UNIVERSITÀ DEGLI STUDI DI BARI ALDO MORO
DIPARTIMENTO DI INFORMATICA
CORSO DI LAUREA IN INFORMATICA

RELAZIONE TECNICA IN
INGEGNERIA DELLA CONOSCENZA

SISTEMA ESPERTO PER LA PREDIZIONE DELLE PARTITE DI PALLACANESTRO

Gruppo composto da:
Lepore Nicola: 739765 n.lepore4@studenti.uniba.it
Federica de Virgilio: 735274 f.devirgilio3@studenti.uniba.it

Anno Accademico 2022-2023

Sommario

INTRODUZIONE

Idea di progetto

Nella realizzazione di questo caso di studio, ci si è posti come obiettivo quello di sviluppare un **sistema esperto** che, mediante l'analisi delle statistiche dei giocatori delle squadre prese in esame, fosse in grado, quando interrogato dall'utente, di effettuare queste operazioni:

1) Prelevare le informazioni aggiornate automaticamente dal sito LBA
2) Strutturare i dati prelevati
3) Predire quali sono i ruoli dei giocatori in base alle caratteristiche fisiche e statistiche
4) Dare una valutazione ai giocatori e metterli in una classifica in base al ruolo
5) Predizione risultati di una partita

È stato effettuato uno studio mediante l'utilizzo di diversi algoritmi di **Machine Learning** per trovare il modello e algoritmo con l'accuratezza più alta, ed è stato progettato un **sistema esperto** tramite una knowledge base.

LINGUAGGIO E AMBIENTI UTILIZZATI

Linguaggi e librerie scelte

Per la realizzazione del progetto è stato scelto come linguaggio **Python**, nello specifico per il Machine Learning e la creazione del dataset, in particolare è stato usato il Jupiter Notebook per organizzare le attività di sviluppo e test in un'unica interfaccia utente che semplifica il lavoro con le librerie, in quanto usa un'interfaccia più user friendly. Il codice è organizzato insieme a visualizzazioni e testo. In questo modo il progetto viene sviluppato e in tempo reale si può scrivere la documentazione tramite un unico software.

Per quanto riguarda il sistema esperto è stato scelto **Prolog,** tramite la libreria PySwip, che permette di consultare e dimostrare le regole scritte in linguaggio Prolog su Python; la versione della libreria adoperata in questo progetto è la 0.2.11. PySwip si interfaccia con l'interprete SWI-Prolog 8.4.3 necessario per l'esecuzione del sistema esperto.

Ambiente di sviluppo

Per la stesura del codice in Python e Jupiter è stato utilizzato come IDE **IntelliJ**.

Il codice riguardante il Machine Learning è stato scritto su un Notebook Python che permette di eseguire porzioni di codice singolarmente e di inserire spiegazioni, commenti, grafici, …, scritti in markdown.

ATTIVITÀ DI RICERCA E DATASET

Creazione dataset

È stato deciso di creare dei dataset per le nostre specifiche necessità, per fare ciò sono stati sviluppati script in Python atti ad ottenere le informazioni necessarie dal sito LBA; i dataset generati sono:

- **giocatori.csv**: contenente il nome del giocatore, il numero del giocatore, la squadra del giocatore, il ruolo, l'altezza, i minuti, i falli commessi e subiti, i tiri da due tentati e realizzati con la percentuale, i tiri da tre tentati e realizzati con la percentuale, i tiri da uno tentati e realizzati con la percentuale, rimbalzi offensivi e difensivi, stoppate date e subite, palle perse e recuperate, assist.

nome_giocatore	numero_giocatore	squadra	ruolo	altezza	minuti
abass_awudu_abass	55	virtus_segafredo_bologna	Ala	198	7
muhammad_ali_abdur_rahkman	5	carpegna_prosciutto_pesaro	Guardia	193	30
dimitrios_agravanis	0	gevi_napoli_basket	Ala	208	23
nicola_akele	45	germani_brescia	Ala	203	15
davide_alviti	40	ea7_emporio_armani_milano	Ala	200	8
karvel_markeese_anderson	23	tezenis_verona	Guardia	188	27
sacar_anim	2	unahotels_reggio_emilia	Guardia	196	27
daijuan_antonio	11	pallacanestro_trieste	Guardia	190	0
darion_atkins	44	dolomiti_energia_trentino	Centro	203	30
ismael_bako	7	virtus_segafredo_bologna	Centro	208	15
tommaso_baldasso	25	ea7_emporio_armani_milano	Play	192	8
moussa_bamba	83	gevi_napoli_basket	Ala	0	2
adrian_banks	1	nutribullet_treviso_basket	Guardia	190	29
billy_baron	12	ea7_emporio_armani_milano	Guardia	188	21
frank_bartley	24	pallacanestro_trieste	Guardia	191	32
pietro_basta	8	happy_casa_brindisi	Guardia	0	1
jordan_bayehe	26	happy_casa_brindisi	Ala	204	16
marco_belinelli	3	virtus_segafredo_bologna	Guardia	196	20
davide_belloni	8	unahotels_reggio_emilia	Ala	0	3
eimantas_bendzius	20	banco_di_sardegna_sassari	Ala	207	31

falli_c	falli_s	t2_r	t2_t	t2_per	t3_r
2	0	0	1	0	0
2	4	3	7	53	1
2	4	2	4	50	0
1	0	1	2	50	0
0	0	0	1	26	0
2	2	2	4	42	3
2	1	2	4	51	0
0	0	0	0	0	0
2	2	5	9	55	0
1	1	1	3	62	0
1	0	0	0	0	1
5	10	2	0	1	3
1	4	3	6	52	1
1	1	1	1	65	2
1	4	4	9	43	2
0	0	0	0	0	0
2	1	1	3	48	0
1	1	1	3	54	1
3	4	0	0	1	2
1	3	1	3	48	2

t3_t	t3_per	t1_r	t1_t	t1_per	rim_o
1	0	0	0	0	0
4	42	2	3	75	1
1	22	2	4	59	1
1	27	0	0	0	1
1	12	0	0	75	0
7	42	2	3	85	0
2	36	1	1	75	0
0	0	0	0	0	0
2	18	2	3	71	2
0	0	1	1	64	1
2	40	0	0	50	0
2	4	7	0	0	3
4	37	4	4	88	0
4	44	2	2	93	0
6	34	4	5	76	0
0	0	0	0	0	0
0	0	1	1	66	1
5	34	2	2	89	0
0	2	2	0	0	4
6	48	2	3	72	0

rim_d	rim_t	stop_d	stop_s	palle_p	palle_r
0	0	0	0	0	0
2	3	0	0	2	3
3	4	1	0	1	1
1	2	0	0	0	0
1	1	0	0	0	0
1	2	0	0	1	1
1	2	0	0	1	1
0	0	0	0	0	0
4	7	0	0	2	2
2	4	0	0	0	0
0	1	0	0	0	0
1	0	12	0	-1	0
2	2	0	0	2	1
2	2	0	0	1	2
4	4	0	0	2	3
0	0	0	0	0	0
2	3	0	0	0	0
0	1	0	0	0	1
0	0	8	0	-4	0
3	4	0	0	1	1

