Classificazione dei prodotti di Scattering Elettrone-Protone

Modelli di Apprendimento Supervisionato per il Machine Learning in Scikit Learn

Documentazione del progetto finale
per il corso di competenze trasvesali
PYTHON PER IL CALCOLO SCIENTIFICO

Autori:
Marco Cecca - m.cecca3@studenti.uniba.it
Giorgia Osella - g.osella@studenti.uniba.it
Vincenzo Scolletta - v.scolletta1@studenti.uniba.it

Contents

1 Abstract & Introduzione

Confronto tra diversi modelli di apprendimento supervisionato per la classificazione di particelle prodotte durante uno scattering inelastico elettrone-protone basato sulla risposta di sei diversi rilevatori. L'obiettivo è quello di identificare le particelle e valutare il migliore modello tra i seguenti:

- Decision Tree
- Random Forest
- Multilayer Perceptron
- K-Nearest Neighbor

I dati utilizzati sono il prodotto della risposta di sei diversi rilevatori mediante la piattaforma di simulazione GEANT4. Il DataSet è reperibile su Kaggle. Qui sotto la descrizione analitica del DataSet, di dimensioni iniziali (5'000'000, 7):

Features	**Significato**
id	Nome Particella
p	Quantità di moto
theta	Angolo di Scattering
beta	Rapporto tra la velocità v e c
nphe	Numero di fotoelettroni
ein	Energia in ingresso
eout	Energia in uscita

id	**Particella**	**Simbolo**
(-11)	Positroni	e^+
(211)	Pioni	$\pi \quad (\pi^0, \pi^+, \pi^-)$
(321)	Kaoni	$K \quad (K^0, K^+, K^-)$
(2212)	Protoni	p

2 Analisi Esplorativa dei Dati

Per la preparazione preliminare del dataset si sono sfruttate diverse librerie tra cui *Numpy, MatPlotLib, Pandas, Seaborn e Imbalanced Learn.*

Come primo passo lo si è reso più leggibile rinominando la colonna "**nphe**" in "**elettroni**", "**id**" in "**particella**" e abbiamo rimappato i valori delle particelle. Successivamente si è eseguita una pulizia dei dati rimuovendo la colonna "**elettroni**", per inefficienza del detector, e si sono rimosse tutte quelle particelle che riportavano un $\boldsymbol{\beta \geq 1}$, fisicamente non coerente.

Successivamente ci si è reso conto che alcune classi (*Kaoni, Positroni*) erano sottorappresentate, pertanto data la dimensione del dataset, si è effettuato un **undersampling**.

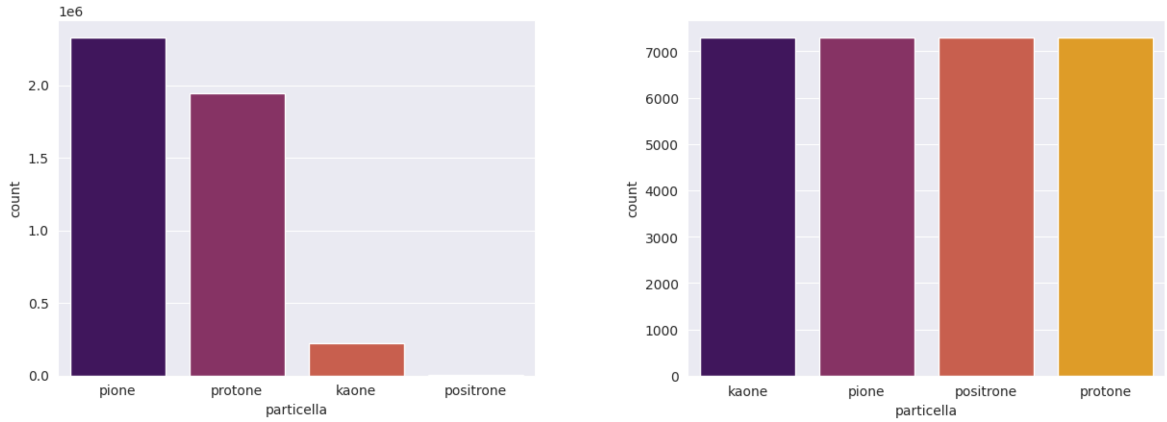

Figure 1: Prima e Dopo l'Undersampling

Infine si è analizzata una eventuale correlazione tra le features, mediante una matrice di correlazione, mostrando nessuna linearità tra le features. Invece sfruttando un modello teorico si è tracciato un *Plot Cinematico* facendo emergere una correlazione non lineare tra $\boldsymbol{\beta}$ e la quantità di moto p, mostrando come sia possibile distinguere le particelle mediante la loro massa.

