UNIVERSITÀ DI SALERNO

DIPARTIMENTO DI INFORMATICA

MetaClassAI

Regressore per la stima della durata ideale dei meeting

Repository GitHub:

https://github.com/smike18181/MetaClass_FIA

https://github.com/Everysimo/MetaClass

Compilato da:

Pesce Michele

Alberti Salvatore

Gatto Francesco

Cavaliere Domenico

La stesura di questo documento si basa su una varietà di fonti accademiche e professionali per sviluppare le argomentazioni e le conclusioni trattate. Di seguito sono elencate alcune delle principali fonti consultate:

- *Russell e Norvig 2009*
- *Géron 2019*
- *Burkov 2020*
- *Wikipedia contributors 2024*

Queste fonti hanno fornito una solida base di conoscenze e hanno aiutato a comprendere meglio il contesto e le sfide relative all'argomento di ricerca.

Dettagli sui riferimenti bibliografici si trovano in fondo al documento.

Indice

1 Definizione del Problema

1.1 Obiettivi

L'obiettivo del progetto è quello di realizzare un agente intelligente in grado di **stimare la durata media consigliata** per un meeting all'interno di una stanza quando questo viene schedulato.

1.2 Specifica PEAS

- **Performance**: La misura di performance dell'agente è la precisione con cui si avvicina alla reale durata media del meeting quando questa viene proposta in fase di programmazione;
- **Environment**: L'ambiente in cui opera l'agente è l'insieme dei feedback degli utenti a fine meeting del nostro applicativo;
- **Actuators**:Gli attuatori a disposizione dell'agente sono le caselle di input del range orario, che presenta già l'offset basato sulla stima fatta sulla fascia oraria proposta quando si va a schedulare un meeting sul sito `https://metaclass.commigo.it`;
- **Sensors**: I sensori tramite il quale l'agente recepisce stimoli dall'ambiente sono i report compilabili sul sito `https://metaclass.commigo.it` e sul tempo di utilizzo del visore per singolo utente all'interno di un meeting;

1.2.1 Caratteristche dell'ambiente

- **Completamente osservabile**: l'agente ha sempre a disposizione tutte le informazioni relative al report degli utenti e agli altri dati raccolti;
- **Stocastico**: lo stato successivo dell'ambiente dipende non solo dallo stato corrente e dall'azione dell'agente ma anche da una componente indipendente dall'agente;
- **Sequenziale**: la stima della durata è fortemente influenzata dalle durate stimate precedenti;
- **Dinamico**: durante l'esecuzione dell'agente gli utenti potrebbero sottomettere nuovi report;

- **Discreto**: ambiente discreto in quanto il numero di caratteristiche analizzate è finito e non varia in base alle condizioni;
- **Singolo agente**: l'ambiente presenta un unico agente che opera su di esso;

1.3 Analisi del problema

Una soluzione al problema inerente sarebbe potuta essere sviluppata mediante una semplice media della durata di tutti i meeting svolti dagli utenti, tuttavia tale soluzione avrebbe comportato diverse limitazioni:

- questa implementazione **non tiene conto di caratteristiche specifiche** dell'utente, come ad esempio una fascia d'età più bassa che può essere maggiormente predisposta ad una durata più lunga di utilizzo del visore;
- non ci sarebbe stato un accesso immediato ad una quantità di dati adeguata per una stima più accurata;
- l'output risultante potrebbe risultare **impreciso** a causa di una quantità di dati limitati (dati dei partecipanti in quella determinata stanza);

Date queste limitazioni, si è deciso di affrontare il problema in esame adottando tecniche di Machine Learning in grado di effettuare analizzare caratteristiche comuni dei partecipanti ai meeting (e.g. età, sesso, etc.) per effettuare una stima accurata della durata di un futuro meeting da schedulare.
Sono stati presi in analisi diversi algoritmi di Machine Learning, la cui scelta è infine ricaduta su di un problema di **regressione non lineare**. (Leggere dal paragrafo 4.1 per ulteriori dettagli).

2 Dataset

2.1 Scelta del dataset

L'acquisizione dei dati rappresenta il primo fondamentale passo nel processo di sviluppo di un modello di intelligenza artificiale. Questo processo consiste nella raccolta, nell'organizzazione e nella preparazione dei dati necessari per addestrare e valutare il modello.
Dopo un'accurata fase di ricerca, si è proceduto con la creazione del dataset necessario per il modello di intelligenza artificiale.
Tale dataset è stato ottenuto tramite la **modellazione** e **l'elaborazione** di dati preesistenti, i quali sono stati ricavati dal sito **kaggle.com**.
In particolare, il dataset scelto è stato il seguente: **Virtual Reality Experiences**.
Questo dataset, reso disponibile con licenza pubblica, consiste in un insieme di **1000 istanze** ricavate dalle esperienze di utenti all'interno di ambienti di realtà virtuale (VR), comprendendo informazioni sulle risposte fisiologiche e emotive degli utilizzatori del sistema.

2.2 Analisi Dataset e ei dati al suo interno

Le fonti del dataset sono costituite principalmente da informazioni ricavate da studi effettuati in laboratorio. **il Dataset si compone delle seguenti sette caratteristiche:**

- **User ID**

  Questa Caretteristica assegna un ID distinto a ciascun utente per differenziare i propri dati nel set di dati.

- **Age**

  Rappresenta l'età dell'utente al momento dell'esperienza VR è rappresentato da un valore intero.

- **Gender**

  Questa variabile indica il sesso dell'utente. Può avere categorie come "Maschio", "Femmina" o "Altro", che rappresentano l'identità di genere dell'utente.

- **VR Headset Type**

  La variabile identifica il tipo di visore VR utilizzato dall'utente durante l'esperienza VR

- **Duration**

  Rappresenta il periodo di tempo trascorso dall'utente nell'ambiente di realtà virtuale.

- **Motion sickness**

  Questa variabile indica la valutazione auto-riferita dell'utente della cinetosi sperimentata durante l'esperienza di utilizzo del visiore VR

- **Immersion Level**

  Questa variabile misura il livello di immersione dello scenario visualizzato durante l'esperienza VR. Viene riportato dall'utente e può essere misurato su una scala da 1 a 5, dove 5 indica il livello di immersione più alto.

Nel dataset, non tutte le caratteristiche presenti risultano essere altrettanto rilevanti per il nostro agente intelligente. Di conseguenza, abbiamo selezionato le caratteristiche più significative per le analisi del nostro agente intelligente, focalizzandoci specificamente sull'elaborazione dei dati necessari al nostro modello di Machine Learning.
Tra le caratteristiche **considerate**, vi sono le seguenti 5:

- **Age**
- **Gender**
- **Duration**
- **Motion sickness**
- **Immersion Level**

La durata (Duration) si configura come **variabile dipendente**, la quale il nostro agente intelligente dovrà stimare.

2.3 Incremento del dataset

Il **retraining**, o riaddestramento, rappresenta una fase cruciale nel ciclo di vita di un modulo di intelligenza artificiale, si riferisce al processo di aggiornamento e miglioramento del modello di intelligenza al fine di migliorare la qualità delle predizioni effettuate.
La **motivazione** principale che ci spinge a voler effettuare il retraining del nostro modulo di intelligenza artificiale risiede nella **scarsità dei dati** all'interno del dataset utilizzato per il training iniziale del modello; questa carenza di dati ha un impatto negativo sulla qualità delle previsioni del modello, di conseguenza, è essenziale aggiornare il modello con nuovi dati per migliorarne l'accuratezza e l'efficacia.

Per raccogliere questi dati ci si basa sulle informazioni degli utenti non appena interagiscono con MetaClass.
Informazioni relative alla motion sickness e all'immersion level, si acquisiscono dopo la compilazione di un report che il partecipante compila dopo **il termine** del meeting in cui ha partecipato. **La durata del meeting** per singolo utente, invece, viene rilevata direttamente dall'applicativo in cui è integrato il modello, che ricerca nei meeting precedenti in cui ha partecipato ogni singolo partecipante per ricavarne la durata totale in cui ha partecipato. Inoltre, le informazioni relative al **sesso ed età** di ciascun utente vengono acquisite automaticamente dal sistema stesso dopo l'autenticazione (necessaria) avvrnuta con account Facebook.
Una volta raccolte le **nuove istanze** di dati, il processo di retraining prevede di **addestrare nuovamente** il modello utilizzando queste nuove informazioni. Per facilitare questo processo, abbiamo sviluppato un **modulo Python dedicato**, come descritto nel 5.3
Questo modulo gestisce l'aggiornamento del modello, l'adattamento dei parametri e la valutazione delle prestazioni del modello aggiornato.

3 Ingegnerizzazione dei dati

3.1 Data cleaning

Nella prima fase di ingegnerizzazione dei dati ci si sofferma sul data cleaning, ovvero l'analisi dei dati per valutare la presenza di **valori nulli o non validi**.

Da un analisi preliminare dei dati a disposizione è risultato che **non sono presenti** valori nulli o invalidi tra quelli presenti, per cui non sono state necessarie tecniche di Data Cleaning:

```
#verifica se ci sono dati null nel dataset
df.isnull().any()
```

```
UserID            False
Age               False
Gender            False
VRHeadset         False
Duration          False
MotionSickness    False
ImmersionLevel    False
dtype: bool
```

Figura 1: Esempio di verifica dei dati per valori nulli

```
#verifica se ci sono dati NA nel dataset
df.isna().any()
```

```
UserID            False
Age               False
Gender            False
VRHeadset         False
Duration          False
MotionSickness    False
ImmersionLevel    False
dtype: bool
```

Figura 2: Esempio di verifica dei dati per valori N/A

3.2 Feature scaling

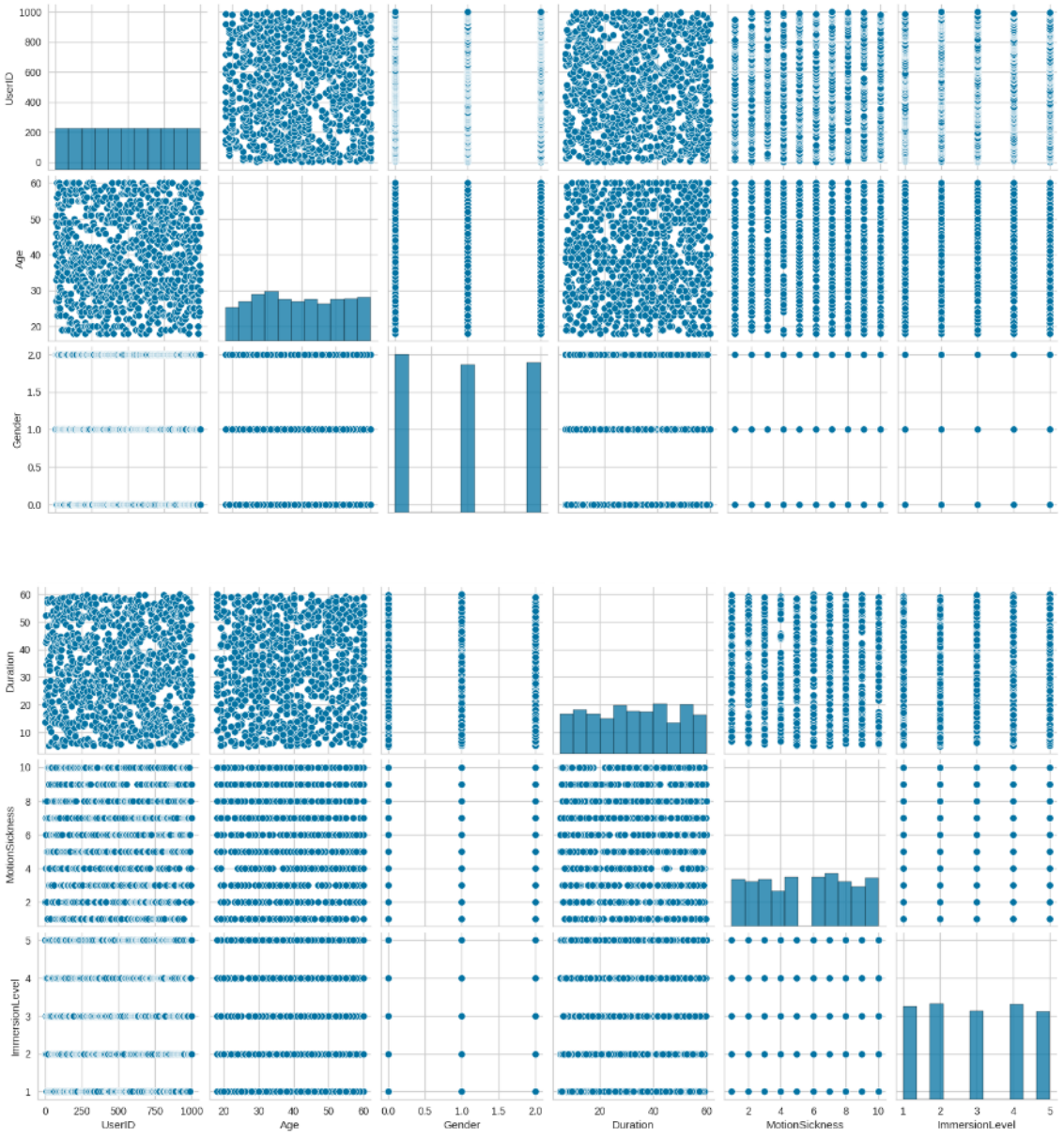
La feature scaling è l'insieme di tecniche che consentono di **normalizzare o scalare** l'insieme di valori di una caratteristica.

Si è utilizzato un metodo della libreria `sklearn` il quale ha permesso di scalare i valori delle variabili prese in considerazione per poi analizzate le prestazioni dell'agente in seguito alla tecnica di normalizzazione usata. Sono state prese in considerazione *la* ***Z-Score*** *Normalization, la* ***MinMax*** *Normalization e la* ***Robust Scaling***.

- la **Z-Score** Normalization: $x' = \frac{x-\mu}{\sigma}$ normalizza i valori in modo da avere la somma delle medie pari a 0 e la deviazione standard a 1.
- la **MinMax** Normalization: normalizza i valori dei dati, in valori compresi fra a e b.
- la **Robust Scaling**: normalizzazione che rimuove la mediana e scala i dati in base al range interquartile.

3.3 Feature selection

La feature selection ha l'obiettivo di definire delle **caratteristiche**, anche chiamate **feature**, metriche, o variabili indipendenti che possano caratterizzare gli aspetti principali del nostro problema in esame e, quindi, avere una buona potenza predittiva. Abbiamo utilizzato un metodo della libreria *matplotlib*, il quale ci ha permesso di visualizzare le dipendenze tra le diverse variabili.



3.4 Data Balancing

Il data balancing è un insieme di tecniche per convertire un dataset sbilanciato in uno bilanciato. Nel caso in questione, trattandosi di un problema di regressione lineare multipla, non **abbiamo avuto la necessità di verificare il bilanciamento delle classi**, poiché abbiamo dati dal valore continuo.

4 Soluzione al problema

4.1 Model evaluation

In seguito alla definizione delle tecniche di ingegnerizzazione dei dati dell'agente intelligente, è necessario stabilire le **metriche** e le **tecniche** di validazione delle prestazioni dello stesso. Occorre suddividere l'insieme dei dati fin'ora analizzato in due insiemi:

- **training set**: composto dalle istanze di dati che saranno utilizzate per l'addestramento.
- **test set**, composto dalle istanze di dati per cui l'agente dovrà predire il valore della variabile dipendente.

Si sono prese in considerazione diverse tecniche per effettuare questa suddivisione: *K-Fold validation, Stratified K-fold validation, Repeated K-fold validation e Repeated Stratified K-fold validation*. In particolare, le tecniche ***Stratified K-fold*** **e** ***Repeated Stratified*** **sono state scartate** in quanto non risultano adatte ad essere applicate su dati continui e a dataset con più variabili indipendenti.
Per tutte è stato necessario definire il valore di **K**, ovvero il numero di gruppi utilizzati per suddividere il dataset in test e training set, mentre per le tecniche di tipo "Repeated" è stato stabilito anche il numero di ripetizioni di validazione da effettuare. Le **metriche** che sono state impiegate per **valutare la bontà** delle previsioni effettuate sono state:

$$\text{MAE(Mean Absolute Error)} = \frac{1}{n}\sum_{i=1}^{n}|y_i - x_i| \tag{1}$$

$$\text{MSE(Mean Squared Error)} = \frac{1}{n}\sum_{i=1}^{n}(y_i - x_i)^2 \tag{2}$$

$$\text{RMSE (Root Mean Squared Error)} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - x_i)^2} \tag{3}$$

4.2 Regressione

4.2.1 Regressione lineare

Una volta definito il processo di **feature engineering** e **scelte le tecniche di valutazione,** è stato possibile definire un modello di regressione lineare tramite la libreria *sklearn.linearmodel*, in particolare si è utilizzato il metodo *fit* **per addestrare il modello** sull'insieme dei dati di training e in seguito *predict* sull'insieme dei dati di test **per predirne il valore della variabile** dipendente basandoci sulle feature dell'insieme di dati di training. Prima di procedere con la creazione del modello di regressione lineare multipla, è necessario verificare che i dati rispettassero le condizioni di:

- **Linearità dei dati**

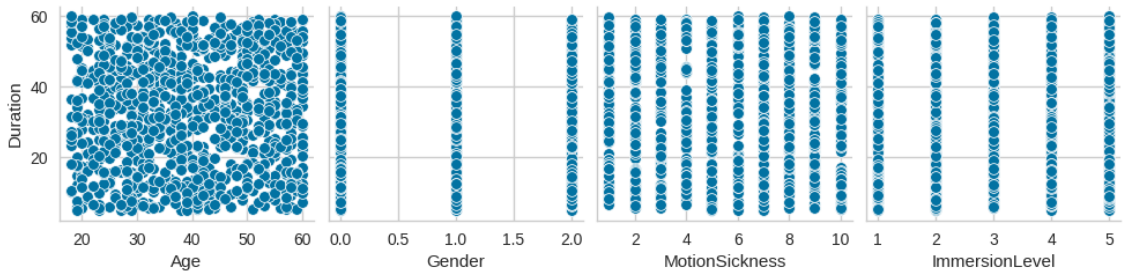


Figura 3: Linearità dei dati

da come si nota in figura, si hanno dei dati che hanno una **distribuzione omogenea e lineare** rispetto alla variabile dipendente *Duration*.

- **Bassa multicollinearità**

Figura 4: Multicollinearità

in quest'altra figura invece si nota che **le feature** utilizzate nel dataset n**on sono fortemente correlate tra loro,** quindi consente al modello di applicare una regressione lineare multipla senza preoccuparsi di avere variabili ridondanti durante l'addestramento.

Al fine di definire il modello che si adatti meglio i dati, è stato necessario applicare il **metodo empirico**, cioè l'utilizzo di vari algoritmi di regressione al fine di trovare quello più adatto al problema in esame.
Per ogni modello creato, sono stati applicate **3** diverse tipologie di normalizzazione del dataset per poter scegliere in modo più accurato il modello da utilizzare. Le 3 normalizzazioni sono quelle descritte nel paragrafo 3.2:

- **Z-Score normalization**

$$Z = \frac{x - \mu}{\sigma} \tag{4}$$

- **MinMax normalization**

$$X_{\text{norm}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}} \tag{5}$$

- **Robust scaling**

$$X_{\text{robust}} = \frac{X - \text{Mediana}}{\text{IQR}} \tag{6}$$

Infine, oltre alle 3 metriche descritte nel paragrafo 4.1, per poter valutare la validità delle stime è stato fatto un controllo e un plot dei **residui** ricavati durante l'esecuzione del modello sul *test set*.
Più in dettaglio abbiamo verificato:

- **normalità dei residui**: se i residui vengono **normalmente distribuiti** cioè se seguono o meno una distribuzione approssimativamente normale. Si guarda se la forma della distribuzione è a campana simmetrica.
- **omoschedasticità**: se gli errori residui hanno una varianza **costante**, cioè commettono un tasso di errore costante.

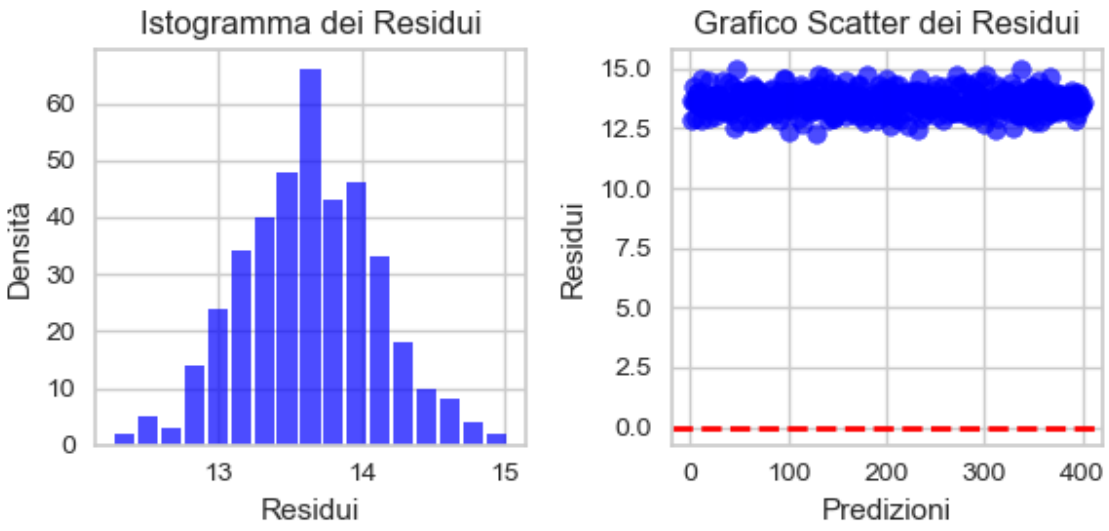


Figura 5: Esempio di plot dei residui

Partendo dalle metriche è stata deinita una semplice classe che contesse 3 array di tutte le metriche ricavate durante la fase di validazione:

```
#oggetto che contiene le metriche
class Metrics1:
  #costruttore
  def __init__(self,mae,mse,rmse):
    self.mae=mae       #MAE
    self.mse=mse       #MSE
    self.rmse=rmse     #RMSE

  #ToString
  def __str__(self):
    return f'Metrics [mae= {self.mae} mse= {self.mse} rmse= {
       self.rmse} mean= {np.mean([self.mae,self.mse,self.rmse
       ])}'
```

Listing 1: Classe Metrics1

Tale classe è servita per definire MetricsResultContainer, un ulteriore classe che funge da "ambiente" per poter eseguire metodi specifici sul plot dei residui e sulla stampa delle metriche in base al modello e alla tipologia di normalizzazione utilizzata:

```
class MetricsResultContainer:
    meanMAE = []
    meanMSE = []
    meanRMSE = []

    def __init__(self, model, alg, scaler, param, metricsMean)
       :
        self.model = model
        self.alg = alg
        self.scaler = scaler
        self.param = param
        self.metricsMean = metricsMean
        self.meanMAE = []
        self.meanMSE = []
        self.meanRMSE = []
```

Listing 2: MetricsResultContainer - struttura

```
    def printMetrics(self):
        for m in self.metricsMean:
            self.meanMAE.append(m.mae)
            self.meanMSE.append(m.mse)
            self.meanRMSE.append(m.rmse)
        print("meanMAE=", np.mean(self.meanMAE))
        print("meanMSE=", np.mean(self.meanMSE))
        print("meanRMSE=", np.mean(self.meanRMSE))

        self.visualizza_grafici()
        self.test_Durbin_Watson()

    # funzione per mostrare la normalit  dei residui e l'
       omoschedasticit
    def visualizza_grafici(self):
        fig, axs = plt.subplots(1, 2, figsize=(6, 3))  # Una
           riga, due colonne

        # Istogramma dei Residui
        axs[0].hist(self.meanMAE, bins='auto', color='blue',
           alpha=0.7, rwidth=0.85)
        axs[0].set_title('Istogramma dei Residui')
        axs[0].set_xlabel('Residui')
        axs[0].set_ylabel('Densit ')

        # Grafico Scatter dei Residui
        lunghezza_dati = len(self.meanMAE)
        axs[1].scatter(np.arange(1, lunghezza_dati + 1), self.
           meanMAE, color='blue', alpha=0.7)
        axs[1].axhline(y=0, color='red', linestyle='--',
           linewidth=2)
        axs[1].set_title('Grafico Scatter dei Residui')
        axs[1].set_xlabel('Predizioni')
        axs[1].set_ylabel('Residui')
        plt.tight_layout()  # Assicura una corretta
           disposizione dei grafici senza sovrapposizioni
        plt.show()
```

Listing 3: MetricsResultContainer - metodi

Nel costruttore è stato inserita anche la tecnica di convalida utilizzata (denominata come model). In questa fase ne sono state decise 2 in particolare:

- **K-fold cross validation**: metodo statistico che consiste nella ripetuta partizione e valutazione dell'insieme dei dati di partenza.
- **Repeated K-fold validation**: Si ripete N volte la K-fold cross validation.

Prima di fare ciò era necessario comprendere il numero adatto di **partizioni** da avere nel dataset utilizzando la formula:

$$k = (len(df)/(len(df) * 0.3)) \tag{7}$$

dove **len(df)** indica il numero di campioni del nostro dataset. Il K così ottenuto è uguale a 3.

4.2.2 Regressione non lineare

I modelli di regressione lineare semplici hanno principalmente 2 limitazioni, che li rendono difficilmente utilizzati nel concreto:

1. Difficoltà nella rappresentazione di relazioni **non lineari.**
2. Propensione all'**overfitting** con l'aumentare del numero di feature.

Il primo problema è semplicemente risolvibile con gli alberi decisionali, per mitigare al secondo problema occorre considerare i **gradi di libertà**, ovvero i punti in cui la funzione tende a flettersi e a curvarsi, ciò viene risolto per esempio attraverso la ***Ridge Regression.***

4.3 Algoritmi

Prima di parlare degli algoritmi è necessario trattare come viene effettuata la **generazione** del modulo. Il tutto viene fatta da una funzione chiamata *generateModel()*:

```
#funzione per generare il modello, divisione training e
    test, features scaling, selection
def generateModel(alg, scaler, model, select)
```

- **alg**: metodo di convalida utilizzato nella validazione del modello
- **scaler**: tecnica di normalizzazione dei dati usata
- **model**: algoritmo di regressione
- **select**: insieme delle feature selezionate

L'obiettivo del metodo è quello di effettuare la **divisione** del dataset in **test** e **training** set per poi applicare operazioni di data engeening per preparare i dati.

```
# suddivisione in training e test test
X_train, X_test = X.iloc[train_index], X.iloc[test_index]
y_train, y_test = y.iloc[train_index], y.iloc[test_index]
#feature scaling sui traing test
X_train_z = scaler.fit_transform(X_train)
X_test_z = scaler.transform(X_test)
#applicazione feature selection su train_z
X_train_z = select.fit_transform(X_train_z, y_train)
X_test_z = select.transform(X_test_z)
```

La suddivisione avviene andando a scegliere il numero di partizioni k per poi applicare l'algoritmo di suddivisione del dataset.

```
    #numero record nel dataset
    k=len(df)
    #calcolo k ideale da usare nelle tecniche di validazione
       deve essere il 30% della lunghezza del dataset
    k= (k/(k*0.3))
   #Kf divisione dataset per k gruppi per testare mediante due
       algoritmi Kfold-RepeateKFold
   kf = KFold(n_splits=int(np.ceil(k)),random_state=42,
      shuffle=True)
   #rKf con k gruppi, e 10 ripetizioni
   rkf = RepeatedKFold(n_splits=int(np.ceil(k)), n_repeats
      =100, random_state=42)
```

Da come si vede dal codice, vengono applicati i metodi descritti nel paragrafo 4.2.1.
A questo punto l'unica cosa da fare è l'**addestramento** per concludere con la **validazione**:

```
     #training dell'algoritmo sui
     training set
    clone_model.fit(X_train_z,y_train)
    #validazione modello e applicazione predizione sui testSet
    y_pred = clone_model.predict(X_test_z)
```

Questo blocco è rinchiuso in un *for*, quindi il valore predetto *ypred* è unico.
Col valore predetto, si ricavano i **residui** descritti nel paragrafo 4.2.1 andando poi a ricavare le 3 metriche di valutazione dei modelli.

```
     metrics1.append(
        Metrics1(metrics.mean_absolute_error(y_test,y_pred),
                 metrics.mean_squared_error(y_test,y_pred),
                 np.sqrt(metrics.mean_squared_error(y_test,
                    y_pred))
                 )
        )
  return metrics1
```

metrics è un'istanza della classe *MetricsResultContainer* che consente di poter stampare le metriche e visualizzare i grafici in caso di utilizzo di regressione lineare.

4.3.1 Linear Regression

Un regressore lineare, o modello di regressione lineare, cerca di modellare la relazione tra una variabile dipendente e una o più variabili indipendenti attraverso una **funzione lineare**.
Applicando la funzione di generazione del modello vista nel paragrafo 4.3 sono stati ottenuti questi risultati:

Tabella 1: Linear Regression

Test	Regressor	Alg	Scaler	MAE	MSE	RMSE
1	Linear	KF	ZScore	13.61091	249.70188	15.79693
2	Linear	RKF	ZScore	13.63839	250.71404	15.82796
3	Linear	KF	MinMax	13.61091	249.70188	15.79693
4	Linear	RKF	MinMax	13.63839	250.71404	15.82796
5	Linear	KF	Robust	13.61091	249.70188	15.79693
6	Linear	RKF	Robust	13.63839	250.71404	15.82796

Possiamo notare come la differenza nelle prestazioni ottenute utilizzando le due modalità di validazione *KF e RKF* siano a favore per la tecnica **K-fold**. Del resto, applicando una delle 3 tecniche di scaling **non si nota** nessuna **differenza** sostanziale.

4.3.2 DecisionTree Regression

In seguito alla realizzazione del modello utilizzante un semplice algoritmo di Regressione Lineare, le sue prestazioni sono state confrontate con altri modelli che fanno utilizzo di diversi algoritmi di regressione. Il primo algoritmo con cui è stato confrontato è stato il **Decision Tree**, che non si limita a predire dati con vincoli di linearità, ma permette anche di predire valori che si presentano sotto forma di curve.
Riportiamo di seguito i risultati ottenuti:

Tabella 2: DecisionTree Regression

Test	Regressor	Alg	Scaler	MAE	MSE	RMSE
1	DecisionTree	KF	ZScore	18.90721	512.38004	22.62610
2	DecisionTree	RKF	ZScore	18.57162	508.82534	22.54597
3	DecisionTree	KF	MinMax	19.22090	529.42730	22.99655
4	DecisionTree	RKF	MinMax	18.58814	509.21055	22.55407
5	DecisionTree	KF	Robust	19.05921	518.14221	22.75477
6	DecisionTree	RKF	Robust	18.56476	508.42116	22.53678

Si può facilmente notare come i risultati ottenuti siano **tutti peggiori** rispetto a quelli ottenuti con l'algoritmo di regressione lineare.

4.3.3 RandomForest Regression

Il confronto del modello è continuato andando a testare le prestazioni utilizzando l'algoritmo **RandomForest** Regression. Questo è un algoritmo di tipo "ensemble", cioè esegue n volte (nel nostro caso, $n = 100$) l'algoritmo **Decision Tree** Regression, ogni albero decisionale è creato in modo autonomo ed effettua le sue personali predizioni. In seguito, le predizioni finali sono poi ottenute tramite una media di quelle effettuate dai singoli alberi.

Di seguito riportiamo i risultati:

Tabella 3: RandomForest Regression

Test	Regressor	Alg	Scaler	MAE	MSE	RMSE
1	RandomForest	KF	ZScore	14.51364	300.07241	17.31909
2	RandomForest	RKF	ZScore	14.53357	300.50807	17.32987
3	RandomForest	KF	MinMax	14.55678	301.34773	17.35545
4	RandomForest	RKF	MinMax	14.49374	299.76998	17.30884
5	RandomForest	KF	Robust	14.59897	303.01378	17.40361
6	RandomForest	RKF	Robust	14.51948	300.19092	17.32114

Notiamo come in questo caso i risultati ottenuti siano notevolmente **migliori** rispetto al DecisionTree Regressor.
Di contro, i tempi di elaborazione dei risultati tramite questa tecnica sono notevolmente aumentati.

4.3.4 Lasso Regression

In seguito abbiamo analizzato l'algoritmo **Lasso** Regression, dove Lasso sta per "*Least Absolute Selection Shrinkage Operator*". Questo algoritmo infatti opera trovando e applicando un vincolo agli attributi del modello che porta i coefficienti di regressione per alcune variabili a diminuire verso lo zero. Le variabili con coefficiente di regressione pari a zero sono poi escluse dal modello. Questo algoritmo aiuta quindi a determinare quali dei predittori sono i più **importanti**.

Per utilizzare questa tecnica è necessario definire un parametro "**alpha**", con valore numerico compreso tra 0 e ∞, dove per ***alpha = 0*** la Lasso Regression si comporta come **Linear Regression**. La scelta di questo parametro è stata affidata alla funzione di Python *linear_model.Lasso()*.

Si può notare come i risultati ottenuti utilizzando questa configurazione siano moto **simili** alla **Linear Regression:**

Tabella 4: Lasso Regression

Test	Regressor	Alg	Scaler	MAE	MSE	RMSE
1	Lasso	KF	ZScore	13.59636	248.80255	15.77305
2	Lasso	RKF	ZScore	13.59176	248.96056	15.77076
3	Lasso	KF	MinMax	13.59636	248.80255	15.77305
4	Lasso	RKF	MinMax	13.59176	248.96056	15.77076
5	Lasso	KF	Robust	13.59636	248.80255	15.77305
6	Lasso	RKF	Robust	13.59176	248.96056	15.77076

4.3.5 Ridge Regression

In fine è stato analizzato l'algoritmo di **Ridge** Regression anche conosciuto come *Tikhonov regularization*. E' una versione regolarizzata della Regressione Lineare, aggiungendo un termine di regolarizzazione "**alpha**" alla *cost function*, l'algoritmo di apprendimento viene forzato a tenere i weight quanto più bassi possibili.

La Ridge Regression è in grado di determinare l'importanza di una feature tramite un fattore di penalità, in particolare L2 (squared size) penalizza il quadrato del valore dei coefficienti del modello. In pratica questo produce coefficienti piccoli, ma nessuno di loro è mai annullato. Quindi i coefficienti non sono mai 0. Il fenomeno è denominato *feature shrinkage*.

Tabella 5: Ridge Regression

Test	Regressor	Alg	Scaler	MAE	MSE	RMSE
1	Ridge	KF	ZScore	13.62292	249.80115	15.80010
2	Ridge	RKF	ZScore	13.62760	250.12743	15.81275
3	Ridge	KF	MinMax	13.62245	249.78355	15.79954
4	Ridge	RKF	MinMax	13.62699	250.10058	15.81189
5	Ridge	KF	Robust	13.62278	249.79550	15.79992
6	Ridge	RKF	Robust	13.62738	250.11838	15.81246

Anche qui possamo notare che le metriche sono molto vicine a quelle del Lasso Regression e della Linear Regression.

4.3.6 SVR Regression

Questo si basa sul più noto algoritmo di classificazione **SVM**, il quale costruisce un iperpiano che utilizza per la predizione di risultati. Come per l'algoritmo Random Forest, anche per questo abbiamo dovuto modificare i parametri della Repeated K-Fold validation.

I risultati ottenuti sono i seguenti:

Tabella 6: SVR Regression

Test	Regressor	Alg	Scaler	MAE	MSE	RMSE
1	SVR	KF	ZScore	13.71190	253.85640	15.92749
2	SVR	RKF	ZScore	13.69320	253.25402	15.91111
3	SVR	KF	MinMax	13.70895	253.73213	15.92403
4	SVR	RKF	MinMax	13.69007	253.17711	15.90868
5	SVR	KF	Robust	13.70853	253.27096	15.90938
6	SVR	RKF	Robust	13.69518	252.91018	15.90045

Anche qui possamo notare che le metriche sono molto vicine a quelle del Lasso Regression, della Linear Regression e del Ridge Regression

4.4 Algoritmo scelto

In base ai vari test e alle metriche prodotte, si è deciso di creare un **Lasso Regressor** applicando la **RobustScaler**.
La scelta dello scaler non inficiava troppo sulle prestazioni del modello in quanto si è visto che non comportava una varianza tra i residui medi. Tale modello per renderlo utilizzabile all'interno di MetaClass deve essere convertito in un file **PMML**.

PMML, acronimo di *Predictive Model Markup Language*, è uno standard XML (e più recentemente anche JSON) che fornisce una rappresentazione standardizzata per modelli di machine learning e modelli statistici. L'obiettivo principale di PMML è **consentire la portabilità** dei modelli tra diverse piattaforme e ambienti software.

Per fare la conversione è necessario prendere il nostro modello addestrato e inserirlo in una **PMMLPipeline**

```
# Adattamento e trasformazione dei dati con RobustScaler
scaler = RobustScaler()
X_scaled = scaler.fit_transform(X)

# Creazione del modello di regressione con LassoRegression
model = lassoReg
model.fit(X_scaled, y)

# Creazione del PMMLPipeline con il modello gi  addestrato
pipeline = PMMLPipeline([("Regression", model)])
```

Una pipeline in informatica e nell'ambito del machine learning è una sequenza di passaggi o operazioni che vengono eseguiti in successione su un set di dati. In particolare, nel contesto del machine learning, una pipeline può essere utilizzata per organizzare in modo strutturato il flusso di lavoro che coinvolge la preparazione dei dati e la creazione del modello.
Tuttavia per vari problemi di compilazione si è deciso di utilizzare la pipeline come mezzo di **conversione** del modello già addestrato, piuttosto che creare un flusso di operazione che partisse dalla preparazione dei dati e finisse alla creazione del modello.

Dopo ciò, si hanno tutti i requisiti per creare il file PMML:

```
# Tentativo di estrazione del pipeline in un file PMML con
   gestione delle eccezioni
try:
    sklearn2pmml(pipeline, "RegressoreDurataMeeting.pmml",
       with_repr=True)
except Exception as e:
    print("Si e' verificato un errore durante l'estrazione del
        pipeline in un file PMML:")
    print(e)
```

Tale file viene utilizzato per fare stime nel momento in cui vi sono meeting da creare. Il tutto è stato integrato in una classe Java che verrà descritta in modo più accurato nel paragrafo 5.1

5 Interazione con MetaClass

MetaClass è una **Companion App** che supporta funzionalità critiche di **MetaClassVR**, un'applicazione a realtà aumentata per la creazione di meeting e stanze virtuali.
Prima della creazione di MetaClass, tutte le funzionalità venivano gestite con l'ausilio di un visore che per novizi del mondo della realtà virtuale non era il massimo in termini di usabilità. MetaClass quindi viene incontro a queste esigenze, mostrando un'interfaccia chiara e funzionale su tutti i requisiti funzionali dell'applicativo iniziale, quali:

- **creazione e modifica stanze**
- **scheduling meetings**
- **modifica dati personali**
- **...**

MetaClass aggiunge però una nuova funzionalità, fornendo la possibilità all'utente di compilare un questionario. Il tutto al fine di prendere tali valutazioni e aggiungere nuove istanze al nostro dataset per cercare di migliorare i problemi di underfitting dovuti alla scarsità dei dati del nostro modello. (Leggere il paragrafo 2.3 per ulteriori dettagli).

5.1 Stima durata dei meeting

L'essenza del nostro regressore, come è stato spiegato nel paragrafo 1.3, è quella di rilevare i dati di un'utente per poi stimare la durata ideale di un meeting. La stima però deve essere eseguita su tutti gli utenti presenti in una stanza per far si che l'organizzatore, possa essere a conoscenza della durata ideale media da effettuare in quella determinata stanza. Il modello quindi deve effettuare stime su ogni partecipante della stanza per poi ritornare una durata media dei meetings. In seguito è riportato un'estratto di codice che consente di fare ciò:

```
@Override
public Double getDurataMeeting(Long idStanza) throws
   RuntimeException403, ServerRuntimeException {
  // controllo la stanza se esiste
  Stanza s;
  if ((s = stanzaRepository.findStanzaById(idStanza)) ==
     null) {
    throw new RuntimeException403("non    possibile
       effettuare la stima: stanza non trovata");
  }
  // mi recapito la lista di utenti in stanza
  List<Utente> users;
  if ((users = statoPartecipazioneRepository.
     findUtentiInStanza(s.getId())) == null) {
    throw new ServerRuntimeException(
        "non    possibile effettuare la stima: " + "errore
           nella ricerca dei partecipanti");
  }
  double mediaDuration = 0;

  for (Utente u : users) {
    mediaDuration += requestPrediction(u);
  }
  return mediaDuration / users.size();
}
```

Listing 4: Stima durata meeting ideale

Dove vengono estratti tutti gli utenti presenti in stanza per poi invocare il metodo *requestPrediction()* che prende in input un utente ed effettua la stima su di esso.

Questa operazione viene eseguita per tutti i partecipanti e le stime eseguite vengono sommate nella variabile *mediaDuration* per poi ritornare la media di tali stime.
Il metodo requestPrediction() ha il compito di preparare i dati dell'utente per renderli compatibili con il regressore (memorizzato in un file PMML) per poi effettuare la stima.

https://github.com/Everysimo/MetaClass per ulteriori dettagli sul codice relativo alle stime.

