Daniele Mariani
29723/410

Predizione del rapporto Like/Visualizzazioni dello Youtuber MrBeast tramite Machine Learning

Introduzione

Per la realizzazione di questo progetto ho scelto come Youtuber da analizzare MrBeast (https://www.youtube.com/@MrBeast).
Nome vero Jimmy Donaldson, è un famoso YouTuber noto per video di generosità estrema e progetti di beneficenza. I suoi contenuti spesso includono sfide uniche e gesti filantropici, guadagnando milioni di fan. Il canale YouTube di MrBeast è molto popolare grazie alle sue azioni stravaganti e alla sua generosità. Lo scopo del lavoro è quello di, dopo aver creato autonomamente il Dataset, creare e confrontare vari modelli di classificazione e regressione per predire il rapporto Like/Visualizzazioni tramite tecniche di Machine Learning.

N.B: Saranno allegati in totale 4 file, rispettivamente: Il Dataset contenente le informazioni dei 400 video dello Youtuber MrBeast (MrBeast400.csv); un file .ipynb JupyterNotebook (ProgettoAIML) contenente il codice dell'API Scraping, l'analisi descrittiva del Dataset, la Classificazione a 3 classi e la Regressione; un file contenente il codice per la Classificazione a 10 classi (Classificazione10Classi.ipynb) e infine questo documento in PDF (PaperProgettoAIML_DanieleMariani.pdf)

Organizzazione del lavoro:

Per affrontare il problema in questione, ho deciso di suddividere il lavoro in tre fasi:

1. Creazione del Dataset tramite API Scraping da Youtube, pulizia e creazione nuove colonne;
2. Analisi descrittiva del Dataset;
3. Creazione e confronto dei modelli di classificazione;
4. Creazione e confronto dei modelli di regressione.

1. Creazione del Dataset tramite Google API Scraping

- Per ottenere il Dataset contenente tutte le informazioni necessarie relative ai video del canale Youtube dell'Influencer in questione, ho utilizzato una tecnica di scraping, interfacciandomi tramite le API di Google.

  Ho importato le librerie necessarie: `csv` per salvare i dati in formato CSV e `googleapi.discovery` build per interagire con le API di Google.

Ho definito una funzione `get_youtube_data` che richiede la propria chiave API, l'ID del canale YouTube e un numero massimo di risultati desiderati, nel mio caso 400. La funzione ottiene i dettagli degli ultimi video dal canale specificato e restituisce i dati in formato tabellare.

Per ciascun video ho scaricato ulteriori dettagli come titolo, data di pubblicazione, numero di like, numero di commenti, numero di visualizzazioni e durata del video. Questi dettagli vengono poi memorizzati in una lista di dizionari denominata data.

Ho definito un'altra funzione `save_to_csv` per scrivere i dati raccolti in un file CSV chiamato `"youtube_data.csv"`.
Nella parte principale dello script (`if __name__ == "__main__":`), vengono forniti la chiave API e l'ID del canale di cui ho parlato in precedenza. La funzione `get_youtube_data` viene quindi chiamata con questi parametri, e i dati risultanti vengono salvati in un file CSV.

1.1 Pulizia e creazione nuove colonne

- Ho importato la libreria `pandas` per gestire e manipolare il Dataset. Inoltre, ho incluso le librerie `datetime`, `timezone` e `timedelta` per gestire le informazioni temporali.

  Successivamente dopo aver rinominato il file CSV creato in precedenza, l'ho letto e ho creato un `DataFrame` utilizzando la libreria `pandas`.

- Ho definito una funzione `calculate_video_age` che accetta la data di pubblicazione di un video e calcola l'età in giorni rispetto alla data e ora correnti. Questo è stato ottenuto convertendo la data di pubblicazione in un oggetto `datetime`, impostando il fuso orario su UTC, ottenendo la data e l'ora correnti calcolandone la differenza.

- Successivamente, ho applicato questa funzione alla colonna `'VideoDate'` del DataFrame, creando così una nuova colonna chiamata `'VideoAge(Days)'` che rappresenta l'età in giorni di ciascun video.

- Per quanto riguarda la creazione delle nuove colonne che ci serviranno per raggiungere il nostro scopo, ho aggiunto una nuova colonna chiamata `'Like/ViewsRatio'` al DataFrame, calcolando il rapporto tra il numero di likes e il numero di views per ciascun video. Ho arrotondato il risultato a 3 decimali per renderlo più interpretabile.

- Ho eseguito un'operazione simile alla precedente, creando una colonna denominata `'Comments/ViewsRatio'` che rappresenta il rapporto tra il numero di commenti e il numero di visualizzazioni per ciascun video. In questo caso, ho arrotondato il risultato a 4 decimali.

- Ho convertito la colonna `'Duration(seconds)'` da tipo di dato float a intero per rappresentare la durata dei video in secondi in modo più compatto.

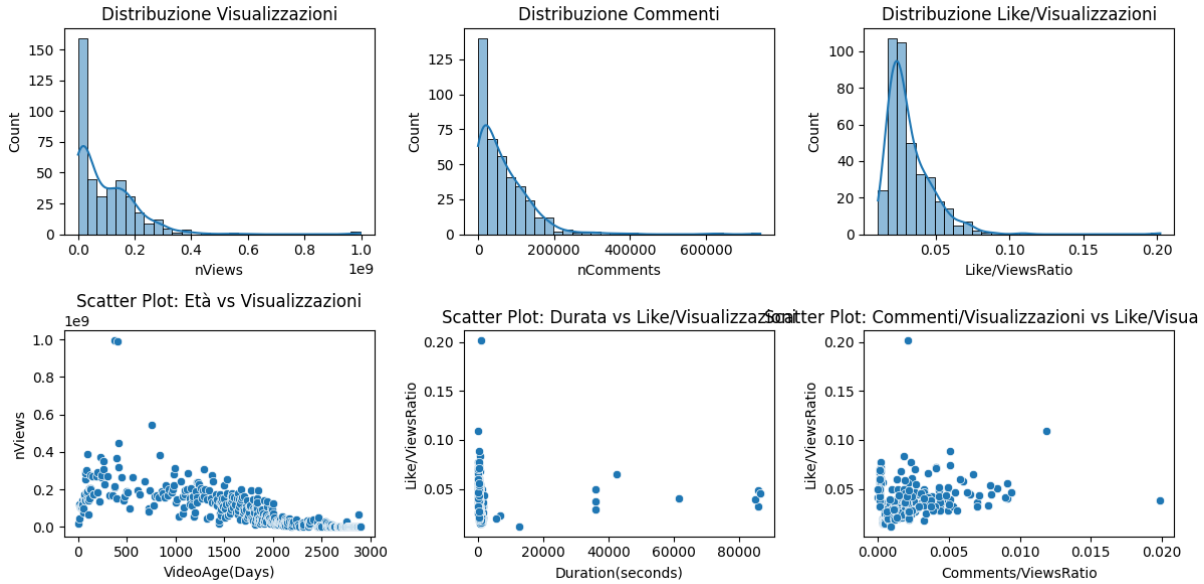
- Ho utilizzato la funzione `print(df)` per visualizzare il DataFrame con le nuove colonne aggiunte.

- Infine, ho salvato il DataFrame aggiornato nel file CSV originale, sovrascrivendo il contenuto precedente.

2. Analisi descrittiva

Prima di procedere nella costruzione dei modelli di classificazione, ho effettuato un'analisi descrittiva per esplorare i dati, tramite grafici e wordcloud, utilizzando librerie quali `seaborn` e `matplotlib` e `Wordcloud.`

Ora descriverò i primi grafici che ho plottato:



Nel primo grafico sulla distribuzione delle visualizzazioni, i valori sulla X sono espressi in miliardi (1e9). Dal grafico si percepisce quindi come la maggior parte dei video si concentri nella fascia da qualche decina di milioni di visualizzazioni a 200 milioni di visualizzazioni. Questo ci fa capire la rilevanza dello Youtuber in questione MrBeast.

