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 GESTIONE DI UN CLUB E PREDIZIONE DELLE PARTITE

Gruppo composto da:
Amendolagine Luigi Pio: 725079
Andriani Claudio: 725613
Coppolecchia Dario: 725614

Anno Accademico 2021-2022

SOMMARIO

1. 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) *Visualizzare la classifica* dei giocatori di un determinato ruolo (classifica che mostra i giocatori dal migliore al peggiore)
2) Visualizzare i *giocatori suggeriti* in base alle preferenze selezionate (ad esempio, giocatori al di sotto di una certa età, ecc…)
3) Visualizzare la *predizione della classifica* di uno dei top cinque campionati europei
4) *Confrontare due giocatori* dello stesso ruolo
5) Trovare i *punti deboli* della squadra
6) Generare la formazione titolare di una determinata squadra
7) Predire il risultato di una partita

Per quanto riguarda la predizione dei risultati delle partite, 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, mentre per gli altri task è stato progettato un **sistema esperto** tramite una *knowledge base* e *programmazione logica*.

2. 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, mentre 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 Prolog è stato utilizzato come IDE **VSCode** e anche l'interprete online di SWI-Prolog per effettuare test in modo rapido e sicuro su predicati generici. Mentre per quanto riguarda l'esecuzione degli algoritmi di Machine Learning è stato utilizzato Google Colab. Il vantaggio del servizio è che fornisce accesso gratuito a elevate risorse computazionali, incluse GPU e TPU. 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.

3. ATTIVITA' DI RICERCA E 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 tramite le API del sito SofaScore; i dataset generati sono:

- ***player_stats.csv***: contenente le informazioni anagrafiche e le statistiche dei giocatori dei top 5 campionati europei (Serie A, Premier League, Bundesliga, Ligue 1 e LaLiga) riguardanti la stagione 21/22

name	team	competition	age	preferredFoot	height	role	mainPosition	secondPosition	country
Victor Osimhen	Napoli	Serie A Tim	23	Right	186	F	ST	None	Nigeria
Hirving Lozano	Napoli	Serie A Tim	27	Right	175	F	RW	LW	Mexico
Giovanni Simeone	Napoli	Serie A Tim	27	Right	179	F	ST	None	Argentina
Giacomo Raspadori	Napoli	Serie A Tim	22	Both	172	F	ST	AM	Italy
Matteo Politano	Napoli	Serie A Tim	29	Left	171	F	RW	None	Italy
Alessio Zerbin	Napoli	Serie A Tim	23	Right	182	F	LW	None	Italy
Piotr Zieliński	Napoli	Serie A Tim	28	Both	180	M	AM	MC	Poland
Khvicha Kvaratskhelia	Napoli	Serie A Tim	21	Both	181	M	LW	None	Georgia
André Zambo Anguissa	Napoli	Serie A Tim	26	Right	184	M	MC	DM	Cameroon
Tanguy Ndombélé	Napoli	Serie A Tim	25	Right	181	M	MC	DM	France

- ***unique_tournaments.csv***: contenente l'elenco delle competizioni presenti nel dataset precedente, utile successivamente per assegnare ad ogni competizione il rispettivo peso in base alla rilevanza dello stesso

Serie A	Italy	19
Premier League	England	19
Ligue 1	France	19
LaLiga	Spain	19
Bundesliga	Germany	19
Liga Portugal	Portugal	14
Eredivisie	Netherlands	12
Superliga	Denmark	10
Premier Liga	Russia	10
Pro League	Belgium	10

- ***team_statistics.csv***: contenente le statistiche complessive delle squadre nella stagione 21/22 dei seguenti campionati: Seria A, Premier League, Bundesliga, LaLiga, Ligue 1, Eredivise, Liga Portugal, Pro League, 2. Bundesliga, LaLiga 2, Championship, Serie B e Ligue 2; le statistiche per squadra verranno successivamente mediate in base al totale delle partite disputate.

name	goalsScored	goalsConceded	ownGoals	assists	shots	penaltyGoals	penaltiesTaken
Milan	69	31	2	41	600	5	8
Inter	84	32	1	59	678	7	11
Napoli	74	31	1	53	579	10	14
Juventus	57	37	1	41	523	5	6
Lazio	77	58	1	53	454	7	9
Roma	59	43	2	35	601	7	9
Fiorentina	59	51	2	34	513	9	12
Atalanta	65	48	3	49	604	5	6
Hellas Verona	65	59	2	47	463	7	8
Torino	46	41	0	32	476	6	6
Sassuolo	64	66	1	41	576	7	7
Udinese	61	58	1	40	509	3	4
Bologna	44	55	2	33	439	4	5
Empoli	50	70	6	29	497	7	7
Sampdoria	46	63	2	30	391	2	4
Spezia	41	71	3	25	386	3	5
Salernitana	33	78	2	19	431	4	5
Cagliari	34	68	1	27	424	3	4
Genoa	27	60	2	19	404	6	7
Venezia	34	69	2	26	355	3	5
Manchester City	99	26	0	63	713	7	9
Liverpool	94	26	1	71	729	7	8
Chelsea	76	33	1	52	592	8	9

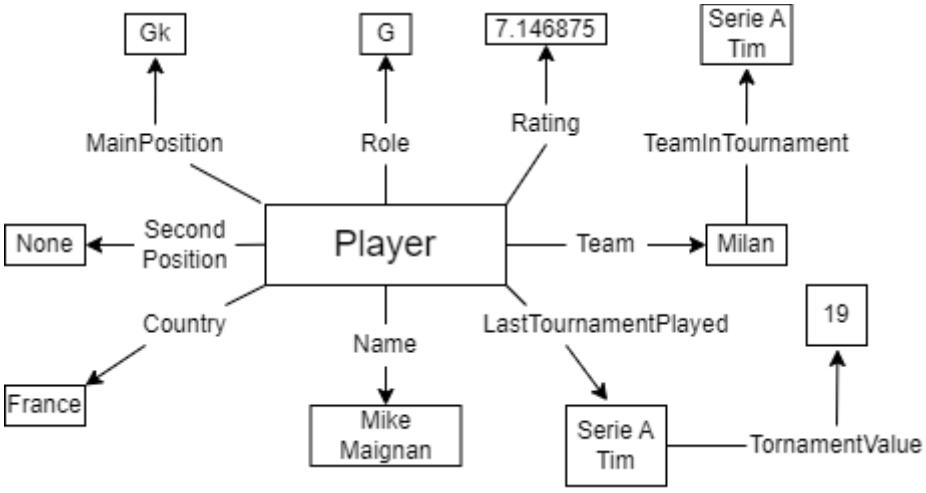
- ***matches.csv***: contenente i risultati delle partite disputate nella stagione 21/22 nei campionati precedentemente citati

homeTeam	awayTeam	homeScore	awayScore
Inter	Empoli	4	2
Genoa	Juventus	2	1
Torino	Napoli	0	1
Sassuolo	Udinese	1	1
Lazio	Sampdoria	2	0
Spezia	Atalanta	1	3
Venezia	Bologna	4	3
Salernitana	Cagliari	1	1
Hellas Verona	Milan	1	3
Fiorentina	Roma	2	0
Empoli	Salernitana	1	1
Udinese	Spezia	2	3
Hellas Verona	Torino	0	1
Roma	Venezia	1	1

4. KNOWLEDGE BASE

La Knowledge Base è stata impostata usando come base le informazioni presenti nel dataset ***player_stats.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: *name(124, mike_maignan)*)
- 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* che durante l'esecuzione del programma possono essere modificati dall'utente. Questi pesi vengono caricati dal file chiamato ***weights.json***.



Questa è la rappresentazione della **rete semantica** di un giocatore preso in esempio (sono stati rappresentati solo gli attributi più importanti per non appesantire il diagramma).

5. 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:

```
Digitare un numero per effettuare una delle seguenti operazioni:
1) Visualizzare la classifica dei giocatori di un determinato ruolo
2) Visualizzare i giocatori suggeriti in base alle preferenze selezionate
3) Visualizzare la predizione della classifica di uno dei top cinque campionati europei
4) Confrontare due giocatori dello stesso ruolo
5) Trovare i punti deboli della squadra
6) Generare la formazione titolare di una determinata squadra
7) Predire il risultato di una partita
0) Uscire dall'applicazione
```

Prima di mostrare il menù vengono effettuate due operazioni:

- viene effettuata una query chiamata "*calc_and_assert_max_attr*" che si occupa di asserire i massimi valori per ogni ruolo (equivalentemente a trattare i massimi come costanti globali) per rendere le operazioni di calcolo delle valutazioni dei giocatori più veloci o senza doverle passare come parametri alle regole per le valutazioni.

```
calc_and_assert_max_attr() :-

    % GK
    findall(Player, is_goal_keeper(Player), Players_gk),

    get_all_age(Players_gk, Age_list_gk),
    get_all_height(Players_gk, Height_list_gk),
    get_all_minutesPlayed(Players_gk, MinutesPlayed_list_gk),

    calc_max(Age_list_gk, Max_age_gk),
    calc_max(Height_list_gk, Max_height_gk),
    calc_max(MinutesPlayed_list_gk, Max_minutesPlayed_gk),

    assertz(get_max_age_gk(Max) :- Max =.. [Max_age_gk]),
    assertz(get_max_height_gk(Max) :- Max =.. [Max_height_gk]),
    assertz(get_max_minutesPlayed_gk(Max) :- Max =.. [Max_minutesPlayed_gk]),
```

  Nella precedente immagine vengono mostrate le 4 principali fasi effettuate per ogni ruolo (in questo caso si è scelto di mostrare come esempio quelle per i portieri ma il funzionamento è analogo per tutti gli altri ruoli) per l'asserzione dei massimi che sono:
  1) Raccolta di tutti i portieri 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;
  4) Asserzione del massimo come predicato.
- viene richiamata la funzione python chiamata "*asserisci_pesi()*" che effettua l'asserzione dinamica dei pesi per poter essere poi modificati dagli utenti nella seconda sezione.

