Classificazione dei prodotti di Scattering Elettrone-Protone

Confronto tra diversi modelli di Machine Learning per la **classificazione** di particelle prodotte durante uno *scattering inelastico elettrone-protone*.

Autori

Marco Cecca
Giorgia Osella
Vincenzo Scolletta

Composizione del Dataset

I dati utilizzati sono il prodotto della risposta di sei diversi rilevatori, usati mediante la piattaforma di simulazione *GEANT4*. Il DataSet è reperibile su *Kaggle*, avente una dimensionalità di (5'000'000, 7)

	id	p	theta	beta	nphe	ein	eout
0	211	0.780041	1.081480	0.989962	0	0.000000	0.000000
1	211	0.260929	0.778892	0.902450	0	0.000000	0.000000
2	2212	0.773022	0.185953	0.642428	4	0.101900	0.000000
3	211	0.476997	0.445561	0.951471	0	0.000000	0.000000
4	2212	2.123290	0.337332	0.908652	0	0.034379	0.049256

Features	Significato	Unità di Misura
id	Nome Particella	$NoDim$
p	Quantità di moto	GeV/c
theta	Angolo di Scattering	rad
beta	Rapporto tra la velocità v e c	$NoDim$
nphe	Numero di fotoelettroni	$NoDim$
ein	Energia in ingresso	GeV
eout	Energia in uscita	GeV

id	Particella
(-11)	Positroni
(211)	Pioni
(321)	Kaoni
(2212)	Protoni

Analisi Esplorativa e Preparazione dei Dati

Per una maggiore leggibilità sono state rinominate la colonna ***id*** e ***nphe***.

Inoltre il *codice* associato a ciascuna particella è stato sostituito con il suo ***nome*** rimappando i valori.

Il dataframe è stato sottoposto ad un controllo di eventuali features mancanti.

Ci sono cinque milioni di particelle, ciascuna con sei features e nessun dato è mancante.

	particella	p	theta	beta	elettroni	ein	eout
0	pione	0.780041	1.081480	0.989962	0	0.000000	0.000000
1	pione	0.260929	0.778892	0.902450	0	0.000000	0.000000
2	protone	0.773022	0.185953	0.642428	4	0.101900	0.000000
3	pione	0.476997	0.445561	0.951471	0	0.000000	0.000000
4	protone	2.123290	0.337332	0.908652	0	0.034379	0.049256
...	...	...	...	...	...	...	...

```
print(df.isnull().sum())
print("-"*20)
print(df.shape)
```

```
particella    0
p             0
theta         0
beta          0
elettroni     0
ein           0
eout          0
dtype: int64
--------------------
(5000000, 7)
```

Analisi Esplorativa e Preparazione dei Dati

Si è osservato che il detector per gli *elettroni* è stato inefficiente (segna spesso 0). Verificata questa ipotesi (manca il segnale il **93,52%** delle volte), si è eliminata la colonna corrispondente.

```python
flop = df['elettroni'].value_counts(normalize = True)
print('Il rilevatore manca il segnale il {volte:.2%} delle volte'.format(volte=flop[0]))
```

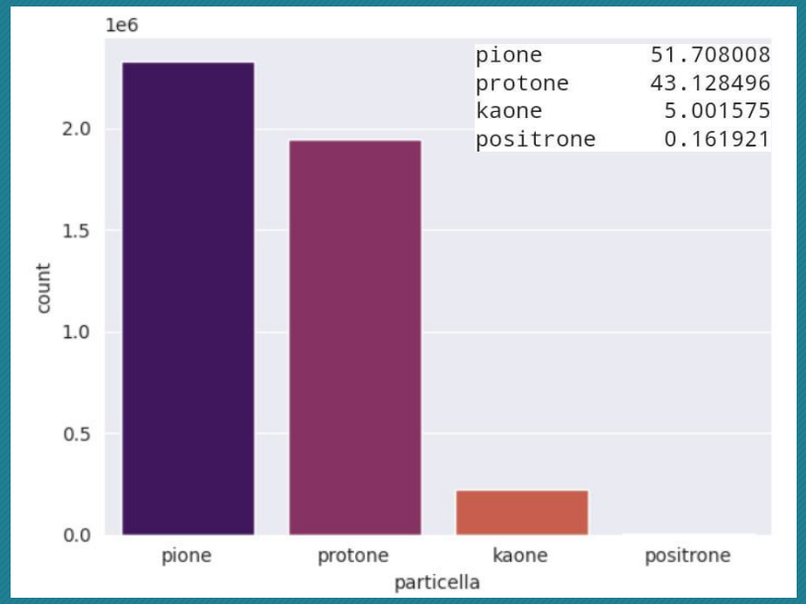
Python

```
Il rilevatore manca il segnale il 93.52% delle volte
```

