Progetto Basket Shots

Predizione dell'esito di tiri nella pallacanestro competitiva
utilizzando Support Vector Machines

Coppola Matteo	Palazzi Luca	Vivace Antonio
793329	793556	793509

Data Technology e Machine Learning,
Febbraio 2019

Indice

1 Introduzione

1.1 Dominio di riferimento e obbiettivi

Il settore sportivo professionistico è in costante ricerca di strumenti, modelli e modalità che permettano di analizzare gli andamenti delle partite e le performance dei singoli giocatori.

Il basket americano, ad esempio, utilizza in maniera intensiva indici statistici per monitorare le prestazioni degli atleti durante il corso della stagione. Ne esistono a centinaia [1] e vengono costantemente consultati da allenatori, giornalisti ed osservatori in quanto, talvolta, si rivelano indicatori del valore reale di un giocatore e descrittori di vari aspetti delle partite.

L'NBA, lega professionistica della regione statunitense, oltre ad essere il più grande spettacolo sportivo al mondo è anche un investimento economico di notevole portata e con ampi margini di profitto. Nel 2017 il valore medio di una squadra ha raggiunto 1.1 miliardi di dollari e, per mantenere una continua crescita ed ambire al primo posto in classifica, ogni team cerca di impossessarsi sia dei migliori *players* sul mercato sia degli strumenti più all'avanguardia.

È in quest'ultimo ambito che il Machine Learning può fornire il suo contributo: analizzando le varie performance e gli esiti delle partite passate si possono fare previsioni di vario genere su molti aspetti di questo sport. Questo è possibile perchè alcuni servizi e siti, ufficiali e non, mettono a disposizione le statistiche e i dati di cui abbiamo parlato prima in grandi datasets, che noi utilizzeremo come training sets.

Strumenti di previsione e di classificazione possono essere utili anche per chi scommette sugli esiti delle partite, ambito anch'esso estremamente remunerativo e popolare negli Stati Uniti.

Il nostro progetto si pone l'obiettivo di individuare ed utilizzare un metodo di Machine Learning per predire il risultato (*made* o *missed*) di un tiro a canestro in NBA, utilizzando alcuni degli indici ritenuti più rilevanti e significativi abbinati alle informazioni inerenti alle partite e ai giocatori coinvolti.

Tutto il lavoro prodotto, i dataset iniziali, intermedi e integrati, gli script

R e Python, le routine Java utilizzate dalla pipeline eseguita da Talend e questo documento LaTeX sono disponibili pubblicamente nel repository Git basket-shots [2].

2 | Data Technology

2.1 Acquisizione

Dopo aver scelto il dominio applicativo di riferimento e aver deciso gli obiettivi principali, abbiamo scelto due dataset ospitati dalla piattaforma Kaggle.

2.1.1 Shot logs

`Shot_logs.csv`[3] contiene 128 000 record riguardanti i tiri a canestro effettuati da 281 giocatori NBA diversi nella stagione 2014-2015. La fonte originale di questi dati è l'API pubblica del sito dell'NBA.

Tabella 2.1: Campi presi in considerazione del dataset `shot_logs.csv`.

Attributo	**Tipo di dato**	**Descrizione**
Matchup	String	Data dell'incontro e squadre partecipanti
Location	factor(home, away)	In casa o fuori casa
W	factor(win, loss)	Partita vinta o persa dalla squadra dell'attaccante
Final margin	int	Scarto tra punteggi delle due squadre a fine partita
Shot number	int	Numero del tiro da inizio partita
Period	factor(1..7)	Periodo della partita. Sono 4 periodi nel basket, più altri 3 supplementari in caso di pareggio tra le due squadre al termine dei periodi 4, 5 e 6.
Game clock	Date/Time	Minuto e secondo della partita in cui si è effettuato il tiro