1. Visualizzare la classifica dei giocatori di un determinato ruolo

Selezionato un ruolo l'utente potrà visualizzare la classifica di giocatori ordinata in base alla valutazione assegnatagli, mentre, 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 portieri vengono considerate le seguenti caratteristiche con i rispettivi pesi:

- Rating → 1 (voto medio con valutazione in decimi)
- AccuratePassingPercentage → 0.5 (percentuale di passaggi completati)
- RedCards → -0.7 (cartellini rossi)
- PenaltyConceded → -0.7 (rigori concessi)
- AccurateLongBallsPercentage → 0.5 (percentuale di lanci lunghi completati)
- OwnGoals → -0.5 (autogol provocati)
- ErrorLeadToGoal → -1 (errori che hanno portato al gol)
- Saves → 1 (parate effettuate)
- CleanSheet → 1 (numero di porte inviolate)
- PenaltyPercentage → 1 (percentuale di rigori parati)
- SavesInsideTheBox → 1 (parate su tiri effettuati all'interno dell'area di rigore)

- SavesOutsideTheBox → 0.9 (parate su tiri effettuati fuori area)
- GoalsInsideTheBox → -0.5 (gol concessi su tiri effettuati all'interno dell'area di rigore)
- GoalsOutsideTheBox → -0.5 (gol concessi su tiri effettuati da fuori area)
- GoalsConceded → -1 (gol concessi)

Inoltre, esistono tra le caratteristiche analizzate, alcune comuni a tutti i giocatori, indipendentemente dal ruolo: Age (età), Height (altezza), MinutesPlayed (minuti giocati) e TournamentValue (difficoltà del torneo).
Per alcuni valori è stata effettuata la normalizzazione, per portarli allo stesso intervallo di valori [0, 1], ad esempio per due variabili come CleanSheet e Saves; normalmente il numero dei CleanSheet è nettamente inferiore rispetto al numero di parate effettuate (Saves), e quindi se tali valori non fossero normalizzati si avrebbe una maggiore influenza di Saves rispetto a CleanSheet anche avendo lo stesso peso.

```
Digitare un numero da 0 a 8 per i seguenti ruoli:
1) Portieri
2) Difensori centrali
3) Terzini
4) Mediani
5) Centrali
6) Trequartisti
7) Punte
8) Ali
0) Indietro
```

Menù iniziale per questa voce

Mike Maignan	2.73
Thibaut Courtois	2.62
Ederson	2.41
Bono	2.38
Alex Remiro	2.28
Alisson	2.22
Predrag Rajkovic	2.06
Marvin Schwabe	2.05
Sergio Herrera	2.04
Gianluigi Donnarumma	2.03

< Esc 1 2 3 ... 21 22 23 Succ >

Classifica dei portieri

Il calcolo della classifica viene effettuato tramite una regola Prolog creata ad hoc *evaluate_all_gk()*, 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_gk(Gk) :-
    get_all_gk(Players),
    evaluate_all_gk(Players, Gk).
```

Predicato che prende tutti i portieri 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.

```
evaluate_all_gk([], []).
evaluate_all_gk([H | T], Res) :-
    evaluation_gk(H, Eval_local),
    evaluate_all_gk(T, Eval_down),
    name(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_age +
    Eval_height +
    (
        Eval_Rating * RatingWeight +
        AccuratePassesPercentage / 100 * AccuratePassesPercentageWeight +
        RedCards_divided * RedCardsWeight +
        PenaltyConceded_divided * PenaltyConcededWeight +
        AccurateLongBallsPercentage / 100 * AccurateLongBallsPercentageWeight +
        OwnGoals_divided * OwnGoalsWeight +
        ErrorLeadToGoal_divided * ErrorLeadToGoalWeightWeight +
        Eval_Saves * SavesWeight +
        CleanSheet_divided * CleanSheetWeight +
        Eval_penaltyPercentage * PenaltyPercentageWeight +
        Eval_InsideTheBox * InsideTheBoxWeight +
        Eval_OutsideTheBox * OutsiteTheBoxWeight +
        GoalsConcededInsideTheBox_divided * GoalsConcededInsideTheBoxWeight +
        GoalsConcededOutsideTheBox_divided * GoalsConcededOutsideTheBoxWeight +
        GoalsConceded_divided * GoalsConcededWeight
    ) * TournamentValue * Eval_mp.
```

Questa parte della regola *evaluation_gk* (che effettua la valutazione di un portiere) è responsabile del calcolo basandosi sui valori di tutte le variabili raccolte in precedenza in questa stessa regola. In questo calcolo viene prima effettuata la normalizzazione dei valori e poi vengono effettuati i prodotti in base ai pesi assegnati da noi ad ogni caratteristica in base alla sua importanza.

2. Visualizzare i giocatori suggeriti in base alle preferenze selezionate

Attraverso questa funzionalità l'utente può effettuare la ricerca dei migliori giocatori in base a delle caratteristiche selezionate da lui.

Inizialmente l'utente dovrà inserire un budget massimo che servirà per filtrare i giocatori che hanno un valore inferiore a quello specificato, e successivamente l'utente potrà selezionare il ruolo su cui effettuare la ricerca e poi le caratteristiche a cui dare maggior risalto (triplicandone il valore di default).

Ovviamente, l'utente potrà incrementare solamente il peso delle caratteristiche considerate più rilevanti relativamente al ruolo precedentemente scelto.

```
Digitare un numero da 0 a 8 per i seguenti ruoli:
1) Portieri
2) Difensori centrali
3) Terzini
4) Mediani
5) Centrali
6) Trequartisti
7) Punte
8) Ali
9) Per effettuare le query
0) Indietro
```

Selezione del ruolo su cui effettuare la modifica di una o più caratteristiche

Selezione della caratteristica di cui aumentare il peso (in questo caso vengono mostrate le feature del portiere)

```
Digitare il nome della caratteristica a cui
si vuole dare maggiore importanza:
1) Età
2) Altezza
3) Media Voto
4) Salvataggi
5) Clean Sheet
6) Rigori Parati
0) Indietro
```

```
Mike Maignan              3.31
Thibaut Courtois          3.06
Bono                      2.80
Alisson                   2.75
Ederson                   2.67
Alex Remiro               2.63
Predrag Rajkovic          2.55
Marvin Schwabe            2.39
Sergio Herrera            2.34
Gianluigi Donnarumma      2.28
< Esc 1 2 3 ... 21 22 23  Succ >
```

Classifica dei giocatori a cui è stato aumentato il peso della Media Voto.
Si può osservare che alcune posizioni sono variate dalla classifica vista al punto 1.

3. Visualizzare la previsione della classifica di uno dei TOP 5 campionati

Con questa funzionalità sarà possibile visualizzare la predizione della classifica di un campionato scelto dall'utente tra quelli supportati (top 5 campionati europei). La previsione è possibile mediante l'ordinamento dei punteggi delle varie squadre. Il punteggio viene calcolato sommando le valutazioni dei primi diciotto giocatori (titolari + riserve) ordinati sempre in base alla valutazione, calcolate tramite regola *get_team_eval_from_team_list*.

```
predict_tournaments_rank([], []).
predict_tournaments_rank([Ht | Tt], TotalRank) :-
    get_all_tournamentTeams(Ht, Teams), % Teams = list of teams for that tournament
    get_teams_evals_from_team_list(Teams, EvalsForTeams),
    sort(2, @>=, EvalsForTeams, Rank),
    predict_tournaments_rank(Tt, SubTotalRank),
    append([[Ht, Rank]], SubTotalRank, TotalRank).

get_teams_evals_from_team_list([], []).
get_teams_evals_from_team_list([H | T], EvalTeams) :-
    get_fullTeam(H, Players),
    get_all_eval_from_players_list(Players, ResEvalList),
    get_teams_evals_from_team_list(T, SubResEvalList),
    sort(ResEvalList, Sorted),
    reverse(Sorted, Reversed),
    take(Reversed, 18, First18),
    sum(First18, SumFirst18),
    append([[H, SumFirst18]], SubResEvalList, EvalTeams).
```

Nell'immagine viene mostrata la regola (o predicato) che permette di restituire, per ogni squadra, la sommatoria delle valutazioni della propria squadra.
Questa operazione viene effettuata ricorsivamente prendendo in input una lista delle squadre e restituendo in output la lista composta da liste il cui primo elemento è il nome della squadra e il secondo è il valore della sommatoria.

```
Digitiare un numero da 1 a 5 per scegliere il
campionato di cui visualizzare la previsione
della classifica:
1) Serie A Tim
2) Premier League
3) LaLiga Santander
4) Ligue 1 Ubereats
5) Bundesliga
0) Indietro
```

```
1° Milan
2° Inter
3° Juventus
4° Fiorentina
5° Lazio
6° Atalanta
7° Napoli
8° Roma
9° Sassuolo
10° Empoli
< Esc 1 2  Succ >
```

```
11° Bologna
12° Udinese
13° Salernitana
14° Spezia
15° Torino
16° Hellas Verona
17° Monza
18° Sampdoria
19° Cremonese
20° Lecce
< Prec 2 Esc >
```

La **prima immagine** mostra l'elenco dei *campionati* per il quale è possibile predire la classifica.
Le **altre due immagini** mostrano la *predizione della classifica* per il campionato di Serie A Tim.

4. 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.