Si sono rimosse anche tutte le particelle che riportavano un ***β ≥ 1***, ***fisicamente non coerente.***

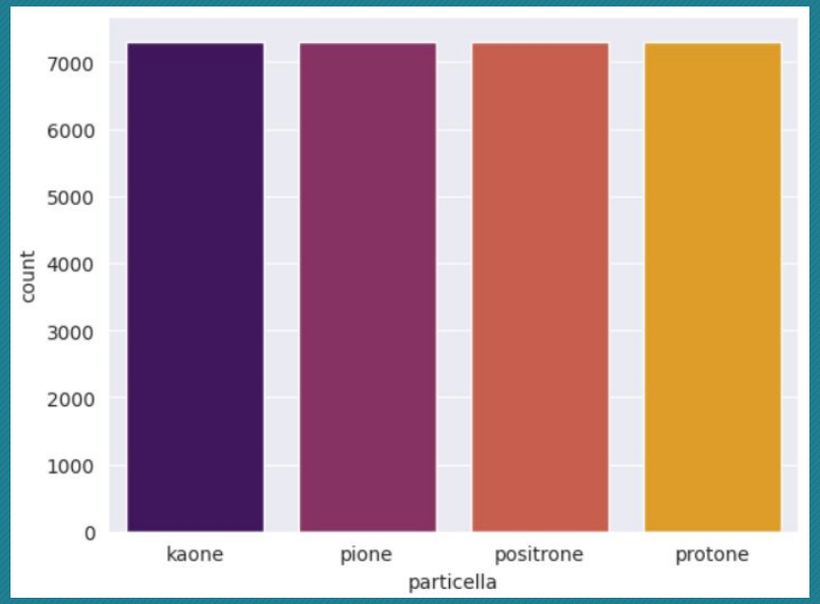
```python
df = df.drop(df[df.beta >= 1].index)
df.head()
```

Analisi Esplorativa e Preparazione dei Dati



Analizzando i dati si nota che i **positroni** e **kaoni** sono classi *sottorappresentate*.

È stato effettuato un **resampling** del dataset per non avere un dataset sbilanciato.