Nel secondo grafico vediamo un'andamento simile a quello delle visualizzazioni, ma dei commenti. Infatti la maggior parte dei commenti sotto i video dello Youtuber si aggirano tra le decine di migliaia alle centinaia di migliaia.

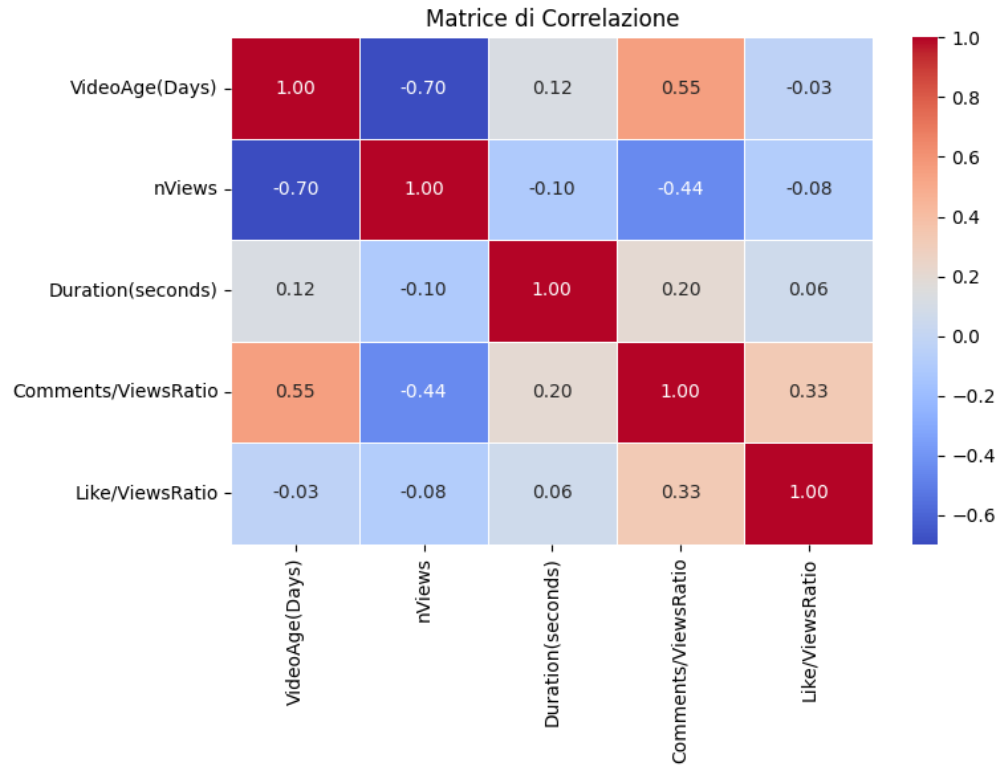
Per quanto riguarda la distribuzione del rapporto Like/Visualizzazioni, variabile di nostro grande interesse dato che sarà quella che andrò a predire nei modelli di Machine Learning, troviamo una distribuzione abbastanza concentrata tra 0.02 e 0.05, che risulta essere, grazie a questa fonte, un buon rapporto Like/Visualizzazioni.

Passiamo ora agli scatter plot. Nel primo ho voluto correlare l'età del video (espressa in giorni) con il numero di visualizzazioni. Si capisce chiaramente come con il passare del tempo il numero di visualizzazioni diminuisca. D'altro canto ci fa capire come MrBeast abbia conquistato una bella fetta di utenti, andando a guadagnare sempre più visualizzazioni nel breve termine.

Nel secondo scatter plot riguardante la durata del video (in secondi) e il rapporto Like/Visualizzazioni, si percepisce come, al contrario di come si possa pensare, meno dura il video più il rapporto sarà alto. Questo fenomeno ha dietro tante motivazioni, quali ad esempio la sempre più scarsa attenzione e concentrazione dei giovani, anche a causa dell'avvento di Social Media strettamente consumistici come TikTok.

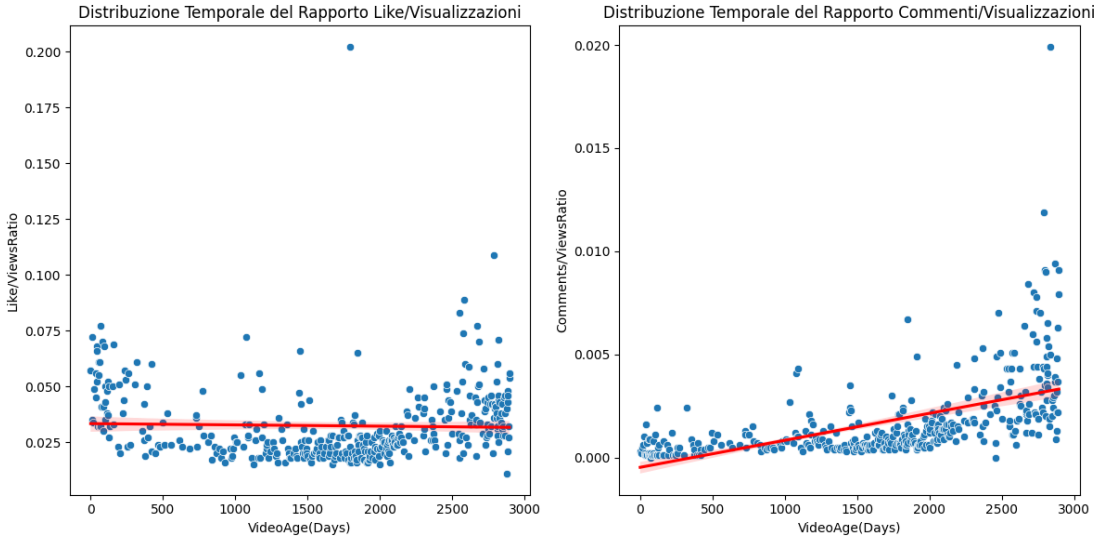
Nell'ultimo scatter plot ho plottato il ratio Like/Visualizzazioni con il ratio Commenti/Visualizzazioni. Come si evince dal grafico c'è una correlazione positiva tra le due variabili; all'aumentare del rapporto Commenti/Visualizzazioni aumenta anche quello dei Like/Visualizzazioni.

Il successivo grafico che ho voluto effettuare è una matrice di correlazione, grazie alla quale si può analizzare la correlazione positiva o negativa tra le variabili:



Dalla matrice si vedono chiaramente le stesse correlazioni che abbiamo visto nei grafici precedenti, come ad esempio la correlazione negativa tra l'età del video con il numero di visualizzazioni. Per il resto delle variabili le correlazioni sono abbastanza moderate, anche se il rapporto Commenti/Visualizzazioni risulta essere leggermente correlato positivamente con l'età del video.

Nel seguente grafico invece ho voluto esaminare la distribuzione temporale dei rapporti Like/Visualizzazioni e Commenti/Visualizzazioni, per analizzare la loro correlazione con il tempo:



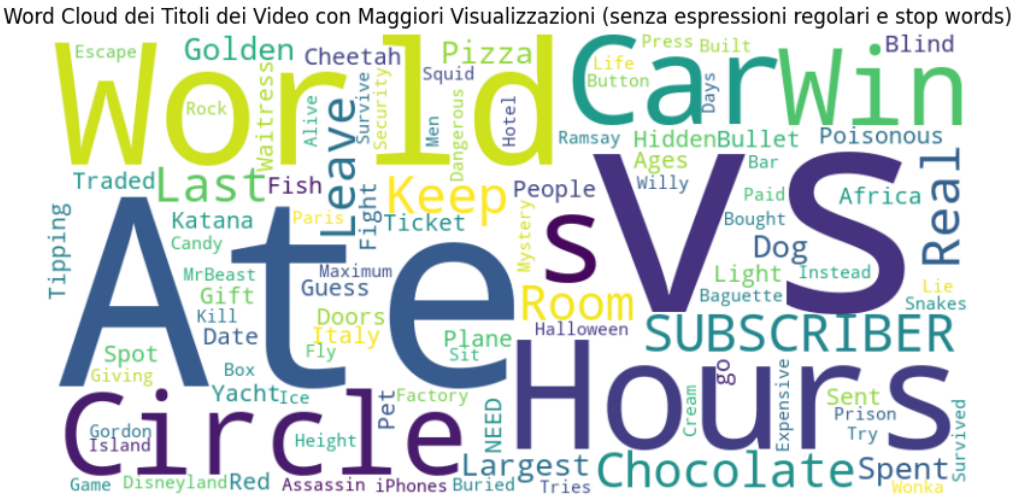
Dai grafici risulta che il rapporto Like/Visualizzazioni non tende nè ad aumentare nè a diminuire con il tempo, al contrario quello dei Commenti/Visualizzazioni tende ad aumentare con il passare del tempo.