```
compare_gk(Name1, Name2, Perc1, Perc2) :-
    name(Id1, Name1),
    name(Id2, Name2),
    evaluation_gk(Id1, Eval1),
    evaluation_gk(Id2, Eval2),
    PercSum is Eval1 + Eval2,
    Perc1 is Eval1 / PercSum,
    Perc2 is Eval2 / PercSum.
```

```
Scegli il ruolo da confrontare tra questi:
1) Portieri
2) Difensori centrali
3) Terzini
4) Mediani
5) Centrali
6) Trequartisti
7) Punte
8) Ali
0) Indietro
> 1
Dammi il primo giocatore: Mike Maignan
Dammi il secondo giocatore: Alex Meret
Mike Maignan 81.24 % - 18.76 % Alex Meret
```

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

5. Visualizzare i punti deboli della squadra

Mediante questa funzionalità, l'utente sarà in grado, scegliendo inizialmente una squadra, di ottenere una lista di giocatori considerati i punti deboli della squadra (il peggiore per ruolo) se presenti.
Inizialmente, viene creata una classifica decrescente dei giocatori in base alla valutazione per ogni ruolo. Successivamente si procede a esaminare ogni giocatore della squadra selezionata. Un giocatore viene considerato un punto debole se la sua valutazione risulta essere inferiore alla valutazione del giocatore che si trova ai 2/3 della classifica menzionata precedentemente.
La selezione dei punti deboli ha questi vincoli:

- Per i portieri sono stati aggiunti dei vincoli supplementari:
    - Se nella squadra è presente solo un portiere, quel portiere non viene considerato;
    - Se una squadra ha più di un portiere, viene esaminato solo quello titolare, poiché le riserve risulterebbero punti deboli nella quasi totalità dei casi, giocando molto meno rispetto alle riserve degli altri ruoli;
- Non vengono selezionati i giocatori che hanno saltato più di 21 partite per infortuni;
- Se vengono selezionati più giocatori per un ruolo, viene selezionato solo il peggiore.

```
Inserisci la squadra che vuoi analizzare: Milan
Difensore centrale: Matteo Gabbia
Terzino: Fode Ballo Toure
Mediano: Tiemoue Bakayoko
Punta: Ante Rebic
```

Inserendo come squadra "*Milan*" risultano solo quattro giocatori, ovviamente gli altri ruoli non compaiono poiché non rispettano i vincoli precedentemente citati.

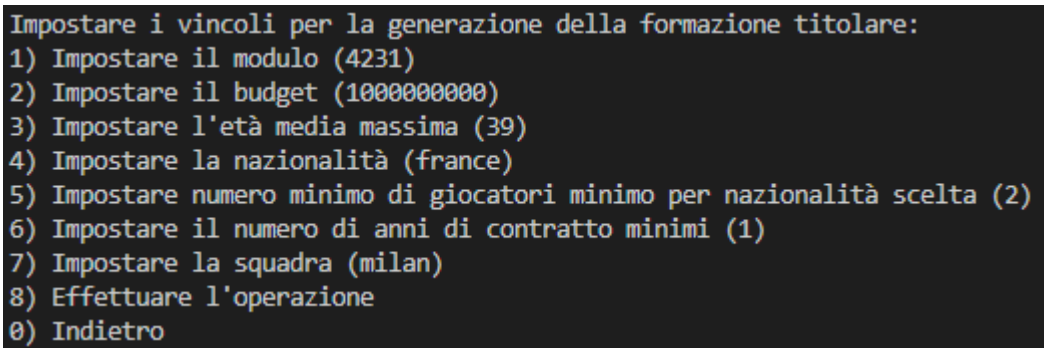
```
team_weak_players(Team, [Weakest_Gk, Weakest_Dc, Weakest_Fb, Weakest_Dm, Weakest_Mc, Weakest_Am, Weakest_St, Weakest_W]) :-
    evaluate_all_gk(Eval_gk_list),
    evaluate_all_dc(Eval_dc_list),
    evaluate_all_fb(Eval_fb_list),

    sort(2, @>=, Eval_gk_list, Eval_gk_listSorted),
    sort(2, @>=, Eval_dc_list, Eval_dc_listSorted),
    sort(2, @>=, Eval_fb_list, Eval_fb_listSorted),

    get_average_last_third(Eval_gk_listSorted, Avg_gk),
    get_average_last_third(Eval_dc_listSorted, Avg_dc),
    get_average_last_third(Eval_fb_listSorted, Avg_fb),

    get_fullTeam(Team, Players),
    filter_weak_players(Players, FilteredPlayers),

    get_gk_from_player_list(FilteredPlayers, FilteredPlayers1),
    get_player_id_eval_weakness(FilteredPlayers1, Gk_List),
    sort(2, @>=, Gk_List, Eval_Gk_listSorted),
    nth0(0, Eval_Gk_listSorted, FirstGkEval),

    get_dc_from_player_list(FilteredPlayers, FilteredPlayers2),
    get_player_id_eval_weakness(FilteredPlayers2, Dc_List),

    get_fb_from_player_list(FilteredPlayers, FilteredPlayers3),
    get_player_id_eval_weakness(FilteredPlayers3, Fb_List),

    per_role_find_lowest_eval([FirstGkEval], Avg_gk, Weakest_Gk),
    per_role_find_lowest_eval(Dc_List, Avg_dc, Weakest_Dc),
    per_role_find_lowest_eval(Fb_List, Avg_fb, Weakest_Fb),
```

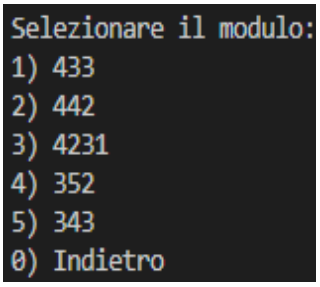
La regola principale usata per riconoscere i punti deboli della squadra (per questioni di spazio, viene mostrata la selezione solo di tre ruoli, ma il codice originale effettua la selezione su tutti i ruoli).

6. Generare formazione titolare squadra

Questa funzionalità permette all'utente di ottenere la migliore formazione possibile per una squadra (scelta in input) in base ai seguenti vincoli impostati dall'utente:

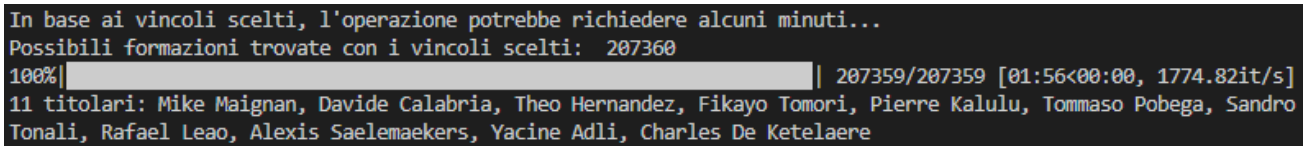
- **Modulo**: la disposizione dei giocatori in campo
- **Budget**: Il budget da non superare sommando i valori di mercato dei giocatori
- **Età media massima**: soglia di età da non superare effettuando la media dei giocatori selezionati
- **Numero minimo di giocatori per nazionalità scelta**: il numero minimo di giocatori di una determinata nazionalità scelta da includere obbligatoriamente nella formazione
- **Numero di anni di contratto minimi**: Il numero minimo di anni rimanenti alla scadenza del contratto
- **Squadra**: squadra per la quale generare la formazione





Questi sono i moduli supportati per la generazione della formazione.

Questa funzionalità consiste di CSP (*Constraint Search Problem*) risolti tramite algoritmi *Depth First Search + Backtracking*. Per la ricerca di un modello, Prolog attraversa l'intera regola, se riesce ad arrivare alla fine della regola, senza effettuare backtracking, vuol dire che ha trovato un modello (cioè un mondo che soddisfa tutti i vincoli impostati) per questo CSP. Quindi il corpo della regola corrisponde ad un percorso dell'albero di ricerca dove l'inizio della regola corrisponde al nodo radice e il nodo foglia corrisponde alla soluzione possibile, il percorso corrisponde al modello. Nel caso in cui una clausola del corpo sia falsa, il mondo parziale generato fino a quel momento non è un modello del CSP e quindi Prolog effettuerà backtracking e tenterà con altri valori del dominio delle variabili.



Dopo aver inserito i vincoli:

1) vengono calcolate tutte le possibili permutazioni in base ai vincoli impostati
2) vengono filtrati risultati dove ci possono essere dei giocatori ripetuti in più ruoli
3) per ogni permutazione filtrata, viene calcolata la sommatoria delle valutazioni dei giocatori e vengono ordinati in modo decrescente in base alle valutazioni delle permutazioni
4) dalla lista ordinata viene presa la prima formazione che è considerata la migliore in base alle valutazioni dei suoi giocatori

L'operazione è possibile tramite la regola prolog "*calcola_perm_rosa*" che prende come parametri i vari vincoli e quando viene eseguita restituisce tutte le permutazioni possibili in base ai vincoli scelti.

```
calcola_perm_rosa(
        Module, BudgetTot, MaxAvgAge, Nationality, MinNationality, MinYearContract, Team,   % input
        IdGk, IdFb1, IdFb2, IdDc1, IdDc2, IdDm1, IdMc1, IdMc2, IdW1, IdW2, IdSt1            % output
        ) :-
    Module = '433',

    %%%% GoalKeeper
    team(IdGk, Team),
    contractYearsLeft(IdGk, YearsLeftGk),
    YearsLeftGk > MinYearContract,
    is_goal_keeper(IdGk),

    %%%% Fullback
    team(IdFb1, Team),
    contractYearsLeft(IdFb1, YearsLeftFb1),
    YearsLeftFb1 > MinYearContract,
    is_fullback(IdFb1),

    team(IdFb2, Team),
    contractYearsLeft(IdFb2, YearsLeftFb2),
    YearsLeftFb2 > MinYearContract,
    is_fullback(IdFb2),

    % assicuro che non siano lo stesso
    not(IdFb1 = IdFb2),
```

Sono state create diverse implementazioni delle regole in base al "Modulo" (*Module*) inserito in input, in questo caso "433".
In questa parte iniziale, vengono scelti i vari giocatori in base alla squadra, il ruolo e gli anni di contratto rimanenti (vincoli).

```
marketValue(IdGk, ValueGk),
marketValue(IdFb1, ValueFb1),
marketValue(IdFb2, ValueFb2),
...