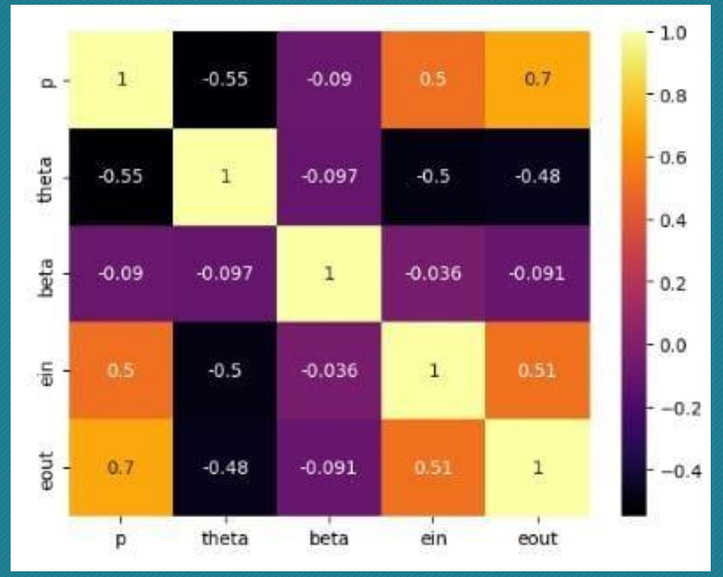


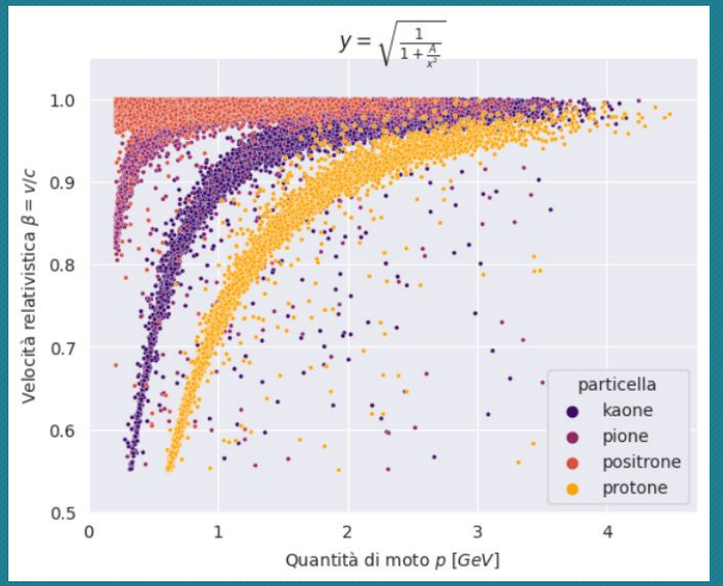

```
x = pd.DataFrame(df)
y = x['particella']

undersample = RandomUnderSampler(sampling_strategy='not minority')
df_us, _ = undersample.fit_resample(x, y)
```

In particolare è stato effettuato un **undersampling**, che ha ridotto il dataset del 99,35%

Analisi Esplorativa e Preparazione dei Dati





A livello teorico è stata considerata la correlazione, non lineare, tra ***p*** e ***β*** e visualizzata in un grafico.

Il comportamento delle diverse particelle è distinguibile in base alle loro *masse*

$$m_{e^+} < m_\pi < m_K < m_p$$

È stata analizzata la correlazione tra le features tramite una *matrice di correlazione*.

L'unica correlazione lineare significativa è tra la *quantità di moto* e l'*energia in uscita*.

```
with sns.axes_style("darkgrid"):
    sns.scatterplot(data=df_us, x='p', y='beta', hue='particella',
        s = 7, palette = 'inferno')
```

Decision Tree

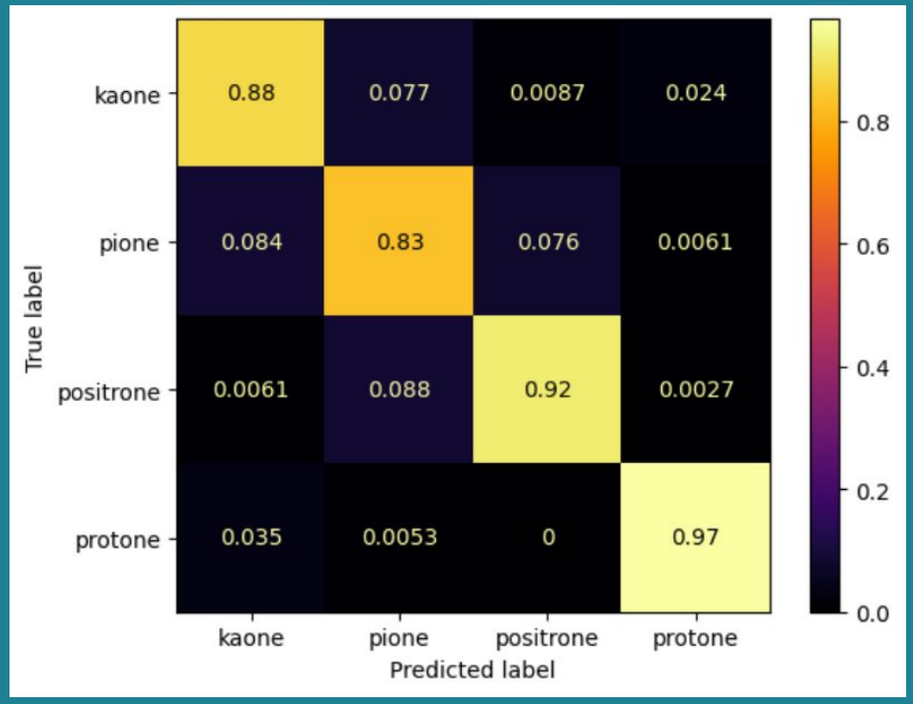
Come primo modello si è usato un ***DecisionTreeClassifier*** lasciando i parametri di default.

L'addestramento è compiuto sull'**80%** del dataset. È stata valutata l'efficienza del metodo tramite una matrice di confusione e l'accuracy score.

