Sistema di predizione calcistico

Gruppo di lavoro

Giannantonio Sanrocco, 758452, g.sanrocco1@studenti.uniba.it

Link Github: https://github.com/Giannantonio26/ICON

A.A 2023-2024

Sommario

Capitolo 0) INTRODUZIONE 3
Capitolo 1) RAGIONAMENTO LOGICO E PROLOG 7
Capitolo 2) WEB SEMANTICO 8
Capitolo 3) BAYESIAN NETWORK 9
Capitolo 4) APPRENDIMENTO SUPERVISIONATO 15
Capitolo 5) APPRENDIMENTO NON SUPERVISIONATO 25
Sviluppi Futuri 28
Riferimenti bibliografici 28

Capitolo 0) INTRODUZIONE

L'obiettivo del progetto è sfruttare le statistiche prestazionali dei calciatori per effettuare predizioni. In particolare, intendiamo effettuare:

Predizione del numero di gol: Utilizzare le statistiche P90 (per 90 minuti) dei calciatori per prevedere il numero di gol che ciascun giocatore realizzerà entro la fine della stagione in base ai minuti giocati.

Predizione del ruolo: Impiegare i dati prestazionali per determinare il ruolo di un nuovo giocatore, basandoci su caratteristiche e prestazioni simili a quelle dei calciatori esistenti.

Clustering dei giocatori: Raggruppare i calciatori in base a caratteristiche e stili di gioco simili, utilizzando le loro statistiche prestazionali per identificare gruppi omogenei.

Queste predizioni possono essere di grande supporto per le decisioni strategiche nel mondo del calcio. Analizzeremo in seguito, nel dettaglio, come queste predizioni possono essere d'aiuto nel contesto calcistico.

DATASET

Il dataset utilizzato contiene le statistiche prestazionali dei giocatori dei top 5 campionati europei (Premier League, Serie A, Liga, Bundesliga e Ligue1) della stagione 2022/2023. Nello specifico, il dataset contiene più 2500 righe e 124 colonne.
Le feature del dataset sono le seguenti:

- Rk: Posizione
- Player: Nome del giocatore
- Nation: Nazionalità del giocatore
- Pos: Ruolo
- Squad: Nome della squadra
- Comp: Campionato in cui la squadra compete
- Age: Età del giocatore
- Born: Anno di nascita
- MP: Partite giocate
- Starts: Partite iniziate come titolare
- Min: Minuti giocati
- 90s: Minuti giocati divisi per 90
- Goals: Gol segnati o subiti
- Shots: Tiri totali (esclusi i calci di rigore)
- SoT: Tiri in porta (esclusi i calci di rigore)
- SoT%: Percentuale di tiri in porta (esclusi i calci di rigore)
- G/Sh: Gol per tiro
- G/SoT: Gol per tiro in porta (esclusi i calci di rigore)
- ShoDist: Distanza media, in yard, dalla porta di tutti i tiri effettuati (esclusi i calci di rigore)
- ShoFK: Tiri da calci di punizione
- ShoPK: Calci di rigore segnati
- PKatt: Calci di rigore tentati
- PasTotCmp: Passaggi completati

- PasTotAtt: Passaggi tentati
- PasTotCmp%: Percentuale di passaggi completati
- PasTotDist: Distanza totale, in yard, che i passaggi completati hanno percorso in qualsiasi direzione
- PasTotPrgDist: Distanza totale, in yard, che i passaggi completati hanno percorso verso la porta avversaria
- PasShoCmp: Passaggi completati (passaggi tra 5 e 15 yard)
- PasShoAtt: Passaggi tentati (passaggi tra 5 e 15 yard)
- PasShoCmp%: Percentuale di passaggi completati (passaggi tra 5 e 15 yard)
- PasMedCmp: Passaggi completati (passaggi tra 15 e 30 yard)
- PasMedAtt: Passaggi tentati (passaggi tra 15 e 30 yard)
- PasMedCmp%: Percentuale di passaggi completati (passaggi tra 15 e 30 yard)
- PasLonCmp: Passaggi completati (passaggi oltre 30 yard)
- PasLonAtt: Passaggi tentati (passaggi oltre 30 yard)
- PasLonCmp%: Percentuale di passaggi completati (passaggi oltre 30 yard)
- Assists: Assist
- PasAss: Passaggi che portano direttamente a un tiro (tiri assistiti)
- Pas3rd: Passaggi completati che entrano nell'ultimo terzo del campo vicino alla porta
- PPA: Passaggi completati nell'area dei 18 yard
- CrsPA: Cross completati nell'area dei 18 yard
- PasProg: Passaggi completati che muovono la palla verso la porta avversaria di almeno 10 yard dal punto più lontano nei sei passaggi precedenti, o qualsiasi passaggio completato nell'area di rigore
- PasAtt: Passaggi tentati
- PasLive: Passaggi con palla in gioco
- PasDead: Passaggi con palla ferma
- PasFK: Passaggi tentati da calci di punizione
- TB: Passaggi completati inviati tra i difensori arretrati in uno spazio aperto
- Sw: Passaggi che percorrono più di 40 yard in larghezza del campo
- PasCrs: Cross
- TI: Rimesse laterali effettuate
- CK: Calci d'angolo
- CkIn: Calci d'angolo a rientrare
- CkOut: Calci d'angolo a uscire
- CkStr: Calci d'angolo dritti
- PasCmp: Passaggi completati
- PasOff: Fuorigioco
- PasBlocks: Bloccato dall'avversario che si trovava nel percorso
- SCA: Azioni che creano un tiro
- ScaPassLive: Passaggi con palla in gioco completati che portano a un tentativo di tiro
- ScaPassDead: Passaggi con palla ferma completati che portano a un tentativo di tiro
- ScaDrib: Dribbling riusciti che portano a un tentativo di tiro
- ScaSh: Tiri che portano a un altro tentativo di tiro
- ScaFld: Falli subiti che portano a un tentativo di tiro
- ScaDef: Azioni difensive che portano a un tentativo di tiro
- GCA: Azioni che creano un gol
- GcaPassLive: Passaggi con palla in gioco completati che portano a un gol

- GcaPassDead: Passaggi con palla ferma completati che portano a un gol
- GcaDrib: Dribbling riusciti che portano a un gol
- GcaSh: Tiri che portano a un altro tiro che segna
- GcaFld: Falli subiti che portano a un gol
- GcaDef: Azioni difensive che portano a un gol
- Tkl: Numero di giocatori contrastati
- TklWon: Contrasti in cui la squadra del contrastatore ha vinto il possesso della palla
- TklDef3rd: Contrasti nel terzo difensivo
- TklMid3rd: Contrasti nel terzo centrale
- TklAtt3rd: Contrasti nel terzo offensivo
- TklDri: Numero di dribblatori contrastati
- TklDriAtt: Numero di volte dribblato più numero di contrasti
- TklDri%: Percentuale di dribblatori contrastati
- TklDriPast: Numero di volte dribblato da un avversario
- Blocks: Numero di volte che la palla è stata bloccata stando nel suo percorso
- BlkSh: Numero di volte che un tiro è stato bloccato stando nel suo percorso
- BlkPass: Numero di volte che un passaggio è stato bloccato stando nel suo percorso
- Int: Intercettazioni
- Tkl+Int: Numero di giocatori contrastati più numero di intercettazioni
- Clr: Sgomberate
- Err: Errori che portano a un tiro avversario
- Touches: Numero di volte che un giocatore ha toccato la palla. Nota: Ricevere un passaggio, poi dribblare, poi effettuare un passaggio conta come un tocco
- TouDefPen: Tocchi nell'area di rigore difensiva
- TouDef3rd: Tocchi nel terzo difensivo
- TouMid3rd: Tocchi nel terzo centrale
- TouAtt3rd: Tocchi nel terzo offensivo
- TouAttPen: Tocchi nell'area di rigore offensiva
- TouLive: Tocchi con palla in gioco. Non include calci d'angolo, calci di punizione, rimesse laterali, calci d'inizio, calci di rinvio o calci di rigore
- ToAtt: Numero di tentativi di superare i difensori dribblando
- ToSuc: Numero di difensori superati con successo, dribblandoli
- ToSuc%: Percentuale di dribbling riusciti
- ToTkl: Numero di volte contrastato da un difensore durante un tentativo di dribbling
- ToTkl%: Percentuale di volte contrastato da un difensore durante un tentativo di dribbling
- Carries: Numero di volte che il giocatore ha controllato la palla con i piedi
- CarTotDist: Distanza totale, in yard, che un giocatore ha mosso la palla controllandola con i piedi, in qualsiasi direzione
- CarPrgDist: Distanza totale, in yard, che un giocatore ha mosso la palla controllandola con i piedi verso la porta avversaria
- CarProg: Dribbling che muovono la palla verso la porta avversaria di almeno 5 yard, o qualsiasi dribbling nell'area di rigore
- Car3rd: Dribbling che entrano nell'ultimo terzo del campo vicino alla porta
- CPA: Dribbling nell'area dei 18 yard
- CarMis: Numero di volte che un giocatore ha fallito nel tentativo di controllare una palla
- CarDis: Numero di volte che un giocatore perde il controllo della palla dopo essere stato contrastato da un avversario

- Rec: Numero di volte che un giocatore ha ricevuto con successo un passaggio
- RecProg: Passaggi completati che muovono la palla verso la porta avversaria di almeno 10 yard dal punto più lontano nei sei passaggi precedenti, o qualsiasi passaggio completato nell'area di rigore
- CrdY: Cartellini gialli
- CrdR: Cartellini rossi
- 2CrdY: Secondo cartellino giallo
- Fls: Falli commessi
- Fld: Falli subiti
- Off: Fuorigioco
- Crs: Cross
- TklW: Contrasti in cui la squadra del contrastatore ha vinto il possesso della palla
- PKwon: Calci di rigore guadagnati
- PKcon: Calci di rigore concessi
- OG: Autogol
- Recov: Numero di palloni recuperati
- AerWon: Duelli aerei vinti
- AerLost: Duelli aerei persi
- AerWon%: Percentuale di duelli aerei vinti

Grazie alle feature del dataset sopra riportate possiamo catturare tutti i differenti aspetti legati al modo di giocare di un giocatore.