ValueTot is ValueGk + ValueFb1 + ValueFb2,
ValueTot =< BudgetTot,

age(IdGk, AgeGk),
age(IdFb1, AgeFb1),
age(IdFb2, AgeFb2),
...

AgeAvg is ((AgeGk + AgeFb1 + AgeFb2) / 11),
AgeAvg =< MaxAvgAge,

country(IdGk, CountryGk),
country(IdFb1, CountryFb1),
country(IdFb2, CountryFb2),
...

count_nazionalita([CountryGk, CountryFb1, CountryFb2], Nationality, TotaleNazionalita),
MinNationality =< TotaleNazionalita.
```

Nella fase finale della regola vengono effettuate queste operazioni:

- recupero dei **market value** (valori di mercato) per tutti i giocatori
- controllo che la sommatoria dei market value sia inferiore al vincolo del **Budget** impostato
- recupero delle **età** per tutti i giocatori
- controllo che la media delle età sia inferiore alla **massima età media** impostata

- recupero delle **nazionalità** per tutti i giocatori
- conteggio dei giocatori che hanno la nazionalità impostata in input
- controllo che il numero di giocatori della nazionalità scelta sia maggiore o uguale al numero impostato

6. Predire il risultato di una partita

Questa parte dell'applicazione è stata gestita tramite degli algoritmi di Machine Learning che verranno spiegati più approfonditamente nel sesto capitolo "**ANALISI DEI RISULTATI: PREDIZIONE DELLE PARTITE**".

In questo caso è stato scelto come algoritmo per effettuare le predizioni "Random Forest Classifier", il quale, nel nostro caso, risulta il migliore in termini di accuratezza.

```
Sto allenando il modello per la predizione...
Dammi la prima squadra: Milan
Dammi la seconda squadra: Juventus
Risultato: vittoria Juventus
```

Inizialmente verrà effettuato il training della Random Forest, terminato l'apprendimento, verrà richiesto all'utente di inserire i nomi delle squadre coinvolte nella partita, ed infine, verrà eseguito il modello per ottenere in output la predizione del risultato. In questo caso viene richiesto di predire un'ipotetica partita tra Milan e Juventus, e risulta che abbia vinto la Juventus.

6. PREDIZIONI DEI RISULTATI DELLE PARTITE

L'obiettivo di questa parte del progetto è di **predire**, il più accuratamente possibile, **il risultato di ipotetiche partite tra due squadre**. In input verranno date le statistiche delle squadre e in output si avranno tre classi: vittoria, pareggio e sconfitta, dal punto di vista della squadra in casa.
Il dataset è stato creato effettuando un join (prodotto cartesiano) tra i due file *.csv*: ***matches.csv*** e ***team_statistics.csv***.

Il primo contiene lo storico delle partite, e, per ogni partita, il nome delle squadre e il numero di gol per ogni squadra.
Il secondo contiene le statistiche come, ad esempio, gol effettuati e gol subiti per ogni squadra.

```
df = pd.read_csv('./../datasets/dataset_for_ML.csv')
X = pd.get_dummies(df.drop(['result'], axis=1))
y = df['result'].apply(lambda x: 2 if x == 'victory' else 1 if x == 'draw' else 0)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.1)
```

La predizione è possibile mediante l'utilizzo di algoritmi di Machine Learning addestrati sul dataset e la qualità dei modelli di predizione creati è stabilita in base all'**accuratezza** delle predizioni degli stessi, quest'ultima è la percentuale di esempi predetti correttamente sul totale degli esempi predetti.

Gli algoritmi adoperati sono:

1. *Random Forest Classifier*
2. *Naive Bayes*
3. *K-Nearest Neighbours*
4. *Support Vector Machine*
5. *Neural Network*

Per ogni algoritmo di Machine Learning (escluso Naive Bayes) è stata effettuata **una ricerca dei migliori valori,** per quanto riguarda gli iper-parametri, e suddividendo il dataset in training set e test set tramite una **cross validation** impostando il numero di partizioni a 10 (90% del dataset per training set e 10% per il test set). Questa operazione di ricerca è effettuata tramite la classe GridSearchCV della libreria sklearn che, dopo aver effettuato il training e i vari test, crea un report per ogni classe dell'accuratezza raggiunta.
Dopo aver effettuato una ricerca dei valori corretti per ogni parametro, si effettua la ricerca delle migliori k features tramite la classe SelectKBest che, dati in input il numero di features da trovare (*k*) e la funzione per ricercare le migliori features (*chi2, f_regression, f_classif, mutual_info_classif, mutual_info_regression*), restituirà in output le migliori k features e il dataframe proiettato sulle features.

1. Random Forest Classifier

a. Definizione di Random Forest

La random forest è un tipo di algoritmo di machine learning supervisionato che consiste nell'addestrare un insieme di alberi di decisione. Quando sarà necessario effettuare predizioni, verranno eseguite predizioni con tutti gli alberi e verranno prese le classi predette più presenti.
Un albero di decisione è un algoritmo di Machine Learning che dati in input un insieme di esempi crea un albero binario con il quale per ogni nodo viene cercata una feature che suddivide nel miglior modo possibile gli esempi in gruppi distinti.

b. Implementazione e studio di iper-parametri

L'implementazione dell'algoritmo è stata effettuata tramite la classe RandomForestClassifier della libreria sklearn che permette di creare delle random forest specificando i valori degli iper-parametri per avere l'accuratezza migliore.
Gli iper-parametri modificati nei vari test sono:

- *bootstrap*: che viene usato per forzare l'uso di tutto il dataset per l'addestramento di ogni albero
- *max_depth:* usato per definire una profondità massima ed evitare overfitting dell'albero
- *max_features*: il numero massimo di features da considerare quando viene cercato il miglior punto di separazione
- *min_samples_leaf*: il minimo numero di esempi richiesti per essere considerato un nodo foglia
- *min_samples_split*: il minimo numero di esempi richiesti per separare un nodo interno in nodi figli
- *n_estimators*: il numero di alberi nella foresta

I valori scelti per gli iper-parametri da essere testati sono:

- bootstrap: True
- max_depth: 80, 90, 100, 110
- max_features: 2, 3
- min_samples_leaf: 3, 4, 5
- min_samples_split: 8, 10, 12
- n_estimators: 100, 200, 300, 1000

```
# Create the parameter grid based on the results of random search
param_grid = {
    'bootstrap': [True],
    'max_depth': [80, 90, 100, 110],
    'max_features': [2, 3],
    'min_samples_leaf': [3, 4, 5],
    'min_samples_split': [8, 10, 12],
    'n_estimators': [100, 200, 300, 1000]
}

# Create a based model
rf_base = RandomForestClassifier()
# Instantiate the grid search model
grid_search = GridSearchCV(estimator=rf_base, param_grid=param_grid, cv=10, n_jobs=-1, verbose=10)
# Fit the grid search to the data
best_model = grid_search.fit(X_train, y_train)
# print best parameter after tuning
print("best params: ", grid_search.best_params_)
# print how our model looks after hyper-parameter tuning
print("best estimator: ", grid_search.best_estimator_)
grid_predictions = grid_search.predict(X_test)
print(classification_report(y_test, grid_predictions))
```

c. Analisi dei risultati

Dopo aver eseguito la ricerca, i migliori valori per gli iper-parametri sono:

- *bootstrap*: True
- *max_depth*: 80
- *max_features*: 2
- *min_samples_leaf*: 5
- *min_samples_split*: 12
- *n_estimators*: 100

La selezione delle migliori k features ha dato questi risultati (per ogni funzione viene selezionata l'accuratezza migliore e il numero di features):

Funzione di selezione	Accuratezza	K-Best Features
chi2	53.1	175
f_regression	52.8	168
f_classif	52.6	112
mutual_info_classif	52.6	47
mutual_info_regression	53.8	87

Quindi la funzione che seleziona le migliori features risulta essere *mutual_info_regression* con 87 features.

2. Naive Bayes

a. Definizione di Naive Bayes

Naive Bayes è un algoritmo di Machine Learning supervisionato che classifica degli esempi sfruttando la probabilità a priori di ogni feature di avere un particolare valore (di solito basata sul numero di esempi che hanno quel valore per quella feature sul totale degli esempi). Dopo aver calcolato le probabilità di ogni valore di feature per ogni classe, si effettuano le predizioni sfruttando il teorema della probabilità condizionata di Bayes.

b. Implementazione e studio

L'implementazione dell'algoritmo è stata effettuata tramite le classi *MultinomialNB*, *ComplementNB*, *BernoulliNB*, *CategoricalNB* della libreria sklearn che permette di creare dei classificatori Bayesiani.

- **MultinomialNB**: è utile per la classificazione di features discrete ma può essere utilizzato anche per valori frazionari
- **ComplementNB**: è stato progettato per correggere le "assunzioni severe" (severe assumptions) create dallo standard MultinomialNB, molto utile per dataset sbilanciati
- **BernoulliNB**: è simile al MultinomialNB.
- **CategoricalNB**: è utile per la classificazione di features discrete che sono distribuite categoricamente. Le categorie di ogni feature sono inferite da una distribuzione categorica

Inoltre, è stata fatta la selezione delle migliori k-features.