Nel successivo grafico, questa volta animato (runnare il codice per visualizzarlo animato) ho plottato uno scatter plot contenente le visualizzazioni e i commenti, dove il colore cambia in base al ratio Like/Visualizzazioni e la grandezza di ogni marker cambia in base alla durata del video. Qui si vede chiaramente il discorso elaborato in precedenza riguardo il rapporto Like/Visualizzazioni e la durata del video. Più il video dura meno questo rapporto sarà alto e viceversa.

Passando ai grafici finali troviamo le cosiddette "Word Cloud", ovvero delle nuvole di parole, che rappresentano, a seconda della grandezza, la loro frequenza e importanza all'interno di tutte le istanze. Quindi più una parola sarà grande più quella parola farà parte di più video e con buone performance ottenute.

Il primo grafico riguardante le Word Cloud, mostra le parole contenute nei titoli vari titoli dei video. Quelle più rilevanti sembrano essere "Win, People, VS, Giving, Gave, Car e Challenge". Dietro queste parole c'è un motivo particolare. Infatti MrBeast è noto per produrre video di sfide e di beneficenza. Infatti alcuni dei suoi famosissimi video sono
▶ $1 vs $100,000,000 Car! ,
▶ I Gave People $1,000,000 But ONLY 1 Minute To Spend It! , o anche
▶ I Gave My 40,000,000th Subscriber 40 Cars .

Il secondo ed ultimo grafico è il seguente:



Questa Word Cloud riguarda sempre i titoli dei video ma questa volta di quelli con maggiori visualizzazioni. Qui troviamo parole come “Ate, Hours, World, Car, Circle”. Anche in questo caso le parole e le motivazioni sono simili a quelle di prima, anche se troviamo l’aggiunta di parole come “Ate e Hours”, Infatti i video rilevanti contenenti queste parole sono I Ate $100,000 Golden Ice Cream , o anche I Spent 50 Hours In Solitary Confinement .

3. Creazione e confronto dei modelli di classificazione

Prima di iniziare, specifico che in questo documento verrà commentato dettagliatamente solo il codice riguardante la classificazione con 3 classi: “Minimo (Classe 0), Medio (Classe 1) e Virale (Classe 2)”. Mentre per quanto riguarda lo script contenente la classificazione con 10 classi è stato commentato brevemente alla fine di questo paragrafo. Il codice lo si può trovare nello script “ClassificazioneCon10Classi.ipynb”.

Classificazione con 3 classi:

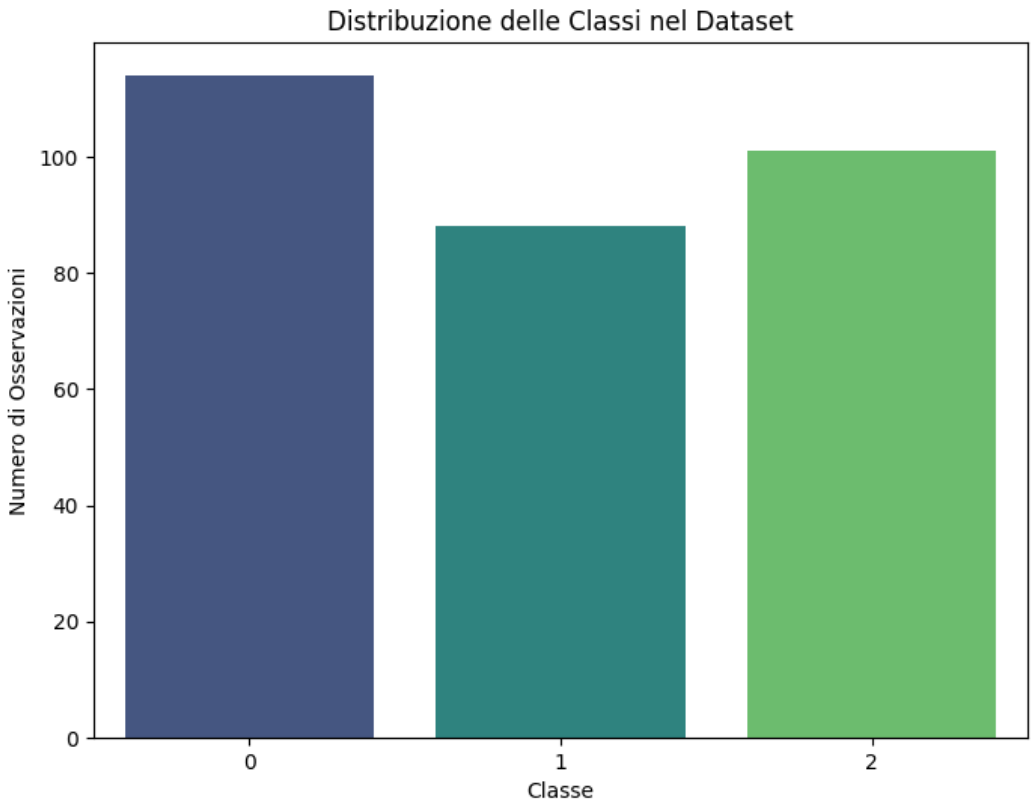
Inizialmente ho importato le librerie necessarie e caricato il Dataset in un Dataframe pandas.

Ho calcolato i percentili (25, 50 e 75) per avere una suddivisione bilanciata delle classi. Ho ottenuto questi risultati:

```
Soglie: [0.022 0.027 0.039], Etichette: ['Minimo', 'Medio', 'Virale']
```

Di conseguenza prima di addestrare i modelli ho definito le soglie e le etichette, per poi utilizzare `LabelEncoder` creando una nuova colonna ‘Class’ con le etichette delle classi che verranno assegnate a ogni istanza. La classe 0 avrà come range da 0 a 0.022 con etichetta “Minimo”, la classe 1 andrà da 0.022 a 0.027 con etichetta “Medio” e infine la classe 2 avrà un range da 0.027 a 0.039 con etichetta “Virale”.

Dopo aver assegnato le features e la variabile target (nel nostro caso ‘Class’), ho plottato la distribuzione in percentuale delle istanze nelle varie classi, questo è il risultato:



Ci troviamo quindi un Dataset abbastanza bilanciato, sicuramente non perfettamente bilanciato.

Successivamente divido il Dataset in Traning e Test set con una percentuale rispettivamente del 75% e del 25%. Subito dopo faccio la standardizzazione delle future utilizzando `RobustScaler,` che rispetto allo `StandardScaler` è appunto più robusto e gestisce meglio gli outliers.

Ho addestrato in totale 5 classificatori: RandomForest, KNN (K-Nearest Neighbors), Naive Bayes XGBoost e SVM (Support Vector Machine). Ho voluto racchiudere le loro prestazioni (Accuracy, Precision, Recall e F-1 Score) in questo unico grafico, in modo poter analizzare singolarmente i classificatori e allo stesso tempo confrontarli. Per quanto riguarda la Cross-Validation ho voluto commentare i risultati separatamente.