Shot Clock	Date/Time	Secondo in cui il tiro viene rilasciato dall'attaccante. L'intervallo va da 0 a 24, in quanto 24 è il tempo massimo a disposizione della squadra per effettuare un'azione
Dribbles	int	Numero di dribbling effettuati dall'attaccante prima del tiro
Touch time	float	Secondo del possesso palla in cui l'attaccante ha tirato
Shot dist	float	Distanza dal canestro al momento del tiro
PTS type	factor(2,3)	Tipo di tiro effettuato (da 2 o da 3)
Shot result	factor(made, missed)	Esito del tiro (a segno o fallito)
Closest defender	String	Nome del difensore più vicino all'attaccante
Close def dist	int	Distanza tra il difensore più vicino e l'attaccante
Player name	String	Nome dell'attaccante

`percentage_previous_game` è una nuova feature che abbiamo computato dai dati esistenti. Rappresenta la percentuale di successo del tiro fino a quel momento in stagione. Il suo valore è stato ottenuto dal seguente script Python che, facendo uso della libreria *Pandas*, aggrega i dati delle partite precedenti a quella a cui si riferisce il tiro in questione.

Listato 1: `datasets/shot_logs_nv.py`

```
import pandas as pd

data = pd.read_csv(r"shot_logs.csv")
df = data.sort_values(by=['player_name', 'GAME_ID'])
df_agg = df.groupby(['player_name',
↪  'GAME_ID']).agg({'SHOT_NUMBER':'max',
↪  'FGM':'sum'}).reset_index()
df_cs =
↪  df_agg.groupby('player_name').agg({'SHOT_NUMBER':'cumsum',
↪  'FGM':'cumsum'})
df_agg = df_agg.drop(columns=['SHOT_NUMBER', 'FGM'])
```

```
df_agg = df_agg.join(df_cs)
df_agg['% Previous'] = round(df_agg['FGM'] /
↪  df_agg['SHOT_NUMBER'], 2)
df_shift = df_agg.groupby('player_name')['% Previous'].shift(1)
df_agg = df_agg.drop(columns=['SHOT_NUMBER', 'FGM', '%
↪  Previous'])
df_final = df_agg.join(df_shift)
df_final = pd.merge(df, df_final, on=['player_name', 'GAME_ID'],
↪  how='inner')
df_final.to_csv(r"shot_logs_wpp.csv")
```

Nel caso in cui la partita sia la prima della stagione, `percentage_previous_game` assume il valore `TS`, ovvero la percentuale di successo del tiro riferito a tutta la stagione (*True Shooting*), proveniente dall'altro dataset che abbiamo preso in considerazione, *Seasons stats*.

2.1.2 Seasons stats

`Seasons_stats.csv` [4] contiene le statistiche inerenti agli atleti e alle loro performance, dal 1950 al 2017. I dati sono originari del sito Basketball Reference[5], prelevati usando tecniche di *web scraping*.

Tabella 2.2: Campi presi in considerazione del dataset `Season_stats.csv`.

Attributo	Tipo di dato	Descrizione
Year	int	Anno di gioco
Player	String	Nome del giocatore
Pos	String	Ruolo nella squadra
Age	int	Età durante quella stagione
Tm	String	Nome abbreviato della squadra
Games	int	Partite giocate in quella stagione
MP	int	Minuti totali giocati
PER	float	*Player Efficiency Rating*
TS%	float	*True Shooting Percentage*
BLK%	float	*Block Percentage*, la percentuale di blocchi effettuati

PER è indicatore tecnico che mira a riassumere diversi fattori che descrivono la performance di un giocatore in un unico valore. Definito nel 1989 da John

Hollinger:

> The PER sums up all a player's positive accomplishments, subtracts the negative accomplishments, and returns a per-minute rating of a player's performance.

Il calcolo del suo valore è piuttosto complesso [6]. Ulteriori informazioni ed analisi sulla rilevanza statistica di questo indicatore sono disponibili in [7].

TS%, misuratore tecnico per valutare l'efficienza di tiro di un giocatore, è invece definito in [8] come:

$$TS\% = \frac{PTS}{2(FGA + (0.44 \times FTA))} \times 100 \tag{2.1}$$

Dove

- PTS (points scored) è il punteggio ottenuto,
- FGA (field goal attempts) sono i tentativi di tiro dal campo (da 2 o da 3 punti),
- FTA (free throw attempts) sono i tiri liberi tentati.

