CAPRICANO Davide (0512110422)
GIARDINETTO Giuseppe (0512114655)
SPERA Alfredo Jeshoua (0512112240)

Comparazione sistemi di machine learning per la categorizzazione di un set di corpi celesti

Progetto di Machine Learning, A/A 2023/24 Università degli studi di Salerno

Sommario

1. Obiettivo Progetto

L'obiettivo di questo progetto è quello di comparare la precisione di tre modelli nel task di categorizzazione *multiclasse a bassa cardinalità*. Più nello specifico, i modelli dovranno riuscire a predire correttamente una categoria per un corpo celeste osservato, questa definisce la natura del corpo.

L'apprendimento di questi modelli è del tipo supervisionato di tipo Batch Learning, le categorie fornite ai modelli sono di tre tipi:

- Galaxy
- Quasar
- Star

I modelli utilizzati in questo progetto saranno:

- K-Nearest Neighbor (KNN)
- Decision Tree
- Random Forest

2. Descrizione dataset

Il dataset utilizzato (https://www.kaggle.com/datasets/fedesoriano/stellar-classification-dataset-sdss17) è in formato CSV.

I dati rappresentano i valori spettrali dei corpi osservati, questi descrivono con precisione la luce emessa e vengono analizzati per la scoperta e l'analisi di fenomeni in astrofisica.

```
import pandas as pd

# https://www.kaggle.com/datasets/fedesoriano/stellar-classification-dataset-sdss17
df = pd.read_csv('star_classification.csv')
```

2.1. Caratteristiche

Il dataset è composto da 100.000 righe e 18 colonne, dieci delle quali sono float, sette sono interi e una è di stringhe.

Colonna	Tipo	Descrizione
obj_ID	float64	Object Identifier, il valore univoco che identifica l'oggetto nel catalogo delle immagini utilizzato dal CAS (Center for Astrophysical Sciences)
alpha	float64	Angolo di ascensione retta (all'epoca J2000)
delta	float64	Angolo di declinazione (all'epoca J2000)
u	float64	Filtro ultravioletto nel sistema fotometrico
g	float64	Filtro verde nel sistema fotometrico
r	float64	Filtro rosso nel sistema fotometrico
i	float64	Filtro nel vicino infrarosso del sistema fotometrico
z	float64	Filtro infrarosso nel sistema fotometrico
run_ID	int64	Numero identificativo dell'esecuzione

rereun_ID	int64	Numero identificativo dell'esecuzione per specificare come è stata elaborata l'immagine
cam_col	int64	Colonna della telecamera per identificare la linea di scansione all'interno dell'esecuzione
field_ID	int64	Numero di campo per identificare ciascun campo
spec_obj_ID	float64	ID univoco utilizzato per gli oggetti spettroscopici ottici (ciò significa che due osservazioni diverse con lo stesso `spec_obj_ID` devono condividere la classe di output)
class	object (stringa)	Classe dell'oggetto (Galassia, Stella o Quasar)
redshift	int64	Valore di redshift basato sull'aumento della lunghezza d'onda
plate	int64	ID della piastra, che identifica ciascuna piastra in SDSS
MJD	int64	Modified Julian Date (data giuliana modificata), utilizzata per indicare quando un dato di dati SDSS è stato rilevato
fiber_ID	int64	ID della fibra che identifica la fibra che ha puntato la luce sul piano focale in ogni osservazione

2.2. Valori mancanti

Il dataset risulta privo di valori mancanti, in quanto frutto di rilevazioni di strumenti elettronici, curate dall'osservatorio astronomico APO (Apache Point Observatory), per conto della SDSS (Sloan Digital Sky Survey).

```
df.isnull().sum()
```

Column	Null Count
obj_ID	0
alpha	0
delta	0
u	0
g	0
r	0
i	0
z	0
run_ID	0
rereun_ID	0
cam_col	0
field_ID	0
spec_obj_ID	0
class	0
redshift	0
plate	0
MJD	0
fiber_ID	0

