

Progetto ICON:

**Predizione andamento calciatori in base alle statistiche di una stagione**

Gruppo di lavoro

* Emanuele Tanzi, 737530, [e.tanzi2@studenti.uniba.it](mailto:e.tanzi2@studenti.uniba.it)

Repository GitHub: *https://github.com/ematanzi/player\_trends*

AA 2022-2023

Introduzione

Determinare quale potrebbe essere l’andamento nelle prestazioni di un calciatore è un compito difficile, specialmente in quei casi in cui le informazioni a disposizione, ossia le statistiche inerenti alle sue prestazioni, sono limitate ad un determinato intervallo di tempo, oppure si hanno informazioni poco consistenti perché, ad esempio, ha giocato in campionati diversi o in categorie differenti.

Per questo motivo, professionisti del settore potrebbero impiegare un supporto che li aiuti a discriminare calciatori in base al loro possibile miglioramento o peggioramento.

L’obiettivo di questo progetto è duplice:

1. In primo luogo, si vuole costruire da zero un sistema che consenta di categorizzare l’andamento dei calciatori, considerandone i risultati sportivi, dati anagrafici e statistiche di due stagioni consecutive.
2. Successivamente, si vuole individuare, a partire dai risultati ottenuti nel punto precedente, un sistema che consenta di predire l’andamento di un giocatore limitandosi ad osservare i dati relativi ad una singola stagione.

Il caso di studio affrontato si concentra sull’analisi delle statistiche di calciatori nelle stagioni 2021/22 e 2022/23. I dati utilizzati sono stati ricavati a partire da dataset disponibili sulla piattaforma *Kaggle*, accessibili ai seguenti link:

* <https://www.kaggle.com/datasets/vivovinco/20212022-football-player-stats>
* <https://www.kaggle.com/datasets/vivovinco/20222023-football-player-stats>
* <https://www.kaggle.com/datasets/vivovinco/20212022-football-team-stats>
* <https://www.kaggle.com/datasets/vivovinco/20222023-football-team-stats>

Prima che all’individuazione di soluzioni ottimali per la risoluzione di questi compiti, l’obiettivo che ha alimentato le mie decisioni progettuali è stato l’apprendimento di soluzioni adatte a Sistemi basati su Conoscenza studiate durante il corso di studio.

Premetto che il task che mi sono assegnato è un compito particolarmente difficile perché implica la predizione di eventi futuri, che in particolar modo in questo ambito possono essere correlati a numerosissimi fattori (infortuni, problemi extra-campo, ecc.). Per questo motivo, i risultati ottenuti non sono dei risultati incredibilmente accurati ma ciò mi ha permesso di fare confronti interessanti tra le varie tecnologie utilizzate.

Per soddisfare gli obiettivi del progetto:

1. ho unificato i dati a mia disposizione, prendendo in considerazione solo le feature da me ritenute più importanti, e ho ridotto il dataset agli esempi consistenti (non privi di alcun dato utile, con statistiche significative per cercare di effettuare una predizione);
2. ho creato una KB utilizzando il linguaggio ***Prolog***, che mi permettesse di inferire l’andamento di un giocatore osservando i dati delle due stagioni consecutive;
3. ho utilizzato diverse tecniche di *Machine Learning*, tra le più adatte alla risoluzione di un task di ***classificazione multi-classe***, e ne ho confrontato i risultati.

Sommario

[Introduzione 2](#_Toc145491491)

[Elenco argomenti di interesse 4](#_Toc145491492)

[Creazione e manipolazione del dataset 6](#_Toc145491493)

[Features 6](#_Toc145491494)

[Creazione e impiego di una KB per l’individuazione dei trend 8](#_Toc145491495)

[Strumenti utilizzati 8](#_Toc145491496)

[Decisioni di Progetto 8](#_Toc145491497)

[Valutazione 9](#_Toc145491498)

[Case Based Reasoning per la classificazione: K-NN 14](#_Toc145491499)

[Descrizione e motivazione scelta 14](#_Toc145491500)

[Strumenti utilizzati 14](#_Toc145491501)

[Decisioni di progetto 14](#_Toc145491502)

[Primo esperimento 14](#_Toc145491503)

[Esperimenti successivi: feature selection 16](#_Toc145491504)

[Apprendimento supervisionato per la classificazione 18](#_Toc145491505)

[Strumenti utilizzati 19](#_Toc145491506)

[Cross Validation 19](#_Toc145491507)

[Grid Search 19](#_Toc145491508)

[Random Forest 20](#_Toc145491509)

[Descrizione 20](#_Toc145491510)

[Motivazione scelta 20](#_Toc145491511)

[Primo esperimento 21](#_Toc145491512)

[Esperimenti successivi: Random Undersampling 22](#_Toc145491513)

[Esperimenti successivi: esplorazione iperparametri 23](#_Toc145491514)

[Esperimenti successivi: feature selection 24](#_Toc145491515)

[Support Vector Machine 25](#_Toc145491516)

[Descrizione 25](#_Toc145491517)

[Motivazione scelta 26](#_Toc145491518)

[Primo esperimento 26](#_Toc145491519)

[Secondo esperimento: undersampling 26](#_Toc145491520)

[Terzo esperimento: feature selection 27](#_Toc145491521)

[Gradient Boosting Machine 27](#_Toc145491522)

[Descrizione 27](#_Toc145491523)

[Motivazione scelta 27](#_Toc145491524)

[Primo esperimento 27](#_Toc145491525)

[Secondo esperimento: undersampling 29](#_Toc145491526)

[Terzo esperimento: feature selection 30](#_Toc145491527)

[Apprendimento probabilistico per la classificazione 31](#_Toc145491528)

[Strumenti utilizzati 31](#_Toc145491529)

[Naive Bayes 31](#_Toc145491530)

[Descrizione 31](#_Toc145491531)

[Motivazione scelta 32](#_Toc145491532)

[Primo esperimento 32](#_Toc145491533)

[Altri esperimenti 33](#_Toc145491534)

[Rete Bayesiana 34](#_Toc145491535)

[Descrizione 34](#_Toc145491536)

[Motivazione scelta 34](#_Toc145491537)

[Primo esperimento 34](#_Toc145491538)

[Secondo esperimento 35](#_Toc145491539)

[Conclusioni 37](#_Toc145491540)

[Risultati ottenuti 37](#_Toc145491541)

[Possibili sviluppi futuri 38](#_Toc145491542)

Elenco argomenti di interesse

Gli argomenti del programma coperti da questo progetto sono i seguenti:

* Rappresentazione e ragionamento Relazionale
  + Clausole di Horn (Prolog)
* Apprendimento supervisionato
  + SVM
  + Random Forest
  + Gradient Boosting
  + Case Based Reasoning
    - K-NN
* Apprendimento probabilistico
  + Naive Bayes
  + Reti Bayesiane

# Creazione e manipolazione del dataset

Come già espresso nell’introduzione, il dataset è stato creato a partire da dati disponibili su Kaggle.

Nello specifico, ho preso in considerazione informazioni ricavate da quattro dataset, i quali contengono statistiche riguardanti i top 5 campionati europei (Premier League, Serie A, LaLiga, League 1 e Bundesliga). Rispettivamente, ogni dataset contiene:

* statistiche riguardanti i giocatori per la stagione 2021/22;
* statistiche riguardanti i giocatori per la stagione 2022/23;
* statistiche riguardanti i team per la stagione 2021/22;
* statistiche riguardanti i team per la stagione 2022/23.

Il primo passo compiuto è stato unificare i quattro dataset in un unico file. L’idea è di avere a disposizione (almeno per la prima parte del progetto) contemporaneamente dei dati di entrambe le stagioni. Ho preso in considerazione anche i dati della squadra in quanto ho ritenuto un elemento valido da considerare anche le prestazioni del team di appartenenza per ogni giocatore. Per questo motivo, ad ogni giocatore ho associato anche le statistiche della squadra in cui ha militato.

Il passo immediatamente successivo è consistito nell’eliminazione di tutti quei campioni con informazioni rumorose o superflue. Nello specifico, sono stati rimossi:

* i portieri, in quanto non ho ritenuto sufficienti le informazioni a disposizione per poter determinare un andamento;
* giocatori che ricoprono molteplici ruoli nel campo. La definizione di clausole per casi particolarmente specifici sarebbe risultata complessa, andando a complicare la KB e potenzialmente anche i modelli di apprendimento automatico;
* giocatori che hanno giocato meno di una determinata soglia di minuti (nello specifico, ho fissato tale soglia a 180 minuti, all’incirca due partite);
* giocatori i quali, nella tabella, risultavano avere dei valori non assegnati.

## Features

I dataset di partenza contenevano una grande quantità di feature per ogni dato (124 per i giocatori, 20 per le squadre). Ho limitato le feature a disposizione, considerando solo le seguenti:

* Age: età del giocatore;
* Pos: ruolo ricoperto in campo dal giocatore;
* 90s: numero di minuti giocati diviso 90 (numero di minuti regolari di una partita);
* Goals: numero di gol segnati dal giocatore;
* Assists: numero di assist forniti dal giocatore;
* GCA: numero di azioni da gol a partita a cui il giocatore ha partecipato;
* SCA: numero di azioni che hanno portato a conclusione a partita a cui il giocatore ha partecipato;
* PasTotCmp%: percentuale di passaggi compiuti sul totale di passaggi effettuati dal giocatore;
* TklWon: percentuale di tackle vinti a partita;
* Int: quantità di palle intercettate a partita;
* Err: quantità di errori che ha portato l’avversario alla conclusione a partita
* GF: gol fatti dalla propria squadra;
* GA: gol subiti dalla propria squadra;
* Pts/G: numero di punti per partita conquistati dalla propria squadra.

Questo insieme di dati è il punto di partenza per l’applicazione della base di conoscenza. Nella fase successiva, quindi, si determinerà la classe di appartenenza di ogni giocatore confrontando le statistiche della prima e della seconda stagione.

# Creazione e impiego di una KB per l’individuazione dei trend

L’impiego della KB è un punto di fondamentale importanza. Infatti, la base di conoscenza consente di suddividere gli esempi in classi. Si è deciso, per categorizzare gli atleti in base ai propri andamenti, di considerare le seguenti cinque classi:

* *very\_bad;*
* *bad;*
* *static;*
* *good;*
* *very\_good;*

il loro significato è abbastanza esplicito.

La creazione di una KB si è resa necessaria in quanto non esiste una KB di questo tipo, realizzata da una organizzazione e riconosciuta globalmente, da poter utilizzare.

## Strumenti utilizzati

Per la creazione e l’interrogazione della base di conoscenza, ho utilizzato il linguaggio relazionale ***Prolog***. L’interazione tra i file Prolog contenenti la KB e il notebook è stata gestita tramite la libreria di *Python* ***pyswip***.

La scelta è ricaduta su *Prolog* per le seguenti motivazioni:

* il linguaggio *Prolog* è particolarmente espressivo nella rappresentazione di relazioni complesse e regole logiche. Ciò mi ha consentito di definire relazioni tra gli esempi e le classi di destinazione in maniera piuttosto semplice.
* Ho supposto di dover creare delle regole anche piuttosto complesse, contenendo più condizioni e criteri. Il vantaggio fondamentale nell’uso di Prolog, quindi, è dovuto alla possibilità di esplicitare *clausole di Horn*.
* L’utilizzo di *Python* e *pyswip* mi ha consentito di automatizzare l’interrogazione alla base di conoscenza, formulando un’unica query valida per ogni calciatore.
* In generale, la sintassi di *Prolog* è molto simile a quanto visto nella teoria (*rappresentazione e ragionamento relazionale*), e per questo motivo il tempo di apprendimento di tale linguaggio è stato ridotto.

## Decisioni di Progetto

Le decisioni prese in questa fase di progetto sono essenzialmente le seguenti:

* siccome, ruolo per ruolo, le caratteristiche da considerare per un giocatore per valutare il suo andamento variano considerevolmente, ho deciso di realizzare tre file *Prolog*, uno per ruolo, in modo tale da rendere più semplice:
  + la leggibilità della base di conoscenza;
  + la realizzazione della base di conoscenza, concentrandomi in maniera separata nell’individuazione delle caratteristiche che meglio partizionano l’insieme di partenza per ogni ruolo.
* Per ogni ruolo, quali dati prendere in considerazione per la valutazione dell’andamento.

Ne sono risultati tre file Prolog separati (*trend\_df.pl*, *trend\_mf.pl*, *trend\_att.pl*). La struttura dei tre file è analoga, nonostante le differenze tra loro dovute alle caratteristiche diverse da analizzare.

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Nell’immagine, è riportata una componente significativa della prima versione del file *trend\_att.pl*.

A seguito dell’applicazione della base di conoscenza sulle tre partizioni, i risultati sono stati riuniti in un unico file. Tale file (*players\_trend.csv*) sarà l’input dei moduli di apprendimento automatico successivi.

## Valutazione

Il risultato dell’applicazione della KB è la creazione del nuovo dataset, composto dalle statistiche inerenti una singola stagione e il trend considerando le statistiche della stagione successiva del giocatore.

Non c’è un modo per valutare il funzionamento della base di conoscenza, in quanto non esistono a priori delle euristiche che consentano di determinare l’andamento di un giocatore. Per questo motivo, ho ritenuto valido valutare la correttezza della base di conoscenza in base a quanto la distribuzione delle classi per ogni ruolo si mostrasse di tipo gaussiano.

A seguito di numerosi esperimenti, ho progressivamente migliorato i risultati ottenuti in base a ciò che mi aspettavo di ottenere.

Immagine che contiene testo, schermata, diagramma, Diagramma

Descrizione generata automaticamenteImmagine che contiene testo, diagramma, schermata, Diagramma

Descrizione generata automaticamenteImmagine che contiene testo, schermata, diagramma, Diagramma

Descrizione generata automaticamente

Dal grafico si possono fare alcune osservazioni:

* La distribuzione dei difensori è come vicina a quanto ci si potesse aspettare, segue una distribuzione accomunabile ad una gaussiana. È comunque importante da sottolineare la sproporzione nelle cardinalità delle classi.
* La distribuzione dei centrocampisti dimostra come i valori associati alle classi “good” e “very\_good” siano quasi gli stessi.
* La distribuzione degli attaccanti mostra la possibilità di miglioramento nella definizione di giocatori con andamento pessimo rispetto a quelli che hanno andamento negativo.

Quindi, i miglioramenti che possono essere apportati sono i seguenti:

* Possiamo ridurre il numero di esempi corrispondenti alla classe “static” per i difensori
* Possiamo aumentare il numero di esempi corrispondenti alla classe “good” a discapito di esempi di classe “static” e “very\_good” per i centrocampisti
* Possiamo aumentare la frequenza di “very\_bad” a discapito di “bad” per gli attaccanti.

La popolosità della classe “static” non è un elemento eccessivamente disturbante al momento, visto che il mio obiettivo è ottenere delle classi la cui distribuzione approssimi una distribuzione gaussiana. Diversi sono gli altri due casi, dove le distribuzioni sono disturbate. Per questo motivo, ho effettuato una messa a punto delle clausole al fine di ottenere il risultato desiderato senza troppo modificare il significato delle clausole stesse. I risultati ottenuti sono i seguenti:Immagine che contiene testo, schermata, Rettangolo, diagramma

Descrizione generata automaticamente Immagine che contiene testo, schermata, diagramma, Diagramma

Descrizione generata automaticamente

Analizziamo di seguito le clausole create:

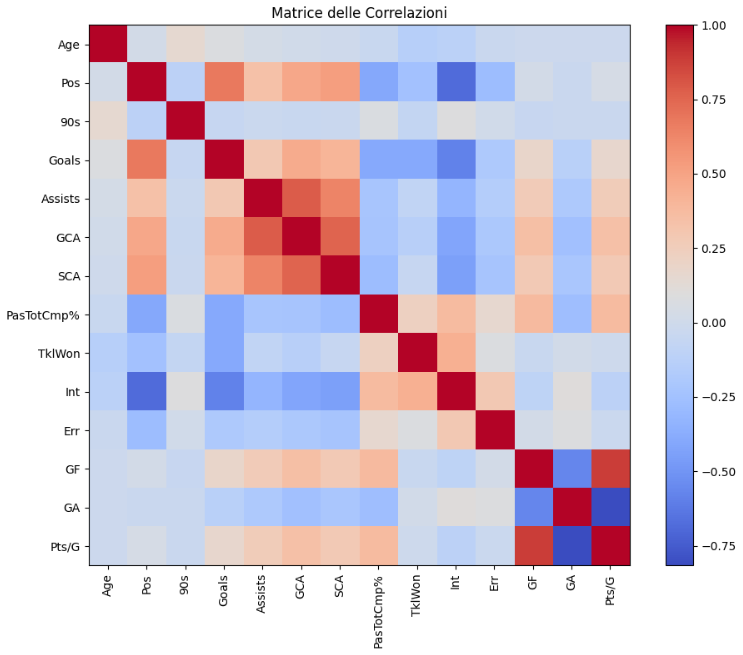
* un difensore è:
  + *very\_good* se:
    - rispetta un limite superiore di età (25 anni);
    - ha giocato un certo numero di minuti (10 partite);
    - ha aumentato il numero di tackle vinti rispetto alla stagione passata o; diminuito il numero di errori, e la propria squadra ha diminuito il numero di gol subiti.
  + *good* se:
    - rispetta un limite superiore di età (30 anni);
    - o è particolarmente giovane (22 anni), o ha giocato almeno 10 partite;
    - rispetta la condizione precedente (tackle, errori, gol subiti), oppure ha aumentato contrasti vinti, diminuito errori ed ha una età limitata superiormente (28 anni);
  + *bad* se:
    - non rispetta il limite superiore di età *very\_good*;
    - ha decrementato il numero di tackle vinti o di errori, e la sua squadra ha aumentato il numero di gol subiti.
  + *very\_bad* se:
    - non rispetta il limite superiore di età *good*;
    - rispetta la seconda condizione di *bad*.
* Un centrocampista è:
  + *very\_good* se:
    - rispetta un limite superiore di età (25 anni);
    - ha giocato un certo numero di minuti (10 partite);
    - ha aumentato la percentuale di passaggi compiuti e la percentuale di partecipazione a gol.
  + *good* se:
    - rispetta un limite superiore di età (30 anni);
    - o è particolarmente giovane (22 anni), o ha giocato almeno 10 partite;
    - rispetta la terza condizione di very\_good.
  + *bad* se:
    - non rispetta il limite superiore di età *very\_good*;
    - ha diminuito la percentuale di passaggi compiuti o la percentuale di partecipazione a gol e non rispetta un limite inferiore di età (28 anni)
  + *very\_bad* se:
    - non rispetta il limite superiore di età *good*;
    - ha diminuito la percentuale di passaggi compiuti ed ha diminiuto la partecipazione a gol.
* Un attaccante è:
  + *very\_good* se:
    - rispetta un limite superiore di età (25 anni);
    - ha giocato un certo numero di minuti (10 partite);
    - ha aumentato il numero di partecipazioni goal;
    - la propria squadra ha aumentato il numero di gol.
  + *good se*:
    - rispetta un limite superiore di età (30 anni);
    - o è particolarmente giovane (22 anni), o ha giocato almeno 10 partite;
    - o ha aumentato il numero di gol, avendo meno di 28 anni, o ha aumentato la propria partecipazione a gol e la squadra ha aumentato il numero di gol segnati.
  + *bad* se:
    - non rispetta il limite superiore di età *very\_good*;
    - ha diminuito il numero di gol o partecipazioni a gol oppure ha diminuito il numero di gol e partecipazione a gol
  + *very\_bad* se:
    - non rispetta il limite superiore di età *good*;
    - ha diminuito il numero di gol segnati e la partecipazione a gol oppure ha diminuito il numero di gol segnati o la partecipaizone a gol se la propria squadra ha aumentato il numero di gol.

Un giocatore rientra nella classe *static* quando non rientra in nessuna delle altre quattro categorie.

I simboli utilizzati sono riassunti nella seguente tabella:

|  |  |
| --- | --- |
| Simbolo | Significato |
| Player | Nome e Cognome del giocatore, serve per le query. |
| Age | Età. |
| Games | Minuti giocati diviso 90 (tempo regolamentare di una partita, in minuti). |
| Games2 | Minuti giocati diviso 90 della stagione seguente. |
| TklWon | Numero di tackle vinti per partita. |
| TklWon2 | Numero di tackle vinti per partita della stagione seguente. |
| Err | Numero di errori che hanno portato al tiro degli avversari a partita. |
| Err2 | Numero di errori che hanno portato al tipo degli avversari a partita della stagione seguente. |
| GA | Numero di gol subiti dalla propria squadra |
| GA2 | Numero di gol subiti dalla prorpia squadra della stagione successiva. |
| PTC | Percentuale di passaggi completati. |
| PTC2 | Percentuale di passaggi completati nella stagione seguente. |
| GCA | Partecipazione ad azioni da gol a partita. |
| GCA2 | Partecipazione ad azioni da gol a partita della stagione seguente. |
| GF | Numero di gol fatti dalla propria squadra. |
| GF2 | Numero di gol fatti dalla prorpia squadra nella stagione successiva. |
| Goals | Numero di gol segnati. |
| Goals2 | Numero di gol segnati nella stagione successiva. |

A valle dell calcolo delle classi, a seguito dell’ottenimento del file *players.data.csv* contenente il dataset da impiegare, prima di partire con il prossimo step, ho deciso di analizzare possibili correlazioni tra le feature del mio dataset.



Gli aspetti più evidenti sono i seguenti:

* gol e assist con GCA e SCA possono creare una ridondanza di informazioni;
* la posizione è fortemente correlata con le statistiche, come ci si poteva attendere;
* le statistiche riguardanti la squadra di appartenenza mostrano la forte correlazione che c’è tra gol fatti/subiti e la media punti.

In fase di addestramento e valutazione dei modelli di Machine Learning che implementerò, considererò anche la riduzione del dataset. Nello specifico, il dataset ridotto sarà composto da:

* *Age;*
* *Pos;*
* *90s;*
* *GCA;*
* *PasTotCmp%;*
* *TklWon;*
* *Int;*
* *Err;*
* *GF;*
* *GA;*

Una volta individuato l’andamento dei giocatori date le due stagioni consecutive 2021/22 e 2022/23, l’obiettivo dello studio è quello di riuscire a predire quale potrebbe essere il trend di un giocatore nella stagione successiva a partire dai soli dati relativi alla stagione corrente. Prenderemo in considerazione, quindi, unicamente i dati e le statistiche relative alla stagione 2021/22 dei giocatori dei quali è stato individuato il trend precedentemente.

# Case Based Reasoning per la classificazione: K-NN

## Descrizione e motivazione scelta

Una valida alternativa per la classificazione alla costruzione di un modello di apprendimento di una o più funzioni discriminanti è il ***k-Nearest Neighbor***, algoritmo che si basa sulla rappresentazione dei campioni in uno spazio multidimensionale (spazio a n dimensioni, dove n è il numero delle feature di input considerate), e ogni campione è un punto/vettore nello spazio.

Nella fase di addestramento, si individuano i vettori all’interno dello spazio multidimensionale rappresentanti gli esempi di training. Durante il test, un esempio del test set viene inserito all’interno dello spazio, e in base ad una misura di similarità, si calcolano i k vettori più vicini all’esempio di test (k numero naturale, può essere pari a 3, a 5, …), e all’interno di tale insieme di vettori, si valuta qual è la classe più frequente. Tale classe è quella che verrà assegnata all’esempio.

È evidente che, nonostante sia un modello estremamente semplice, sia concettualmente che dal punto di vista computazionale, può rivelarsi utile in un problema di classificazione multi-classe.

## Strumenti utilizzati

Si è fatto uso della libreria *Python* ***Scikit-learn*** per la sua implementazione. La libreria si è dimostrata utile non soltanto perché mette a disposizione il modello, ma anche perché consente di effettuare operazioni più complicate quali la ***Grid Search***, la ***k-fold Cross Validation*** e la *normalizzazione* delle feature.

* La normalizzazione delle feature è fondamentale nel caso del k-NN, in particolar modo quando si usano misure della distanza quali la ***distanza euclidea*** o la ***distanza di Manhattan***, le quali non sarebbero valide allo stesso modo se i parametri utilizzati per la creazione dello spazio avessero scale diverse.
* Tra gli esperimenti, sfrutterò anche la possibilità di modificare il peso attribuito ad ogni singola scala di valori, al fine di attribuire una maggiore importanza a determinate feature.

Sono state impiegate anche altre librerie di supporto quali *pandas* e *numpy*.

## Decisioni di progetto

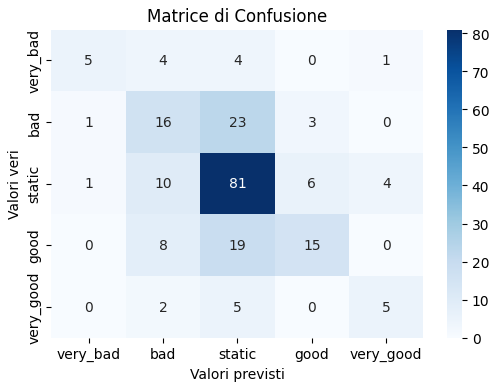
### Primo esperimento

Si è deciso di valutare i seguenti parametri tramite *Grid Search*:

* Numero di vicini da considerare, ossia il valore *k*.
  + La scelta di tale valore, deve essere effettuata in modo tale da evitare il più possibile parità, ma più è inferiore il valore di k e meno vicini possono essere presi in considerazione.
  + I valori di k valutati sono: [3, 5, 7, 9, 10];
* Attribuzione di pesi ai vicini:
  + Ho ritenuto valido valutare anche l’attribuzione di pesi dei k vicini in base alla distanza.
    - Un neighbor più lontano in base alla misura di similarità impiegata avrà un peso minore nella decisione della classe per un esempio.
* Metrica da utilizzare per il calcolo della distanza: ho valutato le seguenti tre metriche:
  + **misura di similarità L1**: anche nota come distanza di Manhattan, calcola la distanza tra due vettori sommando le differenze assolute tra le loro componenti;
  + **misura di similarità L2**: anche nota come distanza euclidea, calcola la distanza tra due vettori come la lunghezza del vettore differenza tra di essi;
  + **misura di similarità del coseno**: calcola la distanza considerando l’angolo individuato dai due vettori. Tale misura è indipendente dalla normalizzazione sulle feature.

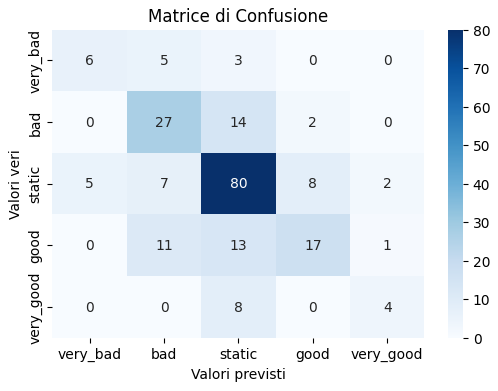
Per quanto riguarda la normalizzazione, si è deciso di confrontare il modello tramite i seguenti esperimenti:

* ad ogni feature viene assegnato lo stesso peso;
* alle feature “Pos” e “Age” viene assegnato un peso doppio rispetto alle altre feature;
* alle feature “Pos” e “Age” viene assegnato un peso triplo rispetto alle altre feature.

I risultati individuati sono i seguenti:

|  |  |
| --- | --- |
| Stesso peso | |
| Metriche | Valori |
| Precision | 0.572962 |
| Recall | 0.572770 |
| F1-score | 0.555896 |
| ROC-AUC | 0.778739 |
| Log Loss | 2.047611 |

|  |  |
| --- | --- |
| Peso doppio | |
| Metriche | Valori |
| Precision | 0.625871 |
| Recall | 0.629108 |
| F1-score | 0.617924 |
| ROC-AUC | 0.826245 |
| Log Loss | 2.255571 |



|  |  |
| --- | --- |
| Peso triplo | |
| Metriche | Valori |
| Precision | 0.635941 |
| Recall | 0.633803 |
| F1-score | 0.623035 |
| ROC-AUC | 0.842053 |
| Log Loss | 1.575286 |

Immagine che contiene testo, schermata, numero, diagramma

Descrizione generata automaticamenteIl secondo e il terzo esperimento hanno riportato miglioramenti rispetto al primo esperimento, ciò indica che l’attribuzione di un peso maggiore a quelle feature che, a priori, si sa essere più influenti si è rivelata una intuizione valida.

In entrambi i casi, gli iperparametri scelti sono stati:

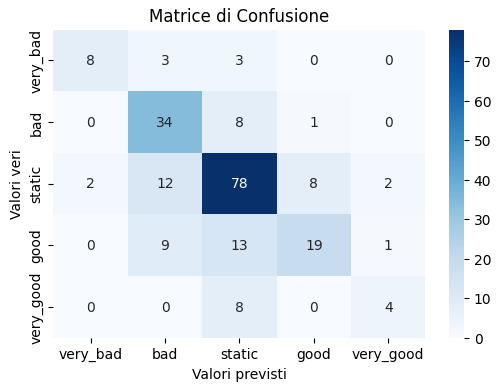
* misura di similarità L2;
* k = 10;
* attribuzione dei pesi ai vicini in base alla distanza.

### Esperimenti successivi: feature selection

Siccome questo algoritmo considera uno spazio n-dimensionale, dove n è il numero di feature di input, per semplificare il problema ho pensato di ridurre il numero di feature.

A seguito della riduzione delle feature di input, sono stati ottenuti i seguenti risultati:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Metriche | Precision | Recall | F1-score | ROC-AUC | Log Loss |
| Valori | 0.676485 | 0.671361 | 0.662878 | 0.82150 | 1.362436 |



I risultati sopra riportati corrispondono all’esperimento con feature ridotte a partire dal terzo test (quello con pesi di *Age* e *Pos* duplicati) ossia quello più promettente.

La selezione delle feature è stata effettuata impiegando conoscenza euristica sul dataset e analizzando la matrice di correlazione. Sono state considerate le seguenti feature:

* *Age;*
* *Pos;*
* *GCA;*
* *SCA;*
* *90s;*
* *PasTotCmp%;*
* *TklWon;*
* *GF;*
* *GA.*

I risultati ottenuti a seguito di quest’ultimo esperimento migliorano quelli ottenuti con il massimo numero di feature, come evidente anche dalle matrici di correlazione, che mostrano una maggior capacità di discriminazione corretta degli esempi.

# Apprendimento supervisionato per la classificazione

Si è deciso di adottare delle tecniche di apprendimento supervisionato per la risoluzione del task di classificazione multi-classe obiettivo di questa seconda parte dello studio.

I modelli che si è deciso di valutare per la risoluzione del compito sono i seguenti:

* ***Random forest***;
* ***Support Vector Machine***;
* ***Gradient Boosting Machine***.

La preferenza di modelli complessi a discapito di modelli più semplici non è casuale. In primo luogo, si è valutato il numero di classi della classificazione:

1. In un primo momento, si vuole provare la classificazione degli esempi in una delle 5 classi corrispondenti al dominio della feature target “Trend” (very\_bad, bad, static, good, very\_good).
2. Si vuole successivamente verificare le variazioni dei risultati dei modelli a seguito dell’ “appiattimento” del dominio della feature target “Trend”. Nello specifico, riduco il numero delle classi a tre, accorpando due coppie di classi (very\_good e good, very\_bad e bad).
   1. Questo secondo step, per limiti logistici, non è stato effettuato.

Di per sé, il task risulta essere particolarmente complesso, per cui modelli semplici potrebbero non risultare in grado di individuare relazioni complicate tra le statistiche che compongono le feature di input e le classi. I modelli compositi, invece, potrebbero offrire la flessibilità necessaria per catturare relazioni complesse tra variabili di input e classi target.

Inoltre, si vuole tenere in considerazione un numero di statistiche piuttosto elevato da poter essere correttamente gestite da un modello semplice (come, ad esempio, un decision tree). Le statistiche non solo sono molteplici, ma alcune di esse hanno domini totalmente differenti:

* Ad esempio, si considerino:
  + “Pos”, feature con dominio discreto di cardinalità 3 (per ogni ruolo di movimento ammissibile);
  + “Age”, feature con dominio corrispondente ai numeri naturali;
  + “GCA”, feature con dominio corrispondente ai numeri reali (corrispondente alle azioni da gol a partita del giocatore).

È evidente la differenza dei domini di queste variabili di input.

I modelli compositi, in generale, consentono di gestire una varierà di tipi di dati e di applicare trasformazioni specifiche delle caratteristiche in modo più efficace rispetto ai modelli semplici.

Modelli di questo tipo, inoltre, essendo più complessi diventano meno facilmente soggetti a problematiche quali sovradattamento del modello ai dati di training (*overfitting*).

Infine, modelli più complessi sono in grado di gestire meglio gli *outlier* (dati rumorosi) rispetto a modelli semplici.

## Strumenti utilizzati

Per l’implementazione dei modelli è stata impiegata la libreria *Scikit-Learn*. La libreria è stata impiegata per l’addestramento e il testing dei modelli, e quindi anche per l’implementazione di altre tecniche quali *Grid Search*.

Per l’implementazione del modello Gradient Boosting è stata impiegata la libreria Python ***XGBoost***. Tale modello verrà discusso più avanti.

Per ogni modello tra quelli analizzati nello studio:

1. si descrive brevemente il modello e le sue caratteristiche;
2. si esprime il motivo della scelta di tale modello;
3. si riporta il primo esperimento e i risultati conseguentemente ottenuti;
4. si riportano eventuali modifiche ai fini di ottenere un risultato migliore nella risoluzione del task e i conseguenti risultati ottenuti.

In generale, per ogni modello sono stati impiegati alcuni accorgimenti per cercare di ridurre il numero di test da effettuare, nella ricerca delle impostazioni e degli iperparametri migliori.

### Cross Validation

Un aspetto fondamentale da considerare è la possibilità che il modello vada incontro al *sovradattamento*: durante l’addestramento, il modello si adatta eccessivamente al training set, trovando regolarità apparenti negli esempi che potrebbero non sussistere nel mondo reale. La conseguenza è l’ottenimento di un modello che predice alla perfezione i dati di training, ma che ottiene prestazioni meno positive sui dati di test.

I rimedi all’overfitting sono essenzialmente tre: *pseudoconteggi*, *regolarizzazione* e *cross-validation*.

Il rimedio principalmente tenuto in considerazione per lo studio di questi modelli è la cross validation, e nello specifico è stata impiegata la *k-fold Cross Validation*.

L’idea di base è quella di validare l’addestramento effettuato sul training set tramite un sottoinsieme di quest’ultimo, che a tal proposito denominiamo “*validation set*”. Nella k-fold Cross Validation suddividiamo l’insieme di dati di addestramento in un numero k di fold (partizioni dell’insieme di stesse dimensioni). La validazione prevede k iterazioni di addestramento e conseguente verifica, e per ogni iterazione viene scelto uno dei fold, si esegue l’addestramento sui restanti e la validazione su quello scelto.

Impiego tale contromisura non solo per la sua funzione primaria, ma anche come uno degli strumenti per verificare se effettivamente si è verificato overfitting, confrontando i risultati ottenuti dalle iterazioni di addestramento sul training set con le predizioni finali sull’insieme di test.

### Grid Search

Tecnica utilizzata per individuare i migliori iperparametri per un modello.

* Gli iperparametri sono quei parametri che non vengono appresi dal modello, ma influenzano il comportamento e le prestazioni del modello.

Per ogni modello, creo una griglia di iperparametri da valutare. In fase di addestramento, si valuta la combinazione di iperparametri migliore in base ad una determinata metrica, sfruttando la *k-fold Cross Validation* precedentemente discussa.

Aspetto critico nello studio è la selezione di una metrica adatta al task da risolvere. La scelta della metrica è stata effettuata considerando i seguenti aspetti:

1. non c’è alcuna preferenza nella riduzione di errori “false-positive” piuttosto che di errori “false-negative”;
2. il dataset risulta essere sbilanciato inerentemente alle classi target (la distribuzione delle classi è una distribuzione gaussiana)

La scelta delle metriche da ottimizzare in determinate fasi quali la *k-fold Cross Validation* è quindi ricaduta su una delle seguenti metriche:

* ***Recall***: misura la quantità di campioni che sono stati etichettati correttamente sul numero totale di esempi che presentano effettivamente quella etichetta.
* ***F1-macro***: calcola l’F1-Score per ciascuna classe separatamente, per poi farne la media.
  + F1-Score: media armonica tra precision e recall, utile per trovare un equilibrio tra precision e recall
* ***Media tra precision, recall e F1-Score pesate***.
  + Precision: misura la quantità di campioni che sono stati etichettati correttamente sul numero totale di esempi che presentano effettivamente quell’etichetta.

Infine, si è deciso di adottare la *recall* come misura per la valutazione, in primo luogo per la sua semplicità (molto più semplice rispetto alla media di tre metriche differenti), e in secondo luogo in quanto nonostante la *F1-macro* sia specificamente adatta per la valutazione di classificazione multi-classe con dataset sbilanciati, il task è piuttosto difficile e concentrarsi su tale metrica finirebbe per peggiorare tutte le altre metriche, con conseguenti prestazioni pessime del sistema.

## Random Forest

### Descrizione

Il ***Random*** ***Forest*** è un modello *ensemble*, che si basa su ***Alberi di Decisione***. Un modello ensemble è un modello che fa uso di più modelli, ognuno dei quali viene addestrato e testato separatamente, e infine i risultati forniti dai vari modelli sono utilizzati per ottenere il risultato finale del modello *ensemble*.

L’algoritmo *Random Forest* fa unicamente uso di alberi di decisione (da ciò ne deriva il nome). Dal set di esempi viene scelto un sottoinsieme di feature per ogni albero, che successivamente viene addestrato e vengono ottenuti i risultati delle predizioni sull’insieme di test. Gli alberi di decisione ottenuti sono noti come “*alberi base*”.

La previsione di un modello di *Random Forest* è ottenuta dalla previsione di tutti gli alberi base che lo compongono. Il risultato è, in un caso analogo a quello di nostro interesse (classificazione), la classe con più “voti” ottenuti dagli alberi base.

### Motivazione scelta

Le motivazioni che hanno condotto alla scelta del *Random Forest* per questo task di classificazione multi-classe sono tendenzialmente quelle già espresse nel paragrafo introduttivo della sezione.

### Primo esperimento

In primo luogo, si è deciso di individuare gli iperparametri più adatti tramite *Grid Search*, Addestrando il modello utilizzando la tecnica di *k-fold Cross Validation* con *5 fold*.

I parametri da valutare tramite *Grid Search* sono i seguenti:

* **numero di stimatori** (il numero di alberi caratterizzanti la foresta);
* **profondità massima**;
* **numero minimo di esempi richiesti per suddividere un nodo interno**;
* **numero minimo di esempi richiesti per divenire un nodo foglia**.

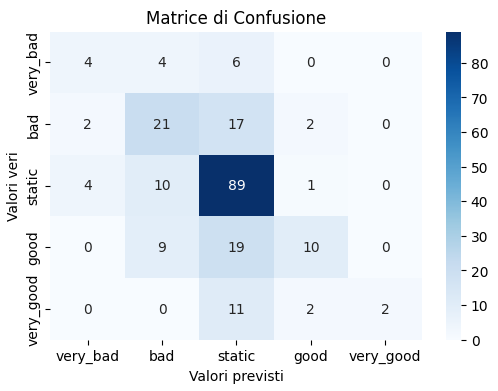
Inoltre, per equilibrare il più possibile training e test set per avere lo stesso numero di valori per classe, si è effettuata la stratificazione sul parametro target “*Trend*”. Nello specifico, tale tecnica consente di preservare la distribuzione dei dati del dataset originale quando suddivide il dataset in training e test set.

* La distribuzione è evidente osservando la matrice di confusione dei risultati.

A seguito del primo esperimento, sono stati ottenuti i seguenti risultati:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Metriche | Precision | Recall | F1-Score | ROC-AUC | Log Loss |
| Valori | 0.615784 | 0.591549 | 0.555396 | 0.811867 | 0.919111 |

e la matrice di confusione corrispondente è la seguente:



La *grid search* ha portato alla selezione dei seguenti iperparametri:

* numero di stimatori = 60;
* profondità massima = 10;
* numero minimo di esempi richiesti per suddividere un nodo interno = 2;
* numero minimo di esempi richiesti per divenire un nodo foglia = 2.

I risultati evidenziano diverse criticità:

* I valori di precision e recall non so particolarmente positivi.
* F1-Score significativamente inferiore rispetto a precision e recall, segno dello sbilanciamento del dataset.
* La curva ROC-AUC mostra valori molto positivi rispetto agli altri, indicando una buona capacità di discriminazione del modello
* La media dei valori per la recall ottenuti ad ogni iterazione della Cross Validation è pari a 0.61, mentre la deviazione standard è pari a 0.02. Tali valori sono piuttosto in linea con la recall ottenuta dal test set, indicando che, nel caso in cui si fosse verificato overfitting, questo sia minimo.
* Poche predizioni per le due classi good e very\_good. Il modello tende a classificare gli esempi come “static” (142 predizioni static, con il test set presentante 89 esempi nella classe static)

### Esperimenti successivi: Random Undersampling

Una delle principali problematiche è lo sbilanciamento delle classi nel training e nel test set. Tale sbilanciamento può essere corretto tramite tecniche di campionamento di dati quali ***undersampling*** e ***oversampling***.

L’idea dell’*oversampling* è difficile da praticare in questo caso, in quanto significherebbe introdurre nel dataset nuovi esempi, creandoli. L’idea non mi piace in quanto voglio considerare unicamente statistiche di giocatori reali, e introducendo nuovi esempi artefatti potrei andare a falsificare lo studio.

L’*undersampling*, al contrario, sembra molto più praticabile, nonostante andrebbe a ridurre il numero di campioni impiegati per l’addestramento (e di conseguenza anche per il test), in quanto comporterebbe unicamente un miglioramento delle proporzioni delle classi nel dataset.

Quindi, addestro una *Random Forest* in maniera analoga al caso precedente e verifico se è osservabile un miglioramento o meno, o addirittura un peggioramento.

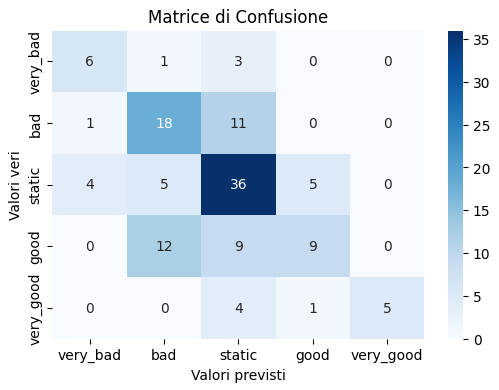
Dopo aver condotto un insieme di risultati sperimentando varie cardinalità per le singole classi, senza riportare tutti i risultati ottenuti, si osserva che:

* Al diminuire della dimensione delle classi, aumenta di conseguenza anche l’overfitting, evidenziato dalla differenza tra il comportamento del modello sul training set in fase di cross validation e il comportamento su test set.
* L’overfitting viene evitato mantenendo la distribuzione delle classi una distribuzione di tipo gaussiano, come nel caso presentato di seguito.

I risultati presenti nella tabella seguente sono stati ricavati a seguito dell’undersampling casuale con la seguente impostazione di cardinalità delle classi:

* |very\_good| = |very\_bad| = 48
* |good| = |bad| = 150
* |static| = 250

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Metriche | Precision | Recall | F1-Score | ROC-AUC | Log Loss |
| Valori | 0.592507 | 0.569231 | 0.558485 | 0.801660 | 1.001691 |



In questo caso specifico, è stato registrato un valore di recall medio nella *k-fold cross validation* di 0.62, con una deviazione standard di 0.01. I risultati ci portano a pensare che questa configurazione generi overfitting.

### Esperimenti successivi: esplorazione iperparametri

Nell’esplorazione degli iperparametri corretti, ho deciso di impiegare l’**ottimizzazione Bayesiana** per migliorare la Grid Search. Ho ritenuto valido questo approccio in quanto l’algoritmo *Random Forest* è caratterizzato da un numero piuttosto elevato di iperparametri.

Nello specifico, prende in considerazione le prestazioni passate del modello su diverse combinazioni di iperparametri per selezionare le prossime combinazioni da valutare.

Definisco quindi uno spazio di ricerca per ogni iperparametro del modello, ossia gli intervalli nei quali effettuare la ricerca. L’ottimizzazione bayesiana cerca di massimizzare o minimizzare una funzione obiettivo, stima della prestazione del modello in base agli iperparametri. Ad ogni iterazione, l’ottimizzazione bayesiana utilizza il modello probabilistico per suggerire una nuova configurazione degli iperparametri da valutare.

Tale tipo di ricerca è stato condotto sia con il dataset originale che con il dataset undersampled:

* numero di stimatori = 50;
* profondità massima = 12;
* numero minimo di esempi richiesti per suddividere un nodo interno = 3;
* numero minimo di esempi richiesti per divenire un nodo foglia = 1.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Metriche | Precision | Recall | F1-Score | ROC-AUC | Log Loss |
| Valori | 0.657011 | 0.619718 | 0.580566 | 0.813326 | 0.942300 |

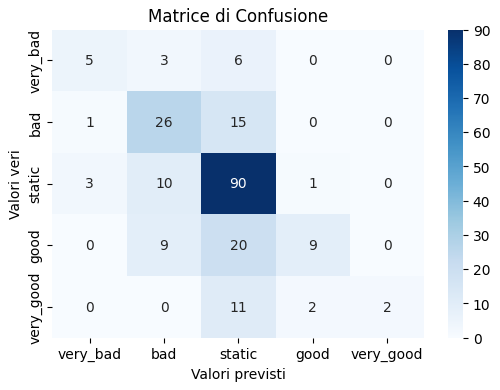
* numero di stimatori = 148;
* profondità massima = 20;
* numero minimo di esempi richiesti per suddividere un nodo interno = 7;
* numero minimo di esempi richiesti per divenire un nodo foglia = 1.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Metriche | Precision | Recall | F1-Score | ROC-AUC | Log Loss |
| Valori | 0.577110 | 0.553846 | 0.532679 | 0.806814 | 0.993668 |

L’utilizzo della ricerca con ottimizzazione bayesiana per l’individuazione dei parametri migliori ha portato ad un miglioramento, anche se minimo, delle prestazioni.

* Dal punto di vista computazionale, siccome il dataset non è eccessivamente grande, il tempo impiegato per l’esecuzione delle due prove è pressappoco lo stesso.

Anche la matrice di confusione riporta risultati molto simili, lievemente migliorativi.



### Esperimenti successivi: feature selection

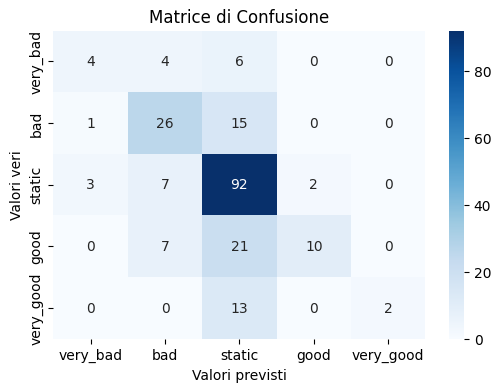
L’ultima valutazione che ho ritenuto valida effettuare è la *feature selection*, in quanto il modello presenta un gran numero di feature di input (estrapolate da un grande numero di statistiche associate ad ogni singolo giocatore), le quali potrebbero non essere ugualmente influenti per la classificazione.

* È stata mantenuta la ricerca dei parametri ottimali tramite ottimizzazione bayesiana.

Le feature selezionate sono quelle discusse nella sezione relativa alla creazione della KB. I risultati ricavati dall’esperimento sono i seguenti:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Metrica | Precision | Recall | F1-Score | ROC-AUC | Log Loss |
| Valore | 0.674053 | 0.629108 | 0.588989 | 0.826636 | 0.911488 |

I risultati ottenuti sono abbastanza in linea con i risultati relativi ossia senza selezione di feature. Non si osserva alcun particolare miglioramento o peggioramento.



Questo rappresenta l’esempio migliore ottenuto tramite questo modello. È comunque necessario sottolineare come la classe meglio classificata sia sempre la classe *static*.

## Support Vector Machine

### Descrizione

Il modello ***Support Vector Machine***, o ***SVM***, è un modello di apprendimento supervisionato impiegato sia per il task di classificazione che per la regressione.

Il modello si basa sulla rappresentazione del mondo in uno spazio multidimensionale (numero di dimensioni dipendente dalla quantità di feature). A partire da tale spazio (nel quale giacciono i vettori corrispondenti agli esempi), l’obiettivo del modello è quello di apprendere, a partire dai dati di addestramento, una funzione detta ***iperpiano***, in quanto è corrispondente al piano che separa lo spazio stesso in due sezioni distinte, in modo tale da dividere gli esempi di una classe dagli esempi di un’altra. La costruzione dell’iperpiano avviene cercando di massimizzare il ***margine***, ossia la distanza tra gli elementi più vicini di ciascuna classe e l’iperpiano stesso.

Per risolvere un problema di classificazione multi-classe, un approccio valido è l’***ovr*** (*One vs Rest*), che consiste nell’addestramento di un *SVM* per ogni classe (nel nostro caso, avremo cinque *SVM*). In fase di testing, ciascuna *SVM* restituisce una decisione, e la classe con il punteggio più alto diviene quella predetta.

Un altro approccio possibile è l’approccio ***ovo*** (*One vs One*), che consiste nell’addestramento di una SVM per ogni possibile coppia di classi (nel nostro caso, si addestreranno SVM su dieci coppie). In fase di test, ciascuna *SVM*, come nel caso precedente, per ogni esempio, restituisce una propria predizione, e la classe con più voti sarà quella assegnata a quell’esempio.

* L’approccio da me impiegato è *ovr*, non una vera e propria scelta in quanto non ho avuto modo di confrontare i risultati dell’altro approccio. Tale confronto potrebbe essere uno spunto per proseguire con la ricerca in questo dominio.

### Motivazione scelta

L’approccio alla risoluzione di un problema di classificazione multi-classe fornito da *SVM* è molto interessante, nonostante possa sembrare complesso, soprattutto per la sua capacità di gestire dati non linearmente separabili.

### Primo esperimento

Il primo esperimento è caratterizzato dalla ricerca dei parametri migliori, tramite uso di *Grid Search*. I parametri che valuto sono i seguenti:

* **Parametro di regolarizzazione *C***: controlla il trade-off tra la massimizzazione del margine e la minimizzazione dell’errore di classificazione. Più *C* aumenta, e più si adatta ai dati di addestramento, a costo di ottenere un margine più stretto.
* ***Kernel***: funzione per il mapping dei dati nello spazio. Valuteremo tra un *kernel polinomiale* e un *kernel RBF*, quest’ultimo particolarmente efficace in caso di dati non linearmente separabili.
* ***Gamma***: utile solo in caso di *kernel RBF*, controlla l’influenza di ciascun punto dati sui punti circostanti nello spazio ad alta dimensionalità creato dal kernel RBF. In breve, regola se tenere conto dei punti dati più lontani quando costruisce l’area di decisione. La regolarizzazione di questo parametro permette di ottenere il giusto equilibrio tra underfitting e overfitting.

I risultati ottenuti sono i seguenti:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Metriche | Precision | Recall | F1-Score | ROC-AUC | Log Loss |
| Valori | 0.630869 | 0.629108 | 0.623453 | 0.801664 | 0.978555 |

Immagine che contiene testo, schermata, numero, Carattere

Descrizione generata automaticamente

I risultati sono nella media, si nota come al solito una preferenza da parte dei modelli nell’etichettare nuovi esempi come *static*.

Confrontiamo i risultati appena ottenuti con due altri casi: *undersampling* e *riduzione delle feature*.

### Secondo esperimento: undersampling

L’applicazione dell’*undersampling* ha portato ad un peggioramento delle prestazioni del modello.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Metrica | Precision | Recall | F1-score | ROC-AUC | Log Loss |
| Valutazione | 0.278337 | 0.407692 | 0.307992 | 0.712699 | 1.161132 |

La riduzione del numero di esempi corrispondenti alla classe *static* ha portato ad un appiattimento della distribuzione gaussiana nei dati, che ha causato di conseguenza un aumento generale degli errori compiuti dal sistema.

### Terzo esperimento: feature selection

L’ultimo esperimento per quanto riguarda la *SVM* ha riguardato la riduzione delle feature analizzate. In base a quanto definito nella sezione relativa alla creazione della KB.

I risultati ottenuti a seguito dell’applicazione di un dataset ridotto sono i seguenti:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Metrica | Precision | Recall | F1-Score | ROC-AUC | Log Loss |
| Valutazione | 0.621359 | 0.610329 | 0.584948 | 0.785727 | 0.977836 |

Immagine che contiene testo, schermata, numero, Carattere

Descrizione generata automaticamente

In generale, i risultati migliori ottenuti sono quelli corrispondenti alla prima configurazione del modello. Nella valutazione finale, saranno quelli presi in considerazione.

## Gradient Boosting Machine

### Descrizione

Il ***Gradient Boosting Machine*** è un esempio di *ensemble learning* (come il *Random Forest*), in quanto combina i risultati ottenuti da modelli più semplici. Nello specifico si basa sulla costruzione di *alberi decisionali deboli*, addestrati e testati separatamente per fornire le proprie previsioni. Per ogni esempio, la previsione più votata è quella che viene selezionata.

### Motivazione scelta

Si è fatto uso della libreria ***XGBoost***, scelta in quanto generalmente fornisce prestazioni migliori rispetto ad altre implementazione dell’algoritmo di *Gradient Boosting*.

Questo algoritmo fa uso di alberi decisionali più profondi rispetto ad altri algoritmi di questo tipo.

### Primo esperimento

Similmente a quanto già fatto per gli altri modelli, selezioniamo i parametri tramite *grid search*. I parametri valutati sono i seguenti:

* **Profondità massima**: la profondità raggiungibile dagli alberi massima.
* **Tasso di apprendimento**: parametro molto interessante, in quanto consente di regolare la “velocità” di apprendimento dei dati durante il processo di addestramento. Tale parametro è un valore compreso tra [0, 1]. Più è basso, più il processo di addestramento è lento ma meno incline all’*overfitting*.
  + Nel nostro caso, è opportuno che tale sia più vicino a zero piuttosto che a uno, in quanto l’addestramento non impiega grandi quantità di tempo e così facendo otterremo dei risultati migliori.
* **Numero stimatori**: numero di alberi semplici considerati.

Il primo esperimento ha fornito risultati abbastanza positivi, se confrontati con i risultati finora ottenuti:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Metrica | Precision | Recall | F1-Score | ROC-AUC | Log Loss |
| Valutazione | 0.670513 | 0.680751 | 0.672231 | 0.877035 | 0.824785 |

Immagine che contiene testo, schermata, diagramma, numero

Descrizione generata automaticamente

Siccome il numero di parametri da regolare è abbastanza elevato (i parametri sono solo tre ma sono tutti parametri a dominio continuo), voglio impiegare la tecnica dell’*ottimizzazione bayesiana*, già applicata per il modello Random Forest per individuare i parametri ottimali.

I risultati ottenuti sono i seguenti:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Metrica | Precision | Recall | F1-Score | ROC-AUC | Log Loss |
| Valutazione | 0.710978 | 0.718310 | 0.702351 | 0.879594 | 0.947452 |

Immagine che contiene testo, schermata, diagramma, numero

Descrizione generata automaticamente

Le metriche Precision, Recall, e F1-score mostrano un miglioramento dei risultati. Dalla matrice di confusione si può osservare che il modello migliora la classificazione di esempi corrispondenti alla classe “*bad*”, mentre per le altre classi non si ha un netto miglioramento, e la distribuzione degli errori rimane pressoché invariata.

### Secondo esperimento: undersampling

Successivamente, si è ripetuto l’esperimento sull’insieme di dati sotto-campionato. I risultati ottenuti sono i seguenti:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Metrica | Precision | Recall | F1-Score | ROC-AUC | Log Loss |
| Valutazione | 0.577098 | 0.576923 | 0.568542 | 0.822680 | 0.969754 |

Immagine che contiene testo, schermata, numero, diagramma

Descrizione generata automaticamente

Risultati simili a ciò che ci si aspettava, osservando anche il comportamento degli altri modelli con undersampling.

### Terzo esperimento: feature selection

L’ultimo esperimento prevede l’addestramento e la conseguente valutazione del modello con impiego delle feature selezionate nel paragrafo precedente.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Metrica | Precision | Recall | F1-Score | ROC-AUC | Log Loss |
| Valutazione | 0.670293 | 0.680751 | 0.667329 | 0.859983 | 0.823820 |

Immagine che contiene testo, schermata, numero, diagramma

Descrizione generata automaticamente

Confrontando questi risultati con quelli precedentemente ottenuti, si osserva una minima variazione. Se le metriche di precision, recall e F1-Score risultano diminuite (quindi peggiorate), la log loss è invece migliorata (diminuisce), mostrando un miglioramento nella “sicurezza” delle assegnazioni effettuate.

# Apprendimento probabilistico per la classificazione

Come soluzione alternativa all’apprendimento supervisionato, è stato considerato l’**apprendimento probabilistico**, metodologia che si basa sulla teoria delle probabilità per modellare il comportamento e le relazioni tra variabili all’interno di un sistema.

A differenza dei modelli precedentemente analizzati, i modelli probabilistici offrono una maggiore flessibilità nel trattare l’incertezza e la variabilità nei dati. Tale caratteristica rende i modelli di questo tipo particolarmente adatti in casi in cui la conoscenza è incompleta o rumorosa.

Ho ritenuto adatto un approccio di questo tipo per la risoluzione del task prefissato per via dell’insieme di dati trattati, piuttosto rumorosi, considerando la necessità di considerare molteplici statistiche per ogni giocatore, che però non tutte per ogni giocatore hanno una stessa rilevanza, perché questa dipende dalla sua correlazione con la posizione stessa. Non solo, ma anche altri aspetti, come l’età o i minuti giocati, possono essere degli elementi da non escludere considerando le prestazioni in generale di un calciatore.

Inoltre, l’apprendimento probabilistico è ampiamente utilizzato nella risoluzione di compiti di classificazione multi-classe, poiché consente di modellare in modo naturale le probabilità delle diverse classi per ciascuna istanza di dati.

Gli approcci selezionati sono essenzialmente due:

* uso di un modello ***Naive Bayes***;
* creazione di ***reti bayesiane***.

Come nel paragrafo precedente, l’analisi delle tecniche seguirà degli step:

1. descrizione del modello utilizzato;
2. motivazione della scelta dell’uso di tale modello;
3. descrizione della prima configurazione e analisi dei risultati;
4. a partire dalle osservazioni dello step precedente ed altri aspetti, si effettuano modifiche al modello e si producono nuovi risultati.

## Strumenti utilizzati

Per il *Naive Bayes*, è stata utilizzata la libreria *Scikit-learn*, mettendo a disposizione molte funzioni per l’apprendimento automatico quali *Grid Search* e *Cross Validation*.

Per la realizzazione delle reti bayesiane prodotte e la loro interrogazione, è stata impiegata la libreria Python *pgmpy*, libreria che mi è sembrata più semplice e valida.

Sono state utilizzate anche altre librerie di supporto quali *pandas* e *numpy*.

## Naive Bayes

### Descrizione

Un classificatore ***Naive Bayes*** è un particolare tipo di **classificatore di Bayes**, che è un modello per la classificazione probabilistico. Questo modello, infatti, si basa sull’idea che la classe di un elemento influenza i valori delle feature di questo. Il suo obiettivo è quindi quello di imparare la dipendenza delle feature dalla classe, impiegando il **Teorema di Bayes**.

Il *Naive Bayes* è un modello semplificativo rispetto al classificatore bayesiano, in quanto assume che tutte le feature utilizzate per la classificazione siano indipendenti tra loro. Questa, come intuibile, è una assunzione drasticamente significativa, perché è impossibile non considerare che alcune delle feature a disposizione sono dipendenti in qualche modo tra loro. Questa semplificazione rende però il modello estremamente efficiente dal punto di vista computazionale.

* Quando si hanno a disposizione tutte le informazioni relative alle feature per ogni esempio, tale modello corrisponde al modello di *Regressione Logistica*.

Il modello Naive Bayes, per individuare la probabilità che una specifica istanza caratterizzata da k feature (, con i da 1 a k) abbia etichetta Y, sfrutta il calcolo della distribuzione a posteriori, impiegando il Teorema di Bayes, l’assunzione di indipendenza e la formula di probabilità condizionata.

### Motivazione scelta

La scelta del dell’utilizzo di un modello di questo tipo è stata considerata tenendo presente i seguenti aspetti:

* l’assunzione di indipendenza rende molto più semplice il calcolo delle probabilità condizionate, e ciò risulta particolarmente vantaggioso nel mio caso, avendo a che fare con un numero non particolarmente limitato di dati;
* nonostante l’assunzione di indipendenza condizionale, il modello Naive Bayes ha dimostrato di riuscire ad ottenere buone prestazioni in molte situazioni reali;
* il modello è noto per essere robusto in presenza di rumore nei dati;
* come già discusso in precedenza, la distribuzione delle classi del modello è di tipo gaussiano. Per questo motivo, scegliere il modello *Gaussian Naive Bayes* è sembrata una buona opzione.

### Primo esperimento

L’unico iperparametro da valutare è il parametro *var\_smoothing*, che regola gli pseudo-conteggi. Tramite *Grid Search* (effettuata tramite k-fold Cross Validation) si è cercato di individuare il valore maggiormente adatto.

* Valore risultante: *var\_smoothing* = 1e-05

A differenza dei casi precedenti, mi è sembrato superfluo valutare il funzionamento sul dataset senza riduzione delle feature. La valutazione è stata effettuata direttamente sul dataset ridotto in termini di feature.

* La riduzione delle feature è stata fatta allo specifico scopo di ridurre la correlazione tra le feature. Addestrare il *Naive Bayes* con dati di cui conosco con certezza la profonda correlazione non avrebbe avuto senso.

