Progetto di FIA

AIron Mike

Autore: Rocco Vigliotta
Matricola: 0512115447
https://github.com/RVigliotta/AIron-Mike

Università degli studi di Salerno
Corso di Laurea in Informatica
Anno Accademico 2024/2025

Indice

1. Introduzione 1

- 1.1 Obiettivo del progetto 1
- 1.2 Specifica PEAS 1
- 1.3 Caratteristiche dell'ambiente 1
- 1.4 Analisi del Problema 2
- 1.5 Contesto Applicativo 2
- 1.6 Struttura del Report 3

2. Data Understanding 3

- 2.1 Descrizione del Dataset 3
- 2.2 Distribuzione dei Dati 3
- 2.3 Matrice di Correlazione 5
- 2.4 Valori Mancanti 5
- 2.5 Distribuzione Outliers 6

3. Data Preprocessing 7

- 3.1 Gestione di Valori Specifici 7
- 3.2 Gestione di Valori Mancanti 7
- 3.3 Encoding delle Variabili 9
- 3.4 Gestione degli Outliers 9
- 3.5 Feature Engineering 10

4. Data Splitting 11

- 4.1 Differenza tra Training Set, Validation Set e Test Set 11
- 4.2 Percentuali di Suddivisione 11
- 4.3 Differenza tra Features (X) e Target (Y) 12

5. Addestramento Modelli 12

- 5.1 Preprocessing e Gestione dei Dati 13
- 5.2 Predizione Risultato (fase 1) 14
- 5.3 Predizione Decisione (fase 2) 15
- 5.4 Valutazione Performance e Metriche 15

6. Agente di Rinforzo Intelligente 18

- 6.1 Introduzione all'Apprendimento per Rinforzo 18
- 6.2 Struttura dell'Agente di Rinforzo 19
- 6.3 Algoritmo di Apprendimento 20
- 6.4 Training dell'Agente 21
- 6.5 Valutazione performance e metriche 22

7. Conclusioni 24

1 Introduzione

1.1 Obiettivo del Progetto

L'obiettivo principale di questo progetto è sviluppare un modello predittivo robusto in grado di prevedere l'esito degli incontri di pugilato in base alle caratteristiche dei combattenti e alle prestazioni passate. Attraverso l'analisi e la modellazione di un set di dati completo di incontri di pugilato, il progetto mira a identificare e quantificare i fattori chiave che influiscono sul risultato di un incontro. Questo approccio basato sui dati fornisce una guida strategica per analisti sportivi, allenatori e appassionati, migliorando la comprensione dei fattori che influenzano la performance.

1.2 Specifica PEAS

- **Performance:** Misure di prestazione adottate per valutare l'operato degli agenti (ResultAgent e DecisionAgent). Nel caso del ResultAgent, misuriamo la capacità di prevedere correttamente i risultati (vittoria, sconfitta, pareggio). Per il DecisionAgent, valutiamo la capacità di prendere decisioni adeguate rispetto ai risultati previsti.
- **Environment:** L'ambiente è composto da due aree di azione, ognuna gestita da un agente specifico (descritto sotto).
- **Actuators:** Gli attuatori disponibili per ciascun agente sono le azioni che possono intraprendere. ResultAgent può scegliere tra `result_win_A`, `result_win_B`, e `result_draw`, mentre DecisionAgent può scegliere tra azioni tattiche definite in base alla situazione, ovvero 9 decisioni possibili.
- **Sensors:** L'agente riceve gli input percettivi dai dati di allenamento. Questi input includono le caratteristiche di ogni stato, come attributi che descrivono la situazione e, per il DecisionAgent, anche le previsioni di risultato generate dal ResultAgent.

1.3 Caratteristiche dell'ambiente

- **Multi-agente cooperativo:** Sono presenti due agenti che collaborano per ottimizzare l'obiettivo generale. Il ResultAgent fornisce previsioni, e il DecisionAgent utilizza queste previsioni per prendere decisioni più mirate.
- **Deterministico:** L'ambiente è deterministico poiché gli agenti sanno come le loro azioni influenzeranno lo stato successivo (non c'è incertezza sugli effetti delle azioni intraprese).
- **Statico:** L'ambiente non cambia in maniera indipendente dagli agenti. I dati di stato (come le caratteristiche del contesto) rimangono statici, e le azioni degli agenti influenzano solo le previsioni o le decisioni.
- **Sequenziale:** Le azioni eseguite dagli agenti influenzano gli stati futuri e l'andamento della ricompensa cumulativa, poiché ogni decisione ha effetto sull'episodio complessivo.

- **Episodico:** Ogni episodio consiste in una sequenza di decisioni da parte degli agenti e termina una volta che tutti gli stati sono stati attraversati (ossia, quando i dati sono stati elaborati completamente).

1.4 Analisi del Problema

- **Stato Iniziale:** Gli stati iniziali per entrambi gli ambienti sono forniti dai dati preprocessati di input (file `x_train.csv` e `y_train.csv`). Il ResultAgent riceve i dati di input (caratteristiche) e le etichette di risultato, mentre il DecisionAgent riceve le previsioni di risultato del ResultAgent come input aggiuntivo per prendere le proprie decisioni.

- **Descrizione delle Azioni Possibili:** Ogni agente ha a disposizione un insieme di azioni che può intraprendere:

  ResultAgent: può scegliere tra `result_win_A`, `result_win_B`, e `result_draw`.

  DecisionAgent: ha un insieme di azioni tattiche che può selezionare, di 9 decisioni, a seconda della situazione e delle scelte effettuate dal ResultAgent.

- **Modello di Transizione:** Il modello di transizione per entrambi gli agenti è deterministico. Per il ResultAgent, ogni azione scelta determina una previsione di risultato che influenza il feedback ricevuto. Per il DecisionAgent, l'azione selezionata in base alla previsione del ResultAgent determina la valutazione dell'esito della decisione (ricompensa).

- **Test Obiettivo:** Gli obiettivi degli agenti sono:

  ResultAgent: prevedere accuratamente i risultati dell'interazione, massimizzando la ricompensa ottenuta tramite previsioni corrette.

  DecisionAgent: prendere decisioni strategiche ottimali basate sulle previsioni, massimizzando il reward cumulativo.

- **Costo del Cammino:** Ogni azione ha un costo unitario; quindi, ogni mossa costa lo stesso in termini di risorse computazionali, ma gli agenti guadagnano più reward per decisioni e previsioni corrette.

1.5 Contesto Applicativo

Nel contesto del pugilato e dell'analisi sportiva, le tecniche predittive rivestono un ruolo centrale. La previsione dei risultati dei combattimenti fornisce un supporto strategico per la pianificazione degli incontri, l'analisi delle caratteristiche dell'avversario e la preparazione ottimale dei pugili. Il progetto può essere utilizzato per facilitare decisioni in tempo reale anche nel contesto delle scommesse sportive come supporto all'intuizione e lo studio di scommettitori, appasionati e bookmakers.

1.6 Struttura del Report

1. **Comprensione dei Dati**: esplora e analizza il dataset, fornendo statistiche di base e visualizzazioni.
2. **Pre-elaborazione dei Dati**: descrive il processo di pulizia e preparazione dei dati.
3. **Ingegneria delle Feature**: dettagli sulle tecniche utilizzate per selezionare e creare nuove feature rilevanti.
4. **Modellazione Predittiva**: esamina i modelli di machine learning utilizzati e le metriche di valutazione.
5. **Risultati e Discussione**: presenta i risultati ottenuti e interpreta le relazioni tra variabili.
6. **Conclusioni e Sviluppi Futuri**: riassume i risultati e suggerisce possibili miglioramenti.

