Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea Magistrale in Informatica

Curriculum in Informatica per il Management

INTELLIGENZA ARTIFICIALE A SUPPORTO DELLE DESIGNAZIONI ARBITRALI NELLO SPORT DELLA PALLACANESTRO

Relatore:
MARCO DI FELICE

Cadidato:
CRISTIAN ROMANELLO

Sessione III
Anno Accademico 2019/2020

Ai miei genitori e alla mia fidanzata...

Abstract

La Data Analytics al giorno d'oggi è applicata in diversi ambiti: sanitario, industriale, economico, sociale e molti altri ancora. Anche lo sport fa uso di questa disciplina, al fine di analizzare i propri giocatori, squadre e società. "Quali sono i più promettenti tra i giocatori emergenti?", "Quali squadre si qualificheranno ai playoff?", "Quale squadra vincerà il campionato?". Queste sono solo alcune delle domande che si pongono gli appassionati del proprio sport, e ancora di più le società stesse. La Data Analytics prova quindi a fornire alcune risposte. La pallacanestro è sul podio tra gli sport più praticati in Italia, e tra le sue categorie certamente la Serie A è la più seguita dal pubblico cestistico. Una componente che permette il corretto svolgimento di tutte le gare, assieme a giocatori, allenatori, società e staff, sono gli arbitri.

Questo elaborato propone uno studio su come fornire un supporto al designatore di tali arbitri, ovvero colui che li designa alle partite, in modo da svolgere più facilmente il suo importante compito. Gli arbitri designati devono essere all'altezza della partita che si disputerà: a tal fine è meglio conoscere preventivamente il possibile grado di difficoltà di quest'ultima. Ciò però potrebbe non bastare, poiché per designare ogni settimana 21 dei circa 35 arbitri di Serie A, è necessario anche saper gestire al meglio la loro rotazione tra le squadre, in modo che non vengano arbitrate dagli stessi arbitri troppo frequentemente al fine di prevenire l'insorgere di problemi. Molte sono le variabili in gioco, come il cercare di mantenere un equo ed omogeneo numero di designazioni all'interno del gruppo e mantenerlo nel corso dell'anno, rispettare le indisponibilità che si possono presentare durante la stagione e diversi altri fattori che verranno dettagliati nel corso di questa tesi.

Introduzione

L'analisi dei dati negli ultimi decenni è diventata un ambito in cui sono confluite moltissime energie e studi. Il costante progresso tecnologico permette la generazione di una grande mole di dati, e sempre più l'uomo ne è incuriosito dall'esplorazione, ma soprattutto anche all'estrapolazione di nuova informazione da essi. Gli ambiti di applicazione al giorno d'oggi hanno raggiunto molti settori, dalla finanza alla sicurezza, dalla biomedicina alle *smart cities*. La pervasività di questa nuova disciplina ha fatto sì che anche il settore sportivo si interessasse ad essa. In quasi tutti gli sport era già prevista una modalità di raccolta dei dati, riguardo ad esempio le azioni di gioco, i punteggi e le statistiche dei giocatori, ma in forma cartacea. Negli ultimi anni però questi processi si sono evoluti e digitalizzati, permettendo così di effettuare analisi più facilmente ed operare su quantità di dati sempre più elevate. Analogamente agli altri settori, anche quello dell'analisi dei dati non prescinde dall'ambito economico: è evidente come il numero di studi e applicazioni sia proporzionale alla diffusione di uno sport, e quindi anche alla sua capacità economica. In Italia lo sport più seguito, in termini di spettatori negli stadi, è chiaramente il calcio, seguito da Tennis, Rugby e Pallacanestro[24].

L'utilizzo principale della ***Data Analytics*** negli sport è rivolto all'analisi delle prestazioni dei giocatori, al *forecasting* (predizione) di: *performance* future, probabilità di vincita di una squadra, infortuni dei giocatori e posizioni del campo più performanti. Ad esempio, un interessante campo di applicazione è la scelta dei giocatori durante una gara da parte dell'allenatore, in base anche alle prestazioni degli avversari, oppure il calcolo del vantaggio/svantaggio di giocare in casa oppure

fuori. Le società, in costante aumento, che adottano l'utilizzo di tali sistemi, possono godere di vantaggi competitivi e strategici rispetto agli avversari. L'aumento di interesse da parte del mondo accademico e industriale farà certamente crescere ancora di molto questo settore.

Un altro importante aspetto in ambito sportivo-tecnologico è costituito dai problemi di **ottimizzazione**. Molteplici, infatti, sono i supporti che la tecnologia può offrire a questo settore, il quale vede la sua applicazione, ad esempio, nella scelta dei giocatori di una squadra, nello *scheduling* di campionati e gare (sia nella *regular season* che nei *playoff*), nelle designazioni arbitrali e, ancora, nell'ottimizzazione delle trasferte. Tutti questi aspetti infatti aiutano notevolmente le società a massimizzare i profitti e a ridurre i costi.

In questo elaborato si affrontano entrambi gli argomenti della *Data Analytics* e dell'ottimizzazione, nell'ambito della pallacanestro italiana; in particolare, per la predizione della difficoltà delle gare di Serie A nella stagione sportiva 2018/2019, focus principale di questo lavoro, e successivamente un sistema automatizzato di designazioni per i relativi direttori di gara. Quest'ultimo si può modellare come problema di ottimizzazione, ed è anche noto come *"Referee Assignment Problem"*. L'obiettivo è quello di unire questi due ambiti per fornire al designatore un supporto alle decisioni nella propria attività operativa.

Il primo capitolo, "Stato dell'arte", esporrà i principali temi dello Sport analytics in generale, per poi entrare nello specifico circa gli studi e i progetti preesistenti sulla previsione della difficoltà delle gare ed il "Referee Assignment Problem". Successivamente, il capitolo prosegue con cenni teorici sul Machine Learning e sugli algoritmi di ottimizzazione. Il secondo capitolo, denominato "Progettazione", dettaglia le specifiche del progetto quali gli obiettivi, l'architettura, la raccolta dei dati, l'analisi, l'etichettatura e, infine, i dataset prodotti. Il capitolo "Predizione della difficoltà delle gare" argomenta tutte le implementazioni realizzate: dagli algoritmi di Machine Learning supervisionati al progetto *"TLGProb"* ed ulteriori

sviluppi. Al termine dello stesso, verrà effettuata un'analisi dei risultati raggiunti per ogni *dataset* e alcune considerazioni.
Il capitolo sull'"Assegnamento automatico delle designazioni arbitrali" articola l'algoritmo di ottimizzazione realizzato per raggiungere l'obiettivo, approfondendo con un'analisi gli studi esistenti e, infine, i risultati ottenuti per l'esperimento condotto per la Serie A italiana di pallacanestro.
Il quinto e sesto capitolo, rispettivamente "Conclusioni" e "Sviluppi futuri", espongono alcune considerazioni personali sui risultati raggiunti per entrambe le principali componenti dell'elaborato, e successivi futuri miglioramenti da apportare a questo progetto.

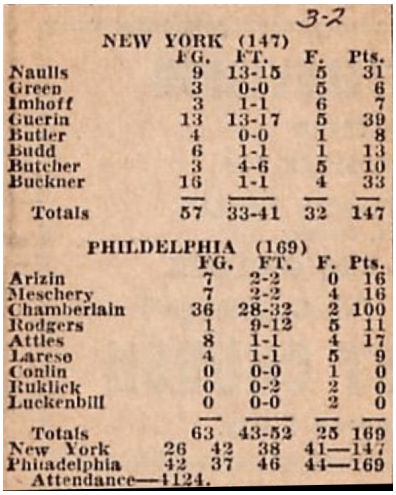
Indice

Capitolo 1

Stato dell'arte

1.1 Sport analytics

Il basket è uno sport di squadra nato nel 1891, a Springfield da James Naismith. Al giorno d'oggi l'analisi quantitativa si è dimostrata un potente strumento per le società e le squadre di basket, al fine di tenere traccia, analizzare e sfruttare le intuizioni prodotte per consentire ai giocatori di sviluppare il loro gioco, le squadre, migliorare *roster* e strategie. Questi strumenti aiutano a comprendere e misurare le prestazioni dei giocatori, nonché a costruire una strategia tattica in campo. Viene raccolta una discreta varietà di dati dai *box score*, che sono semplici tabelle contenenti informazioni su ogni giocatore sulle metriche più conosciute (come, ad esempio, punti e falli), fino ad arrivare a dati spaziali quasi continui riferiti alla posizione di ogni giocatore in campo, compresa la palla. Inoltre, negli ultimi anni, si è assistito all'introduzione, nel campo dell'analisi della pallacanestro, di metodi statistici e di Machine Learning avanzati, nonché l'impiego dell'intelligenza artificiale[3]. Per contrapposto, si mostra come risultavano i primi *box score* del 1961:



3-2

NEW YORK (147)	FG.	FT.	F.	Pts.
Naulls	9	13-15	5	31
Green	3	0-0	5	6
Imhoff	3	1-1	6	7
Guerin	13	13-17	5	39
Butler	4	0-0	1	8
Budd	6	1-1	1	13
Butcher	3	4-6	5	10
Buckner	16	1-1	4	33
Totals	57	33-41	32	147

PHILDELPHIA (169)	FG.	FT.	F.	Pts.
Arizin	7	2-2	0	16
Meschery	7	2-2	4	16
Chamberlain	36	28-32	2	100
Rodgers	1	9-12	5	11
Attles	8	1-1	4	17
Lareso	4	1-1	5	9
Conlin	0	0-0	1	0
Ruklick	0	0-2	2	0
Luckenbill	0	0-0	2	0
Totals	63	43-52	25	169

New York	26	42	38	41—147
Philadelphia	42	37	46	44—169

Attendance—4124.

Figura 1.1: New York – Philadelphia NBA *box-score* stagione sportiva 1961/1962

Nell'NBA (*National Basketball Association*), la lega di basket più conosciuta al mondo, ogni *team* ha un dipartimento statistico che offre approfondimenti ad allenatori e giocatori sul gioco e su come la propria squadra può migliorare. Nel corso degli anni, molti giocatori hanno iniziato ad assumere consulenti privati per analizzare i loro risultati e aiutarli ad evidenziare punti di forza e di debolezza per migliorarsi costantemente, persino in tempo reale durante le partite.

La *data analytics* dagli ultimi decenni offre quindi potenzialità e risorse davvero rilevanti, tant'è che sempre più società stanno adottando sistemi avanzati di modelli statistici e analisi dei dati. Di seguito si illustrano i più recenti ambiti d'applicazione specifici per la pallacanestro.

Lo studio "*Optimization of Harmony in Team Formation Problem for Sports Clubs: A Real-Life Volleyball Team Application*"[6] approfondisce come gli allenatori abbiano il compito e la necessità di formare una squadra che vinca le prossime gare. Un *team* è formato da diversi giocatori, e ciò costituisce anche un problema, poiché la combinazione di questi ultimi può portare a complicate e svariate situazioni in campo. In quest'ambito gli allenatori hanno bisogno di un supporto scientifico per comporre la squadra che giocherà la prossima gara. Negli ultimi anni i vincoli

finanziari che possono portare all'adozione di ulteriori strumenti decisionali si sono fatti sempre più pressanti. Il miglior *team* possibile da formare deve prendere in considerazione diversi aspetti, tra cui l'armonia tra i componenti, le performance dei giocatori, gli schemi e le strategie di squadra e il *team* avversario. Il modello matematico che gli autori propongono punta a formare la miglior composizione di giocatori possibile, rispettando una serie di vincoli e priorità stabiliti.

Un'analisi simile, ma orientata a scelte in-game (cioè durante una partita di pallacanestro) è descritta da Lazaros Ntasis nel suo paper "*Optimization techniques for basketball players under the convex risk measures*"[27]. Quando una squadra gioca in casa (oppure al contrario fuori casa) può godere di vantaggi (oppure svantaggi - ad esempio il tifo del pubblico) che possono influenzare, oltre che le performance dei giocatori, anche il risultato stesso. La selezione dei giocatori da sostituire è per gli allenatori un problema ancora irrisolto. L'autore prova come il suo algoritmo ARC ("Athletes-Return-Correction") cerchi di scegliere sempre la miglior strategia possibile in ogni gara in base agli scenari che si presentano.

Il progetto "BDsports"[4] sviluppato nell'ambito del "Big&Open Data Innovation Laboratory" (BODaI-Lab) dell'Università di Brescia, raccoglie numerose analisi effettuate sulla pallacanestro, tra cui:

- "*Interactive analytics gallery using data from the Olympic basketball tournament Rio 2016*": una raccolta contenente tutte le statistiche dei *box-score* durante le Olimpiadi di Rio 2016;

- "*Rshiny application for the analysis of the Italian 2019/2020 Serie C Gold data*": una dettagliata e interattiva applicazione per visualizzare i punti specifici del campo da cui i giocatori hanno tentato/realizzato 2 oppure 3 punti, riguardanti il massimo campionato regionale lombardo;

- "*BallR: Interactive NBA Shot Charts with R and Shiny*": grafici interattivi su statistiche nell'NBA;

- "*Animating NBA Play by Play using R*": un interessantissimo progetto che genera **grafici animati** degli spostamenti dei giocatori in un'azione, inclusi i movimenti della palla (ad esempio tiri o passaggi);

- Numerosi altri articoli riguardanti analisi sulle performance dei giocatori di NBA generaliste o anche specifiche a seconda delle situazioni di gioco, scenari strategici negli ultimi secondi di un'azione e molti altri.

La maggior parte degli studi sono stati svolti sull'NBA poiché negli Stati Uniti la pallacanestro risulta essere molto più seguita ed economicamente rilevante rispetto all'Italia. Questo ha fatto sì che la quantità di dati a disposizione per analizzare tale campionato fosse decisamente più alta rispetto alle altre nazioni. Ulteriori ambiti[19] in cui l'analisi dei dati è intervenuta a supporto delle decisioni sono:

- **Definizioni di strategie**: l'NBA ha installato 6 telecamere in ogni campo di gioco, al fine di raccogliere dati estremamente raffinati sui movimenti dei giocatori. Questi dati sono stati analizzati da appositi modelli di Machine Learning per raccomandare strategie vincenti. Storicamente, le squadre erano in grado di raccogliere solamente statistiche riguardanti i punti realizzati, assist e altre metriche. Con questa tecnologia però, grazie alle telecamere installate, possono analizzare in profondità dati come ad esempio la frequenza con la quale un giocatore si muove in una certa direzione oppure quale piede usa più frequentemente. Tutto ciò ha aiutato gli allenatori a pianificare migliori strategie rispetto agli avversari da incontrare, raccomandando ad esempio alcuni movimenti offensivi basati su dove debbano posizionarsi i propri giocatori per massimizzare la realizzazione di tiri da tre punti.

- **Predire ed evitare gli infortuni dei giocatori**: le squadre hanno collezionato dati rilevanti sui propri giocatori tramite dispositivi indossabili, monitoraggio notturno e addirittura test della saliva per determinare il loro livello di affaticamento e predire le loro performance future. L'applicazione rivoluzionaria di questi dati ha permesso di prevenire ed evitare diversi infortuni. L'idea è quella che chiaramente più un giocatore è stanco, più

è probabile che si infortuni. Alcuni studi hanno dimostrato che vi è meno propensione all'infortunio se i giocatori riposano almeno 30 giorni dopo 30 partite. In tal senso, le squadre hanno adottato strategie decisionali per evitare questi infortuni, soprattutto per i giocatori più importanti (spesso andando anche contro le aspettative dei loro *fans*).

- ***Scouting***: l'analisi dei dati è diventata fondamentale anche nello *scouting* di nuovi giocatori, poiché gli allenatori si affidano anche all'analisi predittiva al momento delle selezioni. Questi momenti sono molto importanti per le società, poiché con le loro scelte potrebbero rimanere vincolati per anni: fondamentale quindi è ridurre al minimo il rischio di fare una scelta "poco produttiva". Gli allenatori possono anche cercare video clip dei giocatori per analizzare i dati monitorati (come, ad esempio, l'efficienza di andare a tirare a canestro con una mano piuttosto che con l'altra) e persino prevedere il ruolo che potrebbe ricoprire quel giocatore in futuro (Guardia, Centro o Playmaker).

Il problema delle assegnazioni arbitrali è tema di studio fin dal 1985 e, con l'evolversi delle tecniche utilizzate e l'affinamento dei vincoli, è arrivato a proporre soluzioni a molteplici problemi nel mondo dello sport. Altri aspetti che hanno visto l'applicazione degli algoritmi di ottimizzazione sono, ad esempio:

- La ricerca di condizioni necessarie al fine di eliminare una squadra in una determinata posizione in classifica[18];

- L'utilizzo dell'*integer linear programming* per il ***playoff problem*** (trovare la combinazione di risultati per cui classificarsi – oppure no – ai playoff)[31];

- Il "***League Scheduling Problem***", dove si cerca di pianificare le gare di una stagione sportiva in modo che le squadre alternino il più possibile il giocare in casa e fuori casa, facendo però altresì in modo che ogni squadra sfidi esattamente due volte tutte le altre[32];

- Il "***Travelling Tournament Problem***", in cui lo scopo primario è quello di minimizzare le distanze percorse dalle squadre per disputare tutte le gare in programma[13].

1.1.1 Previsione della difficoltà delle gare

La bibliografia sulla previsione della difficoltà delle gare di pallacanestro purtroppo non offre alcun articolo scientifico/ricerca rilevante in merito. Tuttavia, nel corso di questo elaborato si dimostrerà come esista una relazione tra la differenza del risultato finale tra due squadre e la difficoltà di tale gara disputata. Alla luce di ciò, la ricerca si è concentrata quindi nell'apprendere gli standard e le metodologie con cui fossero previsti i risultati delle partite di pallacanestro (in inglese, *outcomes*). Molti articoli trattano prevalentemente l'ambito delle scommesse e probabilità di vincita, il nostro focus però si è concentrato unicamente sugli studi dove si tratta esplicitamente il concetto di misurazione delle *performance* di due squadre e, in particolare, dello *spread* (differenza) tra queste due sotto diversi punti di vista e che, inoltre, non si fossero limitati ai *playoff*, ma estesi ad un'intera *regular season*.

Uno tra i primi studi risale al 1984, quando Kathleen Jean Shanahan, tesista dell'Università dell'Iowa, scrive una tesi intitolata "*A model for predicting the probability of a win in Basketball*"[28]. Nello studio si descrive come, all'epoca, gli allenatori di pallacanestro si basassero, per la quasi totalità delle loro decisioni, sulle statistiche sotto forma di percentuali. Il modello è stato costruito sui dati di un campionato studentesco locale maschile e femminile nelle stagioni sportive 1981 e 1982. I risultati delle predizioni sono state rispettivamente del 64% per il campionato maschile, e del 54% per quello femminile. In particolare, l'autore evidenzia come i precedenti lavori in quest'ambito non includessero alcuna tecnica di analisi matematica, e propone quindi l'utilizzo di una regressione logistica per predire ma soprattutto per spiegare la probabilità di tali eventi.

Successivamente, nel 1994, tre professori americani hanno scritto un articolo inti-

tolato "*A Model for Predicting the Outcomes of Basketball Games*"[17], dove venne presa in esame la predizione degli outcomes di 78 gare durante la stagione NBA 1989/1990, con un'accuratezza del 70%. Gli autori si sono basati sulla costruzione di un modello matematico a partire da predizioni effettuate dagli studenti di un college.

Col passare degli anni le tecniche di analisi e Machine Learning si sono evolute, permettendo così un'analisi più ampia e approfondita rispetto al passato, la quale ha permesso di incrementare i risultati ottenuti. I seguenti articoli riguardano tutti l'NBA.

Nel 2008 Beckler, Wang e Papamichael creano "*NBA Oracle*"[5], uno tra i progetti che ha riscosso più successo in quegli anni, raggiungendo la percentuale mai raggiunta prima del 73% per le cinque stagioni dal 1992 al 1995. Gli autori hanno impiegato algoritmi di Machine Learning quali Regressione Lineare, Support Vector Machines, Logistic Regression e Artificial Neural Networks. In questo studio vengono anche predette le posizioni ottimali per i giocatori (Centro, Guardia, Play) e l'identificazione di giocatori definiti "*oustanding*" (fuori dal comune).

Nel 2010, Eftim Zdravevski e Andrea Kulakov scrivono "*System for prediction of the winner in a sports game*"[36], e sono tra i primi ad utilizzare gli algoritmi di predizione e classificazione con Weka[8]. Con un approccio *team-oriented* riescono a definire e calcolare alcuni importanti parametri puntuali prima di ogni gara da predire, quali ad esempio: numero di giocatori infortunati in una squadra, serie di vittorie, affaticamento del team (per spostarsi tra le varie città in cui giocare), percentuali di vittore/sconfitta, *ratings* (valutazioni) offensive, difensive e generali. Tra i 36 *classifier* testati il più performante è stata la Logistic Regression (72,8%).

Sempre nel 2010 l'articolo "*The Use of Data Mining for Basketball Matches Outcomes Prediction*"[25] di Dragan Miljković, Ljubiša Gajić, Aleksandar Kovačević, Zora Konjović (Serbia), implementa tecniche di data mining per predire i risul-

tati delle gare di NBA e formalizza il problema come un *classification problem*. Per ogni gara viene calcolato, attraverso metodi Naive Bayes, lo spread tra le due squadre dove, per la stagione 2009/2010, hanno raggiunto un'accuratezza del 67%.

Nel 2013, dall'Università del Wisconsin-Madison, Renato Amorim Torres scrive "*Prediction of NBA games based on Machine Learning Methods*"[30], dove utilizza metodi di Machine Learning per determinare quale tra le due squadre sfidanti possieda le migliori statistiche e sceglierla come vincente, anche in relazione al minor numero di partite perse tra le due. Interessanti sono le *feature* che seleziona come più rilevanti: percentuale di partite vinte/perse in tutta la stagione e solo nelle ultime 8 gare, differenziale di punti di tutte le gare passate, percentuale di partite vinte/perse giocando in casa (*home*) e fuori casa (*away*). L'accuracy media tra le stagioni 2006 e 2012 in questo caso è stata del 64%.

Un approccio diverso, invece, offrono Ge Cheng, Zhang, Kyebambe e Kimbugwe nel 2016, con il loro articolo intitolato "*Predicting the Outcome of NBA Playoffs Based on the Maximum Entropy Principle*"[7]. Il problema viene formalizzato come un *classification problem*, a cui viene però applicato il "Principio di massima entropia" per costruire un modello denominato "NBA Maximum Entropy (NBA-ME)", il quale raggiunge un'accuratezza del 74.4%. Secondo gli autori il modello della Massima entropia è più rivolto alla costruzione delle *features* e al processamento di tali dati; utilizzando questo modello si cerca di oltrepassare l'ipotesi di indipendenza delle variabili che limita invece il modello Naive Bayesian.

Nel 2017 Paolo Giuliodori, dell'Università di Camerino, scrive "*An Artificial Neural Network-based Prediction Model for Underdog Teams in NBA Matches*"[16], proponendo l'uso di reti neurali per predire i team vincenti (con il 72% di accuratezza), lavorando con particolare attenzione su quelli "*underdogs*" (team che probabilmente non vinceranno l'incontro). Aspetto interessante è che Giuliodori ha confrontato il proprio sistema con una tra le principali aziende di scommesse per rafforzare i propri risultati.

Il più recente è il lavoro di Max W. Y. Lam il quale nel 2017 scrive "*One-Match-Ahead Forecasting In Two-Team Sports With Stacked Bayesian Regressions*"[20], articolo preso in analisi da questa tesi ed approfondito nel corso dei prossimi capitoli. L'autore propone l'utilizzo in sequenza di due regressioni SSGPR (Sparse Spectrum Gaussian Process Regression), il cui modello prende il nome di "TLGProb", al fine predire le performance future dei singoli giocatori e dei team. Relativamente alla stagione sportiva NBA 2014/2015, l'accuratezza risulta essere dell'85%, la più alta mai raggiunta dai precedenti lavori in quest'ambito.

Di seguito è proposta una tabella riassuntiva dei riferimenti bibliografici.

Rif. bibl.	Problema affrontato
[28]	Predizione della probabilità di vincita in una *conference* e identificazione dei punti di forza e debolezza delle squadre (NBA 1981-1982)
[17]	Costruzione di un modello matematico a partire da predizioni effettuate dagli studenti di un college (NBA 1989/1990)
[5]	Impiego di algoritmi di Machine Learning per predire le posizioni ottimali per i giocatori e l'identificazione di giocatori "*oustanding*" (NBA 1992-1995)
[36]	Utilizzo di algoritmi di predizione con Weka. Un approccio *team-oriented* ha permesso di definire 36 parametri prima di ogni gara da predire (ad es. affaticamento del team, *ratings* offensivi, ecc.)
[25]	Implementazione di tecniche di data mining per predire i risultati formalizzando il problema come un *classification problem* attraverso metodi Naive Bayes (NBA 2009/2010)
[30]	Predizione squadra vincente con algoritmi di Machine Learning con selezione delle *features* più rilevanti (NBA 2006-2012)
[7]	Creazione di un modello che segue il "Principio di massima entropia" per predire gli *outcomes* dei *Playoffs* di NBA
[16]	Uso di reti neurali per predire i team vincenti confrontandoli con una tra le principali aziende di scommesse (2016/2017)
[20]	Predizione della probabilità di vincita tramite stima delle performance future dei singoli giocatori e di team, utilizzando in sequenza di due regressioni SSGPR (2014/2015)

Tabella 1.1: Riassunto lavori precedenti sulla predizione degli *outcomes* delle gare di NBA

1.1.2 Il "Referee Assignment Problem"

Il "Referee Assignment Problem" (abbreviato anche come "RAP") è stato trattato in letteratura da numerosi articoli, in particolare da Alexandre Duarte nel 2007[10], il quale presentò una versione generica del problema, descrivendo i motivi per cui ogni sport, nazione e lega hanno le loro peculiarità. Duarte è considerato al giorno d'oggi uno dei più importanti autori che ha contribuito in questo campo di ricerca.

Il RAP è stato studiato precedentemente anche da Evans nel 1984[14] e 1988[15], dove il problema è stato applicato alle designazioni degli arbitri nella American Baseball League. In tale versione il tempo risulta essere una componente fondamentale, attorno alla quale diversi vincoli e regole sono stati imposti, come ad esempio il numero di giorni che devono trascorrere tra due gare da arbitrare in base alla distanza che deve percorrere l'arbitro. Per risolvere il problema Evans usò tecniche di ottimizzazione per prendere decisioni, unite ad euristiche e giudizio umano.

Nel 1991, un algoritmo euristico è proposto da Wright[33] per risolvere il RAP, applicato alla lega del cricket in Inghilterra, e nel 2015 scrisse un altro articolo[34], dove migliorò la soluzione precedentemente proposta e la ampliò applicandola a più leghe.

Duarte, nel suo primo articolo del 2007, utilizzò un *integral model* per il RAP, risolvendolo utilizzando tre fasi basate su algoritmi euristici. Nella prima fase un'euristica *greedy* cerca una soluzione accettabile al problema assegnando gli arbitri a quante più gare possibili senza violare alcun vincolo, se possibile. Se almeno un vincolo (*hard*) viene violato allora viene intrapresa la seconda fase, dove un algoritmo di ricerca locale viene iterato per "riparare" la soluzione precedente modificando le assegnazioni una alla volta, per un dato numero massimo di iterazioni. In ultima fase, un algoritmo basato sulla meta-euristica cerca un massimo locale. Questo non garantisce però di trovare una soluzione né ottima né accetta-

bile, poiché la seconda fase potrebbe non trovarne una. Nel 2007[9] quindi, sempre Duarte, pubblica un altro articolo dove estende il suo lavoro precedente, nel quale utilizza un nuovo approccio misto ad algoritmi di ricerca locali e un modello lineare apposito per migliorare la terza fase per la ricerca di un ottimo locale.

Guillermo Duran è un altro importante autore sul tema del RAP. Nel 2005[11] propose una soluzione allo *scheduling* della prima lega di calcio del Cile, dove trattò il problema di ottimizzare la sequenza di gare nelle fasi di *playoff*, con lo scopo di minimizzare il tempo impiegato tra *match* consecutivi. Nel 2010[12] risolse poi un'altra applicazione del RAP, per la seconda divisione di calcio sempre cilena, in cui dovette minimizzare il numero di volte che una squadra giocava consecutivamente in casa oppure fuori.

Nel 2008[35] Yavuz propose un modello per la lega di calcio della Turchia in cui utilizzò algoritmi di ricerca locale per evitare la frequente assegnazione degli arbitri alle stesse squadre. Nel 2007[22] e 2010[21] Ferland e Lamghari presentarono diverse versioni del RAP risolte con la *tabu search* e diversificazione di strategie basate su differenti *neighborhoods*.

Sempre Duran, nel 2013[1] affrontò nuovi aspetti legati al RAP, come il bilanciamento del numero di designazioni per ogni arbitro, assegnazioni frequenti alla stessa squadra, la distanza percorsa e la differenza tra il livello di abilità di un arbitro e l'importanza della gara.
Similarmente, ma nel 2019[23] Linfati, Gatica ed Escobar hanno presentato un articolo dove viene proposto un modello binario non lineare, approfondendo quello del 2013 di Duran, al fine di ottimizzare le differenze tra il livello di competenza di un arbitro e l'importanza della gara (applicandolo però a più sport, tra cui calcio, pallavolo e pallacanestro), utilizzando CPLEX. Quest'ultimo è un risolutore della programmazione matematica ad elevate prestazioni per programmazione lineare, a interi misti e quadratica prodotto da IBM[8].

Tra gli ultimi lavori prodotti vi è quello di Farners Vallespí i Soro dell'Università della Catalogna, la quale nel 2019 ha scritto una tesi intitolata proprio "*The Referee Assignment Problem*"[29], dove viene proposta la soluzione del RAP in due versioni: una più semplice e generica ed una specifica per il KNVB (la Federazione calcistica dei Paesi Bassi) per la stagione sportiva 2018/2019. Gli algoritmi utilizzati sono stati quelli di ricerca locale, l'Hill Climbing[1], il Simulated Annealing[2], e l'ILP (*Integer Linear Programming*)[3] con il CPLEX.

Di seguito è riportata una tabella riassuntiva dei riferimenti bibliografici e relative problematiche affrontate.

[1] `https://en.wikipedia.org/wiki/Hill_climbing`

[2] `https://en.wikipedia.org/wiki/Simulated_annealing`

[3] `https://www.sciencedirect.com/topics/computer-science/integer-linear-programming`

Rif. bibl.	Problema affrontato
[10],[9]	Prima formalizzazione del problema del RAP basata su algoritmi euristici *greedy* (2007)
[14],[15]	Designazioni nella American Baseball League attraverso vincoli e tecniche di ottimizzazione unite ad euristiche e giudizio umano (1984 e 1988)
[11],[12]	Ottimizzazione della sequenza di gare nelle fasi di *playoff* con lo scopo di minimizzare il tempo trascorso tra *match* consecutivi per la divisione di calcio cilena (2005 e 2010)
[35]	modello per la lega di calcio turca implementato con algoritmi di ricerca locale per evitare la frequente assegnazione degli arbitri alle stesse squadre (2008)
[22],[21]	Risoluzione del RAP con la *tabu search* e diversificazione di strategie basate su differenti *neighborhoods* (2007 e 2010)
[1]	Introduzione di nuove aspetti legati al RAP: bilanciamento del numero di designazioni per ogni arbitro, distanza percorsa e differenza tra il livello di abilità di un arbitro e l'importanza della gara (2013)
[23]	Realizzazione di un modello binario non lineare per ottimizzare le differenze tra il livello di competenza di un arbitri e l'importanza della gara utilizzando CPLEX (2019)
[29]	Risoluzione del RAP per la KNVB con algoritmi ricerca locali quali l'Hill Climbing, il Simulated Annealing e l'Integer Linear Programming (2019)

Tabella 1.2: Riassunto lavori precedenti sul "Referee Assignment Problem"

1.1.3 Difficoltà del *task*

L'articolo scientifico "*Luck is Hard to Beat: The Difficulty of Sports Prediction*"[2] ha effettuato nel 2007 un'analisi di quanto predire i risultati di una gara futura sia difficile, calcolando un coefficiente che misura la distanza tra i risulta-

ti finali osservati e le competizioni perfettamente bilanciate in termini di abilità. Questo ha indicato la presenza sia di fortuna che di abilità, e ciò sembra essere normale; poiché uno sport totalmente basato sulla fortuna (come può essere ad esempio la lotteria) lo renderebbe totalmente casuale, mentre se fosse basato solo sulle abilità, sarebbe facilmente predicibile (e quindi poco interessante per gli appassionati).
Gli autori hanno calcolato la probabilità di vincita delle squadre in NBA (per le stagioni dalla 2012 alla 2016) che avevano un livello di abilità inferiore rispetto agli avversari (in inglese "*underdog*", perdente). Questa si è attestata al 36%, ed è stata rilevata come la componente casuale della massima Lega di pallacanestro americana. Si evince quindi la presenza di aleatorietà in questo sport, e come possa essere quindi difficoltoso, a volte, predire la difficoltà di una gara futura basandosi sui livelli di abilità delle squadre.

1.2 Machine Learning

1.2.1 Introduzione

Il Machine Learning (ML) è un sottoinsieme dell'intelligenza artificiale (AI) che si occupa di creare sistemi che apprendono o migliorano le performance in base ai dati che utilizzano.
Alcune delle attività tipiche del ML sono: *clustering*, classificazione, categorizzazione, filtro/selezione, riconoscimento (*pattern recognition*), simulazione di giochi e comportamento autonomo. Il Machine Learning può risultare particolarmente utile nel caso in cui, ad esempio, vi sia la disponibilità di grandi quantità di dati da poter analizzare, quando non vi sono abbastanza persone per risolvere un problema, oppure ancora quando occorre trovare dei modelli (*pattern*).
Gli algoritmi sono i motori che alimentano il Machine Learning. I principali due tipi comunemente utilizzati sono gli algoritmi di Machine Learning supervisionato e non supervisionato. La differenza tra queste due tipologie viene definita dal modo in cui ciascun algoritmo apprende i dati per fare previsioni ed inferenze.

L'Intelligenza Artificiale coinvolge, quindi, tutte quelle operazioni caratteristiche dell'intelletto umano ed eseguite da un computer, come la pianificazione, la comprensione del linguaggio, il riconoscimento di oggetti e suoni, l'apprendimento e la risoluzione dei problemi. Un esempio interessante è la relazione tra AI inteso come cervello e l'IoT (*Internet of Things*) come il corpo umano: quest'ultimo, attraverso i vari *input* come la vista e il tatto, riesce a riconoscere determinate situazioni e ad eseguire le corrispondenti azioni decise dal nostro cervello. Analogamente nell'IoT, tramite i sensori presenti, si invia un insieme di informazioni ad un sistema centrale di Intelligenza Artificiale, la quale prende le opportune decisioni ed eventualmente aziona determinati attuatori per eseguire le azioni (ad es. componenti di un macchinario industriale). L'apprendimento automatico o Machine Learning è invece un modo per attuare l'Intelligenza Artificiale, mentre l'apprendimento approfondito o Deep Learning, è uno dei molteplici approcci relativi all'apprendimento automatico.

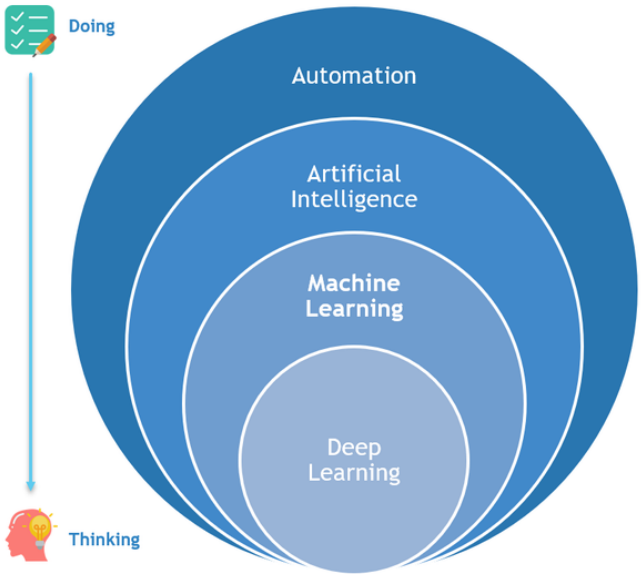


Figura 1.2: Gerarchia dei diversi ambiti nell'intelligenza artificiale

Definizione di Intelligenza Artificiale

Diverse sono le definizioni attribuite all'Intelligenza Artificiale. Per comprenderne però il significato è necessario prima definire il concetto di "intelligenza": l'*Oxford English Dictionary* la definisce come "la rapidità di comprensione", il *Cambridge Advanced Learner's Dictionary* "la capacità di imparare, capire e fare giudizi o avere opinioni che siano basati sulla ragione". L'enciclopedia *Rizzoli-LaRousse* come "la facoltà di conoscere, di comprendere e di intendere", ed infine Wikipedia "l'intelligenza è l'insieme di funzioni conoscitive, adattive e immaginative generate dall'attività cerebrale dell'uomo e degli animali". Tutte queste definizioni possono essere riassunte come "la capacità di imparare dall'esperienza ad applicare conoscenza per risolvere i problemi e sopravvivere in diversi ambienti sociali e geografici". Thurstone, nel 1930, identifica diversi fattori che descrivono l'intelligenza di un individuo: fluidità verbale, abilità numerica, inferenza, abilità spaziali, velocità di percezione e memoria. L'intelligenza artificiale può essere quindi definita come "l'abilità di un computer di eseguire azioni solitamente proprie di esseri intelligenti". Tom M. Mitchell, nel suo libro "Machine Learning"[26] ha fornito una definizione più dettagliata: "*si dice che un programma apprende dall'esperienza E con riferimento a alcune classi di compiti T e con misurazione della performance P, se le sue performance nel compito T, come misurato da P, migliorano con l'esperienza E*". Questo evidenzia come un programma apprenda solo se c'è un miglioramento delle prestazioni dopo un compito svolto.
Il Machine Learning è impiegato principalmente per la risoluzione di tre tipologie di problemi:

- *Classificazione*: quando è necessario decidere a quale categoria appartiene un determinato dato;
- *Regressione*: ovvero prevedere il valore futuro di un dato avendo noto il suo valore attuale;
- *Raggruppamento (clustering)*: quando si vuole raggruppare i dati che presentano caratteristiche simili.

Metodologie di apprendimento nel Machine Learning

Il Machine Learning si basa su due distinti approcci, identificati da Arthur Samuel alla fine degli anni '50, che permettono di distinguere l'apprendimento automatico in due sottocategorie. Ciò si basa sul fatto di fornire al computer esempi completi da utilizzare come indicazione per eseguire il compito richiesto, oppure di lasciar lavorare il software senza alcuna guida particolare; nel primo caso si tratta di apprendimento "supervisionato", mentre nel secondo di apprendimento "non supervisionato". Nello specifico, si hanno quindi le seguenti metodologie di apprendimento:

- *Supervised Learning*;
- *Unsupervised Learning*;
- *Reinforcement Learning.*

Esiste inoltre un approccio intermedio tra le prime due noto come Semi-Supervised Learning.
Nell'immagine seguente le varie metodologie di apprendimento sono descritte e messe in relazione alle tipologie di problematiche che il Machine Learning si propone di affrontare, descritte nelle sezioni precedenti.

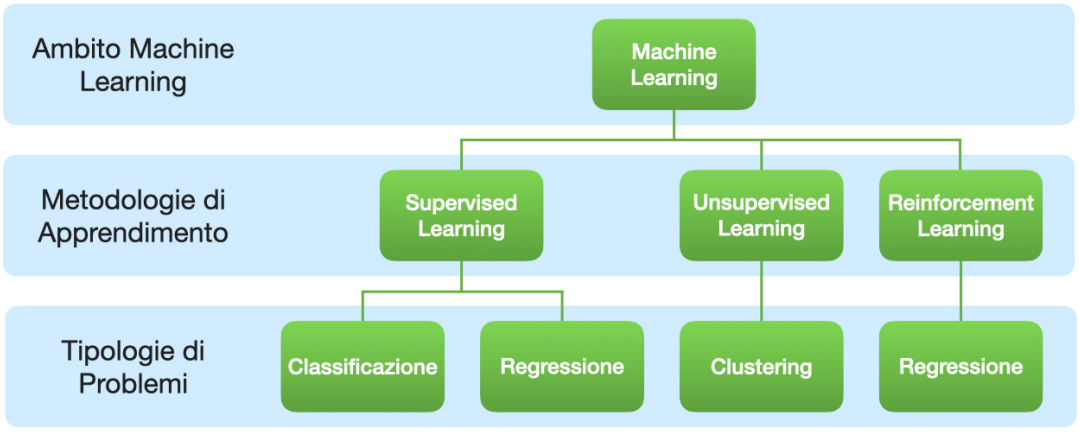


Figura 1.3: Gerarchia delle metodologie di apprendimento e tipologie di problemi nel Machine Learning

Processo

A prescindere dal tipo di modello di Machine Learning utilizzato, si ha quindi una fase iniziale di *training* dove il modello viene addestrato, ed una di *testing* dove il modello viene messo alla prova con predizioni su dati che non ha mai elaborato precedentemente. Il set di dati per il *testing* consiste in una porzione di dati scelta casualmente dall'intero dataset a disposizione. A queste due tipologie di dataset può aggiungersene una terza, ovvero il "*validation*" dataset, il quale viene utilizzato per controllare i risultati del processo di addestramento.

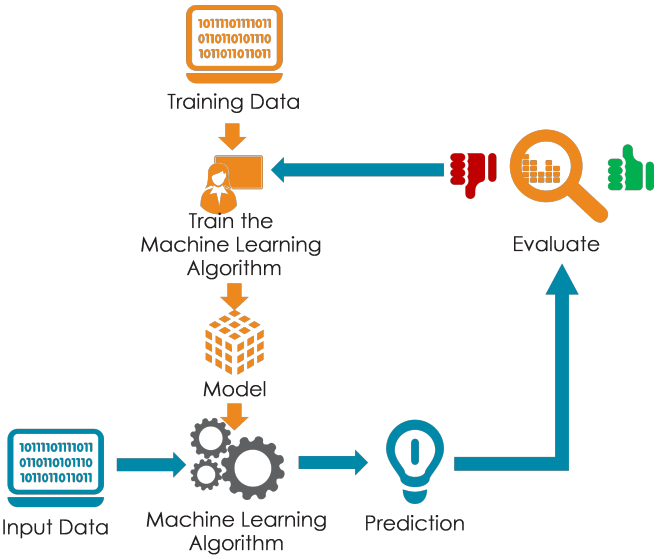


Figura 1.4: Schema sequenziale dei processi nell'utilizzo del Machine Learning

1.2.2 Supervised Learning

La tecnica del Supervised Learning (apprendimento supervisionato) è composta da due fasi: training (allenamento) e prediction (predizione). La macchina viene inizialmente istruita attraverso un set di dati etichettati, ovvero dati per i quali si conosce già la risposta obiettivo (ad esempio la classe di appartenenza). In base a tali dati appresi, la macchina è in grado di fare le opportune predizioni. Nel dettaglio si ha quindi:

- *Training*: fase in cui la macchina apprende da un insieme di esempi ideali formati da coppie di input e output desiderato, detto anche "*training set*". Quest'ultimo è ottenuto attraverso una raccolta di dati oppure è generato da un esperto del dominio;

- *Prediction*: fase in cui la macchina utilizza il modello ottenuto dalla fase di *training* ed applica il *mapping* ai nuovi input, producendo così in output la predizione.

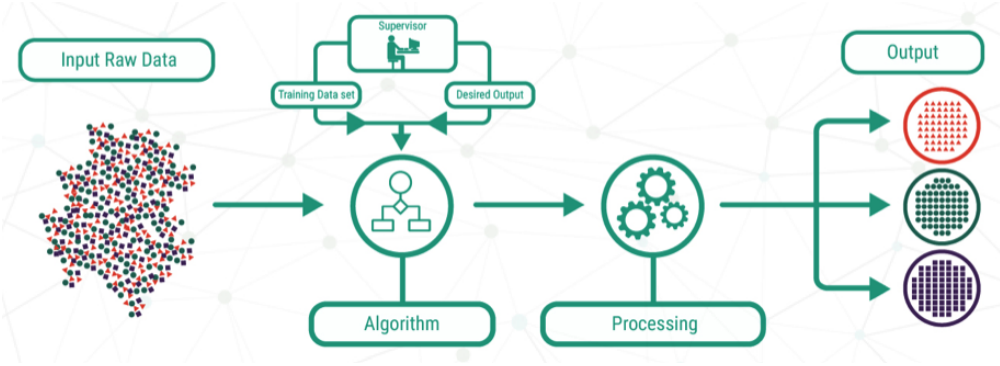


Figura 1.5: Schema illustrativo delle diverse fasi del Supervised Learning

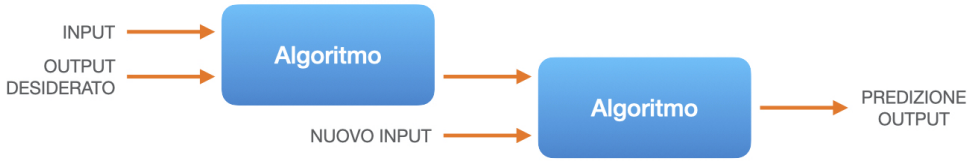


Figura 1.6: Schema illustrativo dell'utilizzo a cascata del Supervised Learning con i diversi input/output

L'apprendimento supervisionato è principalmente utilizzato per i problemi di classificazione, come ad esempio nel marketing per classificare i clienti potenziali e proporre i prodotti a cui potrebbero essere interessati sulla base del profilo e della storia degli acquisti. Un altro esempio sono i sistemi antispam delle e-mail che, all'arrivo di un messaggio, riescono a decidere se una determinata e-mail debba essere etichettata come spam o meno. L'apprendimento supervisionato è inoltre utilizzato per i problemi di regressione, come ad esempio nelle previsioni meteo. Nella classificazione l'output corrisponde alla probabilità dell'input di appartenere ad una data categoria; esempi di tecniche di Machine Learning: alberi decisionali, Bayesian Network, Support Vector Machines, K-Nearest Neighbor e Neural Network. Nella regressione, invece, l'output è rappresentato da valore numerico ottenuto tramite combinazione lineare (somma o prodotto) di input lineari o non lineari (ad es. esponenziale, sigmoide o logaritmo).

Nell'ambito meteorologico, un esempio di applicazione di un algoritmo di classificazione potrebbe essere che, in base alla temperatura, umidità e vento, si determini la previsione per il giorno successivo (soleggiato, nuvoloso o piovoso). La regressione potrebbe essere utilizzata, sfruttando gli stessi parametri in input, per predire la temperatura del giorno successivo.

Di seguito vengono presentate le tecniche che saranno poi utilizzate nel progetto.

Regressione Logistica

La regressione logistica è un algoritmo di Machine Learning ed appartiene ai supervised learning. In generale, la regressione logistica la si intende come binaria (cioè con variabili target binarie), ma possono esserci altre due categorie di variabili target che possono essere previste da essa. In base a questo numero di categorie, la regressione logistica può essere suddivisa nei seguenti tipi:

- *Binario o binomiale*: una variabile dipendente avrà solo due possibili esiti: 1 oppure 0. Ad esempio, queste variabili possono rappresentare "successo" o "fallimento", "sì" o "no", "vittoria" o "perdita";
- *Regressione multinomiale*: la variabile dipendente può avere 3 o più esiti non ordinati possibili. Ad esempio, queste variabili possono rappresentare "Tipo A" o "Tipo B" o "Tipo C";
- *Regressione ordinale*: in questo tipo di classificazione, la variabile dipendente può avere 3 o più possibili esiti ordinati. Ad esempio, queste variabili possono rappresentare "scarso" o "buono", "molto buono", "eccellente" e ogni categoria può avere punteggi come 0, 1, 2, 3.

La regressione logistica prende il nome dalla funzione utilizzata al centro del metodo: la "funzione logistica". Quest'ultima, chiamata anche "funzione sigmoide", è stata sviluppata per descrivere le proprietà della crescita della popolazione in ecologia, aumentando rapidamente e massimizzando la capacità di carico dell'ambiente. È una curva a forma di "S" che può prendere qualsiasi numero a valore

reale e mapparlo in un valore compreso tra 0 e 1, ma mai esattamente a quei limiti. Nell'apprendimento automatico, la funzione sigmoide viene utilizzata per mappare le previsioni alle probabilità, ed è rappresentata dal grafico seguente.

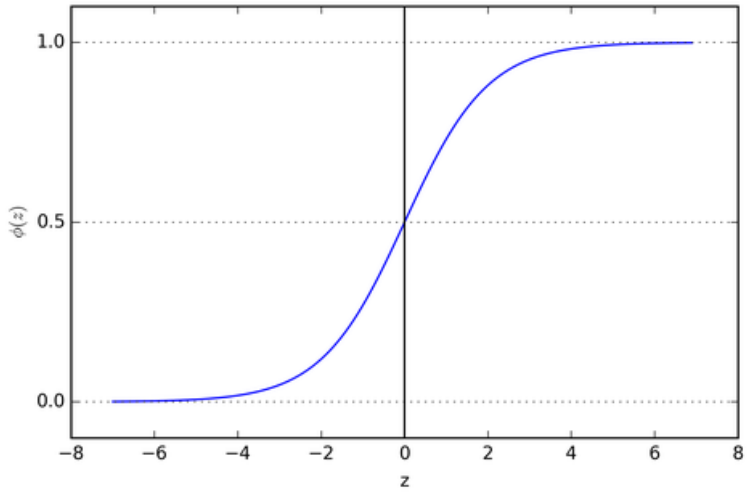


Figura 1.7: Funzione sigmoide

Come è possibile notare, i valori dell'asse y sono compresi tra 0 e 1 e incrociano l'asse a 0,5. L'equazione utilizzata per rappresentare la funzione sigmoide è: $1/(1+e^{-z})$. Le classi possono essere suddivise in positive o negative. L'output è la probabilità di una classe positiva se è compresa tra 0 e 1.

Support Vector Machine

Il Support Vector Machine (SVM) è un algoritmo di apprendimento automatico supervisionato che può essere utilizzato sia per scopi di classificazione che di regressione. L'SVM è basato sull'idea di trovare un *iperpiano* che divida al meglio un set di dati in due classi.
L'*iperpiano*, per un'attività di classificazione con solo due dimensioni spaziali x1 e x2 (o x e y), è raffigurato come una linea che separa e classifica un insieme di dati. A 3 dimensioni, un iperpiano è rappresentato da un piano. Con più di tre dimensioni viene definito genericamente "iperpiano".

I Support Vector (vettori di supporto, in italiano) sono i punti dati più vicini all'iperpiano. Tali punti dipendono dal set di dati che si sta analizzando e se vengono rimossi o modificati alterano la posizione dell'iperpiano divisorio. Per questo motivo, possono essere considerati gli elementi critici di un set di dati.
Il margine, invece, è definito come la distanza tra i vettori di supporto di due classi differenti più vicini all'iperpiano. Alla metà di questa distanza viene tracciato l'iperpiano, o retta nel caso si stia lavorando a due dimensioni. Il Support Vector Machine ha l'obiettivo di identificare l'iperpiano che meglio divide i vettori di supporto in classi. Per farlo esegue i seguenti passi:

1. Cerca un iperpiano linearmente separabile o un limite di decisione che separa i valori di una classe dall'altro. Se ne esiste più di uno, cerca quello che ha margine più alto con i vettori di supporto, per migliorare l'accuratezza del modello;

2. Se tale iperpiano non esiste, SVM utilizza una mappatura non lineare per trasformare i dati di allenamento in una dimensione superiore (se a due dimensioni, valuterà i dati in 3 dimensioni). In questo modo, i dati di due classi possono sempre essere separati da un iperpiano, che sarà scelto per la suddivisione dei dati.

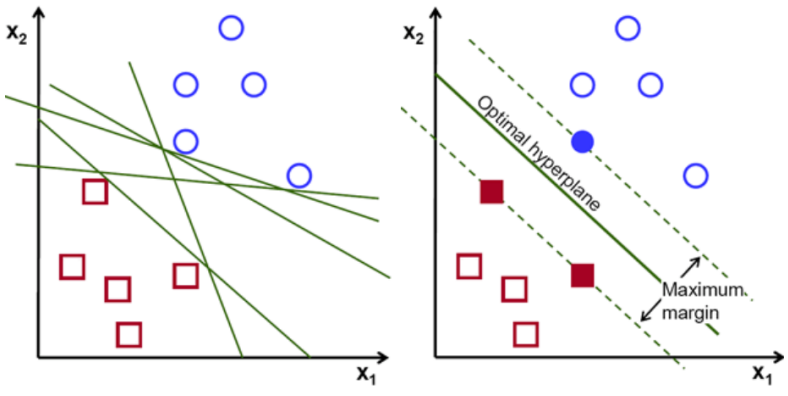


Figura 1.8: Scelta dell'iperpiano ottimale per SVM

1.2.3 Unsupervised Learning

Mediante la tecnica dell'Unsupervised Learning, la macchina viene istruita con un set di dati non etichettati. Lo scopo è quello di raggruppare gli elementi che hanno caratteristiche simili.
L'applicazione principale è il *clustering*, ovvero il raggruppamento dei dati in gruppi omogenei, definiti "*cluster*". L'apprendimento non supervisionato, quindi, serve generalmente ad estrarre informazioni non ancora note. Per questo motivo è un metodo spesso utilizzato ad esempio nella medicina per l'analisi diagnostica, ma anche nel marketing per l'individuazione di potenziali mercati e clienti. Esso trova grande applicazione nell'ambito dei Big-Data, quando è necessario correlare diversi dati ed estrarre informazioni non note (detto anche "*knowledge discovery*", scoperta di conoscenza), oppure nel raggruppamento di volti di persone simili, ad esempio in base all'età o a specifiche caratteristiche.

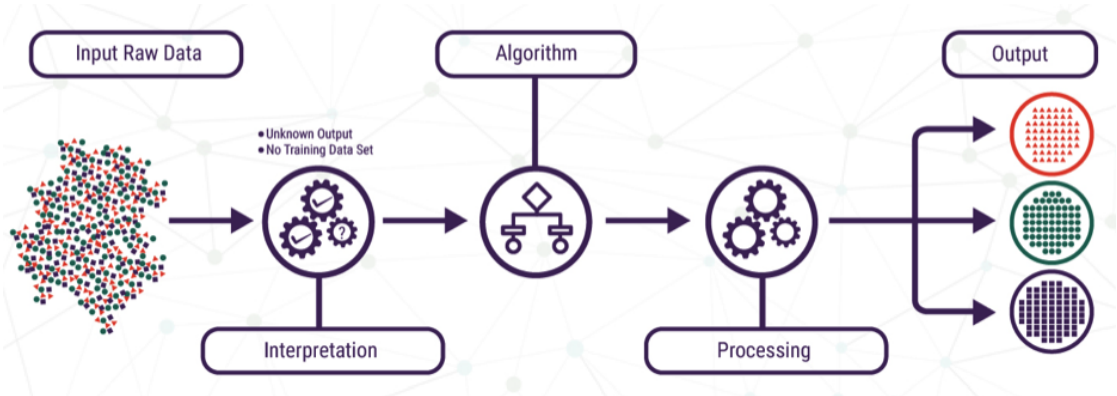


Figura 1.9: Schema illustrativo delle diverse fasi dell'Unsupervised Learning

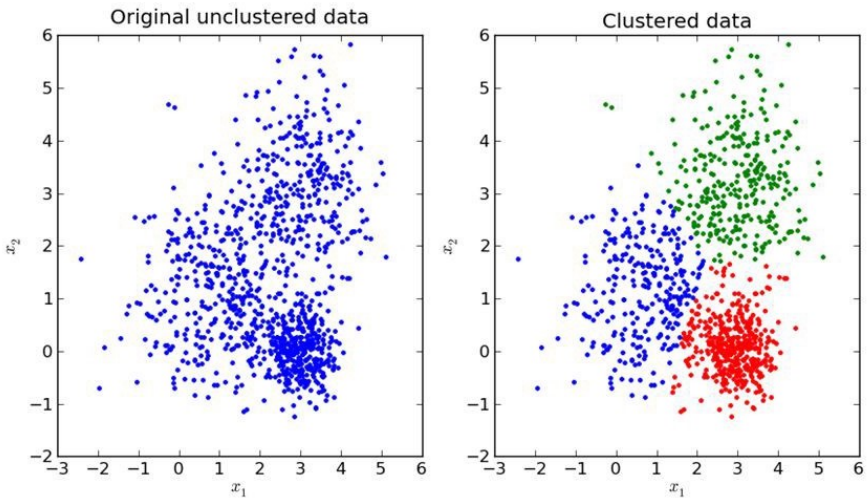


Figura 1.10: Esempio di clustering utilizzando l'Unsupervised Learning

K-Means

Il K-Means è un algoritmo di apprendimento non supervisionato che trova un numero fisso di cluster in un insieme di dati.
I cluster rappresentano i gruppi che dividono gli oggetti a seconda della presenza o meno di una certa somiglianza tra di loro, e vengono scelti a priori, prima dell'esecuzione dell'algoritmo. Ognuno di questi cluster raggruppa un particolare insieme di oggetti, che vengono definiti data points. L'insieme dei data points analizzati definisce il set di dati, che rappresenta l'insieme di tutte le istanze analizzate dall'algoritmo.

Quando si utilizza un algoritmo K-Means, per ogni cluster si definisce un centroide, ossia un punto (immaginario o reale) al centro di un cluster.

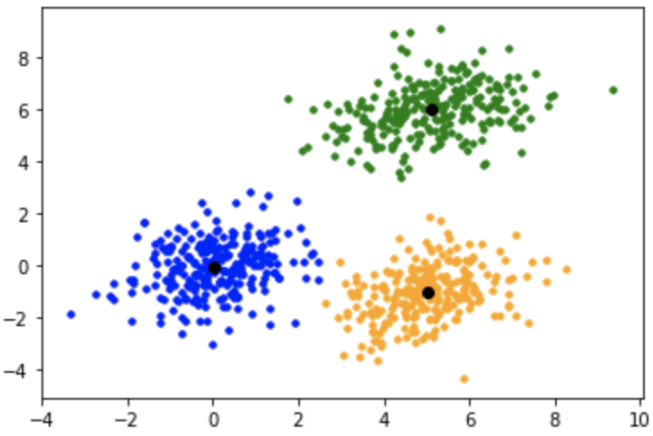


Figura 1.11: Cluster e centroidi con il K-Means

L'algoritmo k-means è un algoritmo iterativo, ossia che esegue ripetutamente alcune sue fasi:

1. *Inizializzazione*: si definiscono i parametri di input per eseguire l'algoritmo, scegliendo l'ampiezza del set di dati e k centroidi iniziali disposti casualmente. Scegliendo il numero di centroidi, si scelgono i cluster cui il data set sarà composto e quindi i raggruppamenti che si vogliono effettuare e visualizzare;

2. *Assegnazione del cluster*: ogni data points viene assegnato al cluster (o centroide) più vicino. Per fare ciò viene calcolata la distanza euclidea tra ogni data points e ogni centroide. Ogni data points sarà poi assegnato al centroide la cui distanza risulti minima;

3. *Aggiornamento della posizione del centroide*: ricalcola il punto esatto del centroide e di conseguenza ne modifica la sua posizione. Dopo il passaggio precedente, è probabile che si siano formati nuovi cluster, in quanto a quelli già esistenti si saranno assegnati (o tolti a seconda che essi siano passati ad un altro cluster) nuovi data points. Di conseguenza, si ricalcola la posizione media dei centroidi: il nuovo valore di un centroide sarà la media di tutti i data points che sono stati assegnati al nuovo cluster.

L'algoritmo procederà fino alla convergenza. Si dice che la condizione di stop in questo caso è stata raggiunta. Di solito essa è rappresentata da una delle seguenti opzioni:

- nessun data points cambia cluster;
- la somma delle distanze è ridotta al minimo;
- viene raggiunto un numero massimo di iterazioni.

L'algoritmo k-means ha il vantaggio di essere veloce, in quanto sono richiesti pochi calcoli e di conseguenza poco tempo di elaborazione al computer per calcolare le distanze tra i data points e i centroidi ad ogni iterazione (chiaramente dipende dal set di dati e dal numero di cluster presenti).
Al contempo però presenta alcuni svantaggi. Il primo riguarda la selezione di quanti gruppi visualizzare, in quanto non sempre è possibile farlo, soprattutto per problemi di complessità maggiore. Il secondo consiste nella scelta casuale di centroidi iniziali e, pertanto, può produrre risultati di clustering diversi su diverse sequenze dell'algoritmo.

1.2.4 Reinforcement Learning

Mediante la tecnica del Reinforcement Learning, la macchina lavora in modo simile all'apprendimento non supervisionato in quanto i dati non sono etichettati. Il metodo utilizzato però è diverso: la macchina viene premiata o penalizzata in base al risultato. L'algoritmo si suddivide in tre fasi:

1. *Sistema di esecuzione (A)*: sulla base dei dati in ingresso (input) riesce a restituire un risultato (output);
2. *Sistema di valutazione (B)*: assegna un premio se il risultato è corretto, oppure una penalità se non è corretto;
3. *Sistema di ottimizzazione (C)*: osserva il comportamento di A e B e modifica il modello utilizzato da A per aumentare il premio e ridurre le penalità che B assegna da A.

Figura 1.12: Schema illustrativo delle diverse fasi del Reinforcement Learning

L'apprendimento per rinforzo è utilizzato in tutti quei campi in cui è fondamentale che la macchina risponda ai cambiamenti dell'ambiente. Viene utilizzato frequentemente, ad esempio, nella robotica per controllare i movimenti degli automi, così come nella guida senza conducente (*driverless car*).

1.2.5 Deep Learning e Reti Neurali

Il Deep Learning è una branca del Machine Learning che caratterizza i processi di reti neurali artificiali dotate di due o più livelli capaci di processare informazioni in modo non lineare. Esso combina computer sempre più potenti a speciali tipi di sistemi neuronali per apprendere i complicati schemi dei grandi volumi di dati. Il deep learning può essere definito come un sistema che sfrutta una classe di algoritmi di apprendimento automatico, i quali:

- Usano **vari livelli di unità non lineari a cascata per svolgere compiti di estrazione di caratteristiche e di trasformazione**. Ciascun livello successivo utilizza l'uscita del precedente come input. Gli algoritmi possono essere sia di tipo supervisionato sia non supervisionato e le applicazioni includono l'analisi di pattern per il primo e di classificazione per il secondo;
- Sono basati sull'**apprendimento non supervisionato di livelli gerarchici multipli di caratteristiche (e di rappresentazioni) dei dati**. Le

caratteristiche di più alto livello vengono derivate da quelle di livello più basso per creare una rappresentazione gerarchica;

- Apprendono multipli livelli di rappresentazione che corrispondono a differenti livelli di astrazioni; tali livelli formano una **gerarchia di concetti**.

Seppur la richiesta di capacità computazionali enormi possa rappresentare un limite, la scalabilità del Deep Learning, grazie all'aumento dei dati disponibili e degli algoritmi, è ciò che lo differenzia maggiormente dal Machine Learning. I sistemi Deep Learning, infatti, migliorano le proprie prestazioni all'aumentare dei dati, mentre le applicazioni di Machine Learning una volta raggiunto un certo livello di performance non sono più scalabili. Ciò avviene perché le caratteristiche di un determinato oggetto vengono selezionate manualmente e servono per creare un modello in grado di categorizzare oggetti, mentre nei sistemi di Deep Learning l'estrazione delle caratteristiche avviene in modo automatico: la rete neurale apprende in modo autonomo come analizzare i dati e come svolgere un compito (ad esempio classificare un oggetto riconoscendone autonomamente le caratteristiche).

Reti Neurali

Le reti neurali sono basate sull'idea di riprodurre l'intelligenza, ed in particolare l'apprendimento, simulando la struttura neurale del cervello animale. Quest'ultimo utilizza un enorme rete di neuroni interconnessi per elaborare le informazioni e modellare il mondo che lo circonda. Un neurone raccoglie input da altri neuroni attraverso i dendriti, ovvero le fibre minori che si ramificano a partire dal neurone stesso. Esso somma tutti gli input ricevuti e, se il valore risultante è maggiore di una certa soglia, si attiva per propagare il segnale ad altri neuroni connessi attraverso l'assone.

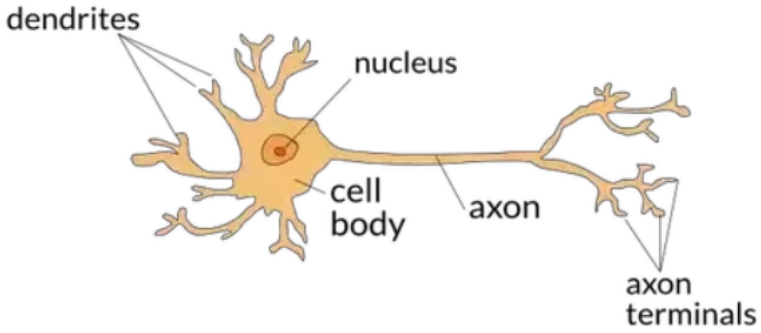


Figura 1.13: Conformazione di un neurone

L'idea di costruire una "macchina intelligente" a partire dai neuroni artificiali risale alla nascita dell'Intelligenza Artificiale, e già alcuni risultati furono ottenuti da McCulloch e Pitts nel 1943, quando nacque il primo modello neurale. Nel 1962 Rosenblatt propose un nuovo modello di neurone, il «percettrone», capace di apprendere mediante esempi.

Un percettrone descrive il funzionamento di un neurone eseguendo una somma pesata dei suoi ingressi ed emettendo un'uscita "1" se la somma è maggiore di un certo valore di soglia, "0" altrimenti. L'apprendimento, così inteso, è un processo di modifica dei valori dei pesi.

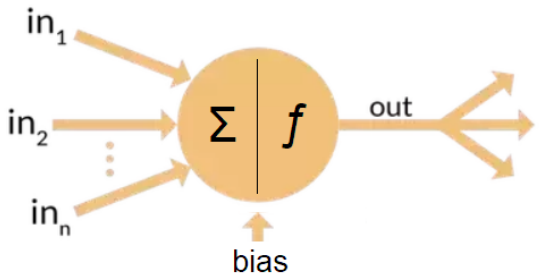


Figura 1.14: Funzionamento di un neurone

Struttura di una rete neurale

Una rete neurale è una rete divisa in tre livelli:

- **Input *layer***: in questo livello vengono trattati i dati attraverso una prima funzione (matematica) di attivazione che può introdurre la non linearità (tangente, coseno, sigmoide, logaritmo...);
- **Hidden *layer(s)***: eseguono una combinazione lineare (somma pesata) dell'output ottenuto dall'input *layer*; in una rete neurale possono esser presenti più hidden *layer*;
- **Output *layer***: presenta lo stesso funzionamento dell'hidden *layer* ed è il livello finale della rete; l'output di questi neuroni è il risultato dell'intera rete.

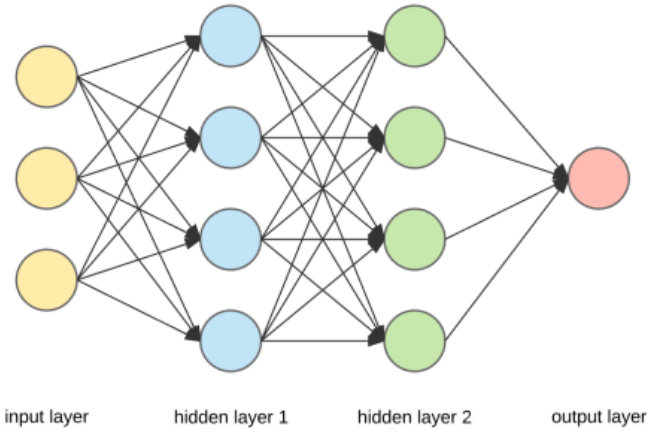


Figura 1.15: Struttura in *layer(s)* di una rete neurale

Con il Deep Learning vengono simulati i processi di apprendimento del cervello biologico attraverso le reti neurali artificiali per insegnare alle macchine non solo ad apprendere autonomamente, ma a farlo in modo più "profondo" (cioè su più livelli) proprio come sa fare il cervello umano. Le reti neurali tradizionali contengono 2-3 *layer* intermedi (hidden), mentre altre reti neurali più profonde ne possono contenere oltre 150.
Le reti neurali "profonde" sfruttano un numero maggiore di strati intermedi per costruire più livelli di astrazione. Quanti più livelli intermedi ci sono tanto più efficace sarà il risultato ma, di contro, la scalabilità della rete neurale è strettamente

legata ai dataset, ai modelli matematici e alle risorse computazionali.
I sistemi basati sul Deep Learning sono difficili da addestrare a causa del numero stesso di strati nella rete neurale. Il numero di strati e collegamenti tra neuroni è tale che può diventare arduo calcolare le regolazioni che devono essere apportate ai pesi in ogni fase del processo di addestramento. Comunemente si usano algoritmi di retro-propagazione dell'errore (*backpropagation*) attraverso i quali si modificano i pesi della rete neurale.
Gli algoritmi di *backpropagation* confrontano il valore in uscita del sistema con quello desiderato (obiettivo). Sulla base della differenza calcolata (errore), l'algoritmo modifica i pesi sinaptici della rete neurale, facendo convergere progressivamente il set dei valori di uscita verso quelli desiderati. L'aggiustamento del peso sinaptico è determinato dall'errore e da due parametri del sistema: il tasso di apprendimento e il momento. Entrambi i parametri possono variare da zero a uno (0 - 1). Il tasso di apprendimento influenza la velocità del processo apprendimento. Quanto più è elevato il tasso di apprendimento, vicino al valore massimo uno, tanto più veloce è il processo di apprendimento, in quanto sono più grandi le variazioni da apportare sul peso sinaptico. Tuttavia, un tasso di apprendimento molto alto aumenta anche il rischio delle variazione erratiche (oscillazioni del risultato) e, quindi, potrebbe portare all'instabilità del sistema. Per ridurre le oscillazioni e favorire la convergenza dei risultati si utilizza anche un secondo parametro (α), detto momento, anch'esso compreso tra zero e uno (0-1). La complessità dell'algoritmo di *backpropagation* cresce con la dimensione della rete neurale. Quando la rete neurale è composta da pochi neuroni si verifica il problema dell'*underfitting* e il processo di apprendimento della rete potrebbe essere inefficace. Viceversa, quando la rete è composta da molti neuroni si verifica il problema dell'*overfitting* facendo diventare più difficoltoso il processo di generalizzazione.

1.3 Problemi di ottimizzazione

Introduzione

Per circoscrivere la famiglia dei problemi di ottimizzazione combinatoria e le caratteristiche che la distinguono da altri insiemi di problemi, dobbiamo focalizzare l'attenzione sul tipo di soluzione che viene richiesta da ciascuna classe di problema: la tipologia della soluzione che si deve individuare per risolvere il problema permette di costruire una prima classificazione che aggrega in famiglie omogenee problemi che per ambito, applicazioni e strumenti di risoluzione risultano molto diversi fra loro.
La formulazione più essenziale in cui può essere posto un problema è quella in cui si chiede semplicemente di rispondere «sì o no», «vero o falso», «0 o 1». Ulteriori problemi, ad esempio, sono la ricerca di:

- uno zero della funzione $f(x)$ nell'intervallo (a, b);
- un cammino di lunghezza inferiore a k tra due vertici u e v del grafo G;
- un cammino chiuso sul grafo per raggiungere ogni vertice senza passare mai due volte per lo stesso spigolo;
- vincere una determinata partita a scacchi in sole tre mosse partendo da una certa configurazione dei pezzi sulla scacchiera.

In tutti problemi la cui soluzione è effettivamente un "semplice" «sì o no», «esiste o non esiste», «è possibile o non è possibile», non si richiede infatti di costruire ed esibire uno zero della funzione $f(x)$, un cammino sul grafo G o la sequenza di mosse che permette di vincere la partita a scacchi, ma soltanto di dire se uno zero, un cammino o una sequenza di mosse vincenti esiste. Questo genere di problemi sono denominati **problemi di decisione** o di **esistenza**[4]. Non solo non è richiesto di esibire una possibile configurazione delle variabili del problema a supporto della soluzione del problema stesso, ma spesso non è neanche necessario costruire

[4] `https://it.wikipedia.org/wiki/Ricerca_operativa`

la soluzione per poter formulare una risposta per una particolare istanza di un problema di decisione: è sufficiente articolare la risposta in modo tale che, al di là di ogni dubbio, questa risulti inconfutabile.

Nei **problemi di ricerca**, invece, viene richiesto di costruire una soluzione del problema, ossia si richiede di esibire esplicitamente una configurazione delle variabili del problema che diano luogo ad una soluzione. Rispetto agli esempi formulati in precedenza, potrebbe essere richiesto di identificare un particolare valore di $x \in (a, b)$ tale che $f(x) = 0$, oppure di costruire uno specifico cammino $p : uv$ sul grafo G con meno di k spigoli, o un circuito su G che non passi mai due volte per lo stesso spigolo, o la sequenza di mosse che permetta di vincere la partita a scacchi a partire da una certa disposizione dei pezzi sulla scacchiera.

Chiaramente una soluzione può essere esibita solo se esiste: in altre parole risolvere un'istanza di un problema nella sua formulazione come problema di ricerca, comporta implicitamente anche la soluzione dello stesso problema nella sua formulazione come problema di decisione.

Se viene richiesto di calcolare ed esibire tutte le possibili soluzioni di una determinata istanza del problema, allora siamo di fronte ad una formulazione del problema come **problema di enumerazione**. Ad esempio si richiede di identificare tutti i valori $x \in (a, b)$ tali che $f(x) = 0$; si richiede di costruire tutti i possibili cammini sul grafo G da u a v composti con meno di k spigoli; ecc. Anche in questo caso risolvere un problema di enumerazione significa, implicitamente, risolvere anche un problema di ricerca in cui si chiede di esibire una sola soluzione ammissibile per il problema ed anche risolvere il problema di decisione associato in cui si chiedeva soltanto di stabilire se una soluzione esiste o meno per quella determinata istanza del problema. Risolvere problemi di enumerazione sarà quindi più complicato che risolvere problemi di ricerca, e risolvere problemi di ricerca sarà più complicato che risolvere problemi di esistenza.

Infine vi sono i **problemi di ottimizzazione**: in questo caso non si dovrà né stabilire soltanto se una soluzione esiste, né identificare una soluzione qualsiasi per l'istanza del problema e nemmeno identificare tutte le soluzioni del problema stesso, indistintamente. È richiesto invece identificare una o tutte le soluzioni

che risultano ottimali rispetto ad un qualche criterio di valutazione delle soluzioni ammissibili per una specifica istanza del problema. Ad esempio, tra tutti gli zeri della funzione $f(x)$ nell'intervallo (a, b) sarà richiesto individuare quelli che rendono minima la funzione $\varphi(x)$ nell'intervallo stesso; tra tutti i cammini che sul grafo G collegano i vertici u e v trovare quelli di lunghezza minima o che minimizzano il costo dato dalla somma dei pesi attribuiti a ciascuno spigolo del grafo, e così via. In un problema di ottimizzazione viene quindi fornita anche una funzione con cui valutare il costo di ciascuna soluzione del problema: questa funzione, detta *funzione obiettivo*, costituisce un criterio di scelta tra le possibili soluzioni del problema, selezionando solo le soluzioni migliori, quelle che "minimizzano" il costo o massimizzano il "profitto".

Classe	**Descrizione**
Problemi di decisione o di esistenza	Verificare se esiste o meno una certa soluzione per l'istanza del problema
Problemi di ricerca	Costruire ed esibire una soluzione per l'istanza del problema
Problemi di enumerazione	Costruire ed esibire tutte le soluzioni per l'istanza del problema
Problemi di ottimizzazione	Costruire le soluzioni ottimali in base ad un determinato criterio (funzione obiettivo o di costo)

Tabella 1.3: Classificazione dei problemi in base al tipo di soluzione richiesta

Formulazione generale dei problemi di ottimizzazione

In termini generali un'istanza di un problema di ottimizzazione è definita come una coppia (A, f), dove A è l'**insieme delle soluzioni ammissibili** ed $f : A \to R$ è la **funzione obiettivo** che si deve ottimizzare (massimizzare o minimizzare). Il problema di ottimizzazione viene posto chiedendo di trovare i valori $x \in A$ tali che $f(x) \leq f(y)$ per ogni $y \in A$ (problema in forma di minimizzazione). Tali valori

x costituiscono **soluzioni ottime globali** per il problema; nel caso in cui risulti invece $f(x) \leq f(y)$ soltanto per ogni y appartenente ad un intorno di x, allora la soluzione x si definisce come **soluzione ottima locale**.
L'insieme A delle soluzioni ammissibili è l'insieme entro il quale devono essere cercate le soluzioni che rendono minima o massima la funzione obiettivo: la definizione di A è una caratteristica specifica di ciascun problema e consente di escludere punti dello spazio su cui sono definite le variabili del problema su cui non è utile cercare la soluzione. L'insieme A non è necessariamente un sottoinsieme o un intervallo della retta reale: spesso, infatti, l'insieme A è un sottoinsieme di $\mathbb{R}^n$ o un insieme discreto definito in modo più articolato, come ad esempio un sotto grafo con determinate caratteristiche.
Nella formulazione di un problema di ottimizzazione un aspetto molto importante è costituito dalla corretta definizione dell'insieme delle soluzioni ammissibili. È necessario circoscrivere questo sottoinsieme dello "spazio delle soluzioni" senza escludere punti dello spazio che possano corrispondere ad una soluzione ammissibile e dunque potenzialmente ottima; al tempo stesso, riducendo al massimo l'insieme entro cui eseguire la ricerca delle soluzioni, per rendere più efficiente e meno oneroso, in termini computazionali, l'algoritmo per individuare le soluzioni ottime. In generale, quindi, l'insieme A delle soluzioni ammissibili viene definito attraverso un insieme di **vincoli** che limitano e circoscrivono l'insieme entro cui possono assumere valori le variabili del problema.

Problema dell'assegnazione di task

Un problema di pianificazione riguarda, ad esempio, l'assegnazione di task ai processori di un calcolatore parallelo, alle macchine di una rete di calcolatori che cooperano per la soluzione di uno stesso problema o ad un gruppo di persone. Si supponga che n processori debbano svolgere m task. Ciascuno di essi deve essere svolto esattamente da un processore ed inoltre, ciascun processore può svolgere al massimo un task. Indicando con c_{ij} il costo dell'utilizzo del processore i per l'esecuzione del task j. Il problema chiede di assegnare i task alle varie risorse (processori, calcolatori o persone, a seconda del contesto applicativo) minimizzan-

do il costo totale per la realizzazione di tutti gli m task previsti. Per formulare questo problema in termini di programmazione lineare intera, è necessario introdurre le variabili binarie x_{ij}, per $i = 1, ..., n$ e $j = 1, ..., m$ corrispondenti all'evento "ij" (esecuzione del task j da parte della risorsa i) definite come segue $x_{ij} = 1$ se il task j viene eseguito dalla risorsa i e $x_{ij} = 0$ altrimenti. Poiché esattamente una ed una sola risorsa deve essere assegnata al task j, si avranno i seguenti vincoli:

$$\sum_{i=1}^{n} x_{ij} = 1, \text{ per ogni } j = 1,2,\ldots,m$$

Al tempo stesso, dato che per ipotesi ogni risorsa non può svolgere più di un task, si avranno anche i seguenti vincoli:

$$\sum_{j=1}^{m} x_{ij} = 1, \text{ per ogni } i = 1,2,\ldots,n$$

È possibile quindi verificare che un vettore $x \in 0, 1$ che soddisfi tutti i vincoli sopra descritti individua un assegnamento ammissibile di risorse ai task. La funzione obiettivo, che deve essere minimizzata, è la seguente:

$$\sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij} x_{ij}$$

Algoritmi euristici

Dato che molti problemi di ottimizzazione sono definiti come "difficili", è spesso necessario sviluppare algoritmi euristici, ossia algoritmi che non garantiscono di ottenere una soluzione ottima, ma in generale sono in grado di fornire una "buona" soluzione ammissibile per il problema. Normalmente gli algoritmi euristici hanno una ridotta complessità in tempo, ma in alcuni casi, per problemi di grandi dimensioni e struttura complessa, può essere necessario sviluppare algoritmi euristici sofisticati e di elevata complessità. Inoltre, è possibile, in generale, che un

algoritmo euristico "fallisca" e non sia in grado di determinare nessuna soluzione ammissibile del problema, pur senza essere in grado di dimostrare che non ne esistano.

Algoritmi greedy

Gli algoritmi *greedy* determinano una soluzione attraverso una sequenza di decisioni "localmente ottime", senza mai tornare sulle decisioni prese. Questi algoritmi sono di facile implementazione e notevole efficienza computazionale ma, sia pure con alcune eccezioni, in generale non garantiscono l'ottimalità, e a volte neppure l'ammissibilità, della soluzione trovata.
È comunque possibile definire uno schema generale di algoritmo greedy, adatto a tutti quei casi in cui l'insieme ammissibile può essere rappresentato come una famiglia $F \subset 2^E$ di sottoinsiemi di un dato insieme "base" E.

```
procedure Greedy( E , F , S )  {
  S := ∅; Q = E;
  do { e = Best( Q ); Q = Q \ { e };
       if( S ∪ { e } ∈ F ) then S = S ∪ { e }
       } while( Q ≠ ∅ and not Maximal( S ) )
  }
```

Figura 1.16: Pseudo-codice algoritmo *greedy*

Nella procedura, S è l'insieme degli elementi di E che sono stati inseriti nella soluzione (parziale) corrente e Q è l'insieme degli elementi di E ancora da esaminare: gli elementi in $E \setminus (S \cup Q)$ sono quelli per cui si è deciso che non faranno parte della soluzione finale. La sotto procedura Maximal restituisce vero se non è più possibile aggiungere elementi alla soluzione S, per cui l'algoritmo può terminare senza esaminare gli elementi eventualmente ancora presenti in Q. In molti casi, le soluzioni ammissibili del problema sono solamente gli elementi "massimali" di F: quindi, se l'algoritmo greedy termina avendo esaurito gli elementi di Q senza che Maximal restituisca vero, allora l'algoritmo "fallisce", ossia non è in grado di determinare una soluzione ammissibile del problema. La sotto procedura Best

fornisce il miglior elemento di E tra quelli ancora in Q sulla base di un prefissato criterio, ad esempio l'elemento di costo minimo nel caso di problemi di minimo.

Capitolo 2

Progettazione

2.1 Obiettivi

L'obiettivo del progetto svolto è stato quello di fornire al designatore degli arbitri di pallacanestro un possibile supporto alle decisioni. Il lavoro è suddiviso in due componenti:

1. Predizione della difficoltà delle gare;

2. Assegnamento automatico degli arbitri.

La prima componente di predizione della difficoltà, focus di questo elaborato, avviene sfruttando i dati pregressi a disposizione ed è fondamentale, in quanto propedeutica alla seconda, ovvero le designazioni automatiche. Ciò è dato dal fatto che gli arbitri devono essere designati dal sistema solamente se in grado di affrontare una gara, a seconda del livello di difficoltà di quest'ultima e del loro livello d'esperienza. Lo scopo di questo vincolo è far sì che arbitri, magari alle prime armi con la Serie A, non vengano proposti per arbitrare gare estremamente difficili o decisive, al fine di evitare evidenti problemi di gestione della gara.

2.2 Architettura

Lo schema generale dell'architettura può essere così rappresentato con le due componenti principali:

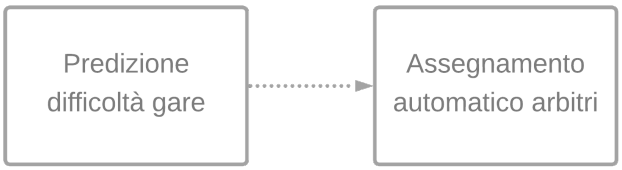


Figura 2.1: Architettura generale del progetto

Le due componenti sono state sviluppate singolarmente e non è stato implementato il collegamento tra le due, il quale è lasciato agli sviluppi futuri; tuttavia, la loro progettazione è stata ideata per poterle combinare facilmente in futuro.
La prima componente della predizione delle gare è articolata nelle seguenti fasi:

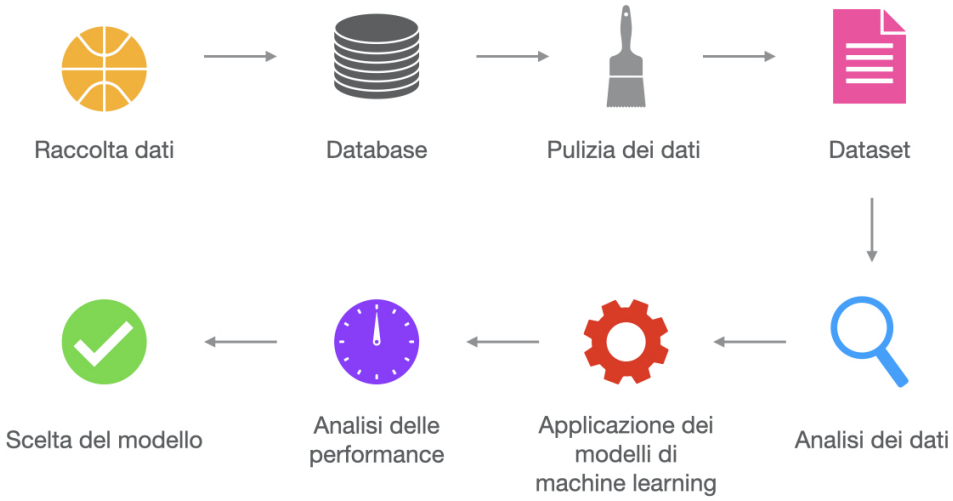


Figura 2.2: Fasi della predizione della difficoltà delle gare

La prima fase del lavoro comprende dalla raccolta dei dati fino alla costruzione di appositi dataset. Nel corso di questo capitolo verranno quindi approfonditi gli aspetti riguardanti:

1. le statistiche della pallacanestro disponibili in rete per la Serie A italiana;

2. la progettazione della base di dati in cui sono state raccolte;

3. le modalità stesse di raccolta;

4. l'etichettatura delle gare in base alla loro difficoltà;

5. l'analisi ed elaborazione dei dati;

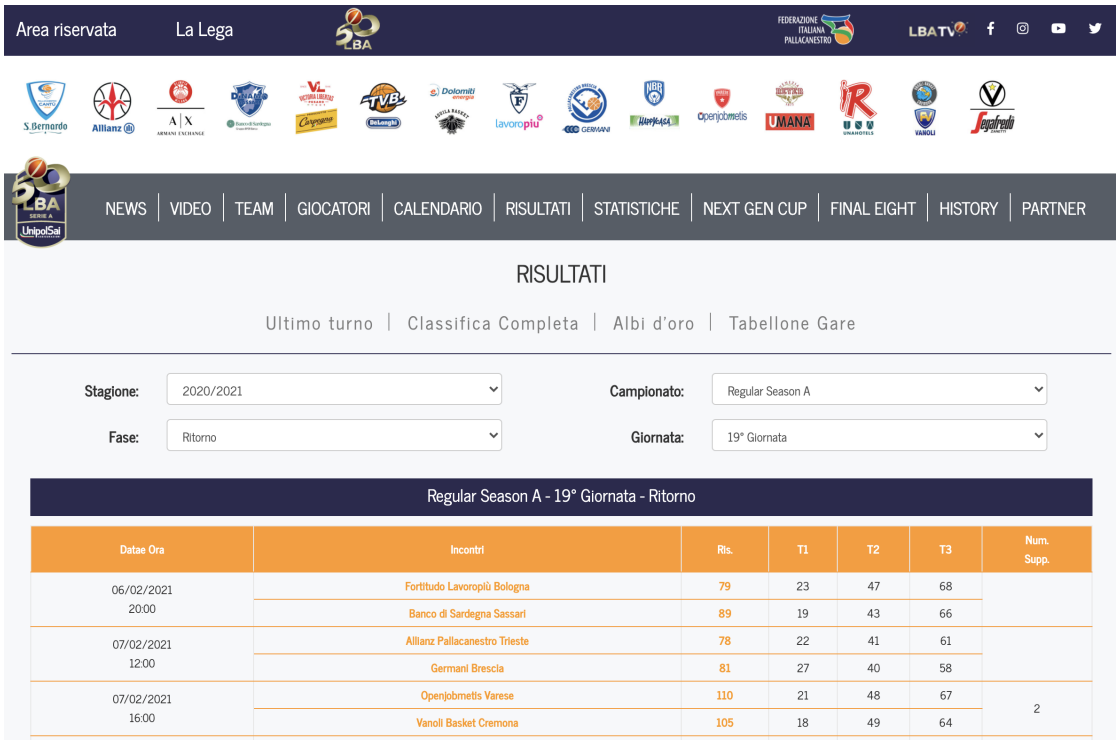
6. la creazione di diversi dataset di lavoro.

2.3 Statistiche Lega Basket

La raccolta più ampia e completa sulle statistiche di pallacanestro della Serie A italiana è contenuta nel sito pubblico e online della Lega Basket[1]. Quest'ultimo contiene tutte le statistiche, stagione sportiva per stagione sportiva, dalla 1948/1949 fino all'ultima, 2020/2021. Le stagioni degli ultimi anni sono chiaramente più complete, e comprendono ad esempio anche le statistiche dei singoli quarti di ogni gara e di ogni giocatore per entrambe le squadre, gli arbitri, le classifiche complete e le informazioni principali sui giocatori. Di seguito si mostrano gli screenshot delle pagine più rilevanti da cui si sono raccolti i dati.

Nella pagina iniziale vi sono i risultati di tutte le gare, suddivisi per: stagione sportiva (es. "2020/2021"), campionato ("Regular Season" oppure "Playoff"), fase (es. "Andata" o "Ritorno") e giornata (es. "1° giornata"). Per ogni gara è riportata la data, l'ora, le squadre partecipanti, il risultato finale, i risultati dei singoli quarti e l'eventuale numero di supplementari.

[1] `http://www.legabasket.it/`



Datae Ora	Incontri	Ris.	T1	T2	T3	Num. Supp.
06/02/2021 20:00	Fortitudo Lavoropiù Bologna	79	23	47	68	
	Banco di Sardegna Sassari	89	19	43	66	
07/02/2021 12:00	Allianz Pallacanestro Trieste	78	22	41	61	
	Germani Brescia	81	27	40	58	
07/02/2021 16:00	Openjobmetis Varese	110	21	48	67	2
	Vanoli Basket Cremona	105	18	49	64	

Figura 2.3: Pagina "Risultati" del sito Lega Basket Serie A

In ogni singola gara sono poi presenti ulteriori dettagli, come il campo di gioco in cui si è svolta (impianto), i progressivi dei singoli quarti e gli arbitri che l'hanno diretta:

Figura 2.4: Pagina di una singola gara (dati generali) del sito Lega Basket Serie A

All'interno della stessa pagina vi sono anche le statistiche complete, di entrambe le squadre, relative ai singoli giocatori, come ad esempio: numero di punti realizzati, minuti giocati, falli, stoppate, assist, tiri da 2pt. e 3pt. e altre valutazioni inerenti alle prestazioni nella gara (Voto Lega, Voto OER e Plus/Minus). Tutte queste statistiche vengono analizzate nel dettaglio nel capitolo successivo.

Accedi al play by play della partita | Intera Partita | 1° quarto | 2° quarto | 3° quarto | 4° quarto

UNAHOTELS Reggio Emilia

UNAHOTELS Reggio Emilia				Falli		Tiri da 2			Sc	Tiri da 3			Tiri Liberi			Rimbalzi			Stoppate		Palle		Ass	Valutaz.		+/-
All.Martino Antimo	Pt	Min	C	S	R	T	%		R	T	%	R	T	%	Off	Dif	Tot	Dat	Sub	Per	Rec		Lega	OER	
* 2 SUTTON Dominique	0	4	1	0	0	2	0.0	0	0	0	0.0	0	0	0.0	0	0	0	0	0	0	0	0	-3	0.00	-8
3 KOPONEN Petteri	8	20	0	1	1	1	100.0	0	2	6	33.3	0	0	0.0	0	0	0	0	0	0	0	3	8	1.14	-15
* 5 BOSTIC Joshua	11	33	1	3	0	2	0.0	0	3	9	33.3	2	3	66.7	0	3	3	0	1	2	1	2	7	0.76	-26
* 7 CANDI Leonardo	12	25	3	3	1	4	25.0	0	2	5	40.0	4	4	100.0	0	1	1	0	0	3	2	1	7	0.86	-18
* 8 BALDI ROSSI Filippo	14	32	2	1	4	7	57.1	0	2	2	100.0	0	0	0.0	2	3	5	0	0	0	2	3	20	1.56	-7
9 PORFILIO Carlo	0	1	0	0	0	0	0.0	0	0	0	0.0	0	0	0.0	0	0	0	0	0	0	0	0	0	0.00	0
* 11 TAYLOR Brandon	10	34	1	5	2	4	50.0	0	2	8	25.0	0	0	0.0	0	0	0	0	0	2	5	4	13	0.71	-17
14 GIANNINI Marco	0	1	0	0	0	0	0.0	0	0	0	0.0	0	0	0.0	0	0	0	0	0	0	0	0	0	0.00	0
32 BONACINI Federico	0	3	0	0	0	0	0.0	0	0	0	0.0	0	0	0.0	0	0	0	0	0	1	1	0	0	0.00	-6
35 DIOUF Mouhamet Rassoul	4	25	3	2	2	4	50.0	0	0	0	0.0	0	2	0.0	2	4	6	1	0	3	0	0	3	0.50	-26
77 KYZLINK Tomas	3	22	4	3	1	3	33.3	0	0	1	0.0	1	2	50.0	0	4	4	0	0	1	0	2	3	0.50	-12
Squadra	0	0	0	0	0	0	0.0	0	0	0	0.0	0	0	0.0	2	2	4	0	0	0	0	0	4	0.00	
Totali	62	200	15	18	11	27	40.7	0	11	31	35.5	7	11	63.6	6	17	23	1	1	12	11	15	62	0.82	

Virtus Segafredo Bologna

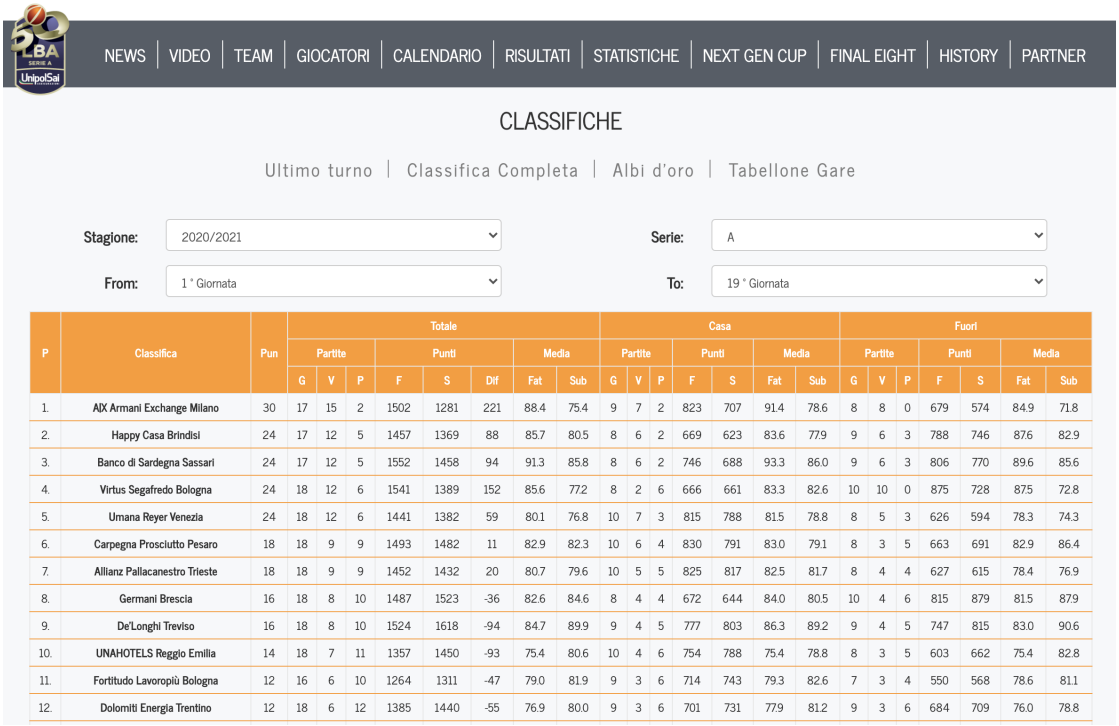
Virtus Segafredo Bologna			Falli		Tiri da 2			Sc	Tiri da 3			Tiri Liberi			Rimbalzi			Stoppate		Palle		Ass	Valutaz.		+/-
All. Djordjevic Aleksandar	Pt	Min	C	S	R	T	%		R	T	%	R	T	%	Off	Dif	Tot	Dat	Sub	Per	Rec		Lega	OER	
0 TESSITORI Amedeo	6	10	2	3	2	6	33.3	0	0	0	0.0	2	2	100.0	1	2	3	0	0	1	0	2	7	0.75	13
1 DERI Lorenzo	0	2	1	0	0	0	0.0	0	0	1	0.0	0	0	0.0	0	0	0	0	0	0	0	0	-2	0.00	4
3 BELINELLI Marco	0	0	0	0	0	0	0.0	0	0	0	0.0	0	0	0.0	0	0	0	0	0	0	0	0	0	0.00	0
7 ALIBEGOVIC Amar	4	7	0	0	2	2	100.0	0	0	0	0.0	0	0	0.0	1	1	2	0	0	1	0	0	5	1.33	10
* 9 MARKOVIC Stefan	2	25	0	3	0	1	0.0	0	0	2	0.0	2	2	100.0	0	3	3	0	0	0	1	9	15	0.50	18
* 11 RICCI Giampaolo	7	25	3	0	2	3	66.7	0	1	2	50.0	0	0	0.0	2	4	6	0	0	2	0	1	7	1.00	14
14 ADAMS Josh	5	24	2	1	1	3	33.3	0	1	5	20.0	0	0	0.0	1	3	4	0	0	1	0	2	3	0.56	28
32 HUNTER Vince	20	18	2	3	5	6	83.3	0	2	2	100.0	4	5	80.0	5	5	10	0	0	0	1	1	31	1.90	12
* 34 WEEMS Kyle	11	29	3	1	3	5	60.0	0	1	1	100.0	2	2	100.0	0	2	2	0	0	0	1	2	12	1.57	9
* 44 TEODOSIC Milos	11	22	1	1	5	6	83.3	0	0	2	0.0	1	2	50.0	1	1	2	0	0	4	0	7	12	0.85	8
* 45 GAMBLE Julian	14	16	2	1	7	8	87.5	4	0	0	0.0	0	0	0.0	0	6	6	0	1	1	0	2	18	1.56	4
55 ABASS Abass Awudu	9	22	2	2	4	7	57.1	2	0	1	0.0	1	2	50.0	0	3	3	1	0	1	2	0	9	0.90	15
Squadra	0	0	0	0	0	0	0.0	0	0	0	0.0	0	0	0.0	0	1	1	0	0	0	0	0	1	0.00	
Totali	89	200	18	15	31	47	66.0	6	5	16	31.3	12	15	80.0	11	31	42	1	1	11	5	26	118	1.09	

*N; Quintetto iniziale OER Rendimento Offensivo R Tiri Realizzati T Tiri Tentati Sc Schiacciate

Figura 2.5: Pagina di una singola gara (statistiche squadra casa e ospite) del sito Lega Basket Serie A

Un'altra importante componente è rappresentata dalle classifiche, dove per ogni

gara è riportata la posizione, i punti e altri dettagli come il numero di partite giocate, vinte, perse (totali, in casa e fuori). Anche qui viene organizzato, come per la sezione dei risultati vista precedentemente, in stagioni sportive e serie. Inoltre, è possibile selezionare le singole giornate d'interesse (ad es. dalla "1° Giornata" alla "19° Giornata").



P	Classifica	Pun	Totale								Casa							Fuori						
			Partite			Punti			Media		Partite			Punti		Media		Partite			Punti		Media	
			G	V	P	F	S	Dif	Fat	Sub	G	V	P	F	S	Fat	Sub	G	V	P	F	S	Fat	Sub
1.	AIX Armani Exchange Milano	30	17	15	2	1502	1281	221	88.4	75.4	9	7	2	823	707	91.4	78.6	8	8	0	679	574	84.9	71.8
2.	Happy Casa Brindisi	24	17	12	5	1457	1369	88	85.7	80.5	8	6	2	669	623	83.6	77.9	9	6	3	788	746	87.6	82.9
3.	Banco di Sardegna Sassari	24	17	12	5	1552	1458	94	91.3	85.8	8	6	2	746	688	93.3	86.0	9	6	3	806	770	89.6	85.6
4.	Virtus Segafredo Bologna	24	18	12	6	1541	1389	152	85.6	77.2	8	2	6	666	661	83.3	82.6	10	10	0	875	728	87.5	72.8
5.	Umana Reyer Venezia	24	18	12	6	1441	1382	59	80.1	76.8	10	7	3	815	788	81.5	78.8	8	5	3	626	594	78.3	74.3
6.	Carpegna Prosciutto Pesaro	18	18	9	9	1493	1482	11	82.9	82.3	10	6	4	830	791	83.0	79.1	8	3	5	663	691	82.9	86.4
7.	Allianz Pallacanestro Trieste	18	18	9	9	1452	1432	20	80.7	79.6	10	5	5	825	817	82.5	81.7	8	4	4	627	615	78.4	76.9
8.	Germani Brescia	16	18	8	10	1487	1523	-36	82.6	84.6	8	4	4	672	644	84.0	80.5	10	4	6	815	879	81.5	87.9
9.	De'Longhi Treviso	16	18	8	10	1524	1618	-94	84.7	89.9	9	4	5	777	803	86.3	89.2	9	4	5	747	815	83.0	90.6
10.	UNAHOTELS Reggio Emilia	14	18	7	11	1357	1450	-93	75.4	80.6	10	4	6	754	788	75.4	78.8	8	3	5	603	662	75.4	82.8
11.	Fortitudo Lavoropiù Bologna	12	16	6	10	1264	1311	-47	79.0	81.9	9	3	6	714	743	79.3	82.6	7	3	4	550	568	78.6	81.1
12.	Dolomiti Energia Trentino	12	18	6	12	1385	1440	-55	76.9	80.0	9	3	6	701	731	77.9	81.2	9	3	6	684	709	76.0	78.8

Figura 2.6: Pagina "Classifiche" del sito Lega Basket Serie A

Nella pagina di ogni singola squadra, sono presenti ulteriori informazioni circa il roster della stessa e, per ogni giocatore, le informazioni principali, quali ad esempio ruolo, anno di nascita, altezza e nazionalità.

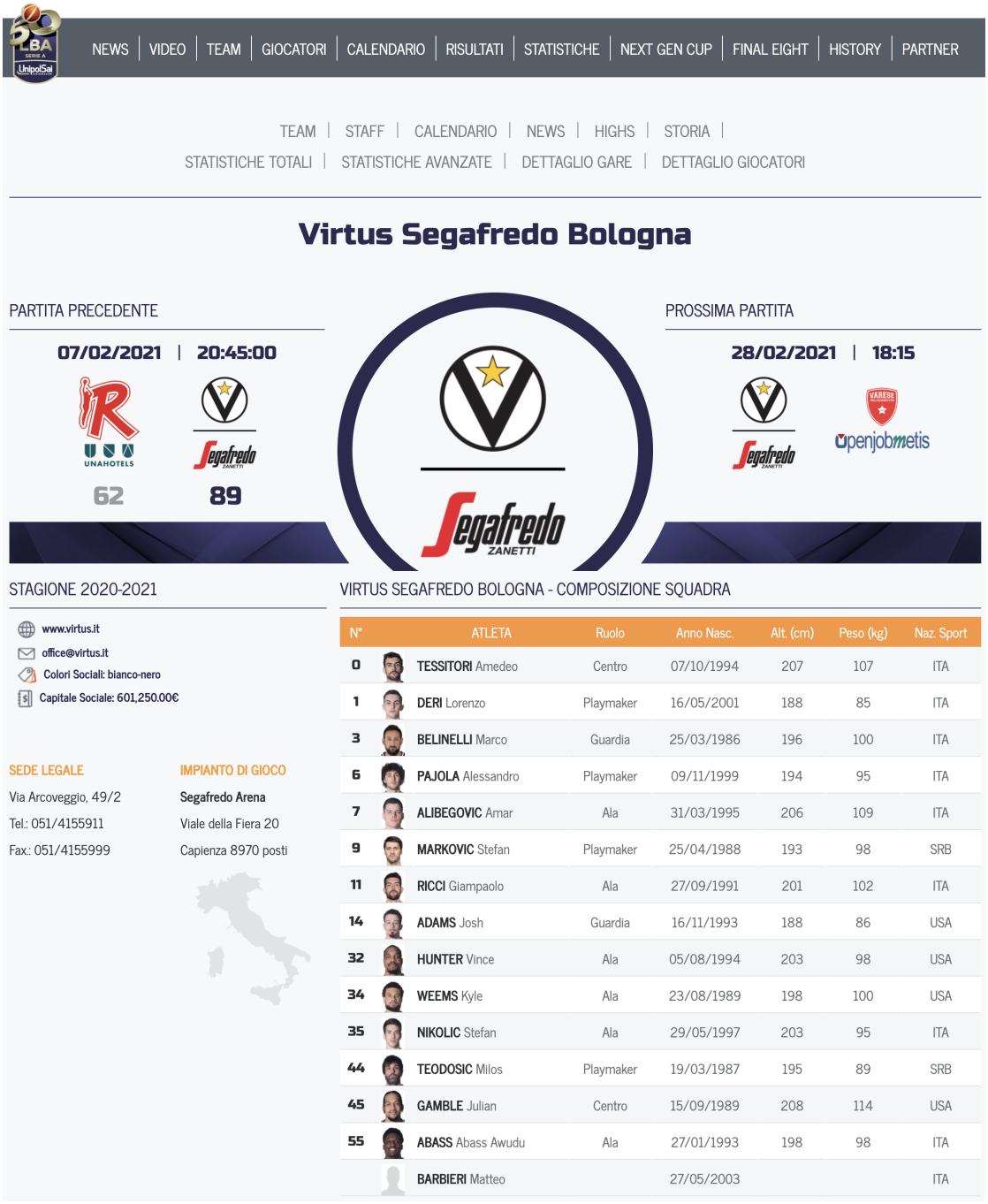


Figura 2.7: Pagina descrittiva di una squadra del sito Lega Basket Serie A

2.4 Database

Alla luce delle statistiche disponibili si è passati poi alla progettazione della base di dati che le dovesse contenere; si è optato quindi per un modello relazionale.
La base di dati è stata costruita al fine di contenere tutte le statistiche presenti sul sito della Lega Basket mostrate precedentemente. Inoltre, la strutturazione delle tabelle è stata pensata per rendere efficace l'interrogazione delle stesse, per mezzo di apposite *query* e viste. In particolare, è stata riproposta l'organizzazione in stagioni sportive/fasi/giornate, così da poter recuperare e, successivamente, elaborare i dati il modo puntuale rispetto ad ogni singola gara.
La tabella "gara" è la principale, dove sono memorizzati tutti i dati principali, come il risultato e gli arbitri. La tabella "giocatore" contiene tutte le informazioni peculiari sui giocatori, come ruolo, altezza e nazionalità. Vi è poi una triade identificata dalle tabelle "gara"-"giocatore"-"statistica", dove, in "giocatore_has_statistica_has_gara", vengono salvate le partecipazioni dei giocatori ad ogni gara con le relative statistiche. Queste ultime sono contenute nella tabella "statistica", dove risiedono tutte le informazioni, nel dettaglio:

- Se il giocatore era nel quintetto iniziale;
- Punti fatti;
- Minuti giocati;
- Falli (fatti e subiti);
- Tiri da 2pt. (totali, realizzati, percentuale);
- Tiri da 3pt. (totali, realizzati, percentuale);
- Tiri Liberi (totali, realizzati, percentuale);
- Rimbalzi (offensivi, difensivi);
- Stoppate (date, subite);

- Palle perse e recuperate;
- Assist;
- Voto Lega[2] (votazione attribuita dalla Lega Basket in base alle prestazioni del giocatore in gara);
- Voto OER[2] (punteggio rappresentativo dell'"*Offensive Efficiency Rating*", ovvero il "Coefficiente di Efficacia Offensiva");
- Plus/Minus[3] (espressione numerica dell'apporto di un giocatore alla gara, tanto più è alto e migliore è stata la performance del giocatore).

Infine, la tabella "classifica" contiene tutte classifiche disponibili, nello specifico:

- Posizione;
- Punti;
- Numero gare giocate, vinte e perse (totali, in casa e fuori casa);
- Punti fatti e subiti (totali, in casa e fuori casa);
- Media punti fatti e subiti (totali, in casa e fuori casa).

Poiché nel sito della Lega Basket sono presenti, oltre alle statistiche dei giocatori, anche quelle di squadra e totali, queste sono state inserite nella tabella "giocatore", preceduti dalla chiave primaria della squadra, con apposito campo di controllo per distinguerle (denominato "Giocatore" e impostato a *false* in questo caso). Ad esempio "15 – Squadra", oppure "15 – Totali", dove "15" è la chiave primaria della squadra (es. "Sassari").
Di seguito viene riportato il modello EER, realizzato tramite il *software* MySQL-Workbench[4].

[2] `http://www.fip.it/public/41/6017/raffaele%20imbrogno%20-%20statistica%20e%20pallacanestro.pdf`

[3] `https://it.wikipedia.org/wiki/Plus/minus`

[4] `https://www.mysql.com/it/products/workbench/`

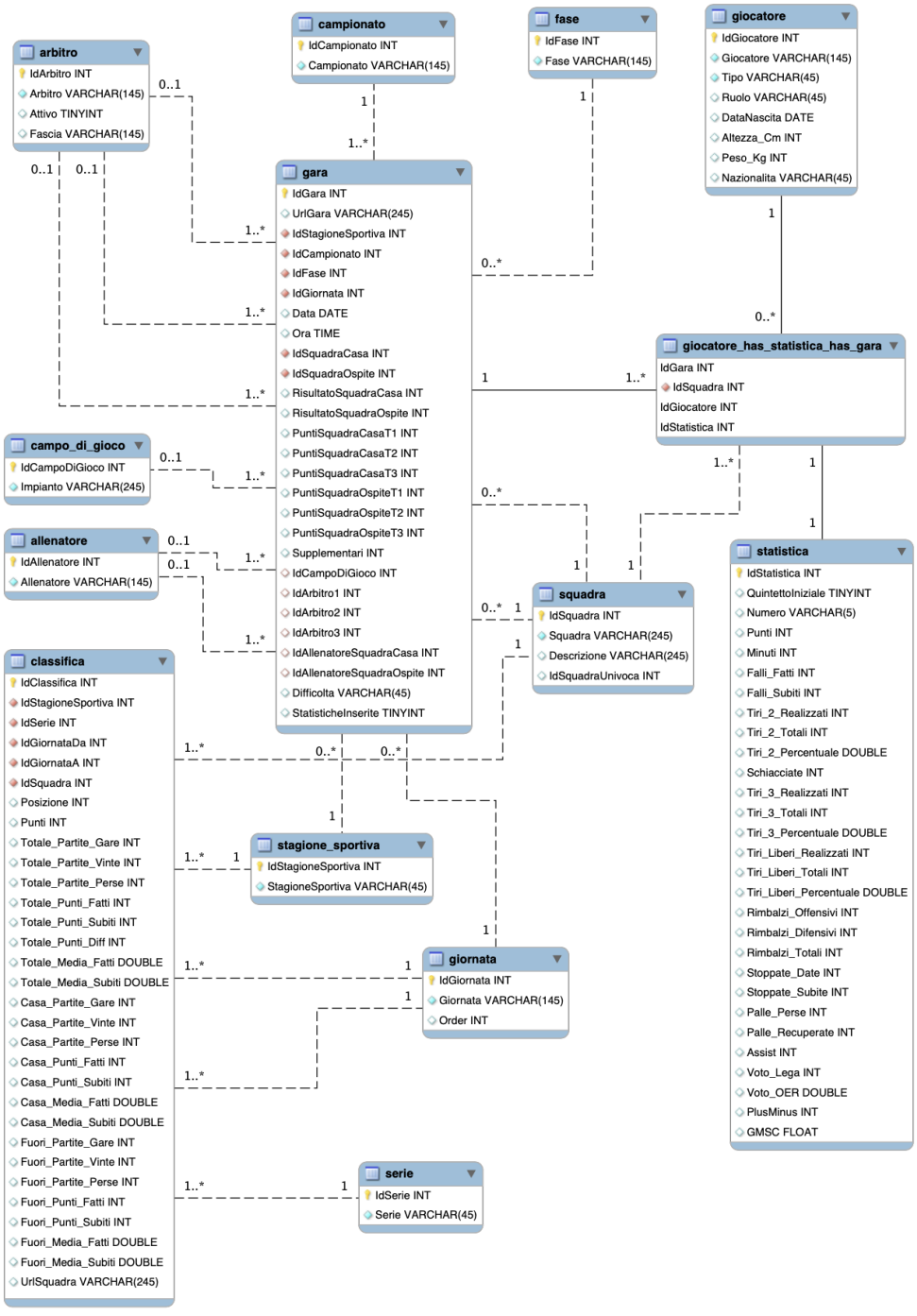


Figura 2.8: Schema EER

2.5 Raccolta dati

Una volta individuati i dati da utilizzare, aver progettato e realizzato la base di dati, si è costruito un algoritmo che fosse in grado di scaricare e memorizzare automaticamente tali dati. Si è optato per UiPath Studio[5], una tra le migliori e note piattaforme professionali e non per l'automazione di processi robotici.

2.5.1 UiPath Studio

UiPath Studio permette di creare processi robotici automatizzati. Nel nostro caso, abbiamo creato un processo che andasse a raccogliere i dati richiesti in modo sistematico, e li salvasse nel database. In UiPath Studio è possibile creare *workflow* in cui inserire apposite *activity*, a seconda del task da svolgere. L'esecuzione principale del programma ("*main*") attraversa diversi *workflow* appositamente creati, nel seguente ordine:

- Classifica (in cui vengono raccolte tutte le classifiche);
- Risultati gare (in cui vengono raccolti tutti i risultati delle partite);
- Statistiche giocatori (in cui vengono raccolte le statistiche dei giocatori nelle gare);
- Informazioni giocatori (in cui vengono raccolte le principali informazioni di tutti i giocatori di ogni squadra).

Questa struttura ci ha consentito di raccogliere, dopo aver investito diverso tempo nella fase di progettazione, tutti i dati necessari; compito che, invece, manualmente non sarebbe stato possibile (o con un tempo estremamente elevato). Di seguito si mostrano alcune schermate principali.
La prima parte del *main* contiene il workflow della Classifica:

[5] `https://www.uipath.com/product/studio`

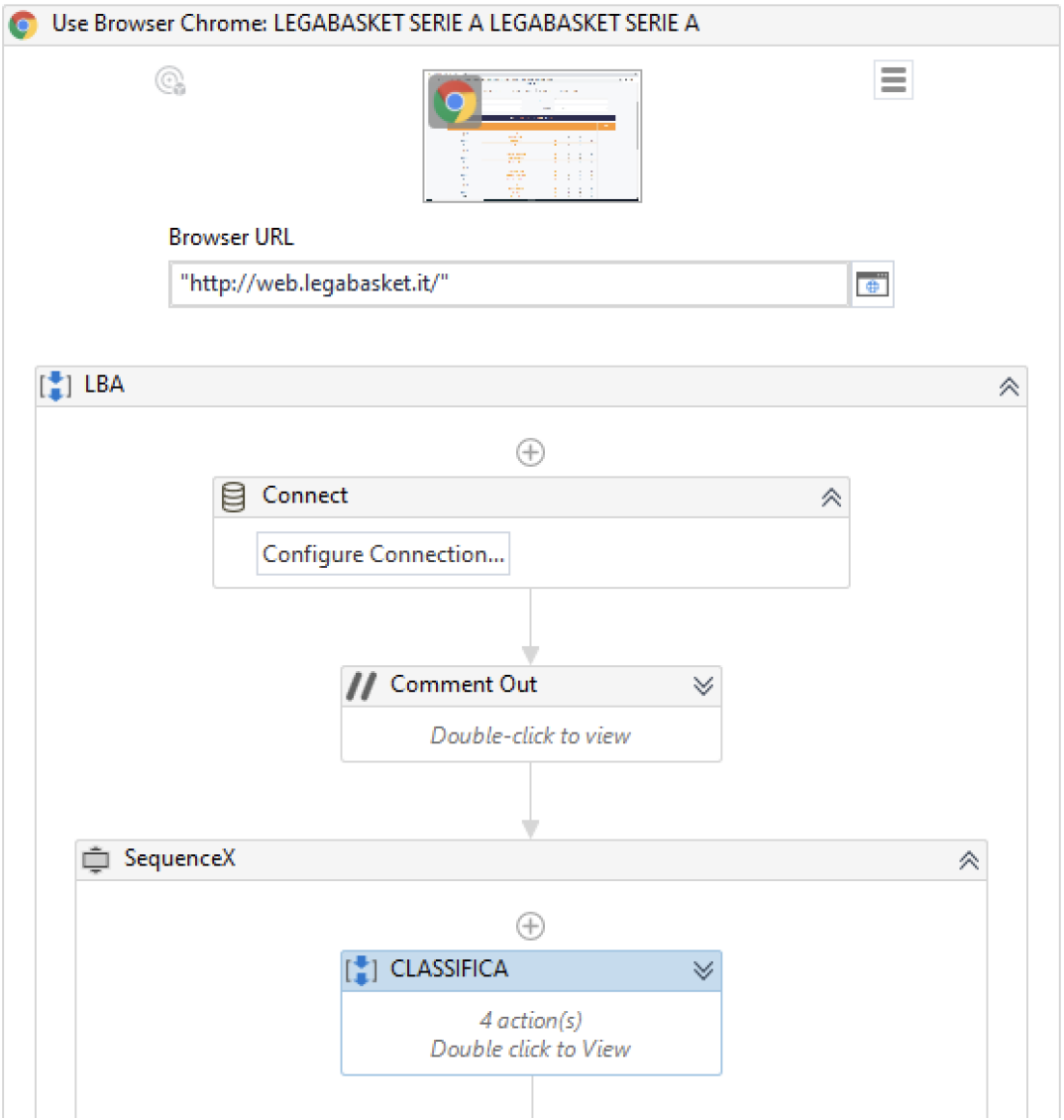


Figura 2.9: Schermata "*main*" del progetto UiPath Studio

In questo passaggio si nota come il *workflow* "CLASSIFICA" sia contenuto all'interno di un'*activity* "Browser", la quale permette di utilizzare una serie di strumenti e facilitazioni per, ad esempio, il reperimento dei valori all'interno delle *select-box*, la selezione di elementi, il click del mouse e altri ancora.
All'interno della sezione dedicata alla Classifica, viene mostrata la navigazione tra

le diverse pagine e la selezione degli elementi dalle *select-box* (ad esempio quelle delle stagioni sportive o giornate):

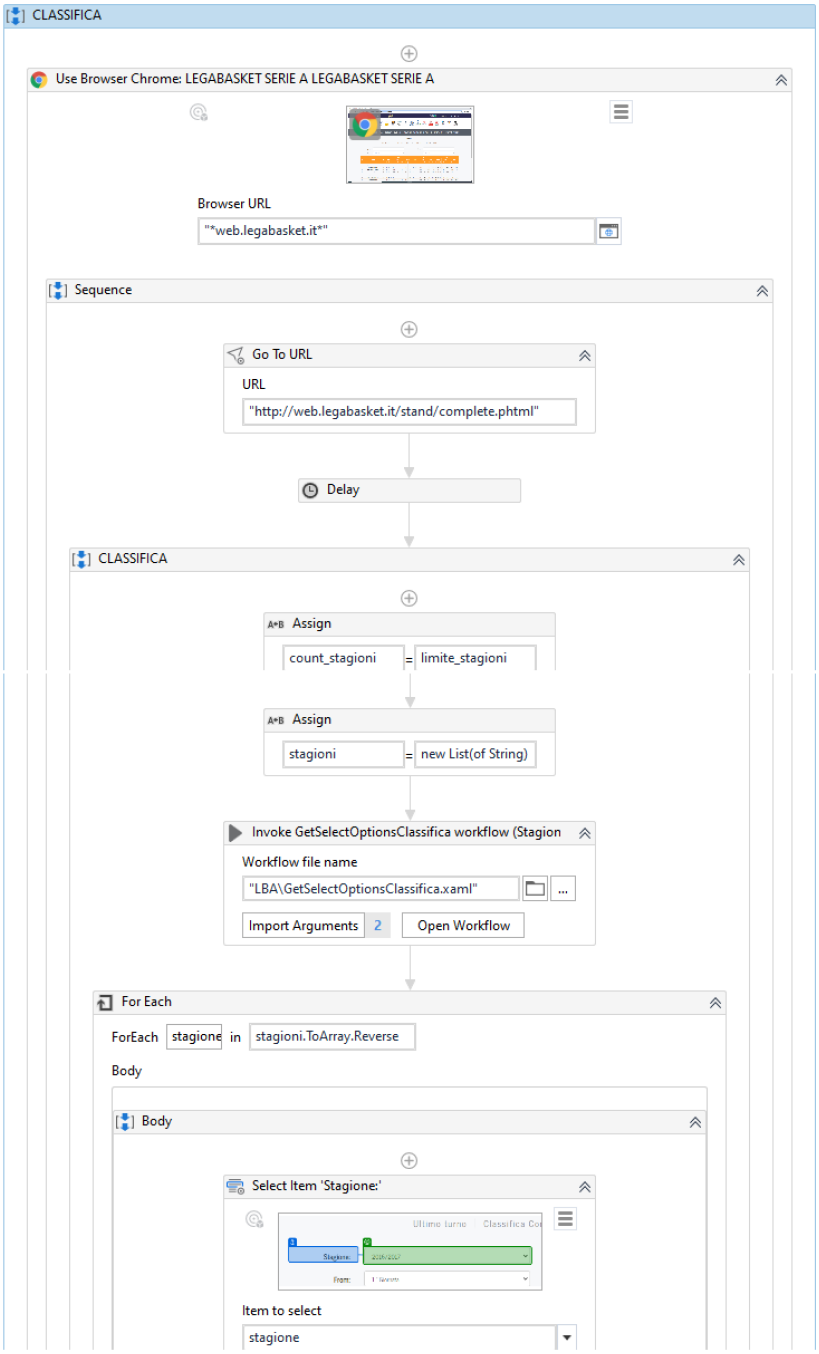


Figura 2.10: Schermata "*CLASSIFICA*" del progetto UiPath Studio

L'utilizzo dell'*activity* "Go To URL" ha permesso di visitare una specifica pagina web, alla quale segue un "Delay" che permettesse il corretto caricamento di tutti gli elementi grafici. Successivamente, viene invocato un altro *workflow* che si occupa di reperire tutte le stagioni sportive a disposizione nella pagina all'interno della select-box dedicata. Infine, un "For Each" permette di iterare su ogni stagione per poi andare a compiere a cascata altra operazioni più specifiche. Tra queste ultime vi è ad esempio l'utilizzo di un potente strumento di UiPath che permette la selezione degli elementi direttamente dall'interfaccia grafica (applicando il concetto di *target-anchor* che individua in modo affidabile gli elementi):

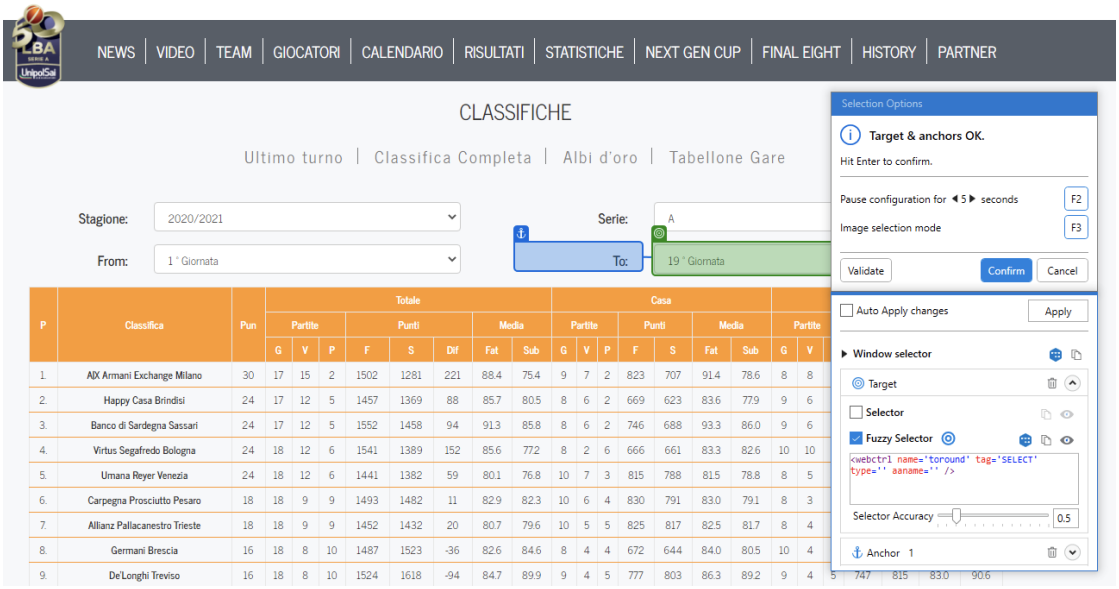


Figura 2.11: Schermata di selezione di un target element nel progetto UiPath Studio

In altri casi, invece, si è dovuto utilizzare "UI Explorer": un editor di *selector* HMTL in grado di individuare un elemento con estrema precisione, grazie alla possibilità di scelta tra tutti attributi utilizzati negli elementi. Ciò è stato necessario quando, ad esempio, si sono presentate situazioni che hanno richiesto un utilizzo dinamico degli identificatori per individuare specifici elementi. Tra le altre funzionalità disponibili, è anche possibile validare la selezione costruita.

2.6 Analisi dati

I dati raccolti riguardano le stagioni sportive dalla 2013/2014 alla 2018/2019, della sola Serie A italiana. Si è deciso di escludere la stagione sportiva 2019/2020 in quanto completamente alterata a causa della pandemia da *COVID-19*, e la 2020/2021 poiché ancora in corso. Di seguito si riportano le specifiche numerosità:

- 6 stagioni sportive;
- 1.634 gare;
- 2.880 classifiche;
- 21 squadre;
- 30 campi di gioco;
- 58 arbitri;
- 62 allenatori;
- 1.435 giocatori;
- 21.982 statistiche giocatori.

In una fase iniziale di *testing* delle metodologie, ci si è limitati a queste sei stagioni sportive a causa del cambiamento nel tempo di regole (anche importanti) che hanno cambiato in parte il mondo della pallacanestro e le prestazioni dei giocatori stessi. Dalle 1.634 gare iniziali ne sono state rimosse 2 che riportavano un risultato di "20-0" (oppure "0-20") in quanto probabilmente oggetto di interventi da parte del giudice sportivo. Sono state rimosse inoltre altre 194 gare poiché relative alle fasi di *playoff*, altamente influenzate dal numero limitato di squadre che vi partecipano e da altre modalità di classifica; un totale netto quindi di 1.438 gare. Nella fase successiva, si è deciso di prendere in esame unicamente la *regular season* della Serie A.
Di seguito si mostra il numero di gare disputate per ogni stagione sportiva presa in analisi.

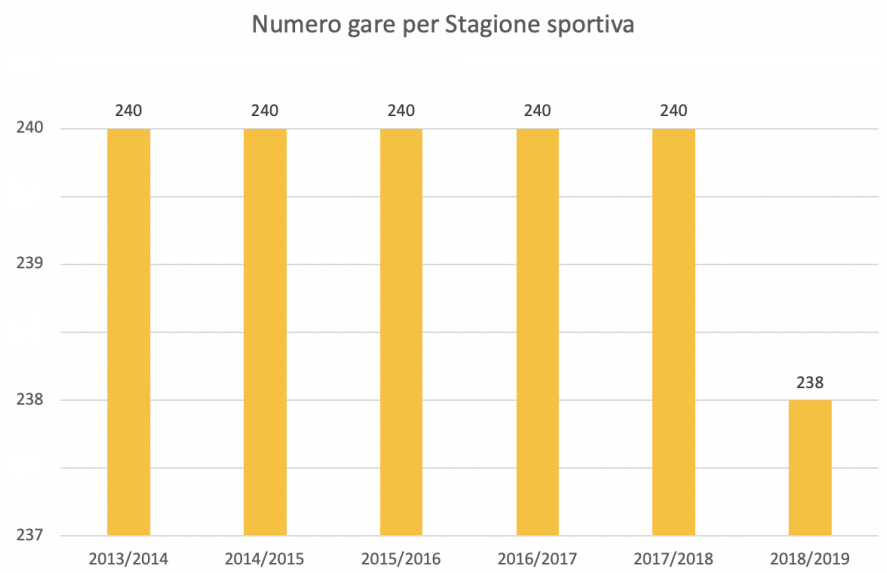


Figura 2.12: Grafico del numero di gare per ogni stagione sportiva

Altrettanto lineare si mostra essere il numero di gare svolte per ogni giornata di campionato, attestandosi sempre su 8 gare (7 per alcune rare eccezioni, ad es. turno di riposo per una squadra).

2.6.1 Etichettatura ed elaborazione dati

Etichettatura

Tutte le gare prese in esame sono state poi classificate in base alla loro difficoltà (facile/media/difficile) da esperti del dominio. La difficoltà è risultata essere così distribuita:

- Gare “facili”: 432;
- Gare “medie”: 423;
- Gare “difficili”: 583.

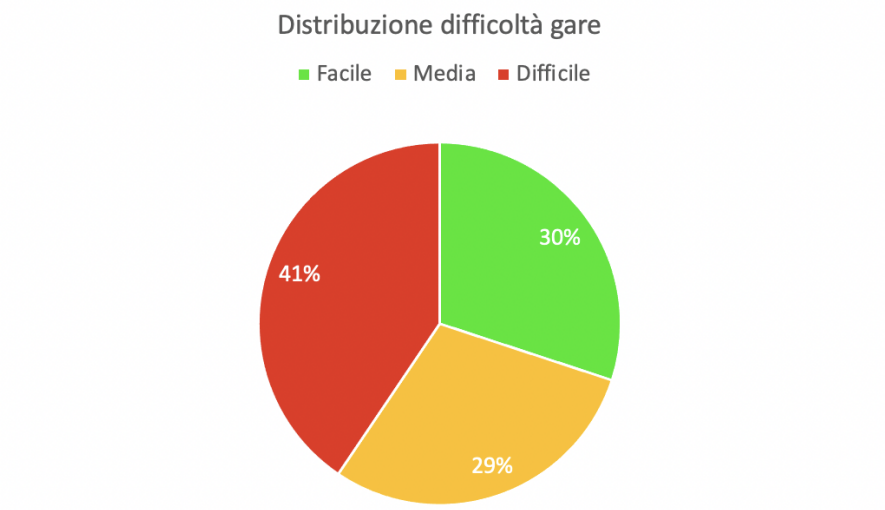


Figura 2.13: Grafico della distribuzione della difficoltà nelle gare

Dal grafico si evince come le gare "facili" corrispondano all'incirca a quelle "medie", mentre le "difficili" sono più numerose in termini di percentuali. Si mostra ora come le stesse siano suddivise all'interno di ogni singola stagione sportiva:

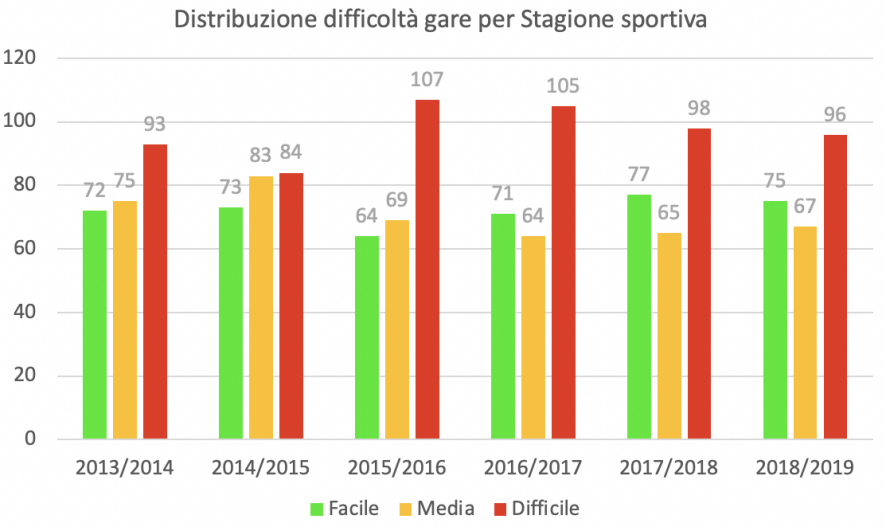


Figura 2.14: Grafico della distribuzione della difficoltà nelle gare per singola stagione sportiva

Come si può notare, il numero di gare "difficili" è in proporzione quasi sempre maggiore rispetto alle altre, in particolar modo per le stagioni 2015/2016 e 2016/2017.

Elaborazione dati

Una volta acquisiti i dati, essi sono stati elaborati in modo da favorire al meglio la successiva fase, ovvero quella di generazione dei *dataset*. L'elaborazione è avvenuta prevalentemente attraverso la costruzione di apposite *query* finalizzate ad unire tramite *join* le diverse tabelle presenti. Di seguito vengono elencate le principali:

- "Gare con classifica": contiene tutte le gare con le relative statistiche;
- "Voti Lega giocatori": un elenco completo di tutti i Voti Lega ottenuti dai giocatori in ogni partita;
- "Classifica": dove sono presenti tutte le classifiche relative alle singole giornate di campionato.

A fronte di alcune analisi preliminari, si è fin da subito colta da centralità del "Voto Lega", il quale, come descritto precedentemente, corrisponde una votazione attribuita dalla Lega ad un giocatore in ogni partita. Esso è rappresentativo quindi delle *performance* dei giocatori.
Di seguito viene mostrato l'andamento del Voto Lega di squadra (che corrisponde alla somma dei singoli Voti Lega dei giocatori) per Milano e Venezia, due squadra di "alta classifica" (rispettivamente al 1° e 3° posto a fine stagione) nel 2018/2019.

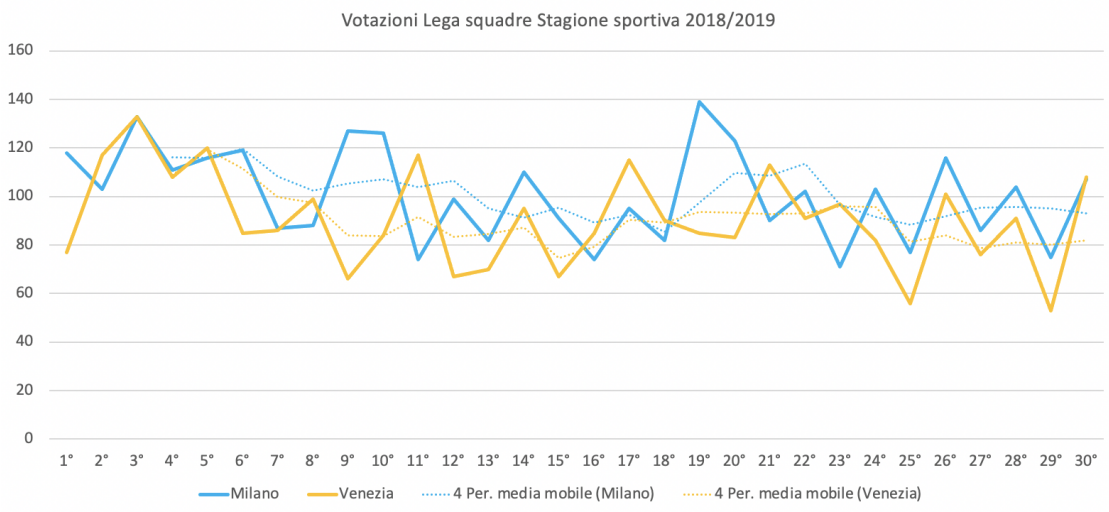


Figura 2.15: Grafico delle Votazioni Lega per Milano e Venezia (2018/2019)

Come si può notare, i valori prestazionali di queste due squadre sono molto simili, anche in termini di media mobile (a 4 periodi).
Per questo motivo si è proceduto col creare diversi script *Python* che generassero delle medie di tali voti, per ogni gara e per ogni squadra: così facendo è stato possibile conoscere puntualmente ad ogni gara, le medie delle squadre che si affrontano e riassumere quindi le prestazioni di entrambe. Nel dettaglio, le medie sono relative, sia per la squadra in casa sia ospite, ad un numero predefinito di ultime gare, più precisamente:

- 999 gare;
- 10 gare;
- 5 gare;
- 3 gare;
- 2 gare;
- 1 gara.

Questa metrica creata *ad hoc* consente, ad esempio, di valutare se una squadra è stata più performante dell'altra nel breve termine (alti valori di media per le ultime 1/2/3 gare), però magari storicamente meno forte (bassi valori di media per le ultime 999/10/5 gare). Così facendo è possibile quindi evidenziare un eventuale *trend* (positivo o negativo che sia) delle squadre per ogni gara nel corso delle stagioni sportive.

2.7 Dataset prodotti

Come menzionato nell'introduzione di questo elaborato, non esiste un *dataset* predefinito con le *feature* da utilizzare per calcolare la difficoltà di una gara. Basandoci sull'analisi degli studi esistenti e sulle statistiche a disposizione della Lega Basket si sono formulati diversi dataset.

Tutti i dataset generati riguardano le 6 stagioni sportive dalla 2013/2014 fino alla 2018/2019. Il numero totale di gare è pari a 1.438.

Dataset 1

Il primo dataset prodotto contiene la principale metrica di riferimento: la classifica. Unita a quest'ultima sono state inserite le differenze (in valore assoluti) tra le medie dei Voti Lega precedentemente calcolate. L'ultima colonna rappresenta la variabile Y (*target*), cioè la difficoltà della gara. Nel dettaglio le *feature* sono:

- Differenza_Classifica_Posizione: la differenza ABS tra il numero di posizione di entrambe le squadre;
- Differenza_Classifica_Punti: la differenza ABS tra il numero di punti di entrambe le squadre;
- Differenza_Media_Voti_Lega_Giocatori_x_Gare: la differenza ABS tra le medie dei Voti Lega dei giocatori delle ultime x gare;
- Difficolta: variabile Y da predire.

Il dataset si presenta quindi in questa forma:

Differenza_Classifica_Posizione	Differenza_Classifica_Punti	Differenza_Media_Voti_Lega_Giocatori_999_Gare	Differenza_Media_Voti_Lega_Giocatori_10_Gare	Differenza_Media_Voti_Lega_Giocatori_3_Gare	...	Difficolta
2	2	2.4238128662109375	0.6952381134033203	0.4312171936035156	...	1
5	2	1.1323471069335938	1.5187492370605469	2.5416669845581055	...	1
4	0	0.19023799896240234	0.4333333969116211	0.2662034034729004	...	1
10	2	0.6920223236083984	0.7791671752929688	2.7083330154418945	...	2
3	2	2.2799549102783203	4.455356597900391	5.883927345275879	...	0
6	4	0.4660191535949707	0.7860121726989746	875	...	2
1	0	0.12778282165527344	2.3374996185302734	1.75	...	2
1	0	0.8668603897094727	1.4447917938232422	3.9166665077209473	...	2
7	2	1.1523237228393555	0.2116403579711914	1.3068780899047852	...	1
11	6	0.16671371459960938	2.474752426147461	0.5535707473754883	...	0
3	2	1.0893230438232422	2.8968257904052734	2.4708995819091797	...	1
2	4	1.4828014373779297	0.2555551528930664	4.518517971038818	...	0
9	12	0.6860713958740234	0.7388887405395508	1.9398155212402344	...	1
2	2	0.07628536224365234	1.0083332061767578	0.8472223281860352	...	0

Figura 2.16: *Dataset 1*

Dataset 2

Si è proceduto poi col fornire anche una versione più semplice del problema, contenente unicamente le due metriche riguardanti la classifica, unite chiaramente alla variabile Y della difficoltà. Le *feature* sono state:

- Differenza_Classifica_Posizione: la differenza ABS tra il numero di posizione di entrambe le squadre;
- Differenza_Classifica_Punti: la differenza ABS tra il numero di punti di entrambe le squadre;
- Difficolta: variabile Y da predire.

Differenza_Classifica_Posizione	Differenza_Classifica_Punti	Difficolta
2	2	1
5	2	1
4	0	1
10	2	2
3	2	0
6	4	2
1	0	2
1	0	2
7	2	1
11	6	0
3	2	1
2	4	0
9	12	1
2	2	0

Figura 2.17: *Dataset 2*

Dataset 3

Similarmente al ragionamento che ha sovrinteso il dataset precedente, se ne è prodotto uno contenente le sole medie dei Voti Lega per entrambe le squadre:

- Differenza_Media_Voti_Lega_Giocatori_x_Gare: la differenza ABS tra le medie dei Voti Lega dei giocatori delle ultime x gare;
- Difficolta: variabile Y da predire.

Differenza_Media_Voti_Lega_Giocatori_999_Gare	Differenza_Media_Voti_Lega_Giocatori_10_Gare	Differenza_Media_Voti_Lega_Giocatori_3_Gare	...	Difficolta
2.4238128662109375	0.6952381134033203	0.4312171936035156	...	1
1.1323471069335938	1.5187492370605469	2.5416669845581055	...	1
0.19023799896240234	0.4333333969116211	0.2662034034729004	...	1
0.6920223236083984	0.7791671752929688	2.7083330154418945	...	2
2.2799549102783203	4.455356597900391	5.883927345275879	...	0
0.4660191535949707	0.7860121726989746	875	...	2
0.12778282165527344	2.3374996185302734	1.75	...	2
0.8668603897094727	1.4447917938232422	3.9166665077209473	...	2
1.1523237228393555	0.2116403579711914	1.3068780899047852	...	1
0.16671371459960938	2.474752426147461	0.5535707473754883	...	0
1.0893230438232422	2.8968257904052734	2.4708995819091797	...	1
1.4828014373779297	0.2555551528930664	4.518517971038818	...	0
0.6860713958740234	0.7388887405395508	1.9398155212402344	...	1
0.07628536224365234	1.0083332061767578	0.8472223281860352	...	0

Figura 2.18: *Dataset 3*

Dataset 4

Il penultimo dataset prodotto contiene le *feature* più rilevanti che riguardano solamente la classifica:

- Differenza_Classifica_Posizione: la differenza ABS tra il numero di posizione di entrambe le squadre;
- Differenza_Classifica_Punti: la differenza ABS tra il numero di punti di entrambe le squadre;
- Differenza_Classifica_Totale_Partite_Vinte: la differenza ABS tra il numero di partite vinte di entrambe le squadre;
- Differenza_Classifica_Totale_Partite_Perse: la differenza ABS tra il numero di partite perse di entrambe le squadre;
- Difficolta: variabile Y da predire.

Differenza_Classifica_Posizione	Differenza_Classifica_Punti	Differenza_Classifica_Totale_Partite_Vinte	Differenza_Classifica_Totale_Partite_Perse	Difficolta
2	2	1	1	1
5	2	1	1	1
4	0	0	0	1
10	2	1	1	2
3	2	1	2	0
6	4	2	1	2
1	0	0	0	2
1	0	0	0	2
7	2	1	1	1
11	6	3	3	0
3	2	1	0	1
2	4	2	2	0
9	12	6	6	1
2	2	1	1	0

Figura 2.19: *Dataset 4*

Dataset 5

L'ultimo dataset, invece, contiene tutte le metriche più rilevanti: dalle classifiche alle medie di squadra dei Voti Lega, incluse le partite vinte e perse:

- Differenza_Classifica_Posizione: la differenza ABS tra il numero di posizione di entrambe le squadre;

- Differenza_Classifica_Punti: la differenza ABS tra il numero di punti di entrambe le squadre;

- Differenza_Media_Voti_Lega_Giocatori_x_Gare: la differenza ABS tra le medie dei Voti Lega dei giocatori delle ultime x gare;

- Differenza_Classifica_Totale_Partite_Vinte: la differenza ABS tra il numero di partite vinte di entrambe le squadre;

- Differenza_Classifica_Totale_Partite_Perse: la differenza ABS tra il numero di partite perse di entrambe le squadre;

- Differenza_Classifica_Totale_Punti_Fatti: la differenza ABS tra il numero di punti realizzati da entrambe le squadre;

- Differenza_Classifica_Totale_Punti_Subiti: la differenza ABS tra il numero di punti subiti da entrambe le squadre;

- Differenza_Classifica_Totale_Media_Fatti: la differenza ABS tra la media di punti fatti da entrambe le squadre;

- Differenza_Classifica_Totale_Media_Subiti: la differenza ABS tra la media di punti subiti da entrambe le squadre;

- Difficolta: variabile Y da predire.

Differenza_Classifica_Posizione	Differenza_Classifica_Punti	Differenza_Media_Voti_Lega_Giocatori_999_Gare	...	Differenza_Classifica_Totale_Partite_Vinte	Differenza_Classifica_Totale_Partite_Perse	...	Difficolta
4	0	0.19023799896240234	...	0	0	...	1
9	2	2.500302314758301	...	1	1	...	0
4	0	1.1682190895080566	...	0	0	...	0
7	2	1.6644611358642578	...	1	1	...	2
3	0	0.15081024169921875	...	0	0	...	2
2	0	1.6486873626708984	...	0	0	...	1
11	4	2.2430906295776367	...	2	2	...	2
3	0	0.7376375198364258	...	0	1	...	0
1	0	4.565260887145996	...	0	0	...	0
11	4	0.2858905792236328	...	2	2	...	0
5	2	2.151185989379883	...	1	0	...	2
2	2	2.4238128662109375	...	1	1	...	1
3	2	1.3346104621887207	...	1	1	...	1
7	2	1.0285234451293945	...	1	1	...	0

Figura 2.20: *Dataset 5*

Capitolo 3

Predizione della difficoltà delle gare

In questo capitolo viene approfondito lo svolgimento di una delle due parti fondamentali dell'elaborato: la predizione della difficoltà delle gare. Verranno descritte nel dettaglio le implementazioni dei vari algoritmi utilizzati, inclusa la rete neurale. Viene poi approfondito il progetto riguardante un articolo scientifico citato nello stato dell'arte. Infine, si valutano e analizzano i risultati ottenuti. L'obiettivo principale di tutti gli algoritmi impiegati è quello di predire la difficoltà delle gare della stagione sportiva 2018/2019, sulla base delle precedenti stagioni dalla 2013/2014 fino alla 2017/2018.

3.1 Tecnologie

Di seguito vengono descritti i *software*, tecnologie e librerie utilizzate per la predizione della difficoltà delle gare.

Python

Python[1] è un linguaggio di programmazione di "alto livello" rispetto alla maggior parte degli altri linguaggi, orientato a oggetti, ed adatto, tra gli altri usi, a sviluppare applicazioni distribuite, scripting, computazione numerica e system testing. È un linguaggio multi-paradigma che ha tra i principali obiettivi: dinamicità, semplicità e flessibilità. Supporta il paradigma object oriented, la programmazione strutturata e molte caratteristiche di programmazione funzionale e riflessione. Le caratteristiche più immediatamente riconoscibili di Python sono le variabili non tipizzate e l'uso dell'indentazione per la sintassi delle specifiche, al posto delle più comuni parentesi.
Altre caratteristiche distintive sono l'overloading di operatori e funzioni tramite delegati, la presenza di un ricco assortimento di tipi e funzioni di base e librerie standard, sintassi avanzate quali slicing e list comprehension.
Il controllo dei tipi è forte (strong typing) e viene eseguito a runtime (dynamic typing): una variabile è un contenitore a cui viene associata un'etichetta (il nome) che può essere associata a diversi contenitori anche di tipo diverso durante il suo tempo di vita. Fa parte di Python un sistema garbage collector per liberazione e recupero automatico della memoria di lavoro.

PyCharm

[1] https://www.python.org/

PyCharm[2] è un ambiente di sviluppo integrato (IDE) utilizzato nella programmazione di computer, in particolare per il linguaggio Python. È sviluppato dalla società ceca JetBrains.

Scikit-learn

Scikit-learn[3] (ex "scikits.learn") è una libreria open source di apprendimento automatico per il linguaggio di programmazione Python. Contiene algoritmi di classificazione, regressione e clustering (raggruppamento) e macchine a vettori di supporto, regressione logistica, classificatore bayesiano, k-mean e DBSCAN, ed è progettato per operare con le librerie NumPy e SciPy. scikit-learn è attualmente sponsorizzato da INRIA e talvolta da Google.

NumPy

NumPy[4] è una libreria open source per il linguaggio di programmazione Python, che aggiunge supporto a grandi matrici e array multidimensionali insieme a una vasta collezione di funzioni matematiche di alto livello per poter operare efficientemente su queste strutture dati. È stato creato nel 2005 da Travis Oliphant basandosi su Numeric di Jim Hugunin.

[2] https://www.jetbrains.com/pycharm/
[3] https://scikit-learn.org/stable/
[4] https://numpy.org/

Pandas

Pandas[5] è una libreria software scritta per il linguaggio di programmazione Python per la manipolazione e l'analisi dei dati. In particolare, offre strutture dati e operazioni per manipolare tabelle numeriche e serie temporali. È un software libero rilasciato sotto la licenza BSD a tre clausole. Il nome deriva dal termine "panel data", termine econometrico per set di dati che include osservazioni su più periodi di tempo per gli stessi individui.

PyTorch

PyTorch[6] è una libreria di apprendimento automatico open source basata sulla libreria Torch, utilizzata per applicazioni come la visione artificiale e l'elaborazione del linguaggio naturale, sviluppata principalmente dal laboratorio di ricerca AI di Facebook.

3.2 Implementazione

Il progetto è stato interamente sviluppato in Python 3.0[7] e creato all'interno di PyCharm. Lo script principale "SupervisedAlgorithms.py" contiene il codice degli algoritmi di apprendimento supervisionato utilizzati. Lo scopo dell'applicazione di questi algoritmi è, tramite i dati storici, quello di predire la difficoltà delle gare,

[5] `https://pandas.pydata.org/`
[6] `https://pytorch.org/`
[7] `https://www.python.org/download/releases/3.0/`

più nello specifico se "facile", "media" o "difficile".
Prima di eseguire gli algoritmi, i dataset (descritti nel precedente capitolo) vengono caricati uno alla volta in un `DataFrame`[8] tramite la libreria Pandas (linea 257). Successivamente viene effettuato un controllo delle dimensioni della matrice ed effettuata la divisione tra il `DataFrame` "`df_X`" (i dati) e il `DataFrame` "`df_Y`" (la classe da predire) – linee 262 e 263. Proseguendo, avviene lo scaling dei dati utilizzando il `MinMaxScaler`[9] di scikit-learn (linee 267-269). Così facendo tutti i dati contenuti nel `DataFrame` vengono scalati, per ogni singola feature, nell'intervallo 0-1.

Una volta scalati i dati, vengono costruiti i set per la fase di train e test, sia per la variabile X che per la Y. Questo viene eseguito dal metodo `train_test_split` (linea 271), dove il dataset iniziale viene diviso in modo casuale: per il 33% al *test set*, e per il restante 67% al *train set*. Successivamente sono stati testati sette diversi algoritmi di Machine Learning supervisionati:

- Regressione logistica;
- Support Vector Machine;
- Random Forest;
- Decision Tree;
- Gaussian NB;
- Gradient Boosting;
- K-Neighbors.

Di seguito viene mostrato il codice del "*main*" dello script.

[8] `https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html`

[9] `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html`

```python
def main():

    # Load the dataset
    df = pd.read_csv("dataset/supervisioned/medie_gmsc.csv")
    print("Rows before dropna():", df.shape[0])
    df = df.dropna()
    print("Rows after dropna():", df.shape[0])

    df_X = pd.DataFrame(df.iloc[:, :-1])
    df_Y = pd.DataFrame(df["Difficolta"])

    columns = df_X.columns

    scaler = MinMaxScaler()
    scaler.fit(df_X)
    df_X = scaler.transform(df_X)

    X_train, X_test, Y_train, Y_test = train_test_split(df_X, df_Y, test_size=0.33,
                                                        random_state=20)

    # Algorithms

    logistic_regression(X_train, X_test, Y_train, Y_test)

    support_vector_machine(X_train, X_test, Y_train, Y_test)

    random_forest_classifier(X_train, X_test, Y_train, Y_test, columns)

    decision_tree_classifier(X_train, X_test, Y_train, Y_test)

    gaussian_nb(X_train, X_test, Y_train, Y_test)

    gradient_boosting_classifier(X_train, X_test, Y_train, Y_test)

    k_neighbors_classifier(X_train, X_test, Y_train, Y_test)

```

Codice 3.1: Script Python “main” del Supervised Learning

Nelle prossime sezioni vengono approfonditi i dettagli implementativi dei singoli algoritmi utilizzati, mentre al termine di questo capitolo verranno analizzati i risultati ottenuti nella sezione “Valutazione risultati”.

Regressione Logistica

Il primo algoritmo testato è stato quello della Regressione Logistica[10] di scikit-learn, la quale classifica i dati (in questo caso la difficoltà delle gare) in base a dati storici.
Nell'esempio sottostante, la miglior performance è stata ottenuta scegliendo di limitare il numero di iterazioni `max_iter` a "100"; trattandosi di un problema multi-classe (poiché sono presenti tre classi: "facile", "media" e "difficile), il `solver` utilizzato è stato *"newton-cg"*. Una volta creato il classificatore `clf`, vengono fittati i dati `X_train` e `Y_train`, predetti i valori di Y, ed infine calcolato lo `score` e visualizzata la *confusion matrix*. Inoltre, viene prodotto anche un report di classificazione tramite la funzione `classification_report` di scikit-learn, il quale contiene metriche come la *precision*, la *recall*, l'*f1-score* e il *support*.

```
91  def logistic_regression(X_train, X_test, Y_train, Y_test):
92      print("Model: LogisticRegression")
93
94      Y_train = np.ravel(Y_train)
95      Y_test = np.ravel(Y_test)
96
97      clf = LogisticRegression(max_iter=100, solver='newton-cg', C=1.0)
98      clf.fit(X_train, Y_train)
99      Y_predicted = clf.predict(X_test)
100
101     # Results
102     print("Avg accuracy: ", clf.score(X_test, Y_test))
103     print(classification_report(Y_test, Y_predicted))
104
105     # Plot the confusion matrix
106     plot_confusion_matrix(clf, X_test, Y_test, cmap=plt.cm.Blues)
107     plt.show()
```

Codice 3.2: Script Python per la Regressione Logistica

[10] `https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html`

Support Vector Machine

Il secondo algoritmo è il Support Vector Machine (SVM)[11], particolarmente performante per i problemi di classificazione come quello sottoposto. Sono stati utilizzati tre diversi kernel offerti da SVM: `linear`, `poly` ed `rbf`. Il kernel `linear` si presta a separare i dati con semplici linee, come suggerisce il nome stesso. Il `poly`, invece, rappresenta la somiglianza dei vettori in uno spazio delle caratteristiche sui polinomi delle variabili originali, consentendo l'apprendimento di modelli non lineari. `rbf`[12] (*radial basis function*), infine, è una funzione il cui valore dipende dalla distanza dall'origine o da qualche punto in particolare; utilizzando la distanza nello spazio originale viene calcolato il prodotto scalare in termini di somiglianza (rappresentata dalla distanza angolare tra due punti). Quest'ultimo kernel, rispetto agli altri, offre anche l'utilizzo del parametro `gamma`, il quale definisce quanto è estesa l'influenza dei singoli elementi del training set. Sostanzialmente quando esso è eccessivamente piccolo (in termini numerici) vincola il modello e quest'ultimo potrebbe non catturare la complessità generale dei dati, portandolo così quasi ad essere paragonabile ad un modello lineare con un insieme di iperpiani che separano i centri di alta densità di qualsiasi coppia di due classi diverse. Tutti i kernel dipendono, inoltre, da una variabile `C`, la quale funge da parametro regolarizzatore e definisce il *trade-off* tra la corretta classificazione dei punti del training set e la massimizzazione del margine della *decision function*. Esso influisce quindi sull'abilità del modello di generalizzare la classificazione su nuovi dati che non ha mai visto precedentemente.

[11] `https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html`
[12] `https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html`

```
114 def support_vector_machine(X_train, X_test, Y_train, Y_test):
115     print("Model: SupportVectorMachine\n")
116 
117     Y_train = np.ravel(Y_train)
118     Y_test = np.ravel(Y_test)
119 
120     # Linear
121     clf = svm.SVC(kernel='linear', C=1)
122     clf.fit(X_train, Y_train)
123     print("Test set accuracy (Linear): {:.2f}".format(clf.score(X_test, Y_test)))
124 
125     # Poly
126     clf = svm.SVC(kernel='poly', C=1, degree=3)
127     clf.fit(X_train, Y_train)
128     print("Test set accuracy (Poly): {:.2f}".format(clf.score(X_test, Y_test)))
129 
130     # RBF
131     clf = svm.SVC(kernel='rbf', gamma='scale')
132     clf.fit(X_train, Y_train)
133     print("Test set accuracy (RBF): {:.2f}".format(clf.score(X_test, Y_test)))
134 
135     # RBF Best
136     clf = svm.SVC(kernel='rbf', gamma=1, C=10)
137     clf.fit(X_train, Y_train)
138     print("Test set accuracy (RBF Best): {:.2f}".format(clf.score(X_test, Y_test)))
```

Codice 3.3: Script Python per il Support Vector Machine

Random Forest Classifier

Il Random Forest Classifier[13] rappresenta un tipo di modello *ensemble*[14], che si avvale del *bagging*[15] come metodo di ensemble e l'albero decisionale come modello individuale. Questo significa che una foresta casuale combina molti alberi decisionali in un unico modello. Individualmente, le previsioni degli alberi decisionali potrebbero non essere accurate, ma combinate insieme, le previsioni saranno mediamente più prossime al risultato. Il risultato finale è rappresentato dalla classe restituita dal maggior numero di alberi. Grazie al Random Forest Classifier è

[13] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[14] https://it.wikipedia.org/wiki/Apprendimento_ensemble

[15] https://it.wikipedia.org/wiki/Bagging

possibile, inoltre, visualizzare il *ranking* (classifica) dell'importanza delle variabili (grazie alla libreria "seaborn"[16]), i cui risultati sono riportati nell'ultima sezione "Valutazione risultati" di questo capitolo.

```
167 def random_forest_classifier(X_train, X_test, Y_train, Y_test, columns):
168     print("Model: RandomForestClassifier\n")
169 
170     rf = RandomForestClassifier(n_estimators=100)
171     rf.fit(X_train, np.ravel(Y_train))
172     rf.predict(X_test)
173 
174     print("Avg accuracy: ", rf.score(X_test, Y_test), "\n")
175 
176     rankVar = pd.Series(rf.feature_importances_, index=columns).sort_values(ascending=False)
177     print(rankVar)
178 
179     sns.barplot(x=rankVar, y=rankVar.index)
180     plt.xlabel('Variable Importance Score')
181     plt.ylabel('Variables')
182     plt.legend()
183     plt.show()
```

Codice 3.4: Script Python per il Random Forest Classifier

Decision Tree Classifier

Il Decision Tree Classifier[17] consiste in un modello predittivo, dove ogni nodo interno rappresenta una variabile, un arco verso un nodo figlio rappresenta un possibile valore per quella proprietà e una foglia il valore predetto per la variabile obiettivo a partire dai valori delle altre proprietà, che nell'albero è rappresentato dal cammino (*path*) dal nodo radice (*root*) al nodo foglia.
Il suo utilizzo è particolarmente utile poiché, per ogni classe predetta, viene costruito un vero e proprio percorso decisionale basato sulle features e i relativi valori che hanno portato alla decisione della classificazione. Con i parametri `max_depth` si impone una profondità massima dell'albero, con `criterion` "*gini*" si misu-

[16] `https://seaborn.pydata.org/`

[17] `https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html`

ra la qualità dello *split* nei vari punti di snodo, e `ccp_alpha` regola la potatura (*prouning*).

```
186 def decision_tree_classifier(X_train, X_test, Y_train, Y_test):
187     print("Model: DecisionTreeClassifier")
188 
189     dtc = DecisionTreeClassifier(max_depth=4, criterion="gini", ccp_alpha=0.0035)
190     dtc.fit(X_train, Y_train)
191 
192     tree.plot_tree(dtc, filled=True)
193     plt.show()
194 
195     text_representation = tree.export_text(dtc)
196     print(text_representation)
197     dtc.predict(X_test)
198     print("Avg accuracy: ", dtc.score(X_test, Y_test))
```

Codice 3.5: Script Python per il Decision Tree Classifier

Gaussian NB

Il Gaussian Naive Bayes Classifier[18] utilizzato è un algoritmo che, oltre ad avvalersi del teorema di Bayes[19] (il quale descrive la probabilità di un evento), si basa sul fatto che tutte le caratteristiche non siano correlate l'una all'altra. La presenza o l'assenza di una caratteristica non influenzano la presenza o l'assenza di altre. L'algoritmo è suddiviso nelle seguenti fasi:

1. *Calcolo della probabilità della classe*: estrazione delle frequenze delle istanze che appartengono a ciascuna classe divisa per il numero totale di istanze;

2. *Calcolo della probabilità condizionata*: viene applicato il teorema di Bayes per determinare le probabilità condizionate delle caratteristiche del problema;

3. *Decisione*: viene calcolata la probabilità per prevedere la classe di appartenenza della nuova istanza rispettando la verifica dell'indipendenza delle

[18] `https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html`

[19] `https://it.wikipedia.org/wiki/Teorema_di_Bayes`

caratteristiche. La decisione finale è identificata nella classe che ottiene il valore di probabilità più elevato.

```
201 def gaussian_nb(X_train, X_test, Y_train, Y_test):
202     print("Model: GaussianNB\n")
203 
204     Y_train = np.ravel(Y_train)
205     Y_test = np.ravel(Y_test)
206 
207     # GaussianNB
208     gnb = GaussianNB()
209     gnb.fit(X_train, Y_train).predict(X_test)
210     print("Avg accuracy: ", gnb.score(X_test, Y_test))
211     print("Mean of the Gaussian Estimators")
212     print(gnb.theta_)
213     print("Std Dev of the Gaussian Estimators")
214     print(gnb.sigma_)
215 
216     # QuadraticDiscriminantAnalysis
217     qda = QuadraticDiscriminantAnalysis(store_covariance=True)
218     qda.fit(X_train, Y_train).predict(X_test)
219     print("Avg accuracy: ", qda.score(X_test, Y_test))
220     print("QDA means per class")
221     print(qda.means_)
222     print("QDA covariance per class")
223     print(qda.covariance_)
```

Codice 3.6: Script Python per il Gaussian Naive Bayes

Gradient Boosting Classifier

Il penultimo classificator utilizzato è stato il Gradient Boosting[20], un modello ensemble, ovvero che si basa sulla combinazione di singoli modelli semplici (*weak learner*) che insieme creano un nuovo modello più potente (*strong learner*). L'algoritmo inizia inserendo un modello iniziale (albero) e successivamente viene creato

[20] `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html`

un secondo modello in sequenza, che si concentra sulla previsione accurata dei casi in cui il primo modello ha prestazioni scarse. La combinazione di questi due si auspica essere migliore di entrambi i modelli presi singolarmente. Il processo di *boosting* consiste quindi nel ripetere molteplici volte questo procedimento. Ogni modello successivo tenta così di correggere le carenze dell'insieme combinato di tutti i modelli precedenti.
Due sono i parametri utilizzati: `n_estimators`, che corrisponde al numero di alberi che saranno adattati in serie per correggere gli errori di predizione; ed il `learning_rate`, cioè la velocità con cui l'errore viene corretto da ciascun albero al successivo.

```
226 def gradient_boosting_classifier(X_train, X_test, Y_train, Y_test):
227     print("Model: GradientBoostingClassifier\n")
228 
229     Y_train = np.ravel(Y_train)
230     Y_test = np.ravel(Y_test)
231 
232     gb = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1)
233     gb.fit(X_train, Y_train)
234     gb.predict(X_test)
235 
236     score = gb.score(X_test, Y_test)
237 
238     print("Avg accuracy: ", score)
```

Codice 3.7: Script Python per il Gradient Boosting Classifier

K-Nearest Neighbors Classifier

Il K-Nearest Neighbors (KNN)[21] è stato il settimo ed ultimo classificatore utilizzato. Il suo funzionamento si basa sulla somiglianza delle caratteristiche: più un'istanza è vicina a un *data point*, più il KNN li considererà come simili. L'algoritmo prevede inoltre di fissare un parametro k (`n_neighbors`), scelto arbitraria-

[21] `https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html`

mente, che identifica il numero di *data point* più vicini. Vengono quindi valutate le k minime distanze ottenute e la classe che ottiene il maggior numero di queste distanze è scelta come previsione.

```
241 def k_neighbors_classifier(X_train, X_test, Y_train, Y_test):
242     print("Model: KNeighborsClassifier\n")
243 
244     Y_train = np.ravel(Y_train)
245     Y_test = np.ravel(Y_test)
246 
247     # KNeighborsClassifier
248     clf = KNeighborsClassifier(n_neighbors=5)
249     clf.fit(X_train, Y_train)
250     print("Test set predictions: {}".format(clf.predict(X_test)))
251     print("Test set accuracy: {:.2f}".format(clf.score(X_test, Y_test)))
```

Codice 3.8: Script Python per il K-Nearest Neighbors Classifier

3.3 Rete Neurale Feed-forward

A completare la panoramica sugli algoritmi di Machine Learning implementati, si è scelto di costruire anche una semplice rete neurale Feed-forward per testarne le performance. Il codice inizia con la consueta procedura di caricamento dei dati dal CSV e la divisione nel dataset `X` (i dati) e `Y` (la variabile target). Successivamente si è passati ad effettuare il `train_test_split` con le medesime modalità degli altri algoritmi: 33% per il *test set* e il restante 67% per il *training set*. Il numero delle classi `num_classes` è recuperato automaticamente dai dati `Y`, e sarà pari a 3, poiché le classi per la difficoltà della gara sono "facile", "media" e "difficile". Il parametro `hidden_size`, scelto dopo diverse esecuzioni pari a "2", rappresenta il numero di hidden layer della rete neurale. Il numero di epoche `num_epochs` è stato impostato a 1.000, ma risultati soddisfacenti si raggiungono già dopo 300 epoche (nella sezione "Risultati" verrà approfondito questo aspetto). Attraverso i tensori `FloatTensor` e `LongTensor` vengono trasformati gli input in matrici multidimensionali di numeri, utilizzabili successivamente sia dal `train_model` che dal `test_model`. Il modello

creato è il `Feedforward` di *Torch*, a cui vengono passati come parametri il numero di features del dataset, il numero di hidden layer e di classi target. Il criterion scelto è stato il `CrossEntropyLoss` poiché il problema non è binario (con sole due classi) ma ne presenta di diverse; come optimizer, invece, l'`SGD`, così da implementare la discesa stocastica del gradiente, con un learning rate pari a "0.01".

```
data = pd.read_csv("../dataset/supervisioned/posizione_punti_vinte_perse_punti_medie.csv")
X = data.iloc[:, :-1].to_numpy()
y = data["Difficolta"].to_numpy()

X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.33, stratify=y, random_state=42)

num_classes = len(np.unique(y_train))
hidden_size = 2
num_epochs = 1000

X_train = torch.FloatTensor(X_train)
y_train = torch.LongTensor(y_train)
X_val = torch.FloatTensor(X_val)
y_val = torch.LongTensor(y_val)

model = Feedforward(X_train.shape[1], hidden_size, num_classes)
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.01)

test_model(model, X_val, y_val)
model, loss_values = train_model(model, criterion, optimizer, num_epochs, X_train, y_train)
plt.plot(loss_values)
plt.title("Number of epochs: {}".format(num_epochs))
plt.show()
test_model(model, X_val, y_val)
```

Codice 3.9: Script Python per la NN Feed-forward (inizializzazione)

Al termine delle inizializzazioni appena illustrate, viene eseguito il `train_model`, il quale addestra il modello con un ciclo attraverso le epoche impostate.

```python
def train_model(model, criterion, optimizer, epochs, X_train, y_train):
    model.train()
    loss_values = []
    for epoch in range(epochs):
        optimizer.zero_grad()

        # Forward pass (the whole training set)
        y_pred = model(X_train)

        # Compute Loss
        loss = criterion(y_pred.squeeze(), y_train)
        loss_values.append(loss)

        print('Epoch {} train loss: {}'.format(epoch, loss.item()))

        # Backward pass
        loss.backward()
        optimizer.step()

    return model, loss_values
```

Codice 3.10: Script Python per la NN Feed-forward (train_model)

Infine, la rete neurale viene testata per ottenere un valore di accuracy:

```python
def test_model(model, X_val, y_val):
    model.eval()
    y_pred = model(X_val)
    y_pred = y_pred.argmax(dim=1, keepdim=True)

    score = torch.sum((y_pred.squeeze() == y_val).float()) / y_val.shape[0]
    print('Test score', score.numpy())
```

Codice 3.11: Script Python per la NN Feed-forward (test_model)

3.4 Relazione tra difficoltà delle gare e risultati finali

Come introdotto all'inizio di questo elaborato, alla luce degli studi approfonditi sull'NBA riguardanti la predizione dei risultati finali (*outcomes*), si è tentato di trovare una relazione tra i risultati finali e la difficoltà delle gare. Questo al fine di implementare ulteriori sistemi, più sofisticati, e che tengano in considerazione anche altri fattori come, ad esempio il concetto di abilità di un giocatore/di squadra.
Per effettuare questa analisi, si è proceduto ad eseguire una serie di interrogazioni (*query*) al nostro database contenente i dati di tutte le gare, raggruppandole per intervalli predefiniti. Questi ultimi sono stati impostati in relazione ad un tipico intervallo che secondo gli esperti del dominio potrebbe raggruppare in modo significativo le difficoltà delle gare in base al loro risultato finale.

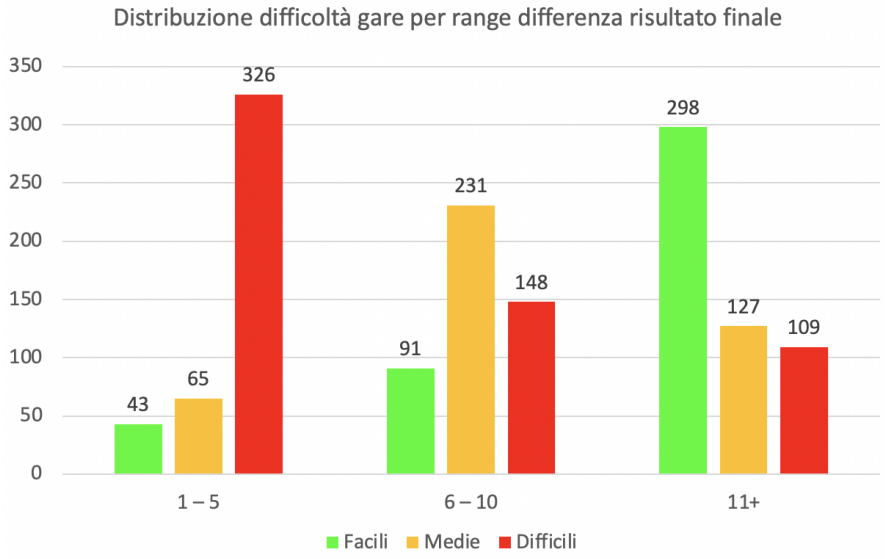


Figura 3.1: Distribuzione della difficoltà delle gare per range di differenza sul risultato finale

Come si evince dal grafico, sembra esserci una relazione tra la differenza del risultato finale delle gare (in valore assoluto) e la loro difficoltà. In particolare, per le

gare terminate con un divario compreso tra 1 e 5 punti si nota una elevata differenza tra le gare facili/medie e quelle difficili, molto probabilmente ciò è causato dal fatto che la partita è stata "punto a punto" durante l'arco della stessa o per la maggior parte del tempo di gioco. Per le gare finite con un divario di più di 11 punti è notevole, invece, come queste siano prevalentemente classificate come "facili". Meno distinta è la differenza nell'intervallo da 6 a 10 punti, dove vi è sì una maggioranza di gare di media difficoltà, ma non così netta rispetto agli altri intervalli; e questo potrebbe essere causato dal fatto che comunque il divario finale tra le due squadre potrebbe essere stato raggiunto solo negli ultimi minuti della partita, o magari nell'ultimo quarto. Possiamo dunque affermare che vi è una ragionevole relazione (inversamente proporzionale) tra la difficoltà di una gara e la differenza assoluta dei punti realizzati da entrambe le squadre. Alla luce di questa considerazione, si è proceduto a provare ad applicare un metodo più complesso rispetto a quelli implementati fino ad ora, i cui dettagli sono descritti ed approfonditi nella seguente sezione.

3.5 Progetto "TLGProb"

Nel primo capitolo di questo elaborato, "Stato dell'arte", è stato riportato un articolo scientifico dal titolo "*One-Match-Ahead Forecasting In Two-Team Sports With Stacked Bayesian Regressions*" di Max W. Y. Lam. L'autore è riuscito a raggiungere la più alta accuracy (85%) sulla predizione degli *outcomes* (risultati) dell'NBA mai raggiunta prima, per altro riguardanti recenti stagioni sportive (relativamente all'anno in cui è stato approvato, nel 2017); questo è stato il motivo per cui è stato preso particolarmente in esame e studio. Il motivo di tale approfondimento deriva dalla necessità e curiosità di espandere l'orizzonte di applicazione per tentare di raggiungere più alti risultati nella predizione delle difficoltà delle gare. La relazione tra gli *outcomes* e la difficoltà delle gare viene approfondita ulteriormente nel corso di questo stesso capitolo.

3.5.1 Descrizione

Il progetto “TLGProb”, così denominato dall’autore, consiste nell’applicazione pratica di ciò che è contenuto nell’articolo scientifico. L’obiettivo è stato quello di predire le squadre vincenti per la stagione sportiva 2014/2015 dell’NBA e, in particolare, esplicitando il concetto in termini di predizione delle performance future. In particolare, Max W. Y. Lam, sostiene che la sua iniziativa di modellare esplicitamente le abilità individuali di ogni giocatore e il loro contributo al risultato delle gare abbia migliorato notevolmente gli approcci esistenti in termini di accuratezza delle predizioni. L’articolo, inoltre, risulta essere il primo a lavorare all’associazione tra le abilità dei giocatori e l’abilità dell’intero team utilizzando tecniche di regressione in sequenza.

Formulazione del problema

Lo scopo dell’analisi è quello di derivare la probabilità di vincita delle due squadre di una gara futura, conoscendo solamente i dati storici prima di tale gara. Più precisamente, si cerca di rispondere quanto sia probabile che la squadra di casa oppure ospite vinca la gara successiva; tale attività è definita come “*one-match-ahead forecasting*”. L’assunzione principale è che le abilità dei giocatori nel corso delle gare varino in modo fluido, anche se le loro prestazioni cambiano notevolmente. In tal senso, le performance passate dei giocatori delle due squadre che si affronteranno saranno sufficienti per predire il risultato della gara futura. Per calcolare le probabilità di vincita, ci si concentra sulla differenza di punti ottenuti da entrambe le squadre, attraverso la deviazione con segno e la variabile casuale $g_{k+1} = g^H_{k+1} - g^V_{k+1}$, dove g^H_{k+1} e g^V_{k+1} rappresentano rispettivamente i risultati della gara futura per la squadra casa e ospite; e g_{k+1} è definita come la differenza tra questi due punteggi. Più precisamente, la probabilità che la squadra in casa vinca corrisponde a $Pr(g_{k+1} > 0)$, poiché se essa realizzerà più punti di quella ospite, il valore di g_{k+1} sarà superiore a zero (il risultato “pari” non è ammesso nella pallacanestro). Con questa assunzione, la distribuzione di g_{k+1} dipende fortemente dalle performance passate di entrambe le squadre. Quindi date le performance passate dei giocatori delle prime k gare note, un potenziale

stimatore potrebbe essere calcolato, e viene definito come un'attività di regressione: $g_{k+1} = f(p_{1:k}) + \varepsilon_{k+1}, \varepsilon_{k+1} \backsim \mathcal{N}(0, \sigma^2)$. Dove $p_{1:k}$ rappresentano le performance passate dei giocatori nelle prime k gare. L'obiettivo è dunque quello di definire un modello $f(\cdot)$ che spieghi l'associazione tra il risultato della gara futura e le performance passate dei giocatori.

Modellazione

Il modello introduce quindi un elenco ben definito di features da prendere in considerazione per ogni giocatore:

Metrica	Descrizione	Metrica	Descrizione
GS	Quintetto iniziale	ORB	Rimbalzi offensivi
MP	Minuti giocati	DRB	Rimbalzi difensivi
FG	Canestri realizzati	TRB	Rimbalzi totali
FGA	Canestri tentati	AST	Assist
FG%	Canestri realizzati in percentuale	STL	Palle recuperate
3P	Canestri da 3 punti realizzati	BLK	Stoppate date
3PA	Canestri da 3 punti tentati	TOV	Palle perse
3P%	Canestri da 3 punti realizzati in percentuale	PF	Falli fatti
FT	Tiri liberi realizzati	PTS	Punti realizzati
FTA	Tiri liberi tentati	GMSC	Game score
FT%	Tiri liberi realizzati in percentuale	+/-	Plus/minus

Tabella 3.1: Statistiche giocatori per "TLGProb"

Nel modello, inoltre, viene presa in considerazione anche la posizione del giocatore, che può essere "Centro", "Ala" e "Guardia".
La distinzione fondamentale che fa l'autore è tra le performance di un giocatore e la sua vera abilità, dove la prima tende ad essere molto variabile tra le singole gare, mentre la seconda è più lineare. Solitamente le performance di un giocatore (relative alla singola gara) possono essere certamente influenzate dalla sua abilità (talento) ma anche da altri fattori esterni, come ad esempio la strategia della propria squadra e le abilità degli avversari. Di seguito è mostrato come esempio le

performance (più altalenanti) e abilità (più regolari) dei canestri da 2 e 3 p.t. di LeBron James nella stagione 2014/2015.

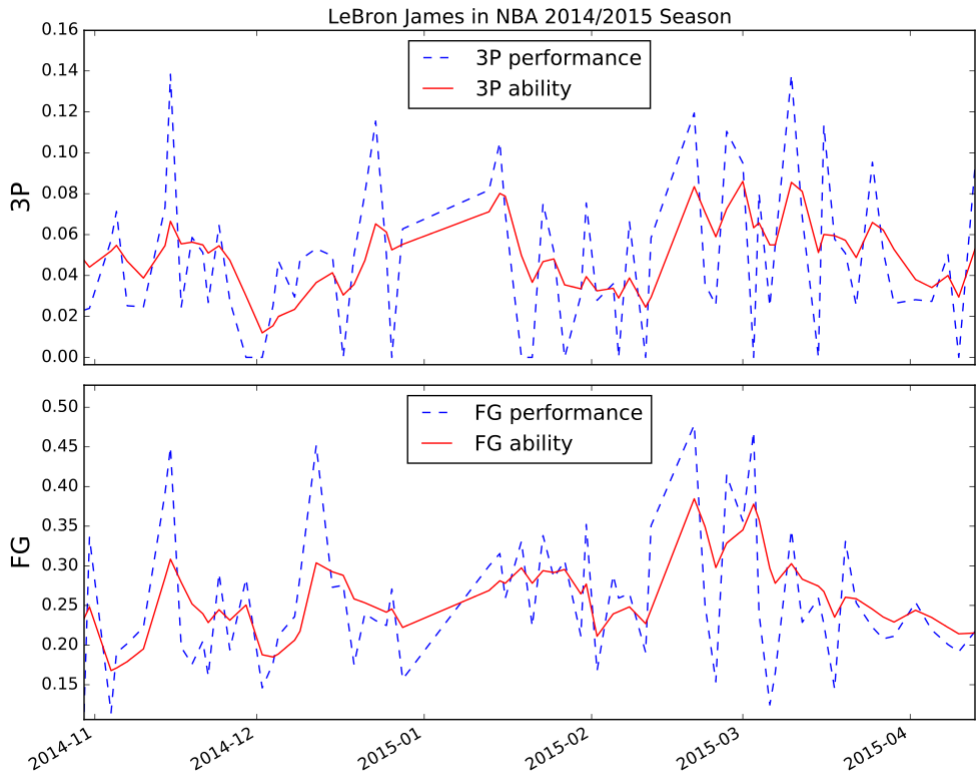


Figura 3.2: Canestri da 2 e 3 p.t. realizzati da LeBron James nella stagione 2014/2015 dell’NBA

L’abilità del giocatore viene modellata attraverso la tecnica di *exponential smooting* e predetta attraverso un modello SSGP (Sparse Spectrum Gaussian Process)[22] appositamente costruito per i giocatori, dove vengono applicati pesi per regolare l’influenza delle performance passata rispetto all’abilità di un giocatore nelle gare recenti a quella da predire. L’abilità di team viene definita come la somma delle abilità dei singoli giocatori, e predetta utilizzando per una seconda volta il modello

[22] `https://jmlr.csail.mit.edu/papers/volume11/lazaro-gredilla10a/lazaro-gredilla10a.pdf`

SSGP (un ulteriore modello appositamente costruito per le squadre). Così facendo vengono utilizzati a cascata due modelli SSGP, come mostrato in figura:

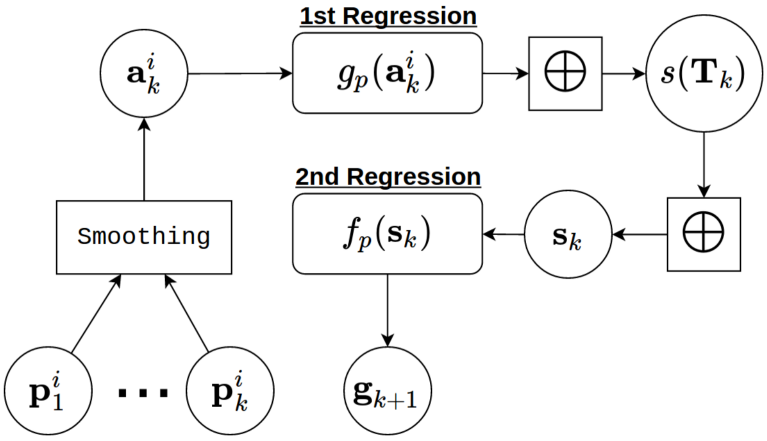


Figura 3.3: Diagramma a blocchi dell'approccio alla modellazione del problema

3.5.2 Implementazione per la Serie A italiana

Una volta effettuato lo studio dell'articolo scientifico ed analizzato nel dettaglio il funzionamento del relativo progetto sull'NBA, si è passato poi ad adattarlo completamente alla nostra realtà italiana della Serie A di basket. Tutte le *features* dei giocatori descritte nella sezione precedente (minuti giocati, punti realizzati, falli fatti, ecc.) erano disponibili anche nel nostro database. Tuttavia, si sono dovuti creare alcuni dataset forniti in input al sistema quali:

- *Elenco delle gare*: questo elenco contiene tutti i risultati delle gare passate quelle future (contrassegnate da un "-1" come risultato);
- *Elenco storico dei giocatori*: tutti i giocatori contenuti nel nostro database e le relative informazioni principali (es. ruolo, altezza, ecc.);
- *Statistiche dei singoli giocatori*: in apposite cartelle (organizzate per stagioni sportive) sono contenuti tutti i file CSV (uno per ogni giocatore) con al loro interno le statistiche di tale giocatore, per ogni gara a cui ha partecipato.

In particolare, l’ultimo punto in elenco è stato affrontato creando un apposito script Python che prendesse ed elaborasse i dati direttamente dal database. Di seguito si mostrano graficamente i dataset prodotti:

date	home_team	home_team_pts	away_team	away_team_pts
...	...	...	...	...
2019-12-15	Pistoia	84	Cremona	71
2019-12-21	Cremona	93	Varese	72
2019-12-22	Pistoia	91	Pesaro	79
2019-12-22	Sassari	91	Virtus Bologna	77
2019-12-22	Fortitudo Bologna	78	Brindisi	72
2019-12-22	Milano	81	Trentino	69
2019-12-22	Reggio Emilia	85	Venezia	104
2019-12-22	Trieste	86	Cantù	96
2019-12-22	Roma	53	Brescia	83
2019-12-25	Virtus Bologna	-1	Fortitudo Bologna	-1
2019-12-26	Pesaro	-1	Cantù	-1
2019-12-26	Varese	-1	Pistoia	-1
2019-12-26	Venezia	-1	Roma	-1
2019-12-26	Brescia	-1	Cremona	-1
2019-12-26	Brindisi	-1	Reggio Emilia	-1
2019-12-26	Trentino	-1	Trieste	-1
2019-12-27	Treviso	-1	Sassari	-1

Figura 3.4: Dataset con l’elenco delle gare

name	position	height	weight	birthday
RUSSELL DeWayne	F	1.8	154	February 10, 1994
LOGAN David	F	1.85	170	December 26, 1982
CHEESE Tyler	G	1.9	176	February 1, 2000
VILDERA Giovanni	G	2.05	232	March 4, 1995
BARTOLI Vittorio	G	2.01	218	June 11, 2002
IMBRÒ Matteo	C	1.92	190	February 12, 1994
PICCIN Lorenzo	G	1.91	176	November 18, 2002
CHILLO Matteo	C	2.03	229	June 15, 1993
FORRAY Toto	C	1.88	187	March 20, 1986
SANDERS Victor	C	1.96	194	February 16, 1995
MEZZANOTTE Andrea	G	2.07	196	April 8, 1998
MORGAN Jeremy	F	1.96	194	May 8, 1995
WILLIAMS JaCorey	F	2.03	225	June 12, 1994
LADURNER Maximilian	G	2.07	209	December 6, 2001
...	...	...	...	...

Figura 3.5: Dataset con l'elenco di tutti i giocatori

date	team	opp	age	gs	mp	fg	fga	fg%	3p	3pa	3p%	ft	fta	ft%	orb	drb	trb	ast	stl	blk	tov	pf	pts	gmsc	+/-
2018-10-07	Brescia	Varese	35.1013698630137	0	17	3	6	0.500	1	3	0.333	0	0	0.0	2	1	3	2	0	0	2	1	7	4.6999	-1
2018-10-14	Brescia	Reggio Emilia	35.12054794520548	1	25	4	5	0.800	2	2	1.0	0	0	0.0	0	5	5	6	0	0	0	3	10	12.5999	7
2018-10-21	Brescia	Trieste	35.13972602739726	1	22	1	4	0.250	1	1	1.0	0	0	0.0	0	3	3	1	0	0	2	4	3	-1.4	2
2018-10-27	Brescia	Avellino	35.156164383561645	1	25	1	3	0.333	1	1	1.0	4	5	0.8	1	3	4	0	0	0	2	3	7	3.3	-11
2018-11-04	Brescia	Pesaro	35.178082191780824	1	26	3	5	0.600	1	2	0.5	2	2	1.0	2	0	2	5	1	0	1	4	9	10.0	5
2018-11-10	Brescia	Trentino	35.1945205479452	0	21	0	2	0.000	0	1	0.0	2	2	1.0	1	1	2	3	2	0	1	1	2	4.2999	0
2018-11-18	Brescia	Brindisi	35.21643835616438	0	18	0	2	0.000	0	0	0.0	0	0	0.0	1	3	4	1	0	0	2	2	0	-1.9	-16
2018-11-25	Brescia	Torino	35.23561643835617	0	18	4	5	0.000	0	0	0.0	0	0	0.0	0	2	2	4	0	0	1	2	8	7.6999	10
2018-12-09	Brescia	Pistoia	35.273972602739725	0	25	2	6	0.000	0	0	0.0	2	2	1.0	2	2	4	0	0	0	3	1	6	1.1999	12
2018-12-16	Brescia	Cantù	35.293150684931504	0	21	1	5	0.200	0	3	0.0	0	1	0.0	0	1	1	4	1	0	0	1	2	2.1999	7
2018-12-22	Brescia	Cremona	35.30958904109589	0	27	2	4	0.000	0	0	0.0	3	4	0.75	3	1	4	0	2	0	2	1	7	6.6	2
2018-12-25	Brescia	Milano	35.31780821917808	0	18	4	5	0.800	2	2	1.0	3	3	1.0	0	1	1	3	1	0	1	2	13	12.7	-10
2018-12-30	Brescia	Sassari	35.33150684931507	0	20	1	4	0.250	0	3	0.0	2	2	1.0	0	4	4	5	2	1	2	2	4	6.1999	-18
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

Figura 3.6: Dataset con l'elenco di tutte le statistiche del singolo giocatore (David Moss)

Una delle assunzioni rilevanti nel progetto è stata la presenza obbligatoria, all'interno di una squadra in ogni gara, di tutti e tre i ruoli previsti dall'articolo, ovvero Centro, Ala e Guardia. Ciò ha portato ad affrontare diversi problemi a causa delle differenze tra la Serie A italiana e l'NBA, infatti in quest'ultima il ruolo di Centro

è ben ricoperto omogeneamente all’interno delle squadre, cosa invece non vera per la pallacanestro italiana. A tale problema si è tentato di risolvere intervenendo direttamente sull’algoritmo e modificandolo in modo da, nel caso in cui in una gara una squadra non presentasse almeno un giocatore come Centro, a rimuovere il giocatore Ala più alto e sostituirlo temporaneamente per la gara in oggetto come Centro. Di seguito viene mostrato il codice appositamente creato per effettuare questa modifica:

```
def remapping_player_positions(self, match_players, player_to_position_all_players, team, year, month, day):
    position_to_player = {"F": [], "G": [], "C": []}
    player_to_position = {}

    for player in match_players:
        pos = player_to_position_all_players[player]
        position_to_player[pos].append(player)

    if len(position_to_player["C"]) == 0:
        forwards = position_to_player["F"]
        forwards_ordered_by_height = list(set(self.player_names_ordered_by_height) & set(forwards))
        new_center = forwards_ordered_by_height[0]

        position_to_player["F"].remove(new_center)
        position_to_player["C"].append(new_center)

        print("(!) Detected no Centers for", team, "on", year, month, day, ": " + new_center,
              "was temporarily moved from F to C")

    for position in position_to_player:
        for i in range(0, len(position_to_player[position])):
            player_to_position[position_to_player[position][i]] = position

    return player_to_position
```

Codice 3.12: Codice Python per la modifica temporanea dei ruoli

Implementazione

Una volta creati tutti i dataset CSV necessari e aver opportunamente modificato l’algoritmo, si è proceduto a testarlo. Aspetto particolarmente interessante dell’implementazione è stato il *train* e *test* iterativi di tali dataset. Più precisamente, si è partiti da un sostanzioso dataset delle precedenti stagioni sportive (dalla 2013/2014 alla 2017/2018), il quale è stato utilizzato in primis per addestrare il modello; successivamente gli si è chiesto di predire la prima giornata della stagione

2018/2019, e si sono analizzati i risultati per verificarne l'accuratezza. Questo procedimento è stato poi ripetuto in modo iterativo, così da allenare il modello con tutti i dati storici a disposizione e chiedendogli di predire la successiva giornata di campionato. L'ultima iterazione, ad esempio, è stata quella di addestrarlo con tutte le stagioni passate e fino alla 29° giornata della stagione sportiva 2018/2019 e chiedergli di predire la 30° giornata (sempre in termini di difficoltà delle gare). Di volta in volta, oltre al dataset delle gare, venivano ricalcolati anche tutti quelli dei singoli giocatori. Di seguito si mostra la parte principale dello script che gestisce il *train* e *test* iterativi.

```
stagioni_sportive = ['2013/2014','2014/2015','2015/2016','2016/2017','2017/2018','2018/2019']

giornate = ['1° Giornata', '2° Giornata', '3° Giornata', '4° Giornata', '5° Giornata', '6° Giornata', '7° Giornata',

all_games = pd.DataFrame(get_all_games())

previous_season_games = all_games.loc[(all_games['StagioneSportiva'] == stagioni_sportive[0]) | (all_games['StagioneS
training_games = previous_season_games.copy()
current_season_games = all_games.loc[all_games['StagioneSportiva'] == stagioni_sportive[5]]

training_games.to_csv("Predictor/database/all_games.csv", index=False)

generate_single_players_stats(stagione_sportiva=[stagioni_sportive[0], stagioni_sportive[1], stagioni_sportive[2], st

prediction_match_difficulty = pd.DataFrame(columns=["Real_Difficulty", "Predicted_Difficulty", "Winning Probability"]
difficulty_accuracy = []

predictor = TLGProb(database_path="Predictor/database/", model_path="Predictor/trained_models/")

for i in range(len(giornate)-1):
    print("\nTRAINING model with all", stagioni_sportive[0], "and", stagioni_sportive[1], "until", giornate[i], "\n")

    # Create training dataset
    # Games
    partial_games = current_season_games.loc[current_season_games['Giornata'] == giornate[i]].iloc[:, :5]
    if partial_games.shape[0] == 0:
        print("Skip training because there are no new games to train with")
        continue
    training_games = pd.concat([training_games, partial_games])
    training_games.to_csv("Predictor/database/all_games.csv", index=False)

    # Players
    generate_single_players_stats(stagione_sportiva=stagioni_sportive[1], fino_a_giornata_index=i)

    # Delete previous trained models
    shutil.rmtree("Predictor/trained_models", ignore_errors=True)

    predictor.load_data()
    predictor.train_player_models(iteration_timestamp=str(datetime.timestamp(datetime.now())))
    predictor.train_winning_team_model()

    # Prediction dataset
    print("PREDICTING", stagioni_sportive[1], "(", giornate[i+1], ")\n")

    future_games = current_season_games.loc[current_season_games['Giornata'] == giornate[i+1]].iloc[:, :6]
    if future_games.shape[0] == 0:
        print("Skip prediction because there are no games to predict")
        continue

    future_games_temp = future_games.iloc[:, :5].copy()
    future_games_temp["team1pts"] = -1
    future_games_temp["team2pts"] = -1
    temp = pd.concat([training_games, future_games_temp])
    temp.to_csv("Predictor/database/all_games.csv", index=False)

    predictor.load_data()
    prediction_result = predictor.generate_next_prediction()
    predicted_games = pd.DataFrame(prediction_result,
                                   columns=['year', 'month', 'day', 'team1', 'team2', 'Pred Winning Team',
                                            'Winning Probability', 'Points Differential 95% C.I. team1',
                                            'Points Differential 95% C.I. team2'])
```

Codice 3.13: Codice Python del *train* e *test* iterativi

3.5.3 Algoritmo non supervisionato di *clustering* per i ruoli dei giocatori

A seguito dei risultati ottenuti, che verranno approfonditi nel dettaglio nell'ultima sezione di questo capitolo "Valutazione risultati", si è cercato di migliorare ulteriormente l'algoritmo basandosi su una metrica di valutazione dello stesso, il NMSE (Normalised Mean Square Error). Per fare ciò si è analizzato un aspetto molto delicato per quest'ultimo: il *clustering* dei ruoli dei giocatori. Per quanto concerneva l'autore dell'articolo, i ruoli dei giocatori dell'NBA erano facilmente reperibili da famosi siti di statistiche per il campionato americano, come ad esempio "Basketball Reference"[23]. In Italia, sono altresì riportati i ruoli dei giocatori, tuttavia in alcuni casi possono risultare non essere così accurati come per l'NBA. A tal proposito, si è costruito un algoritmo non supervisionato che cercasse di assegnare un ben preciso ruolo al giocatore tenendo in forte considerazione le sue statistiche, ma anche quelle degli altri giocatori. Infatti, l'algoritmo di clustering tiene conto dell'omogeneità tra i dati e le misure relative alla somiglianza tra gli elementi.
L'algoritmo utilizzato è stato il K-Means, impostando a 3 il numero di cluster. I dati sono stati scalati come approfondito nel precedente capitolo, sempre tramite il `MinMaxScaler` di scikit-learn in un intervallo da 0 a 1, come mostrato di seguito:

[23] `https://www.basketball-reference.com/`

```
df = pd.read_csv("single_players.csv")

df_X_stats = df.iloc[:, 5:27]

scaler = MinMaxScaler()
scaler.fit(df_X_stats)
df_X_stats = pd.DataFrame(scaler.transform(df_X_stats))

n_clusters = 3

kmeans = KMeans(n_clusters=n_clusters, init='k-means++', max_iter=300, n_init=10, random_state=0)
kmeans.fit(df_X_stats)

clusters = pd.DataFrame(transform_cluster_to_role(kmeans.labels_), columns=["ruolo"])

clusters.reset_index(drop=True, inplace=True)
df.reset_index(drop=True, inplace=True)

df_cluster = pd.concat([clusters, df], axis=1)

valutazione_ruoli(df_cluster)
```

Codice 3.14: Codice Python del clustering per i ruoli dei giocatori utilizzando il K-Means

Per ogni giocatore, una volta analizzate tutte le statistiche (anche in relazione a quelle degli altri giocatori), l’algoritmo fornisce per ogni gara in cui ha giocato il ruolo attribuitogli secondo il K-Means. Dopodiché viene calcolata la frequenza del ruolo più presente e scelto come ruolo definitivo. Di seguito si mostra una parte del clustering ottenuto e la tabella riassuntiva per esaminare i risultati.

player	C	F	G	main_role	main_role_frequency
PRUNOTTO Marco	0	0	3	G	1.0
FARLEY Liam	0	0	22	G	1.0
GRAVAGHI Francesco	0	0	7	G	1.0
TELESCA Samuele	0	0	1	G	1.0
VOLTOLINI Alessandro	0	0	8	G	1.0
KYZLINK Tomas	0	21	1	F	0.9545454545454546
BUFORD William	1	21	0	F	0.9545454545454546
JONES DeQuan	3	17	0	F	0.85
JERRELLS Curtis	17	1	2	C	0.85
SEVERINI Luca	3	0	15	G	0.8333333333333334
RICHARDSON Malachi	1	5	0	F	0.8333333333333334
AKELE Nicola	2	16	2	F	0.8
JAKOVICS Ingus	15	0	4	C	0.7894736842105263
MEKEL Gal	2	15	2	F	0.7894736842105263
JOHNSON-ODOM Darius	5	16	0	F	0.7619047619047619
JUSTICE Kodi	3	15	2	F	0.75
...	...	...	...	...	...

Intervallo frequenza	Conteggio
0.0 – 0.3	0
0.3 – 0.5	25
0.5 – 0.7	67
0.7 – 1.0	180
	272

Tabella 3.2: Risultati del clustering per i ruoli dei giocatori utilizzando il K-Means

Come si può notare, accanto ad ogni giocatore è riportato il numero di volte in cui le sue performance sono state classificate come Centro (C), Ala (F) e Guardia (G). A secondo del numero più alto tra questi, viene scelto il ruolo definitivo del giocatore. Come mostrato nella tabella riassuntiva a destra, per la maggior parte dei giocatori è predominante un ruolo rispetto agli altri (più del 70% delle gare in quel ruolo); tuttavia, vi è anche una parte giocatori particolarmente versatili per i quali non è stato possibile distinguere in modo chiaro il loro ruolo principale. Molto probabilmente perché quest'ultimo varia non solo nel corso della stagione sportiva, ma anche in relazione alle partite giocate e agli avversari.

3.6 *Training* e *test* iterativi con dataset pesati dinamicamente

Dall'analisi approfondita dell'articolo scientifico "*One-Match-Ahead Forecasting In Two-Team Sports With Stacked Bayesian Regressions*", infine, sono emersi alcuni spunti davvero interessanti, tra cui il più rilevante è il sistema di pesi implementato. Si tratta concettualmente di dare più rilevanza alla performance recenti

e meno a quelle del passato. Tale concetto è stato quindi aggiunto alle implementazioni passate, ovvero quelle basate sugli algoritmi supervisionati di Machine Learning.
Più nello specifico, è stato creato uno script Python in grado di creare sempre i dataset visti all'inizio di questo capitolo, ma le cui medie storiche fossero pesate. La metrica di riferimento è stata la differenza in giorni tra la gara da predire e quelle passate. Si è deciso di intraprendere questa strada invece di considerare in modo statico le ultime n gare del giocatore poiché potrebbe essere passato molto tempo dalla sua ultima partita (ad esempio se tornato da un lungo infortunio oppure all'inizio di una nuova stagione sportiva). Una volta calcolata la differenza assoluta in giorni tra la gara da predire e tutte le passate, i criteri di attribuzione dei pesi alle *features* sono stati i seguenti:

Intervallo di giorni trascorsi	Peso assegnato
<30	0.4
30 – 59	0.3
60 – 89	0.2
90+	0.1

Tabella 3.3: Pesi assegnati in base all'intervallo temporale

Così facendo si è cercato di creare intervalli semi regolari per suddividere le finestre temporali. Queste ultime sono state impostate relativamente anche alla frequenza delle gare (circa una a settimana) e alla durata della stagione sportiva (da fine ottobre ad inizio maggio, la stagione regolare). L'algoritmo supervisionato scelto per applicare questa tecnica è stato il SVM. Di seguito è mostrato il codice principale dello script che itera sulle singole gare.

```
database = mysql.connector.connect(
    host="localhost",
    user="root",
    password="root",
    database="progetto_ai"
)

cursor = database.cursor(dictionary=True)
cursor.execute('''
    SELECT IdGara, StagioneSportiva, Fase, Giornata, Data, Difficolta
    FROM dataset_ml
    WHERE StagioneSportiva IN ('2017/2018', '2018/2019')
    ORDER BY Data
''')
gare = cursor.fetchall()

gare = pd.DataFrame(gare)

accuracy_result = []
accuracy_result_giornate = {}
plot_x_giorni = []
plot_y_giorni = []
plot_x_giornate = []
plot_y_giornate = []

rendimento_giocatori = RendimentoGiocatori(database=database)

i = 231  # train with the first games (all 2017/2018)
while i < len(gare) - 1:
    index_up_to_train = i + 1
    gare_train = gare.iloc[:index_up_to_train, :]
    gare_test = gare.iloc[index_up_to_train:(index_up_to_train + 1), :]

    id_gare_train = gare_train["IdGara"].tolist()
    id_gare_test = gare_test["IdGara"].tolist()
    data_gara_test = gare_test["Data"].tolist()
    giornata_gara_test = gare_test["Giornata"].tolist()
    target_match_date = str(data_gara_test[0])
    target_match_giornata = str(giornata_gara_test[0])
    target_match_giornata = target_match_giornata.replace(" Giornata", "")

    if target_match_giornata not in accuracy_result_giornate:
        accuracy_result_giornate[target_match_giornata] = []

    rendimento_giocatori.generate_training_dataset(id_gare_train, target_match_date)
    rendimento_giocatori.generate_test_dataset(id_gare_test, target_match_date)

    # Train
    df_train = pd.read_csv("generated_dataset/training_dataset.csv")
    df_train = pd.concat([df_train, gare_train["Difficolta"]], axis=1)

    df_train_X = pd.DataFrame(df_train.iloc[:, :-1])
    df_train_Y = pd.DataFrame(df_train["Difficolta"])

    scaler = scaler_data(df_train_X)
    df_train_X = scaler.transform(df_train_X)

    trained_model = support_vector_machine(X_train=df_train_X, Y_train=df_train_Y)

    # Prediction
    df_test = pd.read_csv("generated_dataset/dataset.csv")
    df_test_X = pd.DataFrame(df_test)

    df_test_X = scaler.transform(df_test_X)

    prediction = trained_model.predict(df_test_X)
```

```
105
106   # Evaluation
107   difficolta_gara_test_temp = gare_test["Difficolta"].tolist()
108   difficolta_gara_test = int(difficolta_gara_test_temp[0])
109
110   prediction_result = True if prediction == difficolta_gara_test else False
111   accuracy_result.append(prediction_result)
112
113   print(target_match_date)
114   print("Prediction:", prediction, "(", prediction_result, ")", "was:", difficolta_gara_test)
115
116   total_predicted_matches = len(accuracy_result)
117   avg_accuracy = accuracy_result.count(True) / total_predicted_matches
118
119   accuracy_result_giornate[target_match_giornata].append(avg_accuracy)
120
121   print("avg_accuracy =", avg_accuracy)
122   print("accuracy_result_giornate")
123   print(accuracy_result_giornate)
124
125   diff_days = rendimento_giocatori.days_between("2018-10-13", target_match_date)
126   plot_x_giorni.append(str(diff_days))
127   plot_y_giorni.append(avg_accuracy)
128
129   i = i+1
```

Codice 3.15: Codice Python dell'algoritmo iterativo di *train* e *test*

3.7 Valutazione risultati

A conclusione di questo capitolo sulla predizione della difficoltà delle gare, vengono mostrati e commentati i risultati ottenuti con tutte le tecniche illustrate fino ad ora, dagli algoritmi supervisionati al progetto "TLGProb" e al *training* e *test* iterativi con *dataset* pesati dinamicamente.
Per i sei *dataset* illustrati ad inizio capitolo si sono eseguiti tutti gli algoritmi supervisionati visti precedentemente, quali:

- Regressione Logistica;
- Support Vector Machine;
- Random Forest Classifier;

- Decision Tree Classifier;
- Gaussian NB;
- Gradient Boosting Classifier;
- K-Nearest Neighbors Classifier;
- NN Feed-forward.

Per il progetto "TLGProb" sono stati utilizzati una serie di dataset descritti nel rispettivo capitolo (elenco gare, elenco giocatori ed elenco statistiche per ogni giocatore per ogni gara). Infine, per il train e test iterativo è stato usato un apposito dataset dinamicamente costruito, contenente però solo le medie storiche dei Voti Lega (ultime 999, 10, 5, 3, 2, 1 gare). Tutti gli algoritmi supervisionati sono stati impiegati per predire le difficoltà della stagione sportiva 2018/2019, sulla base delle precedenti stagioni dalla 2013/2014 fino alla 2017/2018.

Per ogni algoritmo supervisionato saranno commentate le specifiche caratteristiche di ognuno e alcune informazioni aggiuntive (prevalentemente, a titolo rappresentativo, per quanto riguarda il prima dataset). Tutte le informazioni sull'*accuracy* (accuratezza), rappresentano in un intervallo 0 – 1 (0% – 100%, in percentuale) il numero relativo di difficoltà correttamente predette, rispetto al totale.

Dataset 1

Il primo dataset testato conteneva due metriche sulla classifica (posizione e punti) e sei sulle medie dei Voti Lega (ultime 999, 10, 5, 3, 2, 1 gare), e la variabile target "Difficoltà" (della gara). I risultati di accuracy sono stati i seguenti.

	Accuracy
Regressione Logistica	0.3969
Support Vector Machine	0.41
Random Forest Classifier	0.3606
Decision Tree Classifier	0.3878
Gaussian NB	0.3545
Gradient Boosting Classifier	0.3545
K-Nearest Neighbors Classifier	0.34
NN Feed-forward	**0.4212**

Tabella 3.4: Accuracy ottenute per gli algoritmi supervisionati con il Dataset 1

L'algoritmo che ha ottenuto il risultato migliore per questo primo dataset si è rivelato essere la rete neurale Feed-forward, con il 42.12% di accuracy.

Regressione Logistica La Regressione Logistica, per il primo dataset, si è attestata al 39.69% di accuracy, producendo una matrice di confusione così rappresentata:

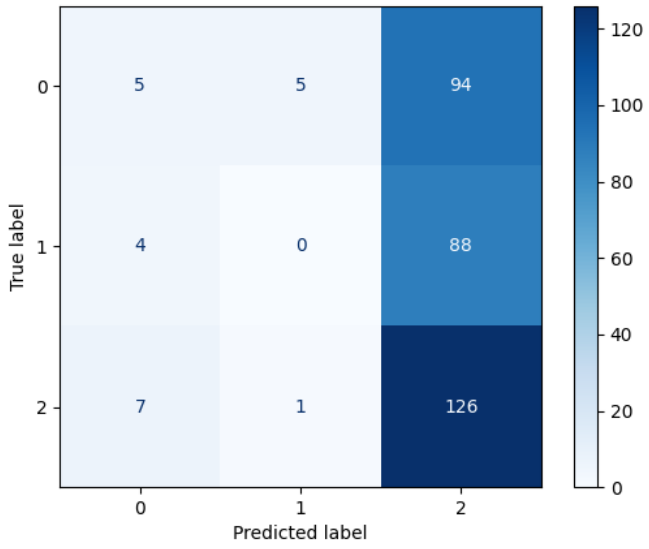


Figura 3.7: *Confusion Matrix* per la Regressione Logistica sul Dataset 1

La *confusion matrix*, permette di analizzare numericamente per ogni classe reale (asse Y, "True label") qual è stata la classe predetta (asse X, "Predicted label").

Come si può notare, per le gare con difficoltà 2 ("Difficile") l'algoritmo è in grado di classificarle col 94% di accuratezza. Tuttavia, per le altre due classi di difficoltà ("Facile" e "Media") l'algoritmo non riesce praticamente mai a classificarle correttamente, se non alcuni rari casi. Scikit-learn offre, inoltre, un'esaustiva serie di parametri di valutazione degli algoritmi supervisionati tramite la funzione `classification_report`, mostrati di seguito.

	precision	recall	f1-score	support
0	0.31	0.05	0.08	104
1	0.00	0.00	0.00	92
2	0.41	0.94	0.57	134
accuracy			0.40	330
macro avg	0.24	0.33	0.22	330
weighted avg	0.26	0.40	0.26	330

Figura 3.8: Parametri di valutazione della Regressione Logistica sul Dataset 1

Nel dettaglio, la *precision* è il rapporto $\frac{tp}{(tp+fp)}$, dove tp corrisponde al numero dei veri positivi, e fp il numero di falsi positivi; ed indica l'abilità del classificatore di non predire una classe come positiva se, invece, è negativa. La *precision* pari al 41% per la classe 2 (gare "Difficili") indica da parte del modello una discreta precisione nella classificazione di tali gare; similarmente anche per la classe 0 (gare "Facili").
La *recall*, invece, è il rapporto $\frac{tp}{(tp+fn)}$, dove tp corrisponde sempre al numero dei veri positivi, ma fn al numero di falsi negativi. Questa metrica permette di valutare quindi l'abilità del classificatore di individuare correttamente tutti i positivi. Infatti, il 94% per la classe 2 (gare "Difficili") evidenzia come il modello sia notevolmente accurato nel predire tale classe, e però altrettanto non accurato nel predire le classi 0 e 1 ("Facili" e "Medie").

L'*f1-score* può essere interpretato come la media armonica pesata della *precision* e della *recall*, dove raggiunge il suo massimo (migliore) a 1 e il suo peggior valore a 0.
Il *support*, infine, indica il numero delle occorrenze in ogni classe.

Support Vector Machine La Support Vector Machine si è rilevata essere, tra tutti gli algoritmi supervisionati utilizzati, quella più performante, anche al variare dei diversi dataset. Tra i diversi kernel implementati, sia il Poly che l'RBF hanno ottenuto il 40% di accuracy, uno tra i più alti risultati conseguiti.

Test set	Accuracy
Linear	0.41
Poly	0.40
RBF	0.40
RBF Best	0.39

Tabella 3.5: Accuracy ottenute per i diversi kernel del SVM con il Dataset 1

Random Forest Classifier Il Random Forest Classifier, oltre a fornire un'accuracy delle predizioni, risultata essere pari al 36.06%, fornisce anche un'interessante classifica (*ranking*) delle features, mostrando per ognuna il grado di rilevanza che ha assunto.

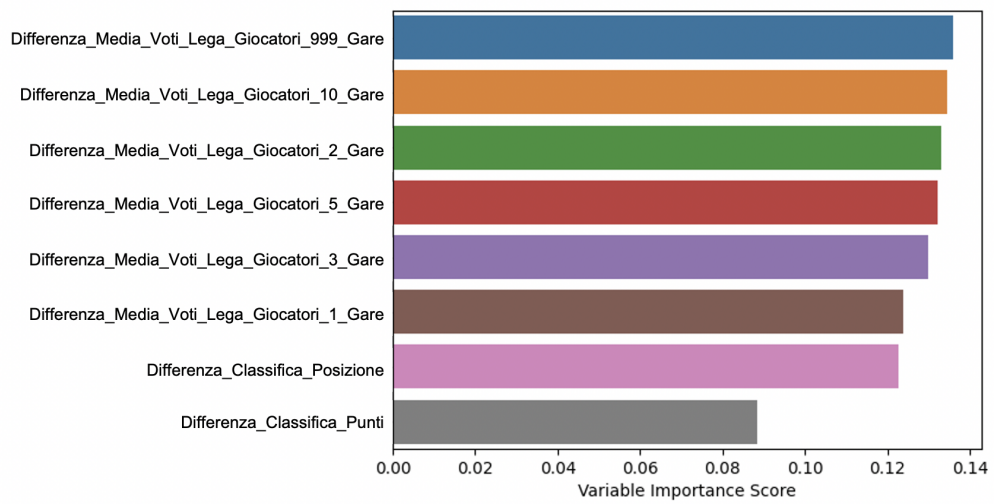


Figura 3.9: *Ranking* delle variabili per il Random Forest Classifier sul Dataset 1

Rispetto alla totalità delle features, le metriche più rilevanti, come si può notare, sono state le differenze delle medie dei Voti Lega per le ultime 999, 10, 2, 5 e 3 gare. Meno importante è stata la media dell'ultima gara, ritenuta quindi meno affidabile, così come la differenza dei punti in classifica tra le squadre.

Decision Tree Classifier Il Decision Tree Classifier si distingue dagli altri algoritmi per la sua semplicità di comprensione. Infatti, oltre a raggiungere il 38.78% di accuracy, esso esplicita nel dettaglio quali sono state le scelte che hanno portato alla classificazione della difficoltà di una gara. Per questo primo dataset si mostra sia la visualizzazione grafica dell'albero decisionale, che la trascrizione dell'output.

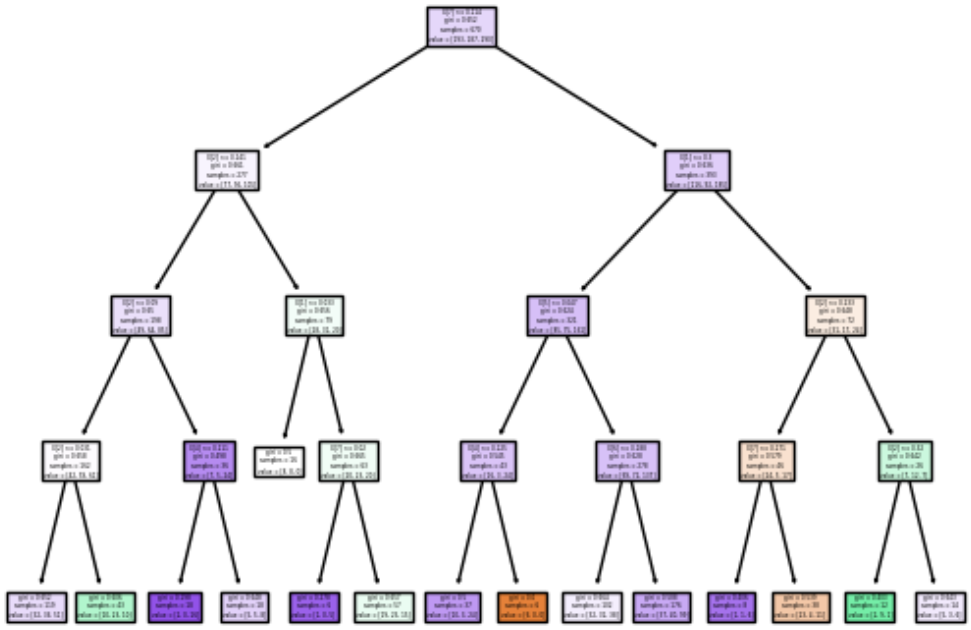


Figura 3.10: Albero decisionale generato dal Decision Tree Classifier sul Dataset 1

```
|--- feature_7 <= 0.11
|   |--- feature_2 <= 0.14
|   |   |--- feature_2 <= 0.09
|   |   |   |--- feature_2 <= 0.03
|   |   |   |   |--- class: 2
|   |   |   |--- feature_2 >  0.03
|   |   |   |   |--- class: 1
|   |   |--- feature_2 >  0.09
|   |   |   |--- feature_4 <= 0.11
|   |   |   |   |--- class: 2
|   |   |   |--- feature_4 >  0.11
|   |   |   |   |--- class: 2
```

```
|   |--- feature_2 >  0.14
|   |   |--- feature_1 <= 0.03
|   |   |   |--- class: 0
|   |   |--- feature_1 >  0.03
|   |   |   |--- feature_7 <= 0.02
|   |   |   |   |--- class: 2
|   |   |   |--- feature_7 >  0.02
|   |   |   |   |--- class: 1
|--- feature_7 >  0.11
|   |--- feature_1 <= 0.30
|   |   |--- feature_5 <= 0.05
|   |   |   |--- feature_4 <= 0.12
|   |   |   |   |--- class: 2
|   |   |   |--- feature_4 >  0.12
|   |   |   |   |--- class: 0
|   |   |--- feature_5 >  0.05
|   |   |   |--- feature_6 <= 0.19
|   |   |   |   |---  class: 2
|   |   |   |--- feature_6 >  0.19
|   |   |   |   |--- class: 2
|   |--- feature_1 >  0.30
|   |   |--- feature_2 <= 0.23
|   |   |   |--- feature_7 <= 0.17
|   |   |   |   |--- class: 2
|   |   |   |--- feature_7 >  0.17
|   |   |   |   |--- class: 0
|   |   |--- feature_2 >  0.23
|   |   |   |--- feature_2 <= 0.32
|   |   |   |   |--- class: 1
|   |   |   |--- feature_2 >  0.32
|   |   |   |   |--- class: 2
```

Figura 3.11: Output del Decision Tree Classifier per il Dataset 1

In particolare, è interessante notare come la feature principale sulla base della quale l'algoritmo inizia a costruire l'albero sia la "feature_7", ovvero "Differen-

za_Media_Voti_Lega_Giocatori_2_Gare", il cui valore soglia è pari a 0.11.

Gaussian NB Il Gaussian Naive Bayes ha raggiunto risultati meno soddisfacenti, garantendo solamente un'accuratezza del 35.45%. Tuttavia, l'algoritmo fornisce alcuni utili strumenti di analisi, quali:

- *Mean of the Gaussian Estimators* (offre la media di tutti gli stimatori);
- *Std Dev of the Gaussian Estimators* (la deviazione standard degli estimatori);
- *QDA means per class* (la media per la Quadratic Discriminant Analysis di ogni classe);
- *QDA covariance per class* (la covarianza per la Quadratic Discriminant Analysis di ogni classe).

Di seguito ne è mostrato l'output (abbreviato).

```
Mean of the Gaussian Estimators
[[0.35164076 0.16908463 0.1438163  ... 0.16856966 0.17135176 0.18644588]
 [0.34545455 0.13172906 0.15369926 ... 0.17151882 0.17021741 0.16770458]
 [0.32965517 0.14988506 0.1517608  ... 0.17539148 0.18701547 0.20152641]]

Std Dev of the Gaussian Estimators
[[0.06448925 0.03029928 0.01683025 ... 0.02278668 0.02320289 0.02485289]
 [0.06022147 0.0173058  0.0244517  ... 0.02765306 0.02989189 0.03049386]
 [0.06187153 0.02345784 0.02434462 ... 0.02733375 0.02950675 0.03018577]]

QDA means per class
[[0.35164076 0.16908463 0.1438163  ... 0.16856966 0.17135176 0.18644588]
 [0.34545455 0.13172906 0.15369926 ... 0.17151882 0.17021741 0.16770458]
 [0.32965517 0.14988506 0.1517608  ... 0.17539148 0.18701547 0.20152641]]

QDA covariance per class
[array([[ 0.06482513, 0.03163374,  ... , 0.00083437],
        [ 0.03163374, 0.03045709,  ... , 0.00485175],
       ...
       ),
 array([[ 6.0545237e-02, 1.9768654e-02, ... , -1.2393528e-03, 2.4461976e-04],
       [ 1.9768654e-02, 1.7398846e-02, ... , 2.5977411e-03, 2.8762606e-03],
       ...
       )]
```

Figura 3.12: Output del Gaussian NB per il Dataset 1

K-Nearest Neighbors Classifier Il K-Nearest Neighbors Classifier è l'algoritmo che, per il primo dataset, ha raggiunto il più basso risultato: solamente il 34% di accuratezza. Tuttavia fornisce, se non altro, una rapida visualizzazione della rispettiva classe predetta per ogni gara appartenente al test set. Eccone un esempio dell'output:

```
Test set predictions: [ 2 0 1 1 0 0 1 2 0 0 1 0 1 2 2 1 0 1 1 2 1
 2 0 1 0 2 0 2 2 1 0 2 2 0 2 0 0 2 2 1 0 2 1 2 0 0 1 0 0 1 0 2 1
 0 2 1 2 0 1 2 2 1 0 0 0 0 2 2 1 0 2 0 2 1 1 0 0 2 2 1 0 0 0 1 2
 1 2 2 2 0 0 2 1 1 2 2 2 0 0 0 1 0 0 2 2 2 0 0 2 1 0 2 2 0 0 2 1
 1 1 2 0 2 0 2 2 0 1 2 2 2 2 2 2 1 2 2 0 0 1 2 2 1 0 2 1 0 1 0 2
 2 0 2 0 2 0 0 0 2 1 0 0 1 1 0 1 0 0 0 0 1 0 2 0 2 1 2 1 0 2 2 0
 2 0 2 0 0 0 2 0 2 0 2 0 2 2 0 1 2 2 2 1 2 2 0 0 0 0 0 1 0 2 1 0
 0 1 1 0 1 2 2 0 0 1 0 1 1 0 2 0 2 1 2 0 2 0 0 0 0 2 0 2 0 2 1 2
 1 0 1 1 0 0 2 0 2 2 1 2 2 1 2 1 2 1 2 1 0 0 0 0 1 2 2 0 0 2 0 1
 2 2 1 0 2 2 2 1 2 2 1 1 2 2 0 0 1 0 2 2 0 1 1 2 2 2 1 2 2 0 2 2
 0 0 0 1 2 2 1 1 2 0 0 2 2 2 0 1 1 0 2 2 2 ]
```

Figura 3.13: Output del K-Nearest Neighbors Classifier per il Dataset 1

NN Feed-forward La rete neurale Feed-forward, eseguita per classificare sempre le tre diverse difficoltà delle gare, è stata eseguita con 2 *hidden layer* e per 1.000 epoche. Oltre ad un grafico che mostra con il passare delle epoche l'andamento del *Loss Value* (con la Cross-entropy), metrica utilizzata per rappresentare numericamente l'errore che sta commettendo il modello, è possibile analizzare dettagliatamente anche ogni singola epoca.

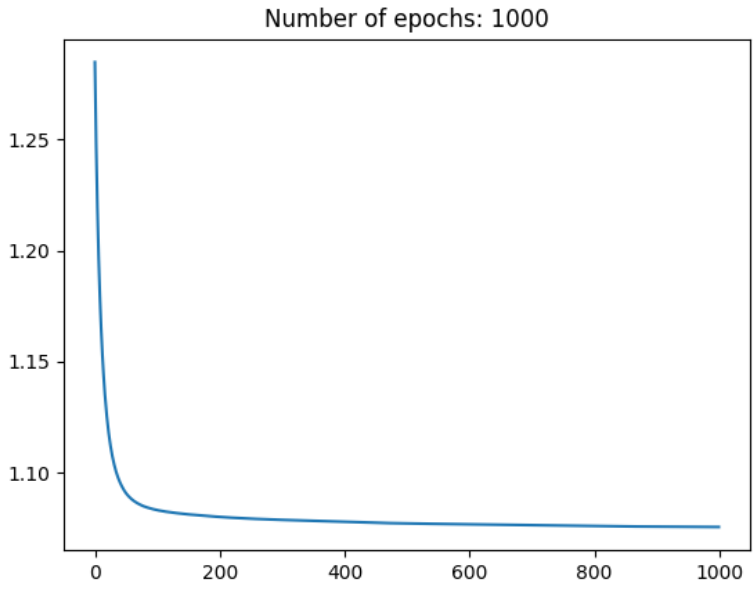


Figura 3.14: Grafico dell'andamento del *Value Loss* per la NN Feed-forward sul Dataset 1

```
Test score 0.3969697
Epoch 0 train loss:   1.2636033296585083
Epoch 1 train loss:   1.2466917037963867
Epoch 2 train loss:   1.2327454090118408
Epoch 3 train loss:   1.2214875221252441
...
Epoch 8 train loss:   1.183471441268921
Epoch 9 train loss:   1.1791205406188965
Epoch 10 train loss:  1.175605297088623
...
Epoch 99 train loss:  1.1118346452713013
Epoch 100 train loss: 1.1116269826889038
Epoch 101 train loss: 1.11142098903656
```

```
...
Epoch 498 train loss: 1.0829801559448242
Epoch 499 train loss: 1.082961916923523
Epoch 500 train loss: 1.0829460620880127
Epoch 501 train loss: 1.0829249620437622
...
Epoch 998 train loss: 1.0801498889923096
Epoch 999 train loss: 1.0801470279693604
Test score 0.4212121
```

Figura 3.15: Output della NN Feed-forward per il Dataset 1

In particolare, si nota come già dopo poche epoche (circa 10) l'errore si riduca fino a raggiungere le 200 epoche, dopo le quali diminuisce molto più lentamente. Inoltre, interessante è confrontare il test effettuato prima dell'inizio, pari al 39.69% e quello finale, attestatosi al 42.12%. La rete neurale Feed-forward è risultata essere la più performante in assoluto, battendo tutti gli algoritmi supervisionati visti fino ad ora.

Dataset 2

Il secondo dataset prodotto è stato quello con meno metriche, infatti conteneva solamente le due principali sulla classifica: posizione e punti, oltre chiaramente alla variabile Y "Difficoltà". Le accuracy ottenute dagli algoritmi supervisionati sono state le seguenti:

	Accuracy
Regressione Logistica	0.4
Support Vector Machine	0.41
Random Forest Classifier	0.3484
Decision Tree Classifier	0.3787
Gaussian NB	0.4
Gradient Boosting Classifier	0.3606
K-Nearest Neighbors Classifier	0.35
NN Feed-forward	**0.4212**

Tabella 3.6: Accuracy ottenute per gli algoritmi supervisionati con il Dataset 2

Le performance ottenute sono risultate essere similari al primo dataset, tuttavia con una maggior precisione rilevata dal Gaussian NB e K-Nearest Neighbors Classifier.

Regressione Logistica La Regressione Logistica è riuscita, su questo particolare dataset, a raggiungere il 40% di accuratezza, mostrando però comunque una confusion matrix fortemente sbilanciata per quanto riguarda le classi di difficoltà delle gare “Facile” e “Media”.

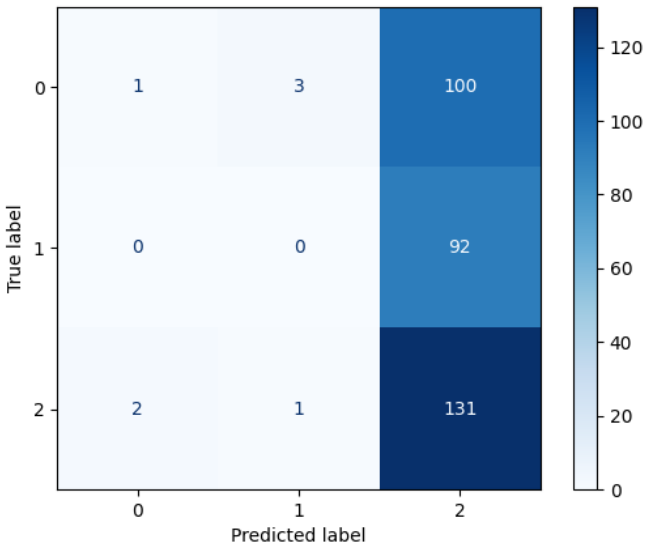


Figura 3.16: *Confusion Matrix* per la Regressione Logistica sul Dataset 2

Support Vector Machine La Support Vector Machine, come evidenziato anche negli altri dataset, ha sempre un comportamento molto lineare, anche all'interno degli stessi kernel offerti.

Test set	Accuracy
Linear	0.41
Poly	0.41
RBF	0.42
RBF Best	0.41

Tabella 3.7: Accuracy ottenute per i diversi kernel del SVM con il Dataset 2

Random Forest Classifier Per il Random Forest Classifier, in questo dataset la metrica più rilevante (con 0.6 a fronte di 0.4) è la posizione in classifica delle squadre, piuttosto che i punti.

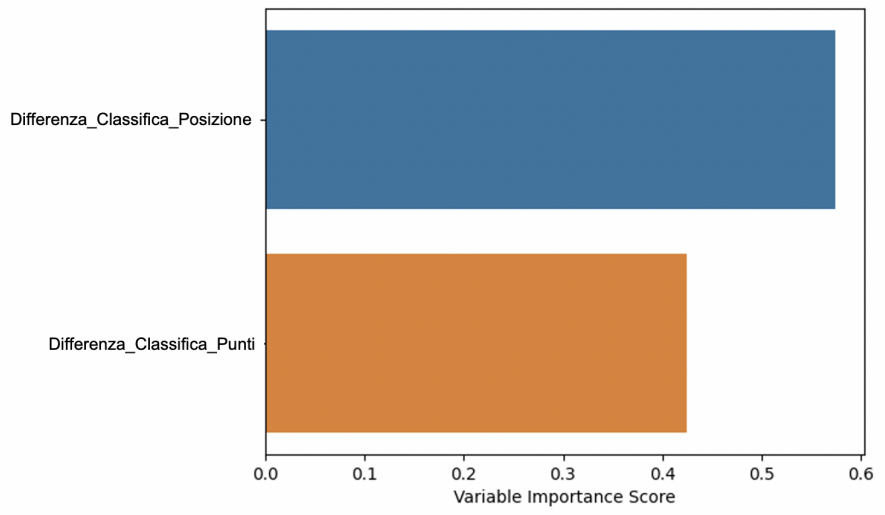


Figura 3.17: *Ranking* delle variabili per il Random Forest Classifier sul Dataset 2

Decision Tree Classifier Il Decision Tree Classifier, che ha ottenuto il 37.87% di accuratezza, mostra chiaramente come l'unica feature presa in considerazione sia la posizione in classifica. In particolare, il valore soglia (rapportato al `MinMaxScaler` $0-1$) di quest'ultima è risultata essere pari a 0.30.

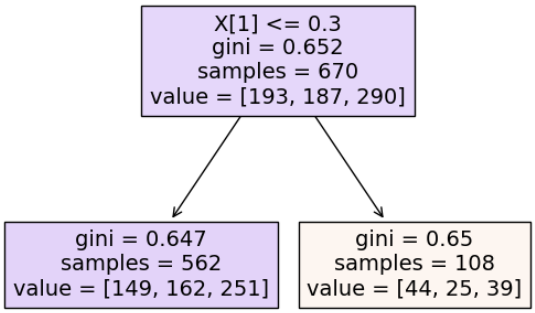


Figura 3.18: Albero decisionale generato dal Decision Tree Classifier sul Dataset 2

```
|--- feature_1 <= 0.30
|   |--- class: 2
|--- feature_1 >  0.30
|   |--- class: 0
```

Figura 3.19: Output del Decision Tree Classifier per il Dataset 2

NN Feed-forward La rete neurale Feed-forward, per il secondo dataset, è riuscita a raggiungere una percentuale di accuratezza pari al 42.12% ed un *Loss Value* leggermente inferiore rispetto al dataset precedente, che asintoticamente tende a 1.08.

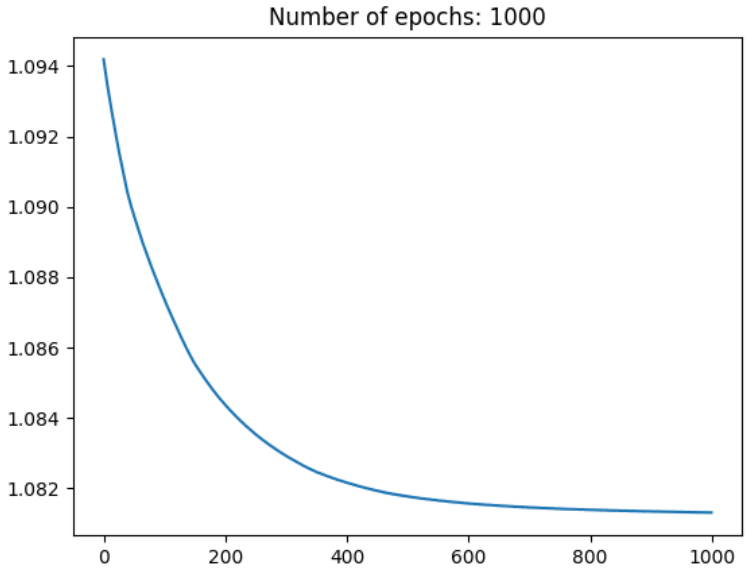


Figura 3.20: Grafico dell'andamento del *Value Loss* per la NN Feed-forward sul Dataset 2

Dataset 3

Il terzo dataset contiene, invece, oltre alla variabile target Y, le sole medie dei Voti Lega delle ultime 999, 10, 5, 3, 2 e 1 gara. Ancora una volta la rete neurale

riesce ad ottenere il miglior risultato, cui però tende anche il SVM e la Regressione Logistica. Di seguito le accuratezze nel dettaglio:

	Accuracy
Regressione Logistica	0.4
Support Vector Machine	0.41
Random Forest Classifier	0.3606
Decision Tree Classifier	0.3878
Gaussian NB	0.3696
Gradient Boosting Classifier	0.3333
K-Nearest Neighbors Classifier	0.33
NN Feed-forward	**0.4242**

Tabella 3.8: Accuracy ottenute per gli algoritmi supervisionati con il Dataset 3

Random Forest Classifier Il Random Forest Classifier, tra le diverse medie proposte, ha individuato come più rilevante quella riferita alle ultime 999 gare. Questo delinea come, per lo specifico dataset, sia più importante il rendimento storico di una squadra, rispetto a quello più recente.

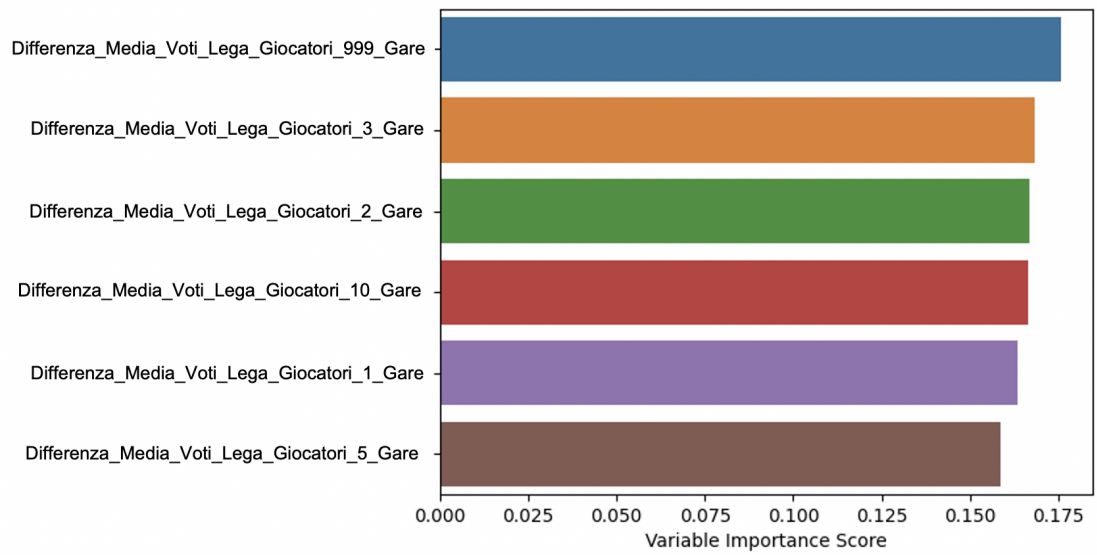


Figura 3.21: *Ranking* delle variabili per il Random Forest Classifier sul Dataset 3

Decision Tree Classifier Il Decision Tree Classifier, invece, ha riscontrato come prima feature per la ramificazione dell'albero, la Differenza_Media_Voti_Lega_Giocatori_2_Gare; consentendogli di ottenere un'accuratezza media del 38.78%.

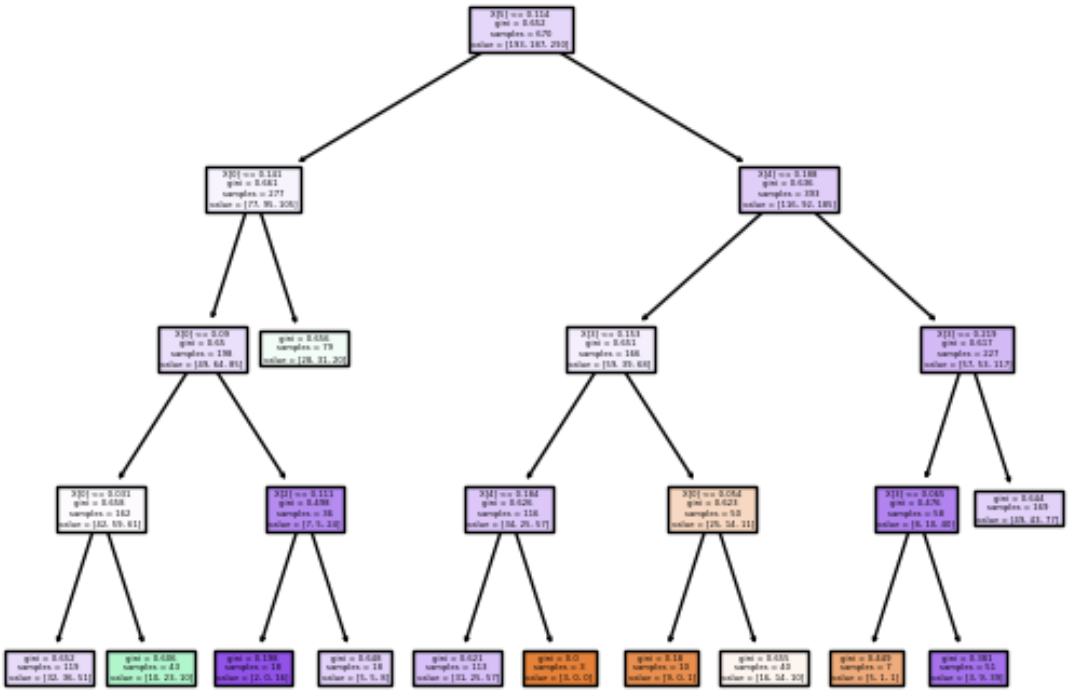


Figura 3.22: Albero decisionale generato dal Decision Tree Classifier sul Dataset 3

```
|--- feature_5 <= 0.11
|   |--- feature_0 <= 0.14
|   |   |--- feature_0 <= 0.09
|   |   |   |--- feature_0 <= 0.03
|   |   |   |   |--- class: 2
|   |   |   |--- feature_0 >  0.03
|   |   |   |   |--- class: 1
|   |   |--- feature_0 >  0.09
|   |   |   |--- feature_2 <= 0.11
|   |   |   |   |--- class: 2
|   |   |   |--- feature_2 >  0.11
|   |   |   |   |--- class: 2
```

```
|   |--- feature_0 >  0.14
|   |   |--- class: 1
|--- feature_5 >  0.11
|   |--- feature_4 <= 0.19
|   |   |--- feature_3 <= 0.15
|   |   |   |--- feature_4 <= 0.18
|   |   |   |   |--- class: 2
|   |   |   |--- feature_4 >  0.18
|   |   |   |   |--- class: 0
|   |   |--- feature_3 >  0.15
|   |   |   |--- feature_0 <= 0.05
|   |   |   |   |--- class: 0
|   |   |   |--- feature_0 >  0.05
|   |   |   |   |--- class: 0
|   |--- feature_4 >  0.19
|   |   |--- feature_3 <= 0.22
|   |   |   |--- feature_3 <= 0.07
|   |   |   |   |--- class: 0
|   |   |   |--- feature_3 >  0.07
|   |   |   |   |--- class: 2
|   |   |--- feature_3 >  0.22
|   |   |   |--- class: 2
```

Figura 3.23: Output del Decision Tree Classifier per il Dataset 3

NN Feed-forward La rete neurale, per quanto riguarda il terzo dataset, è riuscita a battere tutti gli altri algoritmi supervisionati, attestandosi al 42.42% di accuracy; e con un *Value Loss* che, come per il dataset precedente, converge asintoticamente all'1.08. Il numero di epoche entro il quale questo valore si stabilizza è individuato all'incirca per 600.

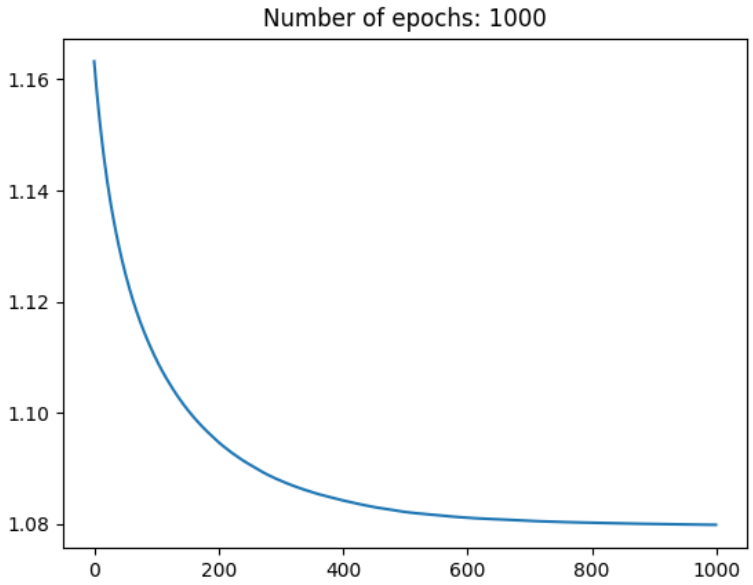


Figura 3.24: Grafico dell'andamento del *Value Loss* per la NN Feed-forward sul Dataset 3

Dataset 4

Il quarto e penultimo dataset analizzato conteneva le sole metriche sulla classifica, quali: posizione, punti, totale partite vinte e perse, oltre chiaramente alla variabile target Y. Le accuracy riportate sono state le seguenti:

	Accuracy
Regressione Logistica	0.4
Support Vector Machine	0.41
Random Forest Classifier	0.3606
Decision Tree Classifier	0.3878
Gaussian NB	0.3696
Gradient Boosting Classifier	0.3333
K-Nearest Neighbors Classifier	0.33
NN Feed-forward	**0.4242**

Tabella 3.9: Accuracy ottenute per gli algoritmi supervisionati con il Dataset 4

Anche per questo dataset il miglior risultato è stato raggiunto dalla rete neurale Feed-forward alla quale seguono, con un'accuratezza di poco inferiore, la SVM e la Regressione Logistica.

Random Forest Classifier Il Random Forest Classifier, tra tutte le metriche della classifica analizzato, ha individuato come nettamente predominante la posizione. Anche grazie ad essa, l'algoritmo è riuscito a raggiungere il 41% di accuratezza.

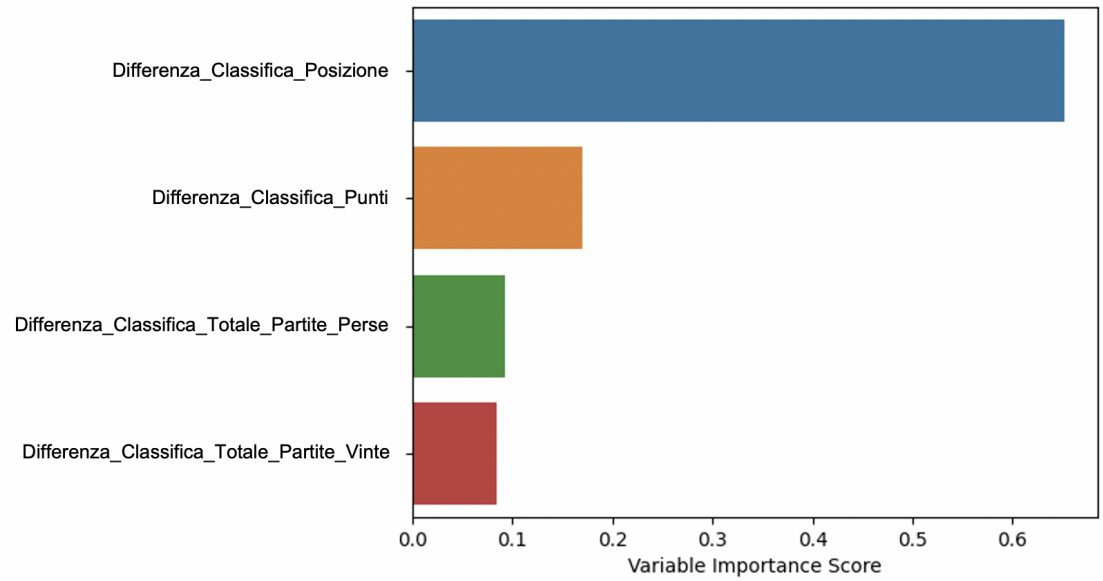


Figura 3.25: *Ranking* delle variabili per il Random Forest Classifier sul Dataset 4

Decision Tree Classifier Il Decision Tree Classifier ha raggiunto un'accuratezza solamente del 36.06%; tuttavia confermando però la rilevanza della feature riguardante la posizione in classifica, ritenendola unico valore discriminante per l'albero decisionale.

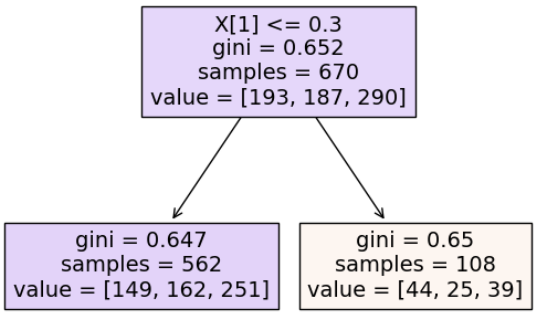


Figura 3.26: Albero decisionale generato dal Decision Tree Classifier sul Dataset 4

```
|--- feature_1 <= 0.30
|   |--- class: 2
|--- feature_1 >  0.30
|   |--- class: 0
```

Figura 3.27: Output del Decision Tree Classifier per il Dataset 4

NN Feed-forward La rete neurale implementata, coerentemente con gli altri dataset, ha raggiunto il più alto valore, pari al 42.72% di accuratezza, sempre con 2 *hidden layer* e 1.000 epoche.

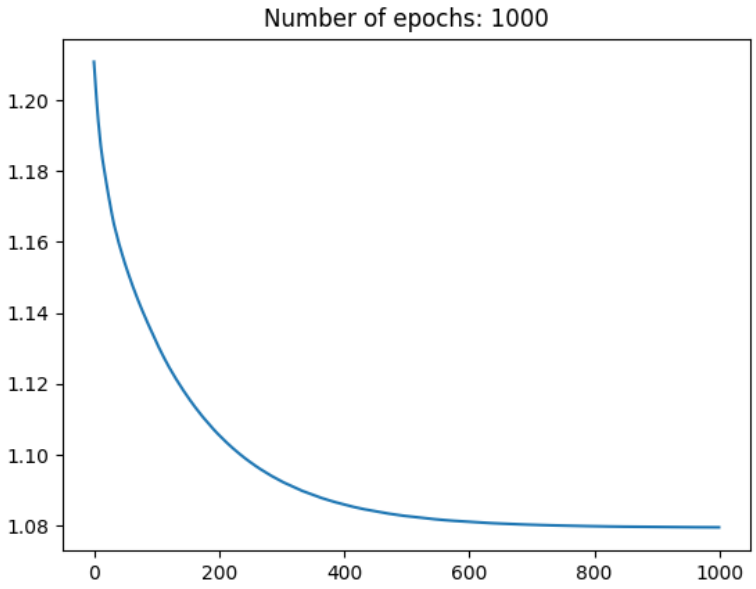


Figura 3.28: Grafico dell'andamento del *Value Loss* per la NN Feed-forward sul Dataset 4

Dataset 5

Il quinto ed ultimo dataset analizzato conteneva il numero di features più alto, ovvero: classifica (posizione, punti, partite vinte e perse, punti fatti e subiti, media

punti fatti e subiti) e medie storiche Voti Lega (ultime 999, 10, 5, 3, 2 e 1 gare). I risultati ottenuti sono stati i seguenti:

	Accuracy
Regressione Logistica	0.4030
Support Vector Machine	0.41
Random Forest Classifier	0.3696
Decision Tree Classifier	0.3757
Gaussian NB	0.3151
Gradient Boosting Classifier	0.3666
K-Nearest Neighbors Classifier	0.39
NN Feed-forward	**0.4242**

Tabella 3.10: Accuracy ottenute per gli algoritmi supervisionati con il Dataset 5

Anche per questo dataset il miglior algoritmo è stata la rete neurale Feed-forward, col 42.42% di accuratezza. Tra gli algoritmi supervisionati, invece, la Support Vector Machine è risultata essere la migliore a poca distanza dalla NN Feed-forward, col 41% di accuracy.

Random Forest Classifier Il ranking delle variabili prodotto dal Random Forest Classifier è particolarmente interessante per due aspetti. Secondo i risultati, infatti, la metrica "Differenza_Classifica_Totale_Media_Fatti" è al primo posto, seguita da "Differenza_Media_Voti_Lega_Giocatori_10_Gare",
"Differenza_Classifica_Totale_Punti_Subiti",
"Differenza_Classifica_Totale_Punti_Fatti" e
"Differenza_Media_Voti_Lega_Giocatori_3_Gare". Da questo si deduce che, per quanto riguarda le medie dei Voti Lega, sembrano essere più rilevanti le ultime 10 e 3 gare, rispetto alle metriche più di breve periodo come le ultime 1 e 2, e di lungo periodo come le ultime 999 gare. Ciò evidenzia come, ad esempio per questa istanza del problema, un giusto equilibrio sembra essere raggiunto su un numero di gare passate né troppo breve, né troppo lungo, a significare come le

performance di medio termine possano essere indicative della difficoltà di una gara futura. Per quanto riguarda le features sulle statistiche, invece, particolarmente rilevanti sono la media dei punti fatti e il totale dei punti subiti fino al momento in cui si è disputata la gara.

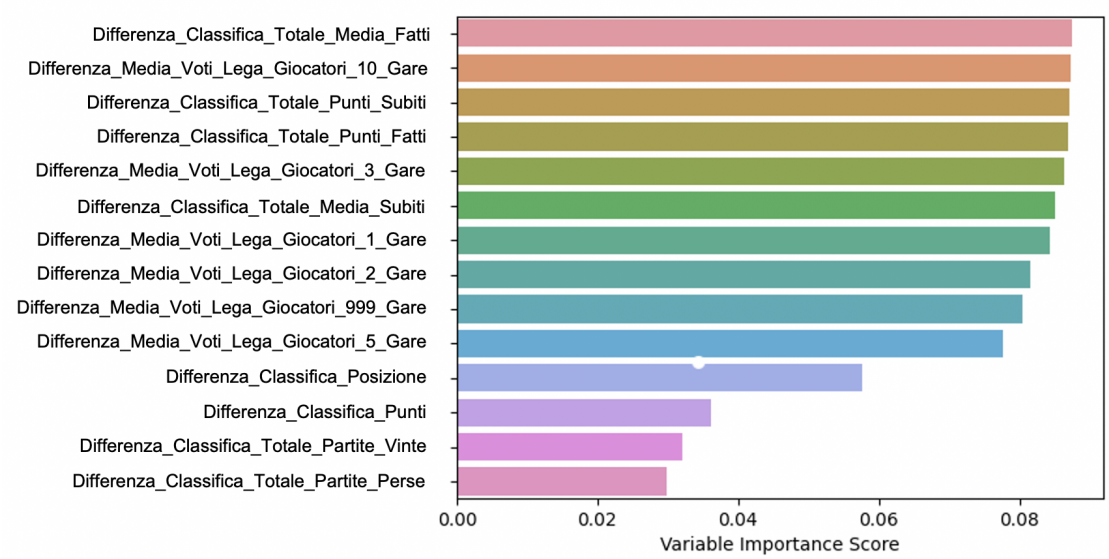


Figura 3.29: *Ranking* delle variabili per il Random Forest Classifier sul Dataset 5

Decision Tree Classifier Il Decision Tree Classifier ha raggiunto, per questo ultimo dataset, un'accuratezza del 37.57%; producendo il seguente albero decisionale:

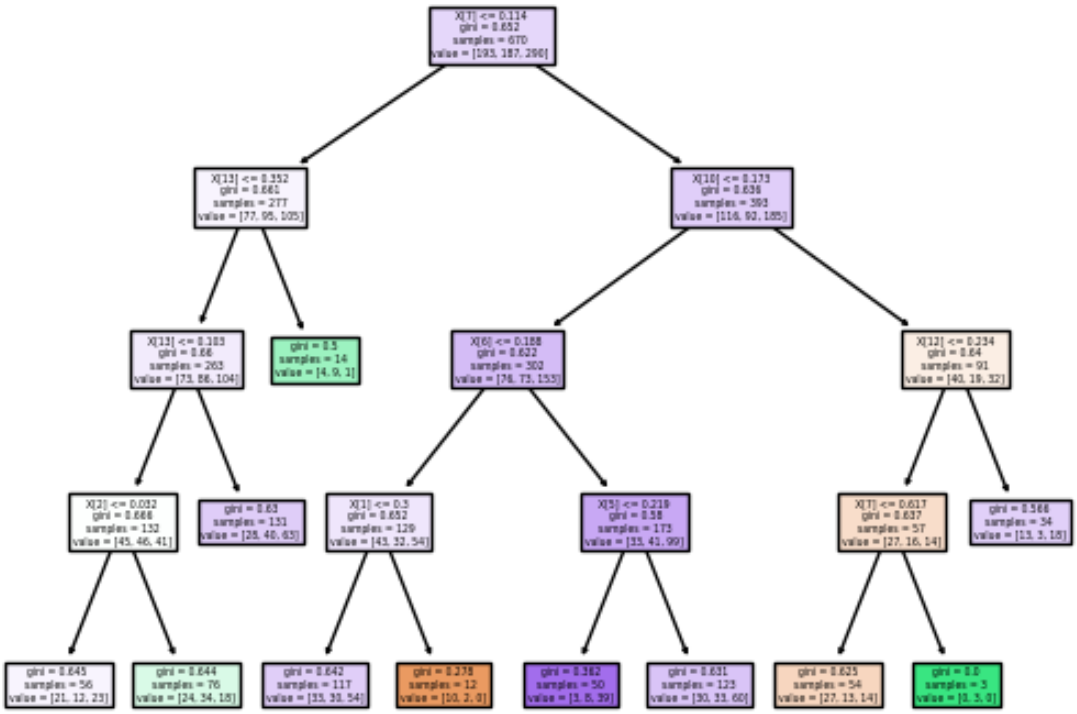


Figura 3.30: Albero decisionale generato dal Decision Tree Classifier sul Dataset 5

```
|--- feature_7 <= 0.11
|   |--- feature_13 <= 0.35
|   |   |--- feature_13 <= 0.10
|   |   |   |--- feature_2 <= 0.03
|   |   |   |   |--- class: 2
|   |   |   |--- feature_2 >  0.03
|   |   |   |   |--- class: 1
|   |   |--- feature_13 >  0.10
|   |   |   |--- class: 2
|   |--- feature_13 >  0.35
|   |   |--- class: 1
```

```
|--- feature_7 >  0.11
|   |--- feature_10 <= 0.17
|   |   |--- feature_6 <= 0.19
|   |   |   |--- feature_1 <= 0.30
|   |   |   |   |--- class: 2
|   |   |   |--- feature_1 >  0.30
|   |   |   |   |--- class: 0
|   |   |--- feature_6 >  0.19
|   |   |   |--- feature_5 <= 0.22
|   |   |   |   |--- class: 2
|   |   |   |--- feature_5 >  0.22
|   |   |   |   |--- class: 2
|   |--- feature_10 >  0.17
|   |   |--- feature_12 <= 0.23
|   |   |   |--- feature_7 <= 0.62
|   |   |   |   |--- class: 0
|   |   |   |--- feature_7 >  0.62
|   |   |   |   |--- class: 1
|   |   |--- feature_12 >  0.23
|   |   |   |--- class: 2
```

Figura 3.31: Output del Decision Tree Classifier per il Dataset 5

NN Feed-forward La rete neurale Feed-forward è riuscita, infine, a raggiungere un'accuratezza del 42.42%. Interessante è notare come il grafico del *Loss Value* sia leggermente diverso rispetto ai dataset precedenti; in questo caso, infatti, la rete riesce già dopo pochissime epoche a ridurre drasticamente il *Loss Value*, fino a farlo comunque poi raggiungere il valore minimo di 1.07.

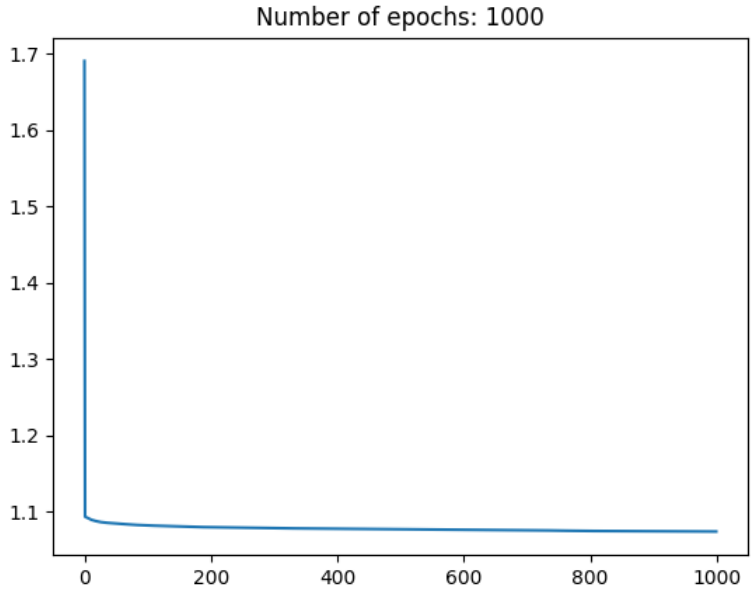


Figura 3.32: Grafico dell'andamento del *Value Loss* per la NN Feed-forward sul Dataset 5

Progetto "TLGProb"

L'accuratezza media ottenuta dal progetto "TLGProb" sulla predizione della difficoltà delle gare è risultata essere pari al 35.86%. In particolare, si mostra il grafico che rappresenta l'andamento dell'accuracy media nel corso delle giornate della stagione sportiva 2018/2019.

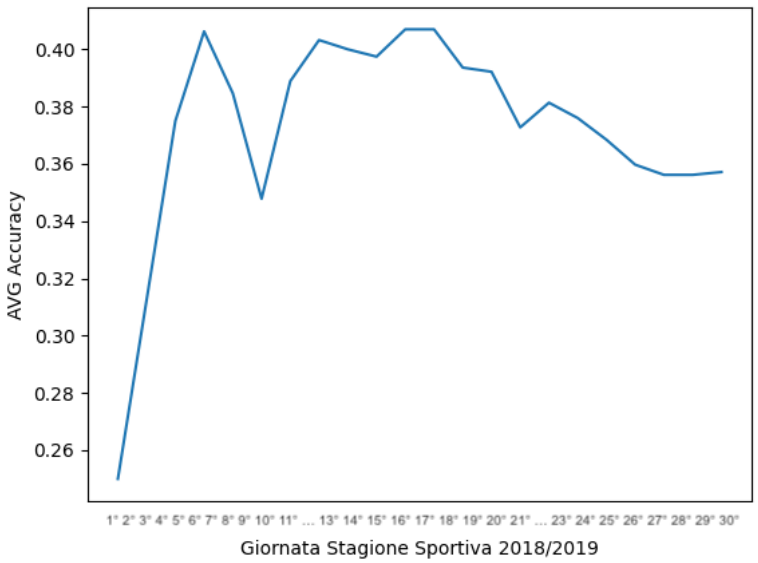


Figura 3.33: AVG accuracy giornata per giornata della stagione sportiva 2018/2019 per "TLGProb"

L'andamento, come evidenzia il grafico, seppur raggiunge una percentuale esigua, denota una crescita a partire dalle primissime giornate (1–3), dove l'accuratezza è davvero bassa, attorno al 25%. Col passare delle giornate poi, pur se con un andamento non lineare, l'accuratezza tende asintoticamente al 36% circa.

Training e *test* iterativi con *dataset* pesati dinamicamente

L'ultimo test effettuato ha riguardato il training e test iterativi con dataset pesati dinamicamente. Come descritto nella rispettiva sezione dedicata, il dataset fornito al training è composto da feature pesate in base alla differenza assoluta in giorni tra le gare passate e quella futura da predire. Di seguito viene mostrato il grafico risultante l'elaborazione dell'algoritmo, il quale descrive l'accuratezza media nel corso delle giornate della stagione sportiva 2018/2019.

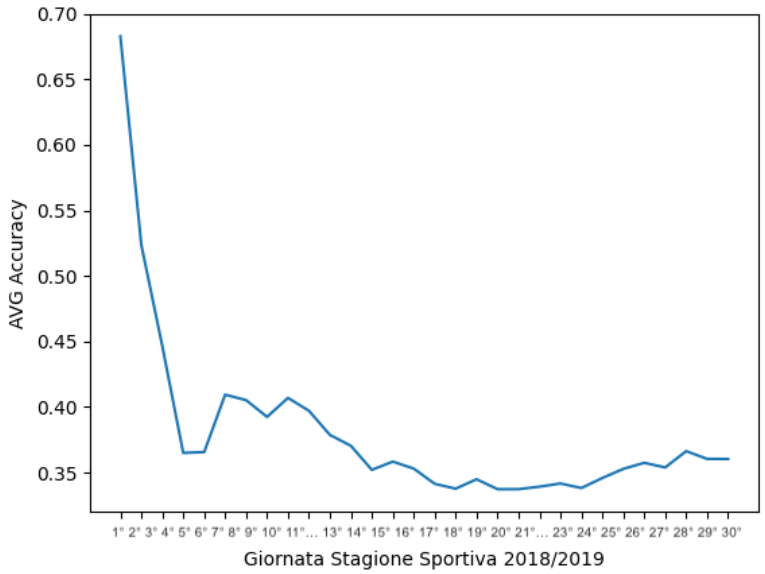


Figura 3.34: AVG accuracy giornata per giornata della stagione sportiva 2018/2019 per il *training* e *test* iterativi con *dataset* pesati dinamicamente

Come si evince dal grafico, per le primissime giornate di campionato l'algoritmo si attesta su un'accuratezza del 67%, proseguendo poi per raggiungere dei minimi ad inizio della seconda metà della stagione anche del 33%, ed infine attestandosi alla 30° ed ultima giornata al 36.03% di accuracy.

Considerazioni

Le accuratezze medie di tutti gli algoritmi e sistemi implementati sono mostrate nella seguente tabella riassuntiva.

	Dataset 1	*Dataset 2*	*Dataset 3*	*Dataset 4*	*Dataset 5*
Regressione Logistica	0.3969	0.4	0.4	0.4	0.4030
Support Vector Machine	0.41	0.41	0.41	0.41	0.41
Random Forest Classifier	0.3606	0.3484	0.3606	0.3666	0.3696
Decision Tree Classifier	0.3878	0.3787	0.3878	0.3787	0.3757
Gaussian NB	0.3545	0.4	0.3696	0.3484	0.3151
Gradient Boosting Classifier	0.3545	0.3606	0.3333	0.3606	0.3666
K-Nearest Neighbors Classifier	0.34	0.35	0.33	0.32	0.39
NN Feed-forward	**0.4212**	**0.4212**	**0.4242**	**0.4272**	**0.4242**
TLGProb	0.3586				
Train e Test iterativi con pesi dinamici	0.3603				

Tabella 3.11: Riassunto accuracy ottenute per tutti gli algoritmi implementati

Alla luce dei risultati ottenuti è necessario effettuare alcune considerazioni. Per interpretare al meglio i dati, è bene individuare quale sia la base di partenza minima (*baseline*) al di sopra della quale considerare i risultati come "accettabili". Questa soglia è rappresentata dal 33.33%, ovvero una scelta casuale tra le tre classi di difficoltà di una gara: "Facile", "Media" e "Difficile". Se un algoritmo producesse una percentuale di accuratezza inferiore a tale valore, significherebbe che una scelta casuale produrrebbe un risultato migliore. L'analisi dei dati e gli algoritmi implementati hanno permesso di raggiungere una percentuale superiore, seppur non di molto, al 33.33%; più precisamente del 42.72%, per la miglior combinazione di algoritmo e dataset. Questi ultimi sono stati individuati nella rete neurale Feed-forward e il dataset 4, cioè quello contenente le metriche relative alla classifica (posizione, punti, totale partite vinte e perse).

La causa principale dell'esiguo risultato ottenuto può essere attribuita a diversi fattori. Il primo potrebbe essere certamente quello dell'individuazione delle features più rilevanti per il task. Quest'ultimo potrebbe non essere agevolato dalla tipologia e eterogeneità dei dati (ad esempio la loro varianza nel corso della stagio-

ne sportiva). Tuttavia, il progetto "TLGProb" ha fornito interessanti spunti che sono stati successivamente implementati ed analizzati, come ad esempio il training e test iterativi con dataset pesati dinamicamente. Seppur questa strada non si sia rivelata "vincente", porta a pensare che potrebbe essere di particolare interesse svilupparla ed approfondirla in futuro. Per ultimo, è importante notare come i risultati ottenuti siano specifici e mirati alla stagione sportiva 2018/2019 e potrebbero variare (anche sensibilmente) per altre stagioni sportive. Tutti questi aspetti saranno analizzati più nel dettaglio nell'ultimo capitolo "Sviluppi futuri".

Capitolo 4

Assegnamento automatico delle designazioni arbitrali

La seconda parte dell'elaborato descrive la componente dell'assegnamento automatico delle designazioni arbitrali. Nel corso di questo capitolo verrà illustrato il "Referee Assignment Problem" e definito nello specifico il task. Proseguendo, verranno elencati e commentati i vincoli (*hard* e *soft*) individuati, concludendo infine con l'algoritmo costruito e i risultati conseguiti dall'esperimento svolto per la Serie A italiana.

4.1 Il "Referee Assignment Problem"

Il "Referee Assignment Problem" (RAP), tradotto come il "problema dell'assegnazione degli arbitri", viene affrontato nella gestione dello sport ogni volta che viene programmato un torneo oppure un campionato, poiché tutti gli sport hanno almeno un arbitro che si assicura che il gioco venga svolto secondo le regole. Tale task consiste fondamentalmente nel trovare un'assegnazione di arbitri agli slot arbitrali che vengono generati per le partite che devono essere disputate. Il calendario deve essere già programmato prima di affrontare questo problema, poiché esso condiziona la scelta delle designazioni; ad esempio, un arbitro non può trovarsi a dirigere contemporaneamente due gare, quindi è necessario sapere in anticipo

quando queste si disputeranno.

Il RAP opera avendo a disposizione una serie di arbitri che possiedono qualità e competenze diverse tra di loro, ed una serie di vincoli devono essere rispettati affinché la designazione di un arbitro sia coerente sia con gli standard della gara che con le disponibilità personali di ognuno. Dato questo elenco di vincoli, alcuni da rispettare obbligatoriamente (denominati come "*hard*") e altri espressi come preferenze (denominati "*soft*"), il RAP cerca una combinazione di designazioni accettabile che rispetti tutti i vincoli e soddisfi quante più preferenze possibili.

Ogni sport, nazione e campionato hanno le proprie regole, e questo significa che ogni caso deve essere preso a sé stante come un problema diverso, poiché molte caratteristiche possono variare, come il numero di slot arbitrali che vengono generati per ogni partita, oppure la frequenza delle stesse. Ad esempio, una partita di football americano necessita di 7 arbitri, una partita di pallacanestro può richiederne 3, il calcio 5 (eventuale riserva inclusa), e così via. Ciò significa che ogni campionato ha la propria serie di vincoli e preferenze.

4.2 Definizione del problema

Nella Serie A di pallacanestro italiana ci sono n squadre (pari a 16), e la stagione è suddivisa in $2n - 2$ giornate (pari a 30) ed una squadra gioca in totale $2n - 2$ gare. Gli incontri possono essere rappresentati nella forma "Squadra A vs. Squadra B", dove la squadra A gioca in casa mentre la B è ospite. Alla fine della regular season ogni squadra deve aver affrontato tutte le altre esattamente due volte, una giocando in casa ed una come ospite.

Dato che il calendario (elenco delle gare) è già noto a priori del problema, per ogni giornata si conosceranno le squadre che si affronteranno, sarà necessario quindi designare gli arbitri per tali gare. Nella Serie A italiana ci sono 3 arbitri a dirigere ogni gara, con tre diversi ruoli: "1° Arbitro", "2° Arbitro" e "3° Arbitro". Ai fini

del problema non vi è alcuna rilevanza tra questi ruoli, tuttavia, in ottica di futuri miglioramenti ed espansioni, è bene sapere che il ruolo del 1° arbitro generalmente è ricoperto dai più esperti.

Una delle caratteristiche fondamentali di una designazione, è che una partita deve essere sempre arbitrata da direttori di gara che siano effettivamente in grado di gestirla al meglio; per questo bisogna tenere conto del livello di esperienza di ognuno di loro. Al fine di designare al meglio, è altresì necessario prendere in considerazione il fatto che è meglio non sovraccaricare eccessivamente gli arbitri con troppe gare (ad esempio due a settimana), ma neanche tenerli eccessivamente fermi (es. una sola gara ogni due mesi). In aggiunta, gli arbitri hanno la possibilità, nel corso di tutta la stagione sportiva, di comunicare di volta in volta eventuali loro indisponibilità e di non essere designati in tali giorni/periodi. Le indisponibilità degli arbitri possono, inoltre, riguardare anche delle società, e sarà bene non designarli per queste ultime; tali indisponibilità vengono anche chiamate "incompatibilità". Infine, per evitare pregiudizi da parte del pubblico o di altre società, gli arbitri non dovrebbero essere assegnati troppo frequentemente ad una squadra in particolare, ma cercare di ottenere una distribuzione la più equa ed omogenea possibile.

4.2.1 Vincoli

Per definire in modo più preciso il problema è necessario definire alcuni vincoli che l'algoritmo dovrà cercare di rispettare, al fine di generare designazioni coerenti col dominio del problema e il modello realizzato. Alcuni di questi vincoli, come descritto nel primo capitolo "Stato dell'arte" dell'elaborato, non possono essere in alcun modo violati (vincoli *hard*), altri invece rappresentano delle preferenze (vincoli *soft*) e possono non essere rispettati, pur di fornire una soluzione.

Vincoli *hard*

Di seguito vengono elencati i vincoli che devono essere obbligatoriamente rispettati dall'algoritmo, qualunque esso sia.

1. Ogni gara deve essere arbitrata da esattamente 3 arbitri;
2. La somma dell'esperienza dei singoli arbitri deve essere uguale o superiore a quella richiesta dalla gara;
3. Un arbitro non può essere designato in un giorno o periodo in cui è indicato come indisponibile;
4. Un arbitro non può essere designato per una gara dove gioca una squadra per la quale è incompatibile.

Vincoli *soft*

Al fine di ottenere designazioni della più alta qualità possibile, vengono inoltre considerati i seguenti vincoli *soft*.

1. Il numero totale di gare assegnate ad un arbitro dovrebbe essere quanto più possibile uguale alla media;
2. Il numero di volte che un arbitro è assegnato alla stessa squadra non dovrebbe essere troppo alto rispetto al valore medio.

4.2.2 Livello di esperienza degli arbitri

Come descritto nella precedente sezione del corrente capitolo, l'algoritmo, a prescindere dalla tipologia di implementazione, dovrà necessariamente rispettare una serie di vincoli *hard* per fornire una soluzione al problema. Tra di essi, il più delicato riguarda l'esperienza richiesta da parte degli arbitri per dirigere una gara. Infatti, proprio su questo vincolo si posa l'intera prima parte di questo elaborato: determinare il livello di difficoltà di una gara per poi scegliere adeguatamente gli arbitri. Fino a questo punto, la difficoltà di una gara è stata definita come 0 (Facile), 1 (Media) oppure 2 (Difficile). Tuttavia l'algoritmo, per poter risolvere il problema, ha necessariamente bisogno di rappresentarla in modo numerico, ma non solo. Per come è stato formulato il vincolo nello specifico, la somma dell'esperienza dei tre arbitri dovrà essere maggiore o uguale a quella richiesta dalla gara.

Gli arbitri, nella quasi totalità dei campionati (Serie A inclusa), vengono suddivisi in fasce che cercano di rispecchiare il loro reale livello d'esperienza. Per questo motivo si è proceduto quindi a tentare di replicare questa suddivisione degli arbitri, più specificatamente in tre distinte fasce: "Top" (più alta fascia), "Advanced" (esperti) e "Normal". Chiaramente non è possibile la reale attribuzione di ognuna di queste fasce per ogni arbitro di Serie A, poiché è un'informazione strettamente confidenziale e riservata. Tuttavia, è possibile stimare, almeno in termini percentuali, la distribuzione di queste fasce all'interno dei 35 arbitri.

Per risolvere numericamente il problema ad ogni fascia è stato attribuito un punteggio. Il livello di esperienza di ogni arbitro è determinato dal punteggio della fascia di appartenenza. Per ogni grado di difficoltà di una gara è stato assegnato un ulteriore punteggio. La somma del punteggio esperienza (abbreviato come "Exp") dei tre arbitri dovrà essere uguale o superiore al punteggio di difficoltà della gara, per far sì che essi la dirigano. Di seguito viene riportata una tabella riassuntiva dove sono presenti tutte le combinazioni di fasce arbitrali e la possibilità o meno di dirigere una gara in base alla sua difficoltà. Si ricorda che l'ordine dei tre arbitri non è rilevante ai fini del problema.

Arbitro 1	Arbitro 2	Arbitro 3	Facile	Media	Difficile
Top	Top	Top	Accepted	Accepted	Accepted
Top	Top	Advanced	Accepted	Accepted	Accepted
Top	Top	Normal	Accepted	Accepted	Accepted
Top	Advanced	Advanced	Accepted	Accepted	Accepted
Top	Advanced	Normal	Accepted	Accepted	Accepted
Top	Normal	Normal	Accepted	Accepted	Refused
Advanced	Advanced	Advanced	Accepted	Refused	Refused
Advanced	Advanced	Normal	Accepted	Refused	Refused
Advanced	Normal	Normal	Accepted	Refused	Refused
Normal	Normal	Normal	Refused	Refused	Refused

Tabella 4.1: Combinazioni fasce terna arbitrale con relative difficoltà gara

Per ogni combinazione di fasce arbitrali è riportato (sotto forma di "*Accepted*" o "*Refused*"), nella parte destra della tabella, quale livello di difficoltà di una gara potrà arbitrare la terna. Ad esempio, tre arbitri "Top" (quindi teoricamente tutti molto esperti), potranno arbitrare qualsiasi tipo di gara, sia essa facile, media o difficile. Invece, tre arbitri di cui uno "Top" e due "Normal" potranno dirigere solamente gare facili o medie, ma non difficili. Tali vincoli sono stati scelti in seguito a consultazioni in merito con esperti del dominio.

Fasce arbitrali e difficoltà delle gare

Dal calcolo combinatorio (statistica) si è iniziato con l'estrarre il numero preciso di combinazioni possibili, pari a 10. Questo poiché i singoli elementi (fasce arbitrali) possono essere ripetuti e formare gruppi da tre.

$$C_{(n,k)} = \frac{(n+k-1)!}{k!(n-1)!}$$

Ad ogni fascia arbitrale è stato poi attribuito uno specifico punteggio in termini di esperienza:

Fascia arbitro	Punteggio Exp.
Normal	50
Advanced	100
Top	250

Tabella 4.2: Punteggi attribuiti alle fasce arbitrali

Successivamente, si è proceduto in tal senso anche per quanto riguarda i livelli di difficoltà di una gara:

Difficoltà gara	Punteggio
Facile	151
Media	301
Difficile	400

Tabella 4.3: Punteggi attribuiti alle difficoltà delle gare

Infine, si è rappresentato il livello di esperienza complessivo della terna arbitrale sotto forma di somma dei singoli punteggi. Se tale somma è superiore o uguale al punteggio di difficoltà della gara, allora la terna potrà dirigerla, altrimenti no. Una visione più dettagliata delle combinazioni di punteggi è riportata nella seguente tabella.

Arbitro 1	Arbitro 2	Arbitro 3	Exp Arbitro 1	Exp Arbitro 2	Exp Arbitro 3	Somma Exp	Facile	Media	Difficile
Top	Top	Top	250	250	250	750	Accepted	Accepted	Accepted
Top	Top	Advanced	250	250	100	600	Accepted	Accepted	Accepted
Top	Top	Normal	250	250	50	550	Accepted	Accepted	Accepted
Top	Advanced	Advanced	250	100	100	450	Accepted	Accepted	Accepted
Top	Advanced	Normal	250	100	50	400	Accepted	Accepted	Accepted
Top	Normal	Normal	250	50	50	350	Accepted	Accepted	Refused
Advanced	Advanced	Advanced	100	100	100	300	Accepted	Refused	Refused
Advanced	Advanced	Normal	100	100	50	250	Accepted	Refused	Refused
Advanced	Normal	Normal	100	50	50	200	Accepted	Refused	Refused
Normal	Normal	Normal	50	50	50	150	Refused	Refused	Refused

Tabella 4.4: Combinazioni fasce terna arbitrale con relative difficoltà gara

Come è possibile notare, ad esempio per il primo caso, la somma totale dell'esperienza della terna è pari a 750 punti e potranno quindi arbitrare qualsiasi gara poiché la somma è sempre superiore ai punteggi di difficoltà di facile (151), media (301) e difficile (400).

4.2.3 Modellazione

Una volta definiti i vincoli, si è passati alla modellazione del problema di ottimizzazione dal punto di vista matematico. A tal fine, è necessario definire i parametri del modello:

- N: il numero totale di squadre;
- M: il numero totale degli arbitri;
- $s_1 ... s_N$: le squadre partecipanti;
- $a_1 ... a_M$: gli arbitri da designare;

- $g_1 ... g_{2N-2}$: le giornate di campionato;
- $p(s_i, s_j, g_k)$: costante pari a 1 se la gara tra le squadre s_i e s_j è giocata nella giornata g_k di campionato.

Si dichiarano inoltre alcune variabili e costanti di supporto alla modellazione:

- $DGD(a_i, a_j)$ variabile booleana pari a 1 se l'arbitro a_i è designato per più gare rispetto all'arbitro a_j, 0 altrimenti;
- $DAG(a_i, a_j, s_k)$ variabile booleana pari a 1 se l'arbitro a_i è designato alle gare dove gioca s_k un numero di volte maggiore rispetto all'arbitro a_j, 0 altrimenti;
- $A(s_i, s_j, g_k, a_l)$ variabile booleana pari 1 se l'arbitro a_l è designato per una gara tra le squadre s_i (casa) e s_j (ospite) nella giornata g_k, 0 altrimenti;
- $ea(a_i)$: costante dal valore $[50, 100, 250]$ a seconda del livello di esperienza dell'arbitro a_i;
- $eg(s_i, s_j)$: costante dal valore $[151, 301, 400]$ a seconda del livello di esperienza richiesto dalla gara disputata tra le squadre s_i ed s_j;
- $is(a_i, s_j)$: costante pari a 1 se l'arbitro a_i è incompatibile con la squadra s_j;
- $ig(a_i, g_k)$: costante pari a 1 se l'arbitro a_i è indisponibile per la giornata di campionato g_k.

I vincoli *soft* sono rispettati grazie alla funzione obiettivo (4.1), dove le variabili DGD e DAG assumono valore 1 ogni volta che un vincolo *soft* è stato violato. Più nello specifico, per ogni variabile DGD uguale a 1, esiste una coppia di arbitri nella quale il primo di essi ha più designazioni del secondo. Per ogni variabile DAG pari a 1, invece, si avrà un trio composto da due arbitri e una squadra per il quale il primo arbitro ha più gare designate per la stessa squadra rispetto al secondo. Dato che la qualità più alta della soluzione proposta è rappresentata dal minor

numero di vincoli *soft* violati (a parità di vincoli *hard* rispettati), il problema si configura come minimizzazione della funzione obiettivo:

$$min(\sum_{i=1}^{M}\sum_{j=1}^{M} DGD(a_i, a_j) + \sum_{i=1}^{M}\sum_{j=1}^{M}\sum_{k=1}^{N} DAG(a_i, a_j, g_k)) \tag{4.1}$$

Secondo quanto imposto dal primo vincolo *hard*, ogni gara deve essere diretta da esattamente 3 arbitri; questo viene garantito dall'equazione (4.2):

$$\sum_{l=1}^{M} A(s_i, s_j, g_k, a_l) = 3p(s_i, s_j, g_k) \tag{4.2}$$

Il secondo vincolo *hard*, per il quale la somma dell'esperienza dei singoli arbitri designati deve essere uguale o maggiore di quella richiesta dalla gara per essere diretta, è rappresentata dalla relazione (4.3).

$$\sum_{i=1}^{M} A(s_i, s_j, g_k, a_l) \cdot ea(a_l) \geq eg(s_i, s_j) \tag{4.3}$$

Il terzo vincolo *hard* circa l'incompatibilità tra un arbitro ed una squadra è garantito dalle equazioni sottostanti (4.4) e (4.5), dove la prima rappresenta l'incompatibilità con la squadra di casa e la seconda con la squadra ospite.

$$A(s_i, s_j, g_k, a_l) \leq 1 - is(a_l, s_i) \tag{4.4}$$

$$A(s_i, s_j, g_k, a_l) \leq 1 - is(a_l, s_j) \tag{4.5}$$

Il quarto ed ultimo vincolo *hard* riguarda l'indisponibilità di un arbitro per le giornate di campionato nelle quali non può arbitrare, ed è rispettato grazie all'equazione (4.6).

$$A(s_i, s_j, g_k, a_l) \leq 1 - ig(a_l, g_k) \tag{4.6}$$

4.3 Algoritmo Greedy

Come anticipa già il nome del sotto capitolo, l'algoritmo implementato è stato di tipo "*Greedy*". Il nome deriva dal comportamento dove, ad ogni iterazione, viene scelta la soluzione che migliora (anche se di poco) quella corrente. Questo tipo di algoritmi si basano sull'idea che per trovare una soluzione globalmente ottima, è necessario scegliere ripetutamente soluzioni ottime localmente. Di seguito viene mostrata la differenza tra punto di ottimo locale e globale.