- **partita.json/partita21.json**: che contiene la telecronaca della partita e tutte le informazioni relative ad essa, strutturati nella seguente classe

```python
class Partita:
    Nicolalepore0
    def __init__(self, squadra_casa, squadra_ospite, risultato_finale_host, risultato_finale_guest):
        self.squadra_casa = squadra_casa
        self.squadra_ospite = squadra_ospite
        self.risultato_finale_host = risultato_finale_host
        self.risultato_finale_guest = risultato_finale_guest
        self.eventi = []
Nicolalepore0
class Evento:
    Nicolalepore0
    def __init__(self, punti_casa, punti_ospite, evento,quarto, time,giocatore):
        self.punti_casa = punti_casa
        self.punti_ospite = punti_ospite
        self.evento = evento
        self.quarto = quarto
        self.time = time
        self.giocatore = giocatore
```

```python
class Giocatore:
    Nicolalepore0
    def __init__(self, nome_giocatore, numero_giocatore, squadra):
        self.nome_giocatore = nome_giocatore
        self.numero_giocatore = numero_giocatore
        self.squadra = squadra
```

Elaborazione dataset

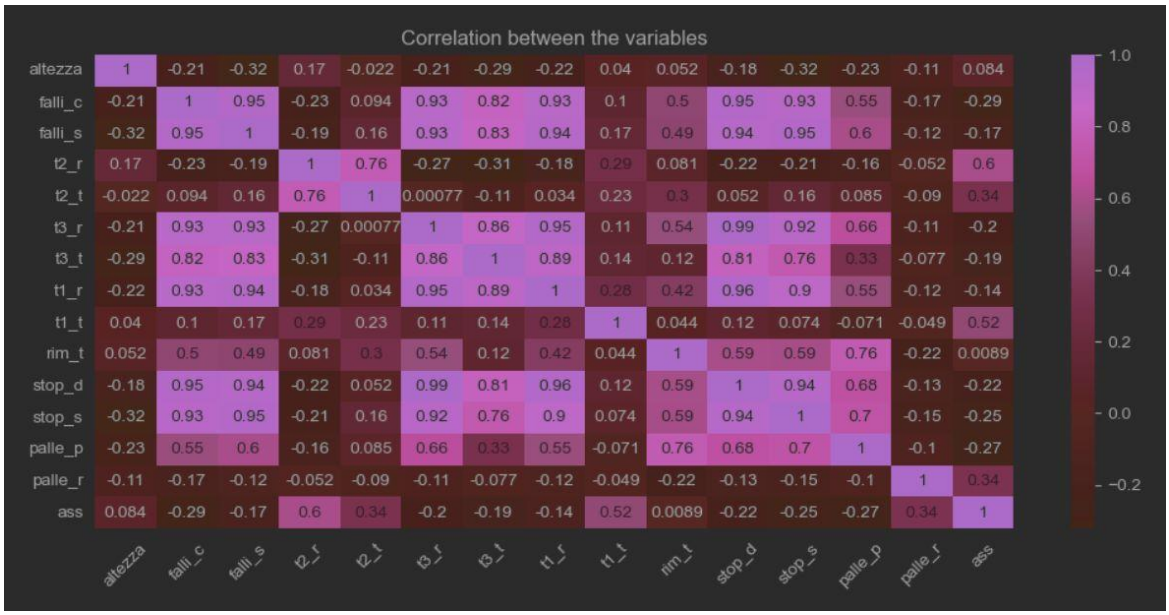
I dati sono stati elaborati da partita21.json e partita.json mediante l'uso dello script c**rea_csv_squadre.py**

- **squadre.csv**: che contiene casa_puntiFatti, casa_puntiSubiti, casa_falliFatti, casa_pallePerse, casa_t2_r, ospiti_puntiFatti, ospiti_puntiSubiti, ospiti_falliFatti, ospiti_pallePerse, ospiti_t2_r, result per la predizione dei risultati della partita tra squadra ospite e squadra in casa

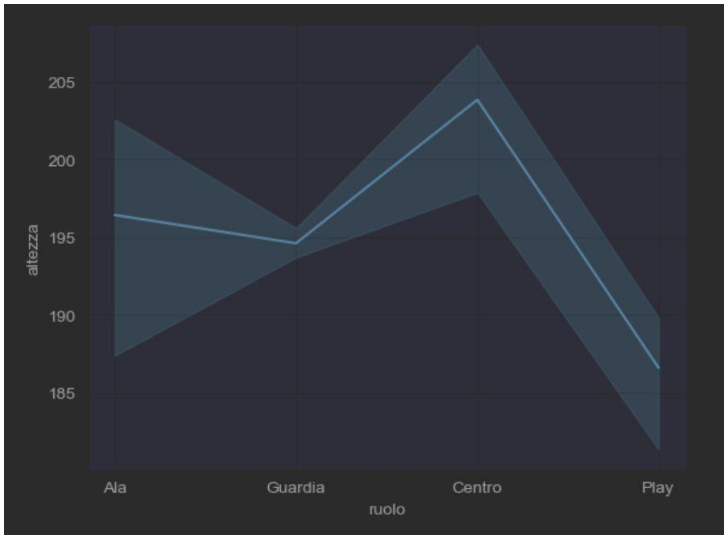
- **giocatori_corretti.csv**: che contiene le stesse informazioni di giocatori.csv ma con i campi nulli sovrascritti con valori corretti calcolati tramite **Apprendimento Supervisionato** il quale andava a capire quali fossero i ruoli dei giocatori non etichettati e li andava a sovrascrivere con i valori corretti presi dalle caratteristiche del giocatore

Studio dataset

Una matrice di correlazione è semplicemente una tabella che mostra i coefficienti di correlazione per diverse variabili. La matrice rappresenta la correlazione tra tutte le possibili coppie di valori in una tabella. È un potente strumento per riassumere un set di dati di grandi dimensioni e per identificare e visualizzare modelli nei dati forniti.