2 Data Understanding

2.1 Descrizione del Dataset

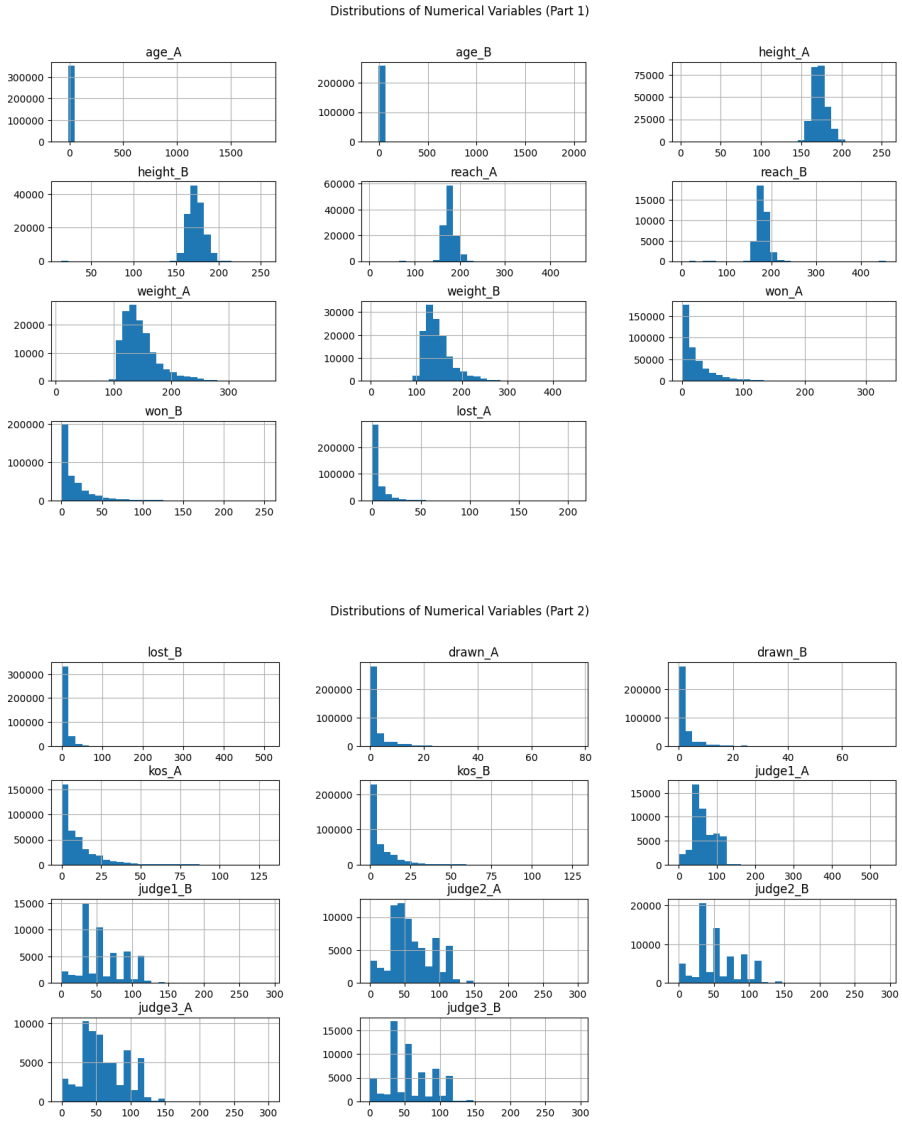
Il dataset utilizzato in questo progetto è composto da oltre 300.000 record di incontri di pugilato, provenienti da [https://www.kaggle.com/datasets/mexwell/boxing-matches]. Il dataset contiene informazioni dettagliate sui combattenti, strutturate in 26 colonne, tra cui:

- **Caratteristiche Fisiche**: età, altezza, reach, peso
- **Risultati Precedenti**: vittorie, sconfitte, pareggi, record di KO.
- **Risultato incontro**: vittoria del combattente A, vittoria del combattente B, pareggio
- **Decisione incontro**: SD, UD, KO, TKO, MD, RTD, TD, PTS, DQ, NWS
- **Posizione e Stile di Combattimento**: posizione ortodossa o mancina.

Le informazioni coprono un lungo periodo temporale, permettendo di analizzare l'evoluzione delle prestazioni in diverse epoche di pugilato. La dimensione e la completezza del dataset lo rendono adatto ad analisi dettagliate e alla costruzione di un modello predittivo accurato.

2.2 Distribuzione dei Dati

Per ottenere una panoramica delle variabili quantitative e qualitative, sono state generate alcune statistiche di riepilogo e grafici esplorativi. Innanzitutto sono state suddivise in variabili numeriche e categoriali. Sono stati utilizzati grafici a barre e istogrammi per le variabili, al fine di visualizzare la distribuzione dei dati in modo chiaro e comprensibile. Questo ha permesso di identificare eventuali anomalie e outlier, e di comprendere meglio la distribuzione dei vari attributi.



L'analisi visuale di questi istogrammi rivela che le variabili mostrano una buona distribuzione complessiva. Ogni istogramma offre una rappresentazione chiara della frequenza con cui si presentano le diverse categorie di dati. La distribuzione delle variabili appare bilanciata senza evidenti distorsioni, suggerendo una raccolta dati omogenea e ben calibrata. In particolare, le variabili relative ad età e peso mostrano curve che si avvicinano a una distribuzione normale, con un picco centrale e una simmetria laterale che evidenzia una varianza contenuta. Le variabili legate all'altezza e alla portata presentano una buona distribuzione con minime deviazioni, indicando una consistenza nei dati raccolti. Anche le variabili legate ai risultati mostrano una distribuzione coerente con una leggera propensione alla concentrazione centrale, che può riflettere le performance medie.

L'immagine mostra chiaramente le distribuzioni delle variabili result e decision, elementi chiave per il modello predittivo. Per la variabile result, abbiamo tre classi ben rappresentate. Tuttavia, è evidente una sovrarappresentazione della classe win A rispetto alle altre due. Questo squilibrio può influenzare le previsioni del modello, quindi potrebbe essere necessario considerare tecniche di bilanciamento per garantire risultati

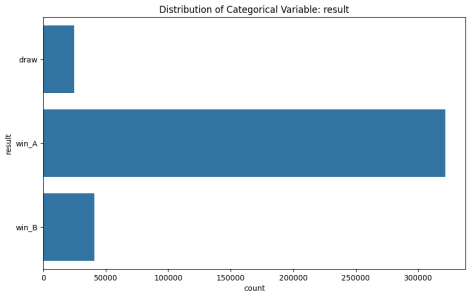


Figura 1: Distribuzione dei risultati

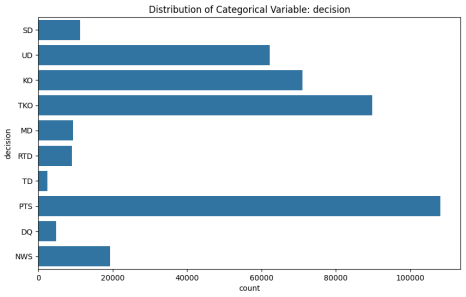
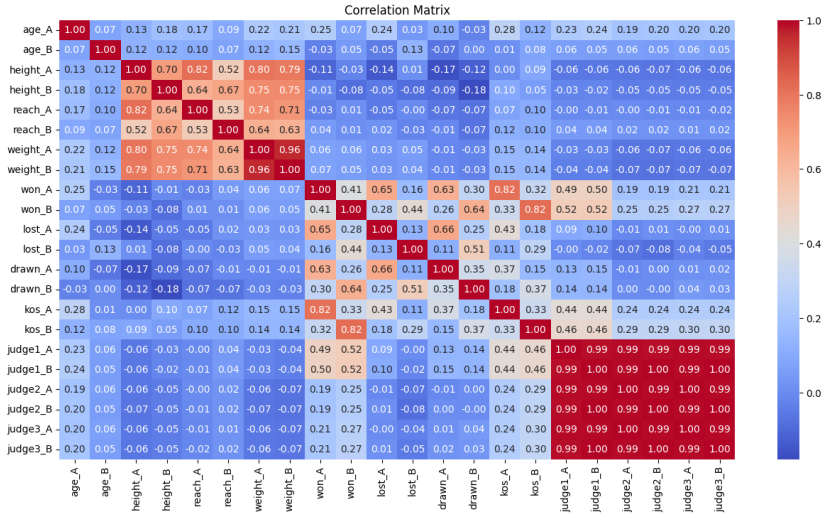


Figura 2: Distribuzione delle decisioni

imparziali. Per quanto riguarda la variabile decision, si osserva una distribuzione tra più classi, alcune delle quali sono piuttosto rare. Il numero di campioni per queste classi rare non è elevato, il che potrebbe rappresentare una sfida per il modello.

