SAPIENZA
UNIVERSITÀ DI ROMA

PROGETTO MACHINE LEARNING (GRUPPO 4)

Classificazione di flavor jet in HEP tramite DNN

Favorito Giovanbattista,[1] Ferri Thomas,[2] e Gasbarri Gabriele[3]

[1]favorito.1939167@studenti.uniroma1.it
[2]ferri.1959757@studenti.uniroma1.it
[3]gasbarri.1943359@studenti.uniroma1.it

Abstract

Per la classificazione di flavor jet generati da quark pesanti (classe Signal) e da quark leggeri (classe Background), si implementano 3 diversi algoritmi: una Feedforward Neural Network (FNN), un XGBoost ed una Long Short Term Memory Recurrent Neural Network (LSTM). Utilizzando un campione limitato del dataset originario, si ottengono i seguenti valori di accuracy sul test set: 91.5% per la FNN, 92.4% per XGBoost e 90.0% per la LSTM. Si discutono infine dei metodi possibili per migliorare l'accuracy del task di classificazione, ragionando principalmente sul bilanciamento del campione tra le due classi.

1. Introduzione

Nell'ambito della fisica sperimentale delle alte energie (HEP, dall'inglese High Energy Physics), il problema della classificazione dei sapori di jet prodotti in collisioni tra particelle fondamentali è di essenziale importanza al fine di ricavare alcune proprietà dei costituenti primi della natura e delle loro interazioni. La presente relazione propone un metodo di classificazione basato su Deep Neural Networks (DNN): l'obiettivo è implementare un algoritmo che riesca a distinguere jet prodotti da quark leggeri (up, down, strange, charm) rispetto a quelli prodotti da quark bottom, circa 4 volte più pesanti dei charm. Nel documento viene riportata l'analisi del dataset utilizzato nell'articolo *Jet Flavor Classification in High-Energy Physics with Deep Neural Networks* [a] del 2016; il dataset è disponibile online [b].

Nella prima parte della relazione si discutono brevemente gli esperimenti di HEP che permettono la produzione di jet tramite la collisione di quark pesanti e leggeri: l'algoritmo di classificazione risulterebbe dunque un potente strumento per l'analisi dei dati prodotti da tali esperimenti. Nella seconda parte viene presentata una discussione approfondita del dataset, in particolare focalizzandosi sulla fase di preprocessing dei dati necessaria al fine di ottimizzare l'algoritmo di classificazione tramite reti neurali. Nella terza parte si analizzano i due tipi di DNN che vengono adottati per affrontare il problema: una Feedforward Neural Network (FNN) ed una Recurrent Neural Network (RNN) di tipo Long Short Term Memory (LSTM). Si passa in seguito a delle considerazioni sull'effettiva implemetazione dei due modelli di DNN, valutando la scelta migliore per gli iperparametri delle reti e discutendo i risultati del training dei due modelli.

2. Esperimento

Gli esperimenti di collisioni adroniche consistono nel far scontrare fasci di protoni o altri adroni ad alta energia. I protoni, similmente ai nuclei, sono fondamentalmente costituiti da vuoto, dal momento che i 3 quark e il numero indefinito di gluoni che li costituiscono hanno le dimensioni di circa $\frac{1}{10000}$ il raggio di un protone. In base al principio del confinamento del colore i quark e i gluoni non possono essere osservati singolarmente. Questo significa che non appena due protoni collidono, le particelle elementari che li compongono creano un jet di particelle (principalmente adroni ma anche mesoni), in quanto le particelle che si frammentano dall'adrone iniziale, interagiscono con i quark e gli anti-quark generati nel vuoto. Una rappresentazione schematica del fenomeno di formazione di jet è visibile in Figura 1.

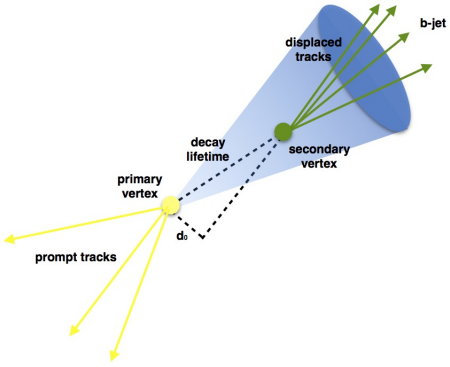

Figura 1. Schematizzazione della formaziona dei jet da classificare.