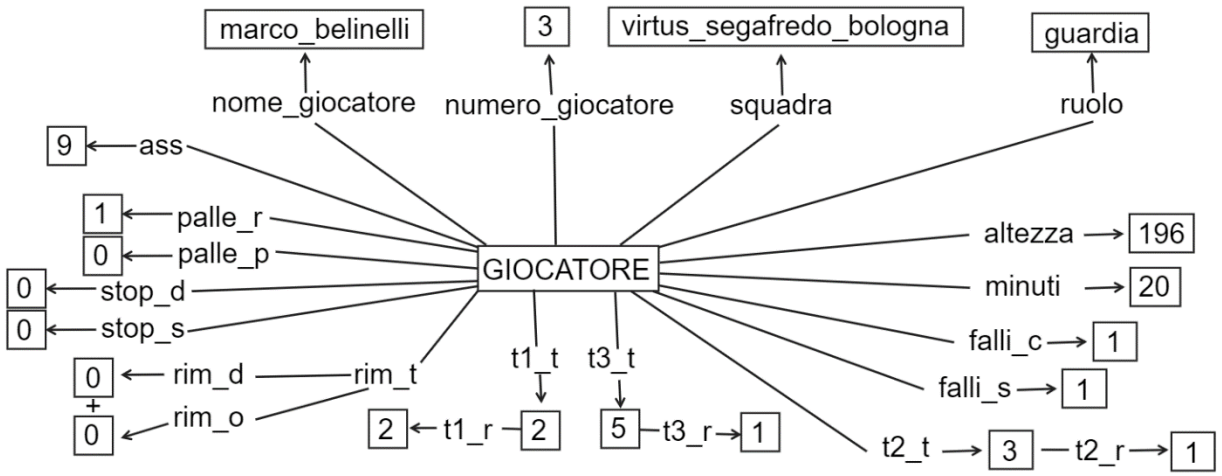
L'esperto del dominio fa notare che il ruolo è particolarmente legato all'altezza dei giocatori



KNOWLEDGE BASE

La Knowledge Base è stata impostata usando come base le informazioni presenti nel dataset **giocatori.csv**, nel seguente modo:

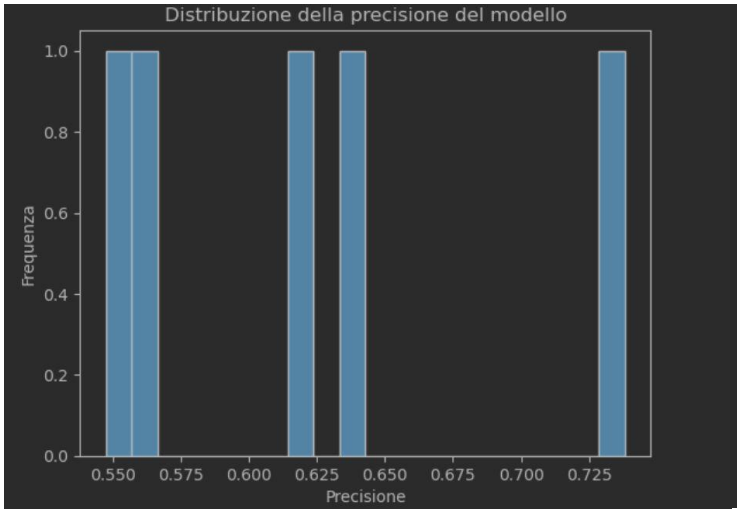
- ad ogni giocatore è stato assegnato un ID incrementale
- per ogni attributo e valore corrispondente è stato **asserito un predicato** con il formato: attributo(id, valore) (ad esempio: nome_giocatore(100,filippo_giberti)
- sono state create delle **regole** per eseguire le query ed effettuare dei calcoli per poi asserirli
- per alcuni predicati sono stati applicati dei pesi ai vari attributi calcolati dal Value Index Rating (VIR):
  VIR= [(Punti fatti + ASS*1,5 + Palle recuperate + stoppate date * 0,75 + rimbalzi offensivi * 1,25 + rimbalzi difensivi * 0,75 + tiri da tre realizzati/2 + falli subiti/2 – falli commessi/2 – (tiri da tre tentati + tiri da due tentati) * 0,75 – palle perse – tiri liberi tentati/2) * Valutazione minuti giocati]

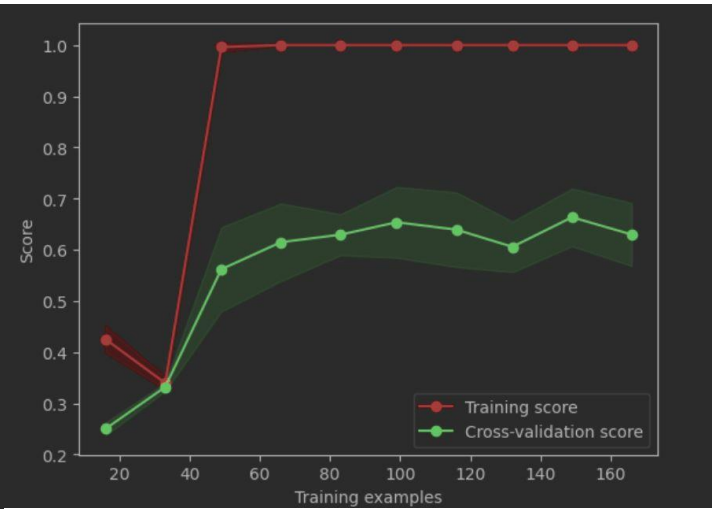


Questa è la rappresentazione della **rete semantica** di un giocatore preso in esempio

Correzione del dataset

Per la correzione del dataset e dunque per la creazione di giocatori_corretti.csv abbiamo applicato un algoritmo di **Hist Gradient Boosting** in quanto è possibile dargli dati incompleti per la predizione del ruolo di un giocatore.
La cui accuratezza è del 62 più o meno 14.





PREDIZIONE IN MANCANZA DI ALCUNI DATI - Gradient Boosting

Implementa una valutazione di validazione incrociata (cross-validation) per un classificatore basato su gradient boosting (HistGradientBoostingClassifier) utilizzando la libreria scikit-learn.

Viene creato un oggetto HGBC che rappresenta il classificatore basato su gradient boosting. Viene impostato il numero massimo di iterazioni (max_iter) a 150 e la velocità di apprendimento (learning_rate) a 0,1. Il classificatore viene addestrato sui dati x e y utilizzando il metodo fit().