Utilizzeremo il *ragionamento logico* e il *web semantico* per integrare nuove informazioni all'interno del dataset. Le nuove informazioni ricavate verranno usate per i task di predizione.

Capitolo 1) RAGIONAMENTO LOGICO E PROLOG

Nonostante la grande quantità di dati memorizzati nel dataset originale, abbiamo bisogno per i task di predizione che verranno effettuati di nuove feature più informative. Per esempio, abbiamo bisogno di una nuova feature in grado di indicare complessivamente se un giocatore ha caratteristiche o meno da "playmaker", quindi ottime abilità di passaggio e di palleggio in mezzo al campo; fa riferimento sia a playmaker difensivi (quindi mediani) che a playmaker offensivi (quindi trequartisti). Altra feature informativa estremamente utile che ci serve ottenere è quella relativa al dribbling, il dataset fornisce molte informazioni sulle statistiche relative al dribbling ma non indica se il giocatore è un buon dribblatore o meno.

Per poter far ciò, abbiamo costruito in modo automatizzato a partire dal dataset una *Knowledge Base* in Prolog, contenente fatti e regole. I fatti sono verità immutabili, le regole, invece, sono usate per inferire nuove informazioni.

Per quanto riguarda il Prolog si è fatto uso della libreria *pyswip*.

Per la creazione della KB (Knowledge Base) ho definito una funzione che per ogni giocatore definisce un fatto contente solo le informazioni del giocatore rilevanti per l'inferenza di nuovi dati.

```
player_stats(1, 'Brenden Aaronson', 'USA', 'MFFW', 'Leeds United',1596 ,1.19, 30.2, 5.65, 23.2, 1.75, 23.2, 3.22, 0.9).
player_stats(2, 'Yunis Abdelhamid', 'MAR', 'DF', 'Reims',1980 ,0.32, 34.5, 0.23, 38.5, 0.27, 38.5, 4.5, 3.77).
player_stats(3, 'Himad Abdelli', 'FRA', 'MFFW', 'Angers',770 ,2.09, 43.4, 5.93, 40.0, 1.51, 40.0, 6.4, 4.42).
```

Le regole definite sono le seguenti:

```
(strong_dribbler(Player) :- player_stats(_, Player, _, _, _, Min, ToSuc, _, _, _, _, _, _, _), ToSuc > 1.10, Min > 1000).

(strong_playmaker(Player) :- player_stats(_, Player, _, _, _, Min, _, Rec, RecProg, PasTotCmp, PasAss, PasCmp, PasProg, PasLonCmp),  Min>1000, Rec>34, RecProg>3, PasTotCmp>33, PasAss>0.85, PasCmp>33, PasProg>3, PasLonCmp>3).
```

Prima di definire le regole è stato trovato il valor medio per le feature relative a dribbling e passaggi all'interno del dataset. Quindi, le regole definiscono se il giocatore è un forte dribblatore e/o playmaker, per stabilirlo esse controllano che le feature di interesse abbiano valori sopra la media. Inoltre, si prendono in considerazione solo i giocatori che hanno giocato almeno 1000 minuti, poiché giocatori che hanno giocato pochi spezzoni di partita potrebbero avere ottime statistiche prestazionali basate però su poche partite.

Usando le due regole ricaviamo i migliori dribblatori e i migliori playmaker.

Di seguito, possiamo vedere una parte dell'output restituito:

```
Forti dribblatori:

Brenden Aaronson
Kevin Agudelo
Naouirou Ahamada
Sergio Akieme
Thiago AlcÃ¡ntara
Jim Allevinah
Domingos AndrÃ© Ribeiro Almeida
Felipe Anderson
Joe Aribo
Jordan Ayew
```

```
Forti playmakers:

Luis Alberto
Trent Alexander-Arnold
Domingos AndrÃ© Ribeiro Almeida
AngeliÃ±o
Ridle Baku
IvÃ¡n Balliu
Jean-Ricner Bellegarde
Jude Bellingham
Cristiano Biraghi
Benjamin Bourigeaud
Julian Brandt
```

Dopodichè aggiorniamo il dataset aggiungendo le due feature booleane *dribbler* e *playmaker*.

Capitolo 2) WEB SEMANTICO

Il web semantico permette ai dati di essere condivisi e riutilizzati, utilizza metadati e ontologie per rendere i contenuti del web comprensibili sia per gli essere umani che per le macchine, migliorando così la capacità dei computer di interpretare, collegare e utilizzare i dati in modo intelligente.

Nel mio caso, il web semantico è stato usato per recuperare l'altezza di ogni calciatore presente nel dataset. Una volta recuperata l'altezza, questa viene aggiunta la dataset come valore associato alla nuova feature *height*.

Si è fatto uso delle librerie *SPARQLWrapper* per eseguire le query SPARQL e della libreria *csv* per la lettura e scrittura di file CSV.

Ho definito la seguente funzione per recuperare l'altezza di un calciatore dato il suo URI su DBpedia:

```
def retrieve_player_height(player_uri):
  height = 0
  sparql = SPARQLWrapper("http://dbpedia.org/sparql")
  sparql.setReturnFormat(JSON)

  query = f"""
    PREFIX dbo: <http://dbpedia.org/ontology/>

    SELECT *
    WHERE {{
     <{player_uri}> a dbo:SoccerPlayer ;
                dbo:height ?height .
    }}
  """

  sparql.setQuery(query)
```

```
results = sparql.query().convert()
if 'results' in results and 'bindings' in results['results']:
    for result in results['results']['bindings']:
        height = result['height']['value']
else:
    print("No data found for the specified player.")

return height
```

Una volta recuperata l'altezza per ogni calciatore, aggiorniamo il dataset aggiungendo l'altezza relatica di ogni giocatore nella nuova feature *height*.

Capitolo 3) BAYESIAN NETWORK

Le **Bayesian Network**, o reti bayesiane, sono modelli grafici probabilistici che rappresentano un insieme di variabili e le loro dipendenze condizionali tramite un grafo aciclico diretto. Ogni nodo nel grafo rappresenta una variabile, mentre gli archi rappresentano le dipendenze condizionali tra queste variabili. Le reti bayesiane sono utilizzate per modellare le relazioni causali tra variabili, effettuare inferenze probabilistiche e aggiornare le probabilità alla luce di nuove evidenze.

Come prima cosa, carichiamo il dataset aggiornato con i nuovi attributi ottenuti tramite *ragionamento logico* e *web semantico*. Successivamente per risolvere errori generati durante la creazione della rete bayesiana a causa di valori nulli presenti all'interno del dataset, andiamo ad eliminare tutte le righe con valori nulli. Poiché il tempo necessario alla creazione della rete bayesiana era estremamente elevato a causa delle grandi dimensioni del dataset andiamo ad eliminare casualmente il 70% delle righe in modo da migliorare le prestazioni del sistema e ridurne il tempo di esecuzione. Infine, andiamo a selezionare solamente le colonne utili alla creazione della rete bayesiana.

```
# 6. BAYESIAN NETWORK
# carico dataset aggiornato
newDataset = loadDataset("new_dataset.csv")
# rimuovo righe con valori nulli
newDataset = newDataset.dropna()
# Calcola il numero di righe da eliminare
rows_to_drop = int(len(newDataset) * 0.70)
# Seleziona casualmente le righe da mantenere
rows_to_keep = newDataset.sample(n=len(newDataset) - rows_to_drop, random_state=42)
# Crea un nuovo dataset con le sole righe selezionate
newDataset = newDataset.loc[rows_to_keep.index]

selected_columns = [
    'Pos', 'TouDefPen', 'TouDef3rd', 'TouMid3rd', 'TouAtt3rd',
    'Tkl', 'PasProg', 'PPA', 'ScaDrib', 'Recov', 'height',
    'dribbler', 'playmaker'
]

newDataset = newDataset[selected_columns]
```

Per la creazione della rete bayesiana selezioniamo le feature più rappresentative di un giocatore che meglio permettono di individuarne il ruolo:

- **Ruolo del giocatore (Pos):** Questo è l'attributo target che si desidera predire utilizzando la rete bayesiana. È la variabile che rappresenta il risultato desiderato dell'analisi e su cui si baseranno le predizioni.
- **Statistiche difensive (TouDefPen, TouDef3rd, TouMid3rd, TouAtt3rd, Tkl, Recov):** Queste variabili potrebbero essere importanti per comprendere il comportamento difensivo del giocatore durante una partita. Ad esempio, il numero di duelli difensivi vinti o il numero di recuperi effettuati potrebbero essere indicatori di abilità difensive.
- **Statistiche offensive e di passaggio (PasProg, PPA, ScaDrib):** Queste variabili potrebbero fornire informazioni sull'abilità del giocatore nell'avanzare con la palla e nel creare opportunità offensive per la squadra. Ad esempio, il numero di passaggi progressivi o il numero di dribbling riusciti possono essere indicatori di abilità nell'avanzamento del gioco.
- **Caratteristiche fisiche (height):** Questo attributo potrebbe essere rilevante per comprendere le caratteristiche fisiche del giocatore, che potrebbero influenzare il suo ruolo o il suo stile di gioco.
- **Abilità specifiche (dribbler, playmaker):** Queste variabili potrebbero rappresentare abilità o caratteristiche specifiche del giocatore, come la capacità di dribblare o di creare gioco per la squadra. Queste abilità potrebbero essere importanti per determinare il ruolo del giocatore in campo.

Inzialmente, ho provato a creare la rete bayesiana con l'agoritmo *HillClimbSearch*, quest'ultimo però richiede la discretizzazione dei dati del dataset. Infatti, l'errore generato inizialmente durante la creazione del dataset era il seguente:

```
numpy.core._exceptions._ArrayMemoryError: Unable to allocate 347. GiB for an array with shape (186549012480,) and data type
int16
```

Per risolvere il seguente errore legato all'allocazione della memoria ho discretizzato il dataset utilizzando la libreria *KBinsDiscretizer* di *scikit-learn*.

Questa scelta ha semplificato la struttura del modello e migliorato le prestazioni computazionali. La discretizzazione ha reso il modello più interpretabile e ha ridotto il tempo necessario per l'addestramento del modello e il calcolo delle probabilità a posteriori.

```
newDataset = newDataset[selected_columns]
#discretizzo il dataset
discretizer = KBinsDiscretizer(encode='ordinal', strategy='uniform', subsample=None)
# Seleziona le colonne dei tipi float64 and int64
continuos_columns = newDataset.select_dtypes(include=['float64', 'int64']).columns
# Applica il "discretizer" alle colonne selezionate
newDataset[continuos_columns] = discretizer.fit_transform(newDataset[continuos_columns])
print(newDataset)
#Creazione o lettura della rete bayesiana in base alle necessità
bayesianNetwork = create_BN(newDataset)
#bayesianNetwork = loadBayesianNetwork()
```

Come mostrato nella seguente immagine, il dataset utilizzato per la creazione della rete bayesiana è stato efficacemente discretizzato:

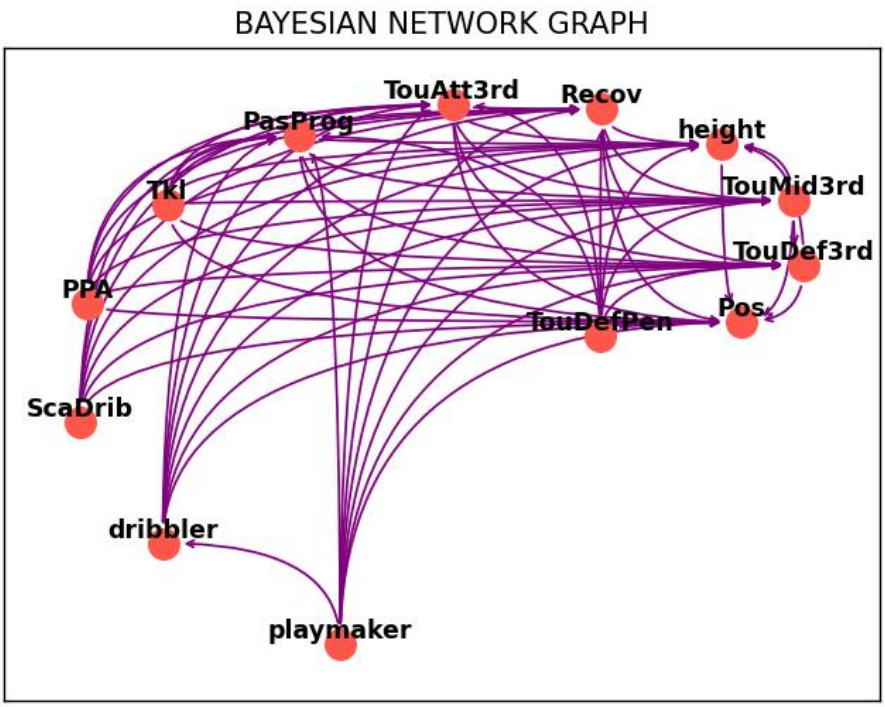
	Pos	TouDefPen	TouDef3rd	TouMid3rd	TouAtt3rd	Tkl	PasProg	PPA	ScaDrib	Recov	height	dribbler	playmaker
2187	MF	0.0	0.0	1.0	1.0	0.0	1.0	0.0	0.0	0.0	3.0	True	False
2366	FW	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	3.0	False	False
857	FWMF	0.0	0.0	1.0	1.0	0.0	1.0	0.0	0.0	0.0	2.0	False	False
1560	DF	1.0	2.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	False	False
641	DF	0.0	2.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	4.0	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...
2474	DF	0.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	3.0	False	False
1734	DF	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	3.0	False	False
642	MF	0.0	0.0	2.0	1.0	0.0	2.0	0.0	0.0	1.0	3.0	False	True
1360	DF	1.0	3.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	3.0	False	False
1448	MF	0.0	1.0	3.0	0.0	0.0	1.0	0.0	0.0	1.0	1.0	False	False

[591 rows x 13 columns]

Creazione Rete Bayesiana

```python
def create_BN(dataSet):
    #Ricerca della struttura ottimale
    hc_k2=HillClimbSearch(dataSet)
    k2_model=hc_k2.estimate(scoring_method='k2score')
    #Creazione della rete bayesiana
    model = BayesianNetwork(k2_model.edges())
    model.fit(dataSet,estimator=MaximumLikelihoodEstimator,n_jobs=-1)
    #Salvo la rete bayesiana su file
    with open('models/modello.pkl', 'wb') as output:
        pickle.dump(model, output)
    visualizeBayesianNetwork(model)
    return model
```

L'oggetto *HillClimbSearch* è responsabile di esplorare le possibili strutture della rete bayesiana modificando iterativamente la struttura per trovare quella che ha il miglior punteggio secondo un criterio specificato. In questo caso, ho utilizzato il metodo di punteggio K2, un metodo basato sulla verosimiglianza che valuta quanto bene la struttura della rete si adatta ai dati osservati.
La funzione *estimate* restituisce il miglior modello trovato.



La rete bayesiana generata ci permette di individuare ed evidenziare relazioni interessanti tra le variabili, che possono essere analizzate per migliorare le strategie di gioco. Analizzando la rete bayesiana possiamo notare che:

- La variabile "Pos" (Posizione del giocatore) è centrale e ha collegamenti con molte altre variabili come "TouDef3rd" (Tocchi nel terzo difensivo), "Recov" (Palloni recuperati), e "height" (Altezza). Questo suggerisce che la posizione di un giocatore ha un impatto significativo su questi aspetti del gioco. Ad esempio, un difensore centrale probabilmente avrà più tocchi nel proprio terzo difensivo e più palloni recuperati rispetto a un attaccante.
- Le variabili "ScaDrib" (Dribbling che porta a una azione da tiro) e "dribbler" sono strettamente collegate, indicando che i giocatori che dribblano frequentemente sono spesso coinvolti nella creazione di opportunità di tiro. Questo collegamento può essere utile per identificare giocatori chiave nel trasporto palla e nella rottura delle linee avversarie.
- Il nodo "playmaker" è collegato a variabili come "PasProg" (Passaggi che avanzano il gioco), mostrando che i playmaker sono cruciali nel dirigere il gioco verso l'area avversaria. Questo sottolinea l'importanza di giocatori capaci di effettuare passaggi chiave per smistare il gioco.
- L'altezza del giocatore ("height") è direttamente collegata ai recuperi ("Recov"). Questo può indicare che i giocatori più alti, come i difensori centrali, sono spesso coinvolti nel recuperare il pallone, probabilmente grazie alla loro presenza fisica e capacità nei duelli aerei.

Il metodo *fit* serve per stimare i parametri del modello, una volta che il metodo ha completato la sua esecuzione, il modello di rete bayesiana è completamente parametrizzato e pronto per essere utilizzato per inferenze, e quindi fare previsioni su dati non visti.