```
function_to_test = [ chi2, f_regression, f_classif, mutual_info_classif, mutual_info_regression ]
NB_to_test = [ MultinomialNB, ComplementNB, BernoulliNB, CategoricalNB ]

df = pd.read_csv('./../datasets/dataset_for_NN.csv')
X = pd.get_dummies(df.drop(['result'], axis=1))
y = df['result'].apply(lambda x: 2 if x == 'victory' else 1 if x == 'draw' else 0)

with open('outNB', 'w') as f:
    for nb in NB_to_test:
        cv = KFold(n_splits=10, shuffle=True, random_state=1)
        print(nb.__name__)
        for func in function_to_test:
            print(func.__name__)
            for k in tqdm.tqdm(range(5, 199)):
                # normalization
                X = (X - np.min(X, axis=0)) / (np.max(X, axis=0) - np.min(X, axis=0))

                X_new = SelectKBest(func, k=k).fit_transform(X, y)
                gnb = nb()
                scores = cross_val_score(gnb, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
                f.write(f'{mean(scores)}\n')
```

c. Analisi dei risultati

Dopo aver eseguito la ricerca delle migliori k features per ogni implementazione dell'algoritmo di Bayes, otteniamo questi risultati:

NB usato	Funzione	Accuratezza media
MultinomialNB	*chi2*	0,45495
	f_regression	0,45471
	f_classif	0,45471
	mutual_info_classif	0,45471
	mutual_info_regression	0,45471
ComplementNB	*chi2*	0,50164
	f_regression	0,50163
	f_classif	0,50163
	mutual_info_classif	0,50163
	mutual_info_regression	0,50163
BernoulliNB	*chi2*	0,50335
	f_regression	**0,50363**
	f_classif	**0,50363**
	mutual_info_classif	**0,50363**
	mutual_info_regression	**0,50363**
CategoricalNB	*chi2*	0,44925
	f_regression	0,44923
	f_classif	0,44923
	mutual_info_classif	0,44923
	mutual_info_regression	0,44923

Quindi il miglior algoritmo è BernoulliNB con la funzione f_regression o f_classif, l'utilizzo delle ultime due richiede più tempo rispetto alle prime tre, quindi ne sconsigliamo l'utilizzo avendo l'accuratezza risultante invariata.

3. K-Nearest Neighbours

a. Definizione di K-Nearest Neighbours

Il K-Nearest Neighbours (o KNN) è un algoritmo di Machine Learning di classificazione, la cui fase di apprendimento consiste nel prendere il dataset di training (o training set), convertirlo in uno spazio vettoriale, cioè convertendo i vari esempi in dei vettori di cui ogni feature corrisponde ad una dimensione, e i cui valori corrispondono alle proprie coordinate. La fase di predizione consiste nel prendere l'esempio da classificare, convertirlo in un vettore e trovando i *k* vettori più vicini (secondo una misura di distanza da specificare). Da questi k viene scelta la classe più prevalente per classificare l'esempio.

b. Implementazione e studio

L'implementazione dell'algoritmo è avvenuta tramite la classe KNeighboursClassifier della libreria sklearn. Il training è stato effettuato tramite cross-validation e successiva selezione delle k migliori feature tramite SelectKBest. Per la gestione degli iperparametri abbiamo selezionato i seguenti con il corrispettivo range di valori:

- **leaf_size:** [1,3,…,50] questo parametro influisce sulla velocità della costruzione e delle query, così come sulla memoria richiesta per memorizzare l'albero
- **n_neighbors:** [1,3,…,30] il numero di vicini per le query
- **p**: [1,2] il parametro di potenza per la metrica Minkowski. Quando p = 1 è equivalente alla distanza di Manhattan e quella Euclidea quando p = 2.

```
hyperparameters = {
    'leaf_size': list(range(1,50, 2)),
    'n_neighbors': list(range(1,30, 2)),
    'p': [1,2],
}
#Create new KNN object
knn = KNeighborsClassifier()
#Use GridSearch
clf = GridSearchCV(knn, hyperparameters, cv=10, verbose=10)

# fitting the model for grid search
best_model = clf.fit(X_train, y_train)

# print best parameter after tuning
print("best params: ", clf.best_params_)

# print how our model looks after hyper-parameter tuning
print("best estimator: ", clf.best_estimator_)

grid_predictions = clf.predict(X_test)
#Fit the model
# report performance
print(classification_report(y_test, grid_predictions))
```

```
function_to_test = [ chi2, f_regression, f_classif, mutual_info_classif, mutual_info_regression ]
hyperparams = clf.best_params_
results = []
for f in function_to_test:
    print(f.__name__)
    results.append([])
    for i in tqdm(range(5, 199)):
        selector = SelectKBest(f, k=i)
        X_new = selector.fit_transform(X_train, y_train)

        # get the column name
        cols = selector.get_support(indices=True)

        # Get columns to keep and create new dataframe with those only for the train example
        X_new = X_train.iloc[:,cols]

        rf_final = KNeighborsClassifier(**hyperparams)
        rf_final.fit(X_new, y_train)

        # Get columns to keep and create new dataframe with those only for the test example
        X_test_new = X_test.iloc[:,cols]

        grid_predictions = rf_final.predict(X_test_new)

        # report performance
        results[-1].append(accuracy_score(y_test, grid_predictions))
```

c. Analisi dei risultati

La grid search ci ha permesso di trovare come migliore configurazione:

- leaf_size: 1
- n_neighbours: 29
- p: 2

Dopo aver eseguito la ricerca delle migliori k features per ogni implementazione dell'algoritmo kNN, otteniamo questi risultati:

Function	Best K-Features	Accuracy
chi2	19	0.4663
f_regression	7	0.4812
f_classif	**5**	**0.4837**
mutual_info_classif	31	0.4688
mutual_info_regression	55	0.4813

I risultati migliori sono stati ottenuti tramite la funzione *f_classif* con cinque features.

4. Support Vector Machine

a. Definizione di Support Vector Machine

SVM è un algoritmo di ML supervisionato usato per la classificazione di esempi. Una SVM, durante l'apprendimento, trasforma gli esempi in uno spazio vettoriale e cerca di costruire degli iperpiani multidimensionali (con almeno una dimensione in più rispetto alla dimensione dei vettori esempi) tali da creare un Support Vector Classifier che suddivide lo spazio vettoriale in sottogruppi di esempi appartenenti alla stessa classe.

b. Implementazione e studio

L'implementazione è stata effettuata utilizzando la libreria SVM di sklearn, il training è stato effettuato tramite cross-validation e successiva selezione delle k migliori feature tramite SelectKBest. Per la gestione degli iperparametri abbiamo selezionato i seguenti con il corrispettivo range di valori:

- **c:** [0.1, 1, 10, 100, 1000], parametro di regolarizzazione
- **gamma:** [1, 0.1, 0.01, 0.001, 0.0001], coefficiente del kernel per *rbf, poly e sigmoide*
- **kernel:** *rbf,* specifica il tipo di kernel usato nell'algoritmo
- **decision_function_shape:** *ovr* (One versus rest) restituisce una funzione di decisione di dimensioni (n_samples, n_classes)

```
param_grid = {'C': [0.1, 1, 10, 100, 1000],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
              'kernel': ['rbf'],
              'decision_function_shape': ['ovr']}

grid = GridSearchCV(svm.SVC(), param_grid, refit = True, verbose = 3, cv=10)
# fitting the model for grid search
best_model = grid.fit(X_train, y_train)

# print best parameter after tuning
print(grid.best_params_)

# print how our model looks after hyper-parameter tuning
print(grid.best_estimator_)

grid_predictions = grid.predict(X_test)
# print classification report
print(classification_report(y_test, grid_predictions))
```

```
best_params = {'C': 0.1, 'decision_function_shape': 'ovr', 'gamma': 0.0001, 'kernel': 'rbf'}
cv = KFold(n_splits=10, shuffle=True, random_state=1)
function_to_test = [ chi2, f_regression, f_classif, mutual_info_classif, mutual_info_regression ]

results = []
for f in function_to_test:
    print(f.__name__)
    results.append([])
    for k in tqdm.tqdm(range(5, 199)):
        X_new = SelectKBest(f, k=k).fit_transform(X, y)
        model = svm.SVC(**best_params)
        scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
        results[-1].append(mean(scores))
```

c. Analisi dei risultati

La grid search ci ha permesso di trovare come migliore configurazione:

- **c**: 0.1
- **decision_function_shape**: ovr
- **gamma**: 0.0001
- **kernel**: rbf

Dopo aver eseguito la ricerca delle migliori k features per ogni implementazione dell'algoritmo SVM, otteniamo questi risultati:

Function	Best K-Features	Accuracy
chi2	5	0.4639
f_regression	5	0.4639
f_classif	5	0.4639
mutual_info_classif	5	0.4639
mutual_info_regression	5	0.4639

Si può notare che l'accuratezza rimane invariata indipendentemente dalla funzione usata o dal numero delle feature selezionate.

5. Neural Network

a. Definizione di Neural Network

Una Rete Neurale è un algoritmo di Machine Learning per la classificazione di esempi. La sua implementazione consiste in un grafo suddiviso virtualmente in più layers i quali nodi tra un livello e l'altro sono completamente connessi. È composto da tre parti principali che sono:

- *input layer*: i quali nodi corrispondono alle feature di input
- *hidden layers*: livelli dove dovrebbero essere riconosciuti i pattern degli esempi
- *output layer*: i quali nodi corrispondono alle classi di output

L'apprendimento consiste nel dare degli esempi in input che modificheranno i pesi in modo da adattarsi ai pattern dei dati, mentre la fase di predizione consiste nell'immettere un esempio nella rete e osservare il nodo di output con il valore di confidenza più alto.

b. Implementazione e studio

Per questa rete neurale è stato scelto di utilizzare solamente un hidden layer la cui grandezza è stata scelta in base ad una GridSearch. L'implementazione è stata effettuata utilizzando la classe KerasClassifier della libreria scikeras e le classi Sequential e Dense della libreria tensorflow, il training è stato effettuato tramite cross-validation e successiva selezione delle *k* migliori feature tramite SelectKBest. Per la gestione degli iperparametri abbiamo selezionato i seguenti con il corrispettivo range di valori:

- *batch_size*: [40, 80], il numero di esempi per batch di computazione
- *epochs*: [100, 200], il numero di epoche per allenare il modello. Un'epoca è l'iterazione sugli interi x e y di dati forniti.
- *optimizer__learning_rate*: [0.001, 0.01, 0.15], ratio di apprendimento dell'ottimizzatore della rete neurale
- *model__neurons*: [50, 100, 150, 200], il numero di neuroni del layer nascosto