Andiamo prima a fare una breve digressione sul funzionamento di questi classificatori e sulle metriche di valutazione:

Digressione Classificatori:

1. **Random Forest:** è un algoritmo di ensemble learning basato su alberi decisionali. Si costruiscono diversi alberi decisionali durante l'addestramento e la previsione viene ottenuta combinando le risposte di ciascun albero;

2. **KNN:** Assegna una classe basandosi sulla maggioranza delle classi dei suoi vicini più prossimi (dove "vicino" è definito dalla distanza, spesso euclidea). Conta le classi dei k vicini più prossimi e assegna la classe più frequente;

3. **Naive Bayes:** Basato sul teorema di Bayes, assume l'indipendenza tra le caratteristiche. Stima la probabilità di appartenenza a ciascuna classe e assegna la classe con la probabilità massima;

4. **XGBoost:** È un algoritmo di boosting che costruisce una serie di alberi decisionali deboli in modo sequenziale. Ogni albero cerca di correggere gli errori del modello precedente. Gli alberi votano per la classe prevista e la somma ponderata delle previsioni degli alberi costituisce la previsione finale;

5. **SVM:** Trova l'iperpiano ottimale che massimizza la separazione tra classi. Trova l'iperpiano che meglio separa le classi e classifica i nuovi punti in base al lato del piano su cui si trovano.

Digressione Metriche:

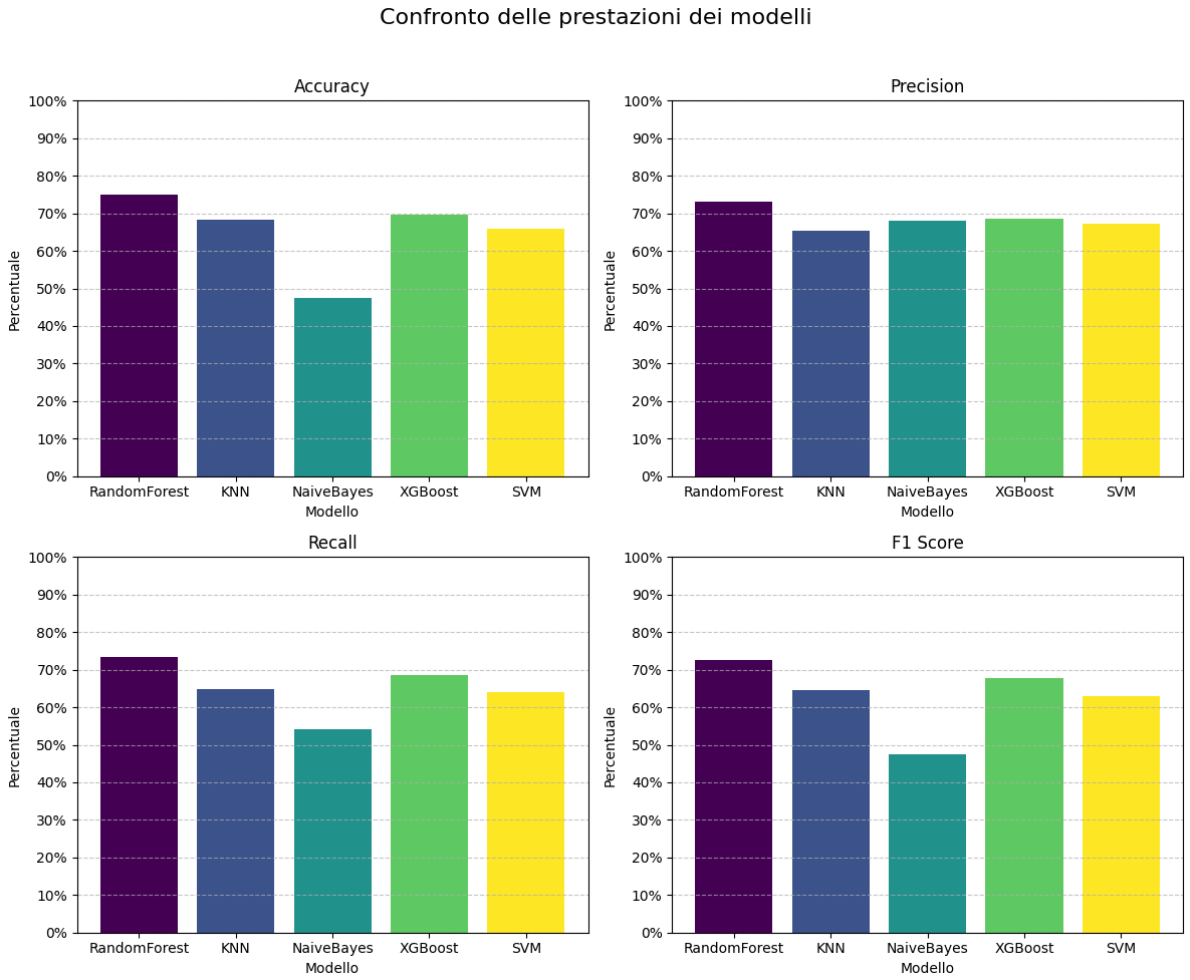
1. **Accuracy:** Misura la percentuale di predizioni corrette rispetto al totale delle predizioni. È una misura generale della correttezza del modello, ma può essere influenzata da sbilanciamenti di classe.
   Calcolo: (TP + TN) / (TP + TN + FP + FN);

2. **Precision:** Misura la percentuale di istanze predette come positive che sono effettivamente positive.
   Indica quanto il modello è preciso quando classifica un'istanza come positiva.
   Calcolo: TP / (TP + FP);

3. **Recall:** Misura la percentuale di istanze positive effettivamente catturate dal modello. Indica quanto il modello è bravo a catturare tutte le istanze positive.
   Calcolo: TP / (TP + FN);

4. **F1 Score:** È la media armonica tra precision e recall, fornendo un bilanciamento tra le due metriche. È utile quando si desidera un equilibrio tra precisione e recall. Calcolo: 2 * (Precision * Recall) / (Precision + Recall)

5. **Cross-Validation:** È una tecnica utilizzata per valutare le prestazioni di un modello suddividendo il dataset in diverse parti, addestrando e testando il modello su diverse combinazioni di queste parti.
   Fornisce una stima più robusta delle prestazioni del modello, riducendo il rischio di overfitting o underfitting.

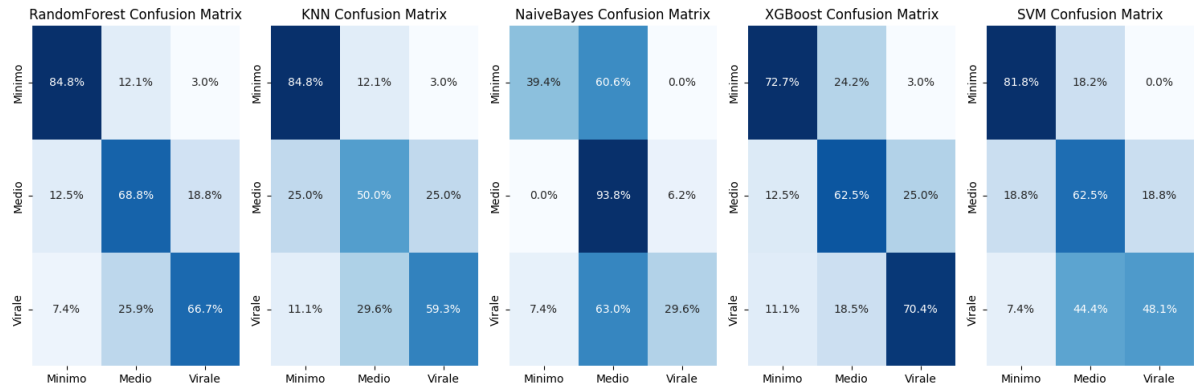


Come si può notare il modello che performa meglio dominando gli altri classificatori è il Random Forest, proprio per il suo approccio tramite Ensemble Learning. Il classificatore peggiore è sicuramente Naive Bayes. Per quanto riguarda le altre metriche risultano tutti molto simili, tuttavia anche XGBoost ha delle buone performance.