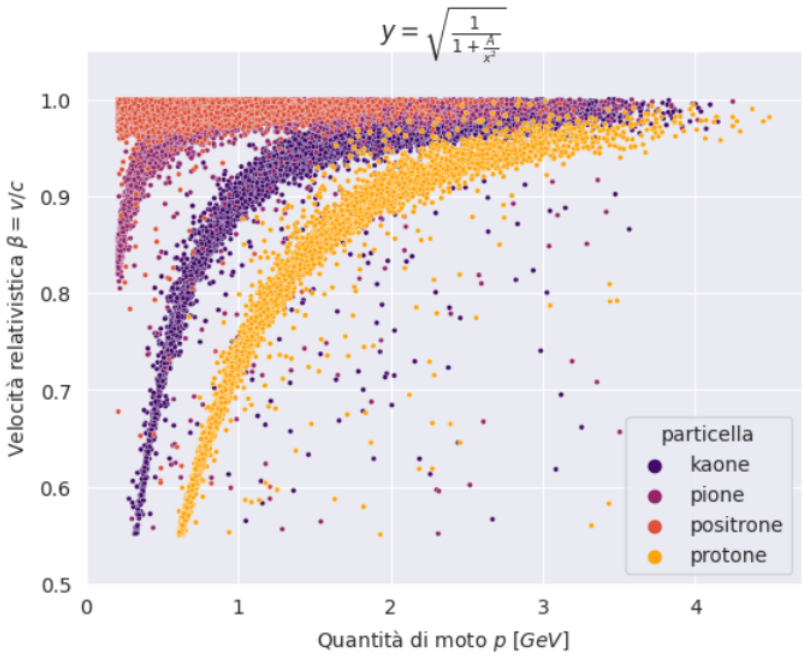

Figure 2: Plot Cinematico

Infine si è osservata la distribuzione di alcune features mediante dei *Kernel Density Estimation plot*, mostrando come la distribuzione non sia di tipo normale. Ciononostante, data la tipologia di algoritmi utilizzati, non si è effettuata nessuna rinormalizzazione.

3 Costruzione dei modelli & Addestramento

Una volta preparato e suddiviso i dati per il *training* e il *test*, si è passato alla costruzione successivo addestramento dei vari modelli di apprendimento.

- Come primo modello si è usato un **DecisionTreeClassifier**, lasciando i parametri di default.
- Come secondo modello si è usato un **RandomForestClassifier** appartenente ai *metodi ensemble*, per il quale si è valutata anche *l'importanza delle features* utilizzate per l'addestramento.
- Come terzo modello si è usata la più semplice *rete neurale*, ossia un **MultilayerPerceptronClassifier**. In questo caso si è anche utilizzata la tecnica di *ottimizzazione degli iperparametri* mediante una *GridSearchCV()*.
- Come ultimo modello si è usato un **KNeighborsClassifier**, basato sull'algoritmo *KNN* che classifica un punto nello spazio delle features attraverso la sua prossimità, e quindi distanza, rispetto ai K punti più vicini.

4 Confronto & Conclusione

Infine, per ciascuno modello, si è utilizzata come metrica per valutarne l'efficienza, l'**accuracy score** e visivamente delle **confusion matrix**. I risultati non mostrano una predominanza di un modello rispetto ad un altro, ma il *KNeighborsClassifier* è il meno efficiente.

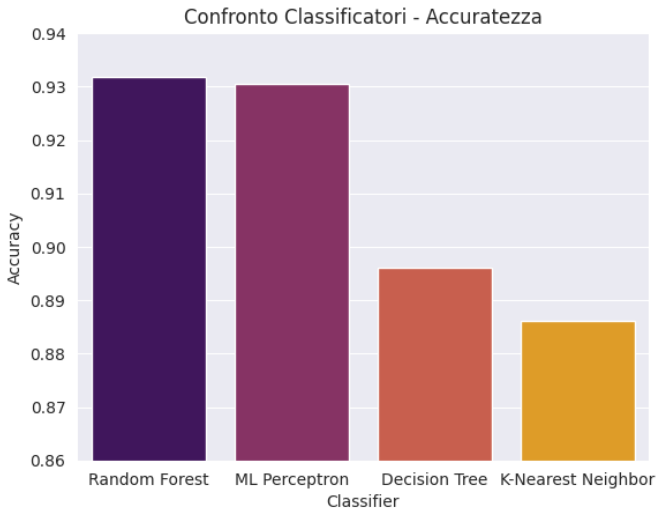

Figure 3: Confronto dell'accuratezza tra i modelli