```python
def create_model(neurons):
    # create model
    model = Sequential()
    model.add(Dense(neurons, input_dim=17, activation='relu', kernel_initializer='he_uniform'))
    model.add(Dense(3, activation='softmax'))
    # Compile model
    model.compile(loss="categorical_crossentropy", optimizer= "adam", metrics=['accuracy'])
    return model

# fix random seed for reproducibility
seed = 7
tf.random.set_seed(seed)
model = KerasClassifier(model=create_model, verbose=0)
hyperparams = {
    'batch_size': [40, 80],
    'epochs': [100, 200],
    'optimizer__learning_rate': [0.001, 0.01, 0.15],
    'optimizer__momentum': [0.0, 0.4, 0.8],
    'model__neurons': [50, 100, 150, 200],
}
print('starting grid search')
grid = GridSearchCV(estimator=model, param_grid=hyperparams, n_jobs=-1, cv=10, verbose=3)
selector = SelectKBest(mutual_info_classif, k=55)
X_new = selector.fit_transform(X_train, y_train)
```

```
cols = selector.get_support(indices=True)

# Get columns to keep and create new dataframe with those only for the test example
X_new_train = X_train.iloc[:,cols]
X_new_test = X_test.iloc[:,cols]

y_test_cat = to_categorical(y_test)
y_train_cat = to_categorical(y_train)

grid_result = grid.fit(X_new, y_train, validation_data=(X_new_test, y_test_cat))

# summarize results

print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
with open('outNN_k.txt', 'w') as f_out:
    for func in function_to_test:
        print(func)
        for k in tqdm.tqdm(range(5, 199, 2)):
            selector = SelectKBest(func, k=k)
            X_new = selector.fit_transform(X_train, y_train)

            # get the column name
            cols = selector.get_support(indices=True)

            # Get columns to keep and create new dataframe with those only for the train example
            X_new = X_train.iloc[:,cols]
            X_test_modified = X_test.iloc[:,cols]

            y_train_modified = to_categorical(y_train)
            y_test_modified = to_categorical(y_test)

            # create model
            model = Sequential()
            model.add(Dense(50, input_dim=k, activation='relu', kernel_initializer='he_uniform'))
            model.add(Dense(3, activation='softmax'))
            adam = Adam(learning_rate=0.001)
            # Compile model
            model.compile(loss="categorical_crossentropy", optimizer=adam, metrics=['accuracy'])