In linea generale, capire da quale quark scaturisce un jet che può poi essere rivelato dai calorimetri, è un compito particolarmente complicato per la quantità di dati da elaborare. In questo frangente possono risultare molto utili algoritmi di intelligenza artificiale. Nel corso del tempo sono stati sviluppati diversi programmi in grado di simulare ciò che avviene a LHC, uno di questi è madgraph5 con cui sono state simulate le collisioni e il decadimento. Per quanto riguarda il jet di particelle adroniche è invece stato usato PYTHIA, mentre le rivelazioni da parte dei detector (calorimetri) sono state simulate mediante DELPHES. I vertici di decadimento sono stati

a. https://arxiv.org/pdf/1607.08633.pdf
b. http://mlphysics.ics.uci.edu/data/hb_jet_flavor_2016/

ricostruiti mediante l'algoritmo RAVE, che fitta la posizione del primo vertice e poi procede per fittare gli altri vertici in cascata. Le tracce sono poi ricostruite a partire dai vertici stessi in base alla loro compatibilità.

3. Dataset

Il dataset è costituito da 10 milioni di righe in un file json; dato l'enorme peso dei dati a disposizione, si è deciso di utilizzare solo una frazione di essi per l'allenamento delle reti neurali, cercando di caricare il maggior numero di righe possibile senza però mandare in overload la RAM. Sono state dunque utilizzate 2 milioni di righe.

Ciascuna riga del dataset è organizzata in 16 features di alto livello disposte nelle colonne 0,1,3,4. Le colonne 3 e 4 contengono due liste rispettivamente da 8 e 6 elementi; le liste sono state quindi riorganizzate in modo da ottenere le 16 colonne di interesse. Nella colonna 2 sono presenti le labels dei jet come riportato in Tabella 1.

Tabella 1. Tabella con le labels assegnate a ciascun tipo di quark, nel codice le labels sono state riassegnate come scritto tra parentesi.

Label	Quark
0 (0)	Light Jet (Background)
4 (0)	Charm Jet (Background)
5 (1)	Bottom Jet (Signal)

Le labels associate ai tipi di quark sono state riassegnate dividendo la classe del segnale (bottom), a cui è stato associato il target 1, dalla classe di background(charm e light), a cui è invece assegnato il target 0.

Per la visualizzazione del dataset sono stati realizzati 16 istogrammi (uno per colonna) per mostrare le distribuzioni di ciascuna feature; in particolare, sono state divise le tre classi corrispondenti ai tre diversi tipi di jet. Gli istrogrammi sono riportati in Figura 9 in Appendice.

Nella colonna 5 sono contenute le features di basso livello. Queste contengono informazioni sulle tracce e i vertici del jet. I dati sono presentati come liste di 28 features diverse che si ripetono per un certo numero di volte, differente per ogni riga del dataset. L'allenamento di una rete neurale DNN per questo tipo di features risulta particolarmente complesso da un punto di vista computazionale, inoltre le features non avendo tutte la stessa dimensione devono essere ridimensionate stabilendo un cutoff fisso e facendo un padding per tutte le righe che hanno un numero di features inferiore a quello stabilto. In ogni caso si è visto come l'uso delle sole features di alto livello sia sufficiente per raggiungere delle prestazioni più che soddisfacienti.

4. Training del modello

Si passa ora all'implementazione delle reti neurali per il task di classificazione. Inizialmente viene analizzata una FNN con 9 strati deep ed una sua variante con un numero inferiore di strati per confrontare le prestazioni. Vengono anche analizzati i risultati ottenuti dall'algoritmo XGBoost e confrontati con quelli ottenuti dalle FNN. Si implementa infine la LSTM, riportando anche in questo caso l'accuracy sul test set.

Il preproccessing dei dati viene realizzato tramite pytorch e le 2 milioni di righe del dataset, ciascuna contenente 16 features, sono divise in tre campioni:

- 1 600 000 righe (80 %) per il training set;
- 200 000 righe (10 %) per il validation set;
- 200 000 righe (10 %) per il test set.

