Preprocessing delle features dell'eye tracker

Preprocessing: definizione

- Il preprocessing è la fase in cui si preparano e si organizzano i dati, prima di far partire l'algoritmo di apprendimento;
- Avviare il processo di machine learning su dati grezzi implica il rischio di prolungare l'apprendimento automatico su dati ridondanti. Perciò, si analizza il dataset per individuare eventuali correlazioni nei dati, riducendo, così, la complessità computazionale.

Preprocessing: passaggi

1. Importazione delle librerie più comunemente utilizzate per il preprocessing: nello specifico, sono state importate:
   - pyplot, da matplotlib;
   - pandas;
   - seaborn;
   - PCA e LabelEncoder, da sklearn.

```
# Librerie per il preprocessing
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.decomposition import PCA
from sklearn.preprocessing import LabelEncoder
```

Preprocessing: passaggi

2. Importazione e copia del dataset contenente tutte le variabili misurate mediante l'eye tracker, vale a dire sia le features sia il target, e sostituzione del carattere "," presente nel dataset con il carattere ".", per permettere la lettura dei numeri reali, inizialmente non riconosciuti;

```
df_originale = pd.read_csv('Dataset eye tracker.tsv', delimiter='\t')     # Lettura del dataset sull'eye tracker
df = df_originale.copy()                                                  # Copia del dataset, utilizzata per il preprocessing
df = df.replace(',', '.', regex=True)       # Sostituzione delle virgole nel dataset con punti, per permettere la lettura dei valori float
```

Preprocessing: passaggi

3. Visualizzazione della dimensione, di un riepilogo e delle prime cinque righe del dataset: le funzioni utilizzate sono, rispettivamente, “shape”, “info”, avente il parametro “verbose=True” per un riepilogo dettagliato del dataset, e “head”;

```
df.shape                    # Visualizzazione del numero di righe e colonne del dataset
df.info(verbose=True)       # Visualizzazione delle informazioni sul dataset, sulle proprie colonne e sui relativi valori
df.head()                   # Visualizzazione delle prime cinque righe del dataset
```

Preprocessing: passaggi

4. Modifica delle colonne di tipo “object” in colonne di tipo “float” e rimozione di quelle non convertibili;

```python
colonne_object_originali = df.select_dtypes(include=['object']).columns.tolist()                                # Selezione delle colonne di tipo 'object'
colonne_da_escludere = ['Task', 'Participant']                                                                   # Esclusione delle colonne 'Task' e 'Participant'
colonne_object = [colonna for colonna in colonne_object_originali if colonna not in colonne_da_escludere]       # Filtraggio delle colonne 'object' escludendo quelle specificate
colonne_eliminate = []                                                                                           # Creazione elenco delle colonne da eliminare

# Visualizzazione dell'elenco originale delle colonne 'object'
print("Colonne 'object' iniziali:")
for i in range(len(colonne_object_originali)):
    print("-",colonne_object_originali[i])

# Conversione delle colonne e gestione degli errori
for colonna in colonne_object:
    try:
        df[colonna] = df[colonna].astype(float)       # Conversione delle colonne in float
    except ValueError:
        colonne_eliminate.append(colonna)             # Aggiunta della colonna tra quelle da eliminare
        df = df.drop(colonna, axis=1)                 # Eliminazione della colonna

# Visualizzazione dell'elenco delle colonne eliminate
print("\nColonne eliminate:")
print(colonne_eliminate)

colonne_object_dopo_conversione = df.select_dtypes(include=['object']).columns.tolist()       # Creazione dell'elenco delle colonne aventi ancora tipo 'object'

# Visualizzazione dell'elenco finale delle colonne 'object' dopo la conversione
print("\nColonne 'object' dopo la conversione:")
print(colonne_object_dopo_conversione)
```

Preprocessing: passaggi

5. Ricerca ed eliminazione delle colonne aventi valori mancanti e con tutti valori 0;

```python
valori_mancanti = df.isnull().sum()       # Controllo della presenza di valori mancanti nelle colonne nel dataset e ne specifica il numero
colonne_con_valori_mancanti = valori_mancanti[valori_mancanti > 0]      # Filtraggio delle sole colonne con valori mancanti

# Visualizzazione dell'elenco dei valori mancanti totali per ogni colonna
if not colonne_con_valori_mancanti.empty:
    print("Elenco delle colonne contenenti valori mancanti e totale valori mancanti:")
    for colonna, contatore in colonne_con_valori_mancanti.items():
        print(f"- {colonna}: {contatore}")
else:
    print("Nessun valore mancante trovato")

colonne_con_tutti_valori_mancanti = df.columns[df.isnull().all()]       # Filtraggio delle sole colonne con tutti valori mancanti

# Eliminazione delle colonne con tutti valori mancanti dal DataFrame
if not colonne_con_tutti_valori_mancanti.empty:
    df = df.drop(columns=colonne_con_tutti_valori_mancanti, axis=1)
    print("\nColonne con tutti valori mancanti eliminate")
else:
    print("\nNessuna colonna con tutti valori mancanti trovata")

colonne_con_tutti_zero = df.columns[(df == 0).all()]                    # Filtraggio delle sole colonne con tutti valori 0

# Visualizzazione dell'elenco delle colonne con tutti valori 0 ed eliminazione delle stesse colonne
if not colonne_con_tutti_zero.empty:
    print("\nElenco delle colonne con tutti valori 0:")
    for colonna in colonne_con_tutti_zero:
        print(f"- {colonna}")
    df = df.drop(columns=colonne_con_tutti_zero, axis=1)
    print("\nColonne con tutti valori 0 eliminate")
else:
    print("\nNessuna colonna con tutti valori 0 trovata")
```