È stata effettuata un'integrazione, descritta nella sezione 2.3, affinchè ogni tiro registrato in *Shot logs* riporti anche le statistiche e gli indicatori tecnici dell'attaccante (colui che effettua il tiro) e del difensore (colui che attua il blocco per ostacolare il tiro).

Abbiamo scelto di rendere anonime le istanze del nuovo dataset rimuovendo nomi di attaccanti e difensori, cosicchè il modello di Machine Learning preveda i risultati indipendentemente dai giocatori coinvolti, e si basi sulle loro statistiche, sul contesto di gioco e sugli altri valori forniti.

2.1.3 Ipotesi fatte a priori

Come per ogni altro processo statistico abbiamo innanzitutto dovuto individuare alcune ipotesi ritenute valide, assumendole tali, per poter orientare la parte di Machine Learning verso l'obbiettivo prefissato.

Come fondamentale assunzione, riteniamo che l'esito del tiro a canestro possa essere dedotto in maniera abbastanza regolare dagli attributi messi a disposizione dai due dataset utilizzati. In realtà, per ogni tentato canestro giocano un'innumerevole serie di fattori fisici ed ambientali: per esempio, l'esito viene anche influenzato da come il giocatore poggia il peso poco prima di tirare.

Lavorando sulle performance dei giocatori dell'NBA, supponiamo anche che le loro abilità e la loro coerenza di tiro rimangano più o meno costanti (ad eccezione di poche partite sopra o sotto il loro standard, situazioni di *outlier*).

I due dataset forniscono valori a noi utili che risalgono alla stagione di campionato 2014-2015. Volendo utilizzare il modello per prevedere gli esiti dei tiri NBA della corrente stagione (2018-2019), anche avendo dati aggiorati, dobbiamo supporre che non siano stati introdotti profondi cambiamenti nel regolamento e nelle strategie attuate dagli allenatori, affinchè le dinamiche di gioco rimangano analoghe.

Inoltre, risulterà infattibile prevedere l'esito di tiri effettuati da nuovi giocatori entrati nella lega, poichè alcune delle metriche che utilizziamo sono relative allo storico delle loro performance. Una possibile soluzione a questo problema è ricavare valori approssimativi per questi indici consultando altre basi di dati che contengono dati relativi alle performance del giocatore in *altre* leghe.

2.1.4 Misure di qualità dei dataset

Season stats

Vengono definiti 53 attributi per ogni giocatore. Alcuni di questi tuttavia rappresentano delle criticità che sporcano il dataset, abbassandone la qualità sia a livello di schema sia a livello di istanze.

Innanzitutto riscontriamo una mancata correttezza rispetto al modello: gli attributi `blanl` e `blank2` sono indubbiamente campi inutili: nessuna istanza ha valori definiti per queste due colonne. Sono probabilmente refusi della fase di scraping dei dati e vanno eliminati perchè non esistenti nel modello ER e perchè intaccano la minimality del dataset.

Molte ennuple hanno valori mancanti soprattutto in alcuni attributi specifici, mostrando un chiaro caso di incompletezza: sul totale di 1 283 984 valori 154 921 sono vuoti, per un'incompletezza che si attesta sullo 0.12%.

Tabella 2.3: Incompletezza di `Season_Stats.csv`.

Year	Player	Pos	Age	PER	TS%	BLK%
0,003	0,003	0,003	0,003	0,024	0,006	0,158

Infine la chiave principale per le istanze è un ID incrementale che va da 0 a 24 692. Si tratta di una chiave non molto utile per il tipo di dataset:

le statistiche dei giocatori andrebbero indicizzati con un ID univoco che si riferisce al singolo atleta. In questo modo sarebbe stato più facile eseguire l'integrazione tra i due dataset e, se avessimo voluto fare un'integrazione più ampia tra dataset di leghe diverse, sarebbe stato più facile rintracciare le statistiche dello stesso giocatore in campionati diversi.