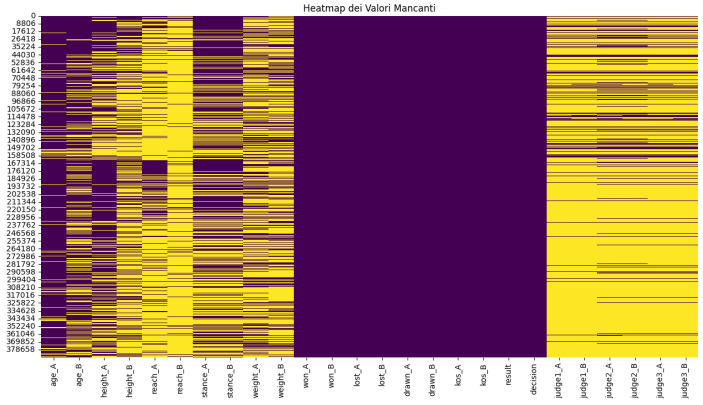
2.3 Matrice di correlazione



Per quanto riguarda la matrice di correlazione, rappresenta visivamente le correlazioni tra diverse variabili in un dataset. È visualizzata come una heatmap con colori che variano dal blu (correlazione negativa) al rosso (correlazione positiva), e mostra valori di correlazione che vanno da -1.0 a 1.0. Analizzando la matrice, possiamo trarre diverse osservazioni: Le variabili dei giudici A presentano forti correlazioni tra loro, con valori vicini a 1.0. Lo stesso vale per i giudici B. Le variabili di altezza e portata A hanno una correlazione moderata (0.82), così come altezza e portata B (0.67). Alcune correlazioni negative significative includono kos A con età A (-0.28) e kos A con sconfitte A (-0.25). Molte altre variabili mostrano correlazioni basse o nulle tra di loro, indicando che non esiste una forte relazione lineare tra queste variabili.

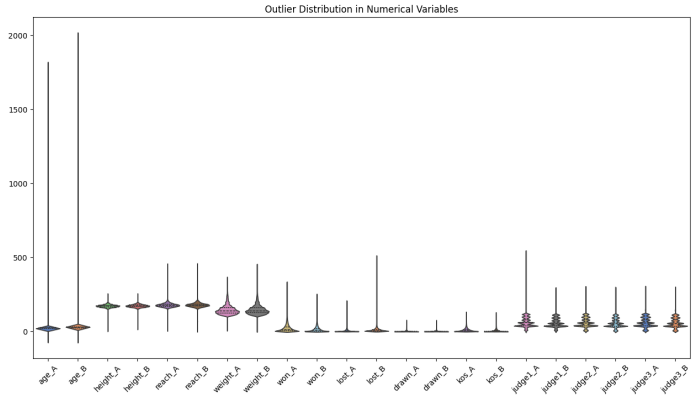
2.4 Valori mancanti

L'immagine mostra una mappa di calore utilizzata per rappresentare la presenza e l'assenza di valori mancanti all'interno di un dataset. I valori mancanti sono colorati in



giallo, mentre quelli presenti sono evidenziati in viola. Sull'asse verticale sono riportati gli indici delle osservazioni, che spaziano da 0 a 378658, mentre sull'asse orizzontale sono rappresentate le diverse variabili del dataset.

2.5 Distribuzione Outliers



Il grafico rappresenta la distribuzione degli outliers nelle variabili numeriche di un dataset, utilizzando delle violin plots per una visualizzazione dettagliata. Le variabili sono disposte sull'asse orizzontale, mentre l'asse verticale indica la frequenza degli outliers. Le variabili considerate includono parametri legati all'età, altezza, peso e altre misurazioni correlate. Un'analisi attenta del grafico mostra che alcune variabili, come quelle relative all'età e all'altezza, presentano un numero considerevole di outliers. In particolare, per le variabili legate all'età, sono presenti valori che superano eccessivamente i limiti plausibili, indicando chiaramente errori di inserimento o di misurazione dei dati. Analogamente, le variabili legate all'altezza mostrano dati che si discostano notevolmente dai valori attesi, suggerendo che molti degli outliers non rappresentano eventi rari o eccezionali, ma semplicemente dati sbagliati. La presenza di un numero elevato di outliers in queste variabili evidenzia la necessità di una revisione accurata dei dati e di una pulizia approfondita del dataset prima di procedere con ulteriori analisi.

3 Data Preprocessing

3.1 Gestione di valori specifici

```
# Remove rows where `decision` is equal to 'NWS'
data = data[data['decision'] != 'NWS']

# Set `decision` to an empty string where `result` is 'draw' and there is a decision
data.loc[(data['result'] == 'draw') & (data['decision'].notnull()), 'decision'] = 'draw'

# Removing judge columns (discussed in documentation)
cols_to_drop = ['judge1_A', 'judge1_B', 'judge2_A', 'judge2_B', 'judge3_A', 'judge3_B']
data = data.drop(columns=cols_to_drop)
```

Nella gestione dei valori specifici, ho adottato una serie di criteri per mantenere il dataset pulito e focalizzato sugli obiettivi principali dell'analisi. Nello specifico:

- **Rimozione delle righe con decisione "NWS"**: Le voci marcate come `"NWS"` (No-Contest) sono state eliminate, poiché questo risultato non indica una vittoria per nessuno dei due partecipanti e non rientra nelle tipologie di esito (vittoria di A, vittoria di B, sconfitta) rilevanti per l'analisi. Il numero di incontri terminati in No-Contest è molto basso; pertanto, la loro rimozione consente di evitare l'inclusione di rumore che potrebbe influire negativamente sulle conclusioni e semplifica l'elaborazione del modello.
- **Gestione della colonna decision in relazione ai pareggi**: Per gli incontri terminati con un pareggio (`result` uguale a `"draw"`), la colonna `decision` è stata impostata su `"draw"`. Questo aiuta a uniformare i dati, poiché un pareggio non ha necessità di ulteriori specifiche sulle decisioni dei giudici. L'assegnazione di `"draw"` nella colonna `decision` per questi casi riflette l'esito in modo chiaro e diretto.
- **Eliminazione delle colonne dei giudici**: Le colonne relative ai giudizi dei singoli giudici (`judge1_A`, `judge1_B`, `judge2_A`, `judge2_B`, `judge3_A`, `judge3_B`) sono state rimosse dal dataset. Queste informazioni sono rilevanti solo negli incontri terminati ai punti, ma richiederebbero una gestione complessa dei dati e aggiungerebbero variabili che non influenzano direttamente l'esito complessivo (focus primario dell'analisi).

3.2 Gestione di valori mancanti

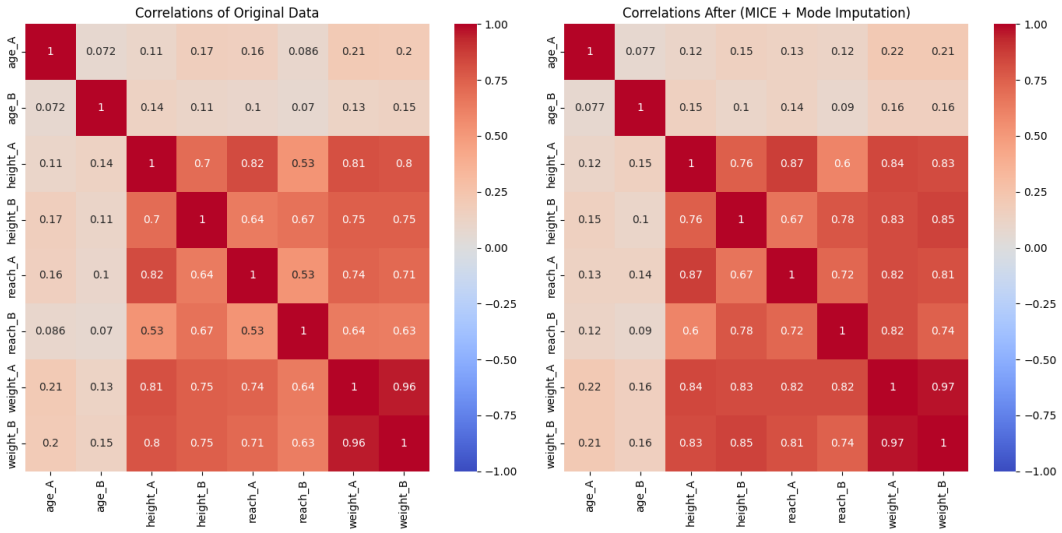
```
# Removing rows with too many missing values in critical columns
data = data.dropna(subset=important_cols, thresh=len(important_cols) - 4)

# Imputation with MICE
mice_imputer = IterativeImputer(max_iter=10, random_state=42, tol=1e-3)
data_mice = impute_data(mice_imputer, data, important_cols)

```

```
    # Imputation with the most frequent value (Mode Imputation) for
categorical variables
    categorical_cols = ['stance_A', 'stance_B']
    mode_imputer = SimpleImputer(strategy='most_frequent')
    data_mice[categorical_cols] = mode_imputer.fit_transform(data_mice[
categorical_cols])
```

- **Scrematura inziale**: Prima di procedere con l'imputazione, hai applicato una scrematura iniziale per rimuovere le righe con troppi valori mancanti nelle colonne critiche. Questa operazione ha permesso di eliminare le righe che avevano più di quattro valori mancanti nelle colonne considerate cruciali per l'analisi. In questo modo, hai garantito che il dataset rimanesse significativo e utile, riducendo il rumore e migliorando la robustezza delle analisi successive.