Per quanto riguarda la Cross-Validation, anche qui si riconfermano come migliori Random Forest e XGBoost, ma allo stesso tempo SVM risulta avere una mean accuracy dell'88%.

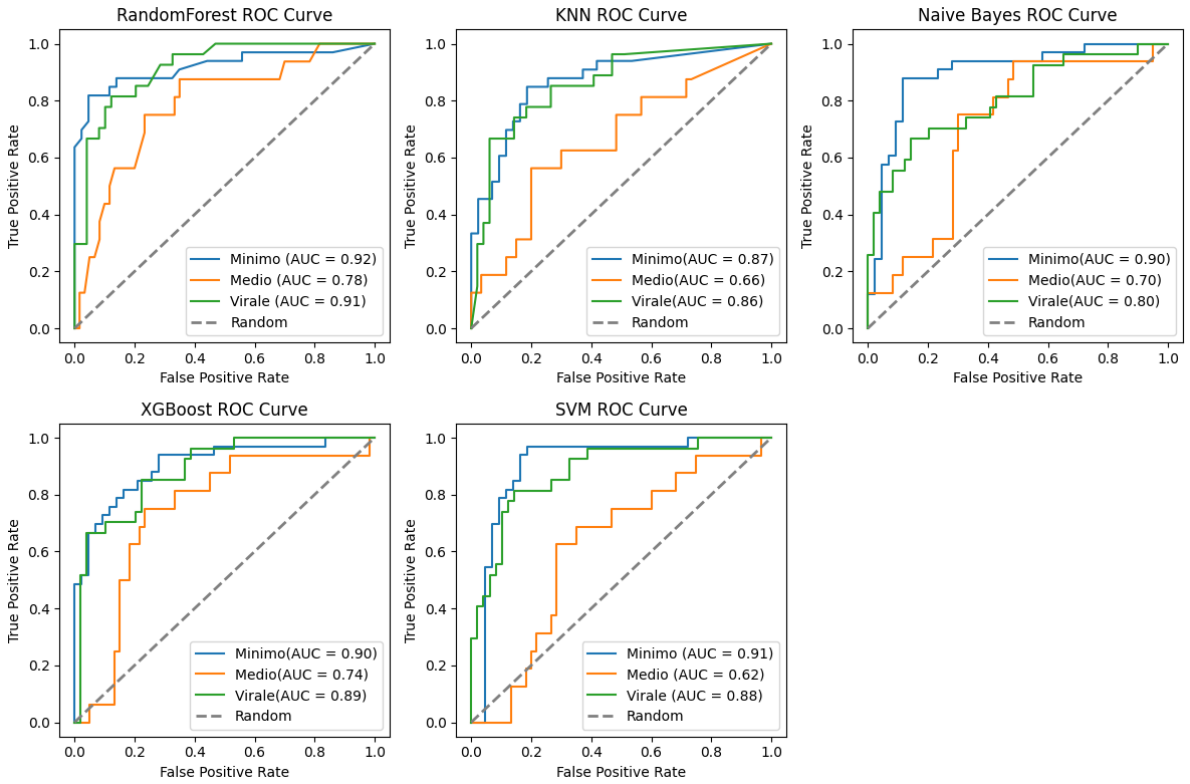
La successiva metrica di valutazione che ho utilizzato è la **matrice di confusione:**
E' una tabella che mostra le prestazioni di un modello di classificazione sui dati di test, confrontando le predizioni del modello con le vere etichette di classe. La diagonale principale (in alto a sinistra e in basso a destra) rappresenta i casi correttamente classificati.
La diagonale opposta rappresenta gli errori di classificazione.

Elementi della Matrice:
True Positive (TP): I casi in cui il modello ha previsto correttamente la classe positiva.
True Negative (TN): I casi in cui il modello ha previsto correttamente la classe negativa.
False Positive (FP): I casi in cui il modello ha previsto erroneamente la classe positiva (falso allarme).
False Negative (FN): I casi in cui il modello ha previsto erroneamente la classe negativa (mancata individuazione).



In generale, i modelli hanno prestazioni abbastanza buone, con accuratezze comprese tra il 65% e il 75%. Tuttavia, è interessante notare che i modelli tendono a sovrastimare la classe virale.

Le successive ed ultime metriche di valutazione dei modelli di classificazione sono le **Curve ROC:**
Una curva ROC è uno strumento grafico utilizzato per valutare le prestazioni di un modello di classificazione variando la soglia di decisione.
Asse delle ascisse (X): Tasso di falsi positivi (FPR), calcolato come FP / (FP + TN).
Asse delle ordinate (Y): Tasso di veri positivi (TPR), sinonimo di recall e calcolato come TP / (TP + FN).
Aree sotto la curva (**AUC**-ROC): Misura l'area sotto la curva ROC e fornisce una valutazione numerica delle prestazioni del modello.



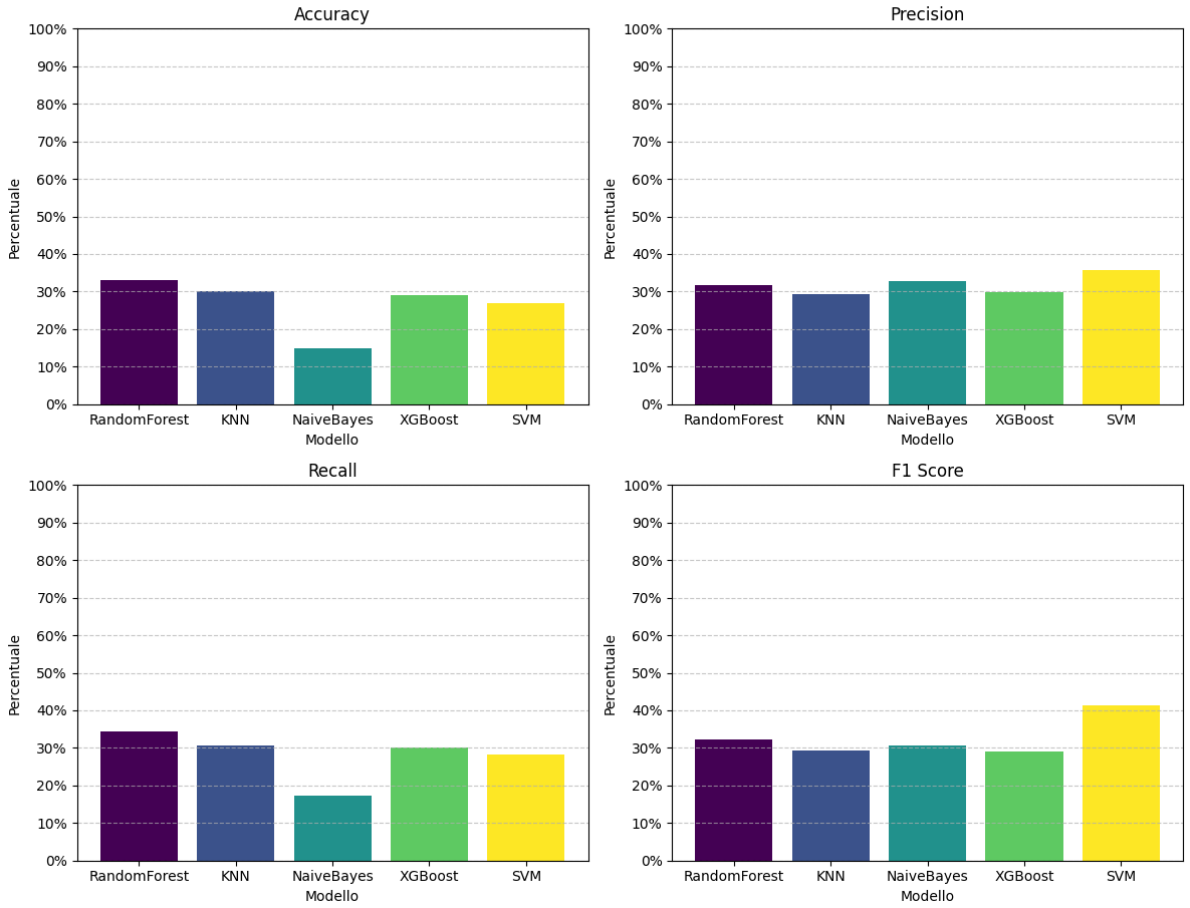
In generale tutti i classificatori hanno delle buone prestazioni, con un' AUC abbastanza alta. Tuttavia alcuni classificatori come Naive Bayes e SVM tendono a fare errori per la classe "Medio". Questo si percepisce dal fatto che la curva vada al di sotto della bisettrice, che sta per una decisione presa "Random".