Preprocessing: passaggi

6. Ricerca ed eliminazione delle colonne con tutti valori uguali;

```
colonne_con_valori_uguali = df.columns[df.nunique() == 1]       # Filtraggio delle sole colonne con tutti valori uguali

# Visualizzazione dell'elenco delle colonne con tutti valori uguali ed eliminazione delle stesse colonne
if not colonne_con_valori_uguali.empty:
    print("Elenco e contenuto delle colonne con valori uguali:")
    for colonna in colonne_con_valori_uguali:
        valore_unico = df[colonna].unique()[0]
        print(f"-'{colonna}' --> Contenuto: {valore_unico}")
        df = df.drop(colonna, axis=1)
        print("Colonna '", colonna, "' eliminata", sep="")
else:
    print("Nessuna colonna con tutti valori uguali trovata")
```

Preprocessing: passaggi

7. Codifica della variabile target: parecchi algoritmi di machine learning accettano in input esclusivamente valori numerici. Di conseguenza, occorre codificare il contenuto della variabile y, definita in precedenza, in numeri. Nel progetto in questione, anche una feature appartenente alla variabile x, definita, come y, al passaggio precedente, è stata convertita in numeri;

```
label_encoder = LabelEncoder()                                  # Creazione di un oggetto LabelEncoder
df['Participant'] = label_encoder.fit_transform(df['Participant']) + 1   # Sostituzione dei valori nella colonna target con cifre da 1 a 4
df['Task'] = (df.groupby('Participant').cumcount() % 21) + 1    # Sostituzione dei valori nella colonna dei task con cifre da 1 a 21 in modo ciclico
print(df[['Task', 'Participant']])                              # Visualizzazione delle prime due colonne del dataset
```

Preprocessing: passaggi

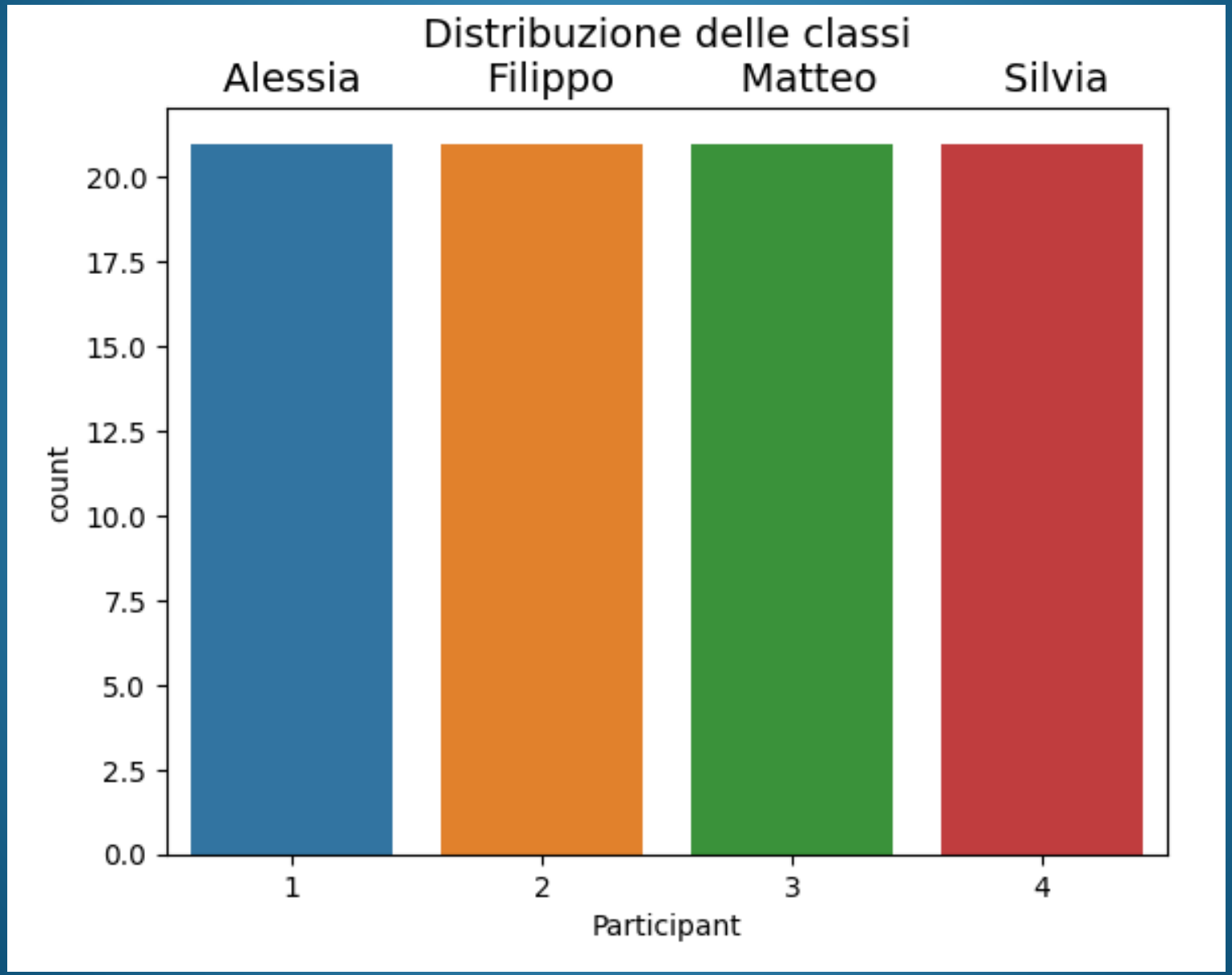
8. Creazione di un array di numpy avendo un DataFrame: le procedure di import di “pandas” creano i DataFrame, ossia strutture dati di tipo tabellare. Quindi, bisogna convertire le colonne del DataFrame in array di numpy, separando quelle con le variabili di input, dette features, da quella con la variabile di output da classificare, chiamata target;

```
X = df.drop('Participant', axis=1)      # Assegnazione a X dell'intero dataset a eccezione della colonna target
y = df['Participant']                   # Assegnazione a y della colonna target del dataset
```

Preprocessing: passaggi

9. Visualizzazione su grafico della distribuzione dei valori di target nelle quattro classi considerate;

```
# Realizzazione di un grafico contenente la distribuzione dei valori della colonna target del dataset nelle quattro classi considerate
sns.countplot(x=y)
plt.title('Distribuzione delle classi\nAlessia          Filippo          Matteo          Silvia', fontsize=14)
```


Distribuzione delle classi
Alessia
Filippo
Matteo
Silvia
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
count
1
2
3
4
Participant

```
# Applicazione del processo di PCA
pcaKM = PCA(n_components=4)
pcaKM.fit(X)
reducedKM_data = pcaKM.transform(X)
```

GRAZIE PER L'ATTENZIONE!