Ho utilizzato il modello di rete bayesiana per predire il ruolo di un calciatore, di cui conosciamo le statistiche prestazionali. La funzione *predici* ci permette di avere la distribuzione di probabilità di una variabile, nel nostro caso *Pos* (il ruolo del calciatore). Quindi mostrerà le probabilità di tutti i possibili stati della variabile *Pos*.

```
#Predico il valore di column per l'esempio
def predici(bayesianNetwork: BayesianNetwork, example, column):
    inference = VariableElimination(bayesianNetwork)
    result = inference.query(variables=[column], evidence=example)
    print(result)

#genera un esempio randomico
def generateRandomExample(bayesianNetwork: BayesianNetwork):
    return bayesianNetwork.simulate(n_samples=1).drop(columns=['Pos'])
```

Ho utilizzato la funzione *generateRandomExample* per generare un esempio casuale attraverso la funzione *simulate* che simula i valori delle variabili in base alle relazioni e alle dipendenze specificate nella rete. (*n_samples=1* perché generiamo un solo campione).

```
ESEMPIO RANDOMICO GENERATO
    TouAtt3rd  Tkl  TouDefPen  playmaker  TouDef3rd  TouMid3rd  height  PPA  ScaDrib  dribbler  PasProg  Recov
0         1.0  0.0        0.0      False        0.0        2.0     3.0  0.0      0.0      True      1.0    1.0
PREDIZIONE DEL SAMPLE RANDOM
+-----------+------------+
| Pos       |   phi(Pos) |
+===========+============+
| Pos(DF)   |     0.0000 |
+-----------+------------+
| Pos(DFFW) |     0.0000 |
+-----------+------------+
| Pos(DFMF) |     0.0000 |
+-----------+------------+
| Pos(FW)   |     0.0000 |
+-----------+------------+
| Pos(FWDF) |     0.0000 |
+-----------+------------+
| Pos(FWMF) |     0.0000 |
+-----------+------------+
| Pos(GK)   |     0.0000 |
+-----------+------------+
| Pos(MF)   |     0.0000 |
+-----------+------------+
| Pos(MFDF) |     1.0000 |
+-----------+------------+
```

Utilizzando le funzioni precedentemente descritte, ho generato un esempio casuale; quindi, un giocatore con statistiche prestazionali generate in maniera casuale. Per questo giocatore, la funzione di predizione ci restituisce una tabella con la distribuzione di probabilità della variabile *Pos* (posizione del giocatore) data l'evidenza fornita dal campione. Ogni riga rappresenta una possibile categoria per la variabile Pos (ad esempio "DF" sta per difensore, "MF" per centrocampista ecc.) e il valore associato mostra la probabilità di quella categoria

data l’evidenza.
In questo caso, il giocatore generato ha ruolo “MFDF” con probabilità 1.0.

Applicazioni nel mondo reale: Quest’applicazione della rete bayesiana può essere particolarmente utile ad un allenatore per capire quale può essere il ruolo adatto di un calciatore in base alle sue caratteristiche fisiche (height) e statistiche prestazionali in partita. Inoltre, un allenatore potrebbe beneficiare di questa distribuzione di probabilità anche per individuare la duttilità di un giocatore scoprendo nuovi ruoli in cui il giocatore potrebbe adattarsi in modo coerente alle sue caratteristiche.

Capitolo 4) APPRENDIMENTO SUPERVISIONATO

L'apprendimento supervisionato è stato usato all'interno del progetto con scopo di predizione. Nello specifico, ciò che vogliamo fare è predire il numero di gol di un calciatore a fine stagione sulla base delle caratteristiche del giocatore (ruolo, se è un dribblatore e/o un playmaker) e sulla base delle sue statistiche prestazionali P90 minuti. Il seguente task di predizione potrebbe essere molto utile per una società di calcio, nello specifico ad un allenatore o ad un dirigente sportivo capire con quanti gol il giocatore finirà la stagione ipotizzando approssimativamente il numero di minuti che totalizzerà al termine della stagione.

Le librerie utilizzate sono le seguenti: *scikit-learn, pandas*. Analizzeremo in seguito, come e perché sono state usate nello specifico.

Per questo progetto ho deciso di utilizzare tre modelli di apprendimento automatico:

LinearRegression

Il modello LinearRegression si basa sulla funzione lineare:

$Y(e) = w_0 + w_1 * X_1(e) + \ldots + w_n X_n(e)$

Dove:

- $X_1, \ldots, X_n$ sono le feature di input
- Y è la target feature
- e rappresenta l'esempio
- $w_0, w_1, \ldots, w_n$ sono i pesi
- w_0 è l'intercetta.
- w_i è il coefficiente della variabile indipendente X, ovvero il peso associato a X.

I pesi del modello vengono stimati in modo tale da minimizzare l'errore di predizione.

Il modello cerca di trovare la "linea di miglior adattamento" che minimizza la somma degli errori al quadrato tra i valori predetti e quelli osservati.

RandomForestRegressor

Il RandomForestRegressor è utilizzato per problemi di regressione. Questo modello di apprendimento automatico è basato sul metodo degli alberi decisionali e sul concetto di enseble learning. Questo modello combina molteplici alberi decisionali indipendenti per migliorare la precisione della previsione e ridurre il rischio di overfitting. Ogni albero decisionale è costruito su un sottoinsieme casuale del dataset originale. Durante la costruzione di ciascun albero viene considerato solo un sottoinsieme casuale di feature in modo da creare diversità tra gli alberi. Questo introduce ulteriore variabilità tra gli alberi, migliorando la robustezza del modello.

La predizione finale è ottenuta calcolando la media delle predizioni fatte da tutti gli alberi nella foresta.

DecisionTreeRegressor

Il DecisionTreeRegressor è un modello di apprendimento automatico utilizzato per risolvere problemi di regressione. Si basa sulla struttura di un albero decisionale, dove il nodo radice è il punto di partenza dell'albero, a partire da questo nodo il dataset viene suddiviso in base ai valori delle feature. Ogni nodo interno rappresenta una condizione su una feature del dataset. Questa condizione è utilizzata per suddividere il dataset in due o più sottogruppi, la scelta della feature e della condizione viene fatta in modo da minimizzare l'errore di predizione. Le foglie, invece, sono i nodi terminali dell'albero e rappresentano la predizione della target feature.
Per effettuare una predizione con un albero decisionale, un nuovo esempio viene propagato dall'alto verso il basso nell'albero, seguendo le condizioni dei nodi interni fino a raggiungere una foglia. Il valore della foglia è la predizione per il nuovo esempio.

Scelta ottimale degli Iperparametri

Gli iperparametri sono parametri esterni che vengono impostati prima dell'addestramento del modello e che non vengono appresi dai dati. Gli iperparametri influenzano il comportamento del modello e la sua capacità di apprendere dai dati. La scelta dei migliori iperparametri è cruciale per ottimizzare le prestazioni del modello.

Iperparametri dei Modelli Utilizzati

1. **Linear Regression**
    - **fit_intercept**:
        - **Descrizione**: Decide se calcolare l'intercetta per il modello (l'intercetta è il valore di y quando tutte le variabili indipendenti X sono uguali a 0)
        - **Valori Esplorati**: [True, False]
    - **Copy_X**:
        - **Descrizione**: Se impostato a True viene creata una copia dei dati di input, questo significa che se i valori delle feature di input cambiano durante l'addestramento del modello, il dataset originale rimarrà intatto.
        - **Valori Esplorati**: [True, False]
    - **Positive**:
        - **Descrizione**: Impone che i coefficienti del modello di regressione siano positivi.
        - **Valori Esplorati**: [True, False]
2. **RandomForestRegressor**
    - **n_estimators**:
        - **Descrizione**: Numero di alberi nella foresta.
        - **Valori Esplorati**: [50, 100, 200]
    - **max_depth**:
        - **Descrizione**: Profondità massima dell'albero. Se None, gli alberi vengono espansi fino a quando tutte le foglie contengono meno di min_samples_split campioni.

        - **Valori Esplorati**: [None, 10, 20, 30]

    - **min_samples_split**:
        - **Descrizione**: Numero minimo di campioni richiesti per dividere un nodo interno.
        - **Valori Esplorati**: [2, 5, 10]
    - **min_samples_leaf**:
        - **Descrizione**: Numero minimo di campioni richiesti per essere in un nodo foglia.
        - **Valori Esplorati**: [1, 2, 4]
3. **DecisionTreeRegressor**
    - **max_depth**:
        - **Descrizione**: Profondità massima dell'albero.
        - **Valori Esplorati**: [None, 10, 20, 30]
    - **min_samples_split**:
        - **Descrizione**: Numero minimo di campioni richiesti per dividere un nodo interno.
        - **Valori Esplorati**: [2, 5, 10]
    - **min_samples_leaf**:
        - **Descrizione**: Numero minimo di campioni richiesti per essere in un nodo foglia.
        - **Valori Esplorati**: [1, 2, 4]
    - **splitter**:
        - **Descrizione**: Strategia utilizzata per scegliere la divisione in ogni nodo.
        - **Valori Esplorati**: ['best', 'random']

Per la scelta ottimale degli iperparametri si è utilizzata la Grid Search con la Cross Validation, dove si sono stabiliti dei valori possibili per ogni iperparametro del modello, formando una "griglia" di combinazioni. La Cross Validation divide il dataset di addestramento in K sottoinsiemi (folds), per ogni combinazione di iperparametri, il modello viene addestrato su k-1 folds e testato sul fold rimanente. Questo processo si ripete k volte cambiando ogni volta il fold di test, si calcola la media delle prestazioni su tutte le iterazioni per ottenere una stima delle prestazioni del modello.

