Ingegneria della Conoscenza applicata al videogioco FIFA 23

Michelangelo Balzano

Gabriele Grossano

a.a. 2022/2023

link repository GitHub:
`https://github.com/michelangelobalzano/progetto-Icon`

Indice

1. Introduzione al progetto

1.1. FIFA 23

FIFA 23 è un videogioco di calcio sviluppato da EA Sports e disponibile su tutte le principali piattaforme di videogiochi della serie di videogiochi chiamata, appunto, FIFA. Al suo interno ci sono le squadre dei principali campionati di calcio di tutto il mondo. Ogni squadra ha ovviamente la sua lista di calciatori, tutti con caratteristiche differenti.

1.2. Il dataset

Il dataset di cui disponiamo è stato trovato su Kaggle e contiene le informazioni su tutti i calciatori presenti sul gioco. Oltre al nome del calciatore e alla squadra di appartenenza, ci sono tanti attributi riguardanti le qualità tecniche. Al calciatore è associato un "overall", ovvero un punteggio complessivo, che rappresenta il calciatore in base alle sue qualità.

Il nostro progetto si concentra principalmente sull'overall dei calciatori che, come tutti gli altri attributi riguardanti la qualità, è un numero, che può assumere un valore tra 0 e 100.

1.3. Compiti effettuati

1.3.1 Apprendimento supervisionato

Il primo compito riguarda l'**apprendimento supervisionato**. In particolare, ci siamo concentrati sulla predizione dell'overall di un calciatore a partire dalle caratteristiche tecniche specifiche. Abbiamo confrontato differenti modelli di apprendimento supervisionato: la regressione lineare, gli alberi decisionali, il knn e il random forest. Per minimizzare problemi di sovradattamento dei modelli abbiamo inoltre applicato la tecnica di convalida incrociata *k-fold cross validation* su tutti i modelli appena citati. Inoltre, per tutti i modelli, tranne che per quello della regressione lineare, sono stati testati valori differenti per i propri iperparametri più incisivi. La metrica di valutazione e di confronto dei modelli utilizzata è l'RMSE.

1.3.2 CSP

Il secondo compito è la risoluzione di un **Costraints Satisfaction Problem** (CSP). L'obiettivo è di trovare la migliore squadra titolare possibile, dato un modulo e un budget massimo presi in input, in base all'overall dei calciatori. Ad ogni modulo corrispondono undici ruoli differenti che l'algoritmo deve rispettare per cercare il miglior giocatore per ruolo.

Il nostro dataset si presta bene a tale problema di ricerca in quanto per ogni calciatore c'è l'attributo "Positions Played" contenente le posizioni che lo stesso può occupare in campo.

1.4. Strumenti utilizzati

Il progetto è stato sviluppato in python. Per quanto riguarda la task dell'apprendimento supervisionato, le librerie utilizzate sono le seguenti:

- **scikit-learn**: per i metodi che implementano i modelli di apprendimento e altre funzioni utili allo scopo;
- **pandas**: per gestire il dataset in formato "csv";
- **numpy**: per metodi riguardanti calcoli matematici;
- **matplotlib**: per la creazione e visualizzazione di grafici.

Per la task della risoluzione del CSP abbiamo utilizzato le seguenti librerie:

- **numpy**: per metodi riguardanti calcoli matematici;
- **pandas**: per gestire il dataset in formato "csv";
- **matplotlib**: per la creazione e visualizzazione di grafici.

2. Apprendimento supervisionato

L'obiettivo è di prevedere il valore di overall di un calciatore a partire dai valori delle singole caratteristiche tecniche.

2.1. Preprocessing

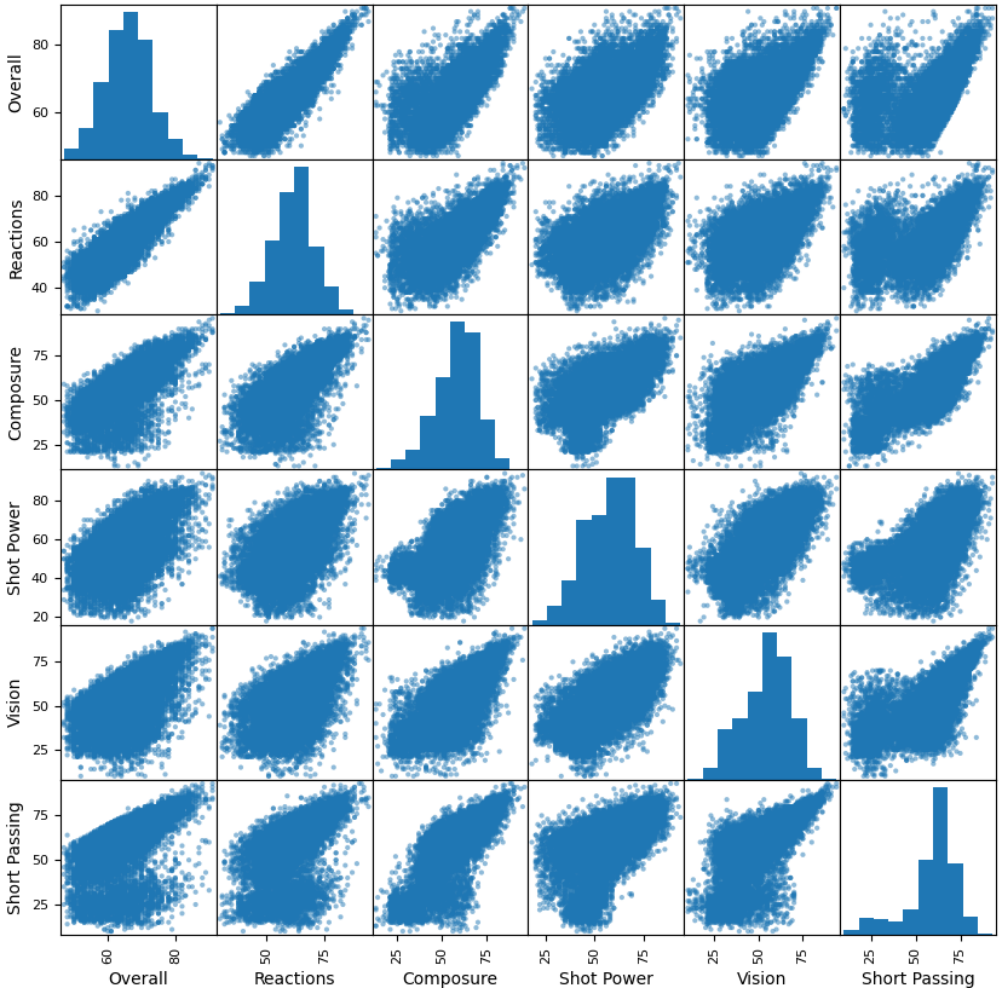
Prima di iniziare ad effettuare le predizioni, abbiamo svolto le seguenti attività di **preprocessing** dei dati:

1. **Verifica dati mancanti**: il primo passo è quello di verificare se nel dataset ci sono dei valori mancanti per uno o più attributi e per ogni calciatore. Il nostro dataset è una tabella 18539 x 89, ovvero 18539 calciatori e 89 attributi per calciatore, ordinati proprio per overall. Successivamente abbiamo stampato le informazioni di ciascun attributo e, per fortuna, tutti gli attributi sono di tipo "not-null", dunque non ci sono valori mancanti.
2. **Rimozione attributi superflui**: una tupla del dataset, che rappresenta il singolo calciatore, contiene tante informazioni che non sono utili al nostro scopo come il nome e cognome, la squadra di appartenenza, il suo valore economico, il suo stipendio, ecc. Tali colonne vanno rimosse lasciando soltanto quelle riguardanti le caratteristiche tecniche. Gli attributi rimasti sono i seguenti:
   [*Overall, Crossing, Finishing, Heading Accuracy, Short Passing, Volleys, Dribbling, Curve, Freekick Accuracy, LongPassing, BallControl, Acceleration, Sprint Speed, Agility, Reactions, Balance, Shot Power, Jumping, Stamina, Strength, Long Shots, Aggression, Interceptions, Positioning, Vision, Penalties, Composure, Marking, Standing Tackle, Sliding Tackle, Goalkeeper Diving, Goalkeeper Handling, Goalkeeper Kicking, Goalkeeper Positioning, Goalkeeper Reflexes*].

3. **Ricerca di correlazioni**: prima di cominciare con l'apprendimento, abbiamo calcolato il coefficiente di correlazione tra tutte le coppie di attributi, tra quelli rimasti, in modo tale da capire quanto l'uno incide sull'altro, creando una matrice di correlazioni. Ovviamente, ciò che ci interessa è la colonna della matrice corrispondente all'overall in modo da visualizzare la correlazione di tutti gli altri attributi con lo stesso. Il risultato ottenuto è quello mostrato in Figura 1.

Overall	1.000000	Heading Accuracy	0.342452
Reactions	0.872789	Finishing	0.331967
Composure	0.700583	Penalties	0.331834
Shot Power	0.552518	Marking	0.313975
Vision	0.523290	Interceptions	0.311216
Short Passing	0.520784	Jumping	0.282737
LongPassing	0.507192	Standing Tackle	0.266683
BallControl	0.457705	Agility	0.254826
Curve	0.416575	Sliding Tackle	0.240499
Crossing	0.395435	Sprint Speed	0.213367
Aggression	0.393433	Acceleration	0.198758
Long Shots	0.392187	Balance	0.142749
Stamina	0.376994	Goalkeeper Positioning	-0.008318
Dribbling	0.376185	Goalkeeper Reflexes	-0.013110
Freekick Accuracy	0.374162	Goalkeeper Handling	-0.013655
Volleys	0.367168	Goalkeeper Diving	-0.016593
Strength	0.353857	Goalkeeper Kicking	-0.016764
Positioning	0.348038		

Inoltre, abbiamo selezionato i migliori 5 attributi, in base al loro valore di correlazione, e abbiamo visualizzato graficamente la loro incisione sull'overall. Più un valore è incisivo, più tende ad essere direttamente proporzionale al valore sul quale incide, quindi, il suo grafico tenderà ad una retta del tipo $y = x$ (Figura 2).
Il grafico dimostra la forte correlazione che c'è tra l'overall e l'attributo *Reactions*.
Dall'analisi delle correlazioni, inoltre, si evince come gli attributi riguardanti i portieri abbiano una correlazione negativa, ovvero inferiore allo 0. Questo risultato afferma che tali caratteristiche diminuiscono all'aumentare dell'overall, e viceversa.


Overall
Reactions
Composure
Shot Power
Vision
Short Passing

4. **Rimozione dei portieri**: a causa delle correlazioni negative degli attributi riguardanti le caratteristiche tecniche dei portieri, abbiamo capito che, per i giocatori di questo ruolo, dovrebbe essere effettuata un'attività di apprendimento separato.
   Non essendo giocatori di movimento, i portieri hanno caratteristiche tecniche totalmente diverse: le caratteristiche riguardanti le abilità con i piedi sono molto basse. Analogamente, per tutti gli altri ruoli, le caratteristiche dei portieri hanno valori molto bassi. Abbiamo deciso di rimuovere dal dataset tutti i portieri e tutte le caratteristiche che li riguardano.
5. **Suddivisione dei dati**: prima di scegliere il modello di apprendimento supervisionato più adatto, abbiamo suddiviso il dataset in due sottoinsiemi: un **training set** per effettuare apprendimento, e un **test set** per applicare il modello scelto ed effettuare le predizioni. In particolare la suddivisione è dell'80% per gli esempi di addestramento e il rimanente 20% per i test, ovviamente scelti in modo casuale.