            model.fit(X_new, y_train_modified, epochs=200, batch_size=80, verbose=0)
            results.append(model.evaluate(X_test_modified, y_test_modified, verbose=0))
```

c. Analisi dei risultati

Dopo aver effettuato la selezione delle migliori k-features per ogni funzione di selezione, si è avuta la massima accuratezza con questi parametri:

Function	Best K-Features	Accuratezza
chi2	165	0.5212
f_regression	155	0.5287
f_classif	83	0.5212
mutual_info_classif	**55**	**0.5362**
mutual_info_regression	169	0.5337

Quindi la miglior funzione è *mutual_info_classif* con *55* best features
Successivamente si è effettuata la GridSearch degli iperparametri che sono risultati:

- batch_size: 80,
- epochs: 200,
- model__neurons: 50,
- optimizer__learning_rate: 0.01,
- optimizer__momentum: 0.8

6. **Bayesian Network**

a. Definizione di Bayesian Network

Una rete Bayesiana è un algoritmo di intelligenza artificiale che sfrutta il teorema di Bayes della probabilità condizionata per predire la classe di appartenenza di un esempio. A differenza del Naive Bayes, dove le probabilità sono considerate indipendenti le une dalle altre, nella rete Bayesiana si costruisce un grafo orientato per indicare le dipendenze delle probabilità. Inoltre, la rete Bayesiana permette di effettuare delle predizioni di esempi dove non tutti i valori delle features sono disponibili, e questi verranno invece calcolati tramite le influenze dei nodi genitori e della probabilità calcolata a priori.

b. Implementazione e studio

L'implementazione è stata possibile tramite la classe BayesianNetwork della libreria *pgmpy*. Per la predizione è stata effettuata una selezione delle feature più importanti e, per ogni classe, è stata calcolata la probabilità a priori come percentuale sul totale di ogni feature. Le features scelte per essere utilizzate dalla rete Bayesiana sono:

- *home_goalsConceded (goal subiti)*
- *home_goalsScored (goal segnati)*
- *home_bigChances (occasioni pericolose)*
- *home_shots (tiri effettuati)*
- *home_successfulDribbles (dribbling riusciti)*
- *home_accurateOppositionHalfPasses (passaggi riusciti nella metà campo avversaria)*
- *home_accurateCrosses (cross riusciti)*
- *home_redCards (cartellini rossi)*
- *home_possessionLost (possesso perso)*
- *home_errorsLeadingToShot (errori che hanno portato al tiro)*
- *away_goalsConceded*
- *away_goalsScored*
- *away_bigChances*
- *away_shots*
- *away_successfulDribbles*
- *away_accurateOppositionHalfPasses*
- *away_accurateCrosses*
- *away_redCards*
- *away_possessionLost*
- *away_errorsLeadingToShot*

I dati, essendo continui, sono stati discretizzati secondo un algoritmo di tipo equal_width.
Le dipendenze delle varie features sono state gestite come mostrato nel diagramma seguente.

c. Analisi dei risultati

Dopo aver eseguito la rete Bayesiana con alcune delle features interessate, l'accuracy è stata del *45.45 %.*

7. ANALISI DEI RISULTATI: PREDIZIONE DELLE PARTITE

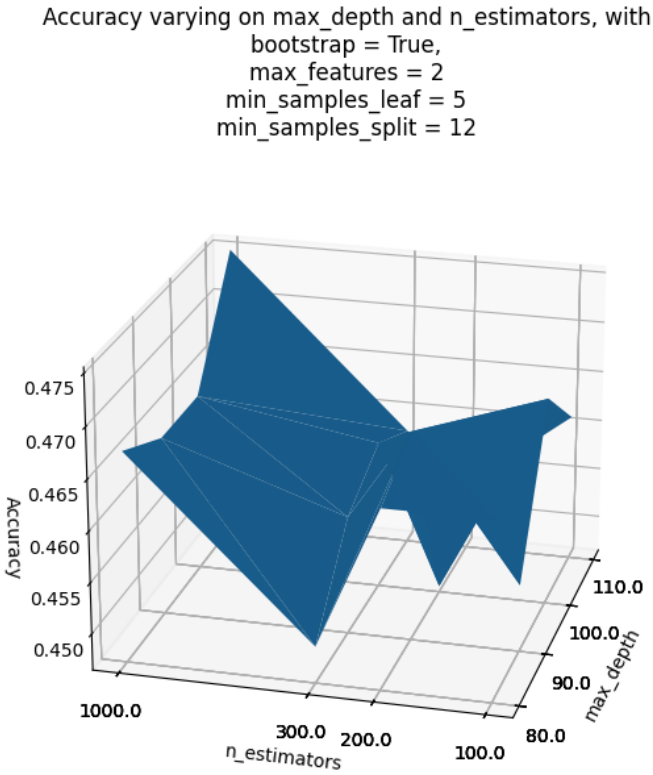
Dopo aver provato con diversi algoritmi di Machine Learning per la predizione dei risultati delle partite, questi sono gli algoritmi che hanno raggiunto l'accuratezza più alta, con le migliori impostazioni per iper-parametri e la migliore selezione delle k features per ognuno:

Algoritmo ML	Iper-parametri	Funzione e k best features	Accuracy
Random Forest	***bootstrap: True*** ***max_depth: 80*** ***max_features: 2*** ***min_samples_leaf: 5*** ***min_samples_split: 12*** ***n_estimators: 100***	***mutual_info_regression: 87***	***53,8 %***