Shot logs

Questo dataset presenta meno criticità, risultando più completo. Nonostante ciò, l'attributo `shot_clock` presenta un'incompletezza del 4.35%: per 5 577 istanze su 128 069 il suo valore non è specificato. Infine i valori delle colonne `player_name` e `closest_def` mancano di correttezza: giocatori con lo stesso id (quindi corrispondenti allo stesso giocatore) sono rappresentati con formati e capitalizzazioni diverse: `player_name` infatti il segue il formato `Cognome, Nome` con le iniziali maiuscole, mentre in `closest_def` il formato è `nome cognome` tutto in minuscolo. È un caso di eterogeneità a livello di istanze.

Questo problema caratterizza il 100% delle istanze.

2.2 Preparazione

Abbiamo fatto notare che nel file `shot_logs.csv` i nomi degli attaccanti (attributo `player_name`) e i nomi dei difensori (attributo `closest_defender`) hanno un formato diverso. Per eseguire una corretta integrazione con l'altro dataset è necessario quindi uniformare preliminarmente questi valori, permettendone l'identificazione, sia esso l'attaccante o il difensore nel tiro in questione.

Non si tratta di *deduplication* perchè quest'ultima mira ad individuare istanze che si riferiscono ad uno stesso concetto all'interno dello stesso database, mentre qui non abbiamo istanze duplicate, ma valori diversi che in domini differenti si riferiscono alla stessa entità.

Inizialmente avevamo optato per una funzione che separasse i nomi in *token*, li ordinasse lessicograficamente e che li risolvesse rilevando un match per una totale distanza di edit minore di 3 o maggiore solo in caso di diminutivi (per esempio: `J.J. Barea` e `Barea Jose Juan`).

Questa funzione è stata implementata in Python, nel file `datasets/match.py` [2].

Il metodo funzionava correttamente, ma ci siamo poi accorti che due attributi del dataset (`player_id` e `closest_def_id`) corrispondevano univocamente. Abbiamo quindi utilizzato la componente *tMap* in Talend per

uniformare i nomi di attaccanti e difensori: abbiamo ricercato giocatori con lo stesso id e abbiamo sostituito il loro nome nella colonna `player_name` (nome dell'attaccante) con quello della colonna `closest_defender`.

Il risultato di tale operazione ci ha permesso una più agevole integrazione con l'altro dataset, poichè quest'ultimo non possiede un ID univoco per i giocatori e il processo integrativo deve essere necessariamente effettuato facendo corrispondere i nomi degli atleti.

Infine bisogna specificare che per ovviare all'assenza di 5577 occorrenze nel campo `shot_clock`, abbiamo deciso di considerare il valore medio per giocatore.

Per i nostri scopi, dal dataset *Seasons Stats* otteniamo un sottoinsieme delle statistiche dei giocatori riferite all'anno 2015. Al suo interno è possibile trovare fino a quattro duplicati della coppia di attributi Year e Player, in quanto nella lega è permesso giocare in quattro differenti squadre ogni stagione. Ciò è stato risolto aggregando i valori degli attributi sopra citati, gestendo le statistiche numeriche (`BLK`, `TS`, `PER`) considerando la media stagionale e considerando invece l'occorrenza più recente per gli attributi `Pos` ed `Age`.

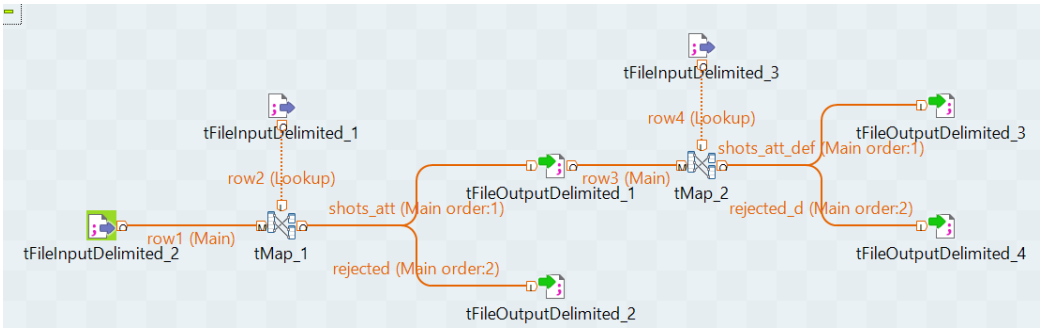
Alcuni record che riguardavano giocatori di cui non abbiamo informazioni per la stagione interessata, sono stati rimossi. E.g. *Atila Dos Santos* compare in alcune istanze del primo dataset ma lo stesso non è presente nel secondo dataset che descrive le caratteristiche dei giocatori.