Prima dell'allenamento delle reti il vettore di features viene opportunamente normalizzato tramite la funzione MinMaxScaler.

Per la realizzazione della LSTM, il dataset è invece trattato in modo diverso in quanto l'implementazione della RNN richiede alcune variazioni nella logica del dataset (cfr sezione 4.3).

4.1 Feedforward Neural Network

La rete neurale implementata inizialmente ha 9 strati ciascuno con un numero di nodi per gli hidden layers che varia da un massimo di 512 a un minimo di 32. Le features in ingresso sono le features di alto livello descritte al paragrafo 2. In uscita a ciascun layer si è usata ReLu come funzione di attivazione a delle funzioni di dropout con una probabilità $p = 0.1, 0.25, 0.5$. In uscita si applica una sigmoide per riportare l'output in un range di [0,1], avendo scelto come funzione di Loss la "BCELoss" della libreria nn, che al contrario di "CrossEntropyLoss" non implementa automaticamente la softmax. I vari iperparametri come la batch size, il numero di epochs e il learning rate sono stati settati manualmente. In particolare, per il learning rate si è usato un decremento di un fattore 0.1 ogni 5 epoche, e il metodo di ottimizzazione che si è scelto è ADAM. Un numero di epoche pari a 10 si è rivelato del tutto sufficiente; infatti, nonostante nell'articolo di riferimento siano state usate 100 epoche in fase di training, si è visto come già con 10 epoche sia la loss che la funzione di metrica utilizzata convergano ad un plateau. Ciò e possibilmente dovuto al numero ridotto di dati analizzati rispetto all'articolo originale.

Si è scelto di usare l'Area Under Curve (AUC) come metrica per visualizzare il progresso della rete in quanto l'accuracy (calcolata come $acc = \frac{n\,corretti}{n\,totali}$) non varia significativamente durante il training. I risultati ottenuti sono riportati in Figura 2.

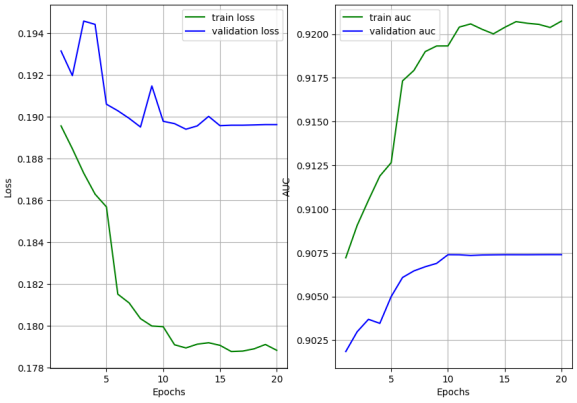

Figura 2. Grafici della loss e dell'AUC in funzione del numero di epoche per la prima rete neurale

Come si vede bene in Figura 2, la rete overfitta i dati in quanto minimizza la funzione di loss per il training set ma non per il validation set. La curva di ROC è presente in Figura 3. Inoltre l'AUC a cui tende il validation set è decisamente inferiore a quella ottenuta per il training.

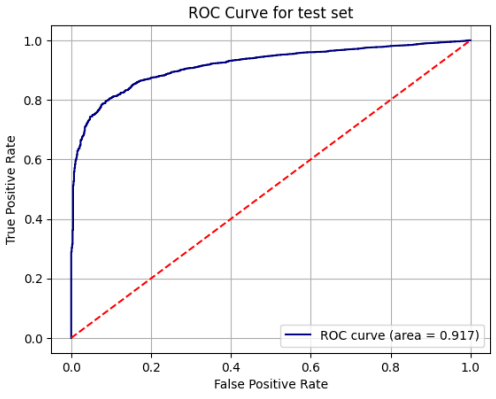

Figura 3. ROC curve per la rete neurale a 9 strati calcolata sul test set.

Si ottiene quindi un'accuracy sul test set del 91.7%. Nonostante i risultati siano soddisfacenti, in quanto per algoritmi di Deep Machine Learning l'hardcore overfitting non sempre è un problema, si è comunque scelto di testare una rete con un numero di layer inferiore. In particolare si è scelta una rete di 5 layer con un numero di nodi per i 3 hidden layers che va da un massimo di 64 a un minimo di 16. Similmente a quanto visto per la rete precedente si sono applicate funzioni di dropout in uscita ai layer questa volta con $p = 0.2, 0.25, 0.5$ in modo da ridurre maggiormente l'effetto di overfitting. I risultati ottenuti sono riportati in Figura 4:

Come si vede chiaramente il problema dell'overfitting è stato del tutto superato da questo nuovo modello di DNN, ad ogni modo è importante notare come il livello al quale si assesta la loss per il validation set è lo stesso mostrato in Figura 2. Invece la loss per il training set rimane sempre leggermente più elevata. Si procede quindi a mostrare la valutazione delle prestazioni della rete con una curva ROC, riportata in Figura 5.

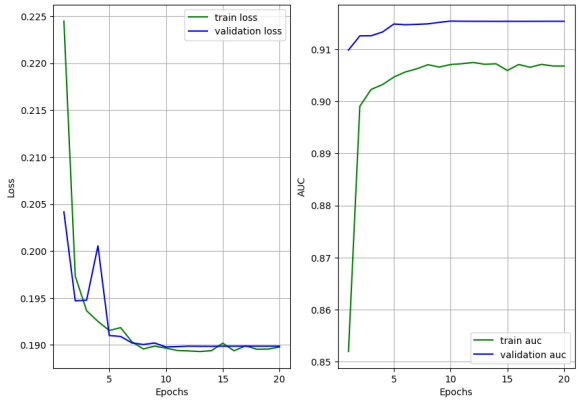

Figura 4. Loss e AUC in funzione delle epochs per la rete con meno parametri.

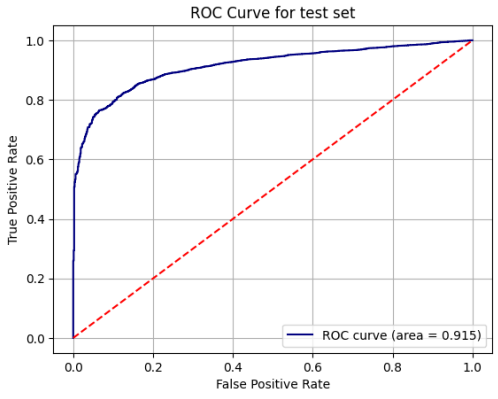

Figura 5. ROC curve per la rete neurale a 5 strati calcolata sul test set

La curva è del tutto equivalente a quella trovata precedentemente per la prima rete e mostrata in Figura 3. Si ottiene un valore dell'accuracy sul test set pari a 91.5%; si nota quindi un leggero peggioramento del 0.2% dell'area sotto la curva ottenuta con il secondo modello rispetto a quella ottenuta con la rete a 9 strati. Ad ogni modo è generalmente preferibile utilizzare un modello che si adatta meglio ai dati soprattutto quando le differenza nelle prestazioni sono minime come in questo caso.

4.2 XGBoost

Un algoritmo di classificazione estremamente efficiente che non fa uso di tecniche di DL è XGBoost. Essendo molto facile da implementare si è scelto di eseguire un test per vedere anche le prestazioni di questo modello. Una volta preimpostati i parametri manualmente, si sono eseguite 150 round e il risultato ottenuto è mostrato in Figura 6.

L'accuracy sul test set è pari a 92.5%. Il risultato è del tutto paragonabile a quello ottenuto con gli altri due modelli, con un miglioramento dell'area sotto la curva di circa l'1 %. Una cosa da tenere presente è che oltre a essere leggermente migliore a livelli di prestazioni, XGBoost è anche significativamente più semplice da implementare, non richiedendo un tuning troppo fine degli iperparametri, e anche molto più veloce come tempo di esecuzione.

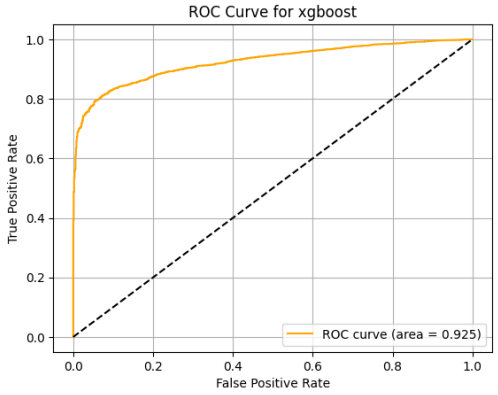

Figura 6. Cruva ROC ottenuta tramite XGBoost calcolata sul test set.