Neural Network	batch_size: 80 epochs: 200 model__neurons: 50 optimizer__learning_rate: 0.01 optimizer__momentum: 0.8	mutual_info_classif: 55	53,62 %
Naïve Bayes	BernoulliNB	chi2: *Accuratezza non dipendente dal numero delle features selezionate*	50,16 %

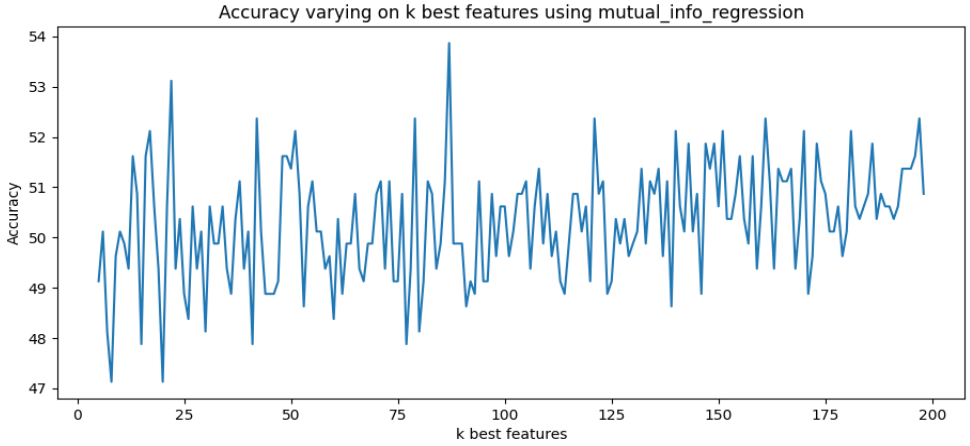
kNN	leaf_size: 1 n_neighbours: 29 p: 2	f_classif: 5	48,37 %
SVM	c: 0.1 decision_function_shape: ovr gamma: 0.0001 kernel: rbf	*Accuratezza non dipendente dalla selezione delle features*	46,39 %

Mostrando i grafici per ogni algoritmo si può osservare l'andamento dell'accuratezza al variare degli iper-parametri e delle selezioni delle best k features:

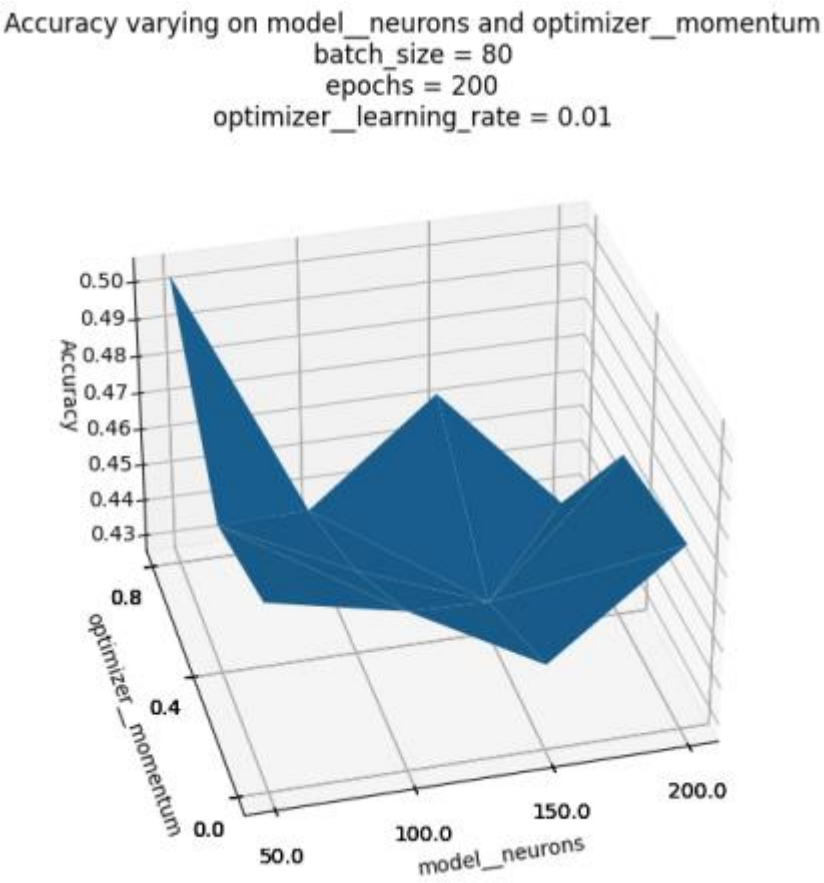
Random Forest Grid search



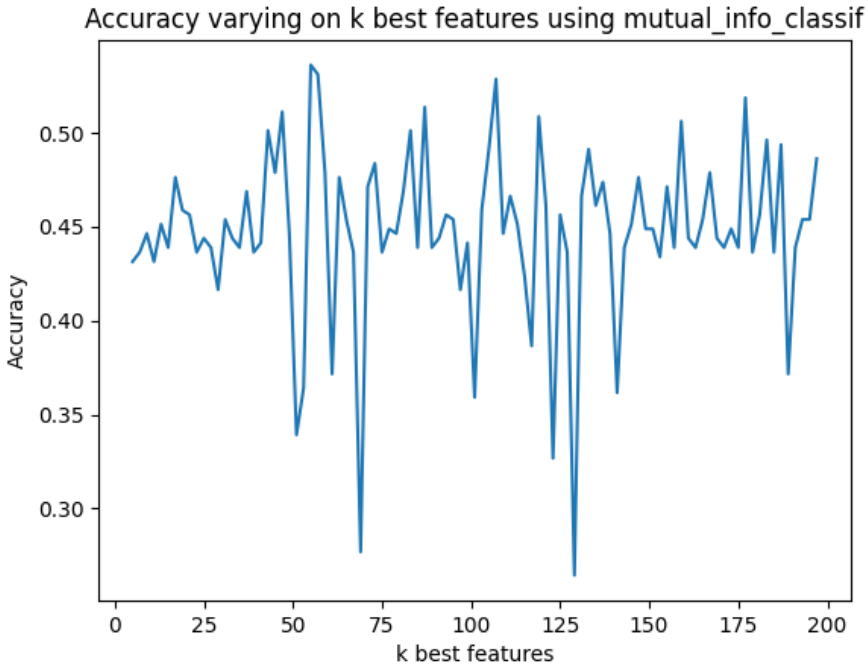
Random Forest KBest Selection



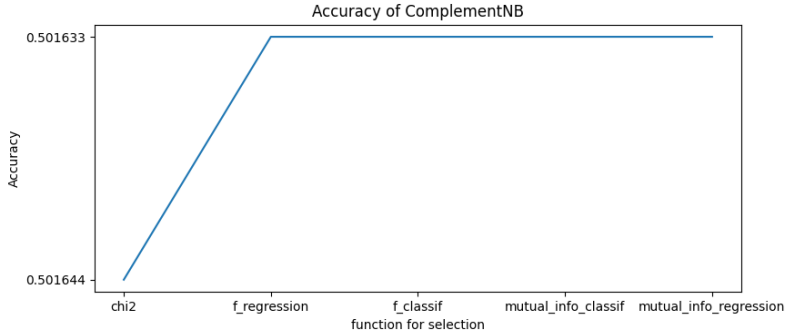
Neural Network Grid search



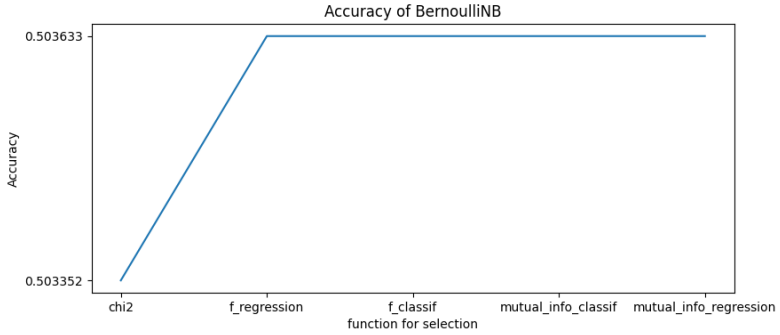
Neural Network KBest Selection



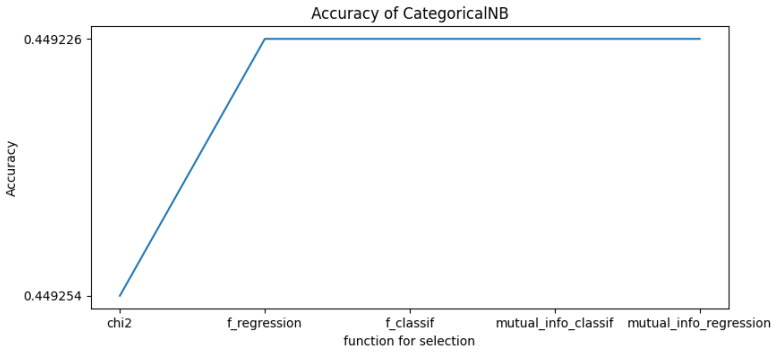
Naïve Bayes ComplementNB (Best)



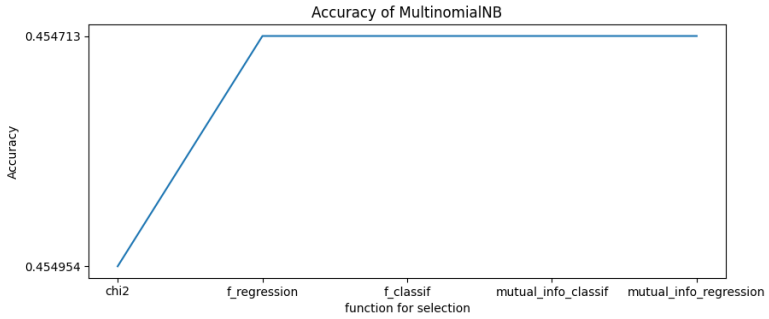
Naïve Bayes BernoulliNB



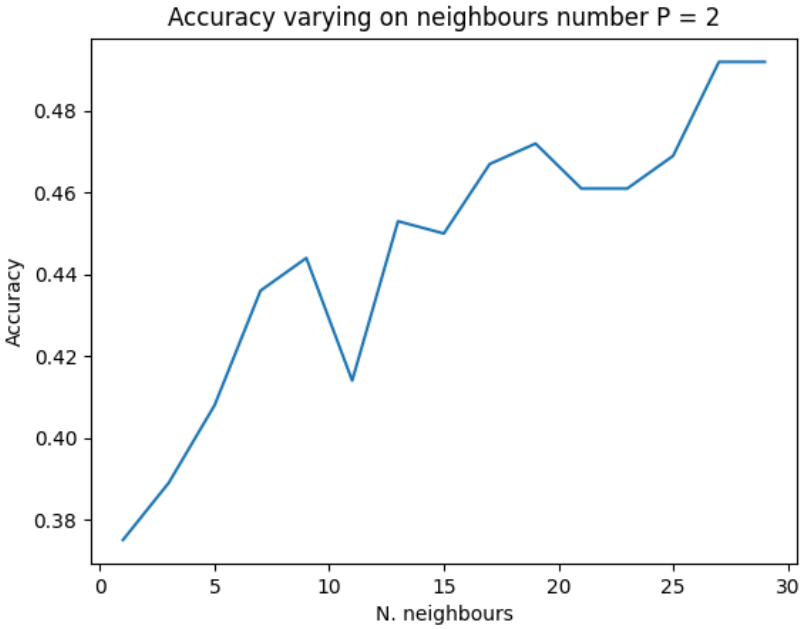
Naïve Bayes CategoricalNB



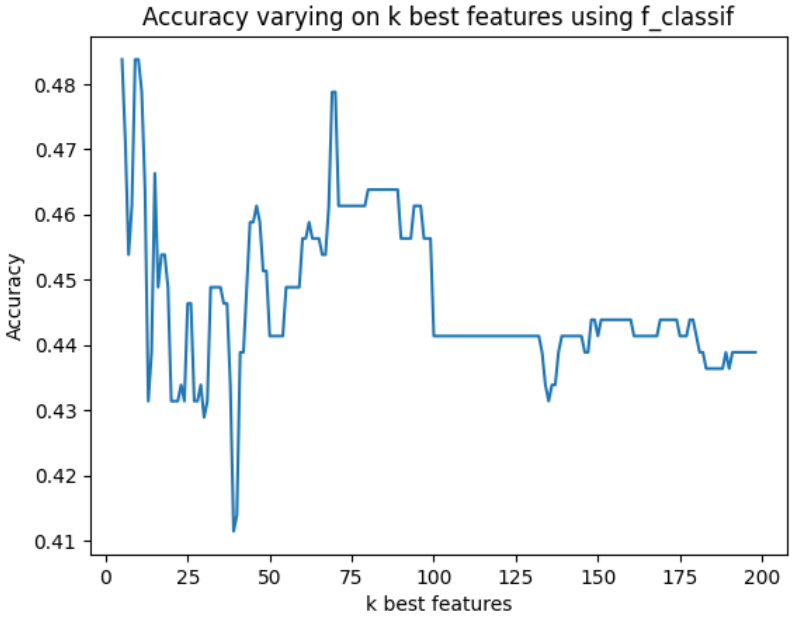
Naïve Bayes MultinomialNB



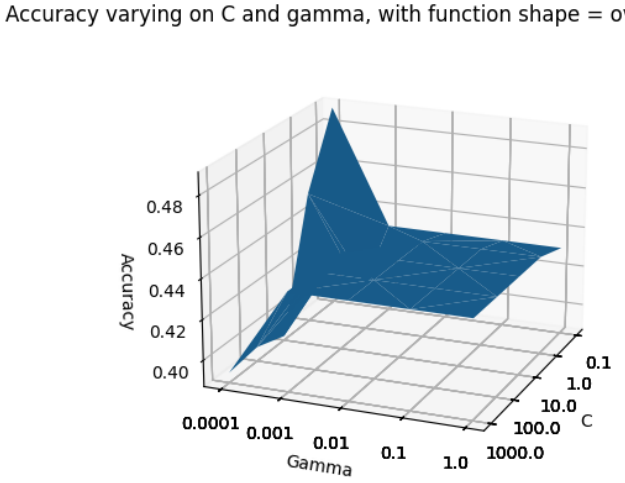
K-Nearest Neighbours Grid search



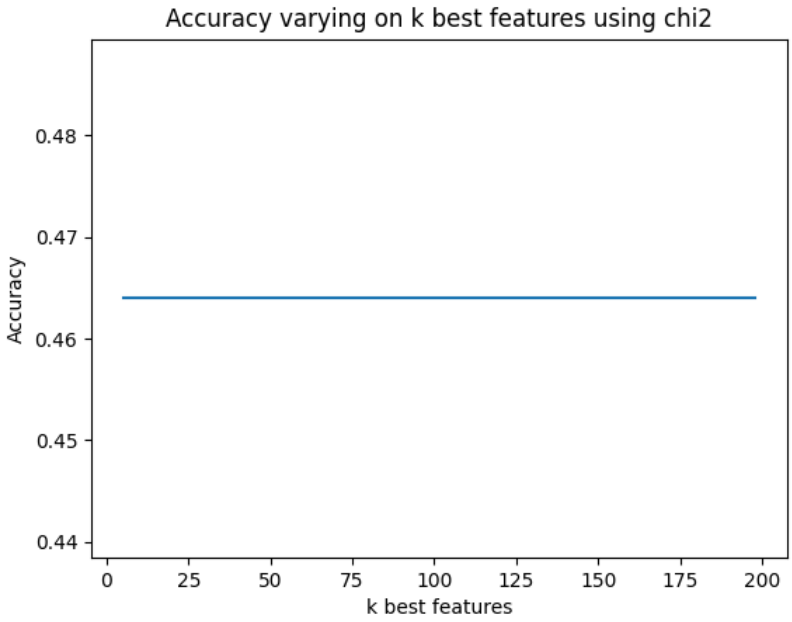
K-Nearest Neighbours KBest Selection



SVM Grid search

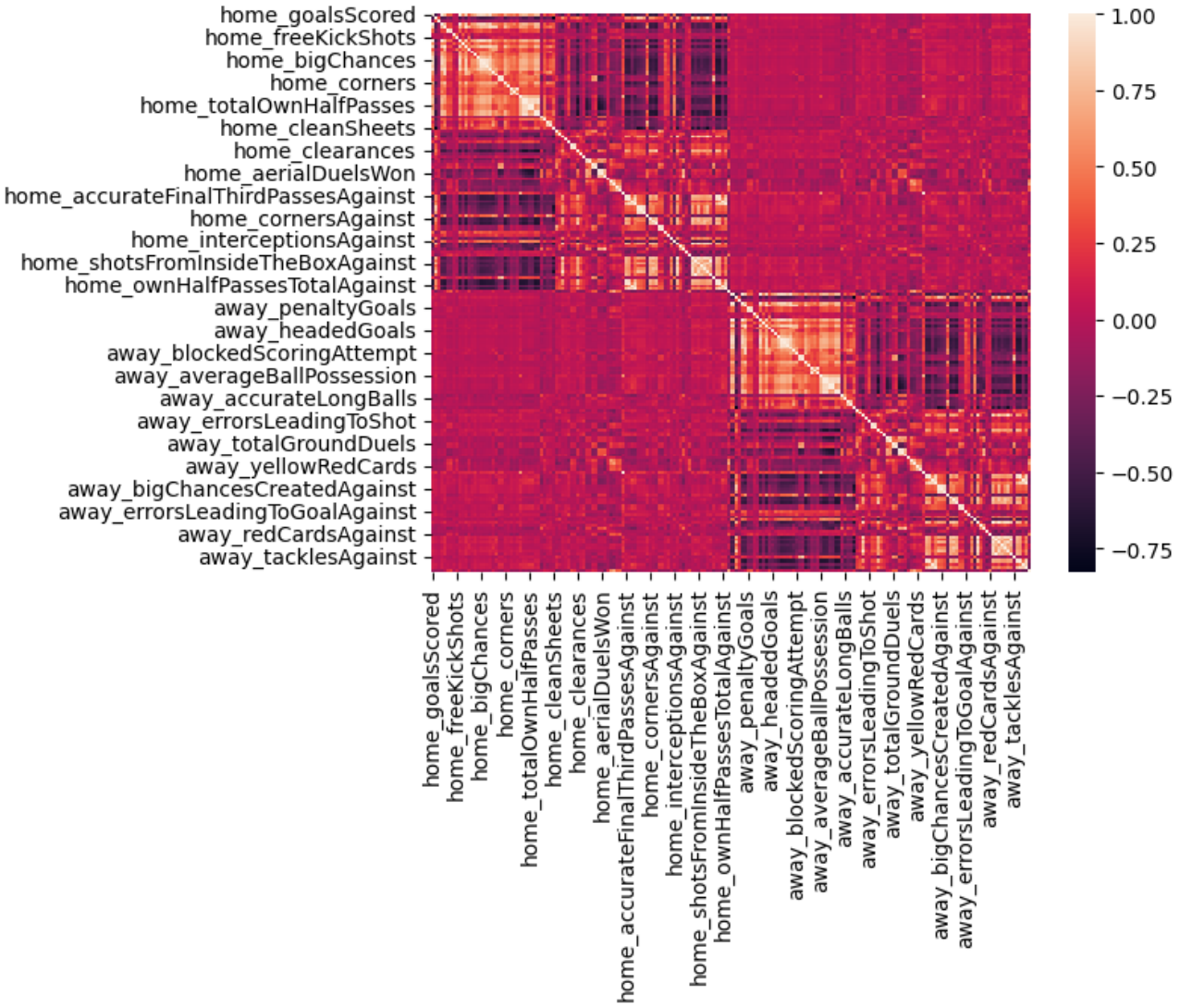


SVM KBest Selection



Quindi l'algoritmo con accuratezza più alta è Random Forest con 53,8 % di accuratezza media.

L'accuratezza "bassa" può essere causata dal dataset. Quindi è stata creata la matrice di correlazione per analizzare il dataset.



Osservando la matrice di correlazione si può notare che non vi è un'altissima correlazione tra le varie features e quindi questo giustifica l'accuratezza non troppo alta riscontrata dai vari algoritmi utilizzati (ML e reti Bayesiane).

8. CONCLUSIONI

Per quanto riguarda il sistema esperto, i risultati ottenuti sono ritenuti più che soddisfacenti e molto vicini alla realtà, nonostante le difficoltà riscontrate nella creazione della knowledge base, soprattutto nell'assegnare i pesi corretti ed equilibrati tramite la normalizzazione dei valori delle features. Inoltre, si è riusciti a rendere gli algoritmi abbastanza veloci considerando la grandezza della knowledge base, anche se per la generazione delle formazioni titolari il sistema può risultare lento viste le possibili combinazioni che possono essere anche di diversi milioni.

Nella parte del progetto riguardante l'apprendimento automatico supervisionato, l'accuratezza massima raggiunta nonostante siano stati usati svariati algoritmi ed è stato effettuato uno studio sul tuning degli iper-parametri e selezione delle migliori features, è stata solo del 53,8 %, il che dimostra che la predizione dei risultati delle partite calcistiche è ancora un task molto complesso e che non offre garanzie sui risultati.

In questo progetto, nonostante sapessimo della difficoltà di questo tipo di task, ci siamo comunque voluti cimentare in questo argomento spinoso per dimostrare le nostre abilità di ingegneri della conoscenza.
In futuro, questo progetto potrebbe essere migliorato con un nuovo dataset con una maggiore correlazione e aggiungendo altre funzionalità al sistema esperto, che potrebbero essere usate, per esempio, da una squadra reale per la gestione del club.