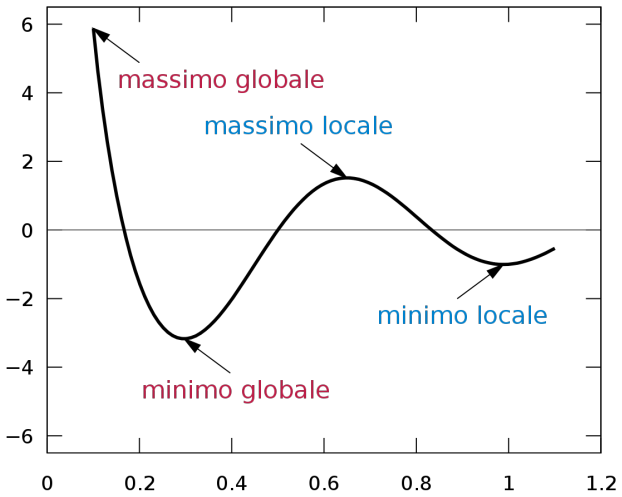


Figura 4.1: Esempio grafico di massimo globale e locale di una funzione

Come mostrato dalla figura, un punto di massimo globale corrisponde al punto in cui la funzione assume il più alto valore possibile. In un punto di massimo locale, invece, la funzione assume un valore massimo ma solamente in un intorno ben specifico di essa.

Gli algoritmi *greedy*, quindi, compiono ad ogni passo la scelta migliore nell'immediato, piuttosto che adottare una strategia a lungo termine. Tuttavia questa strategia potrebbe non condurre sempre ad una soluzione ottima globale. Ad ogni passo la scelta migliorativa deve comunque necessariamente rispettare i vincoli *hard* (al fine di produrre una soluzione ammissibile), poiché devono essere sempre

rispettati (invariante). Per ognuno dei vincoli *soft* non rispettati, invece, l'algoritmo calcola il costo della soluzione come la somma di tutti i punteggi dei vincoli *soft* violati. La funzione euristica dell'algoritmo scelto, quindi, punta a minimizzare tale costo, al fine di proporre una soluzione di più alta qualità possibile. La tecnica *greedy* è stata scelta poiché molto spesso nei problemi di ottimizzazione di questo tipo, la loro efficacia è determinata sia dalla qualità accettabile della soluzione prodotta, ma anche dai tempi di calcolo necessari per produrla, che è un aspetto molto importante per l'ambito di applicazione.

L'algoritmo implementato è stato creato con C++ e di seguito vengono mostrati i passaggi più rilevanti. Quest'ultimo inizialmente assegna un arbitro ad ogni gara fino ad arrivare ad un massimo di tre. Ad ogni scelta poi, viene calcolato il costo (*delta*) della stessa rispetto alla soluzione attuale. Di seguito viene mostrato il relativo codice.

```
71      for (int arbitro : arbitri)
72      {
73        designazioniOld = out.ArbitriDesignati(gare);
74        designazioniNew = out.ArbitriDesignati(gare);
75        designazioniNew.insert(arbitro);
76
77        if (IndisponibilitaArbitriDeltaCost(in, out, gare, designazioniOld, designazioniNew) > 0)
78          continue;
79
80        if (EsperienzaRichiestaDeltaCost(in, out, gare, designazioniOld, designazioniNew) > 0)
81          continue;
82
83        if (IncompatibilitaSquadraDeltaCost(in, out, gare, designazioniOld, designazioniNew) > 0)
84          continue;
85
86        deltaCost = 0;
87        deltaCost += in.DistribuzioneGareWeight() * DistribuzioneGareDeltaCost(in, out, gare, designazioniOld, designazioniN
88        deltaCost += in.FrequenzaDesignazioniWeight() * FrequenzaDesignazioniDeltaCost(in, out, gare, designazioniOld, desig
89
90        if (deltaCost < minDeltaCost)
91        {
92          minDeltaCost = deltaCost;
93          arbitroDaDesignare = arbitro;
94          equalMinDeltaNum = 1;
95        }
96        else if (deltaCost == minDeltaCost)
97        {
98          equalMinDeltaNum++;
99
100         if (Random(1, equalMinDeltaNum) == 1)
101           arbitroDaDesignare = arbitro;
102       }
103     }
104
105     if (arbitroDaDesignare != 0)
106     {
107       out.Designa(arbitroDaDesignare, gara);
108       designato = true;
109     }
110   }
```

Codice 4.1: Codice C++ dell'algoritmo *greedy* (*main*)

Per semplicità, il peso di tutti i vincoli *soft* è uguale ad 1. Proseguendo, si analizza nel dettaglio una specifica funzione di costo, in particolare quella dedicata al computo del *delta cost* per quanto riguarda le incompatibilità degli arbitri con le squadre.

```
720  int IncompatibilitaSquadraDeltaCost(const &in, const &out, int gara, const set<int> &designazioniOld, const set<int> &designa:
721  {
722    int cost = 0;
723    set<int> designazioniOld, designazioniNew;
724    set<int>::iterator it;
725    set<int>::iterator obeg = designazioniOld.begin();
726    set<int>::iterator oend = designazioniOld.end();
727    set<int>::iterator nbeg = designazioniNew.begin();
728    set<int>::iterator nend = designazioniNew.end();
729    const Squadra &squadraCasa = in.GetSquadra(in.GetGara(gara).SquadraCasa());
730    const Squadra &squadraOspite = in.GetSquadra(in.GetGara(gara).SquadraOspite());
731
732    set_difference(obeg, oend, nbeg, nend, inserter(designazioniOld, designazioniOld.end()));
733    set_difference(nbeg, nend, obeg, oend, inserter(designazioniNew, designazioniNew.end()));
734
735    for (it = designazioniOld.begin(); it != designazioniOld.end(); ++it)
736    {
737      const Arbitro &arbitro = in.GetArbitro(*it);
738
739      if (arbitro.IncompatibleTeam(squadraCasa.Id()))
740        cost--;
741
742      if (arbitro.IncompatibleTeam(squadraOspite.Id()))
743        cost--;
744    }
745
746    for (it = designazioniNew.begin(); it != designazioniNew.end(); ++it)
747    {
748      const Arbitro &arbitro = in.GetArbitro(*it);
749
750      if (arbitro.IncompatibleTeam(squadraCasa.Id()))
751        cost++;
752
753      if (arbitro.IncompatibleTeam(squadraOspite.Id()))
754        cost++;
755    }
756
757    return cost;
758  }
```

Codice 4.2: Codice C++ dell'algoritmo *greedy* (funzione "*IncompatibilitaSquadraDeltaCost*")

Il codice sopra riportato esplicita il calcolo del costo differenziale (*delta*) per le incompatibilità degli arbitri con le squadre. Più nello specifico, vengono fornite alla funzione le due configurazioni (soluzioni): quella attuale e quella nuova (temporanea). Per entrambi i set di designazioni vengono controllate le incompatibilità registrate degli arbitri per la gara in questione. In particolare, è interessante notare come la variabile `cost` venga decrementata di uno per ogni squadra incompatibile designata per le attuali designazioni, e invece aumentata di uno per le designazioni temporanee. Al termine di questo calcolo, se `cost` sarà pari a zero significa che nessun vincolo è stato violato, se invece è superiore o minore a zero significa che è stato violato; più precisamente se minore di zero indica una minore appetibilità

della configurazione di designazioni attuale.

Nel seguente sotto capitolo dedicato all'esperimento realizzato vengono dettagliate anche le specifiche riguardanti l'input fornito all'algoritmo e il relativo output.

4.4 Esperimento per la Serie A italiana

Una volta creato l'algoritmo ed effettuato alcuni test per accertarne il corretto funzionamento, si è proceduto a creare un'istanza specifica dedicata alla Serie A di pallacanestro italiana per la stagione sportiva 2018/2019.

Input

L'input fornito all'algoritmo consiste in un file di testo contenente tutti i dati del problema, ovvero: arbitri (e relativi livelli di esperienza, indisponibilità e incompatibilità), campi di gioco, squadre e gare. Il file specifico (troncato per brevità) è mostrato di seguito:

```
ARBITRI % id, esperienza, squadre incompatibili, indisponibilità

A1, 100, [], []
A2, 100, [], [07/10/2018]
A3, 50, [], []
A4, 250, [Virtus Bologna], []
A5, 100, [], []
A6, 50, [], []
A7, 100, [], []
A8, 250, [], []
A9, 50, [], []
A10, 250, [Brindisi], []
A11, 50, [], []
A12, 100, [], [15/01/2019]
A13, 250, [], []
A14, 50, [], []
A15, 250, [Virtus Bologna], []
A16, 50, [], [26/03/2019]
A17, 50, [], []
A18, 100, [], []
A19, 100, [], []
A20, 50, [], [15/11/2018]
A21, 50, [], []
A22, 250, [Trieste], []
A23, 250, [], []
A24, 100, [], [05/04/2019]
A25, 250, [], []
A26, 50, [], []
A27, 50, [], []
A28, 250, [], [06/10/2018, 25/11/2018]
A29, 100, [Milano], []
A30, 250, [], []
A31, 100, [], [22/11/2018, 18/12/2018]
A32, 250, [], []
A33, 250, [Cremona], []
A34, 250, [], []
A35, 100, [], []
A36, 50, [], []
```

```
GARE % squadra casa, squadra ospite, data, ora, esperienza richiesta

Avellino, Cantù, 06/10/2018, 18:00, 151
Trentino, Cremona, 06/10/2018, 20:30, 400
Reggio Emilia, Sassari, 07/10/2018, 17:00, 301
Milano, Brindisi, 07/10/2018, 17:30, 301
Venezia, Torino, 07/10/2018, 18:15, 400
Pesaro, Pistoia, 07/10/2018, 18:30, 400
Varese, Brescia, 07/10/2018, 19:30, 400
Trieste, Virtus Bologna, 07/10/2018, 20:45, 400

Torino, Trieste, 13/10/2018, 20:30, 301
Sassari, Varese, 14/10/2018, 12:00, 301
Cremona, Avellino, 14/10/2018, 17:00, 400
Virtus Bologna, Milano, 14/10/2018, 17:30, 301
Brescia, Reggio Emilia, 14/10/2018, 18:00, 400
Pistoia, Venezia, 14/10/2018, 18:30, 151
Brindisi, Pesaro, 14/10/2018, 19:00, 151
Cantù, Trentino, 14/10/2018, 20:45, 301
...
Trentino, Milano, 05/05/2019, 20:45, 400
Cremona, Venezia, 05/05/2019, 20:45, 151
Trieste, Sassari, 05/05/2019, 20:45, 151
Cantù, Torino, 05/05/2019, 20:45, 301
Pesaro, Reggio Emilia, 05/05/2019, 20:45, 151
Varese, Pistoia, 05/05/2019, 20:45, 151
Avellino, Brindisi, 05/05/2019, 20:45, 400

Brindisi, Trentino, 12/05/2019, 20:45, 400
Venezia, Brescia, 12/05/2019, 20:45, 151
Pistoia, Avellino, 12/05/2019, 20:45, 151
Virtus Bologna, Varese, 12/05/2019, 20:45, 301
Sassari, Cantù, 12/05/2019, 20:45, 400
Reggio Emilia, Cremona, 12/05/2019, 20:45, 400
Torino, Pesaro, 12/05/2019, 20:45, 151
Milano, Trieste, 12/05/2019, 20:45, 400
```

Figura 4.2: File di testo di input per l'algoritmo *greedy*

La prima parte del file è composto dall'elenco degli arbitri, dove per ognuno è riportato: identificativo, esperienza, squadre incompatibili e indisponibilità. La seconda parte, invece, l'elenco gare con: squadra casa e ospite, data, ora e livello d'esperienza necessario. Come è possibile notare, nell'elenco degli arbitri accanto ad alcuni sono state inserite (a fine esemplificativo) alcune indisponibilità come giorni ed incompatibilità con squadre. La singola gara, invece, è individuata univocamente dalla coppia ordinata delle squadre sfidanti, poiché all'interno di una stagione regolare due squadre non si affronteranno mai nello stesso ordine (casa-ospite).

Output

Una volta terminato, l'algoritmo *greedy* fornirà un output contenente il medesimo elenco delle gare nell'input, ma con accanto ad ognuna riportate le designazioni proposte. Di seguito è riportato l'output per l'input sopra riportato (troncato per brevità).

```
Avellino, Cantù, 06/10/2018, 18:00, 151, A5 (100), A32 (250), A34 (250)
Trentino, Cremona, 06/10/2018, 20:30, 400, A2 (100), A15 (250), A26 (50)
Reggio Emilia, Sassari, 07/10/2018, 17:00, 301, A4 (250), A35 (100), A30 (250)
Milano, Brindisi, 07/10/2018, 17:30, 301, A3 (50), A11 (50), A8 (250)
Venezia, Torino, 07/10/2018, 18:15, 400, A7 (100), A21 (50), A13 (250)
Pesaro, Pistoia, 07/10/2018, 18:30, 400, A18 (100), A19 (100), A25 (250)
Varese, Brescia, 07/10/2018, 19:30, 400, A1 (100), A29 (100), A33 (250)
Trieste, Virtus Bologna, 07/10/2018, 20:45, 400, A24 (100), A16 (50), A28 (250)

Torino, Trieste, 13/10/2018, 20:30, 301, A3 (50), A19 (100), A25 (250)
Sassari, Varese, 14/10/2018, 12:00, 301, A32 (250), A16 (50), A24 (100)
Cremona, Avellino, 14/10/2018, 17:00, 400, A22 (250), A30 (250), A34 (250)
Virtus Bologna, Milano, 14/10/2018, 17:30, 301, A27 (50), A18 (100), A28 (250)
Brescia, Reggio Emilia, 14/10/2018, 18:00, 400, A15 (250), A29 (100), A33 (250)
Pistoia, Venezia, 14/10/2018, 18:30, 151, A4 (250), A5 (100), A7 (100)
Brindisi, Pesaro, 14/10/2018, 19:00, 151, A13 (250), A14 (50), A23 (250)
Cantù, Trentino, 14/10/2018, 20:45, 301, A6 (50), A9 (50), A8 (250)
```

...

```
Trentino, Milano, 05/05/2019, 20:45, 400, A4 (250), A9 (50), A34 (250)
Cremona, Venezia, 05/05/2019, 20:45, 151, A5 (100), A22 (250), A29 (100)
Trieste, Sassari, 05/05/2019, 20:45, 151, A15 (250), A20 (50), A26 (50)
Cantù, Torino, 05/05/2019, 20:45, 301, A18 (100), A28 (250), A33 (250)
Pesaro, Reggio Emilia, 05/05/2019, 20:45, 151, A19 (100), A21 (50), A25 (250)
Varese, Pistoia, 05/05/2019, 20:45, 151, A3 (50), A11 (50), A30 (250)
Avellino, Brindisi, 05/05/2019, 20:45, 400, A13 (250), A23 (250), A31 (100)

Brindisi, Trentino, 12/05/2019, 20:45, 400, A13 (250), A14 (50), A23 (250)
Venezia, Brescia, 12/05/2019, 20:45, 151, A9 (50), A18 (100), A26 (50)
Pistoia, Avellino, 12/05/2019, 20:45, 151, A4 (250), A30 (250), A32 (250)
Virtus Bologna, Varese, 12/05/2019, 20:45, 301, A19 (100), A22 (250), A28 (250)
Sassari, Cantù, 12/05/2019, 20:45, 400, A1 (100), A7 (100), A10 (250)
Reggio Emilia, Cremona, 12/05/2019, 20:45, 400, A5 (100), A15 (250), A34 (250)
Torino, Pesaro, 12/05/2019, 20:45, 151, A3 (50), A12 (100), A29 (100)
Milano, Trieste, 12/05/2019, 20:45, 400, A31 (100), A21 (50), A33 (250)
```

Figura 4.3: File di testo di output per l'algoritmo *greedy*

Per prima cosa possiamo notare come sia stato rispettato il primo vincolo *hard*: tutte le gare devono essere arbitrate da esattamente 3 arbitri. Analizzando nel dettaglio poi, si evince anche come sia stato rispettato il secondo vincolo *hard*, ovvero quello riguardante l'esperienza. Infatti, se prendiamo in analisi tutte le gare sopra riportate, la somma delle esperienze dei singoli arbitri (riportata tra parentesi) è sempre uguale o superiore rispetto a quella richiesta dalla specifica gara (riportata accanto all'orario). Per quanto riguarda il terzo vincolo *hard* (l'indisponibilità degli arbitri in determinati giorni), ad esempio l'arbitro A2 era stato indicato come indisponibile nella giornata del 07/10/2018, ed infatti non riporta designazioni per tale giorno; allo stesso modo, per l'arbitro A28 per il giorno 06/10/2018. Il quarto ed ultimo vincolo *hard* riguardava l'incompatibilità tra arbitri e squadre: gli arbitri A4, A10 e A33 non sono stati rispettivamente designati per le squadre Virtus Bologna, Brindisi e Cremona. Per la soluzione prodotta il costo dei vincoli

soft violati è stato pari a 3.496.

4.5 Risultati e considerazioni

L'algoritmo creato è in grado di rispettare tutti e quattro i vincoli *hard*, e tenere in considerazione anche i due vincoli *soft*. La soluzione precedentemente esposta ha necessitato 0.658 secondi per essere generata; tale valore è coerente con la natura propria degli algoritmi *greedy* di essere particolarmente performanti.

La soluzione fornita riguarda l'intera stagione 2018/2019, tuttavia potrebbe essere più realistico in futuro limitare il periodo sul quale operare a poche singole e specifiche giornate (eventualmente selezionate dal designatore stesso). In ottica di migliorare ulteriormente le soluzioni proposte, potrebbe essere iterato diverse volte l'algoritmo fino a selezionare la soluzione con il più basso numero di vincoli *soft* violati. Alcuni di questi importanti accorgimenti ed estensioni verranno approfonditi nell'ultimo capitolo "Sviluppi futuri".

Capitolo 5

Conclusioni

Questo elaborato ha avuto lo scopo di progettare e realizzare un sistema per supportare le designazioni arbitrali nella pallacanestro, in particolare per la Serie A italiana. L'implementazione del progetto è avvenuta in due fasi: la predizione della difficoltà delle gare e l'assegnamento automatico degli arbitri.

La predizione della difficoltà degli incontri è stato il focus principale di questo progetto ed anche il più complesso, poiché inerente ad un campo di ricerca estremamente contemporaneo ai giorni nostri e senza una chiara e ben definita soluzione. Non essendoci articoli scientifici inerenti a questo task, si è proceduto inizialmente ad analizzare le feature ritenute più rilevanti, e cercare di predire la difficoltà delle gare in tre distinte classi: "Facile", "Media" e "Difficile". Queste ultime sono state inserite in cinque diversi dataset, contenenti le gare dalla stagione sportiva 2013/2014 fino alla 2018/2019. Il miglior algoritmo è risultato essere la rete neurale Feed-forward, la quale è riuscita a raggiungere un'accuratezza media massima del 42.72% con il quarto dataset, contenente le principali metriche sulla classifica (posizione, punti, totale partite vinte e perse). Il risultato conseguito è certamente modesto, ma pur sempre superiore al 33.33%, base di partenza presa come riferimento (scelta casuale tra le tre classi di difficoltà). La rete neurale è riuscita comunque a performare meglio di qualunque altro algoritmo implementato, attestandosi sempre al di sopra del 42% circa. L'ipotesi di esistenza di una

correlazione tra la probabilità di vincita di una squadra e la difficoltà della relativa gara si è dimostrata essere debole. Tuttavia, dall'articolo scientifico del progetto "TLGProb" si è tratta l'interessante idea di pesare dinamicamente i dataset in base alla gara da predire, in termini di distanza temporale tra le gare.
I motivi di tali risultati possono essere molteplici. Il primo potrebbe essere proprio la difficoltà della scelta delle metriche più rilevanti: il Voto Lega si è dimostrato essere correlato al 65% con la difficoltà delle gare, ma è un'informazione nota solamente a posteriori, e quindi non direttamente utilizzabile per la predizione. Si potrebbe però cercare di stimare tale voto, per entrambe le squadre, ma introdurrebbe un ulteriore livello di complessità al problema. Dopo alcuni brevi test, è possibile però arrivare anche alla conclusione che il Voto Lega non sia una metrica davvero rilevante per il nostro fine. La predizione, inoltre, potrebbe risultare difficoltosa anche a causa dell'eterogeneità dei dati.

La seconda parte di questo elaborato, invece, si è concentrata sull'implementazione di un algoritmo che permettesse l'assegnamento automatico degli arbitri alle gare. Una volta definito il problema e i relativi vincoli *soft* ed *hard*, si è proceduto a creare un algoritmo greedy che svolgesse il task richiesto. La semplicità di creazione e la velocità di questi algoritmi, anche se a volte a discapito della qualità della soluzione trovata, li rendono particolarmente interessanti. In poco meno di 1 secondo è stato in grado di assegnare 36 arbitri alle 238 gare di Serie A per la stagione sportiva 2018/2019. La prima parte del progetto, cioè la predizione della difficoltà delle gare, è fondamentale poiché in base a tali predizioni l'algoritmo sarà in grado di assegnare gli arbitri solamente alle gare che potranno, dal punto di vista dell'esperienza, arbitrare. Molteplici sono i miglioramenti apportabili all'algoritmo implementato, e i principali saranno discussi nel prossimo ed ultimo capitolo.

L'idea di questo progetto è quella di agevolare e supportare l'attività del designatore, il quale compie scelte cruciali sia per il corretto svolgimento delle gare, sia per la crescita che la carriera degli arbitri stessi. Assegnare i giusti arbitri a

tutte le gare può sembrare un compito facile, ma non lo è. Necessita di grande conoscenza del mondo della pallacanestro, delle squadre, classifiche, giocatori e in particolare degli arbitri stessi. La potenza dell'intelligenza artificiale in quest'ambito sarà sicuramente in grado di apportare un enorme contributo: in termini di qualità delle designazioni (poiché gli algoritmi sono in grado di elaborare quantità enormi di dati) e in termini operativi (dove si hanno un gran numero di gare e l'obiettivo è quello di assegnare ad ognuna gli arbitri, pur rispettando tutti i vincoli preposti). Per quanto possa essere ottima una soluzione proposta al designatore, è bene che sia sempre l'uomo a convalidarla (ed eventualmente modificarla) in modo definitivo prima di renderla ufficiale, poiché potrebbe essere a conoscenza di dinamiche non rilevabili dalla macchina.

Capitolo 6

Sviluppi futuri

Questo elaborato vuole essere solo una base di partenza dalla quale migliorare notevolmente le tecniche utilizzate e gli algoritmi implementati.

Predizione difficoltà gare

Il primo aspetto che verrà migliorato sarà quello riguardante la predizione della difficoltà delle gare future, poiché è cruciale. Questo potrà essere fatto percorrendo alcune possibili strade tra cui, ad esempio, provare a sostituire (o affiancare) il Voto Lega con l'equivalente dell'NBA: il "GMSC" (*Game Score*). Un ulteriore sviluppo potrebbe riguardare l'applicazione più estesa degli algoritmi non supervisionati per cercare di clusterizzare le singole gare oppure le squadre in base ad altre metriche. L'importante apporto del progetto "TLGProb" è sembrato davvero all'avanguardia nel suo settore, tuttavia è necessario lavorare ulteriormente per adattarlo alla realtà italiana e continuare ad approfondire l'eventuale correlazione tra la probabilità di vincita di una squadra e la difficoltà della gara.

Referee Assignment Problem

L'algoritmo realizzato per l'assegnamento automatico degli arbitri (noto anche come RAP) è sicuramente una buona base di partenza. Tuttavia può essere ulteriormente affinata aggiungendo altri vincoli *hard* e *soft*, come ad esempio:

- Impostare un numero variabile di arbitri a seconda dei campionati delle gare da designare (vincolo *hard*): in questo elaborato si è tenuto conto esclusivamente della Serie A, tuttavia potrebbe essere davvero una svolta poter offrire questo sistema a tutta la pallacanestro italiana, a qualunque livello. Per fare ciò, quindi, è necessario adattarlo alle varie realtà, poiché ogni regione e campionato ha le proprie regole e il proprio numero di arbitri (ad esempio la Serie A ed A2M hanno 3 arbitri, la Serie B ne ha 2 e la U18 regionale dell'Emilia-Romagna 1);

- Considerare la distanza percorsa dagli arbitri (vincolo *soft*): per i comitati regionali è molto importante ridurre i costi. Questo significa che, al pari di tutti gli altri vincoli, è possibile contenere le spese legate agli spostamenti degli arbitri;

- Indisponibilità tra arbitri (vincolo *hard*): nel corso del tempo potrebbero nascere rare esigenze per le quali due arbitri è bene che non siano designati assieme, e nel caso in cui si presentassero è necessario rispettarle;

- Limitare le designazioni per ogni arbitro ad una sola per giornata (vincolo *soft*): in particolare per i campionati nazionali è estremamente raro che un arbitro venga designato per due partite all'interno della stessa giornata (weekend). Diversamente avviene per gli arbitri regionali, seppur con una distinzione tra campionati;

- Imporre un intervallo temporale prima del quale un arbitro possa tornare ad arbitrare la stessa squadra (vincolo *hard*): è molto importante prevenire l'insorgere di possibili problemi evitando di assegnare un arbitro troppo frequentemente ad una squadra (e questo vincolo *soft* già esiste), ma potrebbe essere necessario evitare due o addirittura tre designazioni consecutive per la stessa società.

Sempre riguardo all'algoritmo relativo al RAP, sarebbe ottimale limitare la sua esecuzione a specifici intervalli temporali/giornate di campionato (selezionabili dal designatore). Nella realtà, infatti, il designatore procede a designare una giornata

alla volta. In tal senso, è opportuno anche che l'algoritmo tenga in considerazione le designazioni delle giornate passate, così da mantenere sempre più equilibrata possibile la media di designazioni e rotazione delle squadre. Fermo restando che la convalida finale della soluzione proposta spetta alla persona (designatore), potrebbe essere molto utile avere la possibilità di fornire all'algoritmo alcune designazioni già prestabilite, in modo tale che quest'ultimo le riproponga nella soluzione così come gli sono state fornite, tenendone però comunque conto nella generazione della soluzione finale.

Struttura

Il progetto realizzato all'interno di questo elaborato vede implementate le due componenti come a sé stanti, se pur con l'ottica di poterle integrare facilmente a vicenda in futuro. Infatti, al momento le due componenti lavorano in ambienti isolati e, al fine di un utilizzo realistico, necessiterebbero di comunicare tra loro, ad esempio anche attraverso lo stesso database.

Ufficiali di Campo

In questo scritto si sono menzionati solamente gli arbitri e il loro designatore, tuttavia nella pallacanestro la squadra CIA (Comitato Italiano Arbitri) è composta sia dagli arbitri che dagli Ufficiali di Campo (UDC). Questi ultimi, all'interno di una partita di pallacanestro, risiedono al tavolo e hanno il compito di trascrivere tutto ciò che avviene durante la gara, gestire i vari cronometri e altro ancora. Anche gli UDC vengono designati alle gare, esattamente come gli arbitri; i ruoli ricoperti però sono differenti: "Segnapunti", "Cronometrista" e "Addetto 24 Secondi". Un futuro miglioramento del progetto corrente riguarderà anche la categoria degli UDC. In particolare, per la parte dell'assegnamento automatico delle partite, potrebbe essere necessario introdurre un ulteriore vincolo *soft* che garantisca una omogenea distribuzione e rotazione tra i diversi ruoli.

6.1 Inserimento in Designazioni.it

L'idea di questo elaborato fonda le sue radici nel sito Designazioni.it[1]. Quest'ultimo nasce da un'idea personale di poter evolvere il sistema di designazioni utilizzato nella maggior parte d'Italia per quanto riguarda gli arbitri e ufficiali di campo. Il suo scopo è quello di agevolare l'attività del designatore aiutandolo ad inserire ed aggiornare in modo automatico tutte le gare, filtrare quelle che deve designare, gestire le indisponibilità dei tesserati, le accettazioni, i rifiuti e molto altro ancora. Nato nel 2016, il progetto è continuato ad evolversi fino a raggiungere una maturità che gli ha consentito di poter essere, ad oggi, utilizzato da tutti i designatori arbitri e UDC dell'Emilia-Romagna e dai designatori nazionali di Serie A, A2 Maschile e A1 Femminile. Di seguito vengono riportate alcune schermate originali.

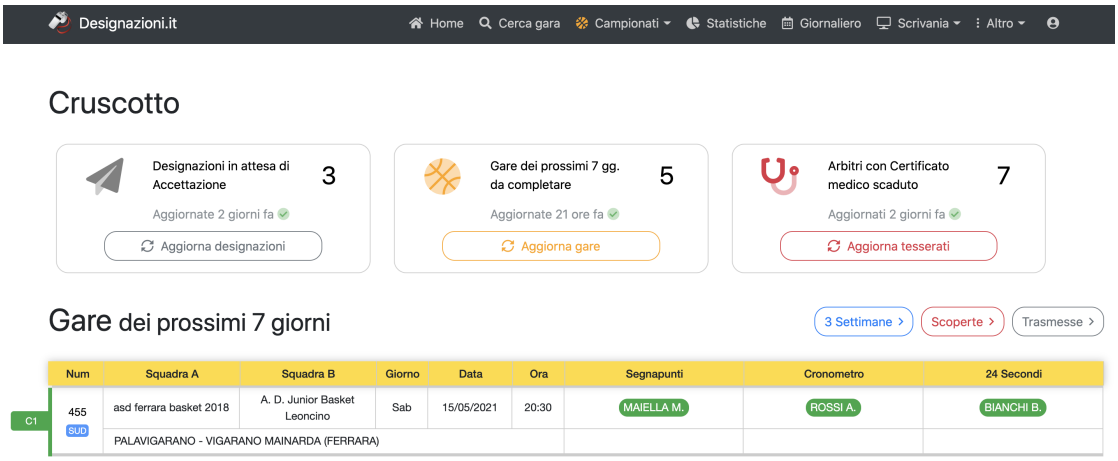


Figura 6.1: Pagina "Home" dell'applicazione web Designazioni.it

[1] `www.designazioni.it`

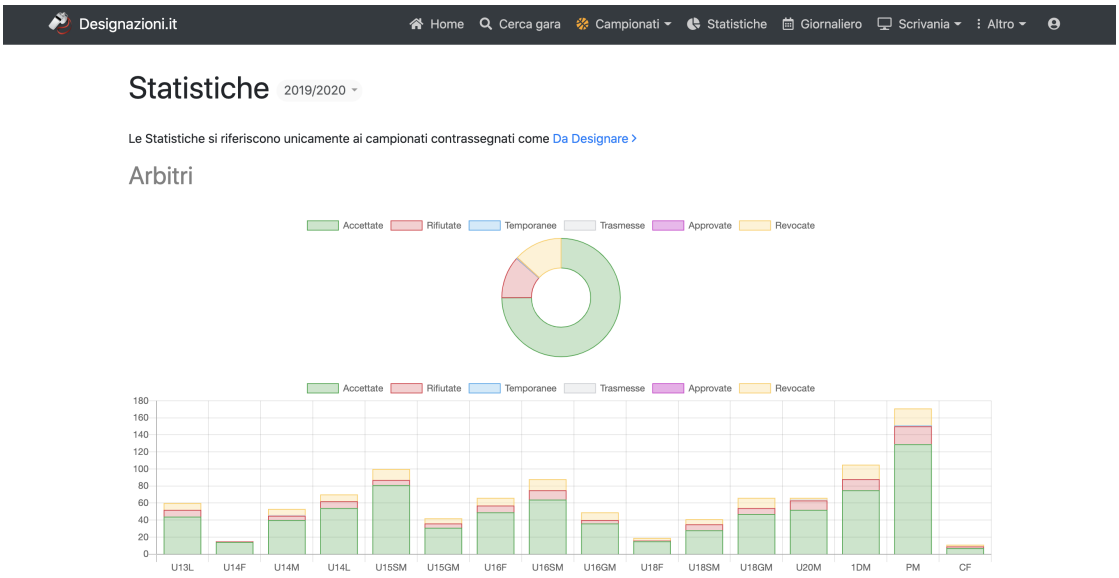


Figura 6.2: Pagina “Statistiche” dell’applicazione web Designazioni.it

La grande evoluzione da introdurre nell’applicazione web è proprio il tema di questo elaborato: predire la difficoltà delle gare future (in particolar modo per i campionati nazionali, per i quali si dispone dei dati statistici) e proporre soluzioni automatizzate per le designazioni arbitrali.

L’utilizzo di un avanzato supporto alle decisioni potrebbe rappresentare una svolta a livello nazionale per quanto riguarda le attività dei designatori arbitri e ufficiali di campo. Ai più alti livelli (campionati nazionali) potrebbe aiutarli a migliorare ulteriormente la qualità delle designazioni, mentre a livello regionale costituirebbe un notevole aiuto nella programmazione delle designazioni di migliaia di gare ogni mese. Infine, un interessante uso d’applicazione potrebbero essere i diversi tornei giovanili nazionali che si svolgono durante l’arco di ogni anno, dove centinaia di arbitri dirigono più di mille gare in pochissimi giorni.

Bibliografia

[1] Fernando Alarcón, Guillermo Durán e Mario Guajardo. «Referee assignment in the Chilean football league using integer programming and patterns». In: (2013). DOI: https://doi.org/10.1111/itor.12049.

[2] Raquel Y.S. Aoki, R. Assunção e Pedro O.S. Vaz de Melo. «Luck is Hard to Beat: The Difficulty of Sports Prediction». In: (2017). DOI: https://doi.org/10.1145/3097983.3098045.

[5] Matthew Beckler, Hongfei Wang e Michael Papamichael. «NBA Oracle». In: (2008). DOI: https://www.mbeckler.org/coursework/2008-2009/10701_report.pdf.

[6] Gercek Budak et al. «Optimization of Harmony in Team Formation Problem for Sports Clubs: A Real-Life Volleyball Team Application». In: (2018). DOI: http://www.ijastnet.com/journal/index/906.

[7] Ge Cheng et al. «Predicting the Outcome of NBA Playoffs Based on the Maximum Entropy Principle». In: (2016). DOI: https://www.mdpi.com/1099-4300/18/12/450.

[9] Alexandre R. Duarte, Celso C. Ribeiro e Sebastián Urrutia. «A Hybrid ILS Heuristic to the Referee Assignment Problem with an Embedded MIP Strategy». In: (2007). DOI: https://doi.org/10.1007/978-3-540-75514-2_7.

[10] Alexandre R. Duarte et al. «Referee Assignment in Sports Leagues». In: (2007). DOI: https://doi.org/10.1007/978-3-540-77345-0_11.

[11] Guillermo Duran et al. «Programación Matemática Aplicada al Fixture de la Primera División del Fútbol Chileno». In: (2005). DOI: http://www.dii.cl/~ris/RISXIX/RISXIXpaper3.pdf.

[12] Duran et al. «Programación del Fixture de la Segunda División del Fútbol de Chile mediante Investigación de Operaciones». In: (2010). DOI: https://ri.conicet.gov.ar/handle/11336/16691.

[13] Kelly Easton, George Nemhauser e Michael Trick. «Solving the Travelling Tournament Problem: A Combined Integer Programming and Constraint Programming Approach». In: (2003). DOI: https://doi.org/10.1007/978-3-540-45157-0_6.

[14] J.R. Evans, John E. Hebert e Richard F Deckro. «Play ball - The scheduling of sports officials». In: (1984). DOI: https://www.researchgate.net/publication/284687263_Play_ball_-_The_scheduling_of_sports_officials.

[15] James R. Evans. «A Microcomputer-Based Decision Support System for Scheduling Umpires in the American Baseball League». In: (1988). DOI: https://doi.org/10.1287/inte.18.6.42.

[16] Paolo Giuliodori. «An Artificial Neural Network-based Prediction Model for Underdog Teams in NBA Matches». In: (2017). DOI: http://ceur-ws.org/Vol-1971/paper-09.pdf.

[17] Evan Heit, Paul C Price e Gordon Bower. «A model for predicting the outcomes of basketball games». In: (1994). DOI: https://onlinelibrary.wiley.com/doi/abs/10.1002/acp.2350080703.

[18] A.J. Hoffman e T.J. Rivlin. «When Is A Team "mathematically" Eliminated?» In: (1971). DOI: https://doi.org/10.1515/9781400869930-023.

[20] Max W. Y. Lam. «One-Match-Ahead Forecasting in Two-Team Sports with Stacked Bayesian Regressions». In: (2017). DOI: https://content.sciendo.com/view/journals/jaiscr/8/3/article-p159.xml.

[21] Amina Lamghari e Jacques A. Ferland. «Metaheuristic methods based on Tabu search for assigning judges to competitions». In: (2010). DOI: https://doi.org/10.1007/s10479-008-0498-8.

[22] Amina Lamghari e Jacques A. Ferland. «Structured Neighborhood Tabu Search for Assigning Judges to Competitions». In: (2007). DOI: https://doi.org/10.1109/SCIS.2007.367696.

[23] Rodrigo Linfati, Gustavo Gatica e John Willmer Escobar. «A flexible mathematical model for the planning and designing of a sporting fixture by considering the assignment of referees». In: (2019). DOI: https://doi.org/10.5267/j.ijiec.2018.6.004.

[25] Dragan Miljković et al. «The use of data mining for basketball matches outcomes prediction». In: (2010). DOI: https://ieeexplore.ieee.org/document/5647440.

[26] Tom Mitchell. «Machine Learning». In: (1997). DOI: http://www.cs.cmu.edu/~tom/mlbook.html.

[27] Lazaros Ntasis. «Optimization techniques for basketball players under the convex risk measures». In: (2019). DOI: http://rua.ua.es/dspace/handle/10045/100412.

[28] Kathleen Jean Shanahan. «A model for predicting the probability of a win in basketball». In: (1984). DOI: https://doi.org/10.17077/etd.8wi48qk1.

[29] Farners Vallespí i Soro. «The Referee Assignment Problem». In: (2019). DOI: http://hdl.handle.net/2117/166628.

[30] Renato Amorim Torres. «Prediction of NBA games based on Machine Learning Methods». In: (2013). DOI: https://homepages.cae.wisc.edu/~ece539/fall13/project/AmorimTorres_rpt.pdf.

[31] Lawrence W.Robinson. «Baseball playoff eliminations: An application of linear programming». In: (1991). DOI: https://doi.org/10.1016/0167-6377(91)90089-8.

[32] D. de WerraL, L. Jacot-Descombes e P.Masson. «A constrained sports scheduling problem». In: (1990). DOI: https://doi.org/10.1016/0166-218X(90)90019-9.

[33] M. B. Wright. «Scheduling English Cricket Umpires». In: (1991). DOI: https://doi.org/10.1057/jors.1991.93.

[34] M. B. Wright. «Scheduling English Cricket Umpires». In: (2015). DOI: https://doi.org/10.1057/9781137534675_6.

[35] Mesut Yavuz, Umut H. Inan e Alpaslan Fığlalı. «Fair referee assignments for professional football leagues». In: (2008). DOI: https://doi.org/10.1016/j.cor.2007.01.004.

[36] Eftim Zdravevski e Andrea Kulakov. «System for Prediction of the Winner in a Sports Game». In: (2010). DOI: https://link.springer.com/chapter/10.1007\%2F978-3-642-10781-8_7.

Sitografia

[3] *Artificial Intelligence in NBA Basketball*. URL: https://www.insidescience.org/news/artificial-intelligence-nba-basketball.

[4] *BDsports*. URL: https://bodai.unibs.it/bdsports/.

[8] *CPLEX Optimizer IBM*. URL: https://www.ibm.com/it-it/analytics/cplex-optimizer.

[19] *How data analytics is revolutionizing the NBA*. URL: https://digital.hbs.edu/platform-digit/submission/how-data-analytics-is-revolutionizing-the-nba/.

[24] Leonardo Mazzeo. *Sport più seguiti in Italia: la classifica*. URL: https://www.sisal.it/scommesse-matchpoint/blog/fuori-campo/classifica-sport-piu-seguiti-in-italia.

Elenco delle figure

Elenco delle tabelle

Elenco dei codici

Ringraziamenti

Desidero ringraziare il professor. Marco Di Felice, relatore di questa tesi, per avermi permesso di lavorare su un argomento che unisce le mie due passioni: la pallacanestro e l'informatica.

Un grande grazie lo rivolgo a Claudio, caro amico di vita, e a tutti i miei amici di Santa Maria, con i quali condivido una vita piena di risate e divertimento.

Un pensiero va ai miei amici arbitri di pallacanestro, insieme ai quali affronto ogni anno nuove sfide, esperienze e soddisfazioni.

Uno speciale grazie a Leonardo Burchi, senza il quale questo progetto probabilmente non esisterebbe e per avermi supportato nel corso degli anni.

Ringrazio anche i miei amici di corso, in particolare Lorenzo e Andrea, che hanno vissuto appieno insieme a me questa avventura, e senza i quali non sarebbe mai stata la stessa cosa.

Ringrazio inoltre i miei nonni, i miei zii e i miei parenti per avermi accompagnato nella mia crescita.

Un enorme e speciale grazie va alla mia fidanzata Mary, per amarmi in un modo che non credevo fosse possibile e per trascorrere insieme e serenamente la nostra vita.

Il ringraziamento più grande, infine, è rivolto ai miei genitori: la soddisfazione di questo importante traguardo raggiunto va a loro, ad entrambi.