4.3 LSTM Network

La rete neurale ricorrente di tipo LSTM viene realizzata organizzando le righe di features in sequenze "temporali"; l'obiettivo della rete è dunque quello di produrre un determinato output data una certa sequenza in input. Poiché il task da risolvere è completamente scollegato da un contesto temporale, si è deciso di far corrispondere una sequenza ad una singola riga del dataset, ovvero ad un singolo vettore di features.

E' stato operativamente verificato come l'uso delle 2 000 000 di righe del dataset utilizzato per il training della FNN comporti un overload della RAM nel caso della LSTM, la cui implementazione richiede più memoria. Si considerano dunque 100 000 righe di features per il training della LSTM; ci si aspetta quindi un'accuracy inferiore in quanto il campione su cui si allena la RNN è notevolmente ridotto. Il dataset costituito da 100 000 samples viene diviso in:

- 80 000 righe (80 %) per il training set;
- 10 000 righe (10 %) per il validation set;
- 10 000 righe (10 %) per il test set.

La normalizzazione dei vettori delle features è realizzata tramite la funzione MinMaxScaler, chiamata direttamente all'interno della classe MyDataset che implementa la logica del dataset per la LSTM.

Il dataloader è poi costruito tramite batch di taglia pari a 50. La rete neurale ha come dimensione in input la lunghezza delle features (16); il numero di unità nascoste dell'unico strato della LSTM è posto pari a 64. Seguendo l'esempio dell'articolo di riferimento, l'output della RNN viene usato come input di una FNN composta da fully connected layers. E' stato inizialmente implementato solo uno strato contenente 64 unità operative, pari al numero di unità nascoste della LSTM, ma si è poi sperimentato con 2, 3 e 4 strati fully connected in uscita alla LSTM. Con un numero troppo elevato di unità di calcolo si osserva il plateau della loss sul validation set ad un valore più alto rispetto al plateau raggiunto dalla loss sul training set, sintomo di overfitting della rete. Sono stati quindi tarati i parametri ottimali della rete a 64 unità nel primo strato, 64 nel secondo, 32 nel terzo e 16 nell'ultimo. Dopo ciascuno strato lineare è utilizzata una funzione di attivazione di tipo Relu ed un dropout al fine di evitare l'overfitting osservato. L'output dell'ultimo strato ha invece dimensione 1; si applica ad esso una funzione sigmoide per riportare l'output tra 0 e 1.

La funzione di loss utilizzata anche in questo caso è BCE, e l'ottimizzatore è nuovamente Adam con un learning rate pari a 10^{-3}; usando infatti come ottimizzatore la discesa stocastica lungo il gradiente (SGD) con momentum pari a 0.9 come suggerito dall'articolo di riferimento, i risultati del training erano leggermente peggiori rispetto a quelli ottenuti con Adam. La funzione di metrica utilizzata nel training è sempre AUC della libreria torchmetrics.

Il modello è stato allenato per un totale di 20 epoche; si ottengono i grafici dell'andamento della loss e della metrica in funzione delle epoche in Figura 7.

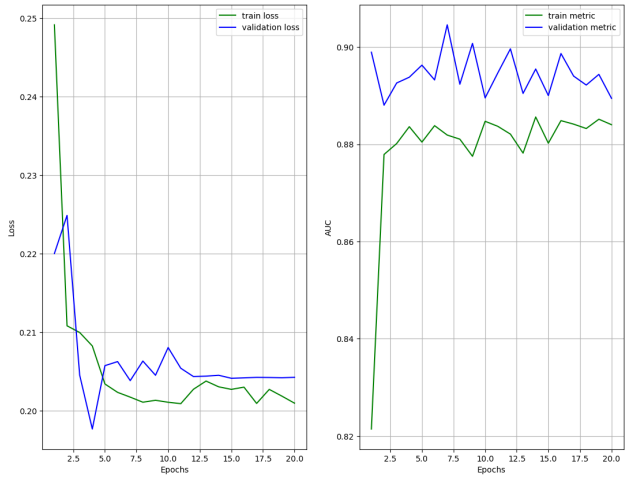

Figura 7. BCE Loss e AUC in funzione delle epoche per la rete ricorrente LSTM realizzata.

