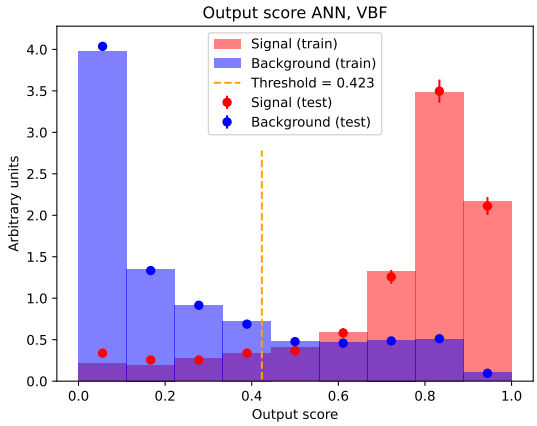

Figura 7: Output score per l'ANN, canale VBF.

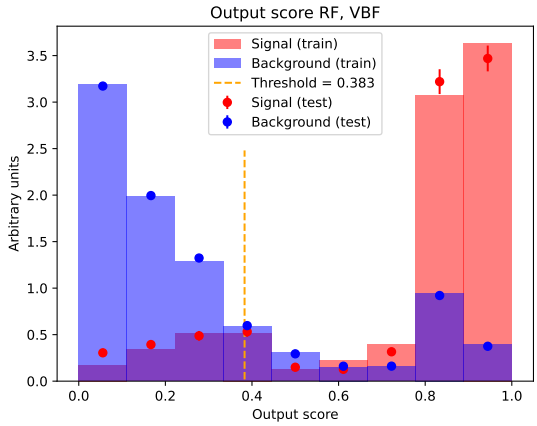

Figura 8: Output score per RF, canale VBF.

Per valutare e confrontare le prestazioni dei due modelli sullo stesso dataset di test, sono state calcolate diverse metriche riportate in tabella 3. Lo stesso dataset di test è stato utilizzato per produrre le distribuzioni normalizzate delle features, separando la componente di segnale da

quella di fondo e comparandole con le predizioni del segnale ottenute dai due modelli ML. A titolo di esempio si riportano tre distribuzioni in figure 9, 10, 11.

Final Value	ANN		RF	
	VBF	ggH	VBF	ggH
Best Threshold	0.42 (f)	0.46 (g)	0.38 (f)	0.46 (g)
Accuracy	0.85	0.74	0.84	0.69
Precision	0.84	0.74	0.83	0.70
Recall	0.88	0.73	0.86	0.67

Tabella 3: Valori finali di alcune metriche ottenuti per i due algoritmi di ML avendo fissato le best threshold. Si indica con (g) quella ottenuta dalla g-mean, mentre con (f) quella con l'f-score.

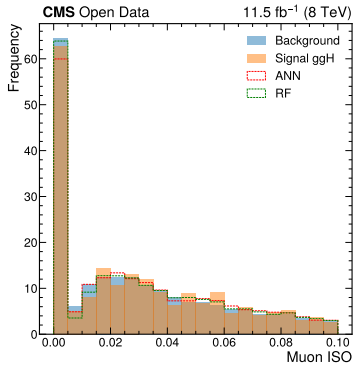

Figura 9: Isolamento μ, canale ggH.

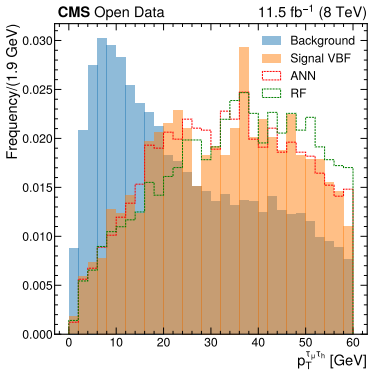

Figura 10: Impulso trasverso visibile, canale VBF.

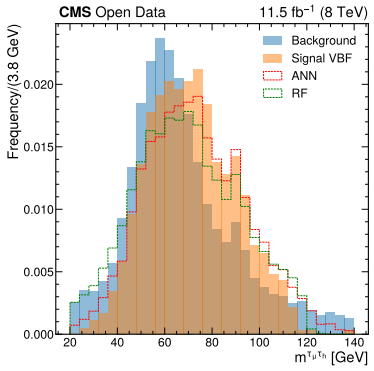

Figura 11: Massa visibile, canale VBF.

Infine, sono stati prodotti sei unittest (`ML_test.py`) per verificare alcune funzionalità degli algoritmi e testare la corretta produzione dei plots.

Per fornire all'utente la possibilità di scaricare i dataset in maniera automatizzata, sono stati implementati degli script in puro linguaggio Python. Tali algoritmi consentono il download dei ROOT files in maniera sequenziale e in parallelo sfruttando, per quest'ultima modalità, alcuni tools delle librerie *multiprocessing* e *threading*. A completamento di ciò è stato realizzato uno unittest per testare la correttezza del metodo di download.

L'intero progetto è stato completamente documentato utilizzando il generatore Sphinx e hostato su GitHub[2]. Inoltre attraverso GitHub Actions è stato implementato un flusso di lavoro (build workflow) che installando delle opportune dipendenze, `requirements.txt`, esegue i test esplicitati prima.

[2]Così come tutto il materiale che per ragioni di spazio non è stato presentato nella relazione.