

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**](#_heading=h.30j0zll) **2**

**2.** [**LINGUAGGIO E AMBIENTI UTILIZZATI**](#_heading=h.1fob9te) **2**

**3.** [**ATTIVITA’ DI RICERCA E DATASET**](#_heading=h.3znysh7) **3**

**4.** [**KNOWLEDGE BASE**](#_heading=h.2et92p0) **5**

**5.** [**SISTEMA ESPERTO PER OPERAZIONI SULLA KNOWLEDGE BASE**](#_heading=h.tyjcwt) **5**

**6.** [**PREDIZIONI DEI RISULTATI DELLE PARTITE**](#_heading=h.1t3h5sf) **14**

**7.** [**ANALISI DEI RISULTATI: PREDIZIONE DELLE PARTITE**](#_heading=h.4d34og8) **25**

**8.** [**CONCLUSIONI**](#_heading=h.2s8eyo1) **32**

# 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](https://github.com/yuce/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](https://www.swi-prolog.org/) 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 onlin](https://swish.swi-prolog.org/https:/swish.swi-prolog.org/)[e](https://swish.swi-prolog.org/) 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](https://colab.research.google.com/). 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](https://www.sofascore.com/it/); 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

Immagine che contiene testo, elettronico

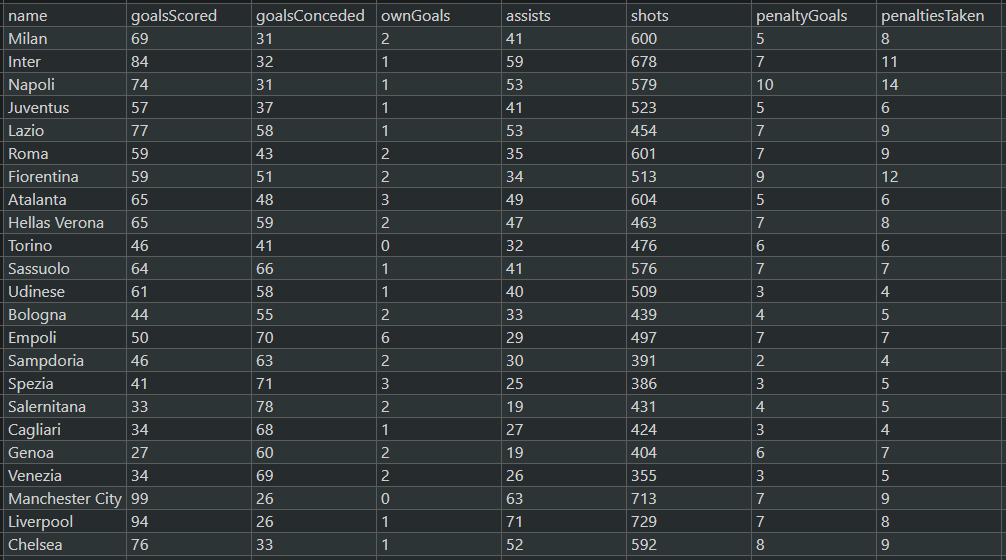
Descrizione generata automaticamente

* ***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

Immagine che contiene testo, tabellonesegnapunti

Descrizione generata automaticamente

* ***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.



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

Immagine che contiene testo, armadietto

Descrizione generata automaticamente

# 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***.

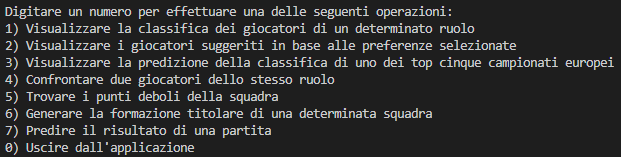
# 

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:



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.

Immagine che contiene testo

Descrizione generata automaticamente

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.

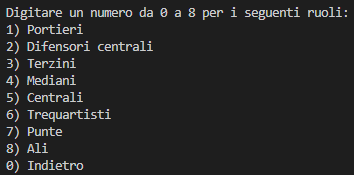
**1. Visualizzare la classifica dei giocatori di un determinato ruolo**

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.Immagine che contiene testo

Descrizione generata automaticamente

Menù iniziale per questa voce 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.

Immagine che contiene testo

Descrizione generata automaticamente

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.

Immagine che contiene testo

Descrizione generata automaticamente

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

Immagine che contiene testo

Descrizione generata automaticamente

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.