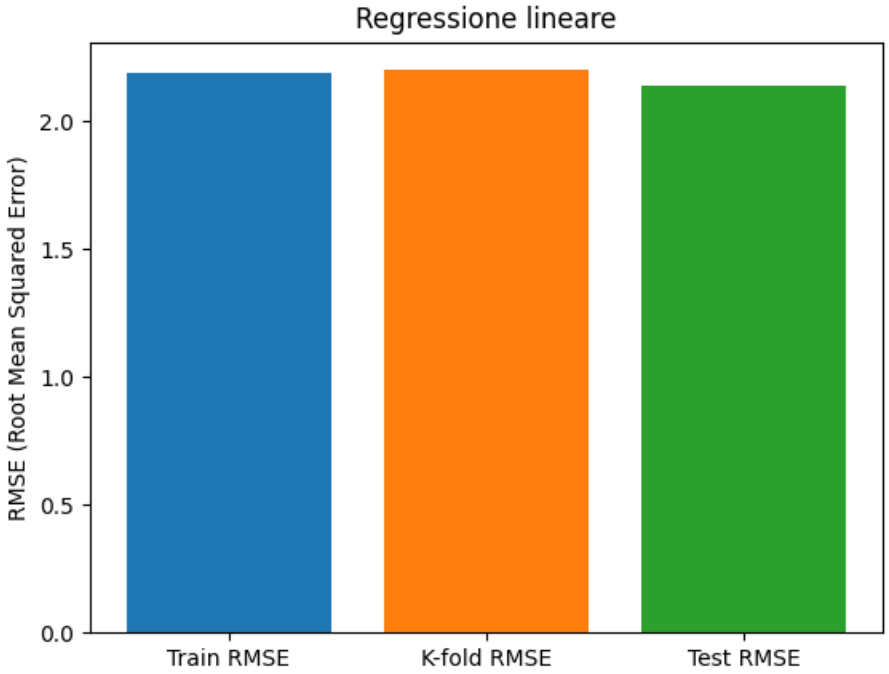
2.2. Valutazione dei modelli

Come metodo di valutazione dell'errore dei modelli utilizzati abbiamo scelto l'**RMSE**, ovvero la radice dell'errore quadratico medio. Tale misura è molto utilizzata nei problemi di regressione. Esso rappresenta, in poche parole, la distanza che c'è tra il vettore di valori previsti e il vettore dei valori osservati.

2.3. Regressione lineare

Il primo modello testato è quello della **regressione lineare**. Gli step che abbiamo effettuato sono i seguenti:

1. **Addestramento del modello**: in questa fase addestriamo il modello utilizzando il training set in modo da migliorarlo. L'RMSE ottenuto è di circa *2.18*, non male;
2. **Validazione del modello**: per convalidare i modelli e ridurre il problema del sovradattamento, abbiamo utilizzato la **k-fold cross validation** con il valore k impostato a 10. Il modello valuta quindi 10 fold differenti. Successivamente abbiamo calcolato la media dei 10 valori ottenuti da ciascun fold e il risultato ottenuto è il linea con il precedente, ovvero *2.19*;
3. **Valutazione del modello**: dopo aver addestrato il modello, l'abbiamo applicato al test set, ottenendo, più o meno, sempre lo stesso risultato: *2.17*.



I tre risultati ottenuti dalle tre fasi dell'apprendimento sono molto simili tra loro. In questo caso, abbiamo un modello ben bilanciato che non presenta problemi di sovradattamento e generalizza bene su dati che non ha mai visto, ovvero quelli del test set.

2.4. Albero decisionale

Il secondo modello analizzato è quello dell'**albero decisionale**. Abbiamo deciso di valutare questo modello con differenti valori per l'attributo *max_ depth*. Questo attributo indica la profondità massima dell'albero. I valori di *max_ depth* testati sono:

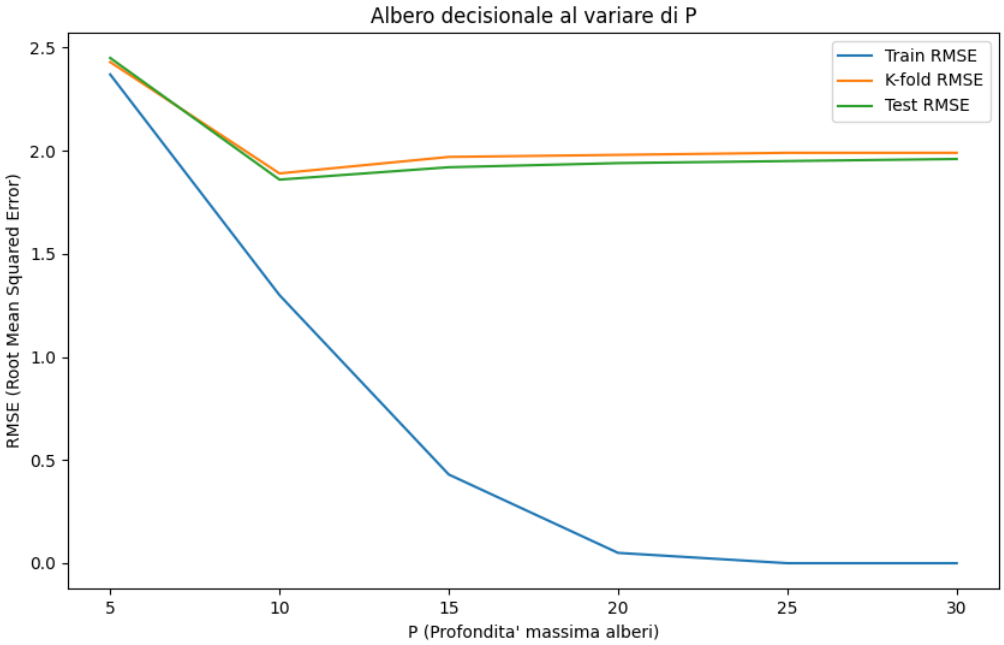
$$5, 10, 15, 20, 25, 30$$

Tutte e tre le fasi dell'apprendimento sono state svolte per ogni profondità massima selezionata.

Nella fase di training (linea blu Figura 4), abbiamo notato che all'aumentare della profondità massima, l'RMSE tende a *0*. Potrebbe essere un modello perfetto oppure potrebbe verificarsi un eccessivo sovradattamento.