Conclusione Classificazione:

I risultati indicano che Random Forest e XGBoost dominano gli altri classificatori, mentre Naive Bayes ha prestazioni inferiori. Le matrici di confusione evidenziano una tendenza dei modelli a sovrastimare la classe virale. Infine, le Curve ROC mostrano buone performance complessive, con AUC abbastanza alte anche se alcuni modelli come SVM e Naive Bayes presentano errori nella classe "Medio".
In conclusione, Random Forest e XGBoost spiccano per le prestazioni globali.

Classificazione con 10 classi:

In merito a quest'ultimo, commento brevemente il risultato ottenuto con la classificazione con 10 classi:
In questo caso le classi sono state create tramite il comando di `pandas pd.qcut,` che taglia in quantili. Nel complesso le performance dei modelli sono molto scarse, ecco un istogramma per visualizzare al meglio i risultati delle metriche di valutazione:



4. Creazione e confronto dei modelli di regressione

Per quanto riguarda i modelli di regressione l'approccio è stato molto simile. Ho iniziato importando le librerie necessarie, e dopo aver assegnato le features e la variabile target ho standardizzato tramite il `RobustScaler.` Per la regressione ho deciso di addestrare 6 regressori, che sono i seguenti:

Regressione Lineare: Utilizza una relazione lineare tra le variabili di input e l'output. Cerca di minimizzare la somma dei quadrati delle differenze tra i valori predetti e i valori effettivi. Assume che la relazione tra le variabili sia approssimativamente lineare.

Support Vector Regression (SVR): Estensione del Support Vector Machine (SVM) alla regressione. Cerca di trovare una funzione che si adatti al meglio ai dati mantenendo una "fascia" di errore tollerabile. Utilizza vettori di supporto per definire la "fascia" e minimizzare l'errore.

Random Forest Regression: Basato su un insieme di alberi decisionali.

Ciascun albero contribuisce con una predizione e la media (regressione) delle predizioni degli alberi è considerata come output finale.
Riduce l'overfitting e migliora la stabilità rispetto a un singolo albero.

Lasso Regression: Variante della regressione lineare che aggiunge una penalità basata sulla somma assoluta dei coefficienti. Può essere utilizzato per la selezione automatica delle caratteristiche, portando alcuni coefficienti a zero.
Utile per la regressione e la selezione delle caratteristiche simultaneamente.

KNN (K-Nearest Neighbors) Regression: Utilizza la vicinanza di punti simili per fare predizioni. Calcola la media dei valori target dei k punti più vicini per predire il valore di un nuovo punto. La scelta di k influisce sulla flessibilità del modello.

Albero di Decisione (CART - Classification and Regression Trees): Divide iterativamente i dati in base alle caratteristiche per minimizzare l'errore nella predizione dell'output.
Crea una struttura ad albero dove i nodi rappresentano le decisioni e le foglie contengono le predizioni. Può essere suscettibile all'overfitting, ma possono essere controllati con parametri di potatura.

XGBoost Regressor: Implementa un algoritmo di boosting basato su alberi di decisione. Costruisce una sequenza di alberi di decisione, correggendo gli errori dei modelli precedenti. Utilizza una funzione obiettivo e regolarizzazione per controllare la complessità del modello e prevenire l'overfitting.

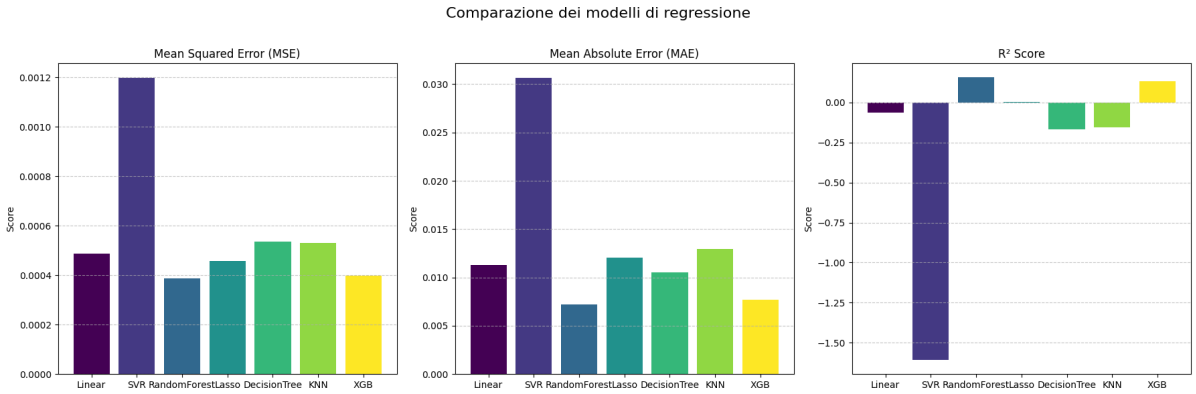
Per quanto riguarda le metriche di valutazione che sono state utilizzate invece sono i seguenti indici:

MSE (Mean Squared Error): MSE è la media dei quadrati degli errori tra i valori predetti dal modello e i valori reali. Maggiore è il MSE, maggiore è la dispersione degli errori. È sensibile agli errori più grandi.

MAE (Mean Absolute Error): MAE è la media degli errori assoluti tra i valori predetti e i valori reali. MAE è meno sensibile agli errori più grandi rispetto al MSE, poiché non tiene conto dei quadrati degli errori.

