Documentazione Caso di Studio Ingegneria della Conoscenza

A.A. 2022/23

Stipendi Data Science

Gruppo di lavoro:

Guerra Davide, 741380, d.guerra7@studenti.uniba.it

Repository GitHub:

Dguerra7/progetto_icon_22-23 (github.com)

Sommario

Introduzione

Lo scopo di questo progetto è analizzare i dati relativi ai lavoratori in diversi campi delle Data Science, con l'obiettivo di estrarre una relazione tra il livello di esperienza del lavoratore all'interno dell'azienda e la sua retribuzione annua. A tale scopo è stato utilizzato un dataset chiamato "Data Science Salaries 2023" [1], disponibile sulla piattaforma Kaggle.

Argomenti di interesse:

- **Clustering**: È stata utilizzata questa tecnica di apprendimento non supervisionato per suddividere i lavoratori in diverse categorie.
- **Knowledge Base**: È stata creata una base di conoscenza per condurre analisi delle informazioni contenute nel dataset, al fine di dedurre relazioni tra lo stipendio dei lavoratori e altri fattori come il livello di esperienza o il loro ruolo all'interno dell'azienda, utilizzando opportune query.
- **Belief Network**: Sono state impiegate reti bayesiane per inferire alcune informazioni statistiche dal dataset, attraverso l'inferenza tra le diverse caratteristiche presenti.
- **Apprendimento supervisionato**: Sono stati utilizzati i metodi dell'apprendimento supervisionato per risolvere un problema di regressione. In particolare, sono stati scelti e testati diversi modelli al fine di stimare lo stipendio di un lavoratore basandosi sulle feature presenti nel dataset.

1- Data cleaning

Come prima cosa, sono state eliminate alcune feature ritenute irrilevanti per l'analisi o che fornivano una scarsa varietà di dati, al fine di ridurre il rischio di overfitting. Queste feature sono: "remote_ratio" (percentuale di lavoro svolto in smart working), "work_year" (anno in cui è avvenuta la retribuzione) e "employment_type" (tipo di contratto del lavoratore, ad esempio part-time, full-time, ecc.).

Un'altra cosa da evidenziare è che il dataset originale conteneva due feature relative al salario annuo del lavoratore e alla valuta dello stipendio, mentre un'altra feature conteneva il medesimo stipendio annuo, ma convertito in valuta USD. Per semplificare il raggiungimento degli obiettivi prefissati in questo progetto, è stata presa la decisione di rimuovere le prime due feature, lasciando nel dataset solo gli stipendi convertiti in un'unica valuta.

In secondo luogo, sono stati eliminati gli esempi che presentavano lo stipendio massimo e lo stipendio minimo, poiché mostravano una certa discordanza rispetto ai loro successori/ predecessori, con il potenziale rischio di generare overfitting.

Infine, il dataset pulito è stato salvato in un file CSV chiamato "ds_salaries_usd".

2- Clustering

Sommario e Strumenti utilizzati

La tecnica del clustering è stata scelta per raggruppare i lavoratori in categorie basate sulle caratteristiche presenti nel file 'ds_salaries_usd.csv'. L'obiettivo è individuare dei cluster, ovvero gruppi di esempi simili a centroidi calcolati in modo automatico. L'utilizzo di questa tecnica nel progetto mira a creare una nuova colonna da aggiungere alle features relative ai dati di ciascun lavoratore, in un nuovo file chiamato 'ds_salaries_usd+clusters.csv'.

Il clustering può essere di due tipi:

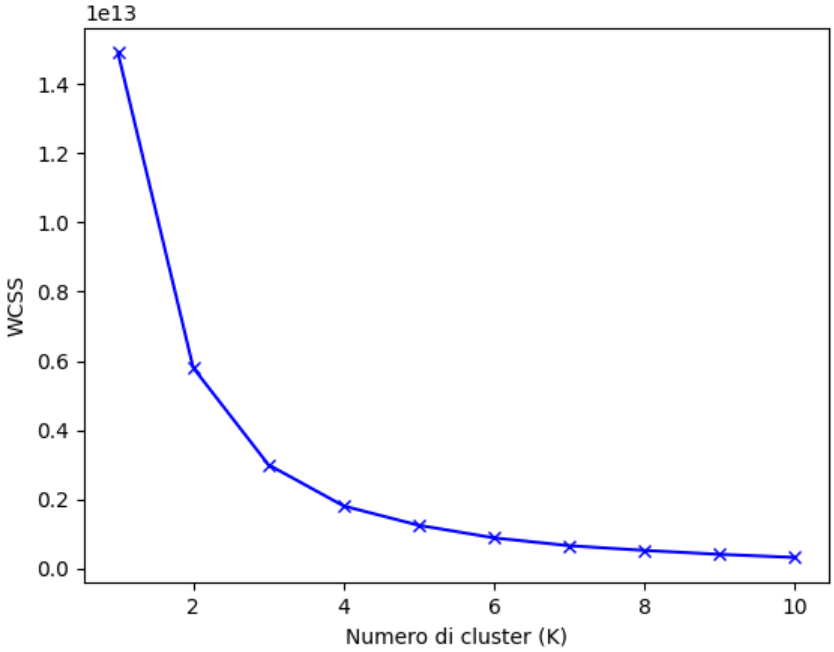
- Clustering rigido (Hard clustering): prevede l'assegnazione statica di ciascun esempio a una classe di appartenenza.
- Clustering morbido (Soft clustering): utilizza distribuzioni di probabilità per assegnare le classi associate a ciascun esempio.

Nel nostro caso, abbiamo scelto di utilizzare una tecnica di clustering rigido, in particolare il k-means [7], implementato tramite la libreria scikit-learn.

Decisioni di Progetto

Dato che la maggior parte delle caratteristiche nel dataset sono di natura categorica e l'algoritmo k-means accetta solo caratteristiche numeriche, è stato necessario convertirle. A tal fine, abbiamo scelto di utilizzare la tecnica di codifica one-hot encoding [6]. Questo approccio crea una nuova variabile binaria univoca per ogni categoria.

Una volta codificate le variabili categoriche, abbiamo dovuto determinare il numero ottimale di cluster. A tal scopo, abbiamo utilizzato la nota tecnica del metodo del gomito (elbow method) [8], in cui sull'asse delle ordinate sono rappresentati i valori delle somme dei quadrati intra-cluster (WCSS), mentre sull'asse delle ascisse sono rappresentati i k cluster.



Come è possibile evincere dal grafico il gomito è piuttosto evidente; dunque, è stato deciso di prendere un valore di k cluster pari a 3.

Successivamente, il modello k-means è stato addestrato sui 3 cluster. Le metriche scelte per la configurazione del modello sono state:

- **N_cluster** = 3: il numero di cluster in cui il dataset deve essere diviso.
- **random_state** = 42: questo parametro controlla la riproducibilità dei risultati. Fissando un valore alto per random_state, il modello fornirà sempre gli stessi risultati quando viene addestrato con gli stessi dati.
- **n_init** = 10: specifica il numero di volte che l'algoritmo viene eseguito con diverse inizializzazioni casuali dei centroidi. L'algoritmo seleziona la soluzione migliore tra i tentativi basandosi sulla somma dei quadrati delle distanze. Un valore maggiore di n_init aumenta le probabilità di ottenere una soluzione di migliore qualità a scapito di un maggior tempo di calcolo. È stato scelto un valore non troppo alto, ma sufficiente per massimizzare la qualità dell'output.

Valutazione

Infine, sono state valutate le prestazioni del modello utilizzando due metriche:

- **WCSS** (Within-Cluster Sum of Squares): calcola la somma dei quadrati delle distanze di ogni punto rispetto al proprio centroide di cluster. Un valore più basso di WCSS indica una maggiore coesione all'interno dei cluster.
- **Silhouette Score**: valuta la coesione all'interno dei cluster e la separazione tra i cluster. È calcolato per ciascun punto e rappresenta il rapporto tra la distanza media al proprio cluster e la distanza media ai cluster più vicini. Un punteggio più alto del Silhouette Score indica una migliore separazione dei cluster.

WCSS	2945778534332.99
Silhouette Score	0.536

In generale, con uno Silhouette Score pari a 0.536 è possibile affermare che i risultati presentano una separazione abbastanza buona tra i cluster, con una divisione omogenea dei dati.

3- Knowledge Base

Sommario e Strumenti utilizzati

Una base di conoscenza (Knowledge Base) in logica del primo ordine è composta da un insieme di proposizioni, chiamate assiomi, che vengono considerate vere senza la necessità di dimostrazione. La base di conoscenza viene utilizzata per rappresentare la conoscenza di un particolare dominio all'interno di una macchina.
Il processo di creazione di una base di conoscenza prevede i seguenti passaggi:

Decidere il dominio da rappresentare: Questo può includere aspetti del mondo reale, un mondo immaginario o un mondo astratto come numeri e insiemi.

1. Definire le proposizioni atomiche: Il progettista deve selezionare le proposizioni atomiche che saranno utilizzate per rappresentare il mondo nel dominio scelto.
2. Assiomatizzare il dominio: Il progettista deve definire le proposizioni che saranno vere nell'interpretazione del dominio. Queste proposizioni costituiscono gli assiomi della base di conoscenza.
3. Porre delle query al sistema: Una volta definita la base di conoscenza, è possibile sottoporre al sistema delle query per determinare se specifiche proposizioni sono conseguenze logiche della base di conoscenza (cioè se sono vere in tutti i modelli della base di conoscenza).

Il sistema, a differenza del progettista, non comprende il significato dei simboli utilizzati, ma è in grado di decidere se una particolare proposizione è una conseguenza logica della base di conoscenza, basandosi sugli assiomi presenti. Successivamente, il progettista, in base all'interpretazione del dominio, può valutare se il risultato ottenuto dal sistema è valido o meno.

Per definire una base di conoscenza è stato scelto di utilizzare Prolog , tramite la libreria pyswip [2]. Le features selezionate sono state assiomatizzate come fatti nella base di conoscenza, e sono stati definiti anche i loro domini. Inoltre, alcune di queste features sono state combinate per creare regole più complesse, ad esempio, il livello di esperienza e il salario, o il ruolo, il salario e il cluster.

Decisioni di Progetto

In seguito, sono state aggiunte regole che consentono all'utente di porre delle query al sistema. Alcune di queste regole saranno illustrate di seguito.

```
count_role_occurrences(Role, Count) :-
            findall(Role, role(Role), Roles),
            length(Roles, Count).
```

La regola **count_role_occurence(Role, Count)** è in grado di contare le occorrenze di un determinato ruolo Role e le assegna alla variabile Count. Nello specifico le operazioni che effettua sono:

1. Il predicato **findall(Role, role(Cont), Roles)** trova tutte le istanze di Role che soddisfano il fatto role(Role), e che verranno salvate all'interno della lista Roles.
2. Il predicato **length(Roles, Count)** calcola la lunghezza della lista Roles e la memorizza nella variabile Count.

Un esempio di query è:

Input count_role_occurrences("Data Engineer", Count)

Output [{'Count': 1040}]

```
common_roles(Role) :-
            employment_type(Role),
            count_role_occurrences(Role, Count),
            Count >= 500.
```

La regola **common_roles(Role, Count)** elenca i ruoli più comuni. Nello specifico le operazioni che effettua sono:

1. Il predicato **employment_type(Role)** specifica il tipo di valoro svolto.
2. La query **count_role_occurence(Role, Count)** conta le occorrenze di un determinato lavoro.
3. La clausola **Count >= 500** impone una condizione in cui il conteggio delle occorrenze dev'essere maggiore o uguale a 500 per considerarsi un ruolo comune.

Un esempio di query è:

Input common_roles(Role)

Output [{'Role': b'Data Engineer'}, {'Role': b'Data Scientist'},

{'Role': b'Data Analyst'}]

```
average_salary_for_job_cluster(Role, Cluster, AverageSalary) :-
            findall(Salary, role_salary_and_cluster(Role, Salary, Cluster), Salaries),
            length(Salaries, Count),
            sum_list(Salaries, Total),
            AverageSalary is Total / Count.
```

La regola **average_salary_for_job_cluster(Role, Cluster, AverageSalary)** è in grado di calcolare lo stipendio medio AvarageSalary in base al ruolo Role, ed al culster Cluster di appartenenza del lavoratore. Nello specifico le operazioni che effettua sono:

1. Il predicato **findall(Salary, role_salary_and_cluster(Role, Salary, Cluster), Salaries)** crea una listacontenente tutti gli stipendi Salary corrispondenti ad un determinato ruolo Role ed un determinato Cluster.
    a. Il predicato **role_salary_and_cluster(Role, Salary, Cluster)**, richiamato in input all'interno del predicato findall, restituisce lo stipendio corrispondente a un determinato ruolo e cluster.
2. Il predicato **length(Salaries, Count)** conta il numero di elementi nella lista Salaries e restituisce il risultato Count, che rappresenta il numero di stipendi trovati.
3. Il predicato sum_list(Salaries, Total) somma tutti gli elementi nella lista Salaries e restituisce il risultato Total.
4. Il predicato **AverageSalary is Total / Count** assegna alla variabile AverageSalary il risultato della divisione tra la variabile Total e la variabile Count.

Un esempio di query è:

Input average_salary_for_job_cluster("Data Engineer", "0", verageSalary)

Output [{'AverageSalary': 145359.64575645755}]

4- Belief Network

Sommario e Strumenti utilizzati

La costruzione e l'apprendimento della Belief Network è stato effettuato al fine di evidenziare la correlazione tra le features presenti nel dataset e di consentire l'esecuzione di interrogazioni basate sull'inferenza probabilistica.

Poiché la struttura della rete bayesiana non era nota a priori, è stato necessario apprenderla dai dati disponibili. A tal fine, è stato utilizzato l'applicativo Weka [9], un ambiente software open source per l'apprendimento automatico e l'analisi dei dati. Weka ci ha permesso di ottenere la struttura della rete bayesiana in base a determinati parametri definiti in base agli obiettivi del progetto.

Successivamente, utilizzando la libreria "pgmpy", la rete bayesiana è stata caricata da un file XML su Python, consentendo di eseguire interrogazioni sulla rete.

Decisioni di Progetto

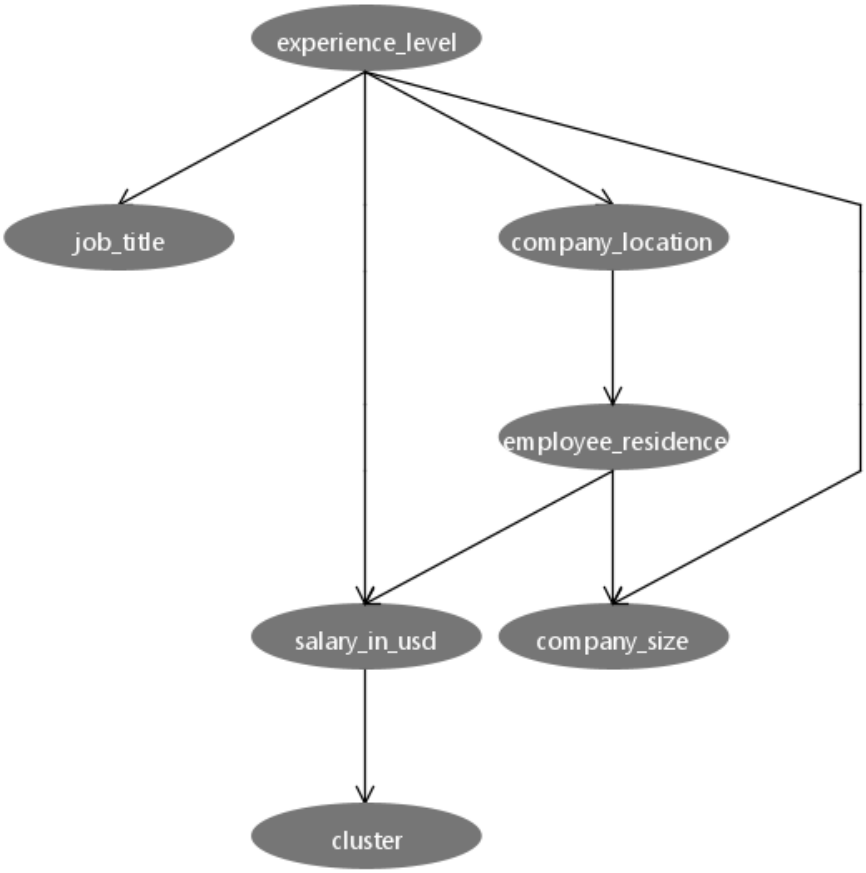
Per l'algoritmo di ricerca della Belief Network è stato scelto "Hill Climber" per la sua efficienza computazionale e la sua capacità di convergere rapidamente verso soluzioni ragionevoli, soprattutto in contesti con dataset di dimensioni ridotte. Per quanto riguarda il numero massimo di genitori, è stato scelto il valore 2, in quanto un valore superiore avrebbe aumentato solo la complessità della rete, aumentando il rischio di overfitting.

Come stimatore è stato utilizzato il "SimpleEstimator" con parametro a = 0,5. Questo parametro viene utilizzato per l'apprendimento della Conditional Probability Table (CPT) ed è interpretato come un conteggio iniziale per ogni valore. Si è scelto di mantenere il valore di default poiché non si conosce la distribuzione dei dati o eventuali pseudo-conteggi.

È stato inoltre ottimizzato il processo di selezione della struttura utilizzando la k-fold Cross Validation con un numero di fold pari a 10.

Valutazione

Qui di seguito mostriamo la struttura della Rete Bayesiana generata da Weka conseguentemente ai parametri scelti:



Weka ha anche permesso di calcolare alcune misure di valutazione dei modelli di reti bayesia:

LogScore Bayes	-26015.578
LogScore BDeu	-100932.404
LogScore MDL	-77333.264
LogScore ENTROPY	-41494.37
LogScore AIC	-50203.375

Sono state selezionate alcune metriche per la classificazione:

Classi esperienza	Precision	Recall	F-Measure
SE	0.755	0.951	0.842
MI	0.593	0.278	0.378
EN	0.500	0.275	0.355
EX	0.333	0.088	0.139

Ed è stata generatala matrice di confusione:

	SE	MI	EN	EX

SE	2393	85	25	13
MI	514	233	61	5
EN	166	64	88	2
EX	98	4	2	10

Dopo aver appreso la struttura della Belief Network, è possibile scrivere da linea di comando delle query con cui interrogarla, effettuando calcoli di inferenza probabilistica facendo uso dell'algoritmo di eliminazione delle variabili.

Uno dei possibili scopi del poter porre queste query è ottenere una stima probabilistica della relazione tra livello di esperienza lavorativa e salario, ma è anche possibile ottenere relazioni di ogni tipo tramite la combinazione di più feature.

Alcuni esempi di possibili query fatte da linea di comando sono:

Input salary_in_usd: experience_level=SE, job_title=Data Scientist,
employee_residence=US, company_location=US, company_size=M

Output 63020-100250 : 0.10037668956348327
100250-165110 : 0.48238422335475295
+165110 : 0.4017283403500997

Input cluster: experience_level=EN, job_title=Business Data Analyst,
company_location=US, employee_residence=US, company_size=L,
salary_in_usd=63020-100250

Output 1 : 0.9984532095901004

5- Supervised Learning

Sommario e Strumenti utilizzati

In questo progetto, l'obiettivo dell'apprendimento supervisionato è risolvere un problema di regressione, in particolare utilizzando la colonna 'salary_in_usd' come variabile target da predire basandosi sulle altre caratteristiche relative ai dati dei lavoratori.

Inizialmente è stato necessario decidere quali modelli utilizzare. A causa della quantità limitata di dati nel dataset, sono stati esclusi modelli con complessità elevata come la regressione lineare e le reti neurali. Invece, sono stati scelti tre altri modelli, e ne è stata valutata l'efficacia e l'efficienza:

- Random Forest, utilizzando la libreria Scikit-learn [3].
- XGBoost, utilizzando la libreria xgboost [4].
- K-nearest-neighbors (KNN), sempre utilizzando la libreria Scikit-learn [5].

Poiché la maggior parte delle feature nel dataset sono categoriche e questi modelli non supportano le caratteristiche categoriche, è stato necessario codificarle in valori numerici. A tal scopo, è stata nuovamente scelta la tecnica dell’one-hot encoding [6].

Una volta codificate le caratteristiche, è stata applicata la tecnica di cross-validation suddividendo il dataset in modo da utilizzare l'80% per il training e il restante 20% per il testing, al fine di massimizzare i dati disponibili per l'addestramento dei modelli.

Andiamo adesso ad analizzare i parametri con cui sono stati addestrati i modelli, le metriche con cui sono stati valutati, ed i risultati.

Decisioni di Progetto

Random Forest

L'algoritmo Random Forest tende ad avere una buona precisione e stabilità nei problemi di regressione, sia con dataset piccoli che con dataset di grandi dimensioni. Inoltre, la sua capacità di addestrare gli alberi in modo parallelizzato consente una rapida esecuzione anche con risorse computazionali limitate.

I parametri scelti sono i seguenti:

- **Max_depth** = 4: in generale, una maggiore profondità può consentire al modello di addestrarsi meglio sui dati di addestramento, ma in questo caso, trattandosi di un dataset non particolarmente complesso, è stato scelto un valore basso di profondità per favorire la rapidità del modello.
- **N_estimators** = 4: anche in questo caso, un valore maggiore tendenzialmente potrebbe aumentare la stabilità del modello, ma con il nostro dataset è stato constatato che anche

un numero ridotto di alberi garantisce una buona stabilità e allo stesso tempo favorisce la rapidità del modello.

- **Random_state** = 42: questo parametro controlla la riproducibilità dei risultati. Fissando un valore alto per random_state, il modello fornirà sempre gli stessi risultati quando viene addestrato con gli stessi dati.

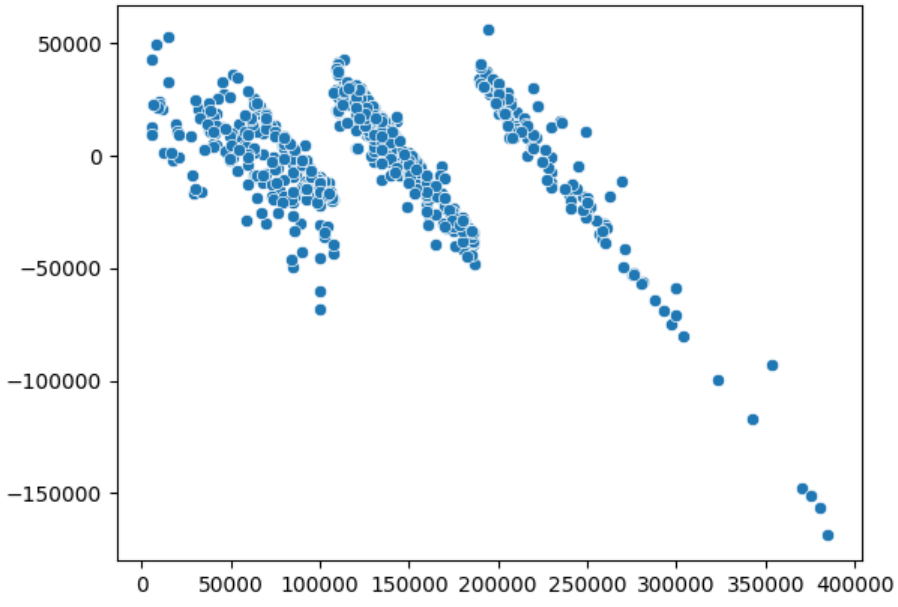
Le metriche per la valutazione del modello:

MSE	669240041.27
MAE	19824.40
MdAE	17436.12
R2-score	0.833

Tempo di esecuzione per X iterazioni:

10	0.061 s.
100	0.053 s.
1000	0.057 s.

I risultati:



I risultati ottenuti mostrano che Random Forest è un modello efficace ed efficiente, con un margine di errore limitato. Tuttavia, come si può notare dal grafico, l'errore tende ad aumentare con l'aumentare del salario.

XGBoost

L'algoritmo XGBoost è noto per la sua velocità di esecuzione e le prestazioni generalmente superiori rispetto ad altri algoritmi di machine learning. Utilizza un'implementazione efficiente del boosting tree e include ottimizzazioni che consentono di addestrare modelli anche complessi in tempi relativamente brevi. XGBoost offre anche una vasta gamma di funzioni obiettivo che consentono di adattarsi a diversi tipi di problemi di regressione. Queste funzioni obiettivo possono essere selezionate in base alle caratteristiche specifiche del problema e al tipo di metrica di valutazione desiderata.

I parametri scelti per XGBoost sono i seguenti:

- **Max_depth** = 5: in generale, una maggiore profondità dovrebbe rendere il modello più preciso. Tuttavia, con il nostro dataset, la precisione del modello tende a diminuire, anche se leggermente, all'aumentare della profondità. Ciò può compromettere anche la rapidità di esecuzione del modello. Pertanto, è stato scelto un valore basso che bilanci tra velocità e prestazioni.
- **N_estimators** = 10: a differenza di Random Forest, XGBoost ha richiesto un maggior numero di alberi per risolvere il problema con maggiore efficacia.
- **Random_state** = 42: questo parametro controlla la riproducibilità dei risultati. Fissando un valore alto per random_state, il modello fornirà sempre gli stessi risultati quando viene addestrato con gli stessi dati.
- **Objective** = 'reg:squarederror': è stata specificata come funzione obiettivo da ottimizzare durante l'addestramento del modello l'errore quadratico medio (MSE), calcolato tra le previsioni del modello e i valori di output attesi.

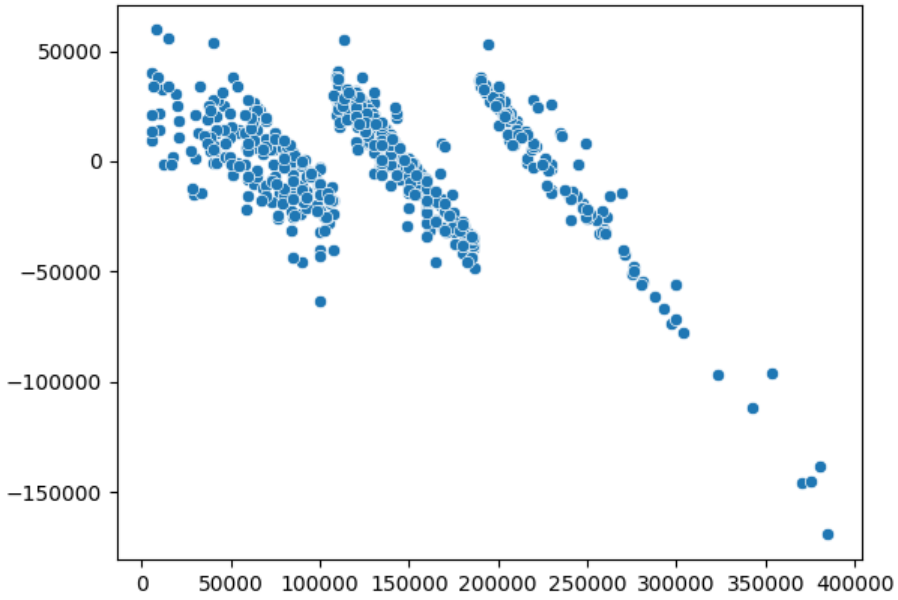
Le metriche per la valutazione del modello:

MSE	679620934.41
MAE	19270.22
MdAE	16237.67
R2-score	0.83

Tempo di esecuzione per X iterazioni:

10	0.254 s.
100	0.077 s.
1000	0.065 s.

I risultati:



I risultati ottenuti mostrano che XGBoost è anch'esso un modello efficace, ma a discapito del tempo di esecuzione che risulta essere superiore rispetto a quello di Random Forest. Inoltre, anche in questo caso, è possibile notare dal grafico che all'aumentare del salario si ha un margine d'errore più ampio.

K-nearest-neighbors (KNN)

KNN è un modello versatile che funziona bene con dataset di qualsiasi dimensione ed è robusto anche in presenza di dati rumorosi o valori anomali. Poiché il modello si basa sui vicini più prossimi, un singolo valore anomalo avrà meno impatto sull'output finale.

I parametri scelti per KNN sono i seguenti:

- **n_neighbors** = 6: questo parametro indica il numero di vicini più prossimi considerati per fare una previsione. Influenza la flessibilità del modello: valori più bassi di n_neighbors possono portare a una maggiore sensibilità al rumore e a una maggiore varianza, mentre valori più alti possono portare a un maggiore bias. Abbiamo scelto il valore 6 come compromesso tra prestazioni, che non aumentano significativamente con valori più alti, ed efficienza.

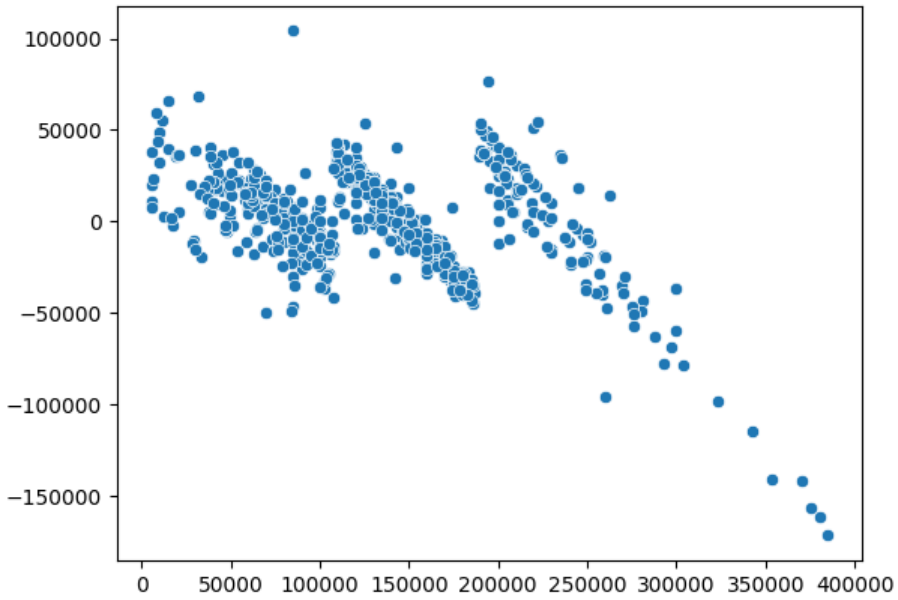
Le metriche per la valutazione del modello:

MSE	753135537.7
MAE	20212.68
MdAE	16462.66
R2-score	0.812

Tempo di esecuzione per X iterazioni:

10	0.079 s.
100	0.065 s.
1000	0.071 s.

I risultati:



I risultati ottenuti mostrano che KNN, pur non avendo prestazioni scarse, risulta leggermente meno preciso e veloce rispetto agli altri due modelli. Pertanto, non offre alcun vantaggio particolare nell'ambito di questo problema di regressione. Inoltre, come evidenziato nel grafico, si osserva un aumento dell'errore con l'aumentare del salario.

Valutazione

In conclusione, possiamo affermare che la Random Forest si è dimostrata il modello più adatto per risolvere il problema di regressione considerato. Ha ottenuto un R2-score pari a 0.833, molto vicino a quello ottenuto dall'XGBoost, ma distinguendosi rispetto agli altri modelli selezionati, dimostrato una maggiore efficienza in termini di velocità.

Conclusioni

In conclusione, questo progetto ha affrontato l'analisi dei dati relativi ai lavoratori nel campo della Data Science, concentrandosi sulla relazione tra il livello di esperienza e la retribuzione annua.

Il processo è iniziato con la pulizia dei dati, eliminando le feature irrilevanti e ridondanti e salvando il dataset pulito in un file CSV. Successivamente, è stata applicata la tecnica di clustering per suddividere i lavoratori in categorie. Il modello k-means è stato addestrato con un numero di cluster pari a 3, ottenuti grazie all'ebow method.

Successivamente, è stata creata una base di conoscenza utilizzando la logica del primo ordine, in particolare con l'uso di Prolog, per dedurre relazioni tra lo stipendio dei lavoratori e altre caratteristiche come il livello di esperienza o il ruolo. Sono state definite regole e query per interrogare il sistema e ottenere informazioni specifiche.

È stata costruita una rete bayesiana per evidenziare le correlazioni tra le feature del dataset e consentire l'inferenza probabilistica. La struttura della rete è stata appresa dai dati utilizzando l'applicativo Weka e successivamente è stata caricata su Python con la possibilità di interrogarla da linea di comando.

Infine, sono stato addestrati tre modelli (Random Forest, XGBoost e Knn) per affrontare un problema di regressione, ovvero predire la feature del salario. Su ciascuno di questi modelli sono state impostati specifici parametri e sono state calcolate delle metriche di valutazione ed il tempo di esecuzione medio su 10, 100, e 1000 iterazioni. Da ciò si è arrivati a concludere che l'algoritmo più adatto a tale scopo è stato quello della Random Forest.

Nonostante il progetto si sia incentrato sulla correlazione tra esperienza lavorativa e salario, sviluppi futuri potrebbero includere l'utilizzo degli strumenti messi a disposizione per effettuare ulteriori analisi come: Differenze tra gli stipendi in varie parti del mondo a parità di ruolo ed esperienza lavorativa; Determinare le differenze tra i lavoratori di piccole, medie e grandi aziende; Valutare la distribuzione dello smart-working nei vari paesi, aggiungendo la corrispondente feature rimossa dal dataset originale; e molto altro ancora. Tutto questo potrebbe essere favorito da un ampliamento del dataset per includere ulteriori informazioni e esplorare nuove relazioni e la sperimentazione di modelli di regressione e ottimizzazione dei rispettivi parametri per ottenere risultati più precisi.

Bibliografia

[1] Data Science Salaries 2023, 13 Aprile 2023, <Data Science Salaries 2023 | Kaggle >

[2] Pypi, Pyswip, <pyswip · PyPI >

[3] Scikit learn, RandomForestRegressor , <sklearn.ensemble.RandomForestRegressor — documentazione di scikit-learn 1.2.2 >

[4] Xgboost, Introduzione agli alberi potenziati, <Introduzione agli alberi potenziati — xgboost 1.7.6 documentazione >

[5] Scikit learn, KNeighborsRegressor, <sklearn.neighbors.KNeighborsRegressor — scikit-learn 1.2.2 documentation>

[6] Scikit learn, OneHotEncoder, <sklearn.preprocessing.OneHotEncoder — scikit-learn 1.2.2 documentation >

[7] scikit learn, Clustering, <2.3. Clustering — scikit-learn 1.2.2 documentation >

[8] scikit learn, Sklearn kmeans equivalent of elbow method, <python 3.x - Sklearn kmeans equivalent of elbow method - Stack Overflow >

[9] Wikipedia, Weka, <Weka - Wikipedia >