2.3. Analisi dei dati

Di seguito sono elencati i dati che descrivono statisticamente il dataset, divisi per colonne (esclusa *class*).

```
df.describe().to_excel()
```

	obj_ID	alpha	delta	u	g	r	i	z	run_ID	rerun_ID	cam_col	field_ID	pec_obj_ID	redshift	plate	MJD	fiber_ID
count	100000	100000	100000	100000	100000	100000	100000	100000	100000	100000	100000	100000	100000	100000	100000	100000	100000
mean	1,24E+18	177,6291	24,1353	21,98047	20,53139	19,64576	19,08485	18,66881	4481,366	301	3,51161	186,1305	5,78E+18	0,576661	5137,01	55588,65	449,3127
std	8,44E+12	96,50224	19,64467	31,76929	31,75029	1,85476	1,757895	31,72815	1964,765	0	1,586912	149,0111	3,32E+18	0,730707	2952,303	1808,484	272,4984
min	1,24E+18	0,005528	-18,7853	-9999	-9999	9,82207	9,469903	-9999	109	301	1	11	3E+17	-0,00997	266	51608	1
25%	1,24E+18	127,5182	5,146771	20,35235	18,96523	18,13583	17,73229	17,46068	3187	301	2	82	2,84E+18	0,054517	2526	54234	221
50%	1,24E+18	180,9007	23,64592	22,17914	21,09984	20,12529	19,40515	19,0046	4188	301	4	146	5,61E+18	0,424173	4987	55868,5	433
75%	1,24E+18	233,895	39,90155	23,68744	22,12377	21,04479	20,3965	19,92112	5326	301	5	241	8,33E+18	0,704154	7400,25	56777	645
max	1,24E+18	359,9998	83,00052	32,78139	31,60224	29,57186	32,14147	29,38374	8162	301	6	989	1,41E+19	7,011245	12547	58932	1000

2.4. Distribuzione delle classi interessate

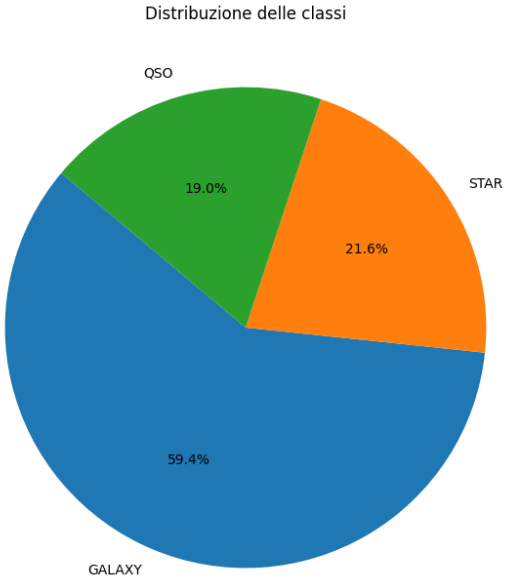
Le classi all'interno della colonna *class* hanno la seguente distribuzione.

```
class_counts = df['class'].value_counts()
```

Classe	Count
GALAXY	59445
STAR	21594
QSO	18961

```
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 8))
plt.pie(class_counts, labels=class_counts.index, autopct='%1.1f%%',
startangle=140)
plt.title('Distribuzione delle classi')
plt.show()
```



3. Data engineering

3.1. Ridenominazione nome colonne

Si è reso necessario, data la formattazione dei dati, di effettuare modifiche al nome delle feature rendendole più maneggevoli e comprensibili durante la revisione e la scrittura del codice.

Per questo sono state effettuate delle operazioni di ridenominazione sulle colonne.

Vecchio nome	Nuovo nome
u	***ultraviolet filter***
g	***green filter***
r	***red filter***
i	***near infrared filter***
z	***infrared filter***

3.2. Feature Engineering

Per il task di classificazione obiettivo del progetto, si ritiene necessario eliminare diverse colonne riguardanti gli identificatori e le date, che inevitabilmente influirebbero negativamente sulla capacità dei modelli di generalizzare e quindi di inferire.

Le colonne selezionate per la *rimozione* sono: *obj_ID*, *run_ID*, *rerun_ID*, *cam_col*, *field_ID*, *spec_obj_ID*, *plate*, *MJD*, *fiber_ID*.

Lasciando come feature del dataset solo quelle interessate, ovvero: ***alpha***, ***delta***, ***ultraviolet filter***, ***green filter***, ***red filter***, ***near infrared filter***, ***infrared filter***, ***class***, ***redshift***.

Queste specifiche informazioni descrivono lo spettro luminoso del corpo osservato.

3.3. Normalizzazione

È possibile affermare che *non sia stata riscontrata la necessità di effettuare un'operazione di normalizzazione*, in quanto i dati non presentano grossi ordini di grandezza differenti tra loro

3.4. Split in set di Training e di Test

Per l'impiego di questi dati in un modello di machine learning, è necessario distinguerli in dati di training e dati di test, e, per permettere a questi di avere una capacità di generalizzazione ampia si è scelto di suddividere i dati in un 80% dati di training e 20% di training.

```
X = df.drop('class', axis=1) # feature
y = df['class']              # target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=30)
```

Si definiscono quindi `X` come variabili indipendenti, e `y` come variabile dipendente, ovvero riuscire a inferire la classe di un corpo celeste nel dataset.

4. Modelli

Come detto in precedenza, i modelli scelti per lo svolgimento del progetto sono:

- K-Nearest Neighbor (KNN)
- Decision Tree
- Random Forest

La scelta di questi modelli è basata sulla loro grande diffusione nella letteratura e nell'impiego ottimale nei task di classificazione.
Per ogni modello verrà indicato il tempo di addestramento e le seguenti metriche di prestazione:

- Precision
- Recall
- F1-Score
- Accuracy

Le prestazioni verranno calcolate utilizzando la funzione `classification_report` di `Sklearn`. Questi dati verranno usati e analizzati poi nel capitolo "Comparazione modelli".

4.1. K-Nearest Neighbor

Il modello K-Nearest Neighbor (KNN) effettua inferenze basandosi sulle *k* istanze vicine. La sua predizione è quindi basata sulle metriche statistiche note in letteratura, ovvero la ***Metrica di Minkowski*** oppure la ***Distanza di Manhattan***.

In Python è possibile realizzare un modello di KNN grazie alla funzione *`KNeighborClassifier`* della libreria *`sklearn.neighbors`*.

È necessario definire un numero intero *k* appropriato per evitare casi di *k* troppo grandi e quindi di conseguente underfitting, ma allo stesso tempo è necessario evitare di impostare *k* troppo piccoli che possono causare overfitting.

Viene scelto *k=3* come conseguenza di varie iterazioni, in quanto risulta essere il numero più appropriato per produrre inferenze con la massima precisione.

```
ORDINATI PER ACCURACY DECRESCENTE
neighbors  accuracy  training_time
        3   0.83730       1.156744
        5   0.83310       1.213075
        1   0.83175       1.052180
        4   0.83175       1.185075
        7   0.83020       1.164158
        6   0.82955       1.163620
        ...
```

```
model_knn = KNeighborsClassifier(n_neighbors=3)
model_knn.fit(X_train, y_train)
y_pred_knn = model_knn.predict(X_test)
```

Il modello ottiene le seguenti prestazioni:

	Precision	Recall	F1-Score
GALAXY	0.83	0.95	0.89
QSO	0.85	0.81	0.83
STAR	0.79	0.53	0.64

Accuracy	Training Time
0.83	1.13 s

4.2. Decision Tree

Il modello Decision Tree effettua inferenze basandosi sulle divisioni del dataset tramite dei nodi che sfruttano un criterio di split. Ogni inferenza è frutto di vari split e topologicamente rappresentano un nodo foglia dell'albero costruito dagli split e dai cammini.

Il Decision Tree è uno degli algoritmi più "semplici" ma al tempo stesso molto potenti e studiati per via della sua capacità di prestarsi a molti problemi. Alcuni dei problemi del decision tree possono essere l'estrema profondità dell'albero che causerebbe un overfitting e il modello sarebbe capace di inferire correttamente solo i dati di training e non quelli di testing. In caso opposto, una profondità molto breve potrebbe portare a casi di underfitting.

In Python è possibile realizzare un modello di Decision Tree tramite la funzione `DecisionTreeClassifier` di `Sklearn.tree`

```
model_dt = DecisionTreeClassifier()
model_dt.fit(X_train, y_train)
y_pred_dt = model_dt.predict(X_test)
```

Il modello ottiene le seguenti prestazioni:

	Precision	Recall	F1-Score
GALAXY	0.97	0.97	0.97
QSO	0.91	0.92	0.92
STAR	1.00	1.00	1.00

Accuracy	Training Time
0.97	2.51 s

4.3. Random Forest

La Random Forest è un modello di tipo ***Ensamble***, infatti questo è composto da "più modelli", in questo caso più alberi decisionali. Il paradigma della random forest è il ***Bagging*** (o bootstrap aggregating): questo consiste nel campionare il dataset, e, sui campioni prodotti addestrare un singolo classificatore (nel caso della random forest un albero decisionale).

Per i random forest è inoltre implementata la tecnica della feature randomness, ovvero la tecnica che prevede che ogni singolo classificatore possa scegliere randomicamente solo da un sottoinsieme di feature da utilizzare per la previsione.