I risultati ottenuti dal primo esperimento sono i seguenti:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Metrica | Precision | Recall | F1-Score | ROC-AUC | Log Loss |
| Valutazione | 0.606628 | 0.521168 | 0.455372 | 0.710083 | 1.085166 |

Immagine che contiene testo, schermata, numero, diagramma

Descrizione generata automaticamente

I risultati ottenuti deludono le aspettative. Nello specifico, si è notato che:

* il modello quasi non effettua predizioni per la classe “*good*” e in generale effettua poche predizioni per classi diverse da “*static*”.
* La conseguenza è l’ottenimento in un valore positivo per la *precision*, meno positivo per la *recall*. Lo squilibrio comporta anche un valore basso per la *F1-Score*.
* La *log loss* ottenuta è tendenzialmente maggiore rispetto a quelle finora ottenute. Ciò vuol dire che questo modello assegna meno bene probabilità alle classi rispetto agli altri.

### Altri esperimenti

Sono state provate in seguito un insieme di soluzioni alternative, sempre nell’ambito del Naive Bayes, ma nessuna di queste ha migliorato i risultati precedentemente ottenuti (anzi, le alternative provate hanno unicamente peggiorato tali risultati).

Le alternative in questione provate sono:

* una ulteriore riduzione delle feature di input, evitando di considerare le feature che più delle altre comportavano dipendenze:
  + Pos;
  + GF, GS.
* Pesatura delle classi: ho provato ad attribuire un maggior peso alle classi meno rappresentare in modo tale da cercare di risolvere il problema di mancate classificazioni per la classe “good”.
* *Random undersampling*, cercando di aumentare la rappresentanza per le classi diverse da static a sfavore di quest’ultima (provando anche ad aumentare separatamente il numero di esempi per la classe *good*).

## Rete Bayesiana

### Descrizione

Una ***rete bayesiana*** è un modello probabilistico grafico che rappresenta le relazioni causali tra diverse variabili mediante un *grafo direzionato aciclico*.

Abbiamo accennato alla struttura di una rete bayesiana descrivendo il *Naive Bayes*. La struttura di una rete di questo genere è abbastanza semplice:

* i nodi rappresentano le feature che si vogliono considerare, quindi:
  + le feature di input;
  + la feature target.
* Gli archi rappresentano le dipendenze tra le variabili. Un arco che va da nodo A al nodo B indica che A influisce su B, e quindi B dipende da A. Questo permette di modellare come le variazioni in una variabile influiscano sulle altre.

Ogni nodo di variabile casuale è associato a una tabella delle probabilità condizionali. Queste tabelle rappresentano la probabilità che il nodo prenda un determinato valore dato il valore dei suoi genitori nel grafo.

### Motivazione scelta

La ragione principale per cui è stata presa questa scelta è la conoscenza relativa alle forti dipendenze tra le feature che descrivono un calciatore. Si è pensato che le reti bayesiane potessero essere in grado di modellare efficacemente le dipendenze.

Un’altra ragione da considerare è la conoscenza a priori inerente il dominio considerato: le informazioni relative alla correlazione tra variabili si possono ricavare da una matrice di correlazione ma le motivazioni della correlazione tra le variabili in questo dominio è facilmente intuibile.

Infine, la rappresentazione sotto forma di rete è semplice ed intuitiva, ed inoltre offre il modo di fare molteplici esperimenti cambiando gradualmente le caratteristiche del grafo stesso.

### Primo esperimento

La libreria utilizzata è *pgmpy*, libreria molto utile in quanto consente la modellazione del grafo orientato, il calcolo delle probabilità condizionate a partire dal grafo e l’interrogazione per ottenere l’inferenza sulle probabilità condizionali. Come libreria di supporto per il *ML* si è continuato ad impiegare *Scikit-learn*.

Indipendentemente da un singolo esperimento, le prove effettuate sono state organizzate come segue:

1. definizione delle variabili della rete bayesiana, sottoinsieme delle variabili del dataset;
2. discretizzazione dei valori:
   1. a ogni feature con dominio continuo è stato discretizzato il dominio, riducendolo a 5 classi, rappresentanti le partizioni di dimensioni uguali considerando l’insieme di dati ordinato in ordine decrescente;
3. suddivisione del dataset in esempi di training e testing;
4. creazione della struttura della rete;
5. creazione delle tabelle CPD in base ai dati di input;
6. inferenza e calcolo delle metriche di valutazione.

In primo luogo, si è provato con una struttura di questo tipo:

Immagine che contiene linea, cerchio, diagramma, schermata

Descrizione generata automaticamente

Si può notare la semplicità del grafo. La mia idea è quella che le feature qui selezionate siano quelle maggiormente collegate all’inferenza del trend.

I risultati prodotti, tuttavia, hanno dimostrato l’inadeguatezza di una soluzione di questo tipo:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Metrica | Precision | Recall | F1-Score | ROC-AUC | Log Loss |
| Valutazione | 0.299000 | 0.342723 | 0.303520 | 0.507602 | 1.941697 |

Immagine che contiene testo, schermata, numero, Carattere

Descrizione generata automaticamente

La motivazione principale del malfunzionamento della classificazione tramite questo modello è l’eccessiva semplificazione del dominio, riducendo i dati ad alcune sole statistiche. Ho ritenuto superfluo cercare di migliorare i risultati a partire da questa configurazione.

### Secondo esperimento

Nel secondo esperimento ho deciso di tenere in considerazione un maggior quantitativo di feature, prendendo in considerazione la matrice di correlazione già impiegata in altre istanze.

Purtroppo, non ho a disposizione sufficienti risorse (memoria RAM) per poter addestrare un modello facente uso di una rete bayesiana che tenga in considerazione di tutti i possibili elementi, quindi ho dovuto semplificare la rete al fine di riuscire ad eseguire il codice.

Immagine che contiene cerchio, linea, diagramma, schermata

Descrizione generata automaticamente

I risultati sono i seguenti:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Metrica | Precision | Recall | F1-Score | ROC-AUC | Log Loss |
| Valutazione | 0.305235 | 0.333333 | 0.315194 | 0.527977 | 1.517354 |

Immagine che contiene testo, schermata, numero, diagramma

Descrizione generata automaticamente

Sono comunque risultati peggiori rispetto a quelli che abbiamo generalmente ottenuto tramite gli altri modelli.

# Conclusioni

## Risultati ottenuti

Come previsto, il compito di previsione dell’andamento dei calciatori in base alle statistiche di una singola stagione si è rivelato parecchio complesso, e in generale non ha condotto a risultati particolarmente soddisfacenti.

Riassumiamo i risultati nella seguente tabella (per ogni tipologia di modello, riportati solo i risultati “migliori”):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Modello | Precision | Recall | F1-Score | ROC-AUC | Log Loss |
| K-NN | 0.647627 | 0.661972 | 0.579789 | 0.826383 | 1.733068 |
| SVM | 0.676485 | 0.671361 | 0.662878 | 0.82150 | 1.362436 |
| Random Forest | 0.674053 | 0.629108 | 0.588989 | 0.826636 | 0.911488 |
| Gradient Boosting | 0.710978 | 0.718310 | 0.702351 | 0.879594 | 0.947452 |
| Naive Bayes | 0.606628 | 0.521168 | 0.455372 | 0.710083 | 1.085166 |
| Rete Bayesiana | 0.305235 | 0.333333 | 0.315194 | 0.527977 | 1.517354 |

* ***Gradient Boosting*** ha mostrato risultati migliori:
  + 72% dei campioni appartenenti ad una classe classificati in tale classe;
  + 71% dei campioni classificato correttamente (29% di falsi positivi);
  + presenta il miglior bilanciamento tra precision e recall;
  + presenta la miglior capacità di discriminare tra classi.
* Il modello che assegna meglio le probabilità alle classi target per ciascun esempio nel dataset è il ***Random Forest***.
* Il modello ***SVM*** non ha restituito dei risultati ottimi, ma si è rivelato promettente.
* I modelli probabilistici hanno restituito dei risultati deludenti.

## Possibili sviluppi futuri

Siccome lo scopo è quello di fornire ai professionisti del settore un modo per aiutarli a monitorare gli atleti, ho affrontato il problema come classificazione multi-classe con un numero relativamente elevato di classi. Si potrebbe pensare, per futuri esperimenti, di ridurre il problema:

* riducendo il numero di classi a 3 (accorpando le classi positive/molto positive, negative/molto negative);
* riducendo il problema ad una classificazione binaria (giocatore in trend positivo / giocatore in trend negativo);
* separandolo, creando classificatori ciascuno per ogni singolo ruolo.

Inoltre, non esiste una base di conoscenza univoca che consenta una valutazione standard per i giocatori, e ciò mi ha spinto alla creazione di una in base a criteri personali accompagnati da osservazioni di tipo matematico.

* Si potrebbe pensare di rendere la KB più sofisticata, considerando altre informazioni (da aggiungere o sostituire quelle attualmente considerate),
  + ad esempio, la quotazione di mercato attuale.
* Si potrebbe pensare di semplificare la KB, semplificando di conseguenza anche la predizione successiva.