Si è utilizzata la funzione *GridSearchCV* di *scikit-learn* per la ricerca degli iperparametri ottimali per ogni modello. Per ogni combinazione di iperparametri nella griglia, *GridSearchCV* esegue la Cross Validation e valuta le prestazioni del modello usando usando come metrica di valutazione l'errore quadratico medio negativo.

Una volta valutate le prestazioni del modello per ogni combinazione di iperparametri, *GridSearchCV* identifica la combinazione di iperparametri per cui il modello ha avuto le migliori prestazioni nella Cross Validation. Questa combinazione di iperparametri è considerata ottimale per il modello, dato il dataset e la metrica di valutazione.

```
  File "C:\Users\Giannantonio\OneDrive\Desktop\PROGETTO_ICON\ICON-1\env\Lib\site-packages\pandas\core\internals\blocks.py", line 758, in astype
    new_values = astype_array_safe(values, dtype, copy=copy, errors=errors)
                 ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\Giannantonio\OneDrive\Desktop\PROGETTO_ICON\ICON-1\env\Lib\site-packages\pandas\core\dtypes\astype.py", line 237, in astype_array_safe
    new_values = astype_array(values, dtype, copy=copy)
                 ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\Giannantonio\OneDrive\Desktop\PROGETTO_ICON\ICON-1\env\Lib\site-packages\pandas\core\dtypes\astype.py", line 182, in astype_array
    values = _astype_nansafe(values, dtype, copy=copy)
             ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\Giannantonio\OneDrive\Desktop\PROGETTO_ICON\ICON-1\env\Lib\site-packages\pandas\core\dtypes\astype.py", line 133, in _astype_nansafe
    return arr.astype(dtype, copy=True)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^
ValueError: could not convert string to float: 'Roberto Navarro'
```

Inizialmente, si è verificato un problema perché il dataset conteneva valori categorici che non potevano essere convertiti automaticamente in numeri dai modelli di machine learning di scikit-learn. Questo causava un errore durante l'addestramento dei modelli, poiché essi richiedono dati numerici.

Per risolvere questo problema, abbiamo implementato un processo di preprocessing dei dati utilizzando le librerie *pandas* e *scikit-learn*.

- Con *pandas*, abbiamo identificato quali colonne del dataset erano numeriche e quali erano categoriche. Questo è un passo cruciale per poter applicare trasformazioni appropriate su ciascun tipo di colonna, garantendo che ogni dato sia trattato nel modo corretto.
- Utilizzando scikit-learn, abbiamo creato due pipeline di preprocessing: una per le colonne numeriche e una per quelle categoriche.
- Per le colonne numeriche, abbiamo utilizzato SimpleImputer per riempire i valori mancanti con la mediana e StandardScaler per normalizzare i dati. Questo assicura che i dati numerici siano privi di valori mancanti e abbiano una scala uniforme.
- Per le colonne categoriche, abbiamo utilizzato SimpleImputer per riempire i valori mancanti con un valore di default ('missing') e OneHotEncoder per convertire le categorie in una rappresentazione numerica. Questo consente di trasformare le stringhe categoriche in un formato numerico che può essere utilizzato dai modelli di machine learning.
- Abbiamo creato una pipeline per ciascun modello (Regressione Lineare, Random Forest Regressor, Decision Tree Regressor) utilizzando *scikit-learn*. Ogni pipeline include il preprocessing dei dati e il modello di regressione vero e proprio. Questo garantisce che ogni modello riceva dati correttamente pre-elaborati.

Il processo di preprocessing dei dati converte ora le colonne categoriche in rappresentazioni numeriche, permettendo ai modelli di machine learning di essere addestrati correttamente sui dati.

Ogni modello è stato addestrato con un valore di k=5 (5 fold, 5 iterazioni).

I valori ottimali restituiti per gli iperparametri sono:

Modello	Parametro	Valore
LinearRegression	regressor__copy_X	True
LinearRegression	regressor__fit_intercept	False
RandomForestRegressor	regressor__max_depth	None
RandomForestRegressor	regressor__min_samples_leaf	2
RandomForestRegressor	regressor__min_samples_split	5
RandomForestRegressor	regressor__n_estimators	50
DecisionTreeRegressor	regressor__max_depth	None
DecisionTreeRegressor	regressor__min_samples_leaf	4
DecisionTreeRegressor	regressor__min_samples_split	10
DecisionTreeRegressor	regressor__splitter	best

Il valore “None” indica che non è stata impostata una profondità massima per gli alberi.

Fase di addestramento e test

Nella fase di addestramento e test i modelli sono stati addestrati usando la K-fold Cross Validation con K=5 vista la dimensione moderata del dataset. I modelli sono stati addestrati su una porzione del dataset coerentemente ai limiti computazionali della macchina su cui è stato eseguito il programma.

```
cv = RepeatedKFold(n_splits=5, n_repeats=5, random_state=42)

scoring_metrics = {
    'mae': make_scorer(mean_absolute_error),
    'mse': make_scorer(mean_squared_error, greater_is_better=False),
}

results_dtc = {metric: cross_val_score(dtc, X, y, scoring=scorer, cv=cv) for metric, scorer in scoring_metrics.items()}
results_rfc = {metric: cross_val_score(rfc, X, y, scoring=scorer, cv=cv) for metric, scorer in scoring_metrics.items()}
results_reg = {metric: cross_val_score(reg, X, y, scoring=scorer, cv=cv) for metric, scorer in scoring_metrics.items()}
```

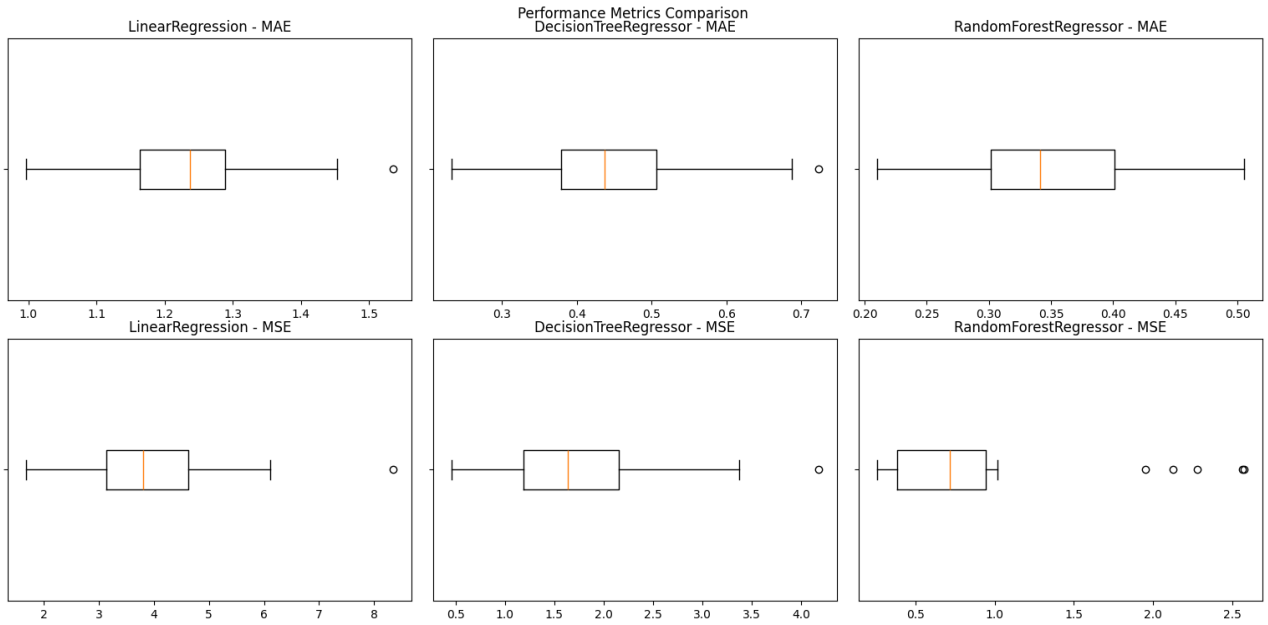
Vautazione delle prestazioni

Per la valutazione delle prestazioni dei modelli ho scelto le seguenti metriche:

- **MAE (Mean Absolute Error):** Rappresenta l’errore assoluto medio, nonché la media delle differenze assolute tra la predizione del modello e il valore reale della target feature. È facilmente interpretabile e penalizza in modo uniforme tutti gli errori, senza dare un peso eccessivo agli errori più grandi.

- **MSE (Mean Squared Error):** Rappresenta l'errore quadratico medio, nonché la media dei quadrati delle differenze tra le predizioni del modello e i valori reali. Questa metrica penalizza più severamente gli errori maggiori essendo il quadrato delle differenze.