Selezione del ruolo su cui effettuare la modifica di una o più caratteristicheImmagine che contiene testo

Descrizione generata automaticamente

# 

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

Immagine che contiene testo, nero, schermo

Descrizione generata automaticamente

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*.

Immagine che contiene testo

Descrizione generata automaticamente

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.

Immagine che contiene testo

Descrizione generata automaticamente Immagine che contiene testo

Descrizione generata automaticamente Immagine che contiene testo

Descrizione generata automaticamente

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.

Immagine che contiene testo

Descrizione generata automaticamenteImmagine che contiene testo

Descrizione generata automaticamente

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.

Immagine che contiene testo

Descrizione generata automaticamente

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

Immagine che contiene testo

Descrizione generata automaticamente

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

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente

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.

Immagine che contiene testo, interni, screenshot, vicino

Descrizione generata automaticamente

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.

Immagine che contiene testo

Descrizione generata automaticamente

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).

Immagine che contiene testo

Descrizione generata automaticamente

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.

Immagine che contiene testo

Descrizione generata automaticamente

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.

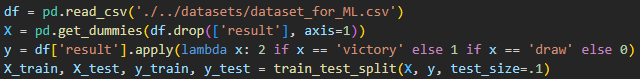
# 6. PREDIZIONI DEI RISULTATI DELLE PARTITE

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.



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](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html) 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](https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html) 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**
   1. **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.

* 1. **Implementazione e studio di iper-parametri**

L’implementazione dell’algoritmo è stata effettuata tramite la classe [RandomForestClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html) 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, 1000Immagine che contiene testo

  Descrizione generata automaticamente
  1. **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.

1. **Naive Bayes**
   1. **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.

* 1. **Implementazione e studio**

L’implementazione dell’algoritmo è stata effettuata tramite le classi [*MultinomialNB*](https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html?highlight=multinomialnb#sklearn.naive_bayes.MultinomialNB)*,* [*ComplementNB*](https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.ComplementNB.html?highlight=complementnb#sklearn.naive_bayes.ComplementNB)*,* [*BernoulliNB*](https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.BernoulliNB.html?highlight=bernoullinb#sklearn.naive_bayes.BernoulliNB)*,* [*CategoricalNB*](https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.CategoricalNB.html?highlight=categoricalnb#sklearn.naive_bayes.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.Immagine che contiene testo, monitor, screenshot, interni

Descrizione generata automaticamente

* 1. **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.

1. **K-Nearest Neighbours**
   1. **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.

* 1. **Implementazione e studio**

L’implementazione dell’algoritmo è avvenuta tramite la classe [KNeighboursClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html) 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.

**Immagine che contiene testo

Descrizione generata automaticamente**

**Immagine che contiene testo

Descrizione generata automaticamente**

* 1. **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.

1. **Support Vector Machine**
   1. **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.

* 1. **Implementazione e studio**

L’implementazione è stata effettuata utilizzando la libreria [SVM](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html) 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)

**Immagine che contiene testo

Descrizione generata automaticamente**

**Immagine che contiene testo

Descrizione generata automaticamente**

* 1. **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.

1. **Neural Network**
   1. **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.

* 1. **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](https://www.adriangb.com/scikeras/stable/generated/scikeras.wrappers.KerasClassifier.html) della libreria scikeras e le classi [Sequential](https://keras.io/api/models/sequential/) e [Dense](https://keras.io/api/layers/core_layers/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

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente

* 1. **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

1. **Bayesian Network**
   1. **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.

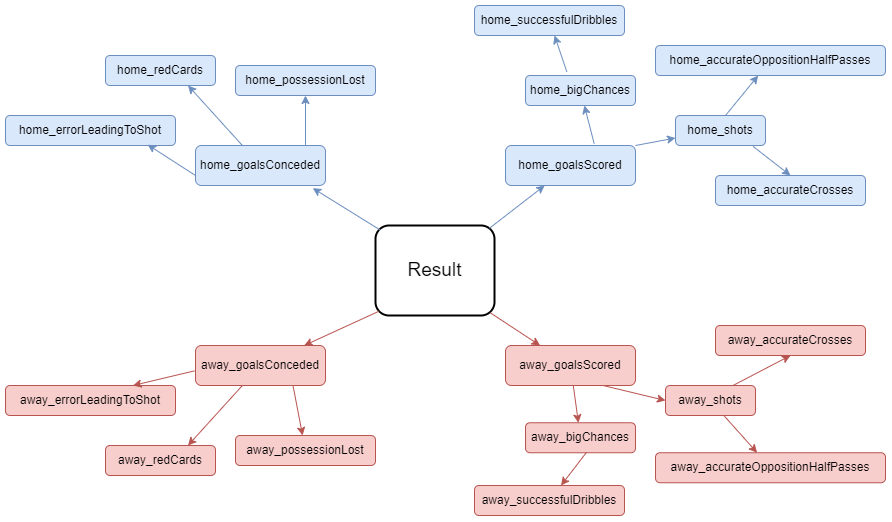
* 1. **Implementazione e studio**

L’implementazione è stata possibile tramite la classe [BayesianNetwork](https://pgmpy.org/models/bayesiannetwork.html) 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.