Vengono dunque eseguite le predizioni sul test set; le prestazioni si misurano in termini della curva ROC, riportata in Figura 8.

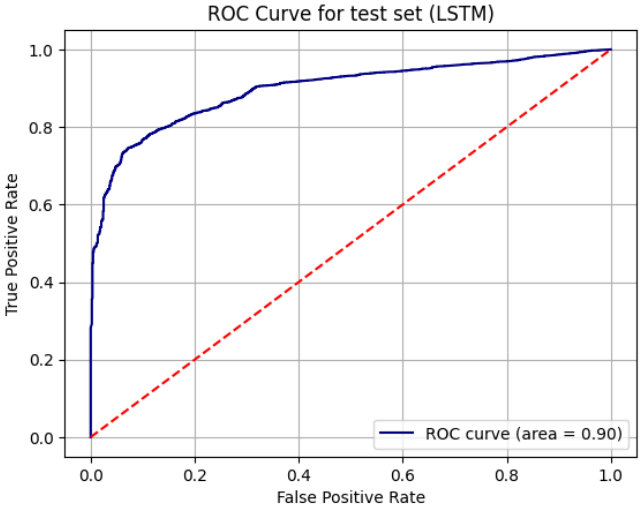

Figura 8. Curva ROC calcolata sul test set per valutare le prestazioni della rete LSTM.

Si ottiene un'accuracy del 90%, vicina ai risultati ottenuti

con la FNN e con XGBoost.

5. Risultati e conclusioni

In Tabella 2 vengono riassunti i risultati ottenuti dagli algoritmi implementati.

Tabella 2. Tabella riassuntiva con i diversi valori dell'AUC del grafico ROC relativi a ciascun algoritmo implementato.

Algoritmo	AUC su test set
FNN	91.7%
XGBoost	92.5%
LSTM	90.0%

L'AUC ottenuta tramite la LSTM è inferiore rispetto a quella calcolata tramite FNN e XGBoost possibilmente a causa del numero ridotto di samples su cui è stata allenata la RNN. I primi due algoritmi hanno infatti eseguito un training su un campione 20 volte più grande del campione di training utilizzato nella LSTM; ciò potrebbe inoltre essere collegato all'andamento irregolare di loss e metrica sul validation set osservato in Figura 7.

I metodi che potrebbero essere implementati per migliorare i risultati ottenuti dipendono molto dalla potenza computazionale a disposizione. Se si potesse utilizzare una RAM dotata di più memoria, si potrebbe allenare la LSTM su un campione maggiore di samples, in modo da ottenere un'AUC pari a quella ottenuta dalla FNN (ad esempio nell'articolo di riferimento, utilizzando solo features di alto livello, si ottiene un'accuracy maggiore nel caso della LSTM). Disponendo invece della stessa potenza di calcolo, si potrebbero realizzare diversi dataset considerando campioni sempre della stessa taglia (2 000 000 per la FNN, 100 000 per la LSTM), ma che contengono ogni volta righe diverse del dataset. Ciò potrebbe aiutare ad una migliore classificazione, in quanto si osserva che la distribuzione del target è sbilanciata a favore della classe di Signal: qualsiasi sia il numero di samples considerati (sia per i 2 000 000 della FNN che per i 100 000 della LSTM), si ottiene un rapporto tra numero di eventi relativi a Background e numero di eventi relativi a Signal pari a circa 10%. Si potrebbero dunque prelevare determinate righe dal dataset originario in modo da costruire un altro dataset che contenga un rapporto più bilanciato di eventi di classe Signal ed eventi di classe Background. Inoltre in linea generale si potrebbero usare anche le features di basso livello, anche se il loro uso risulta complesso computazionalemente. Infatti queste, rappresentando i dati come sono stati rilevati dal rivelatore, includono informazioni utili che potrebbero essere state trascurate nel processo di produzione delle features di alto livello.