Gli **alberi decisionali** creano un modello che predice una classe o un valore in output a partire da regole di tipo *binario* inferite dalle *feature* dei dati.

```
dt_class = DecisionTreeClassifier(random_state=2022)
dt_class.fit(x_train, y_train)
y_pred_dt = dt_class.predict(x_test)
dt_accuracy = accuracy_score(y_test, y_pred_dt)
```

```
Accuratezza DecisionTree:
89.61%
```



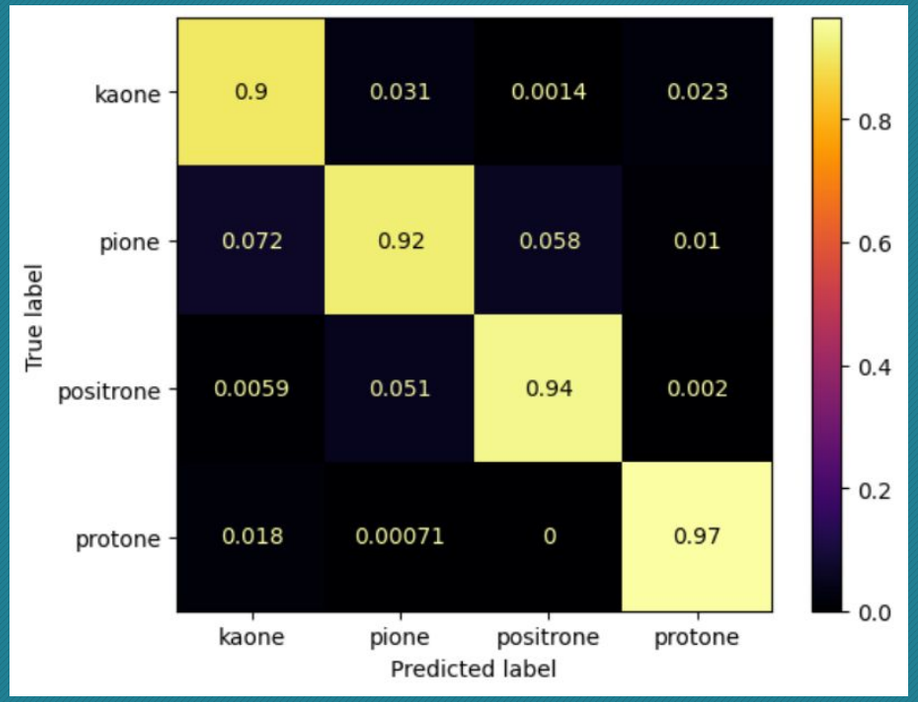
Random Forest

Come secondo modello si è usato un ***RandomForestClassifier***.

Un **metodo ensemble** basato su *alberi decisionali*. Ossia combina diversi algoritmi, ottenendo in generale risultati migliori.

```
rf_class = RandomForestClassifier(n_estimators=100,
                                  n_jobs=-1,
                                  random_state=2022)
rf_class.fit(x_train, y_train)
y_pred_rf = rf_class.predict(x_test)
rf_accuracy = accuracy_score(y_test, y_pred_rf)
```

```
Accuratezza RandomForest:
93.19%
```



Random Forest

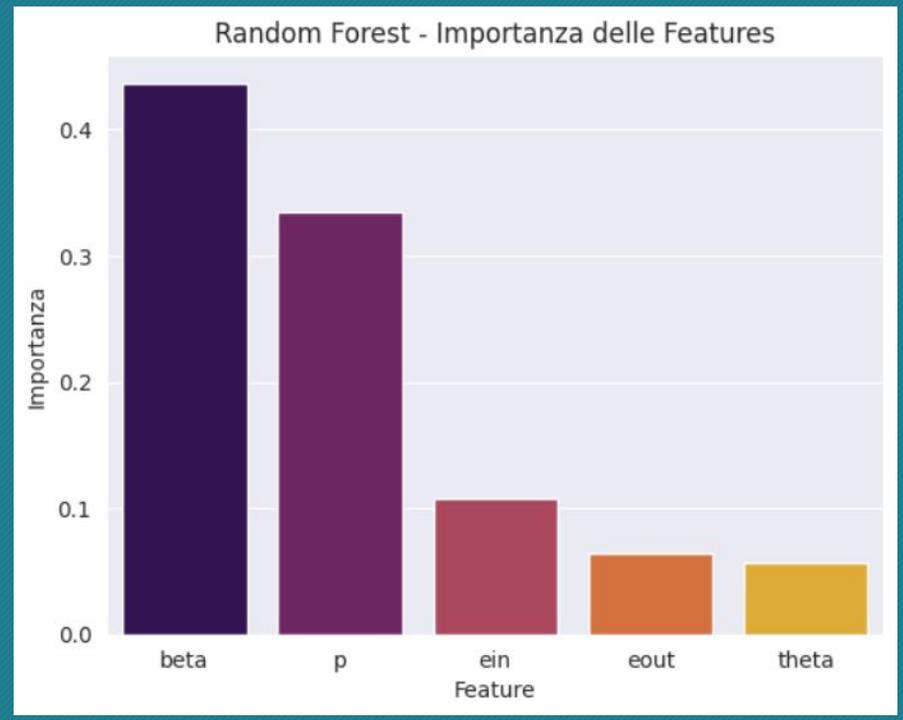
È stata valutata anche l'**importanza delle features** associata dal modello per la classificazione delle particelle.