* 1. **Analisi dei risultati**

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

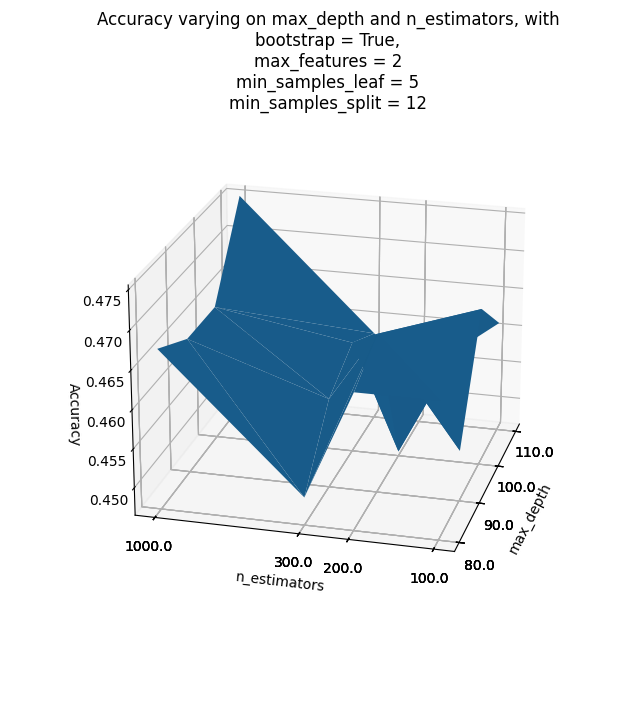
# ANALISI DEI RISULTATI: PREDIZIONE DELLE PARTITE

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:

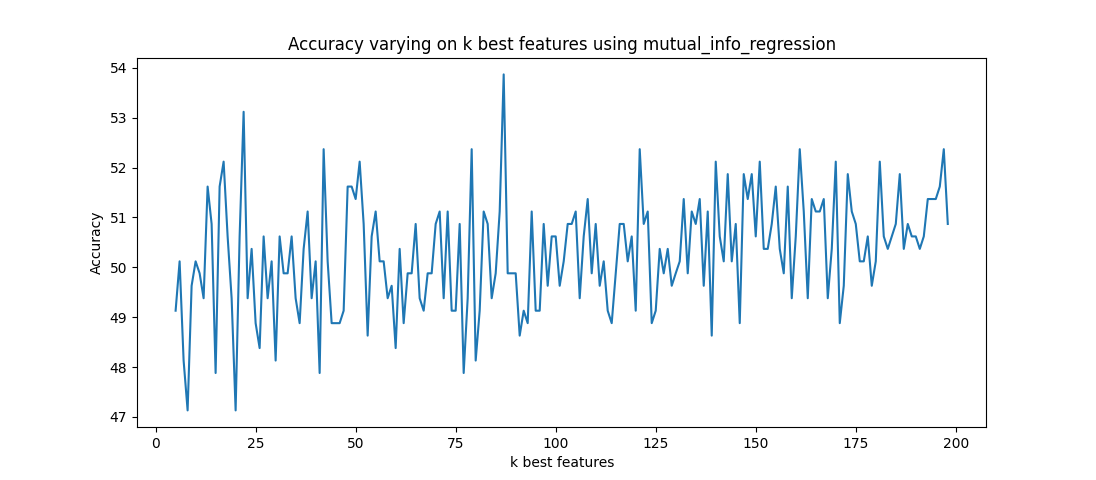
|  |  |  |  |
| --- | --- | --- | --- |
| **Algoritmo ML** | **Iper-parametri** | **Funzione e k best features** | **Accuracy** |
| ***Random Forest*** | ***bootstrap: True***  ***max\_depth: 80***  ***max\_features: 2***  ***min\_samples\_leaf: 5***  ***min\_samples\_split: 12***  ***n\_estimators: 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:

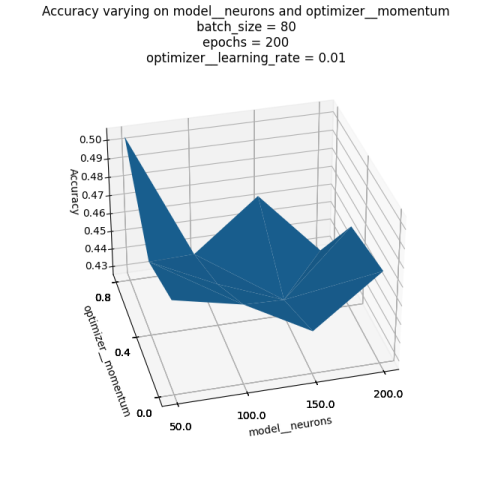
Random Forest Grid search



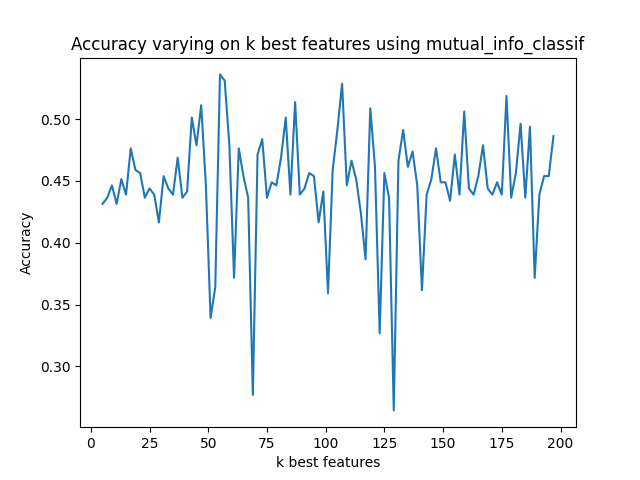
Random Forest KBest Selection



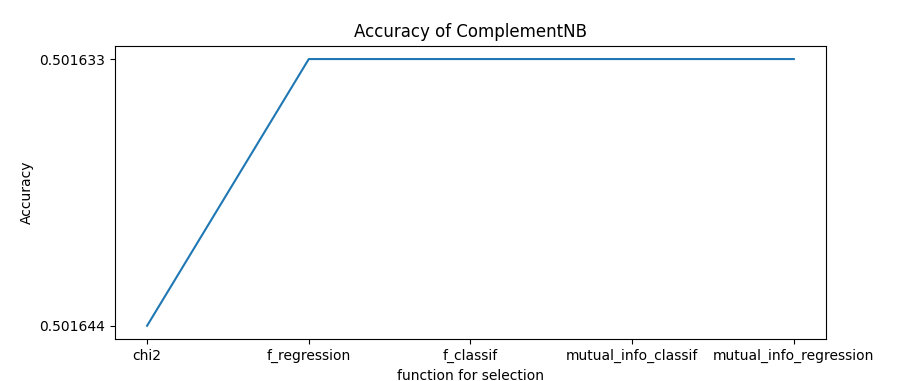
Neural Network Grid search



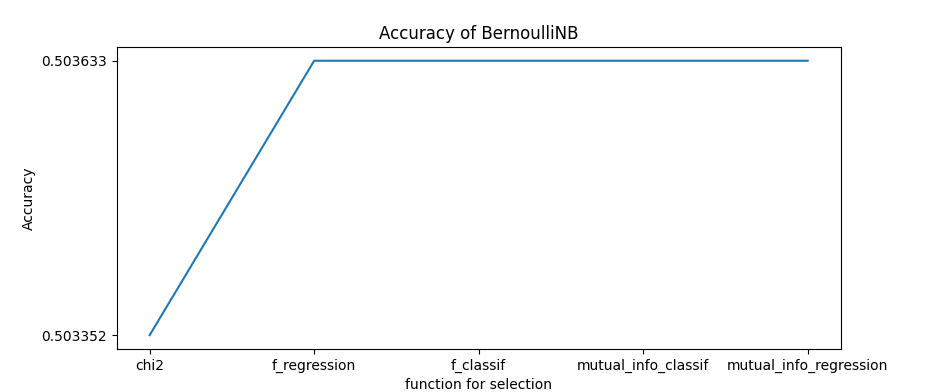
Neural Network KBest Selection



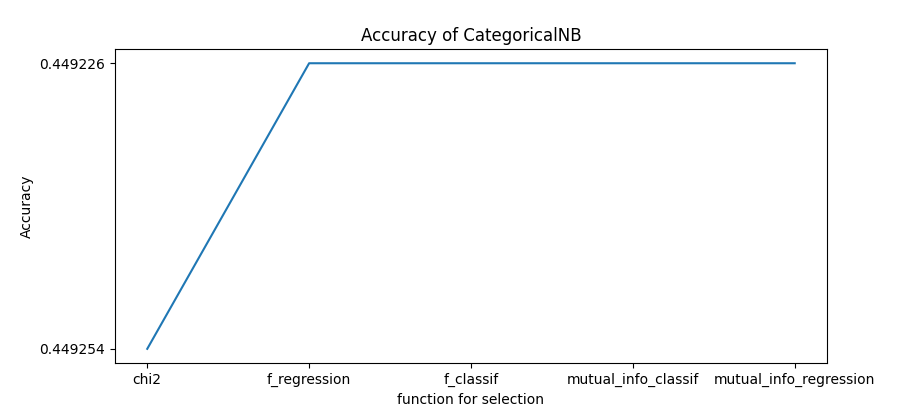
Naïve Bayes ComplementNB (Best)