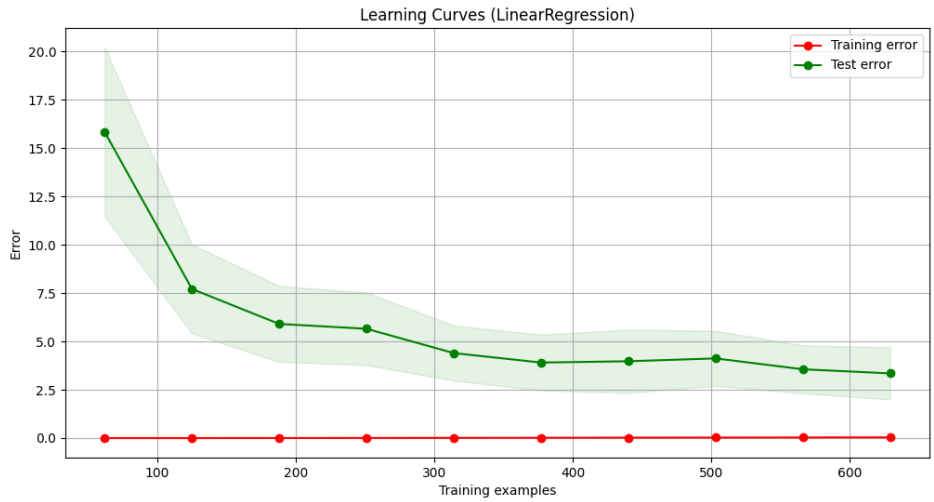
RISULTATI:



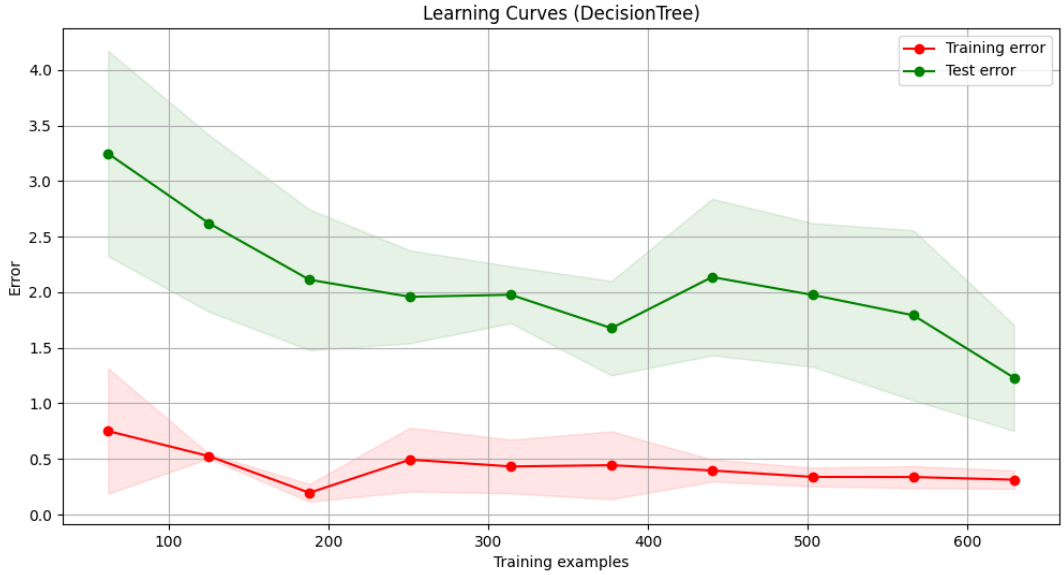
Analizzando i grafici (box-plot) possiamo notare come il modello Random Forest Regressor abbia una minore presenza di outlier rispetto agli altri due modelli, suggerendo che il modello ha una buona capacità di generalizzazione e non mostra segni di overfitting.

CURVE DI APPRENDIMENTO

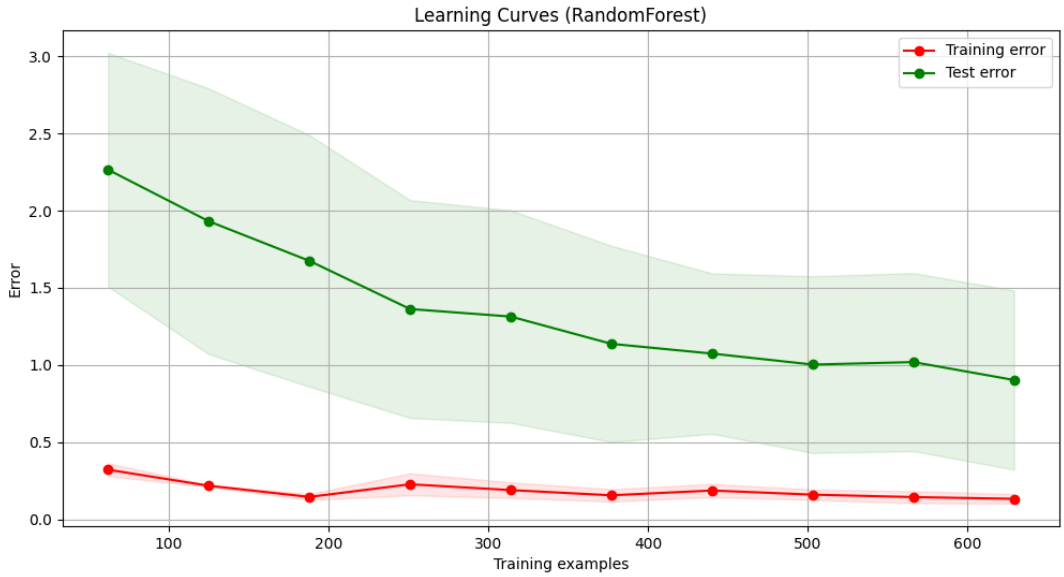
Le curve di apprendimento sono uno strumento estremamente utile per valutare il comportamento di un modello all'aumentare degli esempi di training.



Per quanto riguarda il modello Linear Regression la curva dell'errore di training è piatta e ferma allo 0, ciò significa che il modello si adatta perfetamente ai dati di training, ma non generalizza bene sui dati di test, come dimostrato dall'errore di test significativamente più alto. Sembra un caso tipico di overfitting.



Per quanto riguarda il modello Decision Tree, l'errore di training è molto basso, indicando che il modello si adatta quasi pefettamente ai dati di training. Invece, l'errore di test è significativamente più alto dell'errore di training suggerendo che il modello non sia in grado di generalizzare bene su dati non visti.



Nel modello Random Forest l'errore di training è basso e diminuisce leggermento con l'aumentare degli esempi di training, rimanendo complessivamente basso. Mentre, l'errore di test diminuisce significativamente all'aumentare degli esempi di training e sembra convergere

verso l'errore di training. Questo modello sembra non avere overfitting poiché il divario tra l'errore di test e l'errore di training è basso ed entrambi gli errori diminuiscono con l'aumentare degli esempi di addestramento. Ciò suggerisce che il modello Random Forest generalizzi bene su dati non visti.

Varianza e deviazione standard

Nell'ambito del machine learning deviazione standard e varianza fanno riferimento alla dispersione dell'errore.

Deviazione standard: La deviazione standard degli errori è la radice quadrata della varianza e fornisce una misura della dispersione degli errori dalla media. Una bassa deviazione standard indica che gli errori di predizione sono più concentrati attorno alla media degli errori.

Varianza: La varianza del modello misura quanto le predizioni del modello variano quando viene addestrato su diversi dati di training. Un'alta varianza indica che il modello è molto sensibile ai cambiamenti nei dati di training, e dunque suggerisce la presenza di *overfitting*.

Un modello che mostra una bassa deviazione standard e varianza su dati sui dati di addestramento ma un alta deviazione standard e varianza sui dati di test è probabile che stia overfittando.