Preprocessing: passaggi

10. Analisi delle componenti principali (PCA): è una tecnica per la semplificazione dei dati che considera la varianza ed effettua una trasformazione lineare delle variabili, che proietta quelle originarie in un nuovo sistema cartesiano in cui la variabile con la maggiore varianza è proiettata sul primo asse, la variabile con la seconda maggiore varianza sul secondo asse e così via.

Implementazione degli algoritmi di boosting

Metodi ensemble

- Nel progetto sono stati implementati algoritmi di classificazione con apprendimento supervisionato;
- I metodi ensemble utilizzano più classificatori, i cui risultati vengono uniti per ottenere delle predizioni migliori;
- I classificatori nei metodi ensemble sono detti “deboli”;
- Uno dei più importanti metodi ensemble è il boosting.

Boosting: caratteristiche

- Il boosting si basa sull'utilizzo di più modelli predittivi. Ad ogni modello viene poi assegnato un peso;
- Il boosting applica un processo iterativo per ridurre gli errori dei modelli precedenti;
- Più modelli vengono generati consecutivamente, dando sempre più peso agli errori effettuati nei modelli precedenti. Ogni modello viene costruito in modo sequenziale in base agli errori del modello precedente.

Boosting: caratteristiche

- Dei dati vengono forniti al primo classificatore, il quale effettua una predizione. Nel classificatore seguente viene dato più peso alle istanze classificate in modo sbagliato, dando enfasi agli errori dei classificatori precedenti;
- Alla base di questo algoritmo sta la realizzazione di un certo numero di classificatori deboli che, imparando dagli errori dei precedenti e in molte iterazioni, si trasformano in un classificatore forte.

Boosting: vantaggi

- Facilità di implementazione, grazie all'ottimizzazione degli iperparametri, per migliorare l'adattamento;
- Riduzione della distorsione, dato che, combinando più classificatori deboli in un metodo sequenziale, si migliorano iterativamente le osservazioni;
- Efficienza computazionale, poiché si selezionano solo le funzioni che aumentano il potere predittivo durante l'addestramento.

Boosting: svantaggi

- Overfitting, perché, quando si verifica, le predizioni non possono essere generalizzate a nuovi set di dati;
- Calcolo intenso, poiché ottenere una buona scalabilità dell'addestramento sequenziale nel boosting è difficile. Inoltre, gli algoritmi di boosting possono essere più lenti da addestrare rispetto agli algoritmi di bagging, per esempio, in quanto un gran numero di parametri può anche influenzare il comportamento del modello.

Boosting: ambiti di utilizzo

- Assistenza sanitaria, per ridurre gli errori nelle predizioni dei dati medici, come i fattori di rischio cardiovascolare e le percentuali di sopravvivenza dei pazienti malati di cancro;
- IT, in particolare per il recupero delle immagini o, nei motori di ricerca, per la classificazione delle pagine web;
- Finanza, per automatizzare attività fondamentali, tra cui il rilevamento delle frodi e l'analisi dei prezzi.

Boosting: migliori algoritmi

- AdaBoost, o boosting adattivo, realizzato da Yoav Freund e Robert Schapire. Esso opera in modo iterativo, identificando i punti di dati mal classificati e regolando i pesi per ridurre l'errore di addestramento. Il modello continua a essere ottimizzato in modo sequenziale fino a quando non produce il predittore più forte.

Boosting: migliori algoritmi

- Gradient boosting, sviluppato da Jerome Friedman sulla base del lavoro svolto da Leo Breiman. Esso aggiunge in sequenza i predittori a un insieme, in modo che ciascuno corregga gli errori del suo predecessore. Tuttavia, invece di modificare i pesi dei punti dati, questo algoritmo viene addestrato sugli errori residui del precedente predittore. Inoltre, combina l'algoritmo di discesa del gradiente e il metodo di boosting.

Boosting: migliori algoritmi

- Light Gradient Boosting Machine, o LGBM, che è un algoritmo che crea alberi leggeri e, invece di espandere l'albero in modo completo durante l'addestramento, lo espande in modo graduale. Questo significa che l'albero ha meno profondità e meno nodi, rendendo l'algoritmo leggero e veloce da addestrare;

Boosting: migliori algoritmi

- CatBoost, che si basa sulla creazione di alberi di decisione e gestisce le variabili trattandole in maniera automatica, ossia senza richiedere una codifica speciale. Questo semplifica di molto il processo di preparazione dei dati. Inoltre, combina le previsioni di tutti gli alberi nel modello per ottenere una previsione finale. Questo processo di combinazione produce una previsione complessiva più accurata.