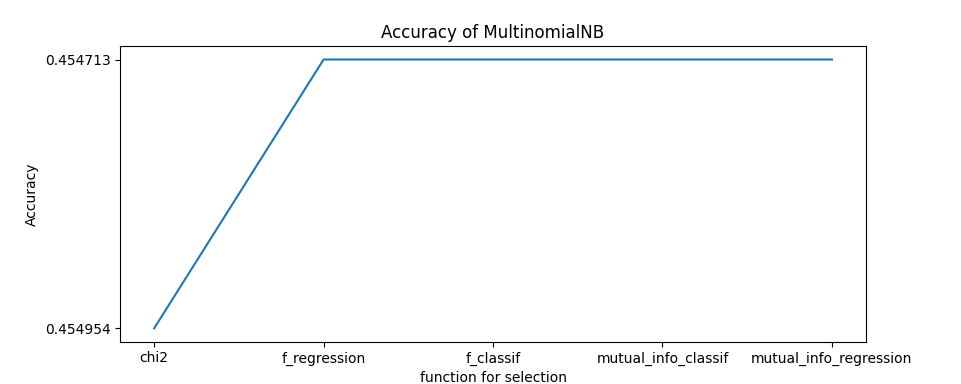
Naïve Bayes BernoulliNB



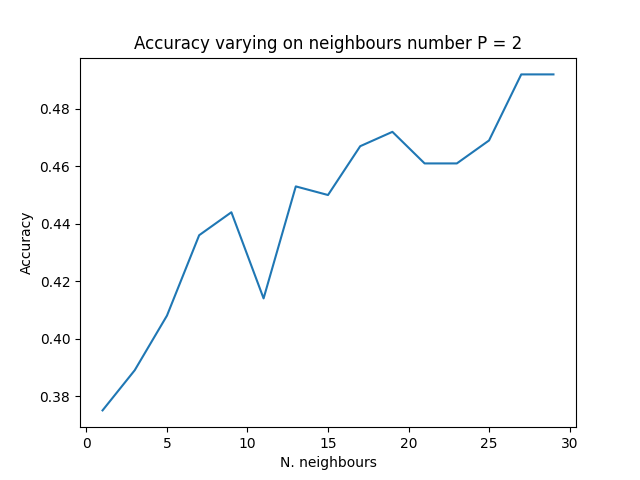
Naïve Bayes CategoricalNB



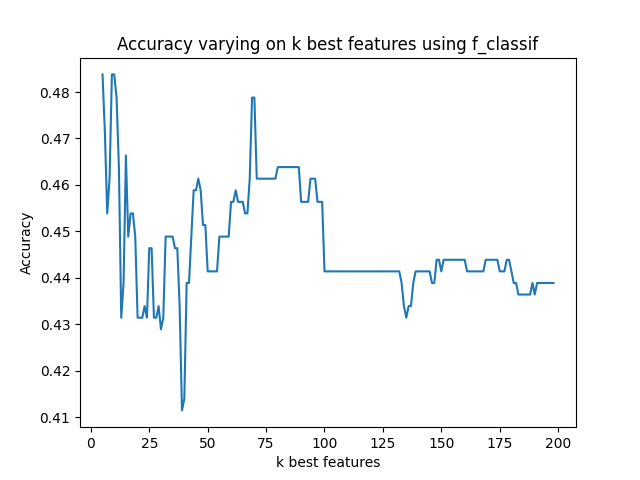
Naïve Bayes MultinomialNB



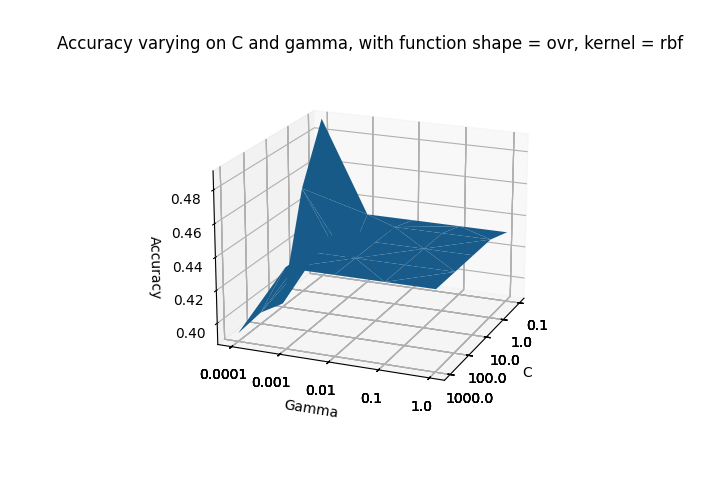
K-Nearest Neighbours Grid search