Per farlo è stato utilizzato l'attributo ***feature_importances_***

Le features maggiormente importanti per l'addestramento del modello, e la successiva predizione, sono ***p*** e la ***β***.

Era prevedibile dal plot cinematico, attraverso il quale è possibile distinguere, e quindi classificare, le varie particelle.

```
importanza = pd.DataFrame(data={'Feature': x_train.columns,
                                'Importanza': rf_class.feature_importances_})
importanza.head()
```



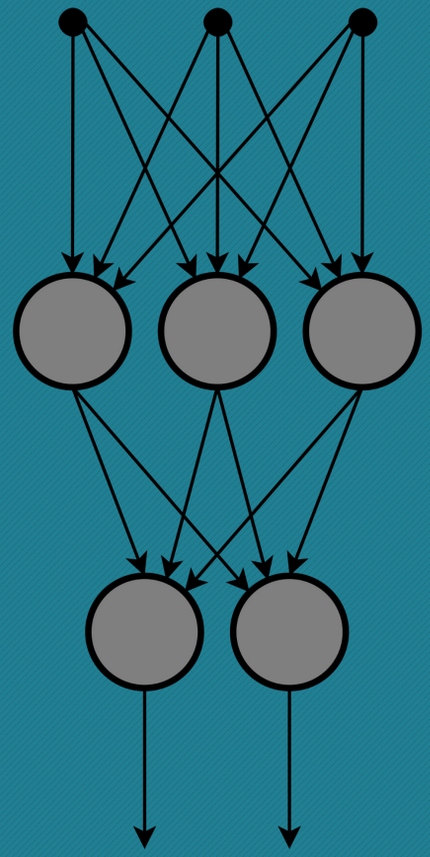
Multilayer Perceptron

Come terzo modello è stato utlizzato un ***MLPclassifier***

Un **Multilayer Perceptron** è il più semplice modello di **rete neurale** che è possibile concepire.

Il *layer di imput* è costituito nel nostro caso da 5 **neuroni**, mentre quello di *output* da 4, nel mezzo ci sono i *layers nascosti*

Ogni neurone nel layer nascosto trasforma i valori del layer precedente con una *sommatoria pesata* seguita da una **funzione di attivazione**



Multilayer Perceptron

È stata utilizzata una tecnica di **ottimizzazione degli iperparametri**, mediante l'oggetto ***GridSearchCV()***

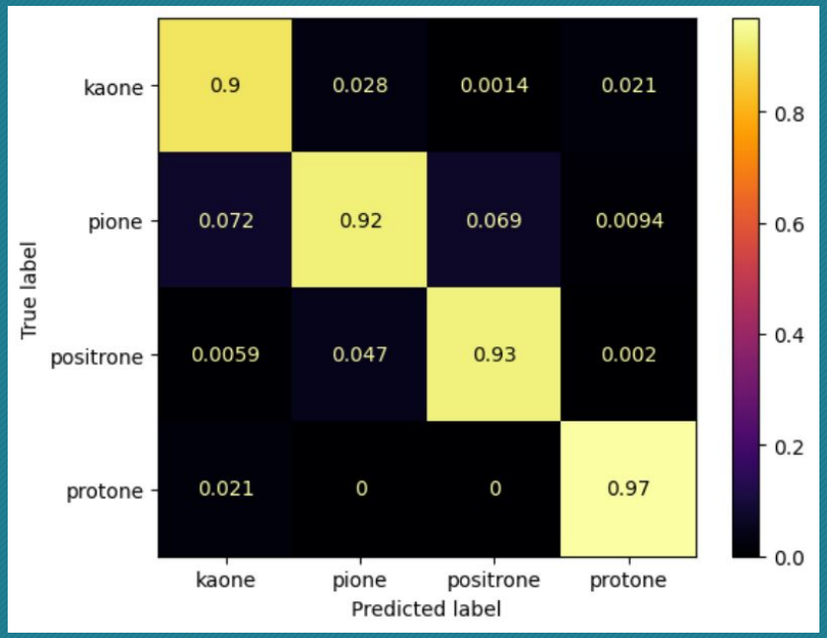
La *grid search* prova ad eseguire l'algoritmo per ogni combinazione di parametri all'interno di una *griglia data*, fino a trovarne la migliore possibile.

```
mlp_class = MLPClassifier(max_iter=300, random_state=2022)
mlp_upgrade = GridSearchCV(mlp_class, griglia_parametri, n_jobs=-1)
mlp_upgrade.fit(x_train, y_train)
```

```
griglia_parametri = [

        {
            'activation' : ['logistic', 'tanh', 'relu'],
            'solver' : ['lbfgs', 'sgd', 'adam'],
            'hidden_layer_sizes': [(1,),(5,),(10,),(100,)]
        }
       ]
```

```
Accuratezza RandomForest:
93.05%
```




```
I migliori parametri per il MLPClassifier sono:
{'activation': 'relu', 'hidden_layer_sizes': (100,), 'solver': 'adam'}
```