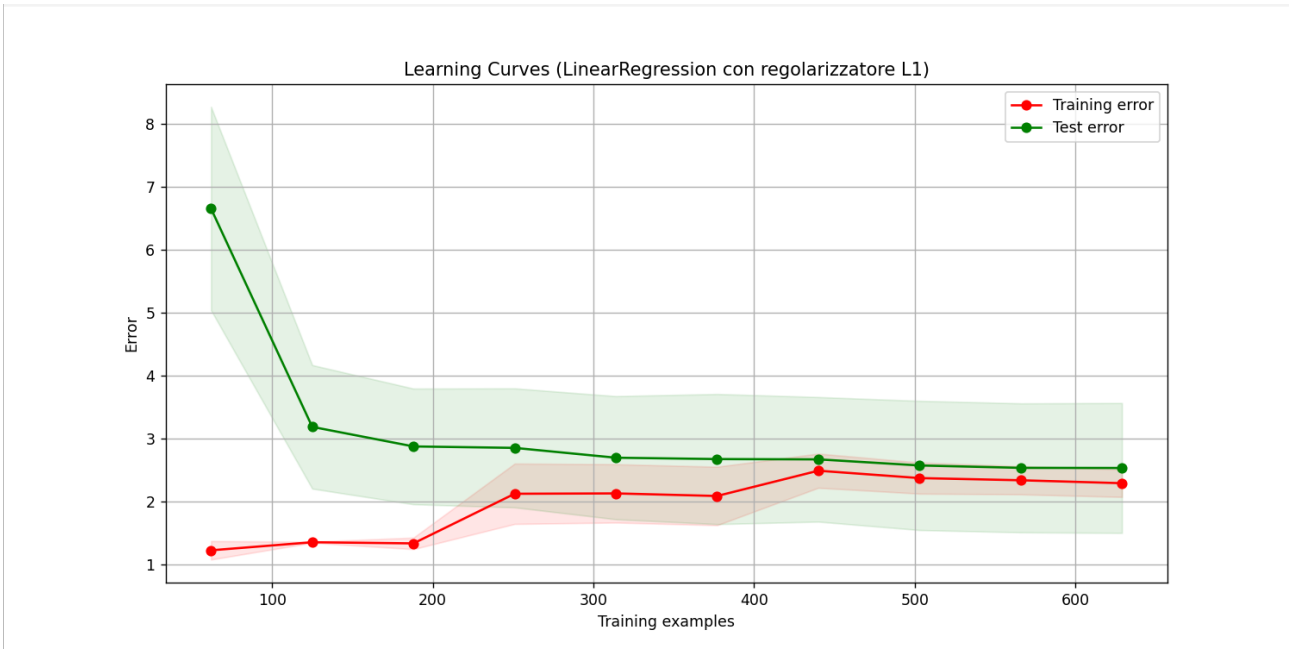
	Modello	Tipo Errore	Deviazione Standard	Varianza
1	DecisionTree	Train	0.08268779634603415	0.006837271664563209
2	DecisionTree	Test	0.47793560544189956	0.22842244294911515
3	RandomForest	Train	0.030953716246960267	0.000958132549493324
4	RandomForest	Test	0.5805353210859832	0.3370212590284055
5	LinearRegression	Train	0.005152583426239744 5	2.6549115964360672e-05
6	LinearRegression	Test	1.3515574869761182	1.8267076406012002

L'analisi della deviazione standard e della varianza conferma le impressioni iniziali relative all'overfitting. È quindi necessario mettere in atto delle strategie per la rimozione dell'overfitting.

Soluzione all'overfitting

Una possibile soluzione all'overfitting per il modello di regressione lineare (LinearRegression) è l'implementazione della **regolarizzazione L1**, nota anche come LASSO (Least Absolute Shrinkage and Selection Operator). Il regolarizzatore, in generale, penalizza la complessità del modello.

La regolarizzazione L1 aggiunge un termine di penalizzazione alla funzione di perdita (funzione di errore) del modello di regressione lineare, che è proporzionale alla somma dei valori assoluti dei coefficienti del modello.
La regolarizzazione L1 tende a ridurre a 0 molti dei coefficienti meno importanti del modello, questo processo è noto come selezione delle feature, dove solo le feature più rilevanti vengono mantenute nel modello finale. Di conseguenza, il modello diventa più semplice e meno suscettibile all'overfitting.
Il motivo per il quale si è deciso di usare la regolarizzazione L1 è dovuto al grande numero di feature presenti nel dataset usato per l'addestramento dei modelli.



Modello	Tipo Errore	Deviazione Standard	Varianza
LinearRegression con L1	Train	0.219457	0.048161
LinearRegression con L1	Test	1.033127	1.067351

Come possiamo vedere, grazie all'utilizzo del regolarizzatore L1 il modello di regressione lineare ha ridotto la deviazione standard e la varianza sull'errore di test rispetto al modello di regressione lineare senza regolarizzatore L1. Inoltre, come possiamo notare dalle curve di apprendimento il modello di regressione lineare con regolarizzatore L1 ha un errore di training più alto rispetto al modello di regressione lineare senza regolarizzatore, ma un errore di test decisamente più basso e stabile con una differenza minore tra i due errori. Questo dimostra

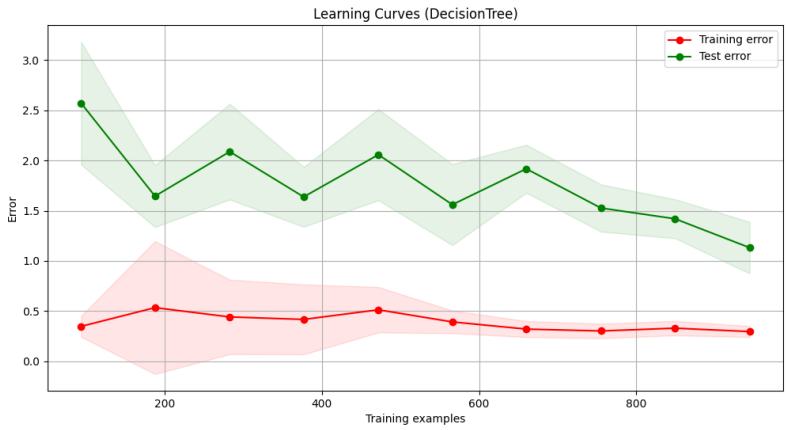
una migliore capacità di generalizzazione su nuovi dati da parte del modello di regressione lineare con regolarizzatore L1. Abbiamo quindi ridotto l'overfitting inziale.

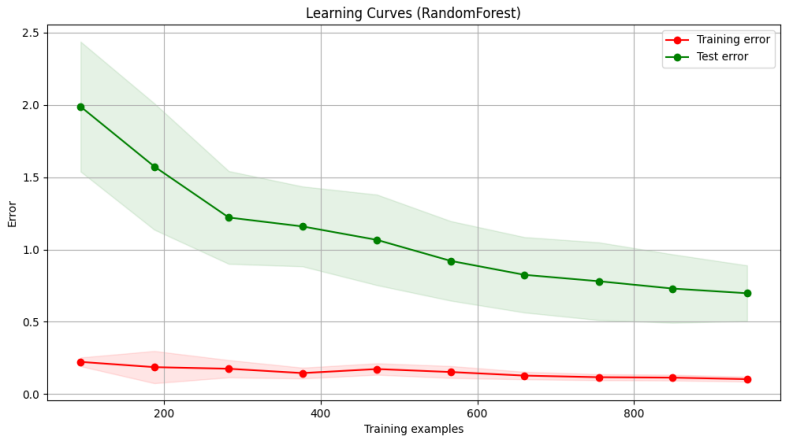
Per ridurre, invece, l'eventuale overfitting nei modelli DecisionTreeRegressor e RandomForestRegressor abbiamo aumentato la dimensione dei dati di training. Infatti, come possiamo vedere dalle curve di apprendimento mente prima si superavano i 600 esempi di training, adesso i modelli sono addestrati su più di 800 esempi di training.

Ciò ha portato ad una riduzione significativa della deviazione standard e della varianza, risolvendo quindi l'overfitting.

	Model	Train Error Std	Test Error Std	Train Error Var	Test Error Var
1	DecisionTree	0.055119912982322966	0.25727490181266777	0.0030382048071788525	0.06619037510271782
2	RandomForest	0.016609669618923743	0.19268149911736693	0.00027588112484979897	0.03712616010211588

La curva di apprendimento del DecisionTree rimane stabile anche dopo l'aggiunta dei nuovi esempi di training, invece, nella curva di apprendimento del RandomForest possiamo notare una riduzione ulteriore dell'errore di test .





Capitolo 5) APPRENDIMENTO NON SUPERVISIONATO

Il clustering è una tecnica di apprendimento non supervisionato utilizzata per raggruppare i dati in insiemi (cluster) basati su somiglianze tra i dati stessi. Ogni cluster contiene dati simili tra loro e dissimili dai dati in altri cluster. Il clustering è utile per scoprire pattern nascosti nei dati e ridurre la dimensionalità dei dati.

Il K-Means è uno degli algoritmi di clustering più popolari. Funziona dividendo un dataset in k cluster. L'obiettivo è minimizzare la somma delle distanze al quadrato tra i punti e il centroide del loro cluster assegnato. Il K-Means funziona in questi passaggi:

1. Selezione di k centroidi iniziali.
2. Assegnazione di ogni punto al centroide più vicino.
3. Calcolo dei nuovi centroidi come la media dei punti assegnati a ciascun cluster.
4. Ripetizione dei passaggi 2 e 3 fino a quando i centroidi non cambiano più posizione significativamente.

Nel nostro caso, il clustering con K-means può essere utilizzato per raggruppare i calciatori in gruppi sulla base delle loro caratteristiche di gioco. Questo può aiutare ad identificare giocatori con stili di gioco simile, facilitando l'analisi delle prestazioni e il processo di scouting.