2.3 Integrazione

L'analisi esplorativa dei due datasets, anche in vista dell'applicazione dei metodi di Machine Learning, ha prodotto come risultato due nuovi datasets composti da un insieme ridotto di attributi, ritenuti rilevanti e non ridondanti per lo studio e gli obiettivi prefissati. Il numero di record è rimasto inalterato.

L'integrazione dei due nuovi datasets ha avuto luogo in due passi separati. Inizialmente, è stata utilizzata una componente *tMap* per associare al dataset dei tiri le statistiche degli attaccanti. Dopodiché, questo dataset consistente dei nuovi attributi è stato utilizzato assieme al dataset delle statistiche per associare, con un'altra componente *tMap*, le informazioni relative al difensore.

Figura 2.1: Pipeline d'integrazione dati implementata con *Talend Studio*



Non utilizzando un ID univoco che identifica un giocatore per effettuare l'integrazione tra i due datasets ma i nomi stessi degli atleti, abbiamo optato per l'utilizzo della stessa funzione descritta nella sezione 2.2, così da uniformare i nomi presenti (scritti in minuscolo ordinati lessicograficamente) nei due datasets.

Effettuata l'integrazione, per verificare la consistenza del matching in ciascuno dei due passi è stato creato un file denominato *rejected* in cui sono state inserite le istanze respinte, non trovando una corrispondenza perfetta.

Circa una decina di nomi sono risultati in questa lista: sono risolti manualmente nella routine `EditShotLogs_0.1.java`2, eseguita in pipeline da Talend, poichè sarebbe stato problematico e dispendioso usare funzioni di edit distance appositamente parametrizzate per così pochi e particolari casi.

Listato 2: Metodo matchNames da EditShotLogs_0.1.java

```
public static String matchNames(String name1) {
        String[] def = {"Barea, Jose Juan", "Hardaway Jr., Tim",
                "Aminu, Al-Farouq", "Nene", "Mbah a Moute, Luc",
```

```
                "Hayes, Charles", "Lucas III, John", "Taylor, Jeff",
                "Rice Jr., Glen", "Datome, Gigi", "McAdoo, James Michael"};
        String[] stats = {"J.J. Barea", "Tim Hardaway",
                "Al-Farouq Aminu", "Nene Hilario", "Luc Mbah",
                "Chuck Hayes", "John Lucas", "Jeffery Taylor",
                "Glen Rice", "Luigi Datome", "James Michael"};
        String[] results = {"barea jj", "hardaway tim",
                "al-farouq aminu", "nene hilario", "luc mbah",
                "chuck hayes", "john lucas", "jeffery taylor",
                "glen rice", "datome luigi", "james mcadoo michael"};
        if (Arrays.asList(def).contains(name1) ||
                Arrays.asList(stats).contains(name1)) {
                for (int i = 0; i < def.length; i++) {
                        if (name1.equals(def[i]) ||
                                        name1.equals(stats[i])) {
                                        return results[i];
                        }
                }
        }
        String[] names1 = name1.replaceAll("[^a-zA-Z ]", "")
                .toLowerCase().split("\\s+");
        Arrays.sort(names1);
        String nuova = "";
        for (int i = 0; i < names1.length; i++) {
                nuova += names1[i];
                nuova += " ";
        }
        nuova = nuova.trim();
        return nuova;
}
```

2.3.1 Misure di qualità del dataset integrato

Currency, Volatility e Timeliness

Un aspetto importante dei dati coinvolti nel nostro processo di integrazione è il loro cambiamento nel tempo. I database a nostra disposizione contemplano l'anno di campionato 2014-2015.