Russell e Norvig 2009

5.2 Aggiunta istanze

MetaClass,come descritto prima, consente la compilazione di un questionario, dopo la conclusione di un meeting.
I dati del questionario vengono presi per **incrementare** il dataset, infatti oltre al motionSickenss e al Immersivity Level, si prelevano dati come età e sesso. (Ulteriori dettagli sono descritti nel paragrafo 2.3) Per poter inserire la tupla nel dataset ci occorre però un modo per poter ricavare lo *UserID* dell'ultima istanza del dataset per incrementarla e farla diventare lo *UserID* della prossima tupla da inserire. Per fare ciò è stato necessario utilizzare la libreria *org.apache.commons* per poter leggere e scrivere in un file formato csv (formato di estrazione del dataset).

```
import org.apache.commons.csv.CSVFormat;
import org.apache.commons.csv.CSVParser;
import org.apache.commons.csv.CSVPrinter;
import org.apache.commons.csv.CSVRecord;
```

Questi sono gli oggetti utilizzati per accedere a un file csv, in particolare:

- **CSVFormat**: Questo oggetto definisce il formato del file CSV, specificando opzioni come il separatore di colonne, il carattere di escape e altri parametri di formattazione. Viene utilizzato per configurare la lettura e la scrittura dei file CSV.
- **CSVParser**: Questo oggetto è responsabile della lettura di un file CSV, analizzandolo e convertendo i dati in oggetti di tipo CSVRecord. Utilizza il CSVFormat per interpretare correttamente la struttura del file CSV.
- **CSVPrinter**: Questo oggetto viene utilizzato per la scrittura di dati in un file CSV. Accetta dati in forma di oggetti CSVRecord e li formatta secondo il CSVFormat specificato, scrivendoli nel file CSV di output.
- **CSVRecord**: Rappresenta una singola riga di dati in un file CSV. I valori dei singoli campi sono accessibili attraverso metodi come get o tramite l'indicizzazione. Viene utilizzato sia durante la lettura dei file CSV da CSVParser che durante la scrittura da CSVPrinter.

Ottenuta la tupla risultante, è possibile inserirla nel dataset:

```
// Aggiunta della nuova tupla di valori al CSV
    csvPrinter.printRecord(
        ultimoUserId + 1, // UserID
        periodo.getYears(), // Age
            (u.getSesso().equals("M") ? 2 : (u.getSesso().
               equals("F")  ? 1 : 0)), // Gender
        null, // VR Headset (non utilizzato e rimosso
           nella feature selection)
        (double) durata.toMinutes(), // Duration
        immersionLevel, // ImmersionLevel
        motionSickness); // MotionSickness
```

identificatore Da come si può notare il sesso è stato espresso come un valore numerico (per consentire lo scaling durante il training del modello) e la cella relativa all'headSet è nulla in quanto verrà eliminata a prescindere durante le feature selection.
Infine una questione importante ce l'ha l'attributo Duration che è stato ricavato andando a fare una media di tutti i minuti in cui è stato l'utente all'interno dei meetings (considerando anche le altre stanze in cui ha accesso).

Infine dopo l'inserimento di dati era necessario stabilire quando effetturare il **retraining del modello** avendo a disposizione 2 alternative:

- **allenarlo dopo ogni utente aggiunto**
- **allenarlo dopo n inserimenti**

A primo impatto sembra che la prima opzione sia la migliore, in quanto si ha fin da subito il modello riallenato e quindi consente di fornire stime, fin da subito, più accurate. Ma questo porta un esecuzione lenta nella compilazione del questionario. In parole povere, quando l'utente compila un questionario, dovrà attendere diversi secondi per avere un feedback di avvenuta compilazione. Il sistema quindi diventa meno usabile. A valle di ciò si è deciso di implentare la seconda opzione, dove **ad ogni 100 inserimenti** veniva effettuato il retraining:

```
    //incremento il contatore che mi tiene traccia di quante
       tuple vengono aggiunte
       counter++;
     //ogni 100 inserimenti avviene il retraining
     if ((counter% 100) == 0) {
       // ri-training del modello
       trainingModel();
     }
```

Per fare ciò, è servito un contatore statico che veniva incrementato ad ogni inserimento, poi in un if veniva controllato se il contatore ha un modulo pari a 100 (cioè se sono stati inserite 100 tuple).

5.3 Retraining del modello

Una volta inserite le nuove istanze, il modello deve essere **riallenato** considerando i nuovi valori inseriti. Per fare ciò è stato necessario creare un **modulo Python** che contenesse un **interprete** e un'**ambiente virtuale** in grado di installare tutte le librerie necessarie per l'esecuzione di un file Python *(trainingModel.py)*.

```
# coding: utf-8
import pandas as pd

FILE_NAME = "../src/main/resources/ModuloAI/data.csv"

# Verifica il contenuto del file
try:
    df = pd.read_csv(FILE_NAME)
except pd.errors.EmptyDataError:
    print("Il file CSV    vuoto.")
except pd.errors.ParserError:
    print("Errore di parsing del file CSV.")
except Exception as e:
    print(e)
```

Listing 5: Prelievo del dataset

In questa parte si preleva il dataset da una specifica cartella del progetto. Tale dataset è stato estratto in formato **CSV**, e utilizzando il modulo **pandas**, si ottiene il **DataFrame** contenente tutte le tuple. In caso di dataset vuoto, viene gestita un'eccezione (che sarà poi gestita dal metodo chiamante in Java), così come per gli errori di **parsing** del file CSV e per errori generici.

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import RobustScaler
from sklearn2pmml import sklearn2pmml
from sklearn2pmml.pipeline import PMMLPipeline

# Scelta variabile dipendente (y) e indipendenti (X)
X = df.drop(columns=['Duration', 'VRHeadset', 'UserID'])
   # Selezione delle feature
y = df.Duration

# Adattamento e trasformazione dei dati con RobustScaler
scaler = RobustScaler()
X_scaled = scaler.fit_transform(X)

# Creazione del modello di regressione con Random Forest
model = RandomForestRegressor()
model.fit(X_scaled, y)
```

Listing 6: Preparazione delle feature

In questo blocco di codice, si trasforma il dataset e i dati al suo interno per prepararli al **training** del modello (vedi paragrafo 3.1 per una descrizione più dettagliata). Dopo aver applicato il **metodo empirico** su vari algoritmi e aver testato **3 tipologie di scaling** diverse, si è constatato che l'algoritmo che si adatta meglio ai dati è il **LassoRegressor**, applicando la **RobustScaler** ai dati. I passi per preparare i dati includono la **feature selection**, il **feature scaling** con il **RobustScaler** e il **feature balancing e cleaning** non sono stati applicati, come descritto nel *paragrafo 3.1*.

```
# Creazione del PMMLPipeline con il modello gi  addestrato
pipeline = PMMLPipeline([("Regression", model)])

# Tentativo di estrazione del pipeline in un file PMML con
   gestione delle eccezioni
try:
   sklearn2pmml(pipeline, "RegressoreDurataMeeting.pmml",
      with_repr=True)
except Exception as e:
   print("Si    verificato un errore durante l'estrazione
      del pipeline in un file PMML:")
   print(e)
```

Listing 7: Conversione in un file PMML

In questa parte, si converte il modello già addestrato in un **file PMML**, che verrà utilizzato per effettuare stime in un altro modulo di MetaClass (descritto nel paragrafo 5.1). Per effettuare la conversione, è necessario creare una **pipeline** contenente il **modello addestrato**. L'istanza model contiene anche tutti i dati del dataset con la relativa separazione tra variabili dipendenti e indipendenti. Infine, nel blocco try/except, si crea il file che viene salvato nella directory corrente del modulo Python appena descritto.

6 Glossario

Termine	**Definizione**
Agente intelligente	Programma informatico che agisce autonomamente e in modo intelligente, spesso utilizzando tecniche di intelligenza artificiale.
Schedulare	Pianificare o programmare eventi in un determinato momento o luogo.
PEAS	Acronimo per Performance, Environment, Actuators, Sensors, usato per descrivere le caratteristiche di un agente intelligente.
Performance	Misura di quanto accuratamente un agente si avvicina alla durata reale di un meeting durante la programmazione.
Environment	Contesto in cui opera un agente, comprensivo dei feedback degli utenti all'interno dell'applicativo.
Actuators	Dispositivi o strumenti che consentono all'agente di interagire con il suo ambiente, come le caselle di input del range orario.
Sensors	Dispositivi o strumenti che consentono all'agente di percepire informazioni dall'ambiente.
Stocastico	Lo stato successivo dell'ambiente dipende non solo dallo stato corrente e dall'azione dell'agente, ma anche da una componente indipendente dall'agente.
Discreto	Ambiente discreto in quanto il numero di caratteristiche analizzate è finito e non varia in base alle condizioni.
Dataset	Insieme di dati utilizzati per addestrare e testare modelli di machine learning.
Motion sickness	Valutazione auto-riferita dell'utente della cinetosi sperimentata durante l'esperienza di utilizzo del visore VR.
Immersion Level	Livello di realismo degli scenari durante l'esperienza VR, misurato su una scala da 1 a 5, dove 5 indica il livello di immersione più alto.

Termine	**Definizione**
Retraining	Processo di riaddestramento del modello di intelligenza artificiale utilizzando nuovi dati.
Z-Score Normalization	Un metodo di feature scaling che normalizza i valori in modo da avere la somma delle medie pari a 0 e la deviazione standard a 1.
MinMax Normalization	Un metodo di feature scaling che normalizza i valori dei dati in un intervallo specificato, tipicamente compreso tra 0 e 1.
Robust Scaling	Un metodo di feature scaling che rimuove la mediana e scala i dati in base al range interquartile.
Model evaluation	Valutazione delle prestazioni di un modello di machine learning utilizzando metriche specifiche.
Training set	Insieme di dati utilizzato per addestrare un modello di machine learning.
Regressione lineare	La regressione lineare è un metodo statistico per modellare la relazione tra una variabile dipendente e una o più variabili indipendenti tramite una linea retta.
Directory	Struttura gerarchica di organizzazione dei file all'interno di un sistema operativo.
Metodo empirico	Approccio basato sull'esperienza pratica e sull'osservazione piuttosto che su teorie o principi scientifici.
DataFrame	Struttura dati tabellare bidimensionale utilizzata in librerie di analisi dati come Pandas in Python.
Feedback	Risposte o reazioni fornite dagli utenti di un sistema all'output prodotto dal sistema stesso.
Tupla	Una sequenza ordinata e immutabile di elementi, spesso utilizzata per rappresentare record o vettori di dati.
Organizzatore	Persona o strumento che organizza e gestisce le attività, le risorse o gli eventi.
Regressore	Algoritmo di machine learning utilizzato per la modellazione della regressione.
Underfitting	Condizione in cui un modello di machine learning è troppo semplice per catturare la complessità dei dati.

Termine	**Definizione**
Requisiti funzionali dell'applicativo	Descrizione delle funzionalità che un'applicazione software deve fornire per soddisfare i bisogni degli utenti.
Interfaccia	Punto di contatto tra un sistema e l'utente o altri sistemi, attraverso il quale avviene lo scambio di informazioni e comandi.
MetaClass	È una Companion App che supporta funzionalità critiche di MetaClassVR.
MetaClassVR	Un'applicazione a realtà aumentata per la creazione di meeting e stanze virtuali.
Scaler	Trasformatore utilizzato per normalizzare o standardizzare le caratteristiche dei dati in un modello di machine learning.
Iperpiano	Sottospazio di dimensione $n-1$ in uno spazio vettoriale di dimensione n.
Feature shrinkage	Fenomeno in cui i coefficienti di un modello di regressione sono ridotti verso lo zero per evitare l'overfitting.
Ensemble	Tecnica che combina i risultati di più modelli di apprendimento automatico per migliorare le prestazioni complessive del sistema.
Overfitting	Condizione in cui un modello di machine learning si adatta eccessivamente ai dati di addestramento, perdendo la capacità di generalizzazione.
Residui	Differenza tra i valori osservati e i valori previsti da un modello statistico.
Istanze	Esempi di dati utilizzati per addestrare e testare un modello di machine learning.
Meeting	Incontro o riunione tra due o più persone per discutere argomenti specifici.
Machine Learning	Branca dell'intelligenza artificiale che si concentra sull'addestramento dei computer a imparare dai dati e migliorare le prestazioni nel tempo senza essere esplicitamente programmati.

Termine	**Definizione**
Pipeline	Sequenza di passaggi o operazioni utilizzate per trasformare i dati grezzi in un formato adatto all'analisi o all'elaborazione successiva.
File PMML	File XML utilizzato per rappresentare modelli di dati di machine learning in modo standardizzato.
Varianza	Misura della dispersione dei dati rispetto alla media, che rappresenta la variabilità o l'eterogeneità dei dati.
Metriche	Misurazione quantitativa delle prestazioni di un modello di machine learning, utilizzata per valutare l'accuratezza e l'efficacia del modello.
Random	Casuale, senza un ordine predefinito o una sequenza.
Algoritmi	Procedura o formula matematica utilizzata per risolvere un problema o compiere una serie di operazioni.
Repeated K-fold validation	Tecnica che ripete N volte la K-fold cross validation.
K-fold cross validation	Tecnica statistica che consiste nella ripetuta partizione e valutazione dell'insieme dei dati in sottoinsiemi detti "fold".
MetricsResultContainer	Classe che contiene le metriche medie MAE, MSE e RMSE per un insieme di modelli.
Robust Scaling	Un metodo di feature scaling che rimuove la mediana e scala i dati in base al range interquartile.
Feature	Una caratteristica o un attributo utilizzato per rappresentare i dati in un modello di machine learning.
Scaler	Trasformatore utilizzato per normalizzare o standardizzare le caratteristiche dei dati in un modello di machine learning.
Iperpiano	Sottospazio di dimensione $n-1$ in uno spazio vettoriale di dimensione n.

Riferimenti bibliografici

Burkov, Andriy (2020). *Machine Learning Engineering*. O'Reilly Media. ISBN: 978-1098123838. URL: https://www.oreilly.com/library/view/machine-learning-engineering/9781098123838/.

Géron, Aurélien (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. 2nd. O'Reilly Media. ISBN: 978-1492032649. URL: https://www.oreilly.com/library/view/hands-on-machine-learning/9781492032632/.

Russell, Stuart J. e Peter Norvig (2009). *Artificial Intelligence: A Modern Approach*. 3th. Prentice Hall. ISBN: 978-0136042594. URL: https://www.amazon.com/Artificial-Intelligence-Modern-Approach-3rd/dp/0136042597.

Wikipedia contributors (2024). *Artificial intelligence*. Pagina di Wikipedia sull'intelligenza artificiale. Wikipedia, The Free Encyclopedia. URL: https://en.wikipedia.org/wiki/Artificial_intelligence.