UniBa | UNIVERSITÀ DEGLI STUDI DI BARI ALDO MORO

Progetto per Ingegneria della Conoscenza

Capuano Marzia, [758281], m.capuano24
Di Maggio Valerio Salvatore, [760424], v.dimaggio10

Link Repository: https://github.com/val-2/ICon.git

A.A 2023-2024

Indice

Introduzione

Il dataset considerato per questo progetto si può trovare al link https://data.world/data-society/used-cars-data. Contiene caratteristiche di macchine usate in vendita tramite gli annunci di eBay. Tramite queste caratteristiche si vuole costruire un modello di apprendimento automatico in grado di prevedere il prezzo di vendita e una base di conoscenza per poter stabilire se il prezzo proposto è conveniente o meno.

Strumenti utilizzati

Il caso di studio è stato sviluppato tramite l'utilizzo di Python, un linguaggio di programmazione che offre molte librerie per trattare ed analizzare dati. L'IDE utilizzato è stato PyCharm e Visual Studio Code.

Inoltre sono state utilizzate diverse librerie, tra cui:

- Pandas: utile per l'analisi dei dati e il preprocessing
- Numpy: per il calcolo scientifico, fornisce supporto per array a più dimensioni e funzioni matematiche
- Scikit_learn: 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.

Capitolo 1) Dataset e preprocessing

Il dataset contiene i seguenti campi:

- **dateCrawled**: La data in cui l'annuncio è stato rilevato per la prima volta.
- **name**: Il nome del modello dell'auto.
- **seller**: Il tipo di venditore (es. privato).
- **offerType**: Il tipo di offerta.
- **price**: Il prezzo del veicolo.
- **abtest**: Indica se l'annuncio era parte di un test A/B.
- **vehicleType**: Il tipo di veicolo (es. coupe, suv, etc...).
- **yearOfRegistration**: L'anno di immatricolazione del veicolo.
- **gearbox**: Il tipo di cambio (es. manuale, automatico).
- **powerPS**: La potenza del veicolo in PS (Pferdestärke, equivalente a CV o HP).
- **model**: Il modello specifico del veicolo.
- **kilometer**: Il chilometraggio del veicolo.
- **monthOfRegistration**: Il mese di immatricolazione del veicolo.
- **fuelType**: Il tipo di carburante (es. benzina, diesel).
- **brand**: Il marchio del veicolo.
- **notRepairedDamage**: Indica se il veicolo ha danni non riparati.
- **dateCreated**: La data di creazione dell'annuncio.

- **nrOfPictures**: Il numero di immagini presenti nell'annuncio.
- **postalCode**: Il codice postale della località in cui si trova il veicolo.
- **lastSeen**: L'ultima volta che l'annuncio è stato visto online**.**

Alcuni dei valori categorici di questo dataset sono in lingua tedesca.

Pre-processing del dataset

Si ha come obiettivo l'adattamento del dataset per gli algoritmi di apprendimento supervisionato, l'eliminazione di valori nulli e di feature categoriche per semplificare l'uso della libreria scikit-learn.

Viene eseguita l'eliminazione di un insieme di colonne di minore rilevanza, specificamente: **dateCrawled**, **dateCreated**, **nrOfPictures**, **lastSeen**, **postalCode**, **seller**, **offerType**, **abtes**, **monthOfRegistration**. Inoltre, vengono rimosse anche le colonne **name** e **model**, poiché la loro trasformazione mediante one-hot encoding comporterebbe la creazione di un numero eccessivo di colonne.

La colonna **Gearbox,** che indica il tipo di cambio, contiene solo due valori, pertanto viene convertita in feature booleana rinominando la colonna in **gearboxAutomatic** e trasformando "automatik" in True e "manuell" in False.

La colonna **notRepairedDamage**, che Indica se il veicolo ha danni non riparati, viene convertita in una feature booleana trasformando "ja" in True e "nein" in False.

La colonna **powerPS**, che indica la potenza del veicolo viene rinominata in 'powerHP'.

Per la colonna **brand**, che indica il marchio del veicolo, vengono mantenuti solo i brand che rappresentano almeno 1.5% delle voci totali, per non avere classi di brand troppo sbilanciate nel dataset.

Nella colonna **vehicleType**, che descrive il tipo di veicolo vengono effettuate delle rinominazioni dei valori, in particolare:

- andere -> other
- kleinwagen -> citycar
- kombi -> stationwagon
- limousine ->
- andere -> other

Inoltre vengono mantenuti solo i tipi di veicoli che appaiono in più del 3% delle voci.

Della colonna **fuelType**, che Indica il tipo di carburante, vengono mantenuti solo i tipi di carburante con più dell'1% delle voci, per lo stesso motivo delle colonne brand e vehicleType. Essendoci adesso solo due valori possibili, viene convertita in una feature booleana trasformando "diesel" in True e "benzin" in False e viene rinominata in fuelTypeDiesel

Sono eliminati i valori anomali nel prezzo, selezionando solo le righe in cui il valore della colonna **price** è compreso tra 100 e 100.000. Similmente, per la colonna **powerHP**, vengono mantenute solo le righe in cui il valore è inferiore a 500.

Dopo tutto ciò, vengono eliminate dal dataset tutte le colonne che contengono valori nulli, per semplificare l'uso degli algoritmi di apprendimento supervisionato.

Infine viene utilizzato il one-hot encoding per le colonne categoriche rimanenti (veichleType e brand), in modo da avere esclusivamente feature continue o booleane.

Se invece il dataset è destinato all'uso per l'apprendimento bayesiano è necessario che tutte le colonne siano discrete. Dunque, al posto di applicare one-hot encoding alle colonne categoriche, viene usata la discretizzazione sulle colonne continue. Ciò viene realizzato tramite il metodo *qcut* di *pandas*, basato sulla divisione dei valori in quantili, nel nostro caso in 10 quantili. Ciò viene applicato sulle colonne **powerHP, yearOfRegistration** e **price**.

Complessivamente dal dataset sono estratti 30 000 esempi.

Capitolo 2) Apprendimento Supervisionato

L'apprendimento supervisionato è una delle tecniche più usate nel campo dell'apprendimento automatico. In questo contesto, l'apprendimento si riferisce alla capacità di un agente di migliorarsi attraverso l'esperienza. Questo miglioramento può manifestarsi in vari modi: l'agente può acquisire nuove competenze, eseguire compiti esistenti con maggiore efficacia, o completare attività più rapidamente.

L'apprendimento supervisionato si basa su esempi costituiti da coppie di input-output, dove l'output, o target, è noto durante la fase di addestramento. Il compito dell'algoritmo è di apprendere una funzione che, dato un nuovo input, possa predire correttamente l'output corrispondente.

A seconda della tipo di target, i problemi di apprendimento supervisionato si suddividono in tre categorie principali:

1. Classificazione, quando il target è discreto.
2. Regressione, quando il target è continuo.
3. Predizione strutturata, quando il target è una struttura dati complessa.

Scelta dei modelli

Il nostro obiettivo è predire il prezzo di vendita a partire da tutti gli altri attributi della macchina usata. Avendo il prezzo un valore continuo, questo è un task di regressione.

Sono stati utilizzati tre modelli di apprendimento automatico:

- **DecisionTreeRegression**

  Gli alberi decisionali sono modelli di machine learning che fanno predizioni basate su condizioni specifiche. Un albero decisionale è strutturato in modo che ogni nodo intermedio rappresenti una condizione che può essere valutata come vera o falsa, con due rami corrispondenti a ciascun esito. Le foglie dell'albero contengono le predizioni finali del modello. Questi alberi possono essere utilizzati per compiti di classificazione, quando le foglie rappresentano categorie discrete, o per compiti di regressione, quando le foglie forniscono valori continui, come nel nostro caso.

  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.

- **RandomForestRegression**

  Rappresenta un metodo composito costituito da un insieme di alberi decisionali. L'idea alla base di questo modello è quella di addestrare un numero significativo di alberi utilizzando parti diverse del dataset di addestramento e aggregare le loro previsioni per ottenere una previsione finale per ciascun esempio. Per garantire

l'efficacia del Random Forest, è essenziale che gli alberi che la compongono producano previsioni diverse. Ci sono diverse strategie per ottenere questa diversità, tra cui l'utilizzo di sottoinsiemi casuali degli esempi di addestramento (bagging) o delle feature per ciascun albero.

- **GradientBoostingRegression**

  È una tecnica che combina più modelli deboli per creare un modello più robusto. I modelli deboli vengono costruiti sequenzialmente, in modo che ognuno corregga gli errori commessi dai precedenti. L'idea alla base del gradient boosting è di ottimizzare iterativamente una loss, aggiungendo ogni volta un nuovo modello che la minimizza. Ad ogni iterazione, gli errori di predizione vengono calcolati e il nuovo modello viene allenato per predire questi residui, migliorando così la precisione del modello complessivo. Questo processo continua fino a che non si raggiunge un livello di accuratezza desiderato o un numero massimo di modelli deboli.

Scelta degli iper-parametri

Per la scelta degli iper-parametri di ogni modello si utilizza la Grid Search con 5-Fold Cross Validation ripetuta 2 volte, in modo da esplorare lo spazio degli iper-parametri assicurandosi che i risultati siano effettivamente validi e non frutto di casualità. È stato scelto questo numero di fold per il grande numero di esempi nel dataset con l'obiettivo di mantenere la durata dell'esecuzione entro un periodo di tempo ragionevole.

```
cv = RepeatedKFold(n_splits=5, n_repeats=2)
```

```
    grid_search = GridSearchCV(model, param_grid, cv=cv, scoring=scoring,
n_jobs=-1)
    grid_search.fit(X, y)
```

Ecco i parametri esplorati per ogni modello:

DecisionTreeRegression

```
dtr_param_grid = {
    'criterion': ["friedman_mse", "poisson"],
    'splitter': ['best'],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10, 20],
    'min_samples_leaf': [1, 2, 5],
}
```

- **criterion**: La funzione di valutazione della qualità di una divisione.

- **splitter**: La strategia utilizzata per scegliere la divisione in ogni nodo. Sceglie la migliore divisione possibile con “best”.

- **max_depth**: La profondità massima dell'albero. Con None l’albero cresce fino a che tutte le foglie contengono meno di min_samples_leaf campioni; con un intero indica la profondità massima dell'albero, per ridurre l'overfitting.

- **min_samples_split**: Il numero minimo di campioni richiesto per dividere un nodo.

- **min_samples_leaf**: Il numero minimo di campioni che deve essere presente in una foglia.

RandomForestRegression

```
rfr_param_grid = {
    'n_estimators': [100, 200],
    'criterion': ["friedman_mse", "poisson"],
    'max_depth': [None, 5, 10],
    'min_samples_split': [2, 5, 10, 20],
    'min_samples_leaf': [1, 2, 5],
}
```

- **n_estimators**: Il numero di alberi nella foresta. Maggiore è il numero di alberi, migliore è la performance fino a un certo punto, ma aumenta anche il tempo di addestramento.

- **criterion**: La funzione di valutazione della qualità di una divisione.

- **max_depth**: La profondità massima del singolo albero. Con None l’albero cresce fino a che tutte le foglie contengono meno di min_samples_leaf campioni; con un intero indica la profondità massima dell'albero, per ridurre l'overfitting.

- **min_samples_split**: Il numero minimo di campioni richiesto per dividere un nodo di un singolo albero.

- **min_samples_leaf**: Il numero minimo di campioni che deve essere presente in una foglia di un singolo albero.

GradientBoostingRegression

```
gbr_param_grid = {
    'n_estimators': [100, 200],
    'learning_rate': [0.01, 0.1, 0.5],
    'max_depth': [None, 5, 10],
    'min_samples_split': [2, 5, 10, 20],
    'min_samples_leaf': [1, 2, 5],
    'subsample': [0.1, 0.5, 1.0],
}
```

- **n_estimators**: Il numero di alberi nella sequenza di boosting. Un numero maggiore di alberi può migliorare la performance ma aumenta anche il tempo di addestramento.
- **learning_rate**: Il tasso di apprendimento, riduce il contributo di ciascun albero. Un valore più basso rende l'addestramento più lento ma può migliorare la generalizzazione.
- **max_depth**: La profondità massima del singolo albero. Con None l'albero cresce fino a che tutte le foglie contengono meno di min_samples_leaf campioni; con un intero indica la profondità massima dell'albero, per ridurre l'overfitting.
- **min_samples_split**: Il numero minimo di campioni richiesto per dividere un nodo di un singolo albero.
- **min_samples_leaf**: Il numero minimo di campioni che deve essere presente in una foglia di un singolo albero.
- **subsample**: La frazione di campioni da utilizzare per addestrare ciascun modello base.

La metrica di scoring scelta è: **neg_mean_absolute_error**. È la media degli errori assoluti (MAE), ma con segno negativo per conformità con le funzioni di scoring in scikit-learn che massimizzano i punteggi. Questa metrica risulta meno sensibile agli outlier rispetto alla RMSE.

Sperimentazione e valutazione

Si calcola una baseline basata su una predizione banale, in questo caso la mediana per MAE.

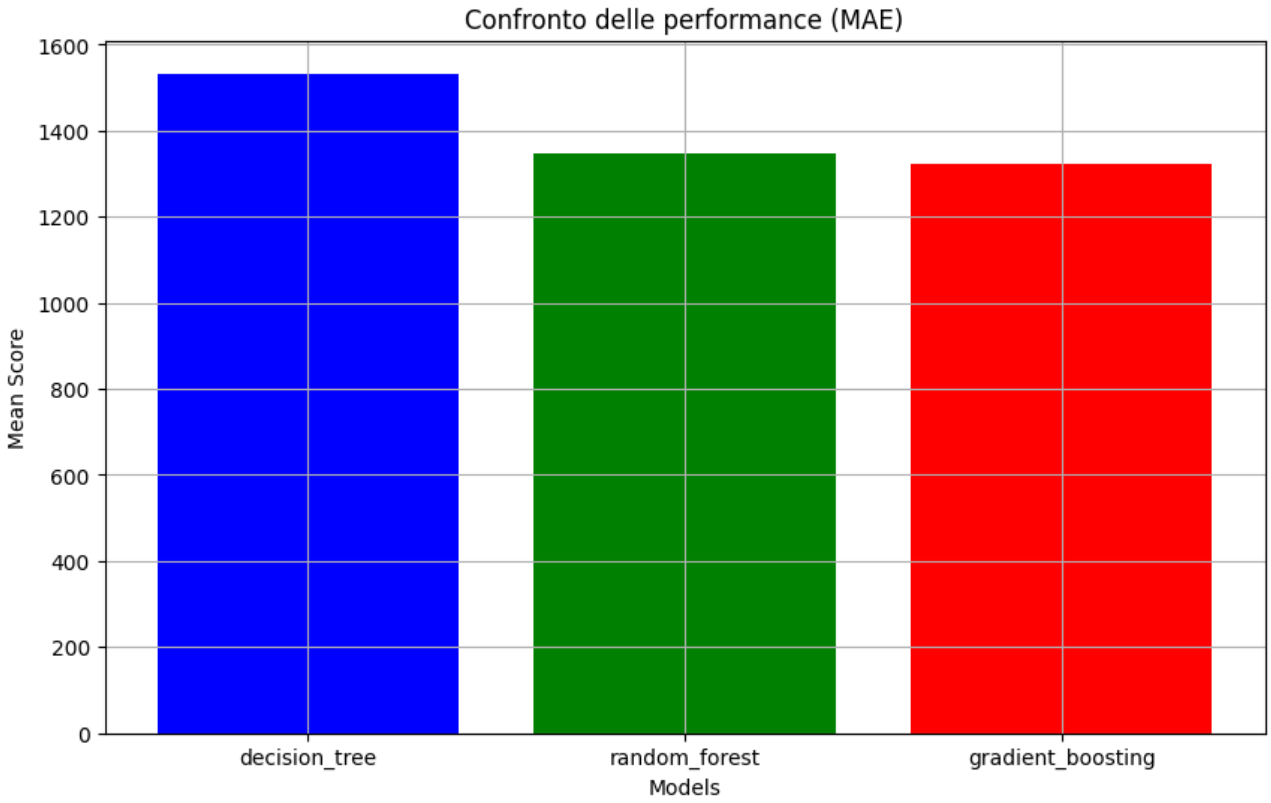
```
if scoring == "neg_root_mean_squared_error":
    baseline_prediction = df[target_column].mean()
    baseline_error = - (df[target_column] - baseline_prediction).pow(2).mean() ** 0.5
elif scoring == "neg_mean_absolute_error":
    baseline_prediction = df[target_column].median()
    baseline_error = - (df[target_column] - baseline_prediction).abs().mean()
```

```
Baseline for MAE: 16051.040010443357
```

Dopo di ciò, si esegue la GridSearch degli iperparametri, ottenendo per i modelli i seguenti risultati:

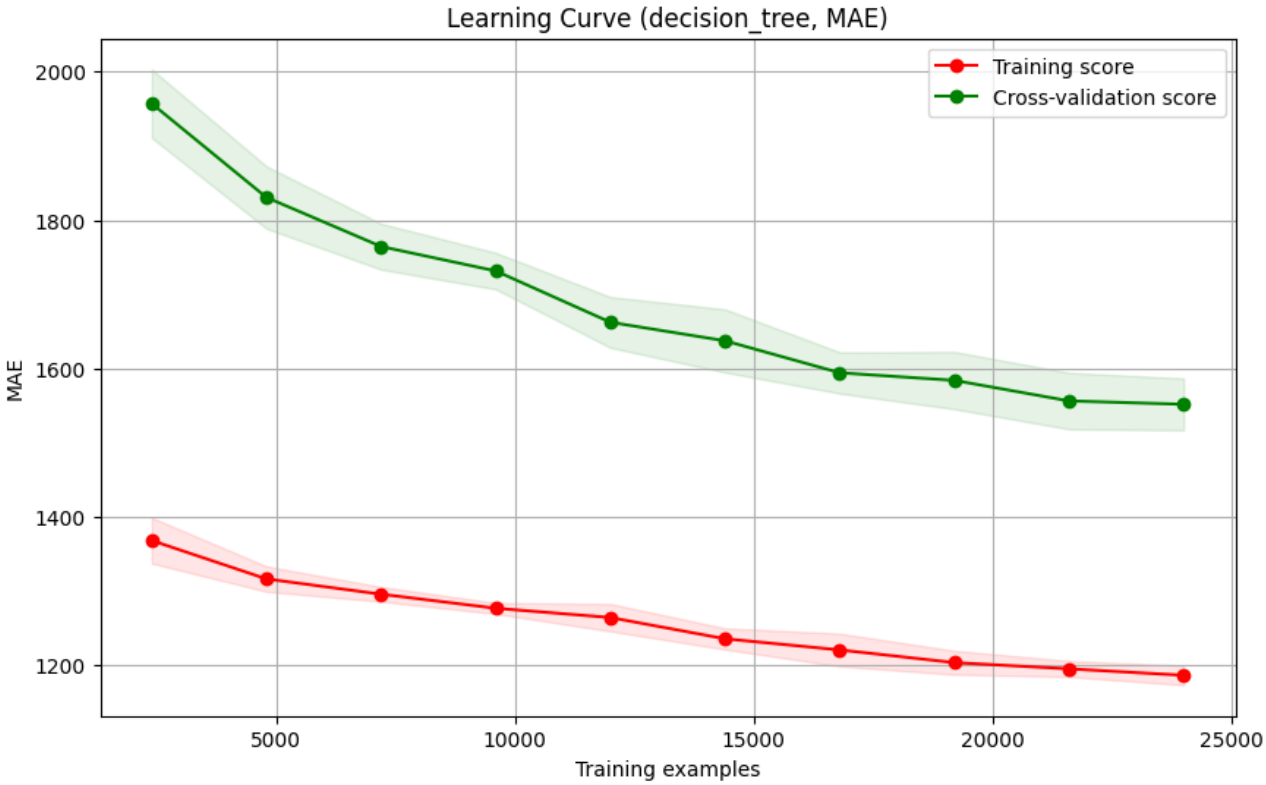
```
Error for Decisional Tree:  1532.6659102347028
Best params for Decisional Tree:  {'criterion': 'poisson', 'max_depth':
20, 'min_samples_leaf': 2, 'min_samples_split': 20, 'splitter': 'best'}

Error for Random Forest:  1347.547345157256
Best params for Random Forest:  {'criterion': 'poisson', 'max_depth':
None, 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 200}

Error for Gradient Boosting:  1321.2536159364656
Best params for Gradient Boosting:  {'learning_rate': 0.1, 'max_depth':
10, 'min_samples_split': 5, 'min_samples_leaf': 1, 'n_estimators': 100,
'subsample': 0.5}
```

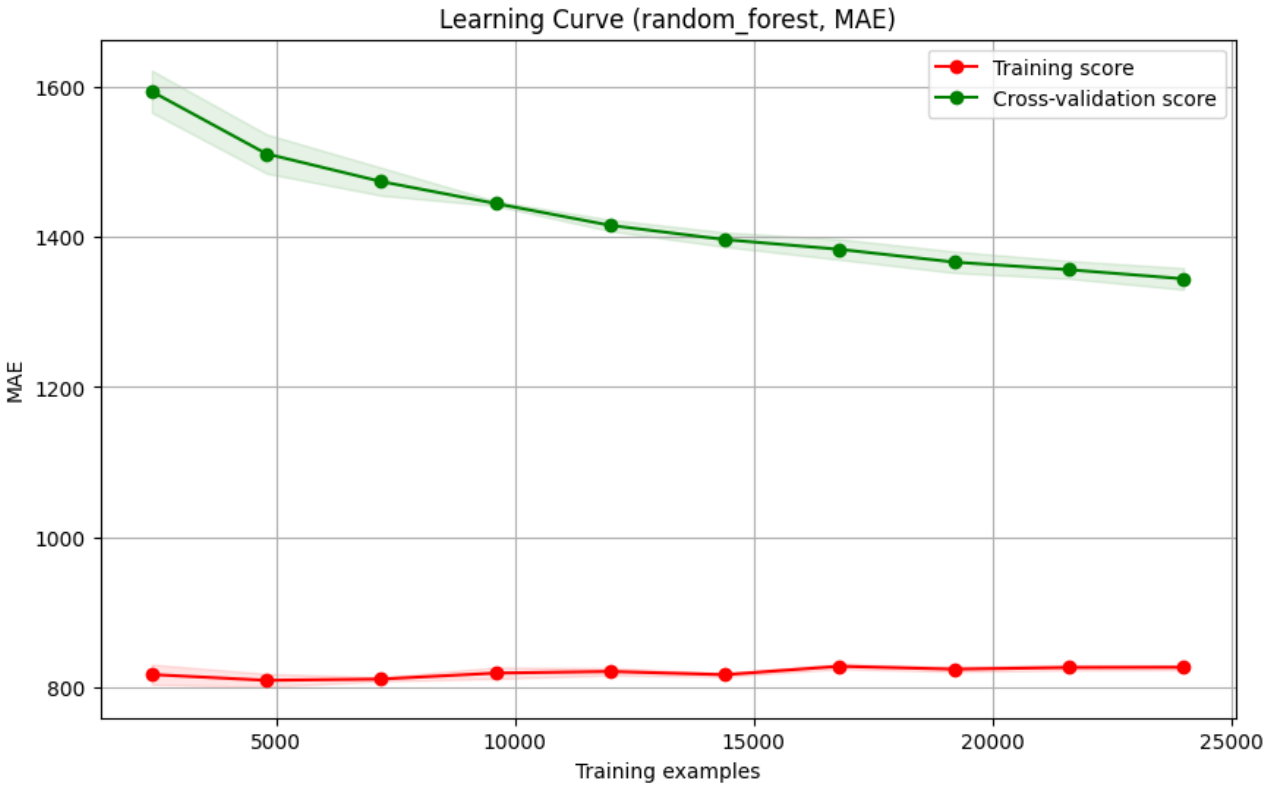
Usando questi iperparametri, ecco i grafici per le performance:

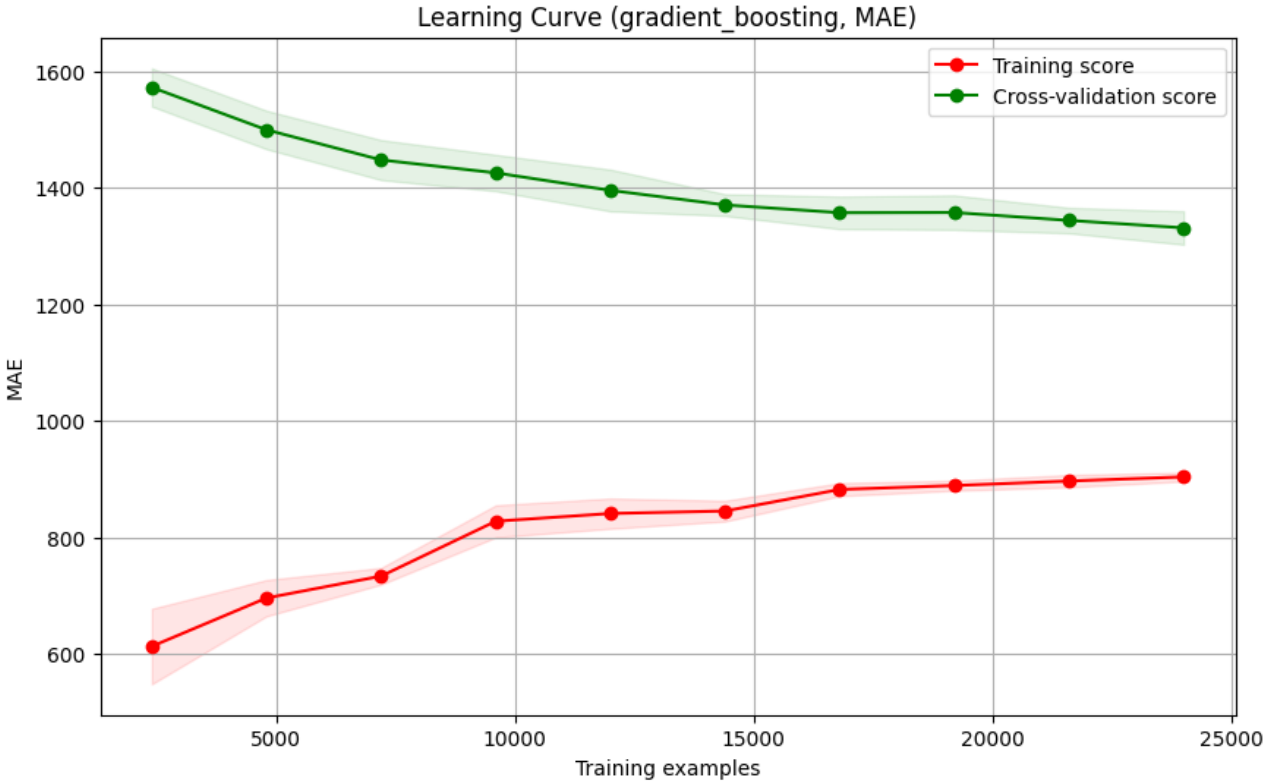


Come si può notare dai grafici, i due modelli più complessi hanno performance nettamente migliori rispetto all'albero decisionale, in particolare il modello Gradient Boosting.

Andiamo ad analizzare le curve di apprendimento di ogni modello con gli iper-parametri migliori trovati e la tabella delle varianze di errore di test e training set.


Learning Curve (decision_tree, MAE)
Training score
Cross-validation score
MAE
2000
1800
1600
1400
1200
5000
10000
15000
20000
25000
Training examples


Learning Curve (random_forest, MAE)
Training score
Cross-validation score
MAE
1600
1400
1200
1000
800
5000
10000
15000
20000
25000
Training examples



Model	Train Score Variance	Test Score Variance	Train Score Std dev	Test Score Std dev
decision_tree	174.339815	1222.833801	13.203780	34.969041
random_forest	12.277321	206.165469	3.503901	14.358463
gradient_boosting	57.856427	819.258402	7.606341	28.622690

Da questi grafici si può notare che l'errore di test diminuisce all'aumentare degli esempi, mentre l'errore di train tende a stabilizzarsi su un determinato valore, variabile in base al modello. Il modello Gradient Boosting ha l'errore di test minore.

Varianza

È una misura statistica che indica quanto i valori di un insieme di dati si discostano dalla loro media. In altre parole, la varianza quantifica la dispersione o la variabilità dei dati.

Nel nostro contesto, è preferibile che l'errore di test tra i vari fold sia il più simile possibile, ovvero che la loro varianza sia il più possibile vicina allo 0, in quanto significherebbe avere lo stesso livello di precisione delle previsioni indipendentemente dall'esempio usato.

Dalla tabella si può evincere che il modello più stabile sotto questo punto di vista è il Random Forest.

Regressione lineare e regolarizzazione

Sono stati provati anche modelli di regressione lineare, senza regolarizzazione e con regolarizzazione L1 e L2, usando le stesse modalità specificate al paragrafo precedente.

La ricerca è avvenuta sui seguenti iperparametri:

Linear Regression

```
lr_param_grid = {
    'fit_intercept': [True, False],
}
```

- **fit_intercept**: Specifica se il modello deve calcolare l'intercetta (bias) per i dati. Se False si assume che i dati siano già centrati rispetto all'origine.

Lasso (Regolarizzazione L1)

```
lasso_param_grid = {
    'alpha': [0.001, 0.01, 0.1, 1, 10, 100],
    'fit_intercept': [True, False],
    'max_iter': [1000, 10000, 50000],
    'selection': ['cyclic', 'random'],
}
```

- **alpha**: Il parametro di regolarizzazione, controlla la forza della regolarizzazione L1.
- **fit_intercept**: Specifica se il modello deve calcolare l'intercetta (bias) per i dati. Se False si assume che i dati siano già centrati rispetto all'origine.
- **max_iter:** Numero massimo di iterazioni per l'ottimizzazione.
- **selection:** Il metodo di selezione dei coefficienti durante l'aggiornamento dei coefficienti del modello.

Ridge (Regolarizzazione L2)

```
ridge_param_grid = {
    'alpha': [0.001, 0.01, 0.1, 1, 10, 100],
    'fit_intercept': [True, False],
    'max_iter': [1000, 10000, 50000],
}
```

- **alpha**: Il parametro di regolarizzazione, controlla la forza della regolarizzazione L1.
- **fit_intercept**: Specifica se il modello deve calcolare l'intercetta (bias) per i dati. Se False si assume che i dati siano già centrati rispetto all'origine.
- **max_iter:** Numero massimo di iterazioni per l'ottimizzazione.

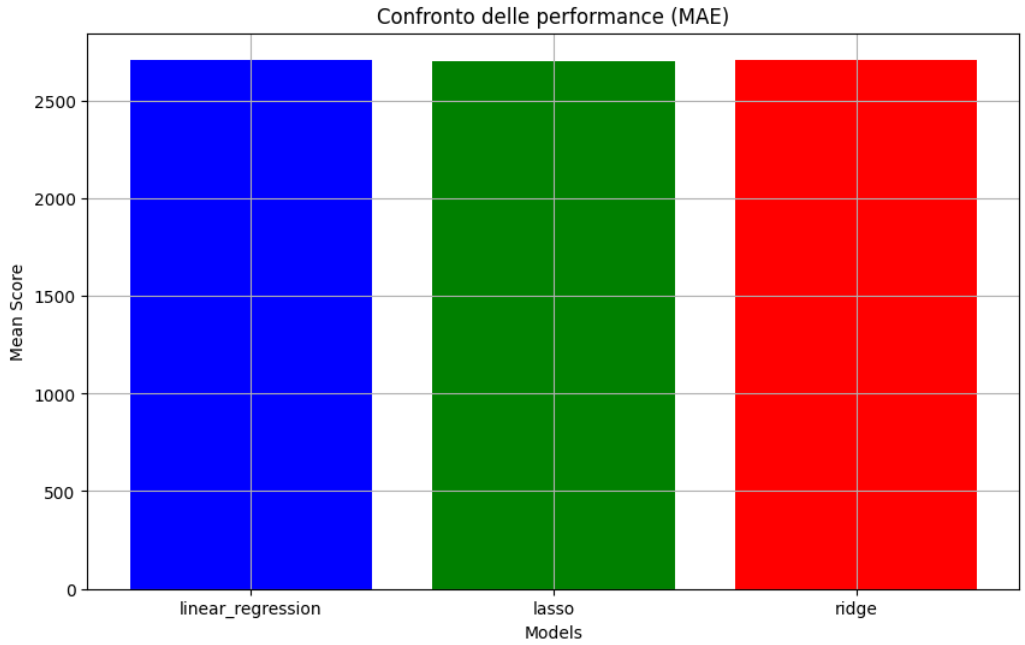
Ecco gli iperparametri trovati per ogni modello:

```
Error for Linear Regression:  2736.8988264189093
Best params for Linear Regression:  {'fit_intercept': True}
```

```
Error for Lasso:  2706.5548288907694
Best params for Lasso:  {'alpha': 10, 'fit_intercept': True, 'max_iter':
10000, 'selection': 'random'}
```

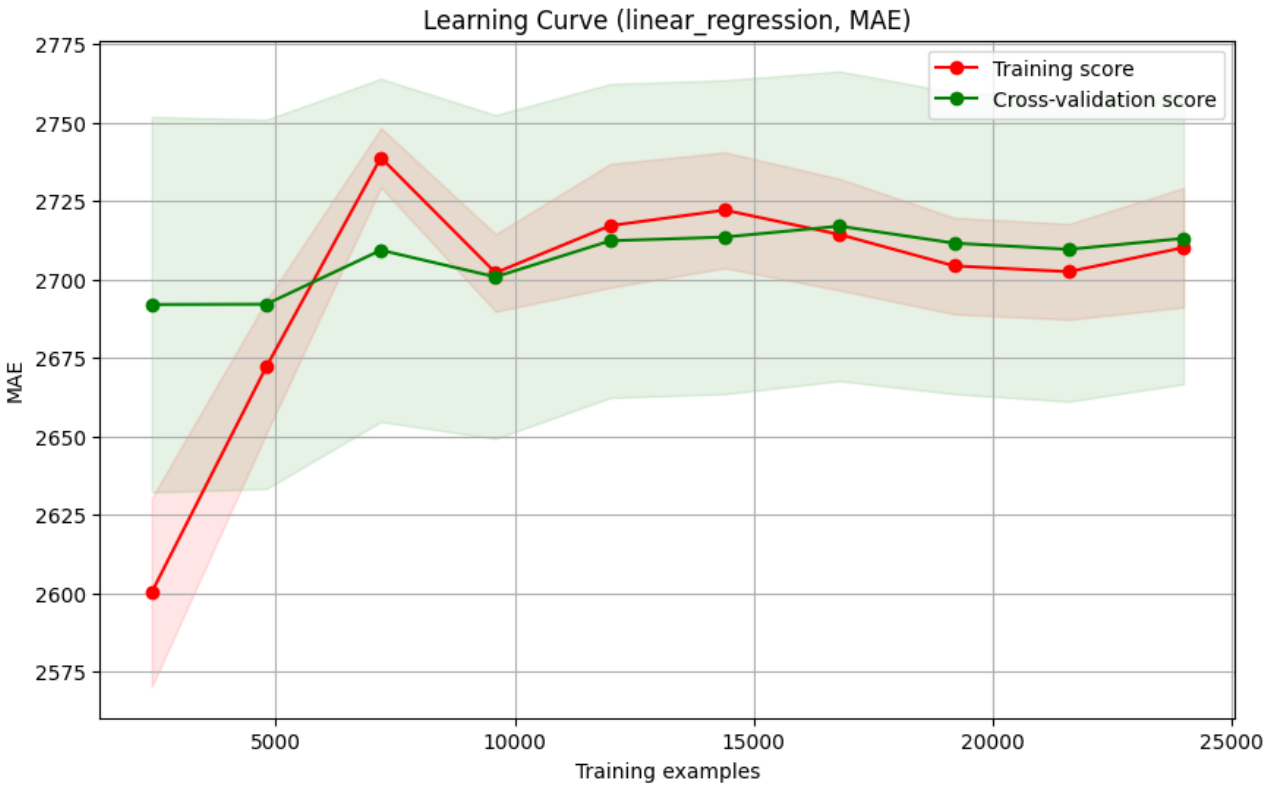
```
Error for Ridge:  2707.49252124905
Best params for Ridge:  {'alpha': 100, 'fit_intercept': True, 'max_iter':
1000}
```

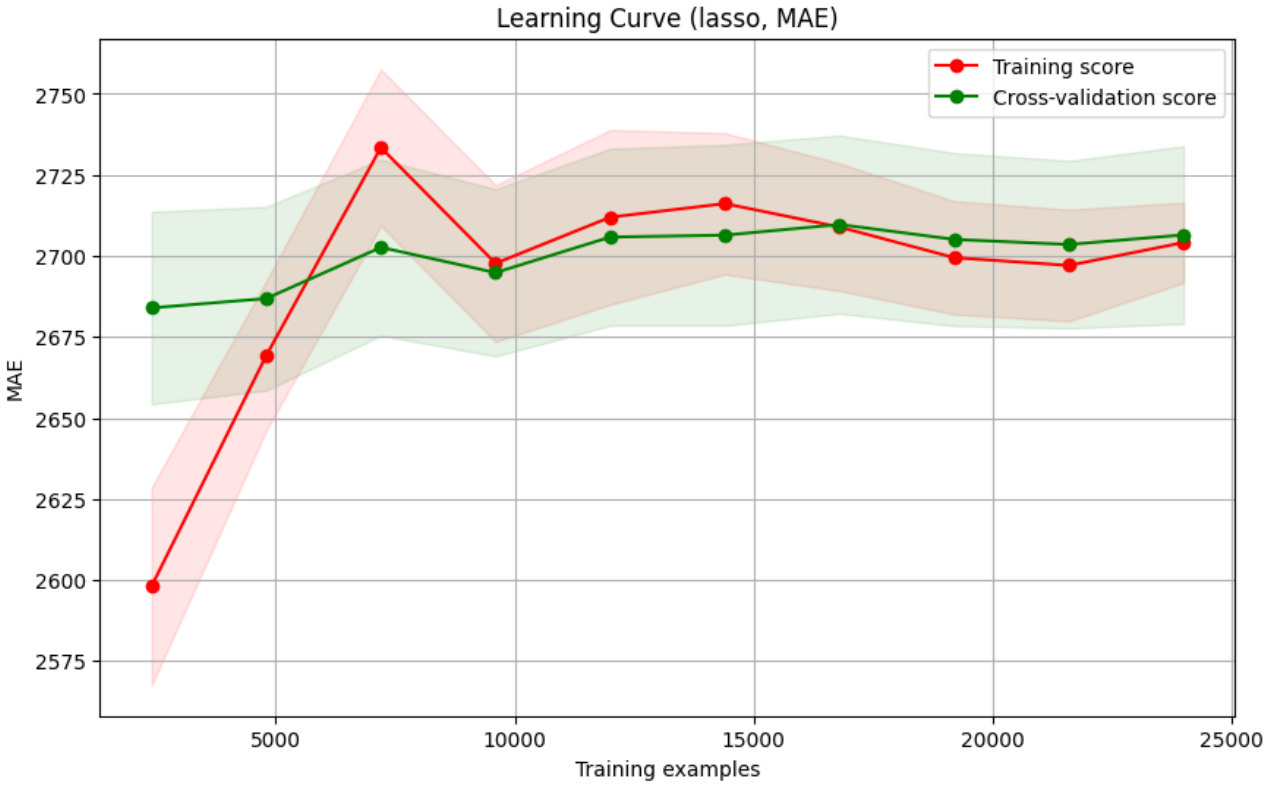
I grafici per il confronto delle performance e le curve di apprendimento sono i seguenti:

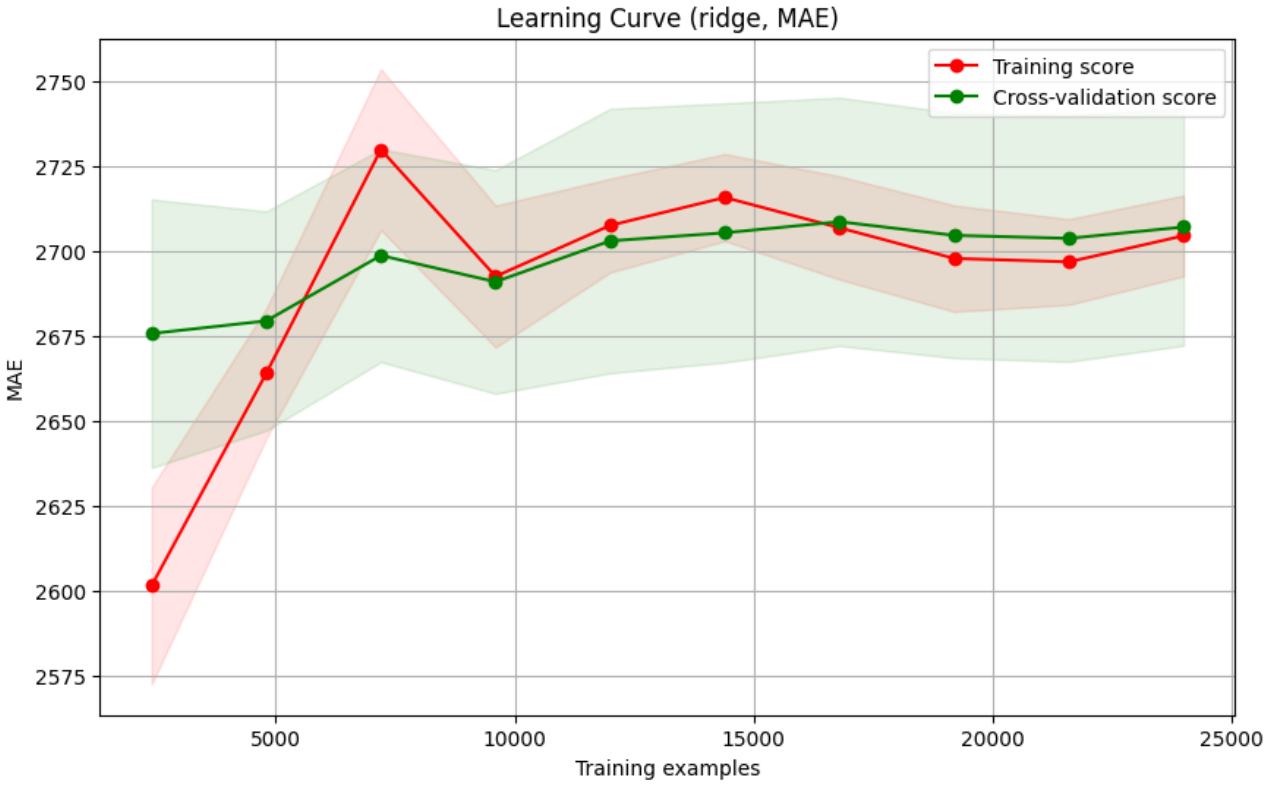


Guardando solo l'errore complessivo, i modelli lineari mostrano prestazioni simili tra loro, ma risultano notevolmente inferiori rispetto ai modelli precedentemente analizzati, incluso l’albero decisionale.

Andiamo ad analizzare le curve di apprendimento e i valori di varianza degli errori di test e train:







Model	Train Score Variance	Test Score Variance	Train Score Std dev	Test Score Std dev
linear_regression	361.986497	2159.071678	19.025943	46.465812
lasso	152.723450	751.573863	12.358133	27.414848
ridge	141.815836	1220.952451	11.908645	34.942130

Le curve di train e di test sono abbastanza vicine, anche se c'è un leggero trend di aumento dell'errore all'aumentare del numero di esempi.

Inoltre, sia le metriche di errore sia le curve di apprendimento sono risultati simili tra questi modelli, senza particolari differenze basate sulla presenza o assenza e sul tipo di regolarizzazione.

Capitolo 3) Apprendimento bayesiano

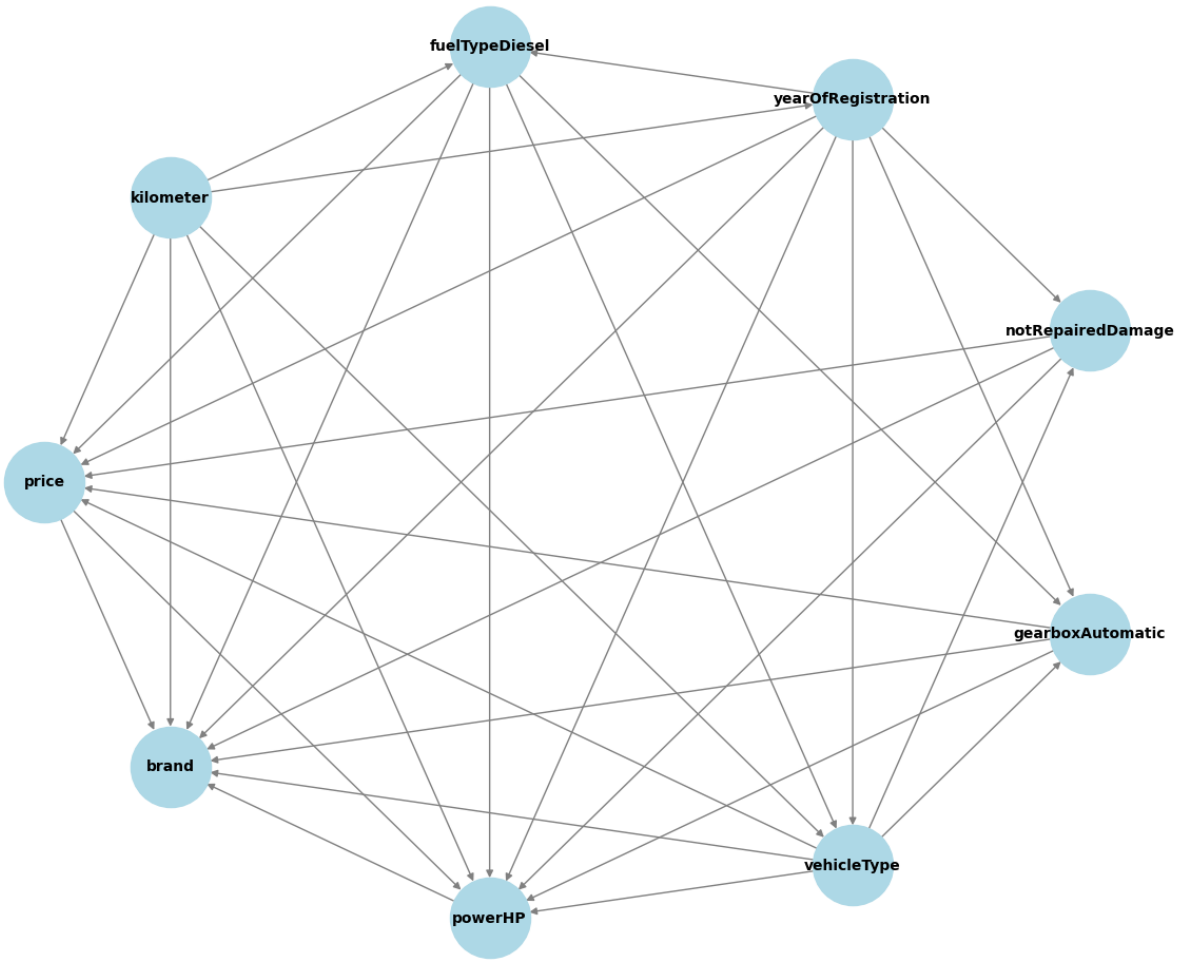
Per questo argomento sono state utilizzate la libreria di Python *pgmpy* per la gestione della rete bayesiana e matplotlib e networkx per visualizzare grafici.

L'apprendimento bayesiano si va a creare una rete bayesiana, che in questo caso può avere diversi obiettivi, ad esempio il poter essere usata come un normale classificatore del prezzo o anche saper fare previsioni in mancanza di valori per una o più variabili. Per questo approccio i valori continui del dataset sono stati resi discreti attraverso la discretizzazione, come descritto nel pre-processing.

Struttura della rete bayesiana

Per poter creare una rete bayesiana, bisogna prima definirne la struttura, ovvero le variabili e le relazioni di dipendenza tra esse. La struttura può essere definita manualmente o appresa dal dataset, come in questo caso. Ciò viene fatto attraverso una tecnica denominata HillClimbSearch, un metodo di ricerca locale che modifica iterativamente la struttura della rete fino ad averne una che rappresenta le dipendenze tra le variabili in modo appropriato.

Struttura della rete bayesiana appresa



Si possono notare dal grafico che sono state rilevate relazioni appropriate tra le variabili, ad

esempio la dipendenza del prezzo da quasi tutte le altre variabili o l'associazione tra anno di immatricolazione e chilometri percorsi.

Addestramento della rete bayesiana

La rete viene addestrata basandosi sulla struttura definita precedentemente, creando le CPD o Conditional Probability Distributions, in cui si definisce la distribuzione di probabilità di ogni variabile date le variabili associate.

```
CPD per variabile price
+--------------------+-----+--------------------------+
| fuelTypeDiesel     | ... | fuelTypeDiesel(True)     |
+--------------------+-----+--------------------------+
| gearboxAutomatic   | ... | gearboxAutomatic(True)   |
+--------------------+-----+--------------------------+
| kilometer          | ... | kilometer(150000)        |
+--------------------+-----+--------------------------+
| notRepairedDamage  | ... | notRepairedDamage(True)  |
+--------------------+-----+--------------------------+
| vehicleType        | ... | vehicleType(stationwagon) |
+--------------------+-----+--------------------------+
| yearOfRegistration | ... | yearOfRegistration(9)    |
+--------------------+-----+--------------------------+
| price(0)           | ... | 0.005070993914807302     |
+--------------------+-----+--------------------------+
| price(1)           | ... | 0.005070993914807302     |
+--------------------+-----+--------------------------+
| price(2)           | ... | 0.005070993914807302     |
+--------------------+-----+--------------------------+
| price(3)           | ... | 0.005070993914807302     |
+--------------------+-----+--------------------------+
| price(4)           | ... | 0.005070993914807302     |
+--------------------+-----+--------------------------+
| price(5)           | ... | 0.005070993914807302     |
+--------------------+-----+--------------------------+
| price(6)           | ... | 0.005070993914807302     |
+--------------------+-----+--------------------------+
| price(7)           | ... | 0.6379310344827587       |
+--------------------+-----+--------------------------+
| price(8)           | ... | 0.005070993914807302     |
+--------------------+-----+--------------------------+
| price(9)           | ... | 0.32150101419878296      |
+--------------------+-----+--------------------------+
```

```
CPD per variabile yearOfRegistration
+----------------------+-----+----------------------+
| kilometer            | ... | kilometer(150000)    |
+----------------------+-----+----------------------+
| yearOfRegistration(0) | ... | 0.13562220718961102  |
+----------------------+-----+----------------------+
| yearOfRegistration(1) | ... | 0.11791019179282947  |
+----------------------+-----+----------------------+
| yearOfRegistration(2) | ... | 0.15972850786512577  |
+----------------------+-----+----------------------+
| yearOfRegistration(3) | ... | 0.1477140638547215   |
+----------------------+-----+----------------------+
| yearOfRegistration(4) | ... | 0.0735527406461694   |
+----------------------+-----+----------------------+
| yearOfRegistration(5) | ... | 0.1381303772020923   |
+----------------------+-----+----------------------+
| yearOfRegistration(6) | ... | 0.11924168945377633  |
+----------------------+-----+----------------------+
| yearOfRegistration(7) | ... | 0.0688305687090904   |
+----------------------+-----+----------------------+
| yearOfRegistration(8) | ... | 0.031068675743399683 |
+----------------------+-----+----------------------+
| yearOfRegistration(9) | ... | 0.008200977543184353 |
+----------------------+-----+----------------------+
```

Si può vedere come una macchina con 150 000km ha più probabilità di essere stata immatricolata non di recente, ovvero con un valore di `yearOfRegistration` più basso.

Utilizzo della rete bayesiana

La rete bayesiana può essere usata come un normale classificatore, facendo prevedere la variabile *price* date tutte le altre variabili, oppure prevedere il valore di una o più variabili (*brand* e *vehicleType* nell'esempio) in mancanza di valori per alcune delle altre (*kilometer*).

```
Evidence: {'vehicleType': 'cabrio', 'yearOfRegistration': 7,
'gearboxAutomatic': False, 'powerHP': 4, 'kilometer': 100000,
'fuelTypeDiesel': False, 'brand': 'opel', 'notRepairedDamage': False}

Predicted variables: {'price': 6}


Evidence: {'price': 7, 'yearOfRegistration': 7, 'gearboxAutomatic': True,
'powerHP': 4, 'fuelTypeDiesel': False, 'notRepairedDamage': True}

Predicted variables: {'brand': 'audi', 'vehicleType': 'sedan'}
```

Capitolo 4) Base di Conoscenza e Prolog

L'obiettivo della base di conoscenza è stimare il prezzo di una macchina a partire dalle sue caratteristiche e confrontarlo con un eventuale prezzo in un annuncio di vendita. Le caratteristiche di interesse della macchina sono le stesse del dataset dopo il pre-processing ma prima della fase di one-hot-encoding, dunque vehicleType, yearOfRegistration, gearboxAutomatic, powerHP, kilometer, fuelTypeDiesel, brand, notRepairedDamage. Il dataset così trattato è stato salvato in **preprocessed_data.csv.**

La base di conoscenza è stata sviluppata in Prolog, mentre il programma vi si interfaccia attraverso la libreria *pyswip* di Python. L'implementazione di Prolog usata è SWI-Prolog. Sono stati definiti diversi assiomi, divisi in fatti e regole.

Assiomi della base di conoscenza

- **predirre_prezzo/2**

```
predirre_prezzo(Features, Prezzo) :-
    calcolare_incremento(Features, 0, Prezzo).
```

  A partire dalla di caratteristiche (**Features**) si predice il **Prezzo** della macchina utilizzando **calcolare_incremento**, ponendo come accumulatore del prezzo parziale 0.

- **calcolare_incremento**
  L'obiettivo è calcolare l'incremento del prezzo causato dalla prima feature nella lista.

```
calcolare_incremento([Feature|AltreFeature], PrezzoParziale, Prezzo) :-
    length(AltreFeature, Indice),
    incremento_feature(Indice, Feature, Incremento),
    NuovoPrezzoParziale is PrezzoParziale + Incremento,
    calcolare_incremento(AltreFeature, NuovoPrezzoParziale, Prezzo).


calcolare_incremento([], PrezzoParziale, PrezzoParziale).
```

  È una regola ricorsiva, che richiama sé stessa rimuovendo la prima feature nella lista fino ad avere una lista vuota (passo base). Viene usato un accumulatore per mantenere il prezzo parziale, che nel passo base, non essendoci feature da analizzare, rappresenta anche il prezzo complessivo.
  Il passo ricorsivo invece consiste nell'ottenere la lunghezza della lista, escluso il primo elemento, che corrisponde l'indice della feature per cui calcolare l'incremento del prezzo tramite **incremento_feature.** L'incremento viene sommato al **PrezzoParziale**. Infine, vi è la ricorsione con cui si calcola l'incremento delle **AltreFeature**.

- **incremento_feature**
  Si intende calcolare la variazione del prezzo causata da una caratteristica della macchina. Il valore della variazione è fissa in caso la feature sia booleana o categorica, mentre è il risultato di una funzione se la feature è numerica.
  L'indice, con cui si identifica il tipo di feature, corrisponde al numero di feature rimanenti nella lista (**AltreFeatures**) quando viene analizzata con la regola precedente, dunque essendoci 8 feature, la prima feature avrà indice 7, la seconda indice 6...
  Ecco tutte le regole definite di questo tipo:

```
% vehicleType, yearOfRegistration, gearboxAutomatic, powerHP, kilometer,
fuelTypeDiesel, brand, notRepairedDamage

incremento_feature(7, bus, 5000).
incremento_feature(7, cabrio, 4000).
incremento_feature(7, citycar, 3500).
incremento_feature(7, coupe, 4500).
incremento_feature(7, sedan, 7000).
incremento_feature(7, stationwagon, 6000).

% yearOfRegistration
incremento_feature(6, Anno, Incremento) :-
    Incremento is 3000 + (2019 - Anno) * -200.

% gearboxAutomatic
incremento_feature(5, false, 0).
incremento_feature(5, true, 2000).

% powerHP
incremento_feature(4, HP, Incremento) :-
    Incremento is HP * 20.

% kilometer
incremento_feature(3, KM, Incremento) :-
    Incremento is 4000 - (KM * 0.05).

% fuelTypeDiesel
incremento_feature(2, false, 0).
incremento_feature(2, true, 1500).

% brand
incremento_feature(1, audi, 10000).
incremento_feature(1, bmw, 9000).
incremento_feature(1, ford, 5000).
incremento_feature(1, mercedes_benz, 10000).
incremento_feature(1, fiat, 6000).
incremento_feature(1, opel, 5000).
incremento_feature(1, peugeot, 7000).
incremento_feature(1, renault, 7500).
incremento_feature(1, seat, 6000).
```

```
incremento_feature(1, skoda, 4000).
incremento_feature(1, volkswagen, 6000).
incremento_feature(1, mazda, 8000).

% notRepairedDamage
incremento_feature(0, false, 1000).
incremento_feature(0, true, -2500).
```

- **prezzo_conveniente**
  Dato un prezzo proposto in un eventuale annuncio, questa regola è vera se e solo se il prezzo predetto con **predirre_prezzo** è più basso.

```
prezzo_conveniente(Features, PrezzoProposto) :-
    predirre_prezzo(Features, PrezzoPredetto),
    PrezzoProposto < PrezzoPredetto.
```

- **macchina_piu_costosa**
  Data una lista di macchine, ovvero una lista di liste di feature, la si scandisce per trovare la macchina con il prezzo predetto più alto.

```
macchina_piu_costosa([Features|AltreMacchine], Macchina, Prezzo) :-
    predirre_prezzo(Features, PrezzoCorrente),
    macchina_piu_costosa(AltreMacchine, MaxMacchina, MaxPrezzo),
    (PrezzoCorrente > MaxPrezzo ->
     Macchina = Features, Prezzo = PrezzoCorrente
     ;
     Macchina = MaxMacchina, Prezzo = MaxPrezzo).

macchina_piu_costosa([], [], 0).
```

  Anche questa è una regola ricorsiva, in cui si analizza separatamente la prima macchina della lista tramite **predirre_prezzo**, si ottiene la macchina più costosa tra le **AltreMacchine** e le si confronta salvando quella più costosa.
  Il passo base consiste nell'avere una lista di macchine vuota, che porta quindi a non avere alcuna macchina più costosa e prezzo nullo.

La base di conoscenza può essere interrogata tramite Python e la libreria *pyswip*. La si può interrogare con una qualsiasi delle righe di preprocessed_data.csv, dopo aver scritto i valori booleani True e False tutti in minuscolo e rimosso gli apici per le stringhe. Ciò è necessario per renderlo compatibile con la sintassi di Prolog.
Con la query
`predirre_prezzo([coupe,2011,false,190,125000,true,audi,true], Prezzo).` la base di conoscenza restituisce `[{'Prezzo': 16450.0}]`.

Se con le stesse caratteristiche si interroga la regola **prezzo_conveniente**, specificando come Prezzo 10000, verrà restituito `True.`

Per testare la regola **macchina_piu_costosa**, si può provare a caricare tutto il dataset da **preprocessed_data.csv**, ad eccezione della colonna price, e passarlo nella query. Il risultato che si ottiene è: `[{'Macchina': ['sedan', 2015, 'true', 455, 5000, 'false', 'mercedes_benz', 'false'], 'Prezzo': 35050.0}].`

Da albero decisionale a base di conoscenza

Sfruttando l'albero decisionale creato precedentemente, è possibile creare una rappresentazione di esso in Prolog, in modo da avere un'alternativa alla predizione del prezzo di una macchina proposta al capitolo precedente.
Per poter portare l'albero decisionale realizzato con Scikit-learn in Prolog, bisogna innanzitutto capire come è implementato.
Ad ogni nodo intermedio si controlla una condizione che è sempre nel formato `feature <= threshold`. Se questa condizione è vera, si percorre l'albero andando nel figlio sinistro, altrimenti nel figlio destro. Se invece il nodo è una foglia allora avrà una stima puntuale denominata `value`.
Scikit-learn usa diversi array per rappresentare le varie caratteristiche dei nodi dell'albero, in cui l'i-esimo elemento rappresenta una caratteristica dell'i-esimo nodo. Questi array sono i seguenti:

- **feature**, in cui viene indicata la feature per cui si controlla la condizione. Assume come valore TREE_UNDEFINED se il nodo è una foglia.
- **threshold**, in cui è presente il valore soglia per cui decidere verso quale figlio continuare;
- **children_left** e **children_right,** in cui vengono indicati gli indici rispettivamente dei nodi sinistro e destro;
- **value**, stima puntuale della predizione, solo se il nodo è una foglia.

Per poter portare nodi intermedi e foglie in Prolog si definiscono due tipi di fatti:

```
leaf(NodeID, Predizione).
```

Per rappresentare una foglia, con NodeID come indice del nodo negli array e Predizione come stima.

```
node(NodeID, FeatureIndex, Threshold, LeftChild, RightChild).
```

Per rappresentare un nodo intermedio, con NodeID indice del nodo negli array, FeatureIndex indice della feature su cui controllare la condizione, Threshold valore soglia della condizione, LeftChild e RigthChild indici dei nodi figli.

Per avere una interfaccia simile a quella presentata precedentemente si è definito una regola che permette di prevedere il prezzo usando l'albero decisionale così rappresentato.

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

percorri_albero è la regola ricorsiva che permette effettivamente di traversare l'albero fino al raggiungimento di una foglia. Implementa la stessa logica utilizzata nell'implementazione dell'albero decisionale in Scikit-learn.

Si parte dal nodo radice, ovvero con indice o NodeID 0. Se è un nodo intermedio si andrà nel passo ricorsivo che ha il compito di trovare gli elementi che compongono la condizione di split, ovvero il valore della feature discriminante, il valore soglia e gli indici del figlio sinistro e destro. Una volta fatto ciò in base al risultato della condizione, vi è la ricorsione di **percorri_albero** sul figlio sinistro o destro.

```
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)
    ).
```

Il valore della feature discriminante viene preso usando l'indice feature specificato nel nodo e la lista dei valori delle feature.

Quando si arriva ad un nodo foglia, si avrà il valore di Predizione che corrisponde alla stima presente nella foglia.

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

A partire dai 5 array di Scikit-learn si vanno a costruire i fatti di tipo node e leaf tramite Python, fatti che poi vengono concatenati alle definizioni delle altre regole nel file **decision_tree.pl.**

Il formato delle feature è uguale a quello accettato nell'albero decisionale in Scikit-learn, in cui ogni feature è rappresentata numericamente, dunque per poter passare una riga dal dataset si possono convertire tutti i valori in float.

Provando a prevedere il prezzo per la stessa riga del dataset usando sia l'albero decisionale in Prolog sia in Scikit-learn, vengono restituiti i seguenti risultati:

```
Features: [1998.0, 1.0, 136.0, 150000.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]

Prolog DT prediction: [{'Prediction': 1600.0}]

Scikit-learn DT prediction: [1600.]
```

Sono state concatenate anche le altre regole presentate nel paragrafo precedente, in modo da poter confrontare il prezzo, stimato in questo caso con l'albero decisionale, con un eventuale prezzo in un annuncio di vendita e recuperare la macchina più costosa a partire da una lista di macchine.

Capitolo 5) Conclusioni

Complessivamente in questo progetto sono stati sviluppati modelli di apprendimento supervisionato per prevedere il prezzo di vendita di macchine usate basandoci su un dataset di annunci di eBay.

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 macchina, usando un albero decisionale o un calcolo "manuale" basandosi dai valori delle caratteristiche, e per confrontare il costo di più macchine.

È stato esplorato anche l'apprendimento bayesiano, creando una rete bayesiana che può prevedere variabili mancanti e anche funzionare come un classificatore di prezzo. Questa rete può essere utilizzata in situazioni dove non è stato possibile osservare la totalità delle variabili.

Possibili Sviluppi

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

Inoltre sarebbe possibile creare un'interfaccia grafica per poter usare più facilmente i modelli e la base di conoscenza creata.

Riferimenti bibliografici

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

Apprendimento bayesiano: D. Poole, A. Mackworth: Artificial Intelligence: Foundations of Computational Agents. 3nd ed. Cambridge University Press [Ch.10]

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