Currency, definita come la velocità degli aggiornamenti [9], è calcolabile con la seguente formula[10]

$$Currency = Age + (DeliveryTime - InputTime) \tag{2.2}$$

Age misura l'età dei dati quando vengono ricevuti, *DeliveryTime* è l'istante in cui il prodotto che utilizza queste informazioni è consegnato all'utente finale mentre *InputTime* è l'istante in cui il dato è stato effettivamente ottenuto. Il termine $(DeliveryTime - InputTime)$ misura quindi il periodo di tempo che trascore prima che il prodotto che utilizza i nostri dati sia pronto ed effettivamente utilizzabile.

Poichè l'anno corrente è 2019, i dati hanno una *Age* di 4 anni. $(DeliveryTime - InputTime)$ risulta quindi trascurabile e *Currency* è equivalente ad *Age*.

Supponendo inoltre che le statistiche dell'NBA rimangano rilevanti per le 3 stagioni successive, la *volatilità* dei dati è uguale a 3 anni.

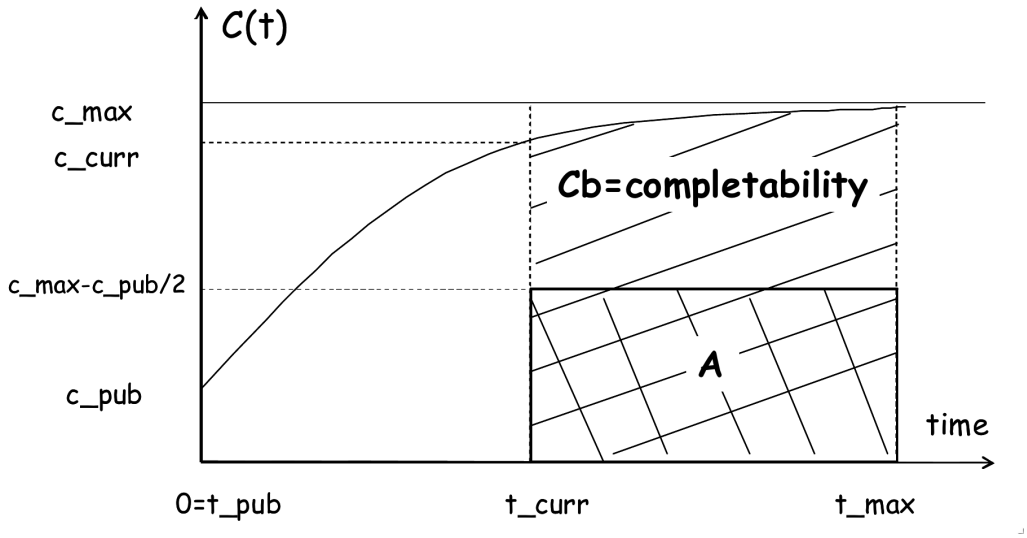
Timeliness, calcolata con

$$\max\{0, 1 - \frac{Currency}{Volatility}\} \tag{2.3}$$

Vale quindi 0, rappresentando una cattiva tempestività del nostro dataset.

Completability

Figura 2.2: Rappresentazione grafica della completabilità [9]



Essendo i nostri dataset originari di servizi Web e riflettendo eventi reali distribuiti durante l'anno, ne vengono costantemente pubblicate versioni aggiornate.

La *Completabilità* è un'interessante metrica che ci permette di visualizzare la dinamica di evoluzione temporale della completezza.

Consideriamo una funzione $C(t)$, definita come il valore della completezza all'istante t, con $t \in [t_{pub}, t_{max}]$, dove t_{pub} è l'istante iniziale di pubblicazione dei dati e t_{max} il tempo massimo in cui verrà completato l'ultimo degli aggiornamenti dei dati previsiti.

La *completabilità* dei dati è quindi definibile come [9]:

$$\int_{t_{curr}}^{t_{max}} C(t) \tag{2.4}$$

Con t_{curr} l'instante in cui essa viene calcolata ($t_{curr} < t_{max}$).

La figura 2.2 generalizza come si presenta, dove c_i valore di completezza stimato per un generico t_i.

La Completabilità è definita dall'area segnata come *Cb*. Il confronto con A, definita dalla formula 2.5 permette di definire gli intervalli $[Alto, Media, Bassa]$ per la completabilità.

$$A = (t_{max} - t_{curr}) * \frac{c_{max} - c_{pub}}{2} \tag{2.5}$$

Per dare un'idea di che completabilità esibiscono i dataset che consideriamo supponiamo che il ritardo con cui otteniamo i dati sulle partite appena avvenute sia trascurabile, i.e. ad un istante t si abbiano i dati delle partite fino a $t-1$.