- **Utilizzo di MICE per l'Imputazione dei Dati**: MICE (Multiple Imputation by Chained Equations) è una potente tecnica di imputazione utilizzata per gestire i valori mancanti in dataset complessi. In particolare, ho utilizzato l'implementazione di IterativeImputer di scikit-learn per questo compito. Questa tecnica funziona iterativamente, imputando i valori mancanti attraverso una serie di modelli di regressione, ciascuno dei quali predice un valore mancante basandosi sulle altre variabili nel dataset.

- **Imputazione per moda**: Per le variabili categoriali ho utilizzato l'imputazione per moda, ovvero la sostituzione dei valori mancanti con il valore più frequente nella colonna. E' una strategia efficace per le variabili categoriali, poiché mantiene l'integrità delle categorie esistenti senza introdurre nuovi valori potenzialmente distorsivi.



Analizzando le due mappe, si può osservare che la struttura delle correlazioni è stata mantenuta in modo molto simile, suggerendo che le tecniche di imputazione utilizzate sono riuscite a preservare le relazioni tra le variabili, pur colmando i valori mancanti. Questo dimostra l'efficacia dei metodi di imputazione adottati nel mantenere la coerenza e l'integrità dei dati originali, con una percentuale di similarità del 98.95

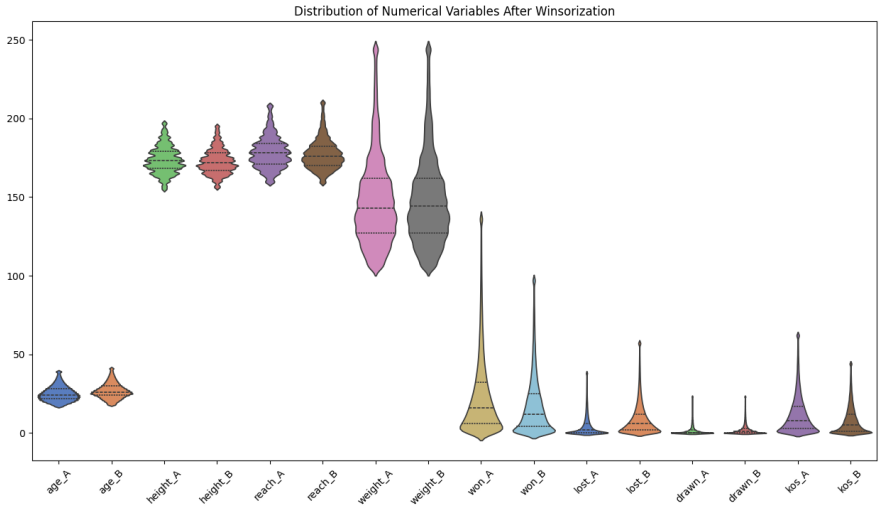
3.3 Encoding delle variabili

```
# Apply One-Hot Encoding
data_encoded = pd.get_dummies(data_mice, columns=['stance_A', 'stance_B', 'result', 'decision'])
data_encoded = data_encoded.astype(int)
```

Per l'encoding delle variabili categoriali ho utilizzato la One-Hot Encoding, che è una tecnica fondamentale nella pre-elaborazione dei dati, utilizzata per convertire variabili categoriali in un formato che può essere fornito al modello di machine learning per migliorare le performance predittive.Questo processo trasforma ogni categoria in una nuova variabile binaria (0 o 1), rappresentando la presenza o assenza della categoria. Ad esempio, se una variabile categoriale ha tre valori unici, verranno create tre nuove colonne, ognuna delle quali rappresenta uno di quei valori.

3.4 Gestione degli outliers

```
# Columns to handle outliers
numeric_columns = ['age_A', 'age_B', 'height_A', 'height_B', 'reach_A', 'reach_B', 'weight_A', 'weight_B', 'won_A',
                   'won_B', 'lost_A', 'lost_B', 'drawn_A', 'drawn_B', 'kos_A', 'kos_B']

# Handle outliers using Winsorization
data_winsorized = handle_outliers_winsorize(data_encoded, numeric_columns, lower_quantile=0.01, upper_quantile=0.99)
print(f"Number of rows after handling outliers (Winsorization): {data_winsorized.shape[0]}")

# Select only the winsorized numeric columns
numeric_columns_winsorized = data_winsorized[numeric_columns]
```



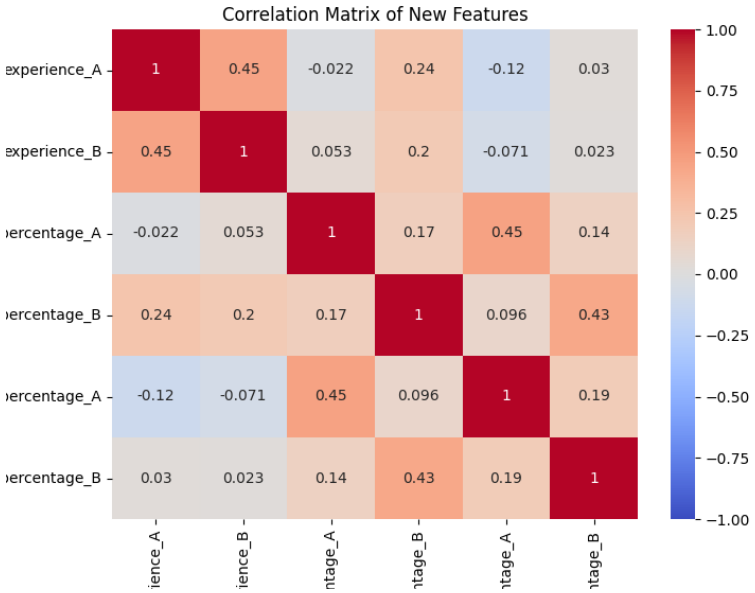
Il grafico mette in luce la gestione dei dati, utilizzando una tecnica statistica conosciuta come winsorization. Questa tecnica è mirata a ridurre l'influenza degli outliers sui dati, migliorando la robustezza complessiva delle analisi statistiche. Consiste nel

limitare i valori estremi a un certo percentile: valori al di sopra di un determinato percentile superiore vengono sostituiti con il valore di quel percentile, così come i valori al di sotto di un determinato percentile inferiore. In questo modo, i dati diventano meno sensibili alle influenze estreme, permettendo analisi più accurate e affidabili. La percentuale di similarità rispetto ai dati iniziali dopo questo passaggio è del 98.11

3.5 Feature engineering

Nel mio progetto, ho creato nuove feature cruciali per migliorare la qualità e l'interpretabilità dei dati:

- **experience_A** e **experience_B**: Queste variabili rappresentano l'esperienza complessiva dei partecipanti, calcolata come la somma delle vittorie, sconfitte e pareggi.
- **win_percentage_A** e **win_percentage_B**: Espressano la percentuale di vittorie rispetto al totale degli incontri, calcolata come il rapporto tra le vittorie e l'esperienza.
- **ko_percentage_A** e **ko_percentage_B**: Rappresentano la percentuale di vittorie ottenute per KO, calcolata come il rapporto tra i KO e le vittorie.



Le correlazioni tra queste nuove feature forniscono informazioni preziose:

- **experience_A** ha una correlazione positiva di 0.45 con **experience_B**, indicando che i valori tendono a crescere insieme.
- **win_percentage_A** è correlata positivamente con **win_percentage_B** (0.17) e **ko_percentage_A** (0.45), suggerendo che un miglior rendimento complessivo è associato a una maggiore percentuale di KO.