Grid search sugli algoritmi

Valori testati per “learning_rate”:
0.001, 0.01, 0.1 e 1.0

Valori testati per “n_estimators”:
50, 100, 500 e 1000

AdaBoostClassifier

```
Migliore combinazione di iperparametri per AdaBoostClassifier: {'learning_rate': 0.1, 'n_estimators': 1000}
Migliore precisione in fase di grid search per AdaBoostClassifier: 0.5227272727272727
```

GradientBoostingClassifier

```
Migliore combinazione di iperparametri per GradientBoostingClassifier: {'learning_rate': 0.1, 'n_estimators': 500}
Migliore precisione in fase di grid search per GradientBoostingClassifier: 0.6578282828282828
```

LGBMClassifier

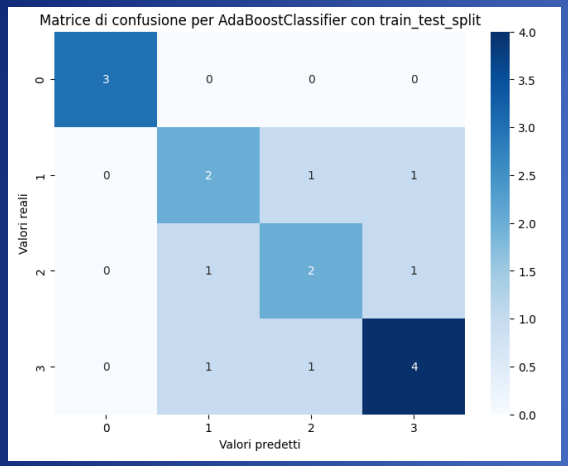
```
Migliore combinazione di iperparametri per LGBMClassifier: {'learning_rate': 0.1, 'n_estimators': 1000, 'verbose': -1}
Migliore accuratezza in fase di grid search per LGBMClassifier: 0.6717171717171717
```

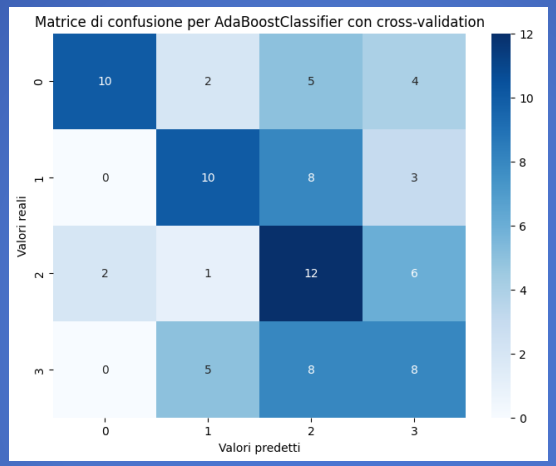
CatBoostClassifier

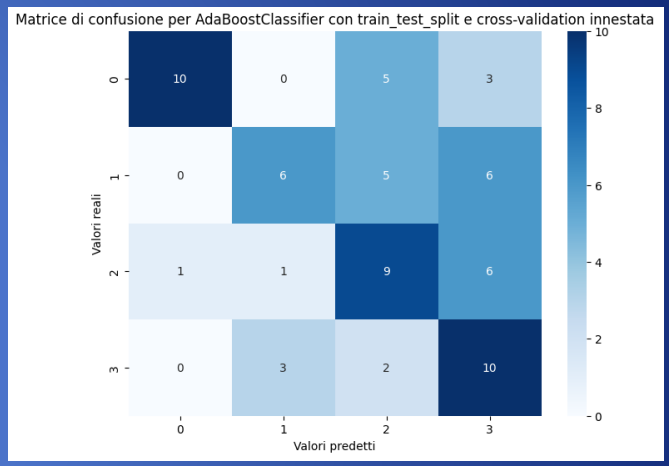
```
Migliore combinazione di iperparametri per CatBoostClassifier: {'learning_rate': 0.1, 'n_estimators': 500, 'verbose': False}
Migliore accuratezza in fase di grid search per CatBoostClassifier: 0.6578282828282828
```

Matrici di confusione

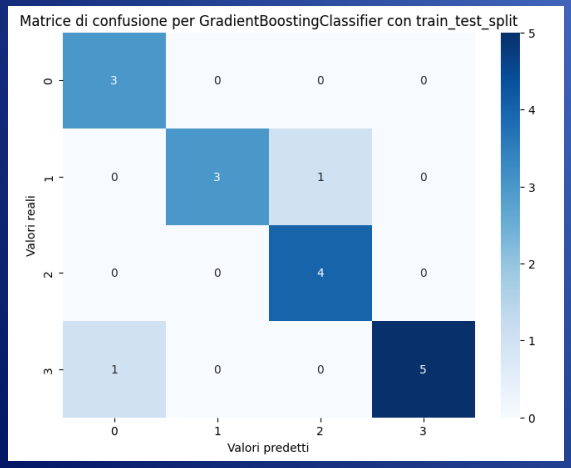
AdaBoostClassifier:

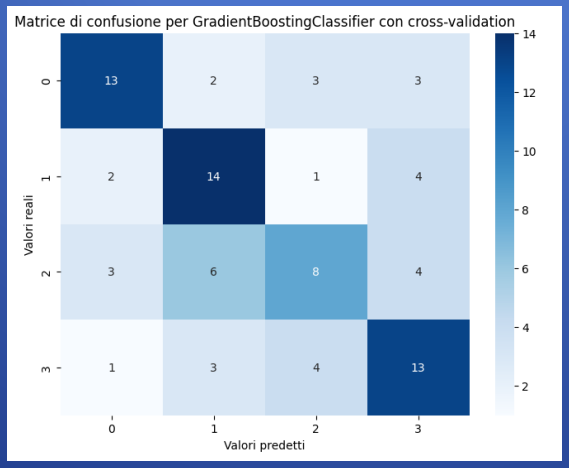


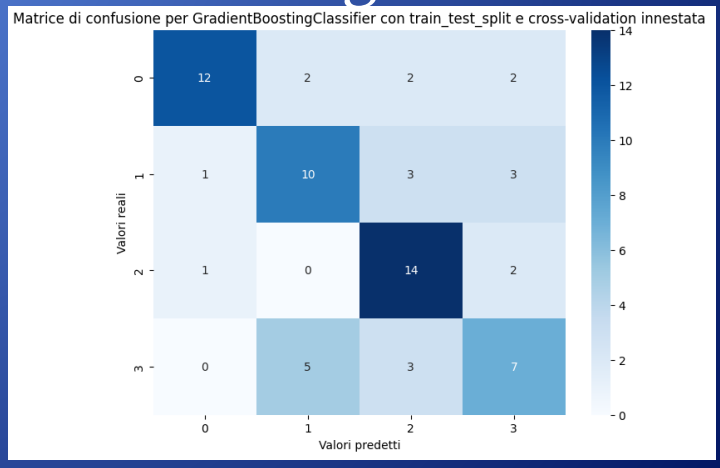




GradientBoostingClassifier:

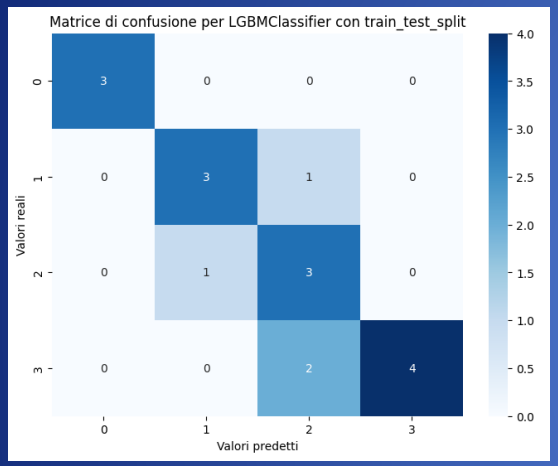


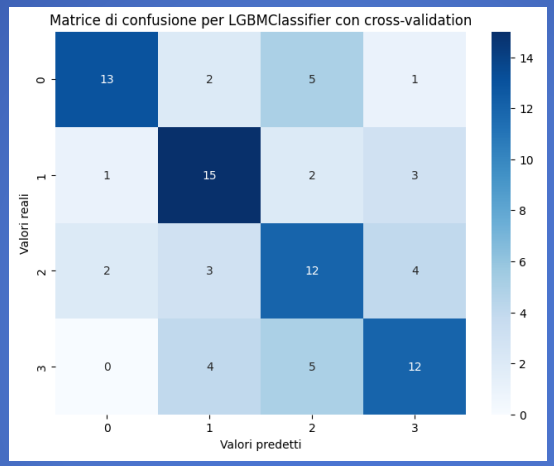


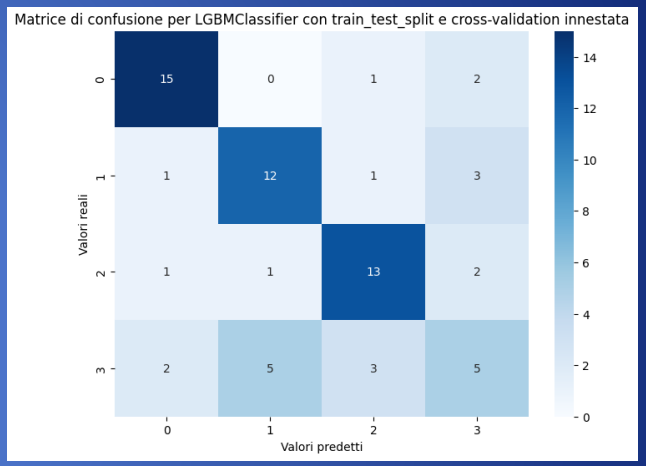


Matrici di confusione

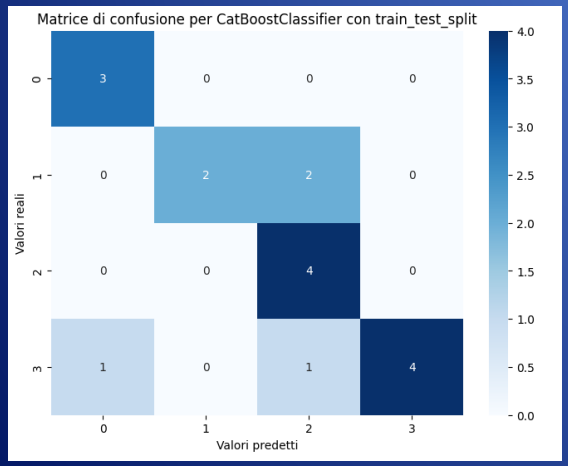
LGBMClassifier:

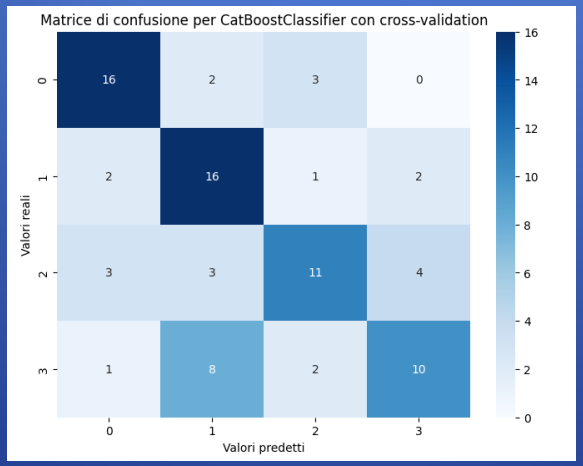


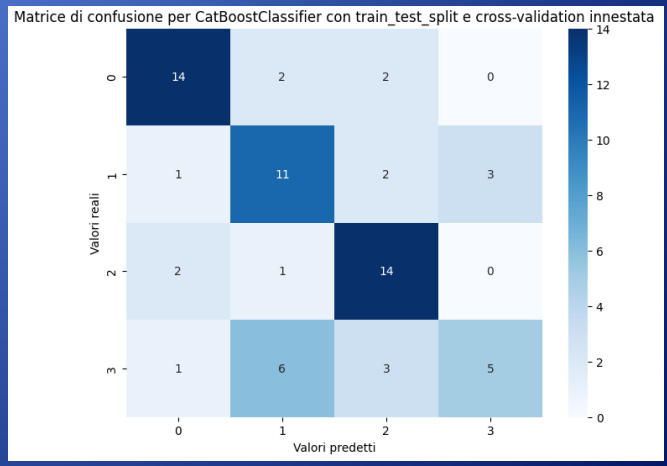




CatBoostClassifier:







Deviazioni standard

AdaBoostClassifier

```
Deviazione standard per AdaBoostClassifier con train_test_split: 1.1439378261076953
Deviazione standard per AdaBoostClassifier con cross-validation: 3.665719574653795
Deviazione standard per AdaBoostClassifier con train_test_split e cross-validation innestata: 3.376735664810025
```

GradientBoostingClassifier

```
Deviazione standard per GradientBoostingClassifier con train_test_split: 1.6381678027601445
Deviazione standard per GradientBoostingClassifier con cross-validation: 4.235268586524354
Deviazione standard per GradientBoostingClassifier con train_test_split e cross-validation innestata: 4.17161164899131
```

LGBMClassifier

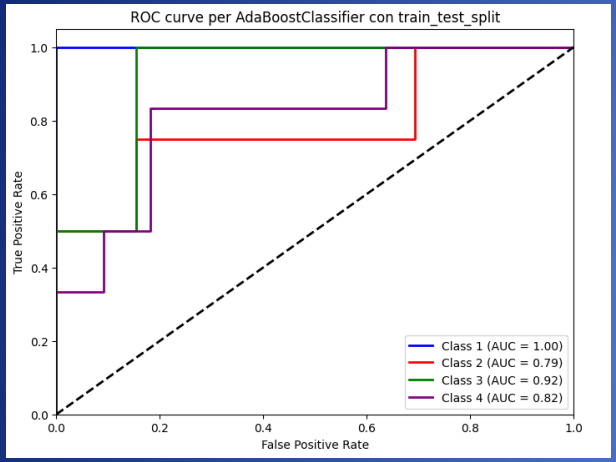
```
Deviazione standard per LGBMClassifier con train_test_split: 1.3905372163304368
Deviazione standard per LGBMClassifier con cross-validation: 4.710360920354193
Deviazione standard per LGBMClassifier con train_test_split e cross-validation innestata: 4.626266718424263
```

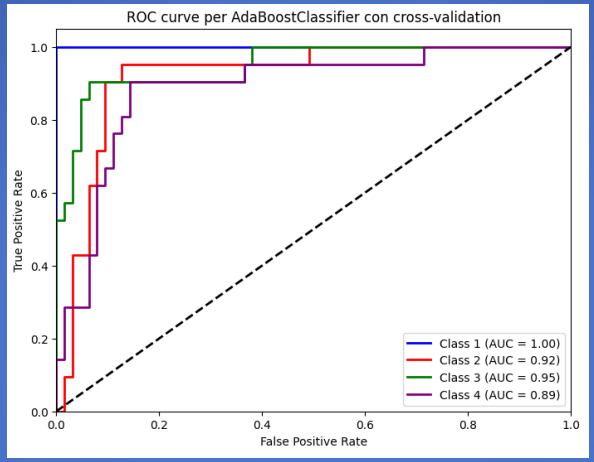
CatBoostClassifier

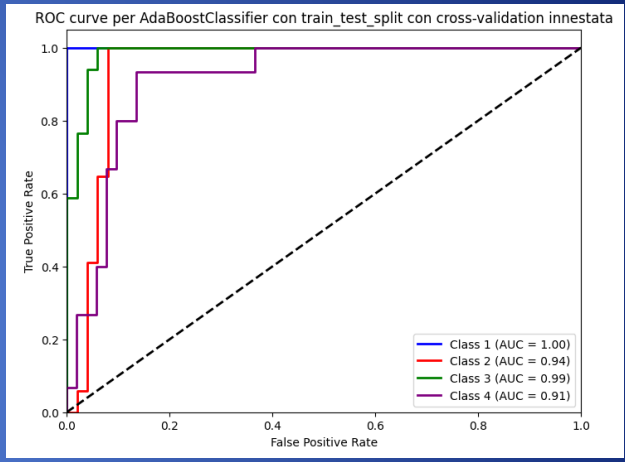
```
Deviazione standard per CatBoostClassifier con train_test_split: 1.4347800354061246
Deviazione standard per CatBoostClassifier con cross-validation: 5.105144464165535
Deviazione standard per CatBoostClassifier con train_test_split e cross-validation innestata: 4.544484981821372
```

ROC curve

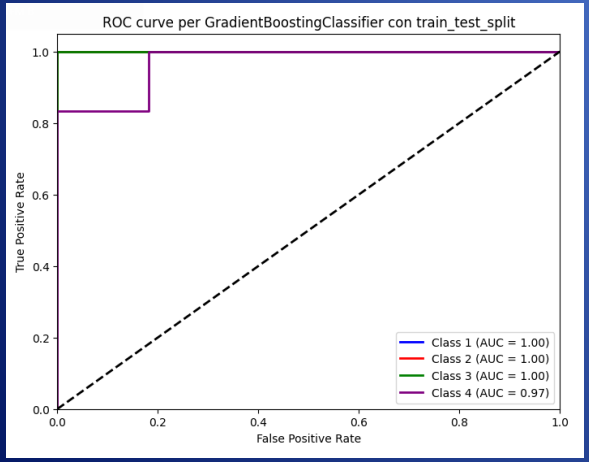
AdaBoostClassifier:

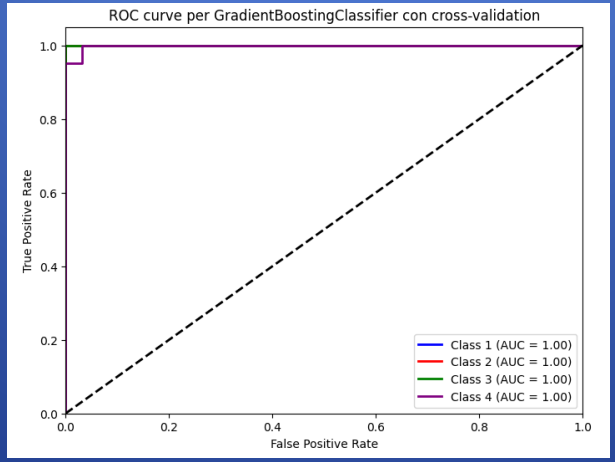


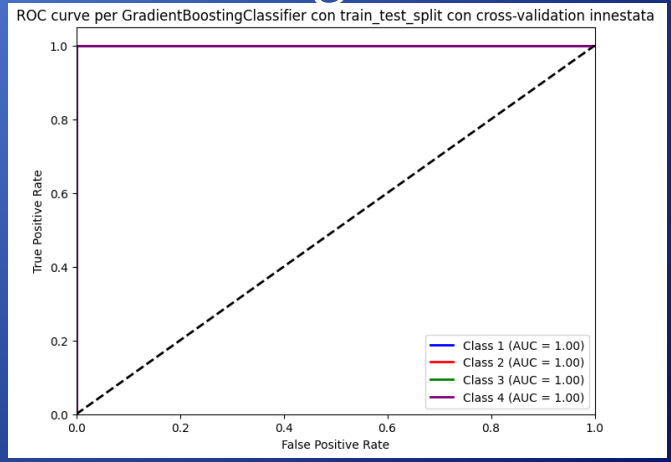




GradientBoostingClassifier:

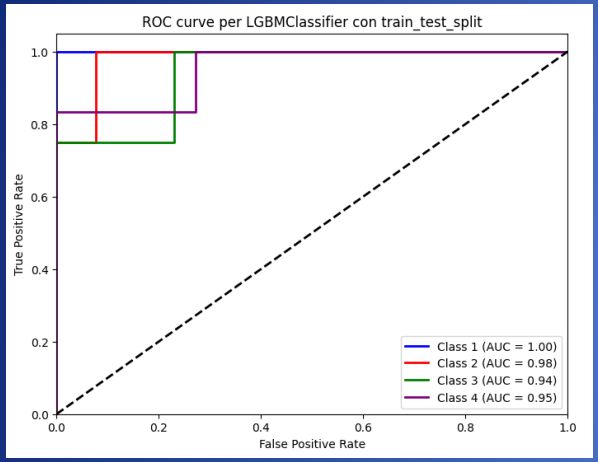


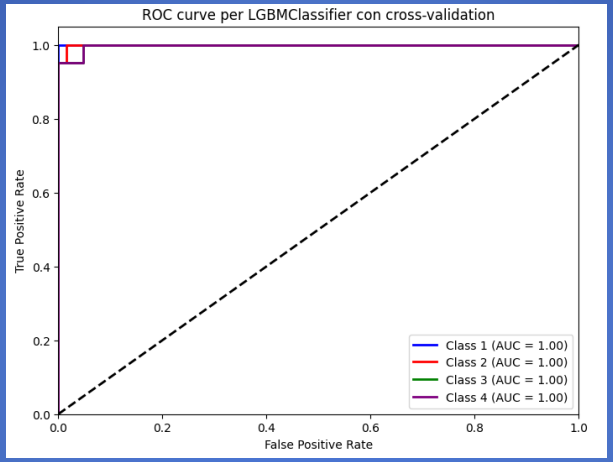


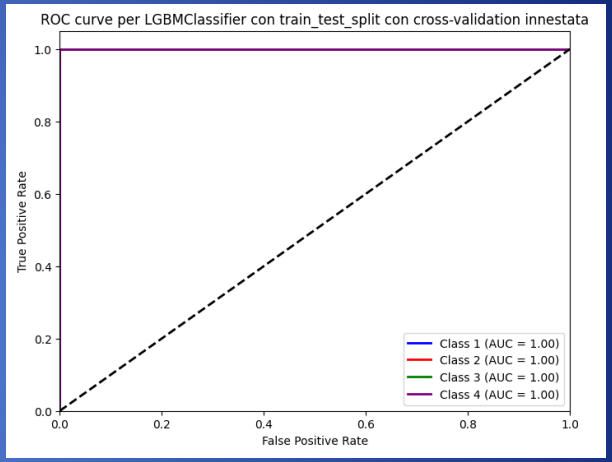


ROC curve

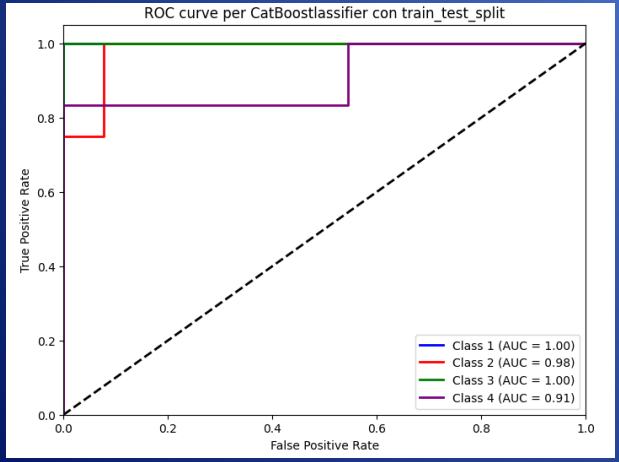
LGBMClassifier:

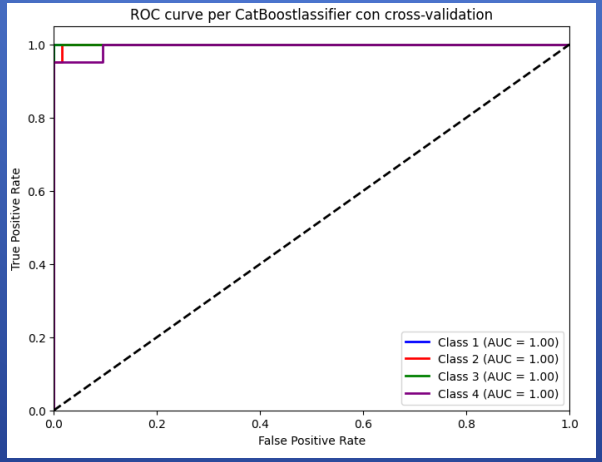


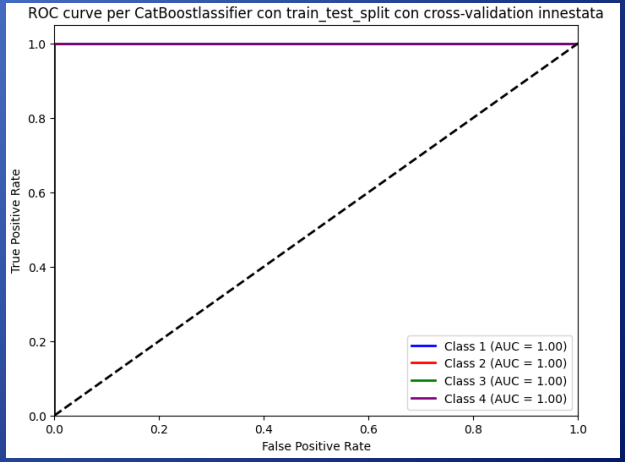




CatBoostClassifier:







Classification report

AdaBoostClassifier:

Classification report per AdaBoostClassifier con train_test_split

	precision	recall	f1-score	support
1	1.00	1.00	1.00	3
2	0.50	0.50	0.50	4
3	0.50	0.50	0.50	4
4	0.67	0.67	0.67	6
accuracy			0.65	17
macro avg	0.67	0.67	0.67	17
weighted avg	0.65	0.65	0.65	17

Classification report per AdaBoostClassifier con cross-validation

	precision	recall	f1-score	support
1	0.83	0.48	0.61	21
2	0.56	0.48	0.51	21
3	0.36	0.57	0.44	21
4	0.38	0.38	0.38	21
accuracy			0.48	84
macro avg	0.53	0.48	0.49	84
weighted avg	0.53	0.48	0.49	84

Classification report per AdaBoostClassifier con train_test_split e cross-validation innestata

	precision	recall	f1-score	support
1	0.91	0.56	0.69	18
2	0.60	0.35	0.44	17
3	0.43	0.53	0.47	17
4	0.40	0.67	0.50	15
accuracy			0.52	67
macro avg	0.58	0.53	0.53	67
weighted avg	0.59	0.52	0.53	67

GradientBoostingClassifier:

Classification report per GradientBoostingClassifier con train_test_split

	precision	recall	f1-score	support
1	0.75	1.00	0.86	3
2	1.00	0.75	0.86	4
3	0.80	1.00	0.89	4
4	1.00	0.83	0.91	6
accuracy			0.88	17
macro avg	0.89	0.90	0.88	17
weighted avg	0.91	0.88	0.88	17

Classification report per GradientBoostingClassifier con cross-validation

	precision	recall	f1-score	support
1	0.68	0.62	0.65	21
2	0.56	0.67	0.61	21
3	0.50	0.38	0.43	21
4	0.54	0.62	0.58	21
accuracy			0.57	84
macro avg	0.57	0.57	0.57	84
weighted avg	0.57	0.57	0.57	84

Classification report per GradientBoostingClassifier con train_test_split e cross-validation innestata

	precision	recall	f1-score	support
1	0.86	0.67	0.75	18
2	0.59	0.59	0.59	17
3	0.64	0.82	0.72	17
4	0.50	0.47	0.48	15
accuracy			0.64	67
macro avg	0.65	0.64	0.63	67
weighted avg	0.65	0.64	0.64	67

Classification report

LGBMClassifier:

Classification report per LGBMClassifier con train_test_split

	precision	recall	f1-score	support
1	1.00	1.00	1.00	3
2	0.75	0.75	0.75	4
3	0.50	0.75	0.60	4
4	1.00	0.67	0.80	6
accuracy			0.76	17
macro avg	0.81	0.79	0.79	17
weighted avg	0.82	0.76	0.78	17

Classification report per LGBMClassifier con cross-validation

	precision	recall	f1-score	support
1	0.81	0.62	0.70	21
2	0.62	0.71	0.67	21
3	0.50	0.57	0.53	21
4	0.60	0.57	0.59	21
accuracy			0.62	84
macro avg	0.63	0.62	0.62	84
weighted avg	0.63	0.62	0.62	84

Classification report per LGBMClassifier con train_test_split e cross-validation innestata

	precision	recall	f1-score	support
1	0.79	0.83	0.81	18
2	0.67	0.71	0.69	17
3	0.72	0.76	0.74	17
4	0.42	0.33	0.37	15
accuracy			0.67	67
macro avg	0.65	0.66	0.65	67
weighted avg	0.66	0.67	0.66	67

CatBoostClassifier:

Classification report per CatBoostClassifier con train_test_split

	precision	recall	f1-score	support
1	0.75	1.00	0.86	3
2	1.00	0.50	0.67	4
3	0.57	1.00	0.73	4
4	1.00	0.67	0.80	6
accuracy			0.76	17
macro avg	0.83	0.79	0.76	17
weighted avg	0.86	0.76	0.76	17

Classification report per CatBoostClassifier con cross-validation

	precision	recall	f1-score	support
1	0.73	0.76	0.74	21
2	0.55	0.76	0.64	21
3	0.65	0.52	0.58	21
4	0.62	0.48	0.54	21
accuracy			0.63	84
macro avg	0.64	0.63	0.63	84
weighted avg	0.64	0.63	0.63	84

Classification report per CatBoostClassifier con train_test_split e cross-validation innestata

	precision	recall	f1-score	support
1	0.78	0.78	0.78	18
2	0.55	0.65	0.59	17
3	0.67	0.82	0.74	17
4	0.62	0.33	0.43	15
accuracy			0.66	67
macro avg	0.65	0.65	0.64	67
weighted avg	0.66	0.66	0.64	67

Conclusioni

GRAZIE PER L'ATTENZIONE!

- In conclusione, è possibile affermare che:
  1. L'algoritmo di boosting con migliore accuratezza è il GradientBoostingClassifier con il train_test_split;
  2. La tecnica di train_test_split permette di ottenere valori di accuratezza migliori della cross-validation e del train_test_split con la cross-validation innestata;
  3. In molti dei casi analizzati si ha un'accuratezza media di almeno il 60-65%.