Appendice 1. Visualizzazione del dataset

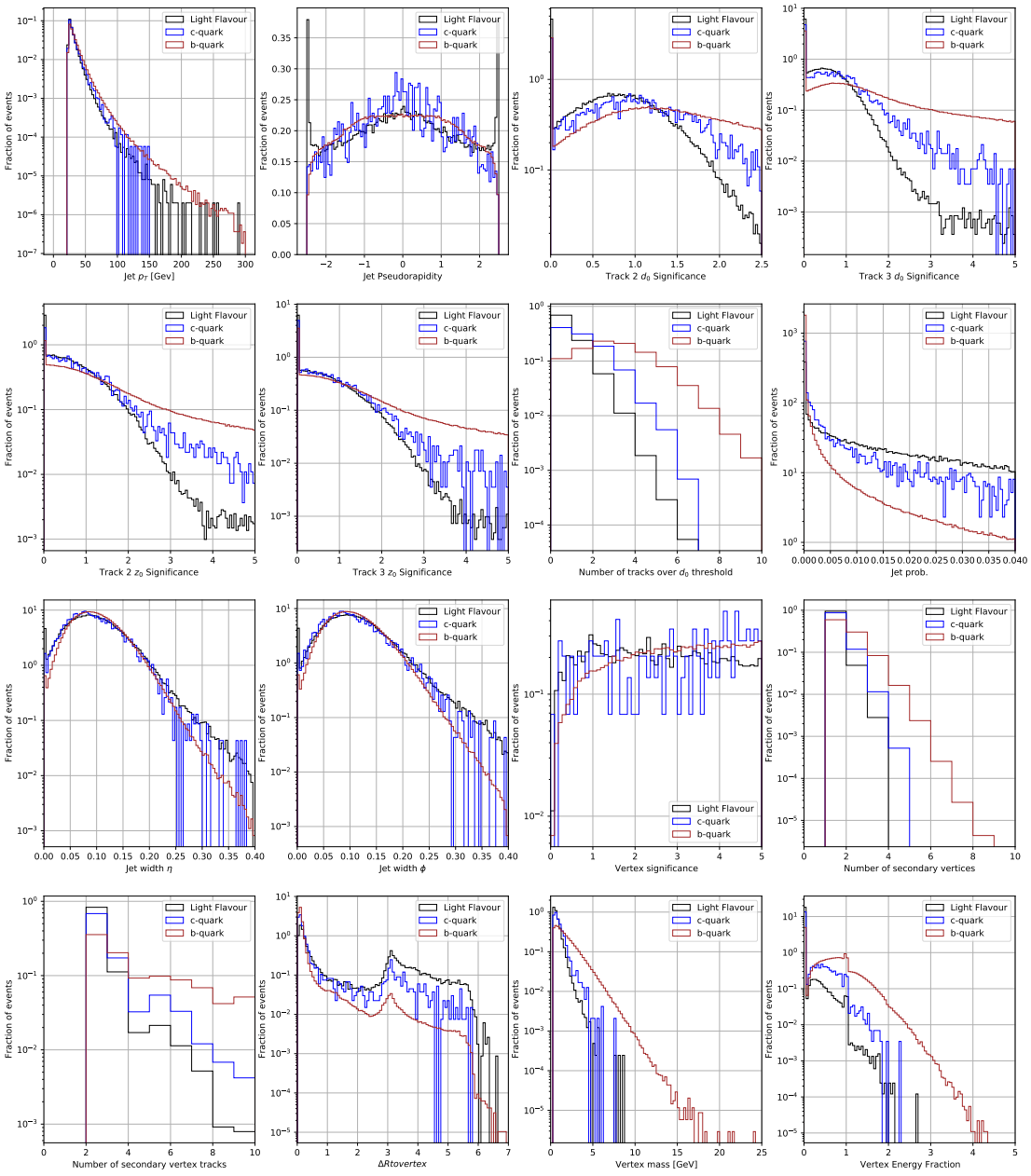

Figura 9. Istogrammi delle diverse features contenute nelle 16 colonne del dataset (eccetto la colonna dei flavor). In ciascun grafico sono presenti i tre istogrammi relativi alle tre differenti classi di quark: light, charm e bottom. Gli istogrammi sono relativi a 2 000 000 di righe del dataset, quelle utilizzate per l'allenamento della FNN.