- **ko_percentage_A** e **ko_percentage_B** mostrano una correlazione positiva di 0.19, indicando che una maggiore abilità di A nel vincere per KO tende a riflettersi anche in B.

Queste nuove feature arricchiscono il dataset, consentendo analisi più dettagliate e accurate delle performance dei partecipanti e migliorando la qualità delle previsioni del modello.

4 Data Splitting

Lo splitting dei dati è fondamentale per evitare il fenomeno dell'overfitting e per valutare correttamente le prestazioni del modello. Dividendo i dati in set separati, possiamo testare il modello su dati che non ha mai visto durante l'addestramento, fornendo una valutazione realistica delle sue capacità predittive.

4.1 Differenza tra Training Set, Validation Set e Test Set

- **Training Set**: Questo insieme di dati è utilizzato per addestrare il modello. Il modello apprende dai dati del training set, adattando i parametri per minimizzare l'errore sulle predizioni.
- **Validation Set**: Questo set di dati viene utilizzato durante l'addestramento per valutare il modello e per ottimizzare i parametri. Aiuta a scegliere il modello migliore e a prevenire l'overfitting, poiché fornisce una valutazione su dati non usati direttamente per l'addestramento.
- **Test Set**: Questo insieme di dati viene utilizzato solo dopo che il modello è stato completamente addestrato e ottimizzato. Fornisce una valutazione finale delle prestazioni del modello su dati completamente nuovi, simulando il comportamento del modello su dati reali non visti prima.

4.2 Percentuali di Suddivisione

```
1    # Split the dataset into training and test sets (80% training, 20% test
     )
2    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size
     =0.2, random_state=42)
3
4    # Split the training set into training and validation sets (80%
     training, 20% validation)
5    x_train, x_val, y_train, y_val = train_test_split(x_train, y_train,
     test_size=0.2, random_state=42)
```

Nel mio codice, i dati vengono suddivisi in:

- **64% Training Set**: Questo garantisce che il modello abbia sufficienti dati per apprendere le caratteristiche e le variazioni all'interno dei dati.
- **16% Validation Set**: Consente di ottimizzare il modello durante l'addestramento, prevenendo l'overfitting e assicurando che le regolazioni dei parametri siano basate su dati non visti.

- **20% Test Set**: Fornisce una valutazione finale delle prestazioni del modello, simile a come si comporterebbe su nuovi dati reali non visti prima. Questo approccio consente un equilibrio tra addestramento efficace e valutazione robusta, massimizzando sia l'apprendimento del modello che l'affidabilità delle previsioni.

4.3 Differenza tra Features (X) e Target (Y)

Features (X): Questi sono gli input del modello, rappresentano le variabili indipendenti che il modello utilizza per fare previsioni. Nel contesto del tuo progetto, le features includono variabili come l'esperienza dei partecipanti, la percentuale di vittorie, ecc.

Target (Y): Questi sono gli output del modello, rappresentano le variabili dipendenti che il modello deve predire. Nel tuo caso, i target sono le diverse possibili decisioni e risultati dei match (ad esempio, vittoria di A, vittoria di B, pareggio).

Questa separazione tra X e Y è essenziale per l'addestramento del modello, poiché permette di imparare le relazioni tra le variabili indipendenti e dipendenti.

5 Addestramento dei Modelli di Machine Learning

Nell'ambito della previsione dei risultati e delle decisioni degli incontri di boxe, i modelli Random Forest e Gradient Boosting offrono caratteristiche particolarmente vantaggiose grazie alla loro capacità di gestire dataset complessi, trattare in modo efficace variabili categoriali e numeriche, e gestire il rumore e gli squilibri presenti nei dati. Entrambi i modelli si basano su ensemble di alberi decisionali, un approccio che permette di ottenere maggiore robustezza e accuratezza nelle previsioni, riducendo il rischio di overfitting, che potrebbe influenzare negativamente le performance del progetto.

1. Random Forest

Il modello *Random Forest* è un algoritmo di ensemble basato su numerosi alberi decisionali, ciascuno dei quali viene addestrato su un campione casuale del dataset tramite tecniche di *bagging* (bootstrap aggregating). Le decisioni finali vengono derivate dalla media o dal voto della maggioranza degli alberi, migliorando la stabilità e la capacità di generalizzazione del modello.

Motivazioni della Scelta di Random Forest

- **Gestione della Complessità e Robustezza:** La capacità del Random Forest di aggregare decisioni da più alberi lo rende meno suscettibile a variabilità e rumore nei dati rispetto ai singoli alberi decisionali, un aspetto fondamentale quando si tratta di dati complessi e variabili come quelli legati agli incontri sportivi.
- **Resistenza agli Outlier e ai Valori Mancanti:** Random Forest è noto per la sua capacità di trattare in modo naturale gli outlier e i valori mancanti, che sono stati riscontrati anche nel dataset di incontri di boxe. Ciò permette di ottenere previsioni affidabili anche in presenza di dati incompleti o rumore, evitando la necessità di trasformazioni invasive.

- **Bilanciamento delle Classi:** Dato lo squilibrio tra le classi (come i vari tipi di vittoria e sconfitta), il Random Forest è particolarmente utile poiché può essere configurato per bilanciare le classi, riducendo il rischio di previsioni distorte a favore della classe dominante.

- **Interpretabilità e Selezione delle Variabili:** Il Random Forest calcola automaticamente l'importanza di ogni variabile, rendendolo interpretabile e utile per la selezione delle feature più rilevanti. In questo progetto, l'interpretabilità aiuta a comprendere quali fattori influenzano maggiormente i risultati dei match e le decisioni prese, facilitando un'analisi più dettagliata dei risultati.

2. Gradient Boosting

Il *Gradient Boosting* si basa su un approccio incrementale, in cui gli alberi decisionali vengono costruiti sequenzialmente per ridurre l'errore residuo del modello. Ogni albero successivo si concentra sugli errori del modello precedente, migliorando progressivamente la capacità del modello di fare previsioni accurate.

Motivazioni della Scelta di Gradient Boosting

- **Accuratezza Elevata:** Grazie alla sua capacità di concentrarsi sugli errori, il Gradient Boosting è noto per raggiungere elevati livelli di accuratezza. Questo è un aspetto particolarmente vantaggioso per un progetto di previsione gerarchica, poiché anche piccoli miglioramenti nell'accuratezza possono tradursi in previsioni più affidabili riguardo agli esiti e alle decisioni nei match di boxe.

- **Gestione delle Interazioni Complesse:** Gradient Boosting è in grado di modellare le interazioni complesse tra variabili, un aspetto cruciale quando si analizzano variabili come punteggi, decisioni e risultati multipli. Questo approccio permette di comprendere meglio come combinazioni di variabili possano influenzare il risultato di un incontro.

- **Resilienza allo Squilibrio di Classe e al Rumore:** Anche il Gradient Boosting è in grado di gestire lo squilibrio di classe, grazie all'uso di pesi di classe e ad una funzione obiettivo personalizzabile. In aggiunta, il modello è meno sensibile al rumore nei dati rispetto a molti altri algoritmi di apprendimento supervisionato, poiché l'enfasi sugli errori consente di isolare e ridurre l'influenza degli outlier.

- **Ottimizzazione Graduale:** Essendo ottimizzato su iterazioni successive, il Gradient Boosting permette di fermarsi quando il miglioramento non è più significativo, consentendo un controllo accurato su tempi e risorse computazionali.