K-Nearest Neighbor

Come quarto e ultimo modello è stato utilizzato il ***KNeighborsClassifier***.

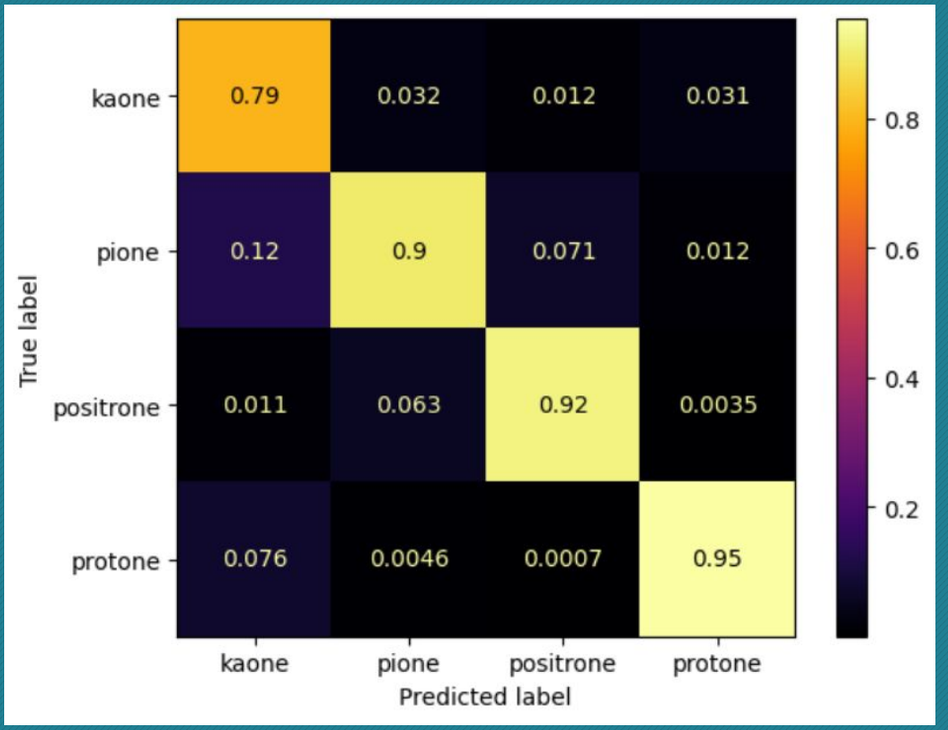
Quest'ultimo classificatore si basa sull'*algoritmo KNN* che classifica un punto nello spazio delle features attraverso la sua prossimità' rispetto ai K punti più vicini.

La *distanza* viene valutata mediante diverse metriche, tra cui quella che è stata usata, la distanza di Minkowski.

```
knn_class = KNeighborsClassifier()
knn_class.fit(x_train, y_train)
y_pred_knn = knn_class.predict(x_test)
knn_accuracy = accuracy_score(y_test, y_pred_knn)
```

$$\left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{1/p}$$

```
Accuratezza K-Nearest Neighbor:
88.60%
```



Confronto finale tra i metodi

I modelli di classificazione sono stati messi a confronto attraverso la loro **accuratezza**

Da cui si può concludere, considerando l'accuratezza dei modelli, che si comportano tutti molto bene, ma il meno efficiente è il modello basato sull'algoritmo del *K-Nearest Neighbor*.

	Classifier	Accuracy
0	Decision Tree	0.896133
1	Random Forest	0.931896
2	ML Perceptron	0.930527
3	K-Nearest Neighbor	0.886037