Successivamente abbiamo effettuato la validazione (sempre con la *k-fold cross validation*) e la valutazione del modello alle diverse profondità massime (linee arancione e verde Figura 4). La profondità massima dell'albero che ci fa ottenere il miglior punteggio è *10*, con un RMSE uguale a circa *1.9*.

Abbiamo effettuato un ulteriore test senza imporre un limite di profondità notando che da una profondità massima di 15 in poi, l'RMSE rimane costante (circa 1.9). I risultati ottenuti sono descritti dalla Figura 5.



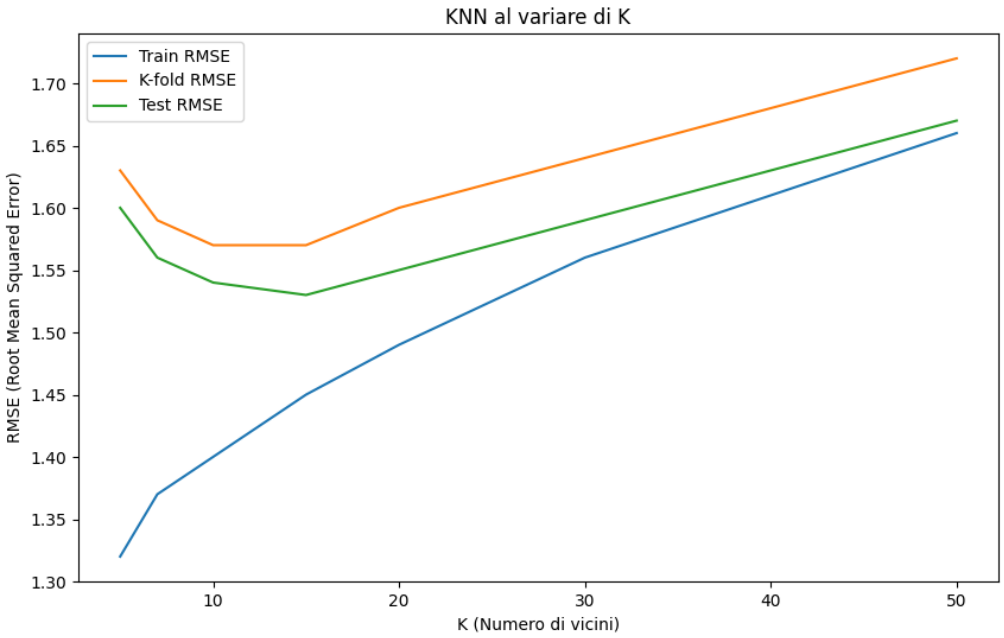
2.5. KNN

Il terzo modello che testiamo è un modello del **case-based reasoning**, ovvero il **KNN**. Per il knn abbiamo valutato differenti valori per l'attributo *n_neighbors* che indica il numero di vicini da considerare. I valori valutati sono i seguenti:

$$5, 7, 10, 15, 20, 30, 50$$

Non abbiamo considerato valori minori di 5 in quanto rendono il modello molto sensibile al rumore, facendosi influenzare da dati anormali. Ciò infatti provoca un alto RMSE. Come numero di vicini massimo ci siamo fermati a 50 per non semplificare troppo il modello. Infatti, l'RMSE del modello sul training set tende ad aumentare man mano che il numero di vicini cresce. Questo si verifica in quanto il modello diventa sempre più semplice, fino a diventare poco adattabile ai dati.

Analizzando i risultati ottenuti con la *k-fold cross validation* e la valutazione sul test set, sembra che il miglior valore per il numero di vicini sia intorno a $10 - 15$, dove otteniamo un RMSE di circa 1.53 (Figura 5).



2.6. Random forest

L'ultimo modello è quello del **random forest**. Questo modello ha differenti iperparametri che determinano di molto il risultato dell'apprendimento. In particolare, abbiamo valutato diversi valori per gli iperparametri *n_estimators* e *max_depth*, che rappresentano rispettivamente il numero di alberi della foresta e la profondità massima del singolo albero, come già visto per il modello dell'albero decisionale.

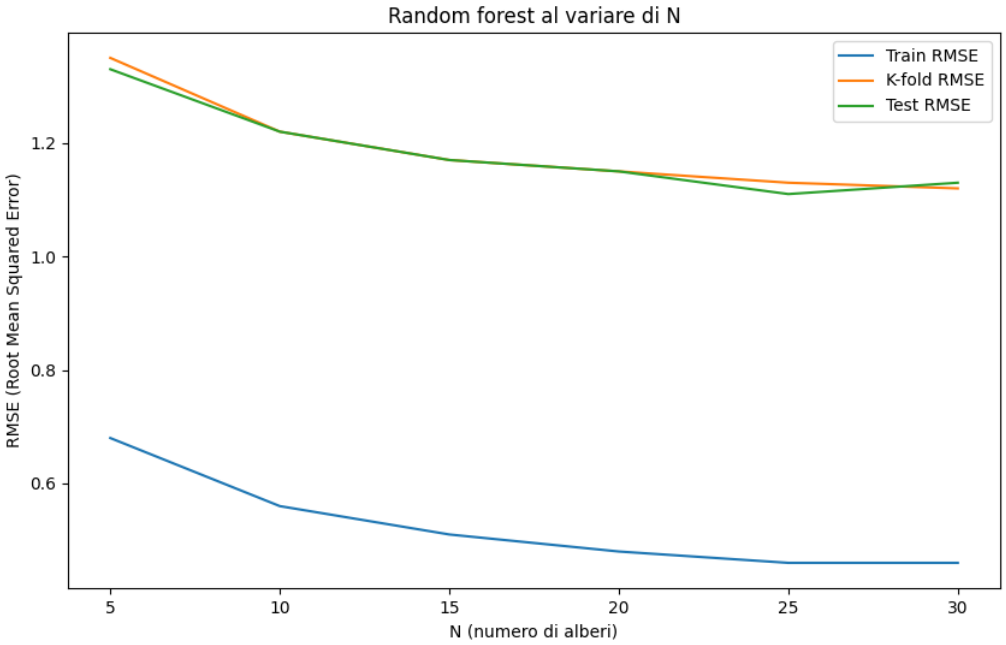
Il random forest ha un altro importante iperparametro: *max_features*, che indica il numero massimo di caratteristiche che vengono considerate per la suddivisione in ogni nodo dell'albero. In altre parole, limita il sottoinsieme delle caratteristiche che ogni albero può utilizzare per prendere decisioni durante la costruzione. Per questo iperparametro, dopo vari test, abbiamo deciso di mantenere il suo valore fisso ad 8. Valori maggiori vanno ad aumentare l'RMSE oltre che il costo computazionale.

Innanzitutto abbiamo provato diversi valori per *n_estimators*, tra i quali

$$5, 10, 15, 20, 25, 30$$

senza imporre un limite sulla profondità massima. Ci siamo fermati a 30 come valore massimo perché valori più alti impiegano un elevato tempo di calcolo.

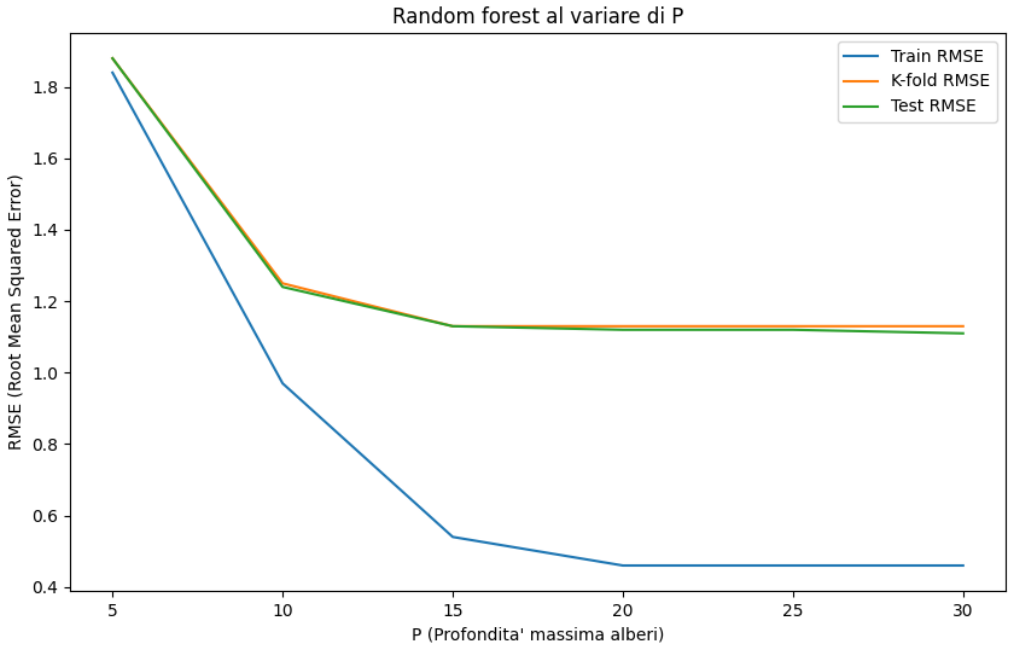
Il miglior risultato l'abbiamo ottenuto proprio con *n_estimators* uguale a 30 e ce l'aspettavamo poiché il modello del random forest aumenta l'adattabilità ai dati all'aumentare del numero di alberi.



Dopo aver selezionato il miglior valore per l'attributo *n_ estimators* (30), abbiamo valutato differenti valori per l'attributo *max_ depth*, ovvero

$$5, 10, 15, 20, 25, 30$$

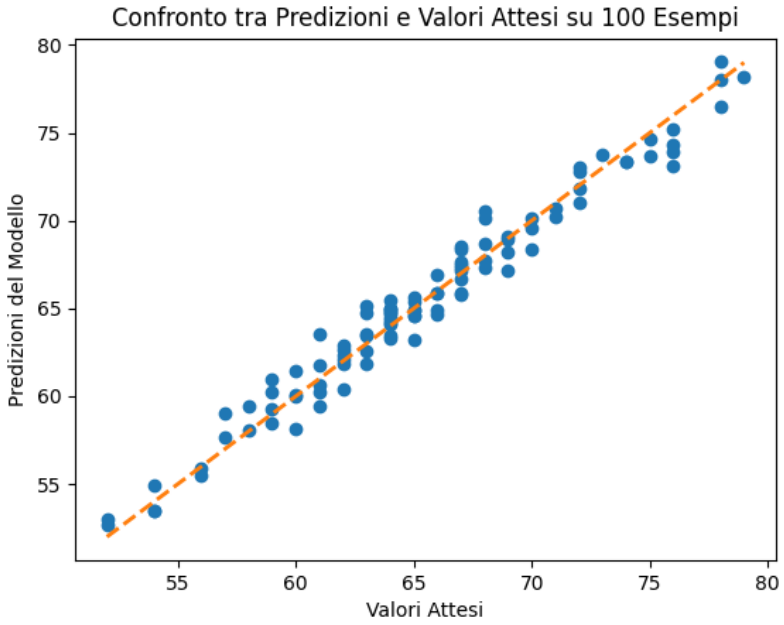
gli stessi utilizzati per *max_ depth* nel modello dell'albero decisionale (Figura 7).



Conclusione: il miglior modello tra quelli testati è il random forest in quanto ha l'RMSE più basso. I suoi miglior valori per gli iperparametri di *n_ estimators* e *max_ depth*, tra quelli provati sono $25 - 30$ per *n_ estimators* e $15 - 20$ per *max_ depth*.

2.7. Considerazioni finali

Come ulteriore dimostrazione dell'efficienza del modello, abbiamo realizzato un grafico di dispersione che mostra come le predizioni del modello si allineano rispetto ai valori attesi. Per far ciò abbiamo selezionato 100 esempi casuali dal test set, abbiamo effettuato le previsioni e le abbiamo confrontate con gli attributi target degli esempi (Figura 8). I valori degli iperparametri di *n_ estimators* e *max_ depth* sono rispettivamente 25 e 20.



3. CSP

Il CSP affrontato consiste nel cercare di ottenere la miglior formazione possibile per un determinato modulo e che rispetti un budget economico. Un modulo corrisponde ad un vettore di undici posizioni differenti, ognuna del quale deve essere ricoperta da un calciatore. Ogni calciatore è dotato di un attributo *Positions played* che comprende l'insieme delle posizioni che può ricoprire. Inoltre, ad ogni calciatore è associato l'attributo *Value(in euro)*, che rappresenta il suo valore economico.

Vista la grandezza del dataset (18000+ tuple), applicare algoritmi di ricerca che analizzano l'intero spazio delle soluzioni non sarebbe stato ragionevole. Poiché il nostro obiettivo è la massimizzazione di un valore, abbiamo deciso di utilizzare metodi di ricerca locale che analizzano soltanto una porzione dello spazio delle soluzioni sapendo però che potrebbero convergere ad un massimo locale e non globale.

Il CSP è stato risolto attraverso cinque algoritmi di ricerca locale differenti: *Hill Climbing*, *Tabu Search*, *Simulated Annealing*, *Most Improving Step* e un *algoritmo genetico*. Ogni algoritmo è stato testato 500 volte sullo stesso modulo e con lo stesso budget a disposizione per poi calcolare la media dei risultati ottenuti. Il modulo testato è il *4-3-3*, con un budget di 350 milioni.

La funzione di valutazione utilizzata si basa sulla media dell'overall dei calciatori. Una singola sostituzione è migliorativa se va ad incrementare l'overall medio della formazione, non peggiorativa se l'overall della sostituzione è uguale, peggiorativa se è inferiore. Inoltre, ogni sostituzione effettuata rispetta il budget. Se una sostituzione fa sforare il budget viene ignorata direttamente.

3.1. Preprocessing

L'attività di preprocessing per il task di risoluzione del CSP consiste nei seguenti passaggi:

1. **verifica dati mancanti**: la verifica è già stata effettuata nell'attività di apprendimento supervisionato. Il dataset non presenta dati mancanti;
2. **rimozione attributi superflui**: per questa task sono necessari soltanto gli attributi del nome del calciatore, le posizioni in cui può giocare, l'overall e il suo valore economico. Tutti gli altri valori sono stati scartati;
3. **conversione euro - milioni di euro**: per rendere più chiare e meno confusionarie le cifre del valore del calciatore, abbiamo effettuato la conversione da euro a milioni di euro (ad esempio: 50000000 $\rightarrow$ 50);
4. **conversione sigla posizione - nome posizione**: come per il valore economico, vogliamo rendere più chiaro anche la posizione, convertendo le sigle in nomi di posizione (ad esempio: GK $\rightarrow$ Portiere);
5. **creazione vettore calciatori**: una volta terminate tutte le fasi precedenti, è stato creato un vettore contenente tutti i calciatori, presenti una volta per ogni posizione in cui possono giocare, dal quale pescare per effettuare le sostituzioni.

3.2. Hill Climbing

Il metodo Hill Climbing è il più semplice e immediato metodo di ricerca locale. Consiste nel Iterative Best Improvement che cerca di massimizzare il valore dato dalla funzione di valutazione.

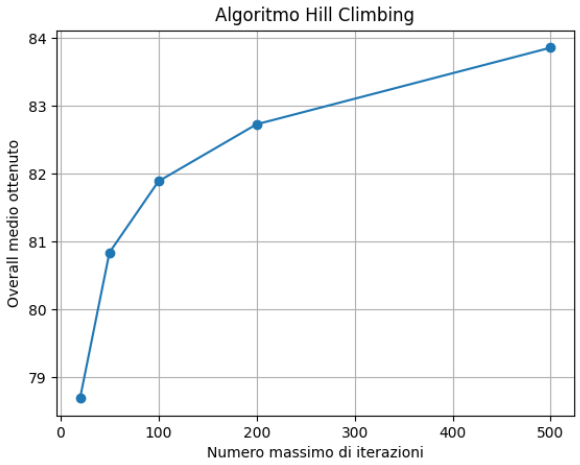
Nel nostro caso, l'algoritmo effettua la sostituzione se si verifica una delle seguenti due condizioni:

- il budget non viene sforato e l'overall medio viene migliorato;
- il costo della singola assegnazione è inferiore a quello attuale e l'overall della singola assegnazione ha un peggioramento massimo del 3% rispetto all'overall da sostituire.

L'Hill Climbing implementato non effettua soltanto mosse migliorative ma anche mosse peggiorative che però vanno a diminuire il costo totale della formazione.

L'algoritmo è stato testato con un diverso numero di iterazioni massime, in modo da poter studiare come i risultati cambiano all'aumentare del numero di mosse effettuate (random walk). I valori testati sono:

$$20, 50, 100, 200, 500$$



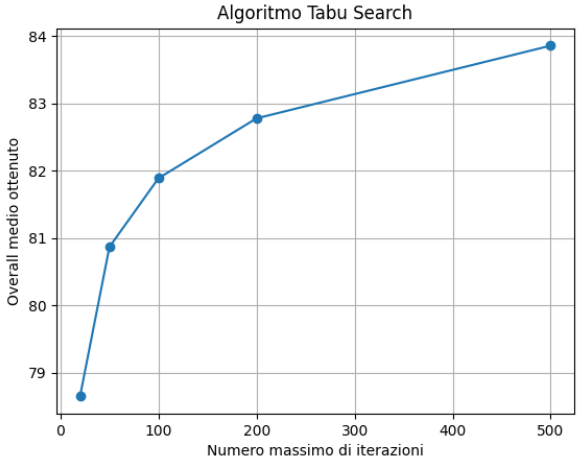
I risultati crescono all'aumentare del numero massimo di iterazioni, arrivando ad ottenere, con un numero di iterazioni massime uguale a 500, ad un overall medio di circa 83.8.

3.3. Tabu Search

Poichè l'algoritmo dell'Hill Climbing, valutando anche soluzioni non migliorative, potrebbe tornare su sostituzioni già effettuate, abbiamo testato anche l'algoritmo del Tabu Search, che mantiene memorizzate le ultime k sostituzioni. Il valore k rappresenta la lunghezza della lista dei tabu e l'abbiamo fissato a 50.

Anche per il Tabu Search abbiamo valutato differenti valori per il numero massimo di iterazioni, gli stessi utilizzati nell'Hill Climbing.

La funzione di valutazione è quella dell'Hill Climbing: la sostituzione viene effettuata se si verifica una delle due condizioni discusse precedentemente.



L'andamento è molto simile a quello dell'Hill Climbing. Non c'è stato un miglioramento. Questo è dato dal fatto che già l'Hill Climbing è stato ottimizzato in quanto non permette sostituzioni molto peggiorative e non permette di sforare il budget. Succede quindi raramente di tornare su sostituzioni già effettuate, dunque si ottengono risultati simili.

3.4. Simulated Annealing

Il Simulated Annealing permette di considerare anche sostituzioni molto peggiorative con lo scopo di superare minimi locali, consentendo l'esplorazione di soluzioni che potrebbero portare a un massimo globale più elevato.

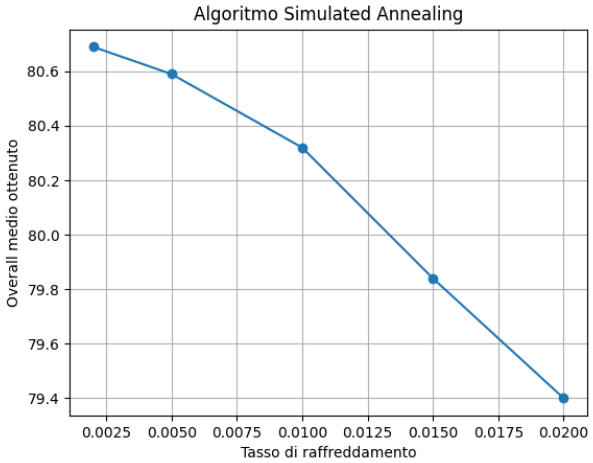
Si parte da una temperatura fissata a 2000. L'algoritmo è stato testato con differenti valori del tasso di raffreddamento. Più la temperatura è alta, più c'è la probabilità di accettare una sostituzione peggiorativa. Con differente tasso di raffreddamento, abbiamo testato differenti velocità di discesa della temperatura. I valori del tasso testati sono:

$$0.02, 0.015, 0.01, 0.005, 0.002$$

dove, ad esempio, un tasso di raffreddamento uguale a 0.002, indica che la temperatura, dopo un iterazione, viene ridotta del $0,2\%$, ovvero è uguale al 99.8% di se stessa.

La funzione di valutazione in questo caso è differente perché si basa su una probabilità dipendente dalla temperatura attuale. La sostituzione si effettua se si verificano entrambe le seguenti condizioni:

- il nuovo costo non sfora il budget;
- si verifica una delle seguenti condizioni:
  - l'overal medio della squadra migliora;
  - si verifica l'evento con probabilità dipendente dalla temperatura attuale.

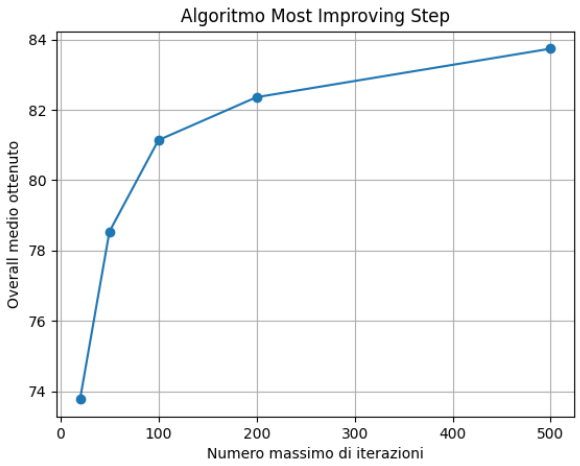


I risultati calano all'aumentare del tasso di raffreddamento, ovvero all'aumentare della velocità di discesa della temperatura. I valori ottenuti sono quelli aspettati. Infatti, con una velocità di raffreddamento maggiore, vengono effettuate meno sostituzioni perché si raggiunge più in fretta la temperatura 0. Le sostituzioni, tra l'altro, possono anche essere peggiorative. Otteniamo dei risultati più bassi rispetto ai due metodi precedenti (il risultato massimo è di circa 80.7).

3.5. Most Improving Step

Con questo algoritmo abbiamo la possibilità di valutare più sostituzioni ad ogni iterazione dal quale poter scegliere la migliore, ovvero quella che apporta un miglioramento maggiore nell'overall medio della squadra. Per implementare tale algoritmo abbiamo deciso di considerare una nuova formazione attraverso un nuovo *random restart*, dal quale poter scegliere il calciatore che, messo nella squadra di partenza nella relativa posizione, apporta il maggior miglioramento.

L'algoritmo è stato testato sui diversi valori di *max iteration* utilizzati nell'Hill Climbing e Tabu Search. Anche la funzione di valutazione funziona allo stesso modo dei due algoritmi.



Il Most Improving Step ha portato dei risultati simili all'Hill Climbing e al Tabu Search.

3.6. Algoritmo genetico

L'algoritmo genetico è un metodo che valuta una popolazione di individui (formazioni, in questo caso) contemporaneamente in modo da analizzare diverse regioni dello spazio delle soluzioni. La popolazione testata ha un numero di individui fissato a 50. Inoltre, è stato testato con diversi numeri di generazioni, dove ogni generazione corrisponde ad un'iterazione dove la popolazione si evolve. I valori testati sono i seguenti:

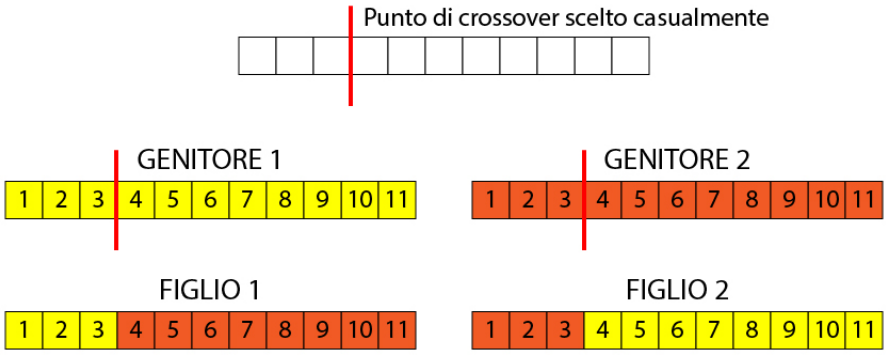
$$20, 50, 100, 200, 500$$

Per ogni nuova generazione, l'algoritmo seleziona due individui genitore con il metodo della *roulette wheel selection*. Essi vengono selezionati in base ad una probabilità dipendente dal loro *fitness*. Come fitness abbiamo considerato l'overall medio della formazione e, in caso di budget sforato, il fitness è uguale a 0.

A partire dai due genitori selezionati, si crea la generazione successiva. Fin quando non si raggiunge il numero di individui della popolazione prefissato, l'algoritmo effettua due procedure metafore della genetica: il crossover e la mutazione.

3.6.1 Crossover

Il metodo di crossover adottato è il *crossover a un punto*. Si seleziona casualmente un punto della formazione in cui applicare la suddivisione e, dati i due genitori, si creano due figli dove vengono mescolate le quattro mezze formazioni.



Il crossover viene accettato solo nel caso in cui le due formazioni figlie ottenute non presentano al loro interno dei calciatori duplicati. Alcuni moduli hanno dei ruoli duplicati (ad esempio, i moduli con difesa a quattro hanno due difensori centrali), dunque potrebbero crearsi formazioni con calciatori ripetuti più di una volta. Il numero massimo di crossover falliti è fissato a 5. Se si raggiunge il numero massimo di fallimenti, il crossover restituisce i genitori stessi.

3.6.2 Mutazione

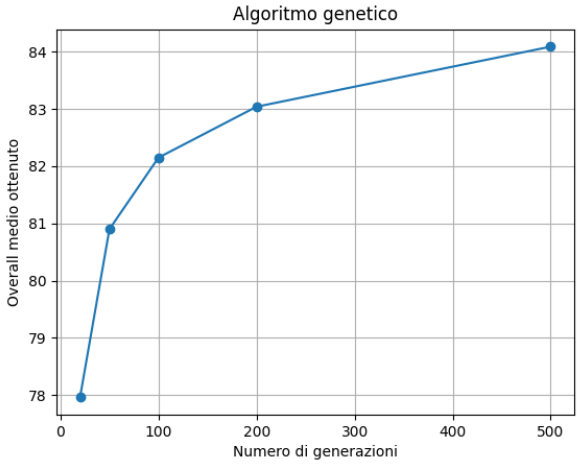
Una volta ottenuti due nuovi individui dal crossover, viene applicata agli stessi la procedura di mutazione. Come mutazione abbiamo utilizzato la sostituzione del calciatore con overall minore dell'individuo. Il calciatore nuovo viene accettato solo se si verifica una delle seguenti due condizioni:

- l'overall migliora;
- il nuovo costo è minore e viene tollerata una diminuzione dell'overall del 3%.

Come per il crossover, anche per la mutazione abbiamo previsto un numero massimo di tentativi falliti fissato a 5.

Una volta effettuato il numero di evoluzioni fissato, si restituisce il miglior individuo appartenente all'ultima generazione.

I risultati del test dell'algoritmo sono riassunti nell'immagine seguente:

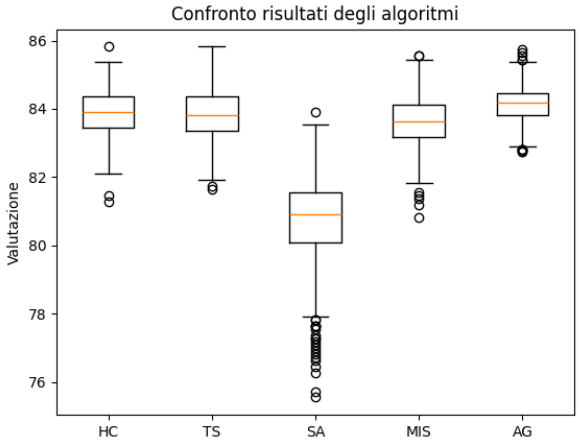


Con l'algoritmo genetico abbiamo ottenuto i risultati migliori con un overall medio di circa 84.2 con un numero di 500 generazioni.

3.7. Considerazioni finali

Come ulteriore prova dell'efficacia dei metodi, abbiamo elaborato un grafico di tipo *box plot* per analizzare la distribuzione statistica dei differenti risultati ottenuti. Nel grafico sono presenti i seguenti elementi:

- box: rappresenta il 50% centrale dei dati. La parte inferiore della scatola indica il 25% inferiore dei dati, mentre la parte superiore indica il 25% superiore;
- linea mediana: rappresenta il valore mediano della distribuzione;
- baffi: sono linee che si estendono dalla scatola fino ai valori più estremi che rientrano ancora entro 1,5 volte l'ampiezza della scatola;
- punti: i punti al di fuori dei baffi che indicano valori estremi che potrebbero essere considerati atipici.



Conclusione: dal grafico si evince che la migliore media è quella dell'algoritmo genetico, anche se di poco confronto all'Hill Climbing, al Tabu Search e al Most Improving Step. L'algoritmo genetico è inoltre il metodo che ha un'intervallo di distribuzione minore, data la minor ampiezza del box e dei baffi. Al contrario, l'algoritmo con ampiezza maggiore è il Simulated Annealing, che presenta anche la media peggiore.