UNIVERSITÀ
DEGLI STUDI DI BARI
ALDO MORO

Progetto per ingegneria della conoscenza

Componenti del gruppo:

- Ciavarella Andrea, [760411], a.ciavarella16

Link repository github: https://github.com/AndreCiav/icon.git

A.A 2023-2024

INDICE

1. INTRODUZIONE

Il dataset considerato per questo progetto si può trovare al link https://www.kaggle.com/datasets/kunwarakash/chennai-housing-sales-price, contiene dati di vendita di proprietà immobiliari a Chennai, in India, e tramite queste caratteristiche si vuole costruire un modello di apprendimento automatico in grado di prevedere il prezzo di vendita delle case e costruire una base di conoscenza complessiva. Successivamente verranno indicate e descritte tutte le caratteristiche del dataset.

Strumenti utilizzati

Il progetto è stato sviluppato in Python, linguaggio di programmazione che offre una serie di librerie utili e adatte allo studio, manipolazioni, all'analisi e apprendimento di enormi quantità di dati.
L'ambiente di sviluppo usato è Pycharm, Visual Studio Code e Colab. Librerie python utilizzate:

- **Pandas**: usato per organizzare e trasformare i dati in strutture, permettendo di filtrare, pulire e preparare i dati importati dal file CSV per l'analisi.
- **Scikit-learn** (sklearn): usato per l'apprendimento automatico, offre una vasta gamma di algoritmi di apprendimento supervisionato e non supervisionato, oltre a strumenti per la selezione e la valutazione dei modelli.

- **Seaborn**: aiuta a creare visualizzazioni statistiche, come le heatmap e i pairplot, per osservare le relazioni tra le variabili nel dataset.
- **Matplotlib**: Costruisce grafici personalizzati, come le curve di apprendimento o le distribuzioni, per monitorare le performance dei modelli e interpretare i dati.
- **RDFlib**: usato per rappresentare(Resource Description Framework) i dati come grafi RDF.

2. DATASET E PREPROCESSING

Il dataset è formato dai seguenti campi(22 colonne,7109 righe):

- **PRT_ID**: Identificativo unico della proprietà.
- **AREA**: Area o località(paese) in cui si trova la proprietà.
- **INT_SQFT**: Superficie interna in piedi quadrati.
- **DATE_SALE**: Data di vendita della proprietà.
- **DIST_MAINROAD**: Distanza della proprietà dalla strada principale in metri.
- **N_BEDROOM**: Numero di camere da letto.
- **N_BATHROOM**: Numero di bagni.
- **N_ROOM**: Numero totale di stanze.
- **SALE_COND**: Condizione di vendita (ad es. nuovo, rivendita).
- **PARK_FACIL**: Indica se è presente un'area di parcheggio.
- **DATE_BUILD**: Data di costruzione della proprietà.
- **BUILDTYPE**: Tipo di costruzione (ad es. casa indipendente, appartamento).
- **UTILITY_AVAIL**: Servizi disponibili (es. acqua, elettricità).
- **STREET**: Tipo di strada (ad es. strada asfaltata, ghiaia).
- **MZZONE**: Zona in cui si trova la proprietà.
- **QS_ROOMS**: Qualità delle stanze in valori da (1-10).
- **QS_BATHROOM**: Qualità del bagno in valori da (1-10).
- **QS_BEDROOM**: Qualità della camera da letto in valori da (1-10).
- **QS_OVERALL**: Qualità complessiva in valori da (1-10).
- **REG_FEE**: Importo della tassa di registrazione associata alla transazione di vendita della proprietà, può variare in base al valore della proprietà.
- **COMMIS**: Commissione di vendita pagata agli agenti o intermediari.
- **SALES_PRICE**: Prezzo di vendita della proprietà.

<u>Pre-processing del dataset</u>

L'obiettivo del preprocessing del dataset è migliorare la qualità dei dati per renderli adatti all'analisi e alla modellazione. Questo processo include la pulizia dei dati, la trasformazione delle variabili, la gestione degli outlier e la selezione delle caratteristiche più rilevanti.

Alcuni valori all’interno del database si presentano in modo errato, a causa di errori di battitura, quindi vengono effettuate le varie correzioni/rinominazioni.

```
'SALE_COND': {
    # Unifica le variazioni di scrittura per le condizioni di vendita
    'AbNormal': 'Ab Normal',
    'Ab Normal': 'Ab Normal',
    'Partiall': 'Partial',
    'Adj Land': 'AdjLand',
    'PartiaLl': 'Partial'
},
'PARK_FACIL': {
    # Corregge errori di battitura nella disponibilità di parcheggio
    'Noo': 'No'
},
'BUILDTYPE': {
    # Corregge errori di battitura nei tipi di costruzione
    'Comercial': 'Commercial',
    'Other': 'Others'
},
```

I valori corretti vengono salvati in un file *Chennai_Sales_Data_Corrected.csv*

Le colonne DATE_SALE (data di vendita) e DATE_BUILD (data di costruzione) sono convertite in formato datetime, permettendo di estrarre informazioni utili come l'anno e il mese, quindi vengono create due colonne SALE_YEAR e SALE_MONTH.

Successivamente vengono identificate le variabili categoriche, ossia variabili che contengono testo(di solito le analisi vengono effettuate con valori numerici, quindi vanno convertiti), e se utili vengono riadattate al database con varie tecniche(label encoders oppure one-hot encoding), se non utili vengono eliminati o non considerate a causa della poca importanza.
La tecnica utilizzata è label encoders ovvero, viene trasformata una colonna con valori categorici in una colonna numerica. Questo metodo assegna un numero intero univoco a ciascun valore unico nella colonna originale. Per esempio, se la colonna "AREA" contiene valori come "Adyar", "Anna Nagar" e "Velachery", il Label Encoding assegnerà numeri come 1, 2, e 3 rispettivamente a ciascuno di questi valori.

Nel nostro caso le variabili categoriche con i rispettivi valori sono:

- 'PRT_ID': id dell'appartamento quindi , 1,2,3,4,5,6,7 ecc.
- 'AREA= `['Adyar', 'Anna Nagar', 'Chrompet', 'Karapakkam', 'KK Nagar', 'TNagar', 'T Nagar', 'Velachery']`, indicano le città
- 'SALE_COND',=`['AbNormal', 'Family' ,'Partial', 'AdjLand' ,'Normal Sale']`

  Specifica la condizione della vendita, con i seguenti possibili valori:

  - AbNormal `(situazione di vendita non ordinaria)`,
  - Family `(vendita per motivi familiari)`,
  - Partial `(vendita parziale dell'immobile)`,
  - AdjLand `(terreno adiacente incluso nella vendita)`,
  - Normal Sale `(vendita standard)`.

- 'PARK_FACIL'=`['Yes' 'No']`, yes se c'è disponibilità di parcheggio, no altrimento
- 'BUILDTYPE', =`['Commercial', 'Others', 'House']` ,descrive il tipo di edificio in cui si trova l'appartamento

- Commercial `(uso commerciale)`,
- Others `(altre tipologie non specificate)`,
- House `(residenza)`

- 'UTILITY_AVAIL',=`['AllPub','NoSeWa']`, mostra le utilità disponibili nella proprietà:
  - AllPub `(tutti i servizi pubblici disponibili)`,
  - NoSeWa `(mancanza di servizi igienico-sanitari)`.

- 'STREET',=`['Paved' ,'Gravel', 'No Access']` ,indica il tipo di strada d'accesso alla proprietà:
  - Paved `(asfaltata)`,
  - Gravel `(ghiaiosa)`,
  - No Access `(nessun accesso diretto)`.

- 'MZZONE', =`['A', 'RH', 'RL', 'I', 'C' ,'RM']`, indicano il codice della zona

Alcune di queste colonne non sono importanti al fine della predizione quindi vengono eliminate

```
# Elimina le caratteristiche che sono meno utili per la previsione o che possono portare all'overfitti
house_data = house_data.drop(columns=['PRT_ID', 'SALE_COND', 'STREET', 'MZZONE', 'UTILITY_AVAIL'])
```

Le restanti vengono codificate:

	AREA	INT_SQFT	DIST_MAINROAD	N_BEDROOM	N_BATHROOM	N_ROOM	PARK_FACIL	BUILDTYPE
0	4	1004.0	131.0	1.0	1.0	3.0	1	0
1	1	1986.0	26.0	2.0	1.0	5.0	0	0
2	0	909.0	70.0	1.0	1.0	3.0	1	0
3	7	1855.0	14.0	3.0	2.0	5.0	0	2
4	4	1226.0	84.0	1.0	1.0	3.0	1	2
...	...	...	...	...	...	...	...	...
7104	4	598.0	51.0	1.0	1.0	2.0	0	2
7105	7	1897.0	52.0	3.0	2.0	5.0	1	2
7106	7	1614.0	152.0	2.0	1.0	4.0	0	1
7107	4	787.0	40.0	1.0	1.0	2.0	1	0
7108	7	1896.0	156.0	3.0	2.0	5.0	1	2

7109 rows x 19 columns

- AREA: variabile che indica il nome della città, convertita in valori da 1 a 8, in base al numero che sono stati identificati, esempio la città Adyar corrisponde a 1, 'Anna Nagar' corrisponde a 2 e così via.
- PARK_FACIL: variabile che indica la presenza di un parcheggio, codificato in true o false(variabile binaria), perché due valori.
- BUILDTYPE: variabile che indica il tipo di edificio codificato con numeri da 1 a 3

I valori mancanti presenti in alcune colonne vengono sostituiti con la media dei valori, della stessa colonna. Al termine viene salvato il database corretto e preprocessato al fine delle previsioni per i modelli(`/preprocessed_house_data.csv`).

Nel progetto è stato deciso di utilizzare `LabelEncoder` anziché `OneHotEncoder` per semplificare il modello e ridurre il numero di caratteristiche risultanti. `LabelEncoder` assegna un valore numerico intero unico a ciascuna categoria, mantenendo una sola colonna per ogni variabile categoriale. `OneHotEncoder`, invece, crea una colonna separata per ciascuna categoria, il che può risultare in un numero elevato di colonne e aumentare la complessità del modello.

Split dataset

Il dataset viene suddiviso in set di addestramento e di test per valutare l'efficacia del modello. Questa divisione permette di addestrare il modello su una parte dei dati (set di addestramento 80% del dataset) e poi verificarne le prestazioni su dati separati e mai visti prima (set di test 20% del dataset). In questo modo, si può ottenere una stima più accurata delle capacità predittive del modello su dati nuovi e valutare se è in grado di generalizzare bene, evitando problemi come l'overfitting, ovvero quando il modello si adatta troppo ai dati di addestramento e non funziona altrettanto bene su nuovi dati.

In questa fase, viene caricato il dataset pre-elaborato e vengono separate le caratteristiche `(X)` dalla variabile target (`y`),quindi le feature non conterranno la variabile target `'SALES_PRICE'` rappresentante il prezzo di vendita delle case. Il modello, una volta addestrato, restituisce l'importanza di ciascuna caratteristica, evidenziando quali variabili influenzano maggiormente la previsione del prezzo. Solo le caratteristiche con un'importanza sopra una soglia prestabilita(nel nostro caso `threshold =0.01`) vengono quindi selezionate, riducendo le variabili meno rilevanti e ottimizzando le prestazioni del modello.

Importanza delle caratteristiche:

	Feature	Importance
12	REG_FEE	0.770348
0	AREA	0.075496
1	INT_SQFT	0.055797
7	BUILDTYPE	0.035141
3	N_BEDROOM	0.015695
4	N_BATHROOM	0.010283
13	COMMIS	0.009797
2	DIST_MAINROAD	0.003198
11	QS_OVERALL	0.002937
8	QS_ROOMS	0.002755
9	QS_BATHROOM	0.002733
6	PARK_FACIL	0.002572
10	QS_BEDROOM	0.002524
14	SALE_YEAR	0.002384
16	BUILD_YEAR	0.002287
17	BUILD_AGE	0.002267
15	SALE_MONTH	0.002081
5	N_ROOM	0.001705

3. APPRENDIMENTO SUPERVISIONATO

L'apprendimento supervisionato è una tecnica di machine learning dove un modello impara a fare previsioni a partire da un dataset etichettato. In questo contesto, ogni esempio di training è composto da un input (le caratteristiche o feature) e un output noto (il target), che il modello deve imparare a predire. Durante l'addestramento, l'algoritmo analizza queste coppie input-output e cerca di identificare un pattern o una funzione che possa collegarli.

L'obiettivo è addestrare il modello in modo che, una volta addestrato, sia in grado di predire correttamente il target anche per dati mai visti prima.

L'apprendimento supervisionato è utilizzato in problemi di *classificazione*,e di *regressione*, e la scelta va fatta in base al tipo di target a disposizione, ***nel nostro caso il target è continuo.***

Le tipologie di target, in base alla loro natura si affrontano con diverse tecniche di apprendimento:

- **Target Continuo**: un target è continuo quando può assumere un *numero infinito di valori in un intervallo specifico*. Esempi comuni sono valori come il prezzo di una casa o la temperatura, che possono variare in modo fluido e non si limitano a una serie di categorie.

  I problemi con target continuo vengono trattati tramite *modelli di regressione*, dove l'obiettivo è predire un valore numerico preciso.

- **Target Discreto**: un target è discreto quando può assumere un *numero finito di categorie o classi.* Per esempio, nel caso della classificazione di email come "spam" o "non spam", i possibili valori target sono solo due categorie. Altri esempi sono la classificazione del colore di un'auto (ad esempio, rosso, blu, verde) o l'esito di un test medico (positivo o negativo).

  Questo tipo di problema si risolve tramite *modelli di classificazione* che predicono categorie ben definite.

- **Target Strutturato**: un target è strutturato quando il valore di output è rappresentato da una *struttura complessa***,** *come un vettore, una sequenza o un grafo*. Ad esempio, nel caso della traduzione automatica, il target è una frase in un'altra lingua, che è una sequenza di parole e non un singolo valore. Un altro esempio è il riconoscimento di oggetti nelle immagini, dove il modello deve restituire le coordinate dei confini di ciascun oggetto (tipicamente un problema di output strutturato).

  I problemi con target strutturato si risolvono tramite tecniche speciali, come le *reti neurali ricorrenti (RNN)* per le sequenze o le *reti neurali convoluzionali (CNN)* per immagini, oppure con modelli come *Conditional Random Fields (CRF)* e *Hidden Markov Models (HMM)*.

Le fasi vengono suddivise in:
- ➔ scelta degli iper-paramenti
- ➔ fase di addestramento
- ➔ fase di test
- ➔ valutazione delle prestazioni

<u>Scelta dei modelli</u>
L'obiettivo quindi è predire il prezzo della casa basandosi sulle caratteristiche influenti e come l'area, l'anno di costruzione,numero di stanze ecc. I modelli scelti sono DecisionTree, RandomForest, Lasso, Ridge e GradientBoosting, ciascuno adatto alla regressione.

- Il **Decision Tree Regressor** costruisce un albero basato su condizioni di soglia applicate ai valori delle feature. Partendo dalla radice, ogni nodo rappresenta una condizione sui dati (condizioni if), quindi può essere true o false, e la risposta determina il percorso da seguire

nell'albero. Ogni decisione suddivide il dataset in gruppi sempre più piccoli e omogenei rispetto alla variabile target, fino ad arrivare alle foglie, che contengono le previsioni finali(la risposta). In questo modo, un decision tree modella il processo decisionale attraverso una sequenza di scelte, che formano un cammino dalle radici alle foglie per ciascuna richiesta data di input.

È preferibile un albero decisionale più piccolo, cioè con meno nodi o meno profondo, che sia comunque coerente con il training set. Dato che esplorare tutto lo spazio possibile degli alberi di decisione è impraticabile, si utilizzano tecniche greedy per costruire l'albero minimizzando una determinata loss

- Il **Random Forest Regressor,** utilizza molti decision tree addestrati su campioni casuali del dataset, migliorando la robustezza del modello. Questa tecnica, chiamata bagging, introduce diversità nei dati di addestramento. A ogni nodo, il modello sceglie casualmente un sottoinsieme di feature, riducendo la correlazione tra gli alberi e migliorando la generalizzazione sui dati di test. Le previsioni finali del modello sono ottenute facendo la media delle previsioni di tutti gli alberi.

- Il **Gradient Boosting Regressor** costruisce una sequenza di modelli semplici, come decision tree poco profondi. Ogni modello successivo viene addestrato per correggere gli errori del precedente. Si calcolano gli errori e un nuovo modello viene addestrato per predire questi residui. La previsione complessiva viene aggiornata sommando le previsioni di tutti i modelli. Questo processo iterativo continua fino a ridurre al minimo l'errore complessivo della previsione. Così facendo, il Gradient Boosting migliora progressivamente la precisione del modello.

- La **Lasso Regression** (Least Absolute Shrinkage and Selection Operator) è una forma di regressione lineare che include una penalizzazione L1 sui coefficienti del modello, la quale tende a ridurre a zero i coefficienti meno rilevanti. Questa penalizzazione L1 ha un effetto di sparse selection (selezione sparsa), che elimina in modo efficace le feature che non contribuiscono significativamente alla previsione. Grazie a questa caratteristica, Lasso è particolarmente utile quando si lavora con dataset che contengono molte feature irrilevanti o ridondanti, rendendo il modello più interpretabile e meno soggetto a sovradattamento. Il modello finale di Lasso risulta quindi più semplice, con solo le feature essenziali, ma preserva una buona capacità predittiva.

- La **Ridge Regression** è un’altra variante della regressione lineare che applica una penalizzazione L2 ai coefficienti del modello, riducendone i valori estremi senza però eliminarli. Questa penalizzazione L2 è utile per evitare che i coefficienti diventino troppo grandi, riducendo la varianza del modello e migliorando la stabilità in presenza di feature correlate. La Ridge Regression è efficace in situazioni in cui tutte le feature hanno un ruolo, anche minimo, nella previsione, poiché tende a preservare tutte le variabili riducendo i pesi dei coefficienti. Questa stabilità rende Ridge adatta per dataset in cui le feature sono strettamente correlate, impedendo al modello di adattarsi eccessivamente a una singola feature e bilanciando l'importanza delle variabili.

Scelta degli iper-parametri

L'importanza e l'efficacia delle prestazioni dei modelli variano in base ai parametri impostati per i modelli. Per condurre questo task viene utilizzata la tecnica GridSearch, inclusa nella libreria Scikit-learn, che si occupa di trovare la combinazione ottimale di iperparametri per un modello di machine learning. Funziona esplorando sistematicamente un insieme predefinito di iperparametri, addestrando e valutando il modello per ogni combinazione possibile.

Nel nostro codice si può vedere, per ottimizzare le prestazioni di ogni modello, il codice esegue una Grid Search con Cross-Validation a 5 fold, esplorando varie combinazioni di iperparametri come la profondità dell'albero, il numero di estimatori e il tasso di apprendimento, in modo da evitare la dipendenza dai dati di training e garantire predizioni affidabili.
La metrica utilizzata per i modelli è *neg_mean_squared_error*

```
print(f"Addestramento {model_name}...")
start_train_time = time.time()
grid_search = GridSearchCV(model, param_grids[model_name], cv=5, scoring='neg_mean_squared_error')
grid_search.fit(X_train, y_train)

print(f"Migliori iperparametri per {model_name}: {grid_search.best_params_}")
```

grid_search.fit(X_train, y_train) addestra il modello sui dati di addestramento, esplorando tutte le combinazioni degli iperparametri specificati in *param_grids[model_name]*

Iperparametri utilizzati per ogni modello:

Decision tree regression:

```
'DecisionTree': {
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10]
},
```

- **max_depth**: Limita la profondità dell'albero, evitando che cresca eccessivamente e si adatti troppo ai dati di training, riducendo così l'overfitting.
- **min_samples_split**: Controlla la suddivisione dei nodi, con valori più alti che impediscono la creazione di nodi molto specifici e quindi riducono il rischio di overfitting.

Randomforest regression

```
param_grids = {
    'RandomForest': {
        'n_estimators': [100, 200],
        'max_depth': [None, 10, 20],
        'min_samples_split': [2, 5, 10]
    },
```

- **n_estimators**: Questo parametro indica il numero di decision tree nel Random Forest. Più alberi in generale portano a una maggiore stabilità del modello, ma aumentano anche il tempo di addestramento.
- **max_depth**: Limita la profondità massima degli alberi, il che riduce la complessità del modello e può aiutare a prevenire l'overfitting.
- **min_samples_split**: Specifica il numero minimo di campioni richiesti per suddividere un nodo. Valori più alti riducono la complessità dell'albero, dato che impongono più campioni per ogni decisione e quindi aumentano la generalizzazione.

GradientBoosting Regressor

```
'GradientBoosting': {
    'n_estimators': [100, 200],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 5, 10]
}
```

- **n_estimators**: È il numero totale di alberi deboli (o stadi) costruiti nella sequenza. Un numero maggiore consente una rappresentazione più dettagliata degli errori, ma richiede un bilanciamento con `learning_rate` per evitare l'overfitting.
- **learning_rate**: Determina quanto ciascun albero contribuisce al modello finale. Un valore più basso rende il modello più stabile ma richiede più alberi per raggiungere una buona accuratezza.
- **max_depth**: Limita la profondità di ciascun albero all'interno del boosting, con alberi più bassi che riducono il rischio di sovradattamento, in particolare nei modelli di boosting, che tendono a essere sensibili all'overfitting quando gli alberi sono troppo complessi.

Lasso

```
'Lasso': {
    'alpha': [0.01, 0.1, 1, 10]
```

- **alpha**: È il parametro di regolarizzazione L1, che controlla l'entità della penalizzazione applicata ai coefficienti. Valori più alti per `alpha` aumentano l'intensità della regolarizzazione, eliminando coefficienti irrilevanti e producendo un modello più semplice, ma al costo di una possibile perdita di precisione.

Ridge

```
'Ridge': {
    'alpha': [0.01, 0.1, 1, 10]
},
```

- **alpha**: Anche qui, `alpha` è il parametro di regolarizzazione (in questo caso L2), che riduce i coefficienti e stabilizza il modello, limitando la varianza. Come in Lasso, valori più alti

aumentano la regolarizzazione e riducono l'importanza delle feature meno rilevanti senza eliminarle.

Valutazione dei modelli

```
Addestramento RandomForest...
Migliori iperparametri per RandomForest: {'max_depth': None,
'min_samples_split': 5, 'n_estimators': 200}
RandomForest MSE: 561440183312.3574
RandomForest MAE: 596192.1844419925
RandomForest R²: 0.9572639906290961

Addestramento Lasso...
Migliori iperparametri per Lasso: {'alpha': 10}
Lasso MSE: 1901049798438.0662
Lasso MAE: 1103898.1108480382
Lasso R²: 0.8552948570205839

Addestramento Ridge…
Migliori iperparametri per Ridge: {'alpha': 1}
Ridge MSE: 1901082187476.7986
Ridge MAE: 1103906.0907978127
Ridge R²: 0.8552923916141097

Addestramento DecisionTree...
Migliori iperparametri per DecisionTree: {'max_depth': 10,
'min_samples_split': 10}
DecisionTree MSE: 847455966686.2823
DecisionTree MAE: 707614.7326468945
DecisionTree R²: 0.9354928855999177

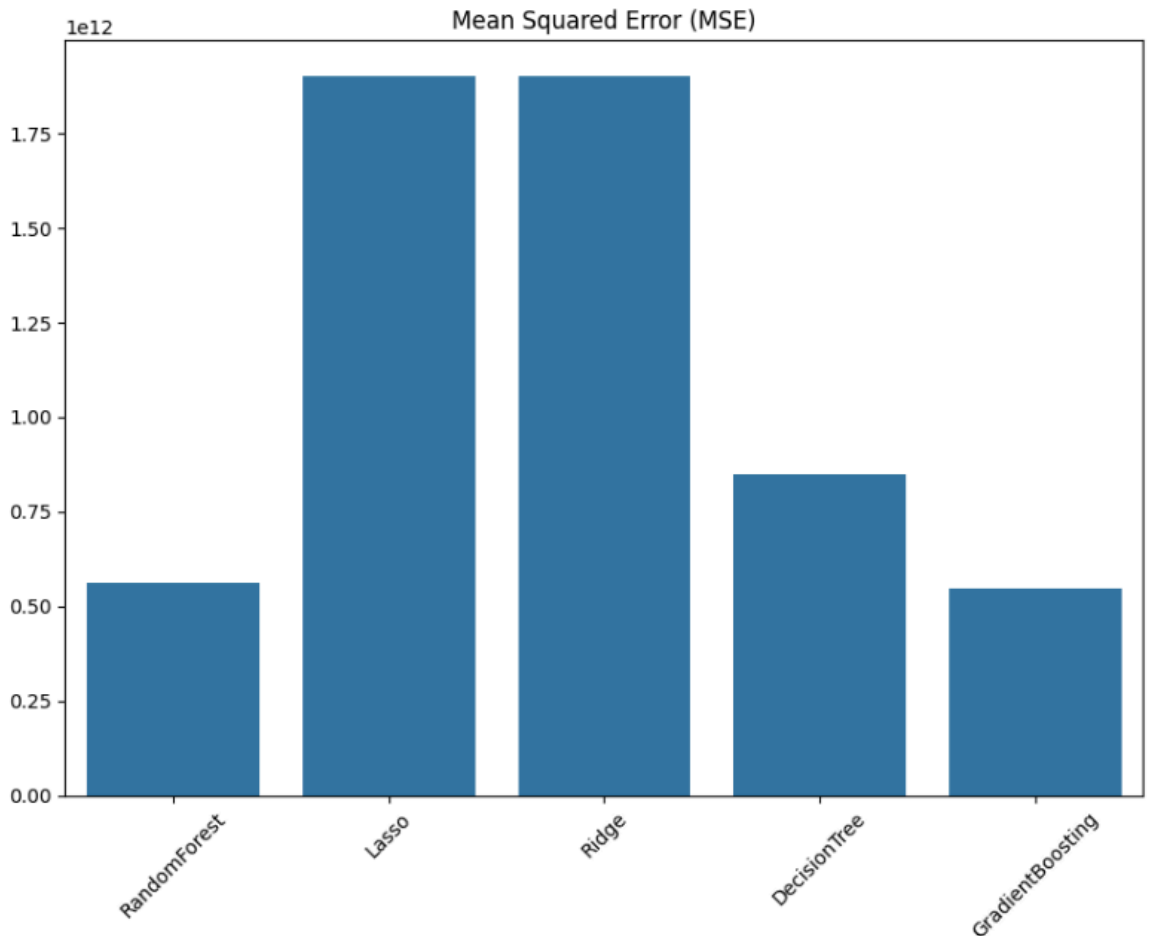

Addestramento GradientBoosting...
Migliori iperparametri per GradientBoosting: {'learning_rate': 0.1,
'max_depth': 5, 'n_estimators': 200}
GradientBoosting MSE: 545096985830.48944
GradientBoosting MAE: 579090.6317725645
GradientBoosting R²: 0.9585080110278765
```

Per valutare i modelli ho usato le metriche,a Mean Squared Error (MSE), Mean Absolute Error (MAE) e al punteggio R^2, che misurano rispettivamente l'accuratezza delle previsioni, la precisione complessiva e la bontà dell'adattamento. Dopo l'addestramento, il modello con il minor MSE viene selezionato come migliore e salvato per l'uso futuro.

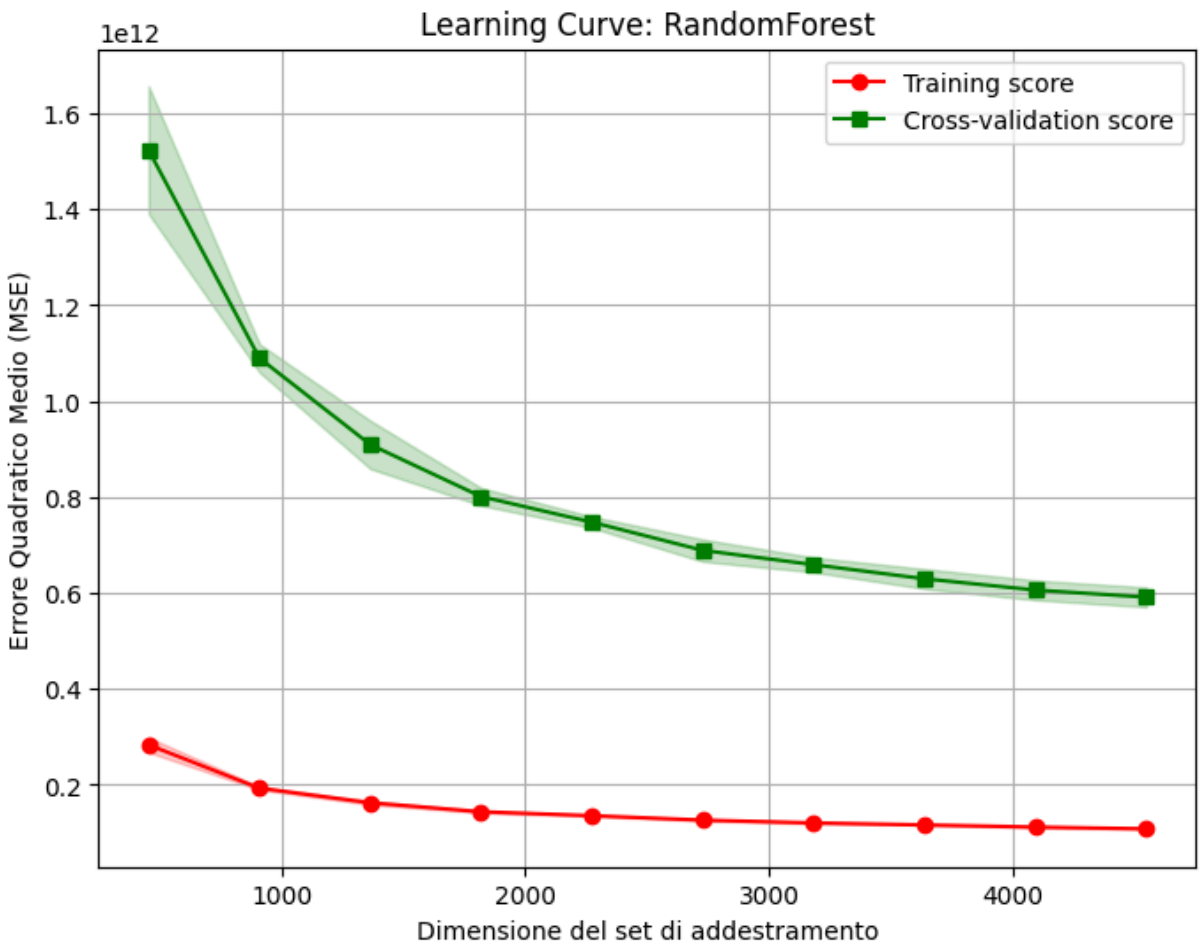
L'**errore quadratico medio (MSE)** misura la media dei quadrati degli errori, ovvero la differenza tra i valori previsti dal modello e i valori reali.

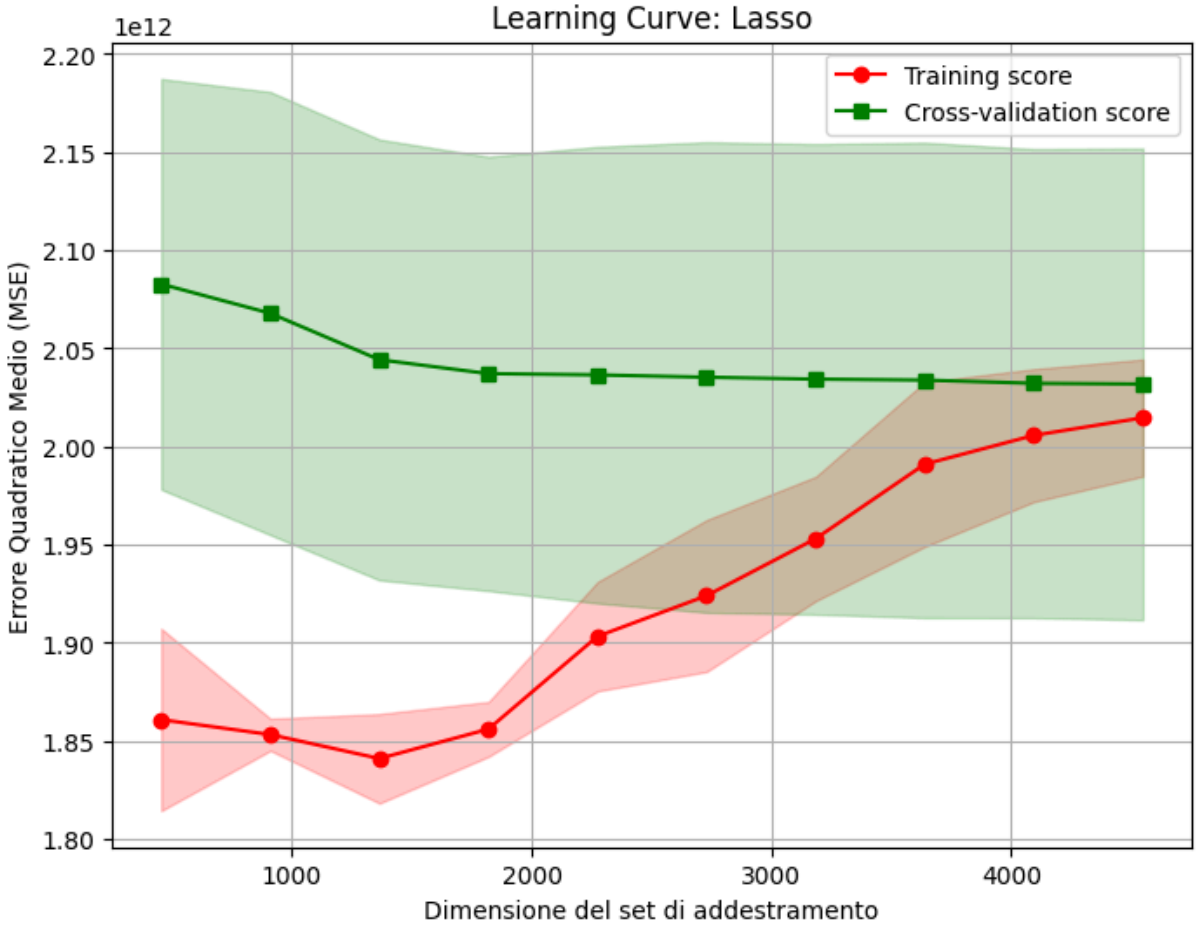
L'**errore assoluto medio (MAE)** è la media dei valori assoluti delle differenze tra i valori previsti e quelli reali.

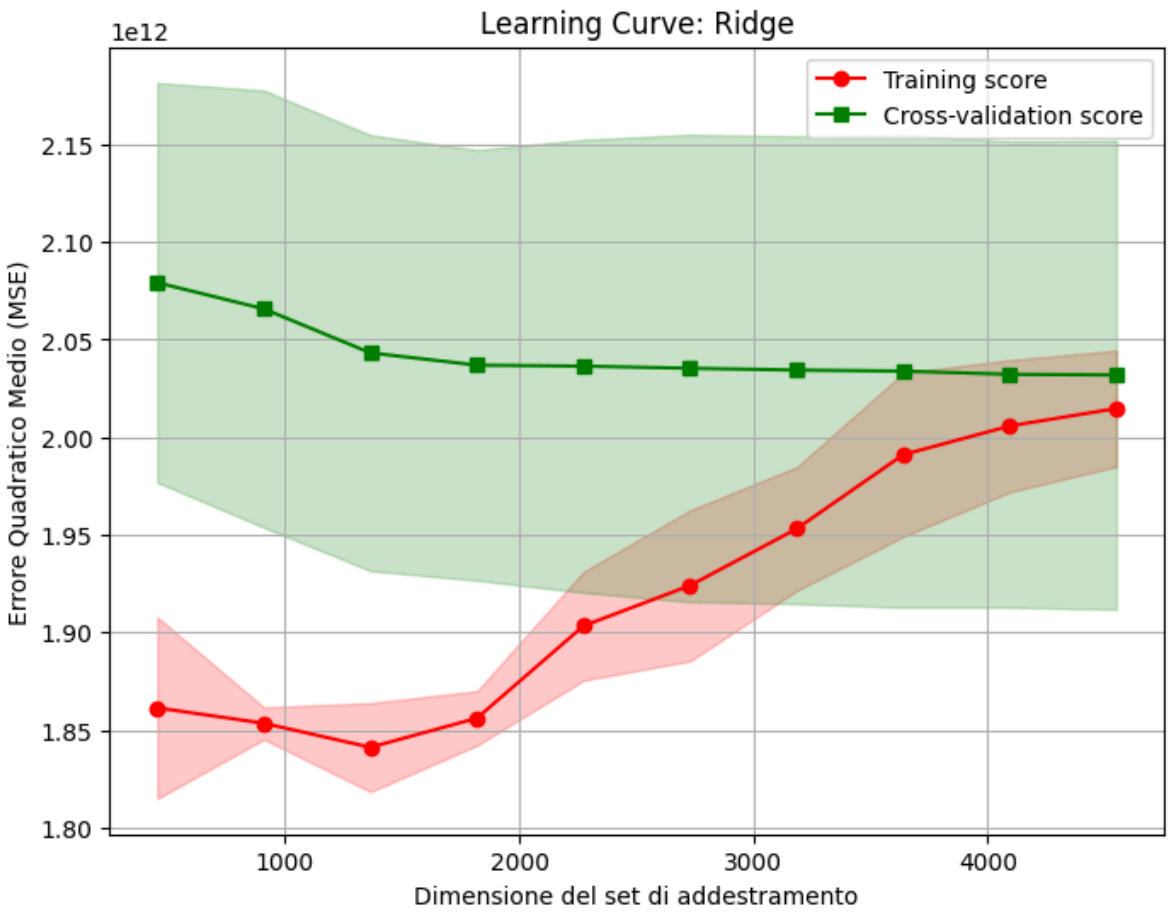
Il **coefficiente di determinazione** R^2 misura la percentuale di varianza nei dati che è spiegata dal modello di regressione.

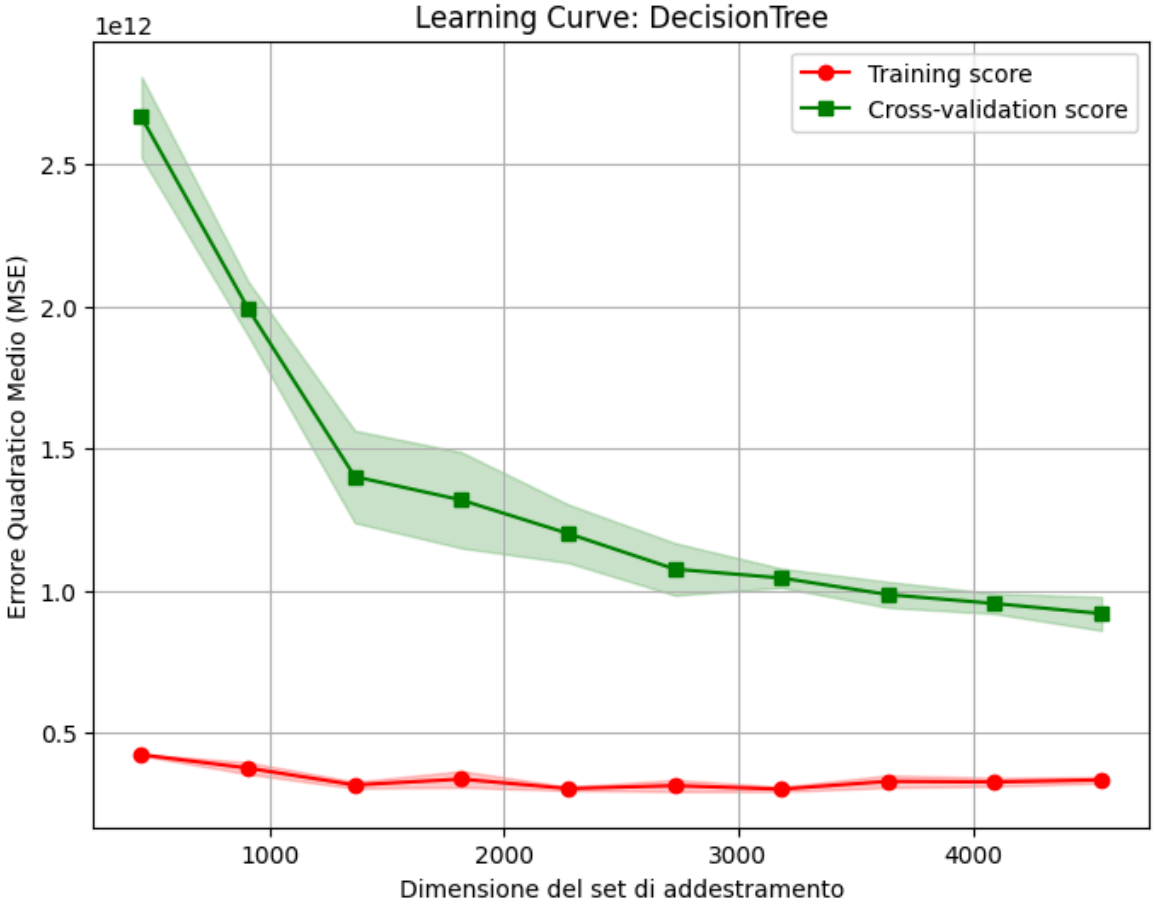


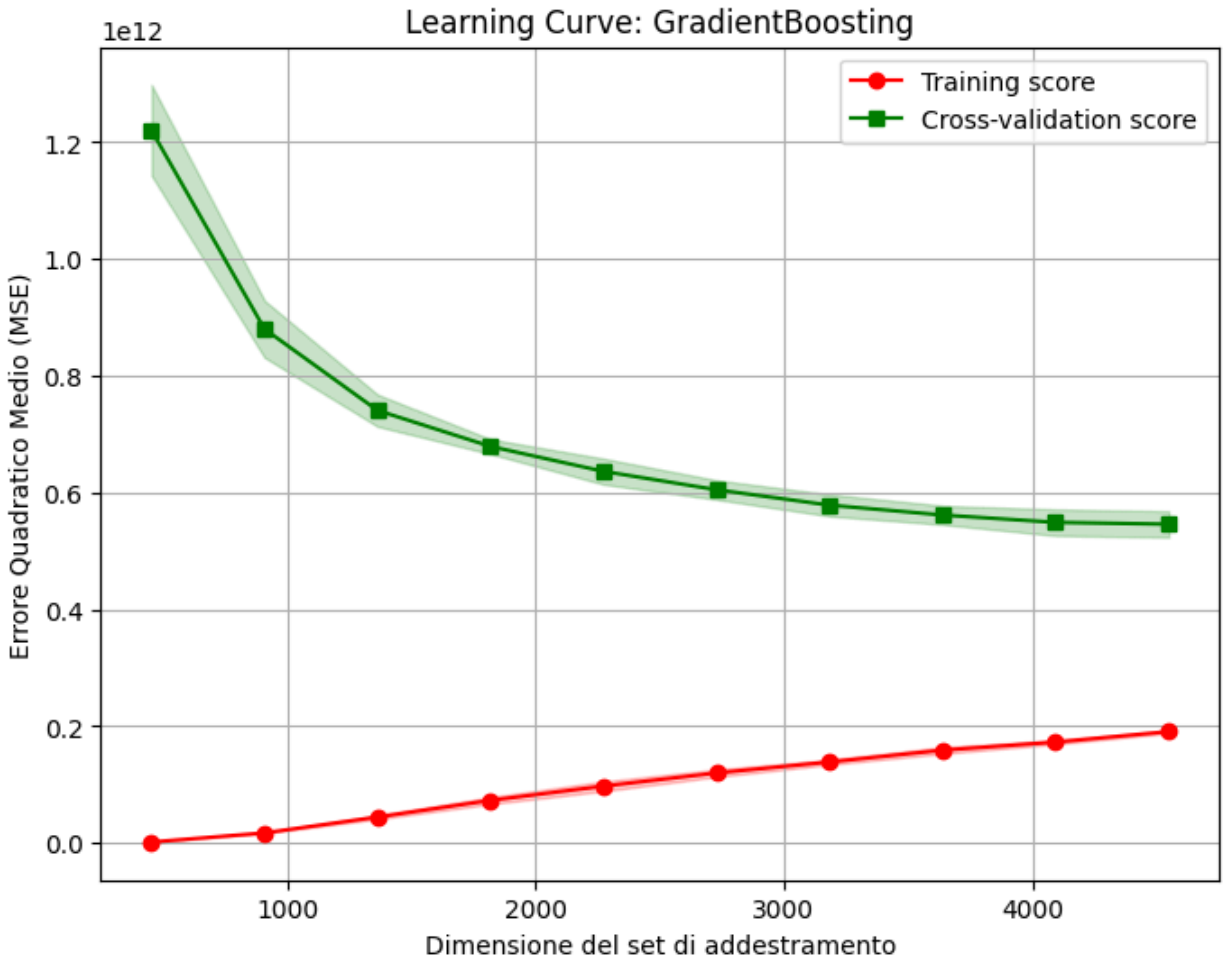
Come si può notare GradientBoosting ha l'errore MSE e R^2 più basso(0.9585) tra tutti i modelli, suggerendo prestazioni complessivamente eccellenti e una maggiore accuratezza nelle predizioni rispetto agli altri modelli.
Ora andiamo ad analizzare le curve di apprendimento per ogni modello.


Learning Curve: RandomForest
1e12
Training score
Cross-validation score
Errore Quadratico Medio (MSE)
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
1000
2000
3000
4000
Dimensione del set di addestramento


Learning Curve: Lasso
1e12
Training score
Cross-validation score
Errore Quadratico Medio (MSE)
2.20
2.15
2.10
2.05
2.00
1.95
1.90
1.85
1.80
1000
2000
3000
4000
Dimensione del set di addestramento


Learning Curve: Ridge
1e12
Training score
Cross-validation score
Errore Quadratico Medio (MSE)
2.15
2.10
2.05
2.00
1.95
1.90
1.85
1.80
1000
2000
3000
4000
Dimensione del set di addestramento


Learning Curve: DecisionTree
1e12
Training score
Cross-validation score
Errore Quadratico Medio (MSE)
2.5
2.0
1.5
1.0
0.5
1000
2000
3000
4000
Dimensione del set di addestramento



Dai grafici analizzati, emerge che i modelli Random Forest e Gradient Boosting generalmente presentano prestazioni migliori rispetto a Lasso, Ridge e Decision Tree. I primi due mostrano una minore tendenza all'overfitting e una maggiore capacità di generalizzare i dati.

Varianza e deviazioni standard

La **varianza** è una misura della dispersione dei dati rispetto alla media. Indica quanto i valori di un insieme di dati differiscano tra loro e dalla media del dataset. Matematicamente, è calcolata come la media dei quadrati delle differenze di ciascun valore dalla media del dataset. Una varianza più alta indica che i dati sono più dispersi, mentre una varianza più bassa indica che i dati sono più concentrati attorno alla media.

La **deviazione standard** è la radice quadrata della varianza e rappresenta anch'essa la dispersione dei dati rispetto alla media, ma è espressa nella stessa unità dei dati originali. È utile per capire quanto, mediamente, i valori si discostano dalla media. Poiché è calcolata come radice quadrata, risulta più intuitiva della varianza in termini pratici di dispersione dei valori.

```
Deviazioni standard e varianze per il modello RandomForest:
 Train Size    Train STD    Train VAR     Test STD     Test VAR
        454 1.784182e+10 3.183307e+20 1.250608e+11 1.564020e+22
        909 3.201615e+09 1.025034e+19 4.521985e+10 2.044834e+21
       1364 6.475591e+09 4.193328e+19 6.243432e+10 3.898044e+21
       3184 1.890192e+09 3.572826e+18 1.946891e+10 3.790386e+20
```

```
       3639 1.416919e+09 2.007659e+18 2.107380e+10 4.441051e+20
       4094 2.322922e+09 5.395967e+18 2.000350e+10 4.001401e+20
       4549 1.761217e+09 3.101885e+18 2.230332e+10 4.974381e+20

Deviazioni standard e varianze per il modello Lasso:
 Train Size    Train STD    Train VAR     Test STD     Test VAR
        454 4.631693e+10 2.145258e+21 1.045591e+11 1.093261e+22
        909 8.091419e+09 6.547106e+19 1.127972e+11 1.272320e+22
       1364 2.271453e+10 5.159501e+20 1.122022e+11 1.258934e+22
       3184 3.166375e+10 1.002593e+21 1.197519e+11 1.434053e+22
       3639 4.193540e+10 1.758578e+21 1.210064e+11 1.464255e+22
       4094 3.380367e+10 1.142688e+21 1.193803e+11 1.425165e+22
       4549 2.973847e+10 8.843767e+20 1.200750e+11 1.441800e+22

Deviazioni standard e varianze per il modello Ridge:
 Train Size    Train STD    Train VAR     Test STD     Test VAR
        454 4.639340e+10 2.152347e+21 1.023098e+11 1.046729e+22
        909 8.142787e+09 6.630497e+19 1.118771e+11 1.251649e+22
       1364 2.269788e+10 5.151938e+20 1.115515e+11 1.244374e+22
       3184 3.166300e+10 1.002546e+21 1.196994e+11 1.432795e+22
       3639 4.193502e+10 1.758546e+21 1.209361e+11 1.462555e+22
       4094 3.380336e+10 1.142667e+21 1.193109e+11 1.423509e+22
       4549 2.973825e+10 8.843634e+20 1.200284e+11 1.440682e+22

Deviazioni standard e varianze per il modello DecisionTree:
 Train Size    Train STD    Train VAR     Test STD     Test VAR
        454 3.294321e+09 1.085255e+19 1.628922e+11 2.653387e+22
        909 2.068246e+10 4.277642e+20 1.018728e+11 1.037807e+22
       1364 1.169943e+10 1.368767e+20 1.749138e+11 3.059483e+22
       3184 8.676393e+09 7.527980e+19 4.157403e+10 1.728400e+21
       3639 2.159552e+10 4.663666e+20 4.657471e+10 2.169204e+21
       4094 1.495878e+10 2.237651e+20 4.450341e+10 1.980553e+21
       4549 1.126693e+10 1.269436e+20 6.258787e+10 3.917242e+21

Deviazioni standard e varianze per il modello GradientBoosting:
 Train Size    Train STD    Train VAR     Test STD     Test VAR
        454 1.093940e+08 1.196705e+16 8.256308e+10 6.816661e+21
        909 7.219902e+08 5.212699e+17 4.816292e+10 2.319666e+21
       1364 3.414059e+09 1.165580e+19 2.662807e+10 7.090543e+20
       3184 4.417245e+09 1.951205e+19 1.867040e+10 3.485839e+20
       3639 5.787218e+09 3.349190e+19 1.464044e+10 2.143426e+20
       4094 3.493253e+09 1.220281e+19 2.260143e+10 5.108246e+20
       4549 2.751780e+09 7.572294e+18 2.248819e+10 5.057186e+20
```

La varianza del training in tutti i modelli tende generalmente a essere inferiore a quella del test, il che è comune e indica che i modelli si adattano meglio ai dati di addestramento.
La varianza elevata nel test rispetto al training suggerisce una difficoltà del modello a generalizzare, evidenziando la possibilità di overfitting.

In generale, *GradientBoosting* appare più stabile e meno incline a varianza elevata, rendedolo potenzialmente migliore per questo problema.
Soluzioni per gli altri modelli: *DecisionTree e RandomForest* mostrano un'alta varianza, in particolare su set più piccoli, e potrebbero beneficiare di tecniche di pruning o del tuning di parametri per migliorare la generalizzazione, invece *Lasso e Ridge* presentano una buona

costanza, ma potrebbeero migliorare con regularizzazione per ridurre ulteriormente la varianza nel test.

Calcolo del prezzo con il miglior modello (gradientboosting)

Vengono richiesti in input all'utente alcuni dati delle feature, successivamente viene creato un dataframe e viene passato al modello i parametri inseriti e tramite la funzione predici_prezzo, il modello calcolerà il valore di output(prezzo). Esempio di output generato

```
- Area: Adyar
- Tipo di Edificio: House
- Facilità di Parco: No
- Metri Quadrati Interni: 3376.92
- Numero di Camere da Letto: 2
- Numero di Bagni: 3
- Totale Stanze: 6
- Qualità Stanze (0-10): 2.28
- Qualità Bagno (0-10): 3.32
- Qualità Camera (0-10): 0.59
- Qualità Complessiva (0-10): 3.75
- Tassa di Registrazione (INR): 12710.13
- Commissione (INR): 5119.25
- Anno di Vendita: 2010
- Anno di Costruzione: 1989
- Mese di Vendita: 12

### Prezzo Previsto della Casa: 61,956.56 Euro
```

4. BASE DI CONOSCENZA E PROLOG

Conversione da albero a prolog

Per il ragionamento logico viene costruita una cosiddetta Knowledge Base, contenete gli assiomi(ovvero affermazioni sempre vere),costituiti da fatti e regole. I fatti indicano verità immutabili, invece una regola è una dichiarazione che afferma che qualcosa è vero in base ad altre cose che sono vere.

Successivamente l'albero 'decision tree' precedentemente addestrato, viene convertito in prolog, linguaggio di programmazione dichiarativo basato sul ragionamento logico, per creare fatti e regole. Libreria usata pyswip .

Regole e fatti

Nel progetto sono state create 3 regole e fatti 2.

- Un **fatto** di tipo '`leaf(NodeID, Prediction)`' viene generato per ogni nodo foglia dell'albero di decisione.

  Questo rappresenta un nodo terminale dell'albero dove viene effettuata una previsione (il valore della foglia).

- Un **fatto** di tipo ‘`node(NodeID, FeatureIndex, Threshold, LeftChild, RightChild)`’ viene creato per ogni nodo interno dell’albero.

  Questo rappresenta un nodo di decisione, ossia un punto in cui il percorso di previsione è suddiviso in base al valore di una caratteristica.

- **Regola** principale ‘`predire_prezzo(Features, Prezzo)`’ che utilizza una chiamata ricorsiva ‘`percorri_albero`’ per attraversare l’albero di decisione fino a raggiungere una foglia e ottenere unaa previsione.

  Questa regola:

  - Fa un confronto tra i valori di soglia e il valore di input delle caratteristiche.
  - Decide in quale direzione muoversi (figlio sinistro o destro) per arrivare alla previsione.

- ```
  predire_prezzo(Features, Prezzo) :-
      percorri_albero(0, Features, Prezzo).
  ```

- ```
  percorri_albero(NodeID, _, Predizione) :-
      leaf(NodeID, Predizione).
  ```

- ```
  percorri_albero(NodeID, Features, Predizione) :-
      node(NodeID, FeatureIndex, Threshold, LeftChild, RightChild),
      nth0(FeatureIndex, Features, FeatureValue),
      (FeatureValue =< Threshold ->
          percorri_albero(LeftChild, Features, Predizione)
          percorri_albero(RightChild, Features, Predizione)).
  ```

ESEMPIO DI OUTPUT

```
Predizione Prolog: 2156875.0
Predizione Decision Tree: 2156875.0
Features usate per Prolog: [5, 4337, 1, 2, 4, 2, 1, 9, 7, 3, 3,
28833.12845182201, 3050.8218212634033, 2004, 2005, 2, 19, 0]
```

5. KNOWLEDGE GRAPH E ONTOLOGIE, SPARQL

Un’ontologia è una rappresentazione formale della conoscenza in un dominio specifico, strutturata in modo da facilitare l’interoperabilità e la comprensione da parte di sistemi automatizzati. Le ontologie sono composte da triple RDF (Resource Description Framework), che rappresentano la relazione tra due entità attraverso la struttura: soggetto, predicato e oggetto.

Le entità in un’ontologia sono rappresentate come individui, che sono istanze di classi. Le proprietà descrivono le relazioni tra individui o attribuiscono caratteristiche specifiche agli individui stessi.

Web semantico

Il web semantico è l'evoluzione del web verso un’infrastruttura basata sui significati, in cui i dati sono collegati tra loro in maniera standardizzata. Gli standard del web semantico includono RDF per la creazione di triple e OWL (Web Ontology Language) per la definizione avanzata di classi, proprietà e relazioni, strutturando in modo più preciso il significato delle informazioni.

SPARQL
```

SPARQL è il linguaggio di query per interrogare dataset RDF, permettendo di estrarre informazioni da ontologie mediante filtri e condizioni logiche. Le ontologie possono essere salvate in formato XML, rendendo i dati leggibili e interoperabili su diverse piattaforme.

Nel progetto viene creata un'ontologia RDF con l'obiettivo di aggiungere valori e attributi di una casa,uno di questi è il valore stimato della casa, effettuato con la predizione. Quindi vengono definite le classi 'House, Area, BuildType, Price, NumRooms, e NumBathrooms', rappresentative delle caratteristiche delle case. Inoltre, sono definite alcune proprietà come 'hasArea, hasBuildType, e hasPrice', per specificare attributi delle case. I dati vengono presi dal dataset preprocessato.

Con SPARQL, il codice estrae informazioni dall'ontologia, come le case in una determinata area e i loro prezzi.

La funzione inizia creando un nuovo grafo RDF e caricando l'ontologia esistente dal file XML. Viene formulata una query SPARQL che cerca tutti gli individui (case) che hanno la proprietà 'hasArea' con il valore "Adyar" e restituisce anche il prezzo previsto associato a ciascuna di queste case.

QUERY:

```
query = f"""

   PREFIX ns: <http://example.org/housing_ontology#>

   SELECT ?house ?price

   WHERE {{

       ?house ns:hasArea "{area}" .

       ?house ns:hasPrice ?price .

   }}

   """
```

INPUT:

```
predicted_price_inr = predict_with_gradient_boosting(2000, 4, 2, "Anna
Nagar", "Commercial", "Yes", 10000, 2000, 2020, 2010, 5, 4, 4, 2, 7, 6, 15,
"Near")
```

OUTPUT:

```
Prezzo previsto per la casa aggiunto all'ontologia: 93358.97 EUR
Casa: http://example.org/housing_ontology#House_2000_4_2, Prezzo: 93358.97
EUR
```

6. CONCLUSIONI

Complessivamente in questo progetto sono stati sviluppati modelli di apprendimento supervisionato per prevedere il prezzo di vendita di case/appartamenti basandoci su dataset online,

Il modello di Gradient Boosting ha mostrato le migliori performance in termini di errore medio assoluto (MAE), seguito da Random Forest e Decision Tree.

I modelli di regressione lineare hanno mostrato prestazioni inferiori rispetto agli altri.

È stata creata una base di conoscenza per stimare il prezzo di una casa, usando un albero decisionale o un calcolo “manuale” basandosi dai valori delle caratteristiche.

È stata esplorata un’ontologia con varie relazioni, proprietà e classi per permettere di descrivere e rappresentare la conoscenza in un dominio, facilitando la comprensione e la condivisione delle informazioni tra diversi sistemi e utenti, per poter effettuare altre operazione sui dati.

POSSIBILI SVILUPPI

Per sviluppare ulteriormente il progetto, si potrebbero testare modelli di apprendimento supervisionato più sofisticati per migliorare ulteriormente l’accuratezza delle predizioni.

Si potrebbe introdurre la grafica per poter prendere i dati di input dagli utenti per utilizzare le predizioni.

Si possono creare nuove applicazioni basate su intelligenza artificiale per poter sfruttare a pieno le ontologie e web semantico, con il ragionamento automatico.

RIFERIMENTI BIBLIOGRAFICI

Apprendimento supervisionato: D. Poole, A. Mackworth: Artificial Intelligence: Foundations of Computational Agents. 3nd ed. Cambridge University Press [Ch.7]

Basi di conoscenza: D. Poole, A. Mackworth: Artificial Intelligence: Foundations of Computational Agents. 3nd ed. Cambridge University Press [Ch.15]

Grafi di conoscenza e ontologie: D. Poole, A. Mackworth: Artificial Intelligence: Foundations of Computational Agents. 3nd ed. Cambridge University Press [Ch.16]