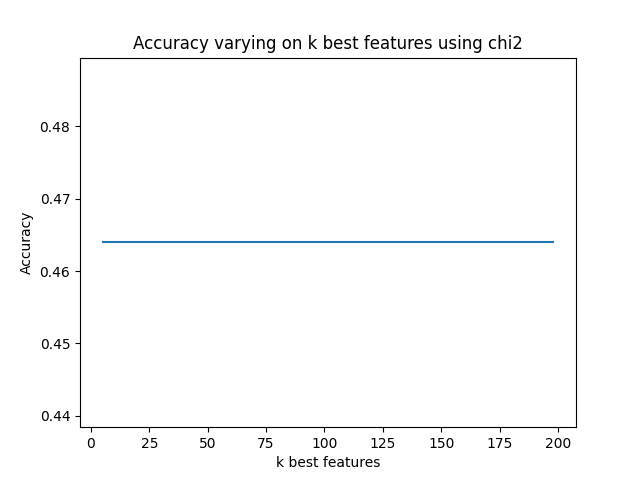
K-Nearest Neighbours KBest Selection



SVM Grid search

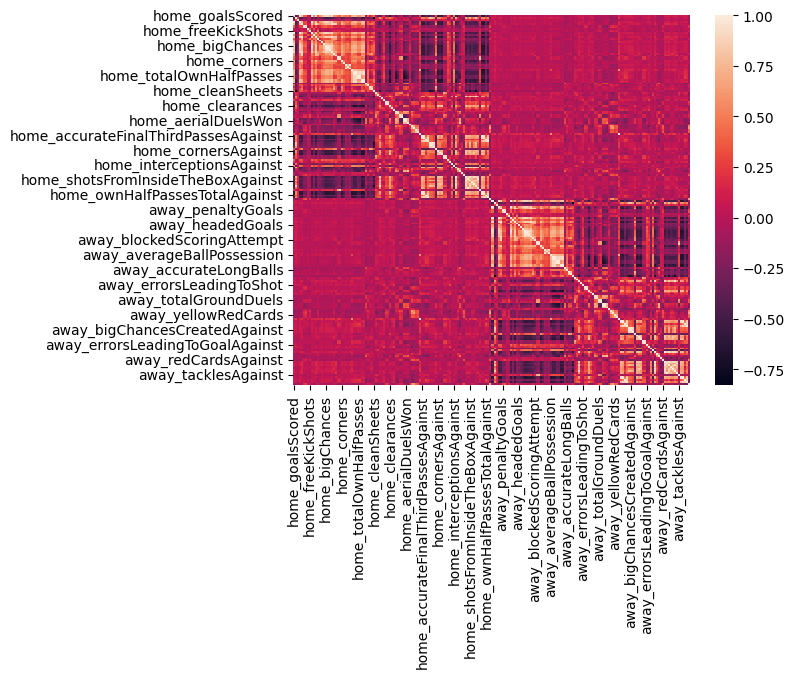


SVM KBest Selection



# 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).

# 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.