Librerie utilizzate:

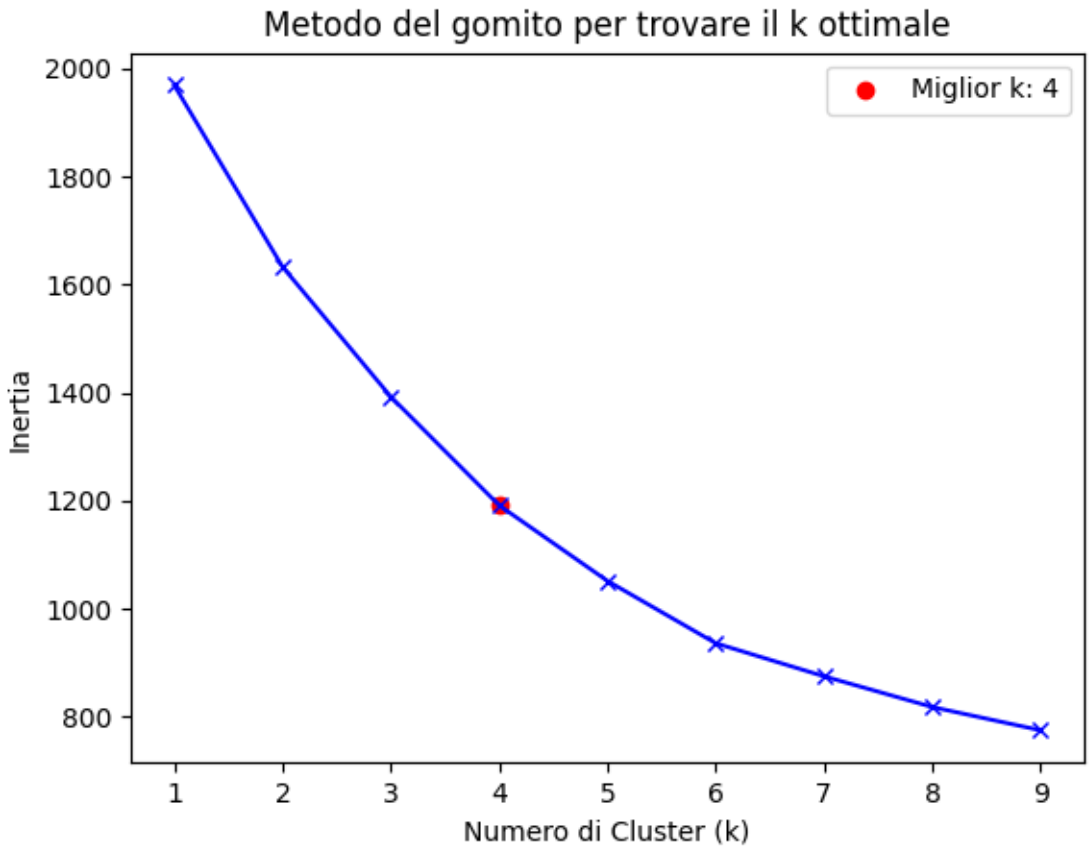
- **Pandas**: per la manipolazione e l'analisi dei dati.
- **Numpy:** per operazioni numeriche.
- **scikit-learn:** per il clustering.
- **Matplotlib:** per la visualizzazione dei grafici.
- **Kneed:** per l'individuazione del punto di gomito.

Regola del gomito:

```python
#Funzione che calcola il numero di cluster ottimale per il dataset mediante il metodo del gomito
def regolaGomito(numerical_features_scaled):
    intertia = []
    #fisso un range di k da 1 a 10
    maxK=10
    for i in range(1, maxK):
        #eseguo il kmeans per ogni k, con 5 inizializzazioni diverse e con inizializzazione random. Prendo la migliore
        kmeans = KMeans(n_clusters=i,n_init=5,init='random')
        kmeans.fit(numerical_features_scaled)
        intertia.append(kmeans.inertia_)
    #mediante la libreria kneed trovo il k ottimale
    kl = KneeLocator(range(1, maxK), intertia, curve="convex", direction="decreasing")
    # Visualizza il grafico con la nota per il miglior k
    plt.plot(range(1, maxK), intertia, 'bx-')
    plt.scatter(kl.elbow, intertia[kl.elbow - 1], c='red', label=f'Miglior k: {kl.elbow}')
    plt.xlabel('Numero di Cluster (k)')
    plt.ylabel('Inertia')
    plt.title('Metodo del gomito per trovare il k ottimale')
    plt.legend()
    plt.show()
    return kl.elbow
```

Per l'individuazione del numero ottimale di cluster (k) in un algoritmo di clustering K-means si è utilizzata la regola del gomito. L'idea è quella di eseguire l'agoritmo k-means per un

intervallo di valori di k e tracciare il grafico della somma delle distanze al quadrato (inerzia) tra i punti e i centroidi del loro cluster assegnato. Si calcola l'inerzia per ogni valore di k, il punto in cui il grafico mostra una diminuzione significativa dell'inerzia è noto come il "gomito". Questo punto suggerisce il numero ottimale di cluster.

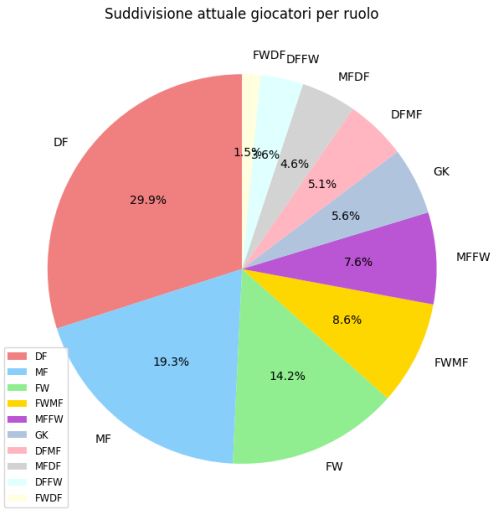


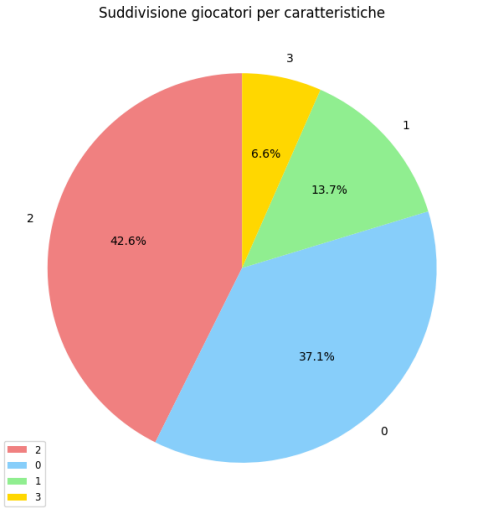
Utilizzando il metodo del gomito, nel mio caso, il numero ottimale di cluster è k=4. Di conseguenza, l'agoritmo di clustering K-means raggrupperà i calciatori in 4 cluster.

```python
#Funzione che esegue il kmeans sul dataset e restituisce le etichette e i centroidi
def calcolaCluster(dataSet):
    numerical_features = dataSet.select_dtypes(include=[np.number])
    #Standardizzazione delle feature numeriche
    scaler = StandardScaler()
    numerical_features_scaled = scaler.fit_transform(numerical_features)
    k=regolaGomito(numerical_features_scaled)
    km = KMeans(n_clusters=k,n_init=10,init='random')
    km = km.fit(numerical_features_scaled)
    etichette = km.labels_
    centroidi = km.cluster_centers_
    return etichette, centroidi
```

Il parametro *n_init* specifica il numero di volte che deve essere eseguito l'algoritmo k-means, nel nostro caso 10. Il valore *random* assegnato al parametro *init* indica che i centroidi iniziali vengono inizializzati casualmente.

Adesso confrontiamo la distribuzione dei ruoli all'interno del dataset con il raggruppamento dei giocatori in cluster ottenuto in base alle loro caratteristiche e stili di gioco.





Grazie al clustering possiamo individuare tutti i giocatori con caratteristiche e stili di gioco simili senza basarci esclusivamente sulla grande varietà di ruoli specifici.

Sviluppi Futuri

Si potrebbero aggiungere al dataset le statistiche prestazionali dei giocatori anche delle stagioni precedenti. In questo modo si potrebbero aumentare i dati di addestramento dei modelli in modo da effettuare predizioni non solo sulla base delle statistiche prestazionali dell'ultima stagione ma anche sulla base delle stagioni precedenti. Inoltre, si potrebbe aggiungere al dataset il valore di mercato di ogni giocatore e addestrare i modelli anche su questa feature di input in modo tale da predire anche il giusto valore di mercato di un giocatore in base alle sue prestazioni.

Riferimenti bibliografici

Ragionamento logico: D. Poole, A. Mackworth: Artificial Intelligence: Foundations of Computational Agents. 3/e. Cambridge University Press [Ch.5]

Prolog: D. Poole, A. Mackworth: Artificial Intelligence: Foundations of Computational Agents. 3e, Cambridge University Press [Ch.15]

Apprendimento supervisionato: D. Poole, A. Mackworth: Artificial Intelligence: Foundations of Computational Agents. Cambridge University Press. 3rd Ed. [Ch.7]

Apprendimento non supervisionato: D. Poole, A. Mackworth: Artificial Intelligence: Foundations of Computational Agents. Cambridge University Press. 3rd Ed. [Ch.10]

Ragionamento probabilistico e reti bayesiane: D. Poole, A. Mackworth: Artificial Intelligence: Foundations of Computational Agents. Cambridge University Press. 3/e. [Ch.9]

Descrizione delle feature del dataset:
https://www.kaggle.com/datasets/vivovinco/20222023-football-player-stats