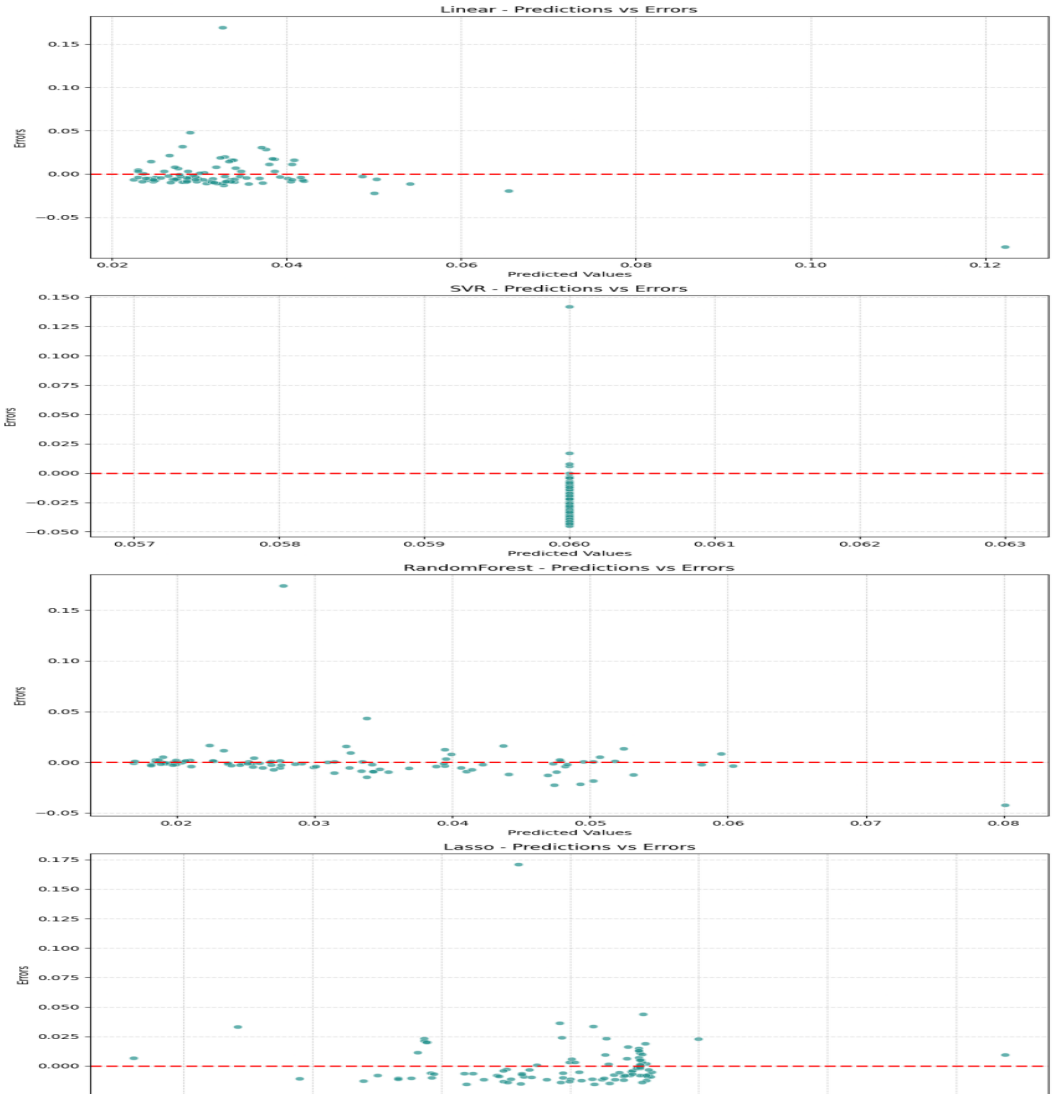
R2 (Coefficiente di determinazione): R2 misura la proporzione di varianza nei dati di risposta che può essere prevista dai predittori del modello. Varia da 0 a 1. Un R2 più vicino a 1 indica che il modello spiega bene la variabilità dei dati, mentre un R2 vicino a 0 indica che il modello non è in grado di spiegare la variabilità.

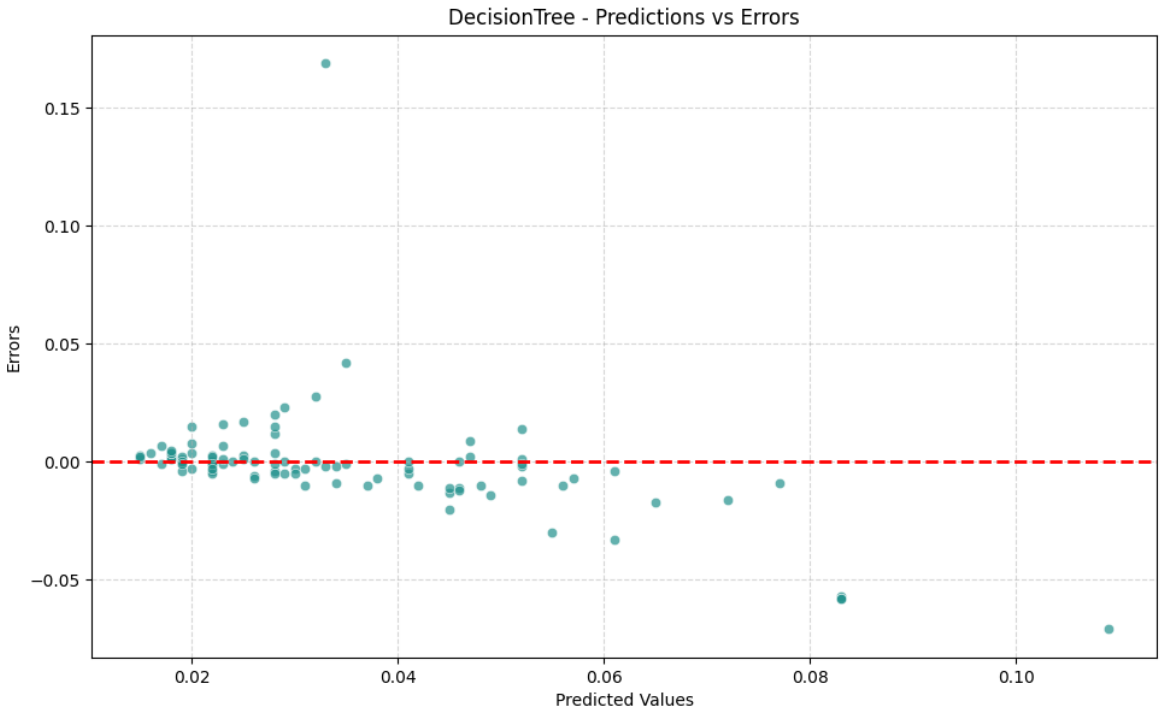
Ora analizziamo nel dettaglio i risultati di questi indici per ogni classificatore tramite questo grafico:

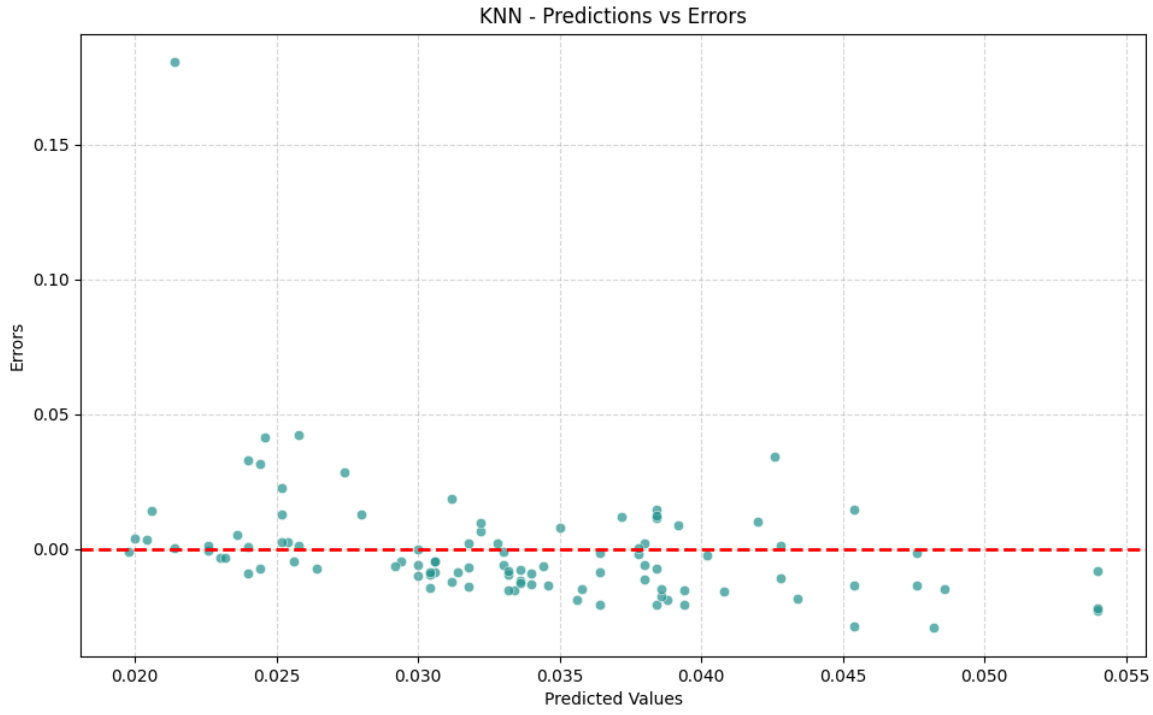


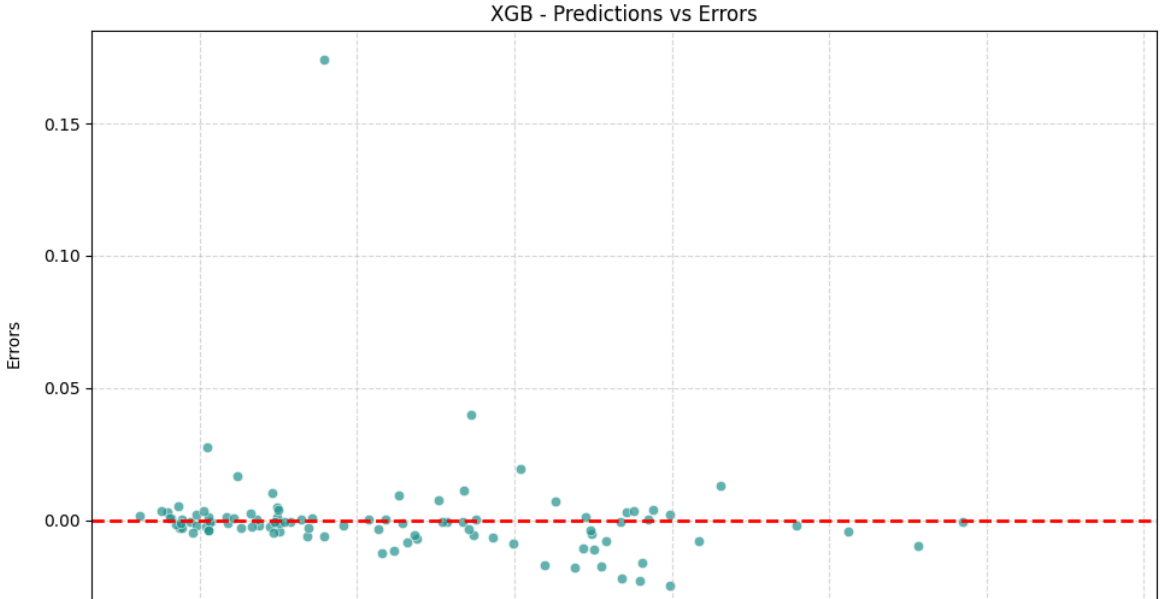
E’ chiaro che il regressore SVR risulta essere quello con maggiore dispersione e con più errori nelle predizioni. Al contrario Random Forest e XGBoost (come nella classificazione) performano meglio, dominando gli altri regressori. Questo si può notare anche dall’indice R2, dove Random Forest e XGBoost sono gli unici ad avere valori positivi.

Successivamente ho voluto plottare uno scatterplot per visualizzare la dispersione degli errori, per ogni regressore.



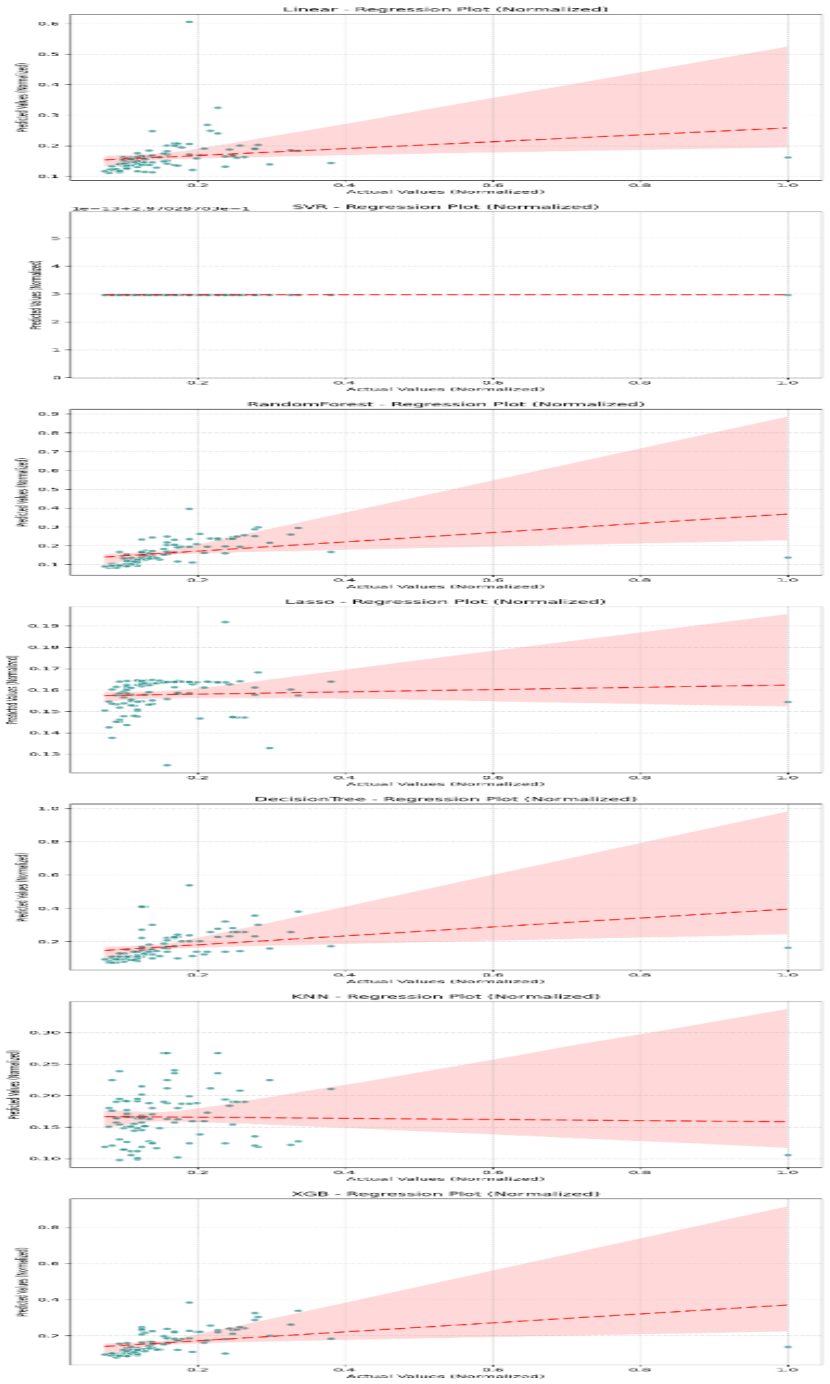

DecisionTree - Predictions vs Errors
Errors
Predicted Values
0.15
0.10
0.05
0.00
−0.05
0.02
0.04
0.06
0.08
0.10


KNN - Predictions vs Errors
Errors
Predicted Values
0.15
0.10
0.05
0.00
0.020
0.025
0.030
0.035
0.040
0.045
0.050
0.055


XGB - Predictions vs Errors
Errors
0.15
0.10
0.05
0.00

Nel complesso tutti i modelli si comportano abbastanza bene, nonostante ciò SVR è il regressore che si comporta peggio. Mentre Regressione lineare, Random Forest e XGBoost hanno una dispersione degli errori bassa.

Terzo ed ultimo grafico. In questo grafico ho voluto visualizzare la dispersione dei valori veri rispetto a quelli previsti:



Anche in questo caso si riconferma ciò che è stato detto prima: SVR è il regressore peggiore, mentre Linear Regression, Random Forest e XGBoost si comportano piuttosto bene, con una

dispersione dei valori previsti rispetto a quelli veri molto più bassa rispetto a quella degli altri regressori.

Conclusione Regressione:

In base ai risultati ottenuti, Random Forest e XGBoost si confermano come i migliori regressori tra quelli testati, evidenziando una maggiore precisione nelle predizioni rispetto agli altri modelli. La Regressione Lineare mostra anche una buona performance, mantenendo una bassa dispersione degli errori.
D'altra parte, il regressore SVR mostra una maggiore variabilità e dispersione degli errori, indicando una minore adattabilità ai dati rispetto agli altri modelli.

Gli scatterplot invece confermano visivamente quanto osservato dalle metriche, sottolineando la coerenza delle prestazioni di ciascun modello.