Nel caso dell'ensamble random forest è necessario specificare quanti estimatori devono essere utilizzati. Dopo vari test si è riscontrato che in questo caso il miglior numero di estimatori è *29*.

ORDINATI PER ACCURACY DECRESCENTE

estimators	accuracy	training_time
29	0.97880	13.844527
28	0.97880	13.422939
18	0.97875	8.813855
30	0.97875	14.341758
27	0.97875	13.216317
25	0.97875	12.093091
...		

In Python è possibile realizzare un modello Random Forest tramite la funzione `RandomForestClassifier` della libreria `sklearn.ensamble`:

```
model_rf = RandomForestClassifier(n_estimators=29, random_state=30)
model_rf.fit(X_train, y_train)
y_pred_rf = model_rf.predict(X_test)
```

Il modello ottiene le seguenti prestazioni:

	Precision	Recall	F1-Score
GALAXY	0.98	0.98	0.98
QSO	0.96	0.94	0.95
STAR	0.99	1.00	1.00

Accuracy	Training Time
0.98	13.84 s

5. Comparazione Modelli

Riportiamo i dati calcolati nel capitolo precedente in una tabella riassuntiva:

	KNN			Decision Tree			Random Forest		
	Precision	*Recall*	*F1-Score*	*Precision*	*Recall*	*F1-Score*	*Precision*	*Recall*	*F1-Score*
GALAXY	0.83	0.95	0.89	0.97	0.97	0.97	0.98	0.98	0.98
QSO	0.85	0.81	0.83	0.91	0.92	0.92	0.96	0.94	0.95
STAR	0.79	0.53	0.64	1.00	1.00	1.00	0.99	1.00	1.00

	KNN	Decision Tree	Random Forest
Accuracy	0.83	0.97	0.98
Training Time	1.13 s	2.51 s	13.84 s

Possiamo notare come il KNN possegga il training time più basso, a scapito però di un'accuracy inferiore agli altri modelli impiegati.

Analogamente possiamo notare come la Random Forest riesca effettivamente a migliorare l'accuracy totale e contestualmente anche la capacità di generalizzare sulle classi di tipo GALAXY e di tipo QSO, questo avviene a scapito di un tempo di addestramento di molto superiore agli altri modelli.

Un modello che offre un ottimo trade-off è il decision tree che riesce ad ottenere un'accuracy totale molto alta incidendo "poco" sul tempo di addestramento.

La scelta di un modello ottimale in questo caso è questione di etica, poiché potrebbe essere necessario un modello che sia capace di avere la massima accuracy nonostante il grande dispendio di tempo per l'addestramento.

In molti casi risulta essenziale massimizzare la precisione e la capacità di generalizzazione del modello piuttosto che la sua velocità, e in tali casi il modello migliore tra quelli analizzati risulterebbe il Random Forest, mentre in altri contesti risulterebbe più importante la scelta di un modello facilmente impiegabile e veloce da addestrare e mettere in funzione nonostante la perdita di capacità di generalizzazione (purché questa non sia ingente), in questi ultimi casi la scelta del miglior modello ricadrebbe sul decision tree.

È necessario inoltre effettuare test sull'affidabilità del modello, sfruttando metriche più complesse della sola accuracy. Si introduce quindi la ***Cross-Entropy Loss Function***, questa è definita come la misurazione della distanza tra due distribuzioni di probabilità. In particolare, misura la sommatoria delle probabilità di ciascun evento nella distribuzione dei dati reali (ovvero le classi), moltiplicate per il logaritmo delle probabilità di ciascun evento nella distribuzione prevista. La sua utilità è principalmente calcolare statisticamente la "bontà" del modello.

In Python è possibile calcolare la Cross Entropy tramite la funzione `log_loss` della libreria `sklearn.metrics`.

KNN:

```
log_loss(y_test, y_pred_probs_knn)
```

Decision Tree:

```
log_loss(y_test, y_pred_probs_dt)
```

Random Forest:

```
log_loss(y_test, y_pred_probs_rf)
```

I parametri utilizzati per questa funzione sono i dati di test (`y_test`) e le probabilità delle inferenze prodotte dai singoli modelli (*`y_pred_probs_knn`, `y_pred_probs_dt`, `y_pred_probs_rf`*)

I risultati prodotti da ogni modello sono:

	KNN	Decision Tree	Random Forest
Cross Entropy	2.746010559525009	1.207462388535425	0.16813876924269983

È possibile notare come, nonostante una accuracy molto simile tra loro, i modelli Decision Tree e Random Forest hanno una forte differenza nella cross entropy.