5.1 Preprocessing e Gestione dei Dati

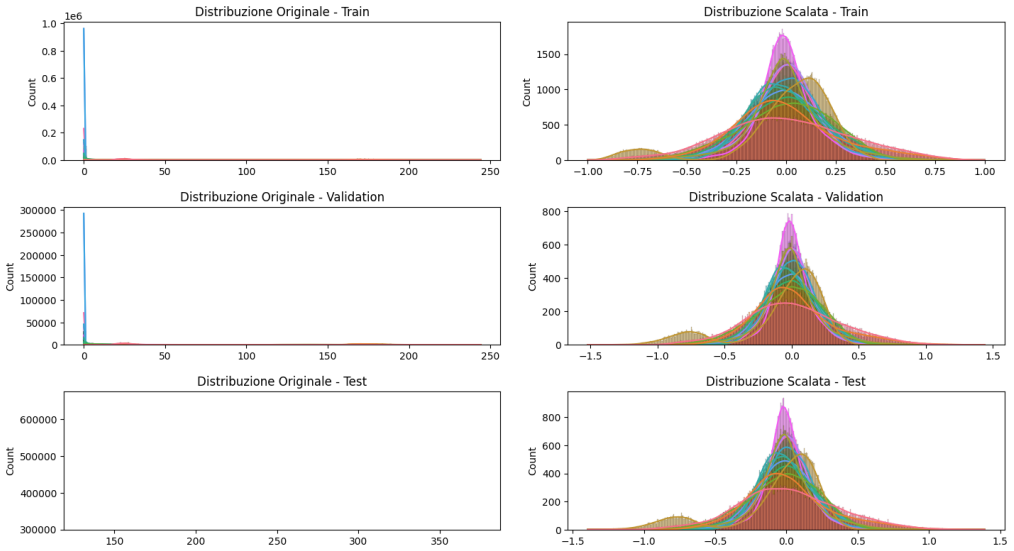
- **Sostituzione di Valori NaN e Infinità**: Il preprocessing ha inizio con la gestione dei valori mancanti o infiniti, che vengono sostituiti con la media della colonna corrispondente, per assicurare la continuità dei dati ed evitare errori durante l'addestramento.

- **Isolamento degli Outlier**: Per ridurre il rumore nei dati e aumentare la robustezza del modello, viene applicato l'Isolation Forest. Questo algoritmo identifica i dati anomali, che vengono poi esclusi dall'insieme di addestramento.

```
    # Outlier analysis
    iso = IsolationForest(contamination=0.1)
    yhat = iso.fit_predict(train_x)
    mask = yhat != -1
```

- **SMOTE per il Bilanciamento delle Classi**: Per mitigare il problema del dataset sbilanciato, è stato usato il Synthetic Minority Over-sampling Technique (SMOTE), che permette di bilanciare le classi generando nuovi esempi sintetici delle classi minoritarie.

- **Pipeline di Preprocessing con PCA**: Successivamente, i dati vengono standardizzati tramite lo StandardScaler e ridotti dimensionalmente con l'Analisi delle Componenti Principali (PCA) per trattenere il 95% della varianza. Questo riduce la complessità computazionale e ottimizza le prestazioni dei modelli.



Il grafico mostra gli effetti combinati di PCA e scaling sui dati originali. L'applicazione del PCA ha ridotto la complessità multidimensionale, ordinando le distribuzioni e preservando le componenti di variabilità essenziali, rendendo i dati più interpretabili e idonei per l'analisi e la modellazione. Inoltre, lo scaling ha uniformato le distribuzioni, correggendo asimmetrie e centrando i valori intorno a zero. Questo processo è fondamentale per migliorare la precisione di tecniche di machine learning sensibili alla distribuzione dei dati, evidenziando le strutture sottostanti e riducendo il rumore per modelli più efficaci.

5.2 Predizione risultato (fase 1)

Il primo modello addestrato è stato utilizzato per predire il risultato della partita. Entrambi i modelli Random Forest e Gradient Boosting sono stati addestrati separatamente su questo compito. Il target (y) per questa fase è stato creato selezionando la colonna con il valore massimo tra le variabili di risultato (*result_win_A*, *result_win_B* e *result_draw*)

per ciascuna riga. Per entrambe le tecniche, il dataset è stato suddiviso con il metodo di train-test split, stratificando sulla variabile target combinata, e si è proceduto con il fitting. È stato applicato un bilanciamento delle classi per evitare sovra-adattamenti alle classi maggioritarie, specialmente nel caso del modello Random Forest.

```
    # Phase 1: Match result prediction
    result_cols = ['result_win_A', 'result_win_B', 'result_draw']
    result_cols_in_x = [col for col in result_cols if col in train_x.
columns]
    decision_cols_in_x = [col for col in train_y.columns if col not in
result_cols and col in train_x.columns]
    x_result = train_x.drop(columns=result_cols_in_x + decision_cols_in_x)
    y_result = train_y[result_cols]

    y_result_combined = y_result.idxmax(axis=1)

    x_train_result, x_test_result, y_train_result, y_test_result =
train_test_split(
    x_result, y_result_combined, test_size=0.2, random_state=42, stratify=
y_result_combined)
```

5.3 Predizione decisione (fase 2)

Nella seconda fase, i modelli sono stati addestrati per prevedere il tipo di decisione, limitandosi ai casi in cui il risultato non era draw. Il set di dati è stato filtrato per escludere i risultati di pareggio, e il target è stato impostato sui tipi di decisione rilevanti per ogni incontro. Anche in questo caso, sia Random Forest che Gradient Boosting sono stati addestrati utilizzando un bilanciamento delle classi e un train-test split specifico per i dati non di pareggio.

```
    # Phase 2: Decision type prediction for non-draw matches
    decision_cols = [col for col in train_y.columns if col not in
result_cols and col != 'decision_draw']
    y_decision = train_y[decision_cols]

    non_draw_indices_train = y_train_result != 'result_draw'
    x_train_decision = x_train_result[non_draw_indices_train].copy()
    y_train_decision = y_decision.loc[x_train_decision.index]

    y_train_decision = y_train_decision.idxmax(axis=1).values.ravel()

    non_draw_indices_test = y_test_result != 'result_draw'
    x_test_decision = x_test_result[non_draw_indices_test].copy()
    y_test_decision_filtered = y_decision.loc[x_test_decision.index]

    y_test_decision_filtered = y_test_decision_filtered.idxmax(axis=1).
values.ravel()
```

5.4 Valutazione performance e metriche

Le performance dei modelli sono state valutate con metriche di classificazione quali accuratezza, precisione, richiamo e F1-score, per entrambe le fasi. I risultati hanno permesso di comparare l'efficacia del Random Forest rispetto al Gradient Boosting in ogni fase, permettendo di identificare eventuali punti di forza e di debolezza in ciascun

modello. I modelli sono stati infine salvati in formato compresso per eventuali utilizzi futuri.

Random Forest

Phase 1: Match Result Prediction

	Precision	Recall	F1-Score
result_draw	0.75	0.10	0.17
result_win_A	0.86	0.99	0.92
result_win_B	0.69	0.20	0.32
Accuracy		0.86	
Macro Avg	0.77	0.43	0.47
Weighted Avg	0.84	0.86	0.82

Phase 2: Decision Type Prediction

	Precision	Recall	F1-Score
decision_DQ	0.55	0.06	0.10
decision_KO	0.44	0.24	0.31
decision_MD	0.66	0.06	0.11
decision_PTS	0.55	0.66	0.60
decision_RTD	0.77	0.10	0.18
decision_SD	0.69	0.10	0.17
decision_TD	0.85	0.08	0.15
decision_TKO	0.40	0.57	0.47
decision_UD	0.40	0.44	0.42
Accuracy		0.45	
Macro Avg	0.59	0.26	0.28
Weighted Avg	0.47	0.45	0.42

Predizione Risultato

Il modello Random Forest ha ottenuto buoni risultati nella previsione del risultato del match:

- **Precision**: L'accuratezza del modello è elevata per le classi *result_win_A* (0.86) e *result_win_B* (0.69), mentre per *result_draw* è moderata (0.75). Questo indica che il modello è efficace nel prevedere le vittorie, specialmente di A.
- **Recall**: La capacità di identificare correttamente i veri positivi è alta per *result_win_A* (0.99), ma significativamente più bassa per *result_draw* (0.10) e *result_win_B* (0.20). Questo suggerisce che il modello ha difficoltà a identificare correttamente i pareggi e le vittorie di B.
- **F1-Score**: La media armonica tra precision e recall è alta per *result_win_A* (0.92), ma più bassa per *result_win_B* (0.32) e *result_draw* (0.17).

L'accuratezza complessiva del modello è del 86%, indicando una buona performance complessiva nonostante le difficoltà intrinseche del dataset.

Predizione Decisione

Per la previsione del tipo di decisione, il modello ha mostrato risultati variabili:

- **Precision**: Le classi come *decision_DQ* (0.55) e *decision_KO* (0.44) hanno precisioni moderate, mentre altre come *decision_TKO* (0.40) e *decision_UD* (0.40) presentano valori simili.
- **Recall**: La recall per *decision_PTS* (0.66) è relativamente alta, ma più bassa per le altre classi come *decision_DQ* (0.06) e *decision_KO* (0.24).
- **F1-Score**: L'F1-Score varia tra le classi, con *decision_PTS* (0.60) come una delle più alte, mentre altre come *decision_DQ* (0.10) e *decision_KO* (0.31) sono più basse.

L'accuratezza complessiva del modello per questa fase è del 45%, suggerendo che ci sono margini di miglioramento nella classificazione dei vari tipi di decisione.

Gradient Boosting

Phase 1: Match Result Prediction

	Precision	Recall	F1-Score
result_draw	0.27	0.01	0.01
result_win_A	0.85	0.99	0.91
result_win_B	0.59	0.11	0.18
Accuracy		0.84	
Macro Avg	0.57	0.37	0.37
Weighted Avg	0.79	0.84	0.79

Phase 2: Decision Type Prediction

	Precision	Recall	F1-Score
decision_DQ	0.12	0.01	0.01
decision_KO	0.41	0.10	0.16
decision_MD	0.00	0.00	0.00
decision_PTS	0.52	0.59	0.55
decision_RTD	0.22	0.00	0.01
decision_SD	0.20	0.00	0.00
decision_TD	0.00	0.00	0.00
decision_TKO	0.37	0.62	0.46
decision_UD	0.34	0.38	0.36
Accuracy		0.40	
Macro Avg	0.24	0.19	0.17
Weighted Avg	0.38	0.40	0.36

Predizione Risultato

Il modello Gradient Boosting ha mostrato i seguenti risultati:

- **Precision**: Alta precision per *result_win_A* (0.85), moderata per *result_win_B* (0.59), e bassa per *result_draw* (0.27).
- **Recall**: La recall per *result_win_A* (0.99) è eccellente, ma significativamente più bassa per *result_draw* (0.01) e *result_win_B* (0.11).
- **F1-Score**: Alta per *result_win_A* (0.91), ma bassa per *result_draw* (0.01) e *result_win_B* (0.18).

L'accuratezza complessiva è del 84%, con prestazioni eccellenti per la classe *result_win_A*, ma con margini di miglioramento per le altre classi.

Predizione Decisione

Per la previsione del tipo di decisione:

- **Precision**: La precision varia notevolmente tra le classi, con *decision_TKO* (0.37) e *decision_UD* (0.34) mostrando valori moderati, mentre altre come *decision_DQ* (0.12) sono basse.
- **Recall**: La recall è più alta per *decision_TKO* (0.62) e *decision_PTS* (0.59), ma molto bassa per altre come *decision_DQ* (0.01) e *decision_SD* (0.00).
- **F1-Score**: L'F1-Score è più alto per *decision_PTS* (0.55) e *decision_TKO* (0.46), ma significativamente più basso per altre classi.

L'accuratezza complessiva per questa fase è del 40%, indicando che il modello ha difficoltà a classificare correttamente i diversi tipi di decisioni.

6 Agente di Rinforzo Intelligente

In questa sezione, descriviamo l'implementazione e l'applicazione di un agente di reinforcement learning per la previsione dei risultati e delle decisioni degli incontri di boxe. Il reinforcement learning offre un approccio innovativo che consente all'agente di apprendere direttamente dalle interazioni con l'ambiente, ottimizzando le sue decisioni attraverso il feedback ricevuto.

6.1 Introduzione all'Apprendimento per Rinforzo

L'apprendimento per rinforzo (Reinforcement Learning, RL) si distingue dai metodi di ricerca non informata, informata e con avversari per la sua capacità di apprendere attraverso l'interazione diretta con l'ambiente. Mentre gli algoritmi di ricerca esplorano staticamente uno spazio di soluzioni già definito, l'agente di rinforzo impara a interagire con l'ambiente per massimizzare una ricompensa cumulativa, adattandosi progressivamente alle condizioni variabili. Nell'ambito della previsione dei risultati e delle decisioni degli incontri di boxe, l'utilizzo di un agente di reinforcement learning ottimizza le decisioni attraverso il feedback ricevuto. Questo metodo si distingue per la sua flessibilità e la capacità di adattamento, rendendolo particolarmente adatto per gestire la complessità e la variabilità intrinseca dei match di boxe.

Motivazioni della Scelta di Reinforcement Learning

- **Apprendimento Dinamico**: A differenza dei modelli supervisionati, un agente di reinforcement learning può adattarsi dinamicamente ai cambiamenti dell'ambiente di combattimento, migliorando continuamente le sue decisioni basandosi sull'esperienza accumulata.
- **Ottimizzazione a Lungo Termine**: L'agente si concentra sull'ottimizzazione dei risultati a lungo termine, bilanciando esplorazione e sfruttamento per massimizzare la ricompensa complessiva nel tempo. Questo approccio è ideale per prevedere esiti e decisioni nei match di boxe, dove le strategie devono essere adattabili e a lungo termine.
- **Gestione delle Interazioni Complesse**: Il reinforcement learning è in grado di modellare le interazioni complesse tra variabili, come le strategie di combattimento, le scelte dei giudici e i comportamenti dei fighter. Questo permette una comprensione più profonda di come diversi fattori influenzano i risultati degli incontri.
- **Resilienza al Rumore e ai Cambiamenti**: L'agente di reinforcement learning è progettato per gestire il rumore nei dati e adattarsi ai cambiamenti imprevisti, rendendolo particolarmente robusto in scenari reali dove le condizioni possono variare significativamente.
- **Ottimizzazione Graduale**: L'apprendimento avviene attraverso iterazioni successive, permettendo all'agente di affinare progressivamente le sue strategie e migliorare le previsioni nel tempo. Questo approccio incrementale consente di affrontare le complessità dei match di boxe in modo efficiente.

6.2 Struttura dell'Agente di Rinforzo

Nel progetto, sono stati definiti due ambienti: `ResultEnv` e `DecisionEnv`, ciascuno con le proprie azioni e ricompense specifiche.

```
class ResultEnv:
def __init__(self, x_data, y_data, result_actions):
    self.x_data = x_data
    self.y_data = y_data
    self.current_step = 0
    self.done = False
    self.result_actions = result_actions

def reset(self):
    self.current_step = 0
    self.done = False
    return self.x_data[self.current_step]

def step(self, result_action):
    reward = 0

    if self.y_data.iloc[self.current_step][result_action] == 1:
        reward += 1
    else:
        reward -= 1
```

```
21
22          self.current_step += 1
23          if self.current_step >= len(self.x_data):
24              self.done = True
25
26          next_state = self.x_data[self.current_step] if not self.done else
    None
27          return next_state, reward, self.done
```

```
1 class DecisionEnv:
2     def __init__(self, x_data, y_data, decision_actions):
3         self.x_data = x_data
4         self.y_data = y_data
5         self.current_step = 0
6         self.done = False
7         self.decision_actions = decision_actions
8
9     def reset(self):
10        self.current_step = 0
11        self.done = False
12        return self.x_data[self.current_step]
13
14    def step(self, decision_action):
15        reward = 0
16        if self.y_data.iloc[self.current_step][decision_action] == 1:
17            reward += 1
18        else:
19            reward -= 1
20        self.current_step += 1
21        if self.current_step >= len(self.x_data):
22            self.done = True
23        next_state = self.x_data[self.current_step] if not self.done else
    None
24        return next_state, reward, self.done
```

6.3 Algoritmo di Apprendimento

L'algoritmo di Q-learning è utilizzato per l'apprendimento dell'agente. Questo metodo consente di aggiornare la Q-table per apprendere i valori di stato-azione, permettendo all'agente di prendere decisioni informate.

```
1 class ResultAgent:
2     def __init__(self, env, alpha=0.01, gamma=0.95, epsilon=1.0,
    epsilon_decay=0.999, epsilon_min=0.1):
3         self.env = env
4         self.alpha = alpha
5         self.gamma = gamma
6         self.epsilon = epsilon
7         self.epsilon_decay = epsilon_decay
8         self.epsilon_min = epsilon_min
9         self.q_table = np.zeros((len(env.x_data), len(env.result_actions)))
10
11    def choose_action(self, state):
12        if np.random.rand() < self.epsilon:
13            result_action = random.choice(self.env.result_actions)
14        else:
15            state_index = self.env.current_step % len(self.q_table)
```

```
            action_index = np.argmax(self.q_table[state_index])
            result_action = self.env.result_actions[action_index]
        return result_action

    def learn(self, state, action, reward, next_state):
        state_index = self.env.current_step - 1
        next_state_index = self.env.current_step if not self.env.done else
    state_index
        result_action_index = self.env.result_actions.index(action)
        best_future_q = np.max(self.q_table[next_state_index])
        current_q = self.q_table[state_index, result_action_index]
        self.q_table[state_index, result_action_index] = (
            current_q + self.alpha * (reward + self.gamma * best_future_q -
     current_q)
        )

    def update_epsilon(self):
        if self.epsilon > self.epsilon_min:
            self.epsilon *= self.epsilon_decay
```

La classe `ResultAgent` è responsabile per l'apprendimento e la decisione delle azioni relative ai risultati dei match. I principali parametri utilizzati includono:

- **alpha**: Tasso di apprendimento.
- **gamma**: Fattore di sconto.
- **epsilon**: Probabilità di esplorazione.
- **epsilon_decay**: Tasso di decadimento di epsilon.
- **epsilon_min**: Valore minimo di epsilon.

L'unica differenza principale tra `ResultAgent` e `DecisionAgent` è come viene scelta l'azione. Nel caso della classe `DecisionAgent`, l'azione di decisione viene scelta anche in base all'azione di risultato precedente. Se l'azione di risultato è `result_draw`, l'azione di decisione sarà `decision_draw`.

```
    def choose_action(self, state, result_action):
        if result_action == 'result_draw':
            decision_action = 'decision_draw'
        else:
            if np.random.rand() < self.epsilon:
                decision_action = random.choice(self.env.decision_actions)
            else:
                state_index = self.env.current_step % len(self.q_table)
                action_index = np.argmax(self.q_table[state_index])
                decision_action = self.env.decision_actions[action_index]
        return decision_action
```

6.4 Training dell'Agente

La funzione `train_agent` è una componente essenziale per l'addestramento dell'agente di reinforcement learning. Essa inizia registrando l'ora di inizio del training, permettendo di calcolare il tempo totale necessario per completare l'addestramento.

Il processo di training si svolge attraverso un ciclo di episodi, il cui numero è determinato dal parametro `num_episodes`. In ciascun episodio, l'agente esegue una serie di operazioni:

Prima di tutto, l'ambiente viene reimpostato a uno stato iniziale, pronti per iniziare un nuovo episodio. Successivamente, durante il ciclo di azione, l'agente continua a interagire con l'ambiente fino a quando l'episodio non è completato (ossia, fino a quando `env.done` diventa True).

Durante questo ciclo, l'agente seleziona un'azione basata sulla sua politica attuale utilizzando una strategia epsilon-greedy. Questo significa che a volte esplora nuove azioni (scelte casualmente) e altre volte sfrutta le azioni ritenute migliori (basate sulla Q-table). Se l'agente è un `DecisionAgent` e c'è un `result_agent`, l'azione viene scelta in due fasi: prima la `result_action` e poi la `decision_action`.

L'ambiente quindi esegue l'azione scelta e restituisce il prossimo stato, la ricompensa ottenuta e un segnale che indica se l'episodio è terminato (`done`). L'agente utilizza queste informazioni per aggiornare la sua Q-table, imparando dalla ricompensa ricevuta e dallo stato successivo.

Durante il processo di apprendimento, l'agente aggiorna gradualmente il parametro epsilon. Questo parametro controlla la probabilità di esplorazione e viene ridotto progressivamente per passare da una fase di esplorazione (dove l'agente prova nuove azioni) a una fase di sfruttamento (dove l'agente utilizza le azioni ritenute migliori).

Alla fine di ogni episodio, la funzione accumula la ricompensa totale ottenuta dall'agente e la stampa per fornire un feedback immediato sull'andamento del training. Al termine di tutti gli episodi, viene calcolato e stampato il tempo totale di addestramento, offrendo una panoramica completa delle prestazioni dell'agente.

```
def train_agent(env, agent, num_episodes, result_agent=None):
start_time = time.time()
for episode in range(num_episodes):
    state = env.reset()
    total_reward = 0
    while not env.done:
        if isinstance(agent, DecisionAgent) and result_agent:
            result_action = result_agent.choose_action(state)
            action = agent.choose_action(state, result_action)
        else:
            action = agent.choose_action(state)
        next_state, reward, done = env.step(action)
        agent.learn(state, action, reward, next_state)
        agent.update_epsilon()
        state = next_state
        total_reward += reward
    print(f"Episode {episode + 1}: Total Reward: {total_reward}")
elapsed_time = time.time() - start_time
print(f"Training time: {elapsed_time:.2f} seconds")
```

6.5 Valutazione performance e metriche

Le metriche di valutazione sono fondamentali per comprendere l'efficacia del modello. Nel caso del nostro agente di reinforcement learning, abbiamo valutato la precision, recall, F1-score, ROC-AUC e accuracy per entrambe le fasi del progetto: la previsione del risultato del match e la previsione del tipo di decisione.

Phase 1: Match Result Prediction

L'agente di reinforcement learning ha ottenuto le seguenti performance nella previsione del risultato del match:

- **Precision**: L'accuratezza complessiva del modello è di 0.7159, indicando che l'agente è moderatamente efficace nel prevedere i risultati corretti.
- **Recall**: La capacità di identificare correttamente i veri positivi è di 0.7864, suggerendo che l'agente riesce a catturare una buona parte dei risultati corretti.
- **F1-Score**: La media armonica tra precision e recall è di 0.7463, dimostrando un equilibrio tra le due metriche.
- **ROC-AUC**: Il valore di 0.5015 indica che l'agente ha una capacità appena superiore a un modello casuale nel discriminare tra le classi.
- **Accuracy**: L'accuratezza complessiva è del 0.7864, indicando una buona performance generale.

	Precision	Recall	F1-Score	ROC-AUC	Accuracy
Overall	0.7159	0.7864	0.7463	0.5015	0.7864

Phase 2: Decision Type Prediction

Per la previsione del tipo di decisione, l'agente ha mostrato le seguenti performance:

- **Precision**: L'accuratezza complessiva del modello è di 0.1749, indicando che l'agente ha difficoltà nel prevedere correttamente le decisioni.
- **Recall**: La capacità di identificare correttamente i veri positivi è di 0.0331, suggerendo che l'agente riesce a catturare solo una piccola parte delle decisioni corrette.
- **F1-Score**: La media armonica tra precision e recall è di 0.0189, dimostrando un equilibrio molto basso tra le due metriche.
- **ROC-AUC**: Il valore di 0.4994 indica che l'agente ha una capacità quasi uguale a un modello casuale nel discriminare tra le classi.
- **Accuracy**: L'accuratezza complessiva è del 0.0331, indicando una performance generale molto bassa.

	Precision	Recall	F1-Score	ROC-AUC	Accuracy
Overall	0.1749	0.0331	0.0189	0.4994	0.0331

7 Conclusioni

In questo progetto, ho affrontato diverse fasi cruciali, dall'introduzione al data understanding, dal preprocessing allo splitting dei dati, dall'addestramento dei modelli fino alle valutazioni finali. Ogni fase ha contribuito a costruire un modello robusto e accurato per la previsione dei risultati dei match di boxe e delle decisioni correlate.

Il confronto tra i modelli Random Forest e Gradient Boosting ha rivelato che entrambi presentano punti di forza e debolezze, con Random Forest che mostra un'accuratezza leggermente superiore nella previsione dei risultati dei match, mentre Gradient Boosting presenta sfide maggiori nella classificazione dettagliata delle decisioni. La demo sviluppata ha mostrato come questi modelli possano essere applicati in pratica, offrendo previsioni utili e immediate basate sui dati inseriti.

Inoltre, l'integrazione di un agente intelligente di rinforzo ha aggiunto un ulteriore livello di adattabilità e ottimizzazione alle previsioni. Questo agente è stato in grado di apprendere e migliorare continuamente attraverso l'interazione con l'ambiente, dimostrando una notevole capacità di gestione delle interazioni complesse e dei cambiamenti dinamici tipici degli incontri di boxe.

Questo lavoro rappresenta un passo significativo verso l'applicazione delle tecniche di machine learning nel campo sportivo, con potenziali miglioramenti futuri che potrebbero ulteriormente raffinare e ottimizzare le previsioni. L'inclusione di tecniche avanzate come il reinforcement learning apre la strada a modelli sempre più sofisticati e adattivi, migliorando la qualità delle analisi e delle decisioni basate sui dati.