Viene quindi eseguita la valutazione di cross-validation sul classificatore addestrato e sui dati x e y utilizzando il metodo cross_val_score(). Viene specificato che la validazione incrociata (cv=5) deve essere eseguita su 5 fette.

Infine, viene stampata la precisione media della cross-validation, insieme alla deviazione standard (espressa come intervallo di fiducia). La precisione viene calcolata come media delle valutazioni ottenute in ogni fetta e viene espresso come percentuale con due cifre decimali.

PREDIZIONE CON DATI COMPLETI

Sono stati utilizzati vari algoritmi di predizione i quali necessitano di dati completi, ed hanno risultati migliori. Sono i seguenti:

- Decision Tree (Albero di Ricerca)
- Random Forest Classifier
- Logistic Regretion
- Neural Network
- KNN

PREDIZIONE CON DATI COMPLETI – Decision Tree

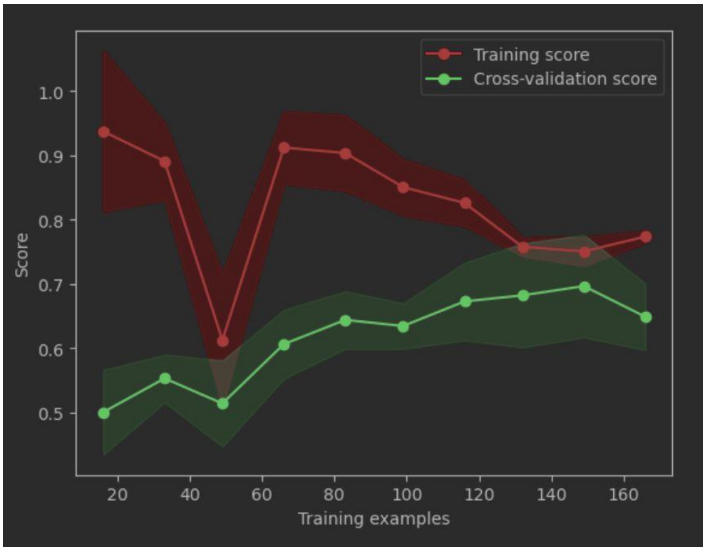
Implementa una grid search per la selezione dei parametri ottimali di un classificatore decisionale basato su alberi (DecisionTreeClassifier) utilizzando la libreria scikit-learn.

Il codice crea un oggetto param_grid che rappresenta la griglia di parametri da testare. In questo caso, la grid search eseguirà la validazione incrociata (cv=5) per testare la profondità massima dell'albero di decisione (max_depth) con valori di 3, 5, 7, 9, 11 e None e il numero minimo di campioni richiesti per dividere un nodo (min_samples_split) con valori di 2, 10, 30 e 50.

Viene quindi creato un oggetto grid_search che rappresenta l'effettiva grid search. Questo oggetto viene addestrato sui dati x e y utilizzando il metodo fit().

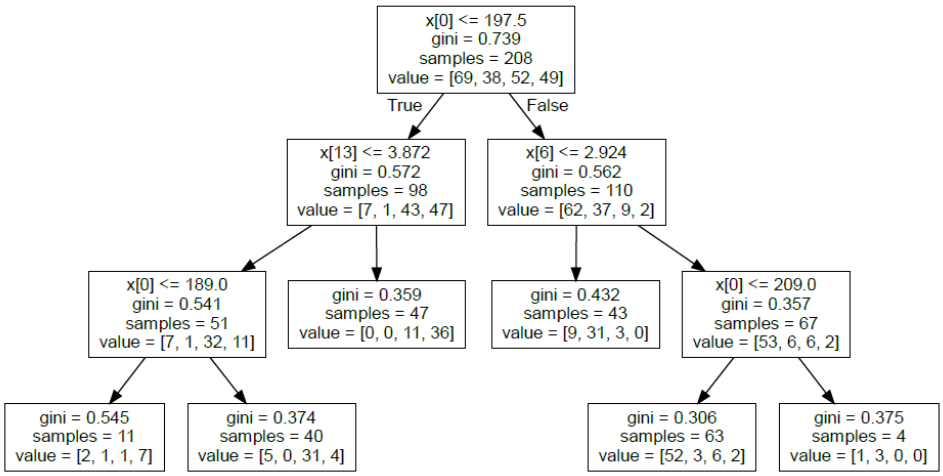
Infine, viene stampato il risultato della grid search: i parametri ottimali (best_params_) e la performance ottenuta con questi parametri (best_score_), espresso come percentuale con due cifre decimali.

Questo modello non può lavorare con valori nulli. Dunque si scarta in caso si abbiano giocatori con valori nulli.



Best parameters found: {'max_depth': 3, 'min_samples_split': 50}
Best score: 0.67

L'albero generato è il seguente:



PREDIZIONE CON DATI COMPLETI – Random Forest Classifier

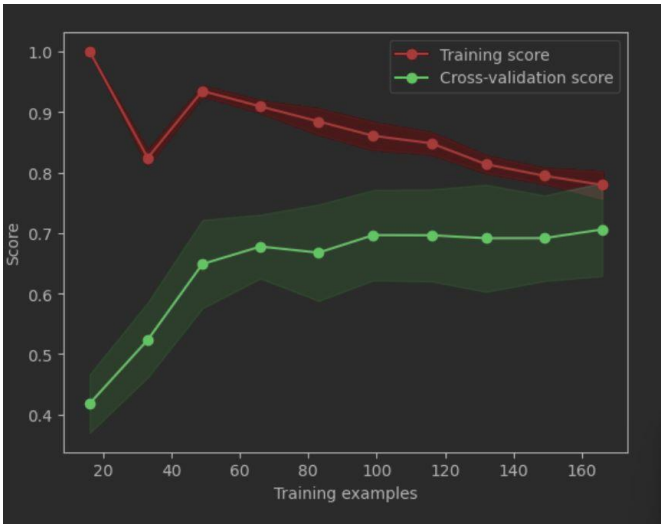
Implementa una valutazione di validazione incrociata (cross-validation) per un classificatore a foresta casuale (RandomForestClassifier) utilizzando la libreria scikit-learn.

Viene definito un intervallo di valori da testare per max_depth e random_state, che rappresentano rispettivamente la profondità massima degli alberi di decisione e il seme utilizzato dal generatore di numeri casuali all'interno del classificatore.

Viene quindi eseguita una valutazione di cross-validation su ogni combinazione di valori di max_depth e random_state e viene calcolata la media delle valutazioni ottenute in ogni fetta. I risultati vengono memorizzati in una lista di tuple scores.

La lista scores viene quindi convertita in un array numpy e viene individuato l'indice del punteggio massimo utilizzando il metodo np.argmax(). Viene quindi ottenuto il valore di max_depth e random_state corrispondenti al punteggio massimo.

Infine, viene addestrato un classificatore a foresta casuale utilizzando i valori di max_depth e random_state ottenuti in precedenza e viene eseguita una valutazione di cross-validation sul nuovo classificatore. Vengono infine stampati i risultati della cross-validation, ovvero la media delle valutazioni e la deviazione standard.



Best max_depth value: 2.0
Best random_state value: 256.0
Cross-validation scores: [0.73809524 0.73809524 0.80952381 0.65853659 0.58536585]
Average cross-validation score: 0.71 +/- 0.08

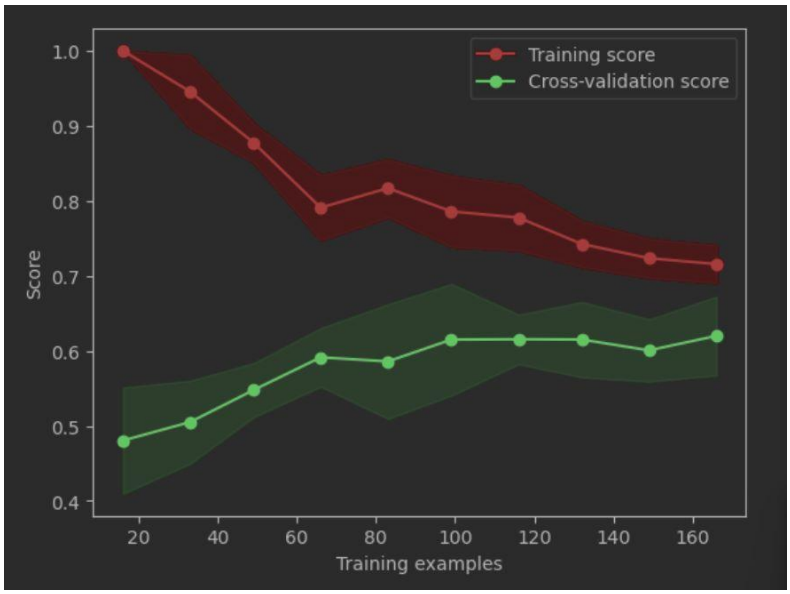
PREDIZIONE CON DATI COMPLETI – Logistic Regretion

Utilizza la libreria scikit-learn per addestrare un modello di regressione logistica (LogisticRegression) sui dati x e y. Prima di addestrare il modello, i dati x vengono scalati utilizzando la classe StandardScaler per garantirne la convergenza.

L'oggetto LR rappresenta il modello di regressione logistica, che viene addestrato utilizzando il metodo fit() sui dati scalati x_scal e y.

Viene quindi eseguita una validazione incrociata (cross-validation) sul modello addestrato utilizzando il metodo cross_val_score(). La validazione incrociata viene eseguita su 5 fette (cv=5) e la precisione viene calcolata come media delle valutazioni ottenute in ogni fetta.

Infine, viene stampata la precisione media della validazione incrociata insieme alla deviazione standard (espresse come intervallo di fiducia), che vengono calcolate come media e deviazione standard delle valutazioni ottenute in ogni fetta. La precisione viene espresso come percentuale con due cifre decimali.



Accuracy: 0.61 (+/- 0.11)

PREDIZIONE CON DATI COMPLETI – Neural Network

Implementa una ricerca dei parametri per un classificatore basato su reti neurali utilizzando la libreria Keras e scikit-learn.

Innanzitutto, viene effettuata la codifica one-hot delle etichette (y) utilizzando la classe LabelEncoder e la funzione to_categorical di Keras. Questo rende le etichette da un vettore con valori interi a una matrice con valori binari.

Viene quindi definita la funzione "create_model" che rappresenta il modello di rete neurale da utilizzare. Il modello è una rete neurale sequenziale con tre strati: il primo strato è una densa (fully connected) con "neurons_first_layer" neuroni, il secondo strato è una densa con

"neurons_second_layer" neuroni e l'ultimo strato è una densa con 4 neuroni e una funzione di attivazione softmax.

Il modello viene poi compilato con la funzione di perdita "categorical_crossentropy" e l'ottimizzatore "adam".

Viene quindi creato un oggetto KerasClassifier che rappresenta il classificatore basato su reti neurali e viene specificato che il modello deve essere creato utilizzando la funzione "create_model". Viene inoltre specificato che l'addestramento del modello deve essere effettuato per 1000 epoche con un batch size di 32 e con la verbosità impostata a 0 (nessuna stampa a schermo).

Viene quindi creata una griglia di parametri con "neurons_first_layer" che può assumere i valori [32, 64, 128] e "neurons_second_layer" che può assumere i valori [16, 32, 64].

Viene quindi creato un oggetto GridSearchCV che rappresenta la ricerca dei parametri sulla griglia specificata e il modello. Viene specificato che la ricerca dei parametri deve essere effettuata su tutti i processori disponibili (-1).

Infine, viene effettuato l'addestramento del modello utilizzando la funzione fit() e viene stampato il risultato con i migliori parametri trovati e il miglior punteggio ottenuto.

Migliori parametri: {'neurons_first_layer': 64, 'neurons_second_layer': 16}
Miglior punteggio: 0.6872241616249084

PREDIZIONE CON DATI COMPLETI – KNN

Implementa un classificatore K-Nearest Neighbors utilizzando la libreria scikit-learn. In particolare, viene utilizzato l'algoritmo di ricerca sulle griglie (GridSearchCV) per selezionare il valore ottimale di K.

La prima riga crea un'istanza del classificatore KNeighborsClassifier. Poi viene importata la classe GridSearchCV dalla libreria scikit-learn.

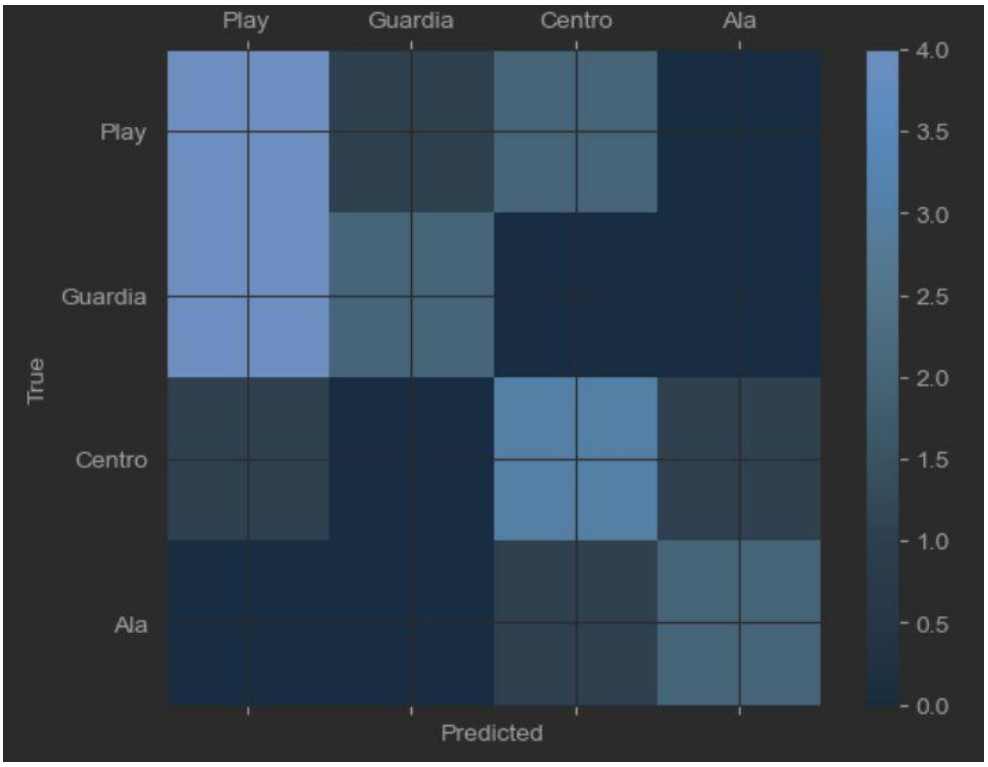
In seguito, viene definita una lista di valori compresi tra 1 e 31 per il parametro K. Questi valori verranno utilizzati come valori possibili per il parametro n_neighbors durante la ricerca sulla griglia.

La funzione di ricerca sulla griglia viene quindi creata specificando l'oggetto knn come modello, il dizionario param_grid con le opzioni per n_neighbors, e altri parametri come il numero di pieghe per la validazione incrociata (cv=10), la metrica di valutazione (scoring='accuracy') e la visualizzazione dei risultati di addestramento (return_train_score=False, verbose=1).

Infine, il modello viene addestrato utilizzando i dati di addestramento (x_train, y_train) e la funzione fit(). Il parametro K ottimale viene quindi selezionato come il miglior parametro n_neighbors dal risultato della ricerca sulle griglie.
L'accuratezza del modello addestrato viene quindi calcolata e stampata a schermo.

Fitting 10 folds for each of 30 candidates, totalling 300 fits
n_neighbors: 17
Accuracy for our training dataset with tuning is: 69.50%

Valutiamo il modello utilizzando i dati di test, a questo scopo abbiamo impostato una matrice di confusione per aiutarci a scoprire veri positivi, veri negativi, falsi positivi e falsi negativi



SISTEMA ESPERTO PER OPERAZIONI SULLA KNOWLEDGE BASE

Il sistema esperto opera tramite la **KB** da noi precedentemente creata, attraverso le regole scritte in Prolog, che è un linguaggio di programmazione di paradigma logico, che permette di esprimere, mediante regole, un ragionamento esprimibile tramite la logica di primo livello. Con le regole implementate, il sistema esperto permette all'utente di effettuare le seguenti operazioni:

```
evaluate_all_pl(Pl) :-
    findall(Player, is_play(Player), Players),

    get_all_height(Players, Height_list),
    get_all_minutesPlayed(Players, MinutesPlayed_list),
    get_all_fouls_c(Players, Fouls_c_list),
    get_all_fouls_s(Players, Fouls_s_list),
    get_all_T2(Players, T2_list),
    get_all_T2T(Players, T2T_list),
    get_all_T3(Players, T3_list),
    get_all_T3T(Players, T3T_list),
    get_all_T1(Players, T1_list),
    get_all_T1T(Players, T1T_list),
    get_all_rebaunds(Players, Rebaunds_list),
    get_all_stopD(Players, StopD_list),
    get_all_stopS(Players, StopS_list),
    get_all_palleP(Players, PalleP_list),
    get_all_palleR(Players, PalleR_list),

    calc_max(Height_list, Max_height),
    calc_max(MinutesPlayed_list, Max_minutesPlayed),
    calc_max(Fouls_c_list, Max_Fouls_c),
    calc_max(Fouls_s_list, Max_Fouls_s),
```

Nella precedente immagine vengono mostrate le 3 principali fasi effettuate per ogni ruolo (in questo caso si è scelto di mostrare come esempio quelle per i playmaker ma il funzionamento è analogo per tutti gli altri ruoli) che sono:

1) Raccolta di tutti i giocatori con lo stesso ruolo tramite il predicato "findall";
2) Recupero di tutti i valori di un particolare attributo dalla lista di giocatori creata al punto 1);
3) Calcolo del massimo per ogni attributo;

Visualizzare la classifica dei giocatori di un determinato ruolo

Selezionato un ruolo l'utente potrà visualizzare la classifica di giocatori e per la valutazione di ogni singolo giocatore vengono selezionate diverse caratteristiche chiave per il ruolo scelto a cui sono stati applicati dei pesi per dare maggior o minore risalto ad una determinata caratteristica; ad esempio, per i playmaker vengono considerate le seguenti caratteristiche con i rispettivi pesi:

- Assist →1,5
- Palle recuperate → 1
- Stoppate date → 0,75
- Rimbalzi offensivi → 1,25
- Rimbalzi difensivi → 0,75
- Tiri da tre realizzati → 0.5
- Falli subiti → 0.5
- Falli commessi → 0.5
- Tiri da tre tentati → 0.75
- Tiri da due tentati → 0,75
- Palle perse → 1
- Tiri liberi tentati → 0.5

```
Cosa vuoi confrontare? (pl/cn/al/gd): >? pl
Vuoi la lista dei giocatori?: (y/n) >? y
shabazz_napier: 1.2417045454545454
andrea_cinciarini: 1.2123722943722945
diego_flaccadori: 1.1078111471861474
julyan_stone: 1.0698295454545454
semaj_christon: 1.0596731601731602
```

Il calcolo della classifica viene effettuato tramite una regola Prolog creata ad hoc evaluate_all_pl(), in base al ruolo di cui si richiede di visualizzare la classifica, che prima prende tutti i giocatori relativi ad un ruolo e poi effettua la valutazione di ogni giocatore passato in input.

```
evaluate_all_pl(Pl) :-

    get_all_height(Players, Height_list),
```

Predicato che prende tutti i playmaker e ne calcola la valutazione restituendolo come lista di coppie di valori, di cui il cui primo valore è il nome del giocatore e il secondo è il valore calcolato.

```
    nome_giocatore(H, Name),
    append([[Name, Eval_local]], Eval_down, Res).
```

Predicato che calcola ricorsivamente per ogni giocatore della lista in input la valutazione, ne trova il nome e lo inserisce nella lista in output

```
Eval is
    Eval_height +
    (
        Fouls_c_divided /2 +
        Fouls_s_divided /2 +
        T2_divided *1 +
        (T2T_divided - T2_divided) *3/4 +
        T3_divided *1 +
        (T3T_divided - T3_divided) *3/4 +
        T1_divided *1+
        (T1T_divided - T1_divided) /2 +
        Rebaunds_divided *1 +
        StopD_divided *3/4 +
        StopS_divided *2/10 +
        PalleP_divided *1 +
        PalleR_divided *1
    ) * Eval_mp.
```

Questa parte della regola evaluation_pl (che effettua la valutazione di un playmaker) è responsabile del calcolo basandosi sui valori di tutte le variabili raccolte in precedenza in questa stessa regola. In questo calcolo vengono effettuati i prodotti in base ai pesi assegnati da noi ad ogni caratteristica in base alla sua importanza.

Confrontare due giocatori dello stesso ruolo

Questa sezione del programma permette di effettuare un confronto testa a testa tra due giocatori dello stesso ruolo mostrando, tramite percentuali, da quale parte pende la bilancia (valutazione maggiore), per ognuno la valutazione viene calcolata tramite le regole usate precedentemente.

```
def comparePlayers(plList):
    p1 = input("Dammi il primo giocatore: ")
    p1Converted = p1
    p1Converted = p1Converted.replace(" ", "_")
    p1Converted = p1Converted.lower()
    p2 = input("Dammi il secondo giocatore: ")
    p2Converted = p2
    p2Converted = p2Converted.replace(" ", "_")
    p2Converted = p2Converted.lower()
    pl = []
    for x in plList:
        if str(x[0]) == p1Converted or str(x[0]) == p2Converted:
            pl.append(x)
    percP1 = f'{pl[0][1] / (pl[0][1] + pl[1][1]) * 100:.2f} %'
    percP2 = f'{pl[1][1] / (pl[0][1] + pl[1][1]) * 100:.2f} %'
    print(str(pl[0][0]) + ": " + percP1)
    print(str(pl[1][0]) + ": " + percP2)
```

```
Cosa vuoi confrontare? (pl/cn/al/gd): >? pl
Vuoi la lista dei giocatori?: (y/n) >? n
Dammi il primo giocatore: >? andrea_cinciarini
Dammi il secondo giocatore: >? corey_davis
andrea_cinciarini: 55.33 %
corey_davis: 44.67 %
```

Dall'implementazione della regola si può notare che per i due giocatori si ottiene l'Id in base al nome, si calcola la valutazione dei due giocatori e, per ogni valutazione, si effettua la percentuale della valutazione sul totale.

```
scelta = input("Cosa vuoi confrontare? (pl/cn/al/gd): ")
if scelta == 'pl':
    scelta = input("Vuoi la lista dei giocatori?: (y/n) ")
    if scelta == 'y':
        for pl in pl_list:
            print(str(pl[0]) + ": " + str(pl[1]))
    comparePlayers(pl_list)
elif scelta == 'cn':
    scelta = input("Vuoi la lista dei giocatori?: (y/n) ")
    if scelta == 'y':
        for pl in cn_list:
            print(str(pl[0]) + ": " + str(pl[1]))
    comparePlayers(cn_list)
elif scelta == 'al':
    scelta = input("Vuoi la lista dei giocatori?: (y/n) ")
    if scelta == 'y':
        for pl in al_list:
            print(str(pl[0]) + ": " + str(pl[1]))
    comparePlayers(al_list)
elif scelta == 'gd':
```

PREDIZIONI DEI RISULTATI DELLE PARTITE

Per effettuare la predizione dei risultati delle partite ci avvaliamo di due sistemi:

- Random Forest Classifier
- Bayesian Network

Predizione mediante Random Forest Classifier

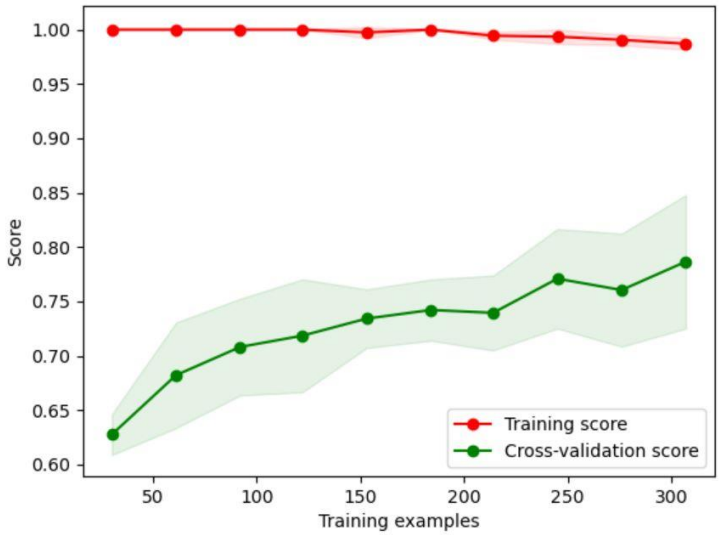
Viene definito un intervallo di valori da testare per max_depth e random_state, che rappresentano rispettivamente la profondità massima degli alberi di decisione e il seme utilizzato dal generatore di numeri casuali all'interno del classificatore.

Viene quindi eseguita una valutazione di cross-validation su ogni combinazione di valori di max_depth e random_state e viene calcolata la media delle valutazioni ottenute in ogni fetta. I risultati vengono memorizzati in una lista di tuple scores.

La lista scores viene quindi convertita in un array numpy e viene individuato l'indice del punteggio massimo utilizzando il metodo np.argmax(). Viene quindi ottenuto il valore di max_depth e random_state corrispondenti al punteggio massimo.

Infine, viene addestrato un classificatore a foresta casuale utilizzando i valori di max_depth e random_state ottenuti in precedenza e viene eseguita una valutazione di cross-validation sul nuovo classificatore. Vengono infine stampati i risultati della cross-validation, ovvero la media delle valutazioni e la deviazione standard

Best max_depth value: 2.0
Best random_state value: 256.0
Cross-validation scores: [0.73809524 0.73809524 0.80952381 0.65853659 0.58536585]
Average cross-validation score: 0.71 +/- 0.08




```
Best max_depth value: 6.0
Best random_state value: 16.0
Cross-validation scores: [0.87012987 0.7012987  0.76623377 0.75324675 0.84210526]
Average cross-validation score: 0.79 +/- 0.06
```

Predizione mediante Bayesian Network

Crea un modello Bayesiano utilizzando la libreria pgmpy. Viene utilizzato un dataset contenente le statistiche di un gioco di basket, che include informazioni sulle squadre di casa e sugli ospiti, come i punti fatti e subiti, i falli, le palle perse e recuperate e i tiri realizzati e subiti da 2 e 3 punti.

Il codice inizia importando le librerie necessarie e leggendo i dati da un file CSV. I dati vengono quindi elaborati attraverso la discretizzazione utilizzando un algoritmo di equal width. La discretizzazione consiste nel dividere i valori di ogni colonna in intervalli di larghezza uguale e assegnare ad ogni valore una lettera maiuscola (A, B, C, ecc.).

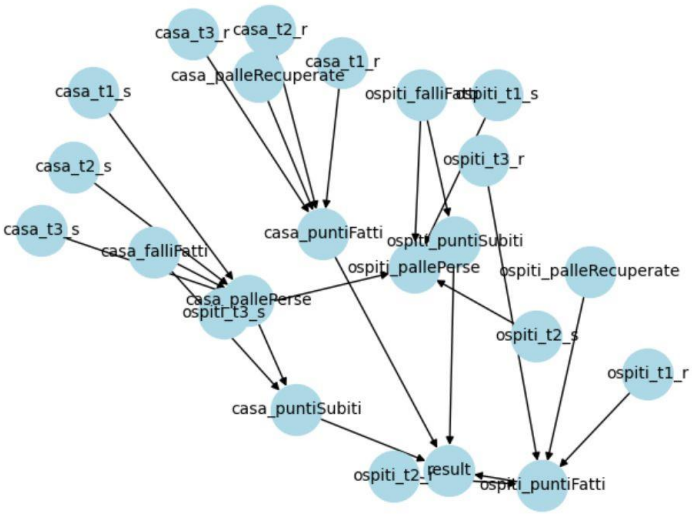
La funzione val_counter conta il numero di volte che ogni lettera maiuscola appare in una colonna specifica del DataFrame. La funzione multi_val_counter conta il numero di righe che soddisfano determinate condizioni su più colonne.

Le funzioni generate_tuples e calc_matrix_with_cols sono utilizzate per creare la matrice CPD (probabilità condizionata) per il modello Bayesiano. La matrice CPD descrive la probabilità di ottenere un determinato valore in una colonna in base ai valori in altre colonne.

```
("casa_puntiFatti", "result"),
("casa_puntiSubiti", "result"),
("casa_falliFatti","casa_puntiSubiti"),
("casa_pallePerse","casa_puntiSubiti"),
("casa_palleRecuperate", "casa_puntiFatti"),
('casa_falliFatti', 'casa_pallePerse'),
("casa_t2_s","casa_pallePerse"),
("casa_t3_s","casa_pallePerse"),
("casa_t1_s","casa_pallePerse"),
("casa_t2_r","casa_puntiFatti"),
("casa_t3_r","casa_puntiFatti"),
("casa_t1_r","casa_puntiFatti"),
```

```
("ospiti_puntiFatti", "result"),
("ospiti_puntiSubiti", "result"),
("ospiti_falliFatti","ospiti_puntiSubiti"),
("ospiti_pallePerse", "ospiti_puntiSubiti"),
("ospiti_palleRecuperate", "ospiti_puntiFatti"),
('ospiti_falliFatti','ospiti_pallePerse'),
("ospiti_t2_s","ospiti_pallePerse"),
("ospiti_t3_s","ospiti_pallePerse"),
("ospiti_t1_s","ospiti_pallePerse"),
("ospiti_t2_r","ospiti_puntiFatti"),
("ospiti_t3_r","ospiti_puntiFatti"),
("ospiti_t1_r","ospiti_puntiFatti"),
```

Infine, il modello Bayesiano viene creato utilizzando la classe BayesianNetwork e la matrice CPD. La classe VariableElimination viene utilizzata per effettuare inferenze sul modello.



Fornisce una solida base per la creazione di un modello Bayesiano utilizzando la libreria pgmpy e per la sua applicazione a un dataset specifico.

```
actual_result: +-----------+---------------+
| result    |   phi(result) |
+===========+===============+
| result(0) |        0.4353 |
+-----------+---------------+
| result(1) |        0.5647 |
+-----------+---------------+
Accuracy: 0.6363636363636364
actual_result: +-----------+---------------+
| result    |   phi(result) |
+===========+===============+
| result(0) |        1.0000 |
+-----------+---------------+
| result(1) |        0.0000 |
+-----------+---------------+
expected_result: victory, 0,
actual_result: +-----------+---------------+
| result    |   phi(result) |
+===========+===============+
| result(0) |        0.4353 |
+-----------+---------------+
| result(1) |        0.5647 |
+-----------+---------------+
expected_result: victory, 0,
```

CONCLUSIONE

I risultati ottenuti, per quanto riguarda il sistema esperto, sono ritenuti più che soddisfacenti e vicini alla realtà, nonostante le difficoltà riscontrate nella creazione della knowledge base.

Nella parte del progetto riguardante l'apprendimento automatico supervisionato, l'accuratezza massima raggiunta nonostante siano stati usati svariati algoritmi e selezione delle migliori features, è stata del 79 %, il che dimostra che la predizione dei risultati delle partite di pallacanestro è ancora un task molto complesso e che non offre garanzie sui risultati.

Nonostante fossimo a conoscenza della difficoltà di questo tipo di obiettivo, ci siamo ugualmente voluti cimentare, per dimostrare le nostre abilità nello studio dell'ingegneria della conoscenza.

Prossimamente, questo progetto, potrebbe essere migliorato con un nuovo dataset con una maggiore accuratezza e aggiungendo altre funzionalità al sistema esperto.

COMPONENTI GRUPPO:
Lepore Nicola: 739765
Federica de Virgilio: 735274