Stimiamo la completezza percentuale dei dati, in un istante t, con

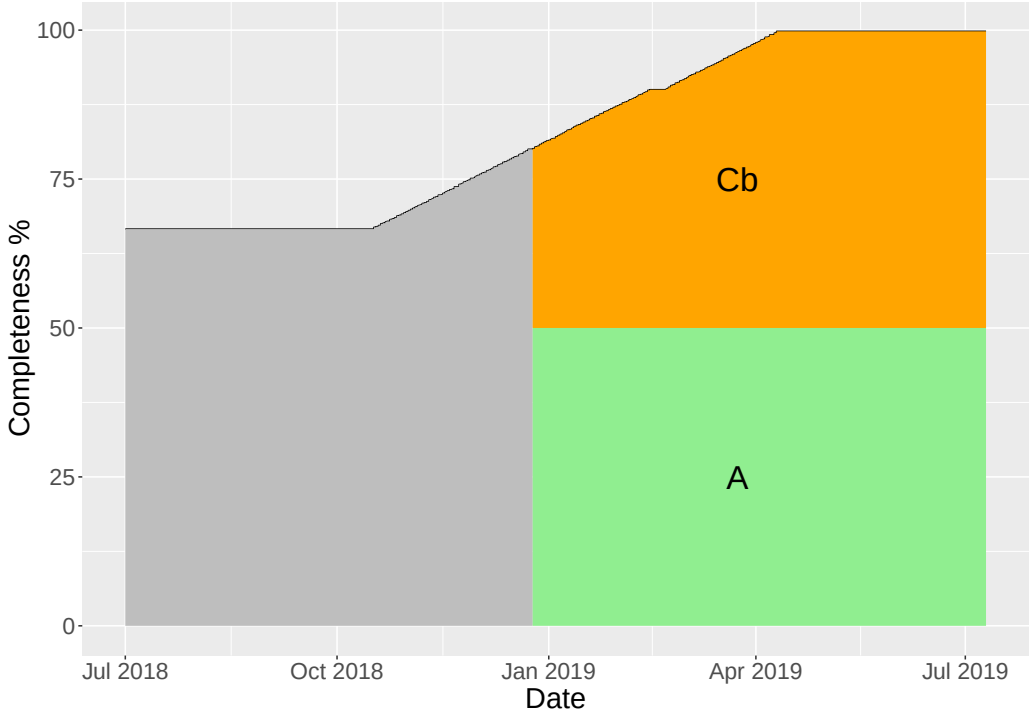
$$C(t) = \frac{\sum_{y=c-p-1}^{c-1} M_y + m_{c,t}}{\sum_{y=c-p-1}^{c} M_y} \times 100 \tag{2.6}$$

Dove

- M_y è il numero totale di partite nella stagione y;
- $m_{c,t}$ il numero totale di partite nella stagione y fino all'istante t;
- c la stagione considerata
- p numero di stagioni precedenti a quella che si vuole considerare che si suppongono rilevanti e parte del dataset.

Considerando tutti gli istanti t in cui siano giocate delle partite, ottenute da [11] e utilizzando $y = 2019, c = 3$ (considerando quindi i dati delle stagioni 2019, 2018, 2017, 2016, con 2019 stagione oggetto del dataset), la completabilità evolve secondo la figura 2.3 attestandosi su un valore medio.

Figura 2.3: Rappresentazione grafica della completabilità nel nostro esempio



Scegliendo una finestra temporale più estesa la completabilità mostra un andamento periodico.

Infatti da inizio stagione (Ottobre) fino alla sua fine (Maggio) continuano ad essere forniti dati nuovi dopo ogni singola partita; nel periodo da Maggio a Ottobre invece non ci sono partite e nessun nuovo dato viene generato; infine ad Ottobre inizia un nuovo campionato, rendendo obsoleti i dati più vecchi della finestra mentre si abbassