

Car price estimator

Progetto del CdS “Ingegneria della Conoscenza”

A.A. 2024/2025

Gruppo di lavoro

- Angelo Polito, 775433, a.polito13@studenti.uniba.it

Repository GitHub

<https://github.com/angpolito/car-price-estimator>

Introduzione

"Car Price Estimator" è un software progettato per valutare il valore di mercato delle automobili. L'applicazione accompagna l'utente attraverso un processo guidato di inserimento delle caratteristiche del proprio veicolo, elaborando successivamente una stima accurata del suo valore commerciale.

Il sistema si distingue per il suo approccio innovativo che combina diverse tecniche di machine learning: utilizza sia metodi di apprendimento supervisionato che non supervisionato per generare le sue valutazioni. La peculiarità del software risiede nella sua capacità di fornire anche un'analisi probabilistica complementare: attraverso tecniche di apprendimento probabilistico, il sistema è in grado di indicare con quale probabilità il prezzo stimato si collochi all'interno di specifici intervalli di valore predefiniti.

Questa doppia modalità di stima offre all'utente una visione più completa e sfaccettata del potenziale valore del proprio veicolo sul mercato attuale.

Sommario

Il sistema sviluppato integra diversi approcci di machine learning per fornire stime accurate dei prezzi delle auto usate. Il cuore del Knowledge-Based System è costituito da tre moduli complementari:

1. Un modello di apprendimento supervisionato basato su Random Forest, ottimizzato attraverso k-fold cross validation e Grid Search per la ricerca degli iperparametri ottimali. Questo permette di ottenere predizioni puntuali a partire dalle caratteristiche del veicolo.
2. Un sistema di clustering non supervisionato che identifica gruppi omogenei di veicoli con caratteristiche e valutazioni simili, fornendo un'ulteriore prospettiva sulla stima del valore.
3. Un classificatore probabilistico addestrato con Stochastic Gradient Descent che associa a ogni predizione una distribuzione di probabilità su intervalli di prezzo predefiniti, quantificando così l'incertezza della stima.

L'integrazione di questi tre approcci complementari - supervisionato, non supervisionato e probabilistico - consente di ottenere stime robuste e di fornire all'utente una visione completa del possibile valore di mercato del proprio veicolo.

Elenco argomenti di interesse

- Apprendimento supervisionato (Random Forest, k-fold , Grid Search)
- Apprendimento non supervisionato (Clustering)
- Apprendimento probabilistico (Modello SGD, Classificatore Probabilistico)

Decisioni di sviluppo preliminari

Ambiente di sviluppo, linguaggio e librerie adottate

Il software è stato sviluppato in PyCharm, interamente in Python 3.13. Le librerie utilizzate sono state le seguenti:

- **Pandas:** gestisce e analizza dati strutturati (come tabelle). Permette manipolazioni avanzate di dati con DataFrame e Series.
- **Numpy:** fornisce array multidimensionali e operazioni efficienti su di essi, essenziale per calcoli numerici ad alte prestazioni.
- **Scikit-Learn (Sklearn):** implementa algoritmi di machine learning e strumenti per analisi di dati (classificazione, regressione, clustering).
- **Matplotlib:** crea grafici e visualizzazioni personalizzabili per dati (2D e 3D).
- **Tqdm:** mostra barre di progresso per cicli e processi iterativi, migliorando la comprensione del tempo rimanente.
- **Tkinter:** crea interfacce grafiche (GUI) per applicazioni desktop Python.

Dataset

Il dataset impiegato per questo progetto è stato acquisito da [Kaggle](#), chiamato "used_cars.csv". Comprende 4009 istanze e include 12 features: brand (nome della casa automobilistica), model (modello dell'auto), model_year (anno di costruzione del veicolo), milage (chilometraggio espresso in miglia), fuel_type (tipo di carburante), engine (tipo di motore), transmission (tipo di trasmissione), ext_col (colore degli esterni), int_col (colore degli interni), accident (storico degli incidenti subiti dal veicolo), clean_title (disponibilità di un titolo senza vincoli legali), price (prezzo).

Predisposizione del dataset

Il dataset è stato predisposto effettuando delle operazioni di conversione di unità di misura e "pulizia" di valori indesiderati.

Sono stati ripuliti i campi da simboli indesiderati come apici, simbolo del dollaro, virgole e trattini. Per avere dati più accurati si è deciso di eliminare i duplicati e i campi con i valori nulli dal dataset. È stata effettuata inoltre la conversione dei valori della feature 'milage' da miglia a chilometri, per adeguarsi all'unità ufficiale del SI per la lunghezza.

Descrizione del progetto

Predizione del prezzo

Il processo di predizione del prezzo è stato ottimizzato attraverso una strategia multifase che combina diverse tecniche di machine learning. Inizialmente, tutti i valori del dataset sono stati normalizzati in un intervallo compreso tra 0 e 1, garantendo così una maggiore uniformità dei dati.

L'architettura del sistema prevede una sequenza di elaborazioni: si parte con tecniche di apprendimento non supervisionato, applicando algoritmi di clustering per identificare pattern e raggruppamenti significativi nei dati. Successivamente, questi risultati vengono integrati nell'fase di apprendimento supervisionato per la predizione del prezzo.

Il sistema è stato ulteriormente arricchito con un modulo di apprendimento probabilistico, che fornisce un'analisi complementare: questo componente valuta la probabilità che il prezzo di un veicolo rientri in specifiche fasce di valore, offrendo così una prospettiva più completa sulla valutazione.

Modelli di apprendimento

L'implementazione del sistema si basa su una metodologia rigorosa di apprendimento e validazione. Il dataset è stato strategicamente suddiviso in due porzioni: l'**80%** dedicato al **training-set** e il **20%** riservato al **test-set**. Questa partizione, applicata ai modelli **Random Forest Regressor**, è fondamentale per prevenire il fenomeno dell'overfitting e permette una valutazione oggettiva delle prestazioni su dati non utilizzati durante la fase di addestramento.

Per quanto riguarda l'apprendimento probabilistico, è stato adottato il modello **SGD (Stochastic Gradient Descent) Classifier**, selezionato per i suoi superiori livelli di accuratezza e affidabilità. Il processo di ottimizzazione è stato ulteriormente raffinato attraverso l'implementazione della tecnica **K-fold**, che prevede la suddivisione del training set in k sottoinsiemi (fold) per un addestramento e una valutazione più robusti.

Entrambi i modelli sono stati sottoposti a **Grid Search**, una tecnica di ottimizzazione degli iperparametri che permette di identificare la combinazione ottimale di variabili per massimizzare le prestazioni. Questo approccio metodico assicura che sia il Random Forest Regressor che l'SGD Classifier operino con la configurazione più efficace per i rispettivi compiti di predizione.

Valutazione della qualità del modello

La valutazione delle prestazioni del sistema si è basata su molteplici metriche di qualità, specifiche per ciascun modello implementato.

Per il **Random Forest Regressor**, sono stati calcolati tre indicatori principali:

- Il **Mean Absolute Error (MAE)** di 10272.66, che quantifica la deviazione media assoluta tra i prezzi predetti e quelli reali nel test-set;

- Il **Mean Square Error (MSE)** di 328509043.45, che penalizza maggiormente gli errori di grande entità;
- L'**R2 score** di 0.78 (78%), che indica che il modello è in grado di spiegare il 78% della variabilità nei dati, evidenziando un buon livello di accuratezza predittiva.

Per quanto riguarda il modello probabilistico basato su **SGD Classifier**, la valutazione si è concentrata sulla metrica dell'**accuracy**, che ha raggiunto il 79.13%. Questo risultato indica che il modello è in grado di classificare correttamente la fascia di prezzo in quasi quattro casi su cinque, dimostrando una notevole affidabilità nella categorizzazione dei prezzi.

```
-----  
MAE RandomForest normale sul test set: 10272.66  
MSE RandomForest normale sul test set: 328509043.45  
R2 Score RandomForest normale sul test set: 0.78  
-----  
Accuracy SGD: 79.13%
```

Accuracy della predizione

Per mostrare l'accuratezza della predizione per ogni singolo feature abbiamo inserito la stampa di grafici che dimostrano tale accuratezza (in blu i valori reali delle features e in rosso la loro predizione), le istruzioni che creano tale grafico sono commentate, in quanto non più utili una volta che si crea il grafico, di seguito qualche esempio rispettivamente delle seguenti features (sqft_living , floors):

***grafico

Scelte progettuali

Scelta di “indexState = 497”

Per massimizzare le prestazioni del modello Random Forest, è stata condotta un'analisi sistematica attraverso un processo iterativo. Il sistema ha esplorato una sequenza di stati, variando l'indexState da 0 a 3000, alla ricerca della configurazione ottimale. Questa ricerca metodica ha portato all'identificazione del valore di indexState che ha prodotto i risultati più accurati, raggiungendo un coefficiente di determinazione (R^2) di 0.7831.

Questo R^2 di 0.7831 indica che il modello ottimizzato è in grado di spiegare il 78.31% della variabilità nei dati, rappresentando un significativo miglioramento rispetto alle configurazioni precedenti e confermando l'efficacia della strategia di ottimizzazione implementata.

Scelta dei range destinati all'apprendimento probabilistico

Le fasce di prezzo utilizzate nel modello sono state definite attraverso un'analisi esplorativa del dataset `used_cars.csv`. La categorizzazione è stata stabilita identificando i cluster di prezzi più frequenti nel dataset, permettendo così una segmentazione naturale e rappresentativa del mercato automobilistico analizzato.

Utilizzo di `get_dummies` e del `LabelEncoder`

Il sistema implementa due diverse strategie di codifica per trasformare le variabili categoriche in un formato numerico elaborabile dai modelli di machine learning:

Per le caratteristiche più complesse ('brand', 'model', 'fuel_type', 'engine', 'transmission', 'ext_col', 'int_col'), viene applicata la tecnica `get_dummies`, che genera una rappresentazione one-hot encoding. Questo processo crea nuove colonne binarie per ogni valore unico presente nelle feature originali, dove:

- Ogni colonna rappresenta una specifica categoria;
- I valori sono codificati come 1 (presenza) o 0 (assenza) della caratteristica.

Per le variabili binarie ('accident' e 'clean_title'), viene invece utilizzato il `LabelEncoder`, che:

- Converte direttamente le stringhe in valori numerici;
- Permette una codifica più efficiente per variabili con due stati;
- Mantiene la natura dicotomica delle variabili senza aumentare la dimensionalità del dataset.

Questa strategia dual-track di preprocessing assicura che tutte le variabili categoriche siano convertite in un formato numerico ottimale per l'elaborazione algoritmica, preservando al contempo l'informazione originale in modo efficiente.

Normalizzazione su tutte le features tranne 'price'

La feature 'price' riceve un trattamento differenziato nel processo di preprocessing dei dati. A differenza delle altre variabili, questa non viene sottoposta a normalizzazione, mantenendo i suoi valori numerici originali interi. Questa scelta metodologica è motivata da due ragioni principali:

- È la variabile target del modello, quindi viene separata dal set di feature utilizzate per l'addestramento
- I valori interi originali sono necessari per effettuare predizioni accurate e interpretabili nel contesto reale dei prezzi di mercato

Questa strategia assicura che le predizioni del modello rimangano direttamente correlate ai valori monetari effettivi, senza necessità di successive trasformazioni inverse.

Apprendimento non supervisionato

DBSCAN è un algoritmo di clustering basato sulla densità che suddivide i dati in gruppi identificando le aree ad alta densità come nuclei dei cluster e trattando i punti meno densi come rumore o outlier.

Nel processo, viene creato un oggetto denominato clusters utilizzando l'algoritmo DBSCAN con due parametri chiave:

- **eps**: il raggio di vicinanza entro cui i punti vengono considerati vicini;
- **min_samples**: il numero minimo di punti richiesti all'interno del raggio eps per formare un cluster.

L'oggetto clusters viene addestrato sui dati contenuti in prices_x. Dopo l'addestramento, l'algoritmo restituisce le etichette di clustering assegnate a ciascun dato di prices_x. Queste etichette vengono poi aggiunte come una nuova colonna chiamata noise sia al dataframe prices_x che al dataframe prices_y.

Successivamente, vengono selezionate solo le righe in cui il valore della colonna noise è maggiore di -1, escludendo così i punti considerati rumore e mantenendo solo quelli appartenenti a un cluster. Infine, la colonna noise viene rimossa da prices_x utilizzando il metodo drop.

DBSCAN è considerato un algoritmo di clustering "soft" in quanto assegna i punti a un cluster o li identifica come rumore, senza costringerli in una struttura predefinita.

L'uso di DBSCAN è stato scelto per il clustering perché offre diversi vantaggi, tra cui:

- la capacità di individuare cluster di forma arbitraria;
- la robustezza rispetto agli outlier, che vengono distinti dai veri gruppi di dati;
- l'assenza della necessità di specificare il numero di cluster in anticipo, a differenza di algoritmi come k-means.

Train/Test split

La decisione di suddividere il dataset in training set e test set utilizzando il parametro `test_size = 0.2` è stata presa dopo diversi tentativi, tenendo conto che, idealmente, una suddivisione bilanciata prevede di riservare circa due terzi dei dati per l'addestramento e un terzo per il test. Tuttavia, in questo caso, si è optato per una proporzione leggermente diversa per ottimizzare le prestazioni del modello.

SGD_model

La scelta di utilizzare **SGD Classifier** è stata fatta dopo aver sperimentato un apprendimento probabilistico con il Gaussian model e il Multinomial model, i cui risultati si sono rivelati poco affidabili. In particolare, il Gaussian model, pur essendo un buon classificatore, non si è dimostrato un efficace stimatore di probabilità, mentre il Multinomial model ha restituito un'accuracy inferiore. Di conseguenza, si è optato per SGD Classifier, che ha garantito prestazioni più soddisfacenti nel contesto analizzato.

Architettura del modello

Lo SGD Classifier si basa sull'algoritmo di ottimizzazione **Stochastic Gradient Descent (SGD)**, che regola iterativamente i parametri del modello (pesi e bias) per minimizzare una funzione di perdita. Durante l'addestramento, il modello utilizza mini-batch di dati per aggiornare i pesi, cercando di ridurre progressivamente il costo (Loss) e trovare i parametri ottimali.

L'obiettivo dell'SGD è raggiungere il minimo globale della funzione di perdita, partendo da un peso iniziale scelto casualmente (Initial Weight). Il processo di aggiornamento dei pesi è guidato dal Learning Rate, un parametro cruciale che determina l'ampiezza dei passi verso il minimo:

- quando il Learning Rate è troppo alto, il modello potrebbe oscillare e non convergere mai, saltando il punto ottimale;
- quando è troppo basso, invece, il modello potrebbe aggiornarsi troppo lentamente, impiegando un tempo eccessivo per raggiungere il minimo o rimanendo bloccato in un minimo locale.

Per la base del modello, SGDClassifier può applicare sia Support Vector Machine (SVM) che Logistic Regression. In questo caso, è stata scelta la Logistic Regression, poiché utilizza una funzione di decisione basata su probabilità, risultando più adatta all'analisi dei dati a disposizione.

Funzione di decisione

Il classificatore calcola una funzione di decisione per ciascuna istanza del dataset, combinando linearmente i valori delle feature con i rispettivi pesi.

Matematicamente, in un problema di classificazione multiclasse, questa funzione può essere espressa come:

$$f_i(x) = w_1x_1 + w_2x_2 + \dots + w_nx_n + b_i$$

dove:

- w_1, w_2, \dots, w_n sono i pesi appresi dal modello,
- x_1, x_2, \dots, x_n rappresentano i valori delle feature dell'istanza,
- b_i è il bias associato alla classe i .

Questa funzione permette di determinare il punteggio di ogni classe per un dato sample, e la classe con il valore più alto viene assegnata come predizione finale.

Calcolo delle probabilità di classe

Le probabilità di classe vengono calcolate trasformando la funzione di decisione in valori probabilistici mediante la funzione di attivazione Softmax, particolarmente adatta per problemi di classificazione multiclasse.

La funzione Softmax assegna a ciascuna classe una probabilità compresa tra 0 e 1, garantendo che la somma delle probabilità su tutte le classi sia pari a 1. È definita matematicamente come:

$$P(y = i|x) = \frac{\exp(score_i)}{\sum_{j \in classes} \exp(score_j)}$$

Dove:

- $P(y = i|x)$ è la probabilità che l'istanza appartenga alla classe i dato l'input x ,
- $\exp(x)$ è la funzione esponenziale,
- $score_i$ rappresenta il punteggio assegnato alla classe i dalla funzione di decisione,
- $\sum_{j \in classes} \exp(score_j)$ è la somma dei valori esponenziali di tutti i punteggi, utilizzata per normalizzare le probabilità.

Grazie alla funzione Softmax, il modello è in grado di assegnare una probabilità a ciascuna classe e selezionare quella con il valore più alto come predizione finale.

Uso di `predict_proba()`

Quando si utilizza il metodo `predict_proba()` su un singolo sample o su un insieme di sample, il classificatore calcola le probabilità di appartenenza a ciascuna classe basandosi sui valori delle feature e sui parametri appresi dal modello (pesi).

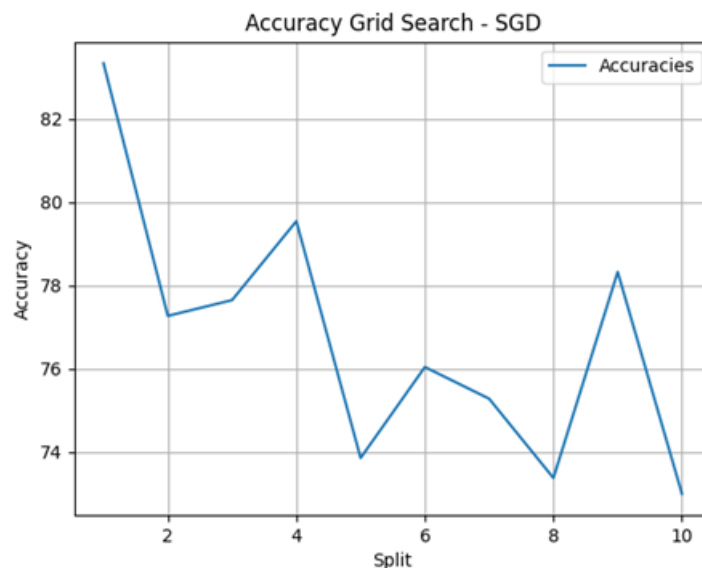
Il metodo restituisce un array in cui:

- **ogni riga** corrisponde a un'istanza del dataset;
- **ogni colonna** rappresenta la probabilità che l'istanza appartenga a una specifica classe.

Nel nostro caso, essendo presenti **6 classi**, l'array risultante avrà **6 colonne**, ognuna contenente la probabilità di appartenenza alla rispettiva classe.

Per quanto riguarda la configurazione dell'`SGDClassifier`, il valore di `alpha` è stato scelto tramite Grid Search, individuando il parametro ottimale tra diversi valori testati. Il miglior risultato è stato ottenuto con **alpha = 0.0001**, corrispondente allo split numero 1, dal valore di accuracy pari a 83.33%.

Il grafico riportato di seguito evidenzia il punto in cui si è ottenuta la massima accuratezza in funzione dei diversi valori testati. Successivamente, vengono mostrati i parametri selezionati, gli split considerati e le relative accuratezze stampate a terminale.



```

Fitting 10 folds for each of 10 candidates, totalling 100 fits
Split n.1 | Best hyper-parameters per SGD: {'alpha': np.float64(0.0001), 'fit_intercept': False}
Accuracy split n.1: 83.33%
Fitting 10 folds for each of 10 candidates, totalling 100 fits
Split n.2 | Best hyper-parameters per SGD: {'alpha': np.float64(0.0002), 'fit_intercept': False}
Accuracy split n.2: 77.27%
Fitting 10 folds for each of 10 candidates, totalling 100 fits
Split n.3 | Best hyper-parameters per SGD: {'alpha': np.float64(0.0001), 'fit_intercept': False}
Accuracy split n.3: 77.65%
Fitting 10 folds for each of 10 candidates, totalling 100 fits
Split n.4 | Best hyper-parameters per SGD: {'alpha': np.float64(0.0002), 'fit_intercept': False}
Accuracy split n.4: 79.55%
Fitting 10 folds for each of 10 candidates, totalling 100 fits
Split n.5 | Best hyper-parameters per SGD: {'alpha': np.float64(0.0002), 'fit_intercept': True}
Accuracy split n.5: 73.86%
Fitting 10 folds for each of 10 candidates, totalling 100 fits
Split n.6 | Best hyper-parameters per SGD: {'alpha': np.float64(0.0001), 'fit_intercept': False}
Accuracy split n.6: 76.05%
Fitting 10 folds for each of 10 candidates, totalling 100 fits
Split n.7 | Best hyper-parameters per SGD: {'alpha': np.float64(0.0002), 'fit_intercept': True}
Accuracy split n.7: 75.29%
Fitting 10 folds for each of 10 candidates, totalling 100 fits
Split n.8 | Best hyper-parameters per SGD: {'alpha': np.float64(0.0001), 'fit_intercept': True}
Accuracy split n.8: 73.38%
Fitting 10 folds for each of 10 candidates, totalling 100 fits
Split n.9 | Best hyper-parameters per SGD: {'alpha': np.float64(0.0002), 'fit_intercept': True}
Accuracy split n.9: 78.33%
Fitting 10 folds for each of 10 candidates, totalling 100 fits
Split n.10 | Best hyper-parameters per SGD: {'alpha': np.float64(0.0002), 'fit_intercept': True}
Accuracy split n.10: 73.00%

```

Grid search

Nei grid search applicati sia al SGD che al RandomForest, è stato scelto di utilizzare un KFold con 10 suddivisioni. Sebbene il valore di default preveda 5 suddivisioni, si è ritenuto che una divisione in un numero maggiore di Fold fosse più vantaggiosa per ottenere una maggiore precisione. Inoltre, i parametri sono stati impostati in modo personalizzato, dopo aver esaminato attentamente la documentazione delle relative funzioni.

Funzionamento del grid search

Partendo dal training set, si è scelto di suddividerlo in k parti (fold) e, di conseguenza, in k splits. Per ogni split, si è optato per utilizzare una parte del fold come test set, al fine di determinare i migliori parametri per il modello. Una volta individuati tali parametri, si è proceduto alla classificazione utilizzando il test set originale. Per quanto riguarda la Grid Search sul SGD, si è scelto di impostare un intervallo per il parametro 'alpha' come `np.arange(0.0001, 0.0006, 0.0001)`, incrementando il valore di alpha di 0.0001 per ogni split, così da eseguire 100 fit. Il parametro `fit_intercept` è stato impostato come una lista `[True, False]`, per provare entrambi i valori per ogni valore di alpha. La stessa metodologia è stata applicata per il Grid Search del RandomForest, utilizzando valori diversi, ma seguendo lo stesso approccio.

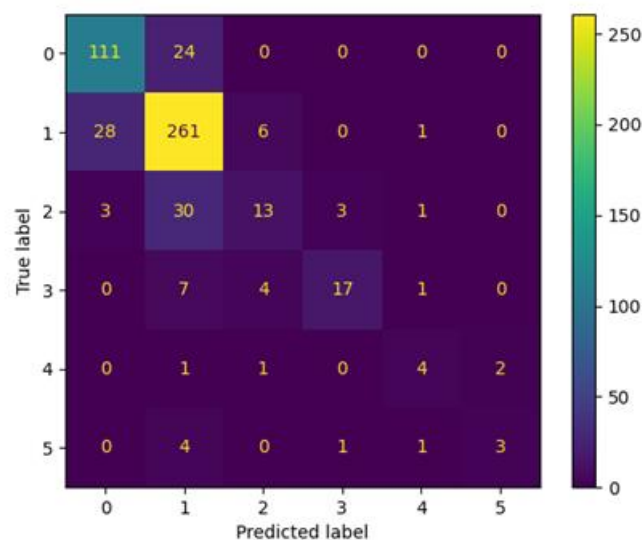
Accuracy di SGD

Per verificare l'accuratezza del modello SGD, si è proceduto ad allenarlo sul training set e a eseguire l'algoritmo per ottenere le probabilità sulle predizioni del test set. Poiché i valori Groundtruth del test set appartenevano a un determinato range (ad esempio "1, 3, 3, 2, 1, 5, 4"), si è calcolato l'indice della probabilità maggiore (np.argmax) sulle predizioni, come descritto in precedenza. Successivamente, attraverso un semplice confronto, si è verificato se gli indici predetti corrispondevano ai valori Groundtruth, popolando la variabile "correct" di conseguenza. Per calcolare l'accuratezza finale del modello, si è diviso il numero di predizioni corrette per il numero totale di sample nel test set $((\text{correct}/\text{prices_y_test.size}) * 100)$. Inoltre, è stata inclusa la stampa del Classification Report, contenente i valori di Precision ($\text{tp}/(\text{tp}+\text{fp})$), Recall ($\text{tp}/(\text{tp}+\text{fn})$), F1-score ($2 * (\text{precision} * \text{recall}) / (\text{precision} + \text{recall})$) e Support (campioni per ogni classe), calcolati per ciascuna classe.

```
Accuracy SGD: 79.13%
Classification report for SGD:
```

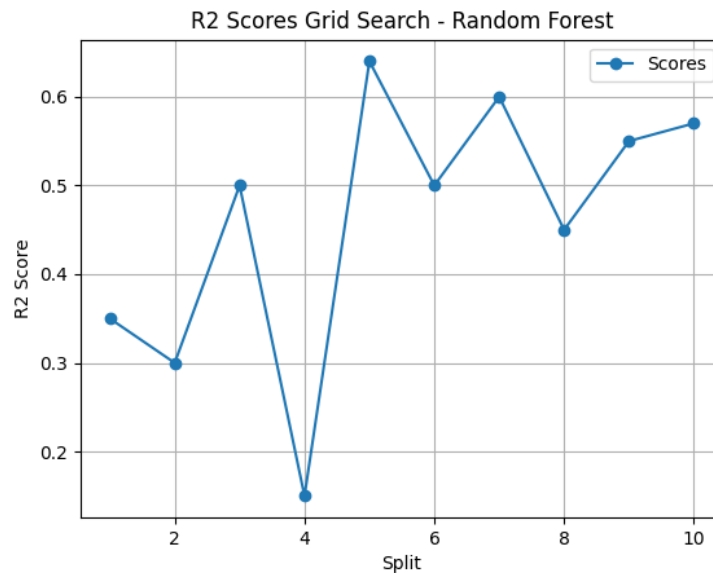
	precision	recall	f1-score	support
0	0.823	0.793	0.808	135
1	0.807	0.905	0.854	296
2	0.621	0.360	0.456	50
3	0.739	0.586	0.654	29
4	0.500	0.500	0.500	8
5	0.600	0.333	0.429	9
accuracy			0.791	527
macro avg	0.682	0.580	0.617	527
weighted avg	0.782	0.791	0.780	527

A supporto del Classification Report, è stata inserita la stampa della Confusion Matrix, che mostra il numero di campioni classificati in ciascuna classe:



Random Forest

Dopo aver testato vari modelli, come il KNN e il LinearRegressor, si è determinato che il ForestModel fosse quello più adatto alle nostre esigenze. Successivamente, applicando il grid search anche a questo modello, sono stati individuati i migliori parametri per la sua implementazione. Come evidenziato dal grafico seguente, i migliori parametri corrispondevano al valore di R2-score ottenuto con lo split numero 5.



Guida all'utilizzo

Scaricare il repository

È possibile scaricare il repository del progetto cliccando [qui](#).

Installare i requisiti

Una volta ottenuta la cartella del progetto, è necessario dirigersi all'interno (da terminale) e installare i requisiti con il seguente comando:

```
pip install -r requirements.txt
```

Questo consentirà di ottenere tutte le dipendenze (librerie e pacchetti) necessarie per l'esecuzione.

Esecuzione del software

È possibile eseguire il programma attraverso un IDE che supporti Python (PyCharm, Visual Studio Code, ...) o da terminale eseguendo il file 'main.py'.

Nel secondo caso, è sufficiente dirigersi all'interno della cartella contenente il file 'main.py' ed eseguire il seguente comando:

```
python main.py
```

Spiegazione della GUI

È stata progettata una GUI (Graphical User Interface) per consentire all'utente un'interazione più intuitiva e user-friendly. Si è voluto progettare un'interfaccia semplice, dove sono presenti undici campi per inserire i dati relativi all'auto che si desidera sottoporre alla valutazione e un dodicesimo campo destinato alla scelta del modello di predizione.

Car Price Estimator

Casa costruttrice: Acura

Modello: ILX 2.0L w/Premium Package

Anno: 2024

Km percorsi: 160.93

Tipo di carburante: Diesel

Motore: 1.2L I3 12V GDI DOHC Turbo

Trasmissione: 10-Speed A/T

Colore esterni: Agate Black Metallic

Colore interni: Beige

Incidenti: 0

Titolo pulito: 1

Modello di predizione: Random Forest

Avvia Predizione

Qui viene mostrato il risultato della predizione

Parametri richiesti in input

Gli undici campi da compilare consentono di scegliere il valore opportuno per ciascuna label associata tra una lista di valori predefiniti. I campi sono i seguenti:

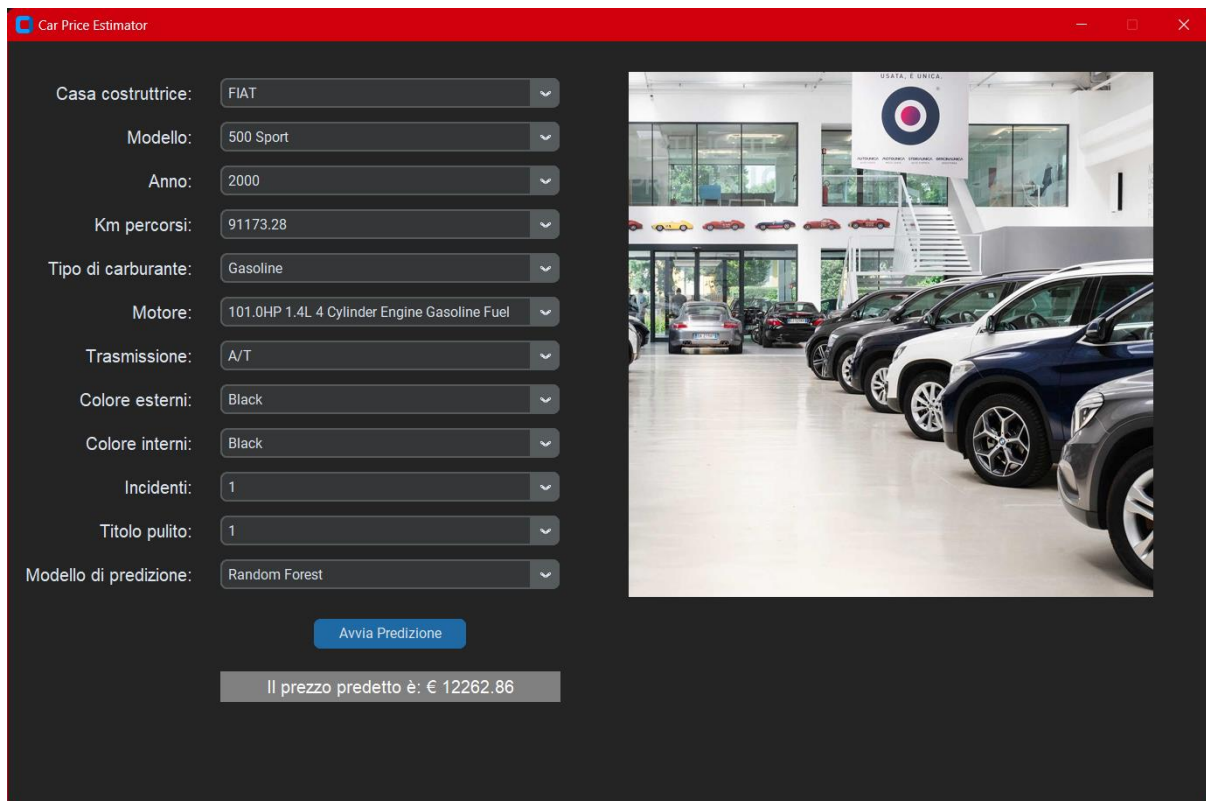
- **Casa costruttrice:** brand dell'auto
- **Modello:** modello dell'auto, selezionabile tra i modelli associati alla casa automobilistica scelta
- **Anno:** anno di immatricolazione dell'auto
- **Km percorsi:** totale dei chilometri percorsi dall'auto
- **Tipo di carburante:** tipo di alimentazione del motore
- **Motore:** modello del motore con caratteristiche relative alla cilindrata
- **Trasmissione:** tipo di cambio e numero di marce
- **Colore esterni:** colore della carrozzeria
- **Colore interni:** colore degli interni dell'abitacolo
- **Incidenti:** indica se l'auto ha subito incidenti registrati
- **Titolo pulito:** certifica che il veicolo non ha vincoli legali o segnalazioni di furto

Modello di predizione

È possibile scegliere tra due modelli di predizione: Random Forest e SGD. Di seguito sono mostrate le principali differenze tra i due modelli.

Random Forest

Questo modello offre in output un valore numerico continuo, risulta adatto per problemi di regressione e ha come vantaggi alta precisione, robustezza e gestione di relazioni non lineari.



Car Price Estimator

Casa costruttrice: FIAT

Modello: 500 Sport

Anno: 2000

Km percorsi: 91173.28

Tipo di carburante: Gasoline

Motore: 101.0HP 1.4L 4 Cylinder Engine Gasoline Fuel

Trasmissione: A/T

Colore esterni: Black

Colore interni: Black

Incidenti: 1

Titolo pulito: 1

Modello di predizione: Random Forest

Avvia Predizione

Il prezzo predetto è: € 12262.86

SGD (Stochastic Gradient Descent)

Questo modello offre in output la probabilità che il sample introdotto appartenga ad un range di valori, risulta ideale per problemi di classificazione e ha come vantaggi la sua velocità e scalabilità.

Casa costruttrice: FIAT

Modello: 500 Sport

Anno: 2000

Km percorsi: 91173.28

Tipo di carburante: Gasoline

Motore: 101.0HP 1.4L 4 Cylinder Engine Gasoline Fuel

Trasmissione: A/T

Colore esterni: Black

Colore interni: Black


Incidenti: 1

Titolo pulito: 1

Modello di predizione: SGD

Avvia Predizione

Questo sample ha probabilità 96.46% di rientrare nella fascia € (0, 16000).



Conclusioni

In definitiva, si può godere di notevole soddisfazione nel risultato finale del progetto che riesce a fornire uno strumento, seppur sviluppato in un contesto didattico, di grande impatto e utilità. Si tratta di un software che potrà permettere, ad utenti intenzionati a vendere la propria auto o a semplici appassionati di motori, di scoprire la valutazione (approssimativa) di un'automobile.

Sono state ipotizzate future implementazioni:

- Estensione del dataset con nuovi brand e modelli
- Inclusione di nuove features, come optional, stato degli interni, condizione della carrozzeria, numero di proprietari
- Considerazione del trend di mercato per affinare la predizione del prezzo

Riferimenti Bibliografici

1. [Kaggle](#)
2. D. L. Poole & A. K. Mackworth - [Artificial Intelligence: Foundations of Computational Agents](#). - 3/e. Cambridge