RegressorComparison

Daniele Russo

February 10, 2024

Contents

1 Introduzione

1.1 Analisi e contestualizzazione del problema

Oggi giorno, ci troviamo sempre più circondati da tool che utilizzano l'intelligenza artificiale. Questi strumenti necessitano di dati che, molte volte, sono manchevoli, parziali o più semplicemente abbiamo bisogno di un modo per stimare dei valori discreti, ma non sappiamo come. Per questo, si ricorre all'utilizzo dei regressori per completare i propri dati, stimare valori o effettuare previsioni. Come capiamo quale tipo di regressore utilizzare? Di qui nasce l'idea di un comparatore generico che renda facile analizzare le prestazione dei vari tipi di regressori, tale idea viene affiancata ad un caso d'uso pratico. L'obiettivo che si prefigge l'applicativo è quello di predire i sussidi aggiuntivi dello stipendio, ovvero i "Benefits", avendo a disposizione un dataset di lavoratori statali (San Francesco, California), caratterizzato da diverse informazioni, come lo stipendio base, le spese, i sussidi, l'anno, la loro posizione contrattuale e molte altre.

La "Regressione" è una tecnica statistica utilizzata per comprendere la relazione tra una variabile dipendente continua e una o più variabili indipendenti. In termini semplici, un regressore cerca di modellare la funzione che meglio si adatta ai dati osservati, consentendo di effettuare previsioni o stime su valori continui.

Ma come funziona un regressore?

1. **Input dei dati**: Il regressore richiede un insieme di dati di addestramento, composti da coppie di input e output. Gli input sono le variabili indipendenti, mentre gli output sono i corrispondenti valori della variabile dipendente.

2. **Addestramento del modello**: Durante la fase di addestramento, il regressore cerca di apprendere la relazione tra gli input e gli output del dataset. Utilizzando un algoritmo di apprendimento, il modello ottimizza i suoi parametri in modo da minimizzare l'errore tra le previsioni e i valori effettivi.

3. **Generazione della funzione di regressione**: Una volta addestrato, il regressore genera una funzione di regressione che rappresenta la relazione stimata tra le variabili indipendenti e la variabile dipendente.

4. **Previsioni**: Il regressore può quindi essere utilizzato per fare previsioni su nuovi dati o dati non visti in precedenza. Utilizzando la funzione di regressione, il modello stima i valori della variabile dipendente in base ai nuovi input.

1.2 Proposta della soluzione

La soluzione proposta dal candidato è affiancata a un caso d'uso pratico, che prende il nome di "RegressorComparator", dove viene applicata l'idea del comparatore di regressori ad un problema pratico. Data la situazione di un lavoratore, possiamo stimare i suoi "Benefit"? I problemi di regressione sono istanze di problemi di apprendimento supervisionato, quindi in maniera simile alla classificazione, un problema di regressione porta alla costruzione di un modello, per predire i nuovi elementi sulla base del training set. I regressori sono essenzialmente delle funzioni matematiche che cercano di descrivere i dati. Esistono diversi regressori che si distinguono tra di loro per via delle assunzioni fatte sui dati, così come delle specifiche proprietà che portano alla regressione, ma anche del numero di variabili indipendenti (predittori) di cui si dispone.

1.3 Obiettivi e Criteri di Successo

Obiettivi del Progetto: Il progetto mira a sviluppare un modo facile per comparare i modelli di regressione in grado di predire in modo accurato i "Benefits" dei dipendenti. L'obbiettivo è di sopperire alla mancanza di esperienza e rendere accessibile tale tecnologia.

Risultati Attesi: Ci aspettiamo che almeno uno dei modelli sviluppati raggiungano un adattamento del modello superiore al 90%, predicendo con successo i "Benefits".

2 Specifiche dell'ambiente

L'ambiente è un'istanza del problema per la quale l'agente rappresenta la soluzione. La descrizione di un ambiente viene generalmente redatta tramite l'utilizzo della formula PEAS, caratterizzata dai seguenti quattro fattori:

- **Performance** (prestazioni): Per la seguente sezione, era riduttivo dedicare un sottoparagrafo; per tale motivo, le è stato dedicato un approfondimento nel capitolo 7.
- **Environment** (ambiente)
- **Actuators** (attuatori)
- **Sensors** (sensori)

Analizziamo, quindi, punto per punto questi fattori relativamente al problema.

2.1 Envirorment

L'environment è la descrizione degli elementi che formano l'ambiente. Considerando, come già detto, che l'ambiente è un'istanza del problema per la quale l'agente rappresenta una soluzione, l'ambiente sarà composto da una entry del dataset (come sono strutturate le entry verrà specificato in seguito). Le seguenti specifiche, ci permettono di definire l'environment in esame:

- L'ambiente è **completamente osservabile**, in quanto i sensori dell'agente (rappresentati in questo caso da una funzione, come specificato in seguito) hanno completo accesso all'ambiente in ogni momento;
- L'ambiente è **deterministico**, poiché lo stato dell'ambiente dipende dall'azione eseguita dall'agente;
- L'ambiente è **episodico**, perché le varie previsioni sono eventi indipendenti tra loro;
- L'ambiente è **statico**, dal momento che rimane invariato mentre l'agente delibera;
- L'ambiente è **discreto**, poiché fornisce un numero limitato di percezioni ben distinte;
- L'ambiente è **singolo**, in quanto è presente un unico agente.

2.2 Actuators

Gli attuatori sono gli strumenti che ha a disposizione un agente per effettuare le azioni. Nell'ambito del progetto, l'azione che dovrà compiere l'agente (che ricordiamo essere un regressore), sarà predire il valore della variabile dipendente di una entry. In questo caso, quindi, l'attuatore dell'agente sarà una funzione che permetterà di assegnare il valore al campo "Benefit" di una determinata persona.

2.3 Sensors

I sensori sono gli strumenti che l'agente utilizza per prendere in input la percezione dell'ambiente. Nel nostro caso è una entry del dataset, ovvero prenderà in input tutte le variabili indipendenti per valutare e predire la variabile dipendete.

3 Modello di sviluppo

Il progetto ha seguito il modello di sviluppo CRISP-DM, che è riuscito a fornire agilità e flessibilità per il corretto completamento del progetto. In generale, sono state affrontate tutte le fasi in modo iterativo, consentendo di portare facilmente a termine il progetto:

- **Business Understanding**:
  - Definizione degli obiettivi del business.
  - Traduzione degli obiettivi in obiettivi di data mining.
  - Valutazione della situazione corrente.
- **Data Understanding**:
  - Raccolta dei dati necessari per il progetto.
  - Caratterizzazione iniziale dei dati.
  - Esplorazione preliminare dei dati per comprendere la loro natura.
- **Data Preparation**:
  - Pulizia dei dati, gestione dei valori mancanti e rimozione degli outlier.
  - Selezione delle variabili rilevanti.
  - Trasformazione dei dati per renderli adatti all'analisi.
- **Data Modeling**:
  - Selezione delle tecniche di modellazione più adeguate.
  - Creazione di modelli usando i dati di addestramento.
  - Validazione e ottimizzazione dei modelli.
- **Evaluation**:
  - Valutazione delle prestazioni dei modelli rispetto agli obiettivi del business.
  - Se necessario, revisione e ripetizione delle fasi precedenti.
- **Deployment**:
  - Implementazione dei modelli nell'ambiente di produzione.
  - Monitoraggio delle prestazioni dei modelli implementati.

3.1 Team Member:

Il progetto è stato interamente pensato, progettato e sviluppato da:

- Russo Daniele

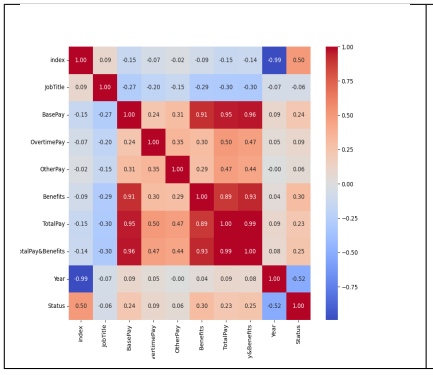
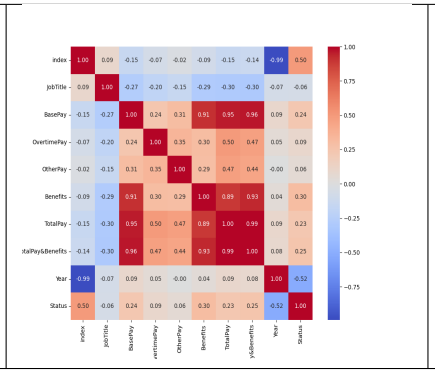
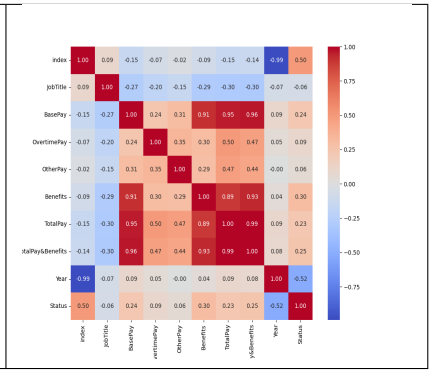
4 Comprensione dei Dati

4.1 Origine, Raccolta e Formato dei Dati

Un regressore è un tipo di algoritmo di machine learning che trae insegnamenti dai dati, per stimare nel nostro caso d'uso, i "Benefits". Questi modelli richiedono un considerevole numero di esempi per un addestramento efficace e per affinare la capacità predittiva del modello. Di qui si intuisce la fondamentale importanza della quantità e qualità dei dati da avere a disposizione. Per stimare i "Benfit", è necessaria una raccolta di dati che comprenda informazioni specifiche su ciascun esempio, insieme alle relative etichette che indicano il valore effettivo dei "Benefits". La ricerca di un dataset appropriato, ha portato all'esplorazione della community Kaggle, dove sono disponibili diversi dataset contenenti informazioni sulla situazione stipendiaria di varie classi di lavoratori. Successivamente a seguito di un'analisi approfondita, è emerso che i dataset che permettano l'utilizzo di regressori, specialmente quelli di tipo lineare, risultano essere pochi. Di conseguenza, la scelta è ricaduta su un dataset composto da colonne che rappresentano i salari dei dipendenti pubblici della California dal 2011 al 2019. Con i seguenti campi: Employee Name, Job Title, Base Pay, Overtime Pay, Other Pay, Benefits, Total Pay, TotalPay&Benefits, Year, Status. Il dataset è composto da circa 357.407 righe e 10 colonne. Di conseguenza, è stato effettuato un lungo studio sui dati, che verrà approfondito nella fase di Data Understanding oltre ad essere stato documentato in un blocco note Collab consultabile qui.

4.2 Analisi Preliminare

Durante il nostro corso, abbiamo studiato 2 tipologie di regressori: i regressori lineari e gli alberi decisionali regressivi. Per applicare la prima tipologia, abbiamo bisogno di determinate condizioni dei dati, che verranno approfondite successivamente. È importante verificare, affinché il regressore sia affidabile, che la variabile dipendente che vogliamo predire, ossia i "Benefits", sia esprimibile in funzione delle altre variabili, quelle indipendenti. In questa sezione, verificheremo l'esistenza di tale condizione, ovvero che i "Benefits" sia esprimibile in funzione delle altre Feature, attraverso la tabella di correlazione. Tutto il lavoro è stato svolto anche con differenti normalizzazioni dei dati, per verificare l'impatto di tale scelta.

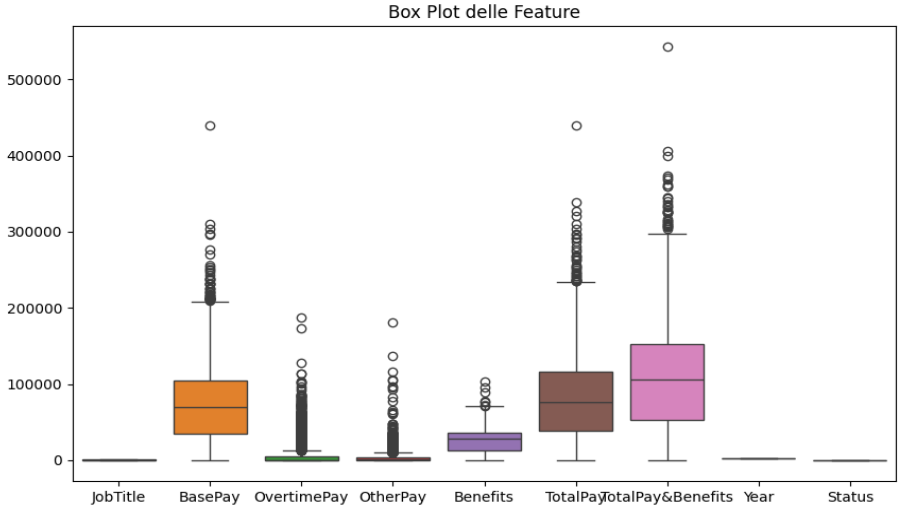
Senza	MinMax	Z-score
		

Come possiamo osservare il campo "Benefits" ha una forte correlazione con tre diverse variabili tra le quali: BasePay, TotalPay e TotalPay&Benefits. Mentre con le altre due ha una correlazione medio-alta come: OvertimePay, OtherPay e Status. Con le restanti, la correlazione è pressoché inesistente.

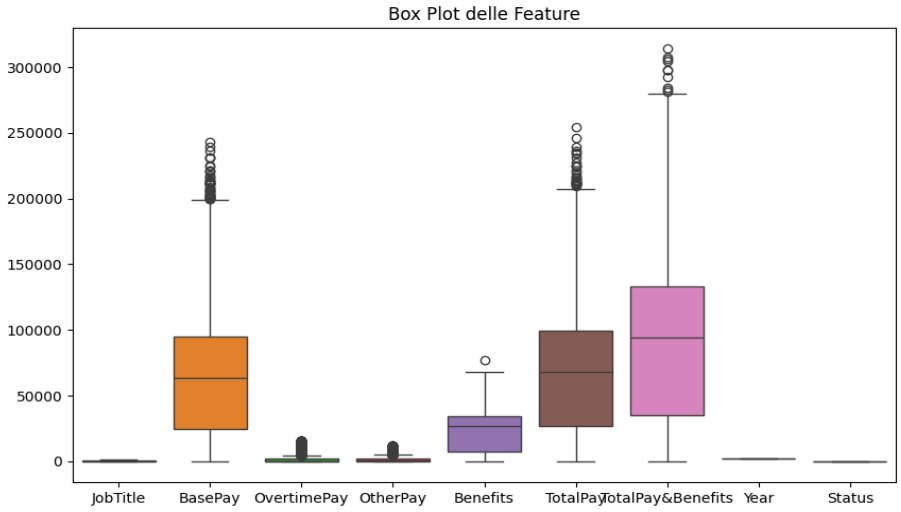
5 Preparazione dei Dati

5.1 Pulizia dei Dati

In questa fase, andremo a sostituire tutti i campi nulli, indicati con Nan, con una stringa facilmente individuabile ovvero "Not Present". Tali valori sono presenti nella colonna "Status", che indica la posizione contrattuale del dipendente. Oltre ai valori Nan, troviamo nel dataset valori non significativi come "Not Provided", vista la grande quantità di entry rimuoviamo anche queste tuple. Inoltre rimuoviamo anche le colonne che non sono di nostro interesse, tra cui: Employee Name, Total Pay, TotalPay&Benefits. Va notato che queste ultime due feature costituirebbero dei leakyPredictor, e per tale motivo sono state eliminate. Oltre alla gestione dei valori non significativi, cerchiamo di effettuare una pulizia degli outlier. Per farlo, utilizziamo il grafico box plot per visualizzare la distribuzione dei valori per ogni feature.



Si può notare che abbiamo diversi valori fuori dalla media, di conseguenza cerchiamo di eliminare tutti i valori fuori dal primo e ultimo quartile. Verifichiamo se così facendo, aumentiamo la coesione dei dati.



Dopo la fase di pulizia, si sono notati aumenti delle prestazioni dei regressori.

5.2 Gestione dei Dati Mancanti

Vista la grande quantità di dati, possiamo rimuovere tutti i campi nulli o vuoti. Nello specifico, si andranno a rimuovere tutte le tuple con i campi "Benefits" e "BasePay" uguali a zero.

Dopo tutte le fasi di DataPreparation, siamo passati da 357.407 istanze a 320.631 istanze.

5.3 Normalizzazione

Siccome l'obiettivo è la fruizione di questi algoritmi ai neofiti in questo campo, ogni test viene eseguito 3 volte, ognuna con un tipo di normalizzazione diverso sui dati. Nel particolare sono utilizzati:

- Dati grezzi: dati così presenti nel dataset.
- MinMax: $x' = a + \frac{(x - min(x))(b-a)}{max(a) - min(x)}$
  Dove a e b rappresentano i valori minimo e massimo che vogliamo ottenere dalla normalizzazione.
- Z-score: $x' = \frac{x - \overline{x}}{\sigma}$
  Dove x è il valore originale, e $\overline{x}$ è la media della distribuzione e σ è la deviazione standard.

5.4 Codifica delle Variabili Categoriche

I regressori lavorano solo con dati numerici; Nel dataset sono presenti due colonne contenenti variabili di categoria: JobTitle e Status. Quando verrà avviata l'util `normilizer`, questa creerà la cartella /dataset, al cui interno troveremo il dataset normalizzato e un file txt. All'interno di quest'ultimo, è stato salvato un dizionario che permette l'interpretazione delle sostituzioni. Si sarebbe potuto optare per una OneHotEncoder, ma a seguito di verifiche sulla complessità spaziale e temporale, ho deciso intraprendere tale scelta. Nello specifico:

- **Complessità spaziale**:
  - **Senza OneHotEncoder**: Occupazione di memoria pari a 61.76MB
  - **Con OneHotEncoder**: Occupazione di memoria pari a 404.54MB

  Quindi abbiamo un incremento del 554.94%.
- **Complessità temporale**:
  - **Senza OneHotEncoder**:
    - Fit time: 0.07s
    - Score time: 0.01s
  - **Con OneHotEncoder**:
    - Fit time: 34.52s
    - Score time: 0.51s

  Anche qui risulta poco vantaggioso l'utilizzo della OneHotEncoder.
- **Metriche**:
  - **Senza OneHotEncoder**:
    - MAE: -3927.75
    - MSE: -30108726.02
    - RMSE: -5487.12
    - R^2: 0.86
  - **Con OneHotEncoder**:
    - MAE: -3823592.76
    - MSE: -3.21
    - RMSE: -283420230.44
    - R^2: -1401424228.77

  Anche in questo caso senza l'utilizzo della OneHotEncoding otteniamo risultati migliori.

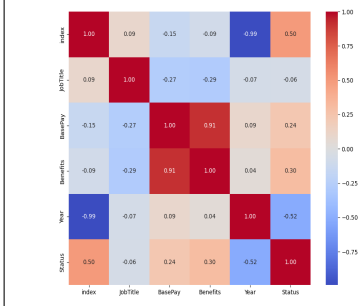
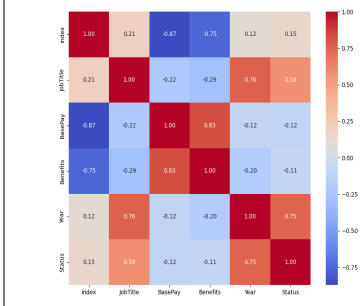
6 Modellazione

6.1 Selezione e analisi delle Caratteristiche

Idealmente la scelta più saggia sarebbe non mantenere tutte le colonne di cui il dataset dispone, ma utilizzare solo quelle con un alta correlazione con "Benefits". Nel nostro caso ho ritenuto che tutte le feature restanti potessero aggiungere informazioni preziose. Quindi l'addestramento avverrà sulle seguenti feature: JobTitle, BasePay, OvertimePay, OtherPay, Benefits, Year, Status.

6.1.1 Nuove Tabelle di Correlazione

Dopo la fase di pulizia, per una maggior accuratezza, sono state ri-create le tabelle di correlazione, per capire se fossero state estratte le feature giuste.

Senza	MinMax	Z-score
		

Nonostante "JobTitle" sembri avere una correlazione negativa, è stato volontariamente lasciato, in quanto si pensa sia influente. Potremmo però ipotizzare che questo dato intuitivamente illogico, derivi dal fatto che i "Benefits" vengono calcolati non in base al lavoro svolto, ma bensì in base al salario e alle condizioni economiche delle persone.

6.1.2 Riduzione Dimensionale

Vista la grande mole di dati presenti nel dataset, per poter comparare tutti i regressori, si è preso in considerazione il 2% del dataset, ovvero 7.148 istanze. Questa scelta è stata obbligata, in quanto l'utilizzo di più istante, portava la macchina sulla quale è stato eseguito il programma, ad un bottleneck, sia per la memoria RAM, dove si sono toccati picchi di 60GB richiesti, sia di CPU in quanto non tutti gli algoritmi sono di default parallelizzabili.

6.2 Scelta del Modello

Sono stati presi in esame tutti i modelli messi a disposizione dalla libreria scick-learn, di seguito viene fornita una lista.

6.2.1 Tipi di Modelli Considerati

- **LinearRegression**:
  È un modello di regressione lineare, che cerca di trovare la migliore retta di regressione per adattarsi ai dati.

- **LARS**:
  Prova la soluzione del modello di regressione lineare in modo incrementale, aggiungendo le variabili più influenti ad ogni passo.

- **Rige**:
  È una variante della regressione lineare, che introduce una regolarizzazione L2 per mitigare il rischio di overfitting.

- **LassoLars**:
  Utilizza il metodo LARS con regolarizzazione L1 (LASSO) per ottenere modelli di regressione sparsi.

- **ARDRegression - Automatic Relevance Determination**:
  Utilizza una combinazione di regolarizzazione L2 e L1 per determinare automaticamente la rilevanza delle variabili.

- **SGDRegressor**:
  Utilizza la discesa del gradiente stocastica, per addestrare modelli di regressione in modo efficiente su grandi set di dati.

- **BayesianRidge**:
  È un regressore basato sulla teoria bayesiana, incorpora conoscenze a priori nel processo di apprendimento.

- **GaussianProcessRegressor**:
  Utilizza processi gaussiani per modellare la distribuzione delle previsioni. È adatto per problemi di regressione non lineari.

- **TweedieRegressor**:
  Modella i dati con distribuzione di Tweedie, adatto per problemi di regressione con dati a dispersione.

- **DecisionTreeRegressor**:
  È un modello basato su alberi decisionali, che suddivide iterativamente i dati in base alle caratteristiche, predice il valore medio di ciascun nodo foglia.

- **RandomForestRegressor**:
  Una foresta di alberi decisionali che aggrega le previsioni di diversi alberi per ottenere una previsione più robusta.

- **KNeighborsRegressor**:
  Stima i valori target, basandosi sui vicini più prossimi in uno spazio delle feature.

- **RadiusNeighborsRegressor**:
  È simile a KNeighborsRegressor, ma stima i valori target basandosi su un raggio di vicini invece di un numero fisso.

- **SVR - Support Vectior Regression**:
  È una variante della regressione lineare che utilizza support vector machines per trovare la migliore retta di regressione.

- **LinearSVR**:
  È una versione lineare di SVR, utile per problemi di regressione lineare con grandi dataset.

- **NuSVR**:
  È una variante di SVR con un parametro aggiuntivo (nu) che controlla il numero di support vectors.

La scelta del regressore, dipende dalla natura del problema e dalle caratteristiche dei dati, oltre che dalle dimensioni. Ogni tipo di regressore ha vantaggi e svantaggi specifici e la scelta dovrebbe essere guidata dalla comprensione sia del contesto che degli obiettivi del problema di regressione che si sta affrontando.

7 Valutazione

7.1 Misurazione delle Prestazioni

7.1.1 Metriche di Valutazione Utilizzate

Per valutare la bontà di un regressore, prenderemo come metriche di riferimento:

- **MAE - Mean Absolute Error**:

$$\frac{\sum_{i=1}^{n} |\tilde{y} - y|}{n}$$

  La MAE è una metrica che misura la media assoluta degli errori tra le previsioni di un modello e i valori effettivi. È calcolata sommando le differenze assolute tra le previsioni e i valori reali e quindi, dividendo per il numero totale di osservazioni.
  Obiettivo: Minimizzare. Valori più bassi, indicano una maggiore precisione del modello.

- **MSE - Mean Squered Error**:

$$\frac{\sum_{i=1}^{n} (\tilde{y} - y)^2}{n}$$

  L'MSE è una metrica che misura la media dei quadrati degli errori tra le previsioni di un modello e i valori effettivi. È calcolata sommando le differenze al quadrato tra le previsioni e i valori reali, e quindi dividendo per il numero totale di osservazioni.
  Obiettivo: Minimizzare. Valori più bassi, indicano una maggiore precisione del modello.

- **RMSE - Root Mean Squered Error**:

$$\sqrt{\frac{\sum_{i=1}^{n} (\tilde{y} - y)^2}{n}}$$

  Il RMSE è la radice quadrata dell'MSE. Il RMSE fornisce una misura della dispersione degli errori di predizione.Valori più bassi di RMSE indicano una migliore adattabilità del modello ai dati, mentre valori più alti indicano una maggiore discrepanza tra le predizioni del modello e i valori effettivi.Può essere interpretato come una stima della precisione media delle predizioni del tuo modello, misurata in unità della variabile che stai cercando di prevedere.
  Obiettivo: Minimizzare. Valori più bassi, indicano una maggiore precisione del modello.

- R^2 **- Coefficient of determination**:

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

  Dove:

  – $$SS_{res} = \sum_{i=1}^{n} (y_i - \tilde{y_i})$$

  – $$\overline{y} = \frac{1}{n} - \sum_{i=1}^{n} y_i$$

  – $$SS_{tot} = \sum_{i}^{n} (y_i - \overline{y_i})^2$$

Misura la proporzione di varianza nella variabile dipendente che è spiegata dal modello. Può assumere valori compresi tra 0 e 1, dove 1 indica un perfetto adattamento del modello ai dati.
Obiettivo: Massimizzare. Un R^2 più alto, indica un modello più adatto, mentre un R^2 vicino a 0 indica che il modello non spiega bene la variabilità dei dati.

- **EVS - Explained variance score**:

$$1 - \frac{Var\{y - \overline{y}\}}{Var\{y\}}$$

L'EVS è una metrica che misura la proporzione della varianza totale della variabile dipendente che è spiegata dal modello. Come R^2, assume valori compresi tra 0 e 1.
Obiettivo: Massimizzare. Un punteggio più vicino a 1, indica una migliore capacità del modello di spiegare la varianza nei dati.

7.1.1.1 Assunzioni

Sebbene sia semplice e talvolta efficace, la regressione lineare può essere utilizzata solo in determinati contesti. Verranno riportati, inoltre, le soluzioni per valutare queste particolari condizioni.

- **Linearità dei dati**. La relazione tra la variabile indipendente X e la variabile dipendente Y deve essere lineare, ovvero può essere rappresentata tramite una funzione lineare.
  - Tabella di correlazione
- **Normalità dei residui**. Gli errori residui devono essere normalmente distribuiti.
  - Test Shapiro-Wilk
  - Test Kolmogorov-Smirnov
  - Test Anderson-Darling
  - Grafico distribuzione errori residui
- **Omoschedasticità**. Gli errori residui devono avere una varianza costante. Questa può essere verificata andando a plottare i residui standardizzati vs i valori predetti. Se la proprietà è soddisfatta, vedremo un trend orizzontale piuttosto che punti sparsi nello spazio.
  - Grafico variazione degli errori
- **Indipendenza degli errori**. Gli errori residui devono essere indipendenti per ogni valore di X. Un test statistico particolarmente utile è noto come Durbin-Watson: quando gli errori sono indipendenti, il valore del test sarà vicino a 2.
  - Test Durbin-Watson

7.2 Validazione incrociata:

La scelta di utilizzare la validazione incrociata, è stata motivata dalla necessità di ottenere una stima più affidabile delle prestazioni del modello. La validazione incrociata, fornisce una valutazione più robusta delle prestazioni del modello rispetto alla suddivisione tradizionale. Ho optato per una K-fold cross-validation con K=10, ripetuta 4 volte. Questa scelta è stata basata sulla dimensione complessiva del dataset e sulla necessità di bilanciare la varianza e la computazionalità.

8 Deploy e Implementazione

In questa sezione vedremo le scelte affrontate e le motivazioni dietro queste ultime, e come tutte le fasi precedenti, siano state poi sviluppate in codice e come esso sia stato strutturato.

8.1 Scelta del Linguaggio

La scelta di Python come linguaggio di programmazione per lo sviluppo di un regressore è motivata da diverse ragioni che ne fanno una delle opzioni preferite nell'ambito del machine learning e dell'analisi dei dati. Ecco alcune motivazioni chiave:

- **Ricca Libreria di Machine Learning**: Python dispone di librerie ampie e mature per il machine learning, tra cui scikit-learn, TensorFlow e PyTorch. Queste librerie offrono implementazioni efficienti di diversi algoritmi di regressione e forniscono strumenti per valutare, ottimizzare e validare i modelli.
- **Versatilità e Integrazione**: Python è un linguaggio versatile che può essere utilizzato in diverse fasi di un progetto, dalla manipolazione dei dati all'implementazione dei modelli e alla creazione di interfacce utente. La sua capacità di integrarsi facilmente con altri linguaggi e tecnologie, è un vantaggio significativo.
- **Ricchezza di Strumenti di Visualizzazione**: Python offre una vasta gamma di librerie di visualizzazione dei dati, come Matplotlib e Seaborn, che semplificano la rappresentazione grafica dei risultati del regressore. La visualizzazione è fondamentale per comprendere e comunicare efficacemente i risultati ottenuti.

In sintesi, la combinazione di una vasta libreria di machine learning, una comunità attiva, facilità di apprendimento e versatilità, fa di Python una scelta solida e popolare per lo sviluppo di regressori e modelli di machine learning in generale.

8.2 Framework e Strumenti Utilizzati

Tra le varie tecnologie e framework precedemente citati, si è scelto per l'implementazione dell'agente la libreria `scikit-learn`. Per la data-visualization invece si è ricorso alla libreria `Matplotlib` e `Seaborn`. Per l'utilizzo del dataset sottoforma di dataframe in modo facile e intuitivo, si è utilizzato `pandas`. Per la creazione delle statistiche, si è ricorso alla libreria `statsmodels`. Per la creazione della GUI si è ricorso alla libreria `appJar`.

8.2.1 Dipendenze necessarie

Appurate tutte le dipendenze del progetto, sarebbe utile in vista di utilizzare tale progetto, risolvere le seguenti dipendenze, di seguito sono riportate tutti gli snippet per installare le librerie necessarie:

- **Pandas:**
  `pip install pandas`
- **Joblib:**
  `pip install joblib`
- **Scikit-learn:**
  `pip install scikit-learn`
- **Matplotlib:**
  `pip install matplotlib`
- **Seaborn:**
  `pip install seaborn`
- **Statsmodels:**
  `pip install statsmodels`

- **appJar:**
  `pip install appjar`
- **PIL:**
  `pip install PIL`
- **Scipy:**
  `pip install scipy`
- **Mlxtend:**
  `pip install mlxtend`

8.2.2 Link utili

Di seguito verrà riportata una breve lista di link che posso essere un ottimo spunto per approfondire le varie tematiche in modo singolo:

- **GitHub:**
  Link al progetto
- **Blocco note Collab:**
  Link al notepad
- **Kaggle:**
  Link al dataset
- **Scikit-learn:**
  Link alla libereria Scikit-learn
- **Pandas:**
  Link alla libreria Pandas
- **Matplotlib:**
  Link alla libreria Matolotlib
- **Statsmodels:**
  Link alla libreria Statsmodels
- **Seaborn:**
  Link alla libreria seaborn
- **Joblib:**
  Link alla libreria joblib
- **appJar:**
  Link alla libreria appJar
- **PIL:**
  Link alla libreria Pillow
- **Scipy:**
  Link alla libreria Scipy
- **Mlxtend:**
  Link alla libreria Mlxtend

8.3 QuickStart progetto - Primo avvio

In questa sezione, verrà spiegato come il progetto funziona logicamente astraendo tutta la parte di programmazione, che verrà sviluppata nel modulo successivo. Dopo aver scaricato il progetto e risolto tutte le dipendenze, si procede con l'avvio del `normalizer.py`, presente nella cartella /util. Il compito di tale classe, oltre a rimuovere la colonna "Employee Name" per una questione di ottimizzazione, è quello di effettuare la sostituzione per i campi di tipo stringa presenti nel dataset. Questo perché i regressori accettano solo numeri come input. Questo modulo come output produrrà due nuovi file "newDataset.csv" e "indexSostitution.txt" entrambi nella cartella /dataset. Dove:

- newDataset: contiene il nuovo dataset, privo di campi stringa.
- indexSostitution: è un dizionario contenente per ogni valore di ogni colonna, il campo sostituito con il relativo indice, così da poter ricostruire il dataset originale eventualmente.

Una volta termina l'esecuzione del `normalizer.py`, possiamo avviare il `main.py`. Qui avremo la possibilità di utilizzare 2 modalità di utilizzo dell'Agent Farm:

- **Automatica**: Il modulo prevederà ad effettuare in automatico le fasi di:
  - Pulizia Outlier: Questo avverrà rimuovendo le tuple presenti nel primo e quarto quartile, in quanto non costituiscono informazioni rilevanti, ma bensì confondono il modello.
  - Feature Scaling: Per ogni feature andiamo a effettuare l'analisi dei curti oltre a valutare la sua distribuzione, per capire se applicare o meno una normalizzazione.
  - Feature selection: Attraverlo l'utilizzo del metodo ExhaustiveFeatureSelector andiamo a prendere le feature che per un determinato modello aumentano la metrica r^2.
  - Data cleaning: Sulle feature restituite da ExhaustiveFeatureSelector andiamo a prendere solo le tuple con valori non uguali a zero.
- **Manuale**: Qui verranno fatte tutte le fasi di datacleaning, featurescaling, featureSelection seguendo le indicazioni del utente.

Questo servirà per effettuare la comparazione tra i vari tipi di regressore; questo viene iterato tre volte, una per ogni tipo di normalizzazione. Al termine dell'esecuzione, si sarà creata una nuova cartella /analysis, nella quale troveremo una cartella per ogni algoritmo, all'interno della quale troveremo un'altra cartella per ogni normalizzazione all'interno della quale avremo: il grafico della distribuzione dell'errore, il grafico per la variazione dell'errore e un report nel quale troviamo tutte le metriche al suo interno. Al termine del `main`, partirà in automatico l'interfaccia grafica per facilitare la valutazione dei risultati.

8.4 Struttura del progetto

Il progetto è stato creato affinché sia facilmente adattabile a più casistiche possibile. Ovviamente andranno modificati i nomi delle colonne presenti sia nel `normalizer.py` e sia `main.py`. Di seguito verrà approfondito singolarmente ogni modulo per consentire a chiunque, una profonda comprensione degli stessi e l'aumento delle possibilità di riutilizzo e ampliamento del codice da me prodotto.

8.5 Normalizer

Suddividiamo il modulo in 4 blocchi principali:

```
## trasformo il vecchio dataset in una matrice
dataframe=pd.read_csv("./dataset/san-francisco-payroll_2011-2019.csv")
dataframe=dataframe.drop(columns=["Employee Name"])
#Sostituiamo i valori Nan con NotProvided per evitare di avere problemi con il dizionario
print("Number of istance Nan :" + str(dataframe.isna().sum().sum()))

print("Number of istance Nan in tutto il dataset:" + str(dataframe.isna().sum().sum()))
print("Number of istance Nan nella colonna 'Status' :" + str(dataframe["Status"].isna().sum()))
dataframe=dataframe.replace(np.nan, value: "Not Present",regex=True)
print("Number of istance Nan in tutto il dataset:" + str(dataframe.isna().sum().sum()))
print("Number of istance Nan nella colonna 'Status' :" + str(dataframe["Status"].isna().sum()))

print("Number of row: "+str(len(dataframe.index)))
print("Number on 'Not Provided': "+str(dataframe[dataframe == 'Not Provided'].count()))

print("Number of row before cleaning: "+str(len(dataframe.index)))
dataframe=dataframe.replace( to_replace: "Not Provided",np.nan,regex=True)
print("Number of istance Nan in tutto il dataset before cleaning:" + str(dataframe.isna().sum().sum()))
dataframe=dataframe.dropna()
print("Number of istance Nan in tutto il dataset after cleaning:" + str(dataframe.isna().sum().sum()))
print("Number of row after cleaning: "+str(len(dataframe.index)))
```

Figure 1: Primo blocco

In questo primo blocco, con l'aiuto di Pandas, viene caricato il dataframe e lo si converte in una matrice. Dopo di che eliminiamo tutti i campo "Nan" e "Not Provided" in modo da ripulire il dataset.

```
## inizializzo un dizionario che mi indicarà per ogni categoria l'ultimo l indice inserito
## inizializzo il dizinario che conterra un dizionario per ogni catergoria dove c'è una stringa
for i in range(len(data[0])):
    if type(data[1][i]) is str:
        if not data[1][i].replace(",","",1).replace(".","",1).isnumeric() or not data[1][i].replace(",","",1).replace(".",
            last[data[0][i]] = 0
            listaDiSostiturioni[data[0][i]] = dict()
```

Figure 2: Secondo blocco

In questa secondo blocco, creiamo il dizionario delle sostituzioni e inizializziamo un dizionario dove, per ogni colonna che verrà sostituita, avremo l'ultimo indice utilizzato.

```
for i in range(1,len(data)):
    for j in range(0,len(data[i])):
        if data[0][j] in listaDiSostiturioni.keys():
            if str(data[i][j]).lower() not in listaDiSostiturioni[data[0][j]].keys():
                listaDiSostiturioni[data[0][j]][data[i][j].lower()]=last[data[0][j]]+1
                last[data[0][j]]+=1
            data[i][j]=listaDiSostiturioni[data[0][j]][data[i][j].lower()]
        else:
            if data[i][j]=="Not Provided":
                data[i][j]=0
            else:
                if "," in str(data[i][j]) or "." in str(data[i][j]):
                    # print("5")
                    data[i][j] = float(data[i][j])
                else:
                    # print("6")
                    data[i][j] = int(data[i][j])
```

Figure 3: Terzo blocco

In questo terzo blocco, per ogni riga della tabella, controlliamo se la colonna della cella in esame è presente nel dizionario delle sostituzioni, in caso positivo andiamo a verificare se è un valore sostituito

in precedenza o invece è un nuovo valore da aggiungere; in caso negativo capiamo se il valore è un intero o un decimale e lo salviamo.

```
try:
    os.mkdir("../dataset")
except OSError as e:
    pass

newFile=open("../dataset/newDataset.csv","w")

for i in range(0,len(data)):
    newFile.write(str(data[i]).removesuffix("]").removeprefix("[").replace( __old: " ", __new: "").replace( __old: "'", __new: "
newFile.close()


indexSostitution = open("../dataset/indexSostitution.txt","w")

for key in listaDiSostiturioni.keys():
    indexSostitution.write(key+str(listaDiSostiturioni[key])+"\n")
indexSostitution.close()
```

Figure 4: Quarto blocco

In quest'ultimo blocco, creiamo la cartella dove andremo a salvare i due file: newDataset e indexSostitution.

8.6 Agente

Il modulo `Agent.py` implementa una serie di metodi dei quali verrà fornita di seguito la firma e una descrizione:

- `Agent(type:str,n_job:int,randState:int)`
  Questo metodo prende in input il tipo di regressore che l'agente deve utilizzare e il numero di thread che il sistema può mettere a disposizione per aumentare le prestazioni, se possibile. In oltre viene passato il randomState affinché la RepeatedKFold avvenga sulla stessa suddivisione del dataset.

- `fit(X_train,y_train)`
  È un metodo con il quale l'agente effettua l'addestramento, prendendo in input le variabili indipendenti X e la variabile dipendete y.

- `predict(X_test)`
  In questo metodo l'agente, sulla base di quanto ha imparato, predice il risultato della variabile dipendete y.

- `valuation(y_test, pred)`
  In questo metodo, sulla base dei valori predetti, vengono restituite le metriche che sono state indicate in precedenza.

- `cross_validation(X_train, y_train)`
  In questo metodo, effettuiamo una kcorss validation 4 volte, per poi ritornare le metriche indicate in precedenza.

8.7 AgentFarm

Il modulo `AgentFarm.py` implementa una serie di metodi dei quali è fornita di seguito la firma e una descrizione:

- `AgentFarm(dataframe:DataFrame,n_job:int,target:str,mode:str,outlier:bool,rangeOutlier:float):`
  Al costruttore passiamo:
  - dataframe: il dataframe che intendiamo utilizzare.
  - n_job: numero di thread che il programma potrà utilizzare.
  - target: il nome della variabile target.
  - mode: la modalità di utilizzo, ovvero se in modalità "automatica" o "manuale".
  - outlier: il flag che indica se eliminare o meno gli outlier
  - rangeOutlier: il moltiplicatore che indica il raggio entro il quale tenere o eliminare i dati durate la pulizia degli outlier. Maggiore è il valore maggiori saranno i valori rimossi e viceversa.
- `dataCleaning(listaRimossi:list,valueTarget:int):`
  In questo metodo eliminiamo, tutte le tuple che hanno i campi presenti in listaRimossi uguali al valore target.
- `correlazioneVariabili(lable:str):`
  In questo metodo, creiamo il grafico di correlazione delle variabili nella cartella /analysis/TabellaDiCorrelazione. Il campo lable è formattato come normalizzazione_value, dove value è "before" o "after" per indicare se è stato chiamato prima o dopo la featureSelection.
- `distriubuzioneFeateure(lable:str)`
  Questo metodo, salverà il grafico della distribuzione dei valori delle feature con il nome lable.png.
- `featureScaling(normalizazione:str,includeTarget:bool=False):`
  Questo metodo effettuerà a seconda della modalità diverse operazioni:
  - **Automatica**: Attraverso l'analisi delle simmetrie e dei curtosi per ogni feature valuterà se normalizzare o meno le varie feature; in caso positivo applicherà la normalizzazione passata per input.
  - **Manuale**: In questa modalità a seconda della stringa passata, effettuerà o meno la normalizzazione sui dati. L'utilizzo del flag permetterà di includere o meno la variabile target nella normalizzazione.
- `featureSelection(listaRimossi:list):`
  In questo metodo, eliminiamo le colonne presenti in listaRimossi, di solito le feature con poca varianza.
- `initComparison(percentageTest:float):`
  In questo metodo dividiamo il dataframe in variabile dipendente e indipendenti, oltre alla divisione in test e train a seconda della `precenpercentageTest`.
- `startComparison():`
  Per ognuno dei regressori presenti in `listaAgenti`, eseguiamo le fasi di fit, prediction, valuation e cross_validation richiamando il metodo `singleComparison()`. Questo metodo è utilizzato in modalità "manuale".
- `singleComparison(NameAgent:str)`: Esegue le fasi di fit, prediction,valuation e cross_validation per un singolo agente passato per input. Questo metodo produce i seguenti file:
  - **distribuzioneErroreResidio**: Dove vediamo graficamente come è distribuito l'errore residuo.
  - **varianzaErroreResiduo**:Dove vediamo graficamente la differenza tra l'errore residuo standardizzato e i valori previsti.

  - **Report.txt**: In questo file sono riportati: i valori per le metriche standard, quelle ottenute con la K cross validation e i test statistici per capire:
    * **Normalità del errore residuo**: vengono effettuati i test di: Shapiro-Wilk, Kolmogorov-Smirnov e di Anderson-Darling.
    * **Indipendeza degli errori residui**: viene eseguito il test di Durbin-Watson.

- `autoDataPreparation(nameAgent:str)`:
  Questo metodo prende la variabili dipendente e le variabili indipendenti e attraverso il metodo `ExhaustiveFeatureSelector()` ottiene quelle che posso essere le feature migliori per l'addestramento. Restituisce la lista con le feature selezionate e quelle da rimuovere.

- `autoCompariosn(typeNorm:str,percentageTest:float)`
  Per ognuno dei regressori presenti in `listaAgenti`, eseguiamo le fasi di feature scaling, feature selection e data cleaning grazie l'utilizzo di `autoDataPreparation()`. Dopo eseguirà le fasi di fit, prediction, valuation e cross_validation richiamando il metodo `singleComparison()`. Questo metodo è utilizzato in modalità "automatica".

- `start(listaRimossi:list,listaCleaning:list,percentageTest:float)`:
  In questo metodo per ogni normalizzazione si compiranno scelte diverse a seconda della modalità:
  - **Automatica**: chiama il metodo `autoCompariosn()`.
  - **Manuale**: eseguirà le fasi di feature scaling, feature selection e data cleaning per poi richiamare `startComparison()`.

8.8 Main

Questa sezione è divisa in due parti:

```
path="./dataset"
file_name="newDataset.csv"



#prepariamo il data frame da pandas
dataframe=pd.read_csv(path+"/"+file_name)
#print(dataframe.info(memory_usage='deep'))
print("Numero di istanze prima dell'partizionamento :"+str(len(dataframe.index)))
dataframe = dataframe.sample(frac=0.02, random_state=42)
print("Numero di istanze dopo dell'partizionamento :"+str(len(dataframe.index)))



listaRimossi=list({"TotalPay","TotalPay&Benefits"})
listaCleaning=["BasePay","Benefits"]
n_job=8
```

Figure 5: Prima parte

In questa prima parte, preleviamo il dataset normalizzato e lo riduciamo di size per poi creare tutte le liste di cui abbiamo bisogno.

```
farm = AgentFarm(dataframe,n_job, target: "Benefits",mode="Manual",outlier=True)
farm.start(listaRimossi,listaCleaning, percentageTest: 0.33)


os.system("python3 gui.py")
```

Figure 6: Seconda parte

In quest'ultima, creiamo l'agentFarm, per poi avviarla e aspettare i risultati, i quali ci verranno mostrati attraverso la GUI.

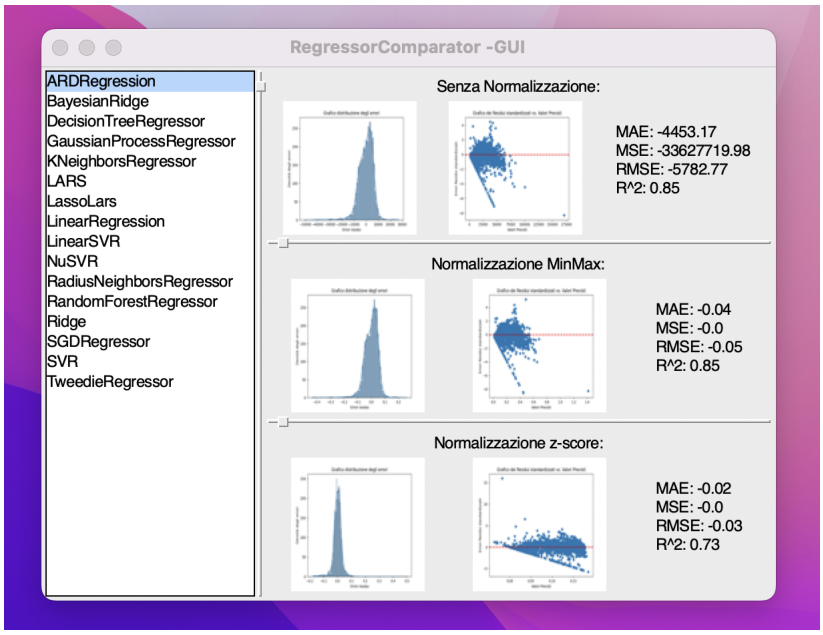
8.9 GUI

Questa sezione spiega come è stata affrontata la progettazione della GUI - Graphical user interface. Essendo stata sviluppata dopo il sistema, questa deve essere indipendente dallo stesso, ovvero poter essere chiamata sia subito dopo l'esecuzione del `main.py` che in maniera estemporanea. Vista la grande quantità di codice, verranno di seguito indicate le classi presenti nel modulo `gui.py` con i relativi metodi:

- `Regressore`
  Questa classe ha il compito di memorizzare in maniera ordinata tutte le informazioni di un singolo regressore, ovvero:
  - Nome: attributo di tipo string.
  - Metriche: attributo di tipo dizionario, dove sono conservate per ogni tipo di normalizzazioni le metriche.
  - Grafici: sotto forma di immagini.
- `Scanner`
  Questa classe ha il compito di prelevare le informazioni dalla cartella ./analysis e incanalare le informazioni sotto forma di istanze della classe precedentemente citata. Questo attraverso un metodo:
  - `scan()`: Questo ritorna una lista di Regressori.
- `Gui`:
  Nelle ultime 100 righe sono condensate le funzioni effettive per la parte di GUI, ovvero:
  - `retuntString(dict)`:
    Prende in input un dizionario e restituisce una stringa formattata per essere stampata.
  - `update(value)`:
    Per aggiornare i grafici ogni qual volta si seleziona un regressore diverso.
  - `press(btn,regressor)`:
    Implementazione delle funzione di Zoom in delle immagine che la libreria non fornisce.

  Le ultime 50 righe sono la creazione della finestra con l'inizializzazione dei vari grafici e metriche.

Si ottiene alla fine, il seguente risultato.



9 Conclusioni e Pianificazione Futura

9.1 Risultati

I questa sezione commentiamo i dati ottenuti dall'esecuzione del file `main.py`. Inizialmente si volevano riportare in formato tabellare i grafici risultanti dall'esecuzione, ma questo risultava essere particolarmente scomodo e poco pratico. I risultati sono stati ottenuti in modalità "manuale", tra i parametri della configurazione utilizzata, vorrei porre l'attenzione sul fatto che la variabile target **è stata inclusa** nella normalizzazione e sui risvolti che questo ha portato.

Dopo varie esecuzioni, ho potuto osservare che la normalizzazione della target tende a diminuire, se non a cancellare drasticamente, le metriche: MAE, MSE, RMSE. Quando invece non si normalizza la variabile target le metriche MAE, MSE e RMSE sono ancora un valido strumento di valutazione. Inoltre, come possiamo vedere, il tipo di normalizzazione influisce molto sia sulla distribuzione degli errori che sulla varianza, quindi può portare sia migliorie che peggioramenti al modello. Va notato che non è stato possibile utilizzare l'algoritmo Radius Neightbors senza applicare alcun tipo di normalizzazione dei dati, in quanto non è stato in grado di individuare dei vicini. Per valutare complessivamente un algoritmo, utilizziamo i risultati normalizzando anche la variabile target, di conseguenza possiamo prendere l'R^2 come metrica unica. La tabella dove sono riportate le metriche per ogni algoritmo si trova in calce al documento.

9.2 Successi e Sfide

Indubbiamente, la parte difficile è stata progettare un sistema che si adatti a tutti i possibili regressori presenti, e organizzare i risultati di tali comparazioni affinché siano fruibili e comprensibili a più persone possibili. È stata possibile implementare questa accessibilità sia grazie all'aggiunta di un interfaccia grafica, sia grazie alla modalità "Auto", che amplia notevolmente la platea di utenti capaci di interagire con il sistema, completando definitivamente l'obiettivo del progetto.
Come risultati del caso d'uso preso in esame, vediamo i diversi regressori che hanno superato i criteri di successo, ovvero:

- **GaussianProcessRegressor** con normalizzazione z-score.
- **KNeighborsRegressor** senza normalizzazione e con z-score.
- **RandomForestRegressor** con tutte le normalizzazioni.
- **NUSVR** con la normalizzazione z-score e MinMax.

Un risultato inatteso sul quale voglio porre l'attenzione, è che includendo o meno la variabile target nella normalizzazione, alcuni modelli migliorano drasticamente. Un esempio è il **NUSVR** il quale senza normalizzazione della variabile target nel caso migliore ha un r^2 di 0.1, invece, normalizzando la target questo valore schizza a 0.99.

9.3 Sviluppi Futuri e Miglioramenti Possibili

Premettendo che è un'ottima base per analisi di regressori, sarebbe sicuramente possibile comprendere tra le metriche anche il "Fit time mean" e il "Score time mean", che nel report sono stati indicati ma non riportati nelle tabelle. Sarebbe **ottimale** che il programma **fornisse un algoritmo migliore**, senza lasciare interpretare i dati all'utente. Idealmente potremmo restituire il migliore per ogni metrica. Inoltre potremmo valutare l'uso di librerie per rendere interattivi i grafici o più facilmente interpretabili e quindi migliorare la data-visualization. In aggiunta si potrebbe pensare ad un miglioramento dell'interfaccia grafica, rendendola più accattivante, ma anche di spostare totalmente le interazioni dell'utente sull'interfaccia grafica, ovvero dalla scelta del dataset, alla parte di dataCleaning, featureScaling e featureSelection rendendolo totalmente accessibile a tutti.

		valutation	k-cross valutation
LinearRegression	non	MAE:4414.02 MSE:34240294.66 RMSE:5851.52 R2:0.85	MAE:-4584.90 MSE:-36363118.70 RMSE:-6012.39 R2:0.84
	minmax	MAE:0.04 MSE:0.00 RMSE:0.05 R2:0.85	MAE:-0.03 MSE:-0.00 RMSE:-0.04 R2:0.83
	z_score	MAE:0.02 MSE:0.00 RMSE:0.03 R2:0.70	MAE:-0.02 MSE:-0.00 RMSE:-0.029 R2:0.71
LARS	non	MAE:4414.02 MSE:34240294.66 RMSE:5851.52 R2:0.85	MAE:-4585.85 MSE:-36388949.97 RMSE:-6019.91 R2:0.84
	minmax	MAE:0.04 MSE:0.00 RMSE:0.05 R2:0.85	MAE:-0.03 MSE:-0.00 RMSE:-0.04 R2:0.83
	z_score	MAE:0.02 MSE:0.00 RMSE:0.03 R2:0.69	MAE:-0.02 MSE:-0.00 RMSE:-0.03 R2:0.70
Rige	non	MAE:4414.06 MSE:34240082.66 RMSE:5851.50 R2:0.85	MAE:-4588.17 MSE:-36421552.08 RMSE:-6013.55 R2:0.84
	minmax	MAE:0.04 MSE:0.00 RMSE:0.05 R2:0.85	MAE:-0.03 MSE:-0.00 RMSE:-0.04 R2:0.83
	z_score	MAE:0.02 MSE:0.00 RMSE:0.03 R2:0.68	MAE:-0.02 MSE:-0.00 RMSE:-0.03 R2:0.68

		valutation	k-cross valutation
LassoLars	non	MAE: 4414.19 MSE:34239402.57 RMSE:5851.44 R2:0.85	MAE:-4585.45 MSE:-36386719.94 RMSE:-6010.52 R2:0.84
	minmax	MAE:0.10 MSE:0.02 RMSE:0.13 R2:0.00	MAE:-0.09 MSE:-0.01 RMSE:-0.12 R2:0.00
	z_score	MAE:0.04 MSE:0.00 RMSE:0.06 R2:0.00	MAE:-0.04 MSE:-0.00 RMSE:-0.05 R2:-0.00
ARDRegression	non	MAE:4433.12 MSE:34265342.43 RMSE:5853.66 R2:0.85	MAE:-4600.41 MSE: -36423938.17 RMSE:-6020.70 R2:0.84
	minmax	MAE:0.04 MSE:0.00 RMSE:0.05 R2:0.85	MAE:-0.03 MSE:-0.00 RMSE:-0.04 R2:0.84
	z_score	MAE:0.02 MSE:0.00 RMSE:0.03 R2:0.70	MAE:-0.02 MSE:-0.00 RMSE:-0.02 R2:0.71
SGDRegressor	non	MAE:2276259820429790720 MSE:78064002507115000915555032702690263 04 RMSE:2793993602482206720 R2:-34985771752266058653400825856	MAE:-7.56 MSE:-1.15 RMSE:-8.86 R2:-4.89
	minmax	MAE:0.08 MSE:0.01 RMSE:0.10 R2:0.37	MAE:-0.07 MSE:-0.00 RMSE:-0.09 R2:0.37
	z_score	MAE:0.03 MSE:0.00 RMSE:0.03 R2:0.62	MAE:-0.02 MSE:-0.00 RMSE:-0.03 R2:0.61

		valutation	k-cross valutation
BayesianRidge	non	MAE:4501.46 MSE:35469603.31 RMSE:5955.64 R2:0.84	MAE:-4653.39 MSE:-37494271.93 RMSE:-6106.87 R2:0.83
	minmax	MAE:0.04 MSE:0.00 RMSE:0.05 R2:0.85	MAE:-0.03 MSE:-0.00 RMSE:-0.04 R2:0.83
	z_score	MAE:0.02 MSE:0.00 RMSE:0.03 R2:0.69	MAE:-0.02 MSE:-0.00 RMSE:-0.03 R2:0.69
GaussianProcessRegressor	non	MAE:25742.20 MSE:885890338.39 RMSE:29763.91 R2:-2.97	MAE:-26203.35 MSE:-918970542.13 RMSE:-30305.83 R2:-2.98
	minmax	MAE:0.08 MSE:1.34 RMSE:1.16 R2:-82.24	MAE:-0.08 MSE:-1.14 RMSE:-0.65 R2:-73.65
	z_score	MAE:0.00 MSE:0.00 RMSE:0.01 R2:0.99	MAE:-0.00 MSE:-4.62 RMSE:-0.00 R2:0.98
TweedieRegressor	non	MAE:4456.86 MSE:34855324.49 RMSE:5903.84 R2:0.84	MAE:-4617.30 MSE:-36976462.38 RMSE:-6064.61 R2:0.83
	minmax	MAE:0.10 MSE:0.02 RMSE:0.12 R2:0.05	MAE:-0.09 MSE:-0.01 RMSE:-0.12 R2:0.04
	z_score	MAE:0.04 MSE:0.00 RMSE:0.05 R2:0.11	MAE:-0.04 MSE:-0.00 RMSE:-0.05 R2:0.10

		valutation	k-cross valutation
DecisionTreeRegressor	non	MAE:2767.09 MSE:24070002.09 RMSE:4906.12 R2:0.89	MAE:-2821.83 MSE:-25113669.21 RMSE:-4990.13 R2:0.89
	minmax	MAE:0.02 MSE:0.00 RMSE:0.04 R2:0.89	MAE:-0.02 MSE:-0.00 RMSE:-0.04 R2:0.87
	z_score	MAE:0.01 MSE:0.00 RMSE:0.02 R2:0.93	MAE:-0.01 MSE:-0.00 RMSE:-0.02 R2:0.92
RandomForestRegressor	non	MAE:2107.78 MSE:14793332.88 RMSE:3846.21 R2:0.93	MAE:-2179.3 MSE:-14564856.99 RMSE:-3793.43 R2:0.93
	minmax	MAE:0.02 MSE:0.00 RMSE:0.03 R2:0.94	MAE:-0.01 MSE:-0.00 RMSE:-0.03 R2:0.93
	z_score	MAE:0.01 MSE:0.00 RMSE:0.02 R2:0.91	MAE:-0.00 MSE:-0.00 RMSE:-0.01 R2:0.90
KNeighborsRegressor	non	MAE:2481.89 MSE:20743583.79 RMSE:4554.51 R2:0.91	MAE:-2600.17 MSE:-21903678.75 RMSE:-4654.92 R2:0.90
	minmax	MAE:0.03 MSE:0.00 RMSE:0.04 R2:0.89	MAE:-0.02 MSE:-0.00 RMSE:-0.04 R2:0.86
	z_score	MAE:0.01 MSE:0.00 RMSE:0.02 R2:0.91	MAE:-0.01 MSE:-0.00 RMSE:-0.01 R2:0.89

		valutation	k-cross valutation
RadiusNeighborsRegressor	non	None	None
	minmax	MAE:0.10 MSE:0.02 RMSE:0.12 R2:0.04	MAE:-0.09 MSE:-0.01 RMSE:-0.12 R2:0.05
	z_score	MAE:0.04 MSE:0.00 RMSE:0.05 R2:0.07	MAE:-0.04 MSE:-0.00 RMSE:-0.05 R2:0.06
SVR	non	MAE:11467.26 MSE:231431412.85 RMSE:15212.87 R2:-0.04	MAE:-11703.46 MSE:-238188649.62 RMSE:-15422.92 R2:-0.02
	minmax	MAE:0.04 MSE:0.00 RMSE:0.05 R2:0.82	MAE:-0.04 MSE:-0.00 RMSE:-0.05 R2:0.81
	z_score	MAE:0.04 MSE:0.00 RMSE:0.05 R2:0.25	MAE:-0.03 MSE:-0.00 RMSE:-0.04 R2:0.23
LinearSVR	non	MAE:10021.96 MSE:136470337.52 RMSE:11682.05 R2:0.39	MAE:-7500.49 MSE:-107987723.58 RMSE:-9635.48 R2:0.53
	minmax	MAE:0.04 MSE:0.00 RMSE:0.05 R2:0.84	MAE:-0.03 MSE:-0.00 RMSE:-0.05 R2:0.83
	z_score	MAE:0.02 MSE:0.00 RMSE:0.03 R2:0.67	MAE:-0.02 MSE:-0.00 RMSE:-0.03 R2:0.68

		valutation	k-cross valutation
NuSVR	non	MAE:11912.50 MSE:219871264.12 RMSE:14828.06 R2:0.01	MAE:-12296.28 MSE:-229898093.02 RMSE:-15153.89 R2:0.00
	minmax	MAE:0.02 MSE:0.00 RMSE:0.04 R2:0.92	MAE:-0.02 MSE: -0.00 RMSE:-0.03 R2:0.89
	z_score	MAE:0.00 MSE:0.00 RMSE:0.01 R2:0.99	MAE:-0.00 MSE:-3.70 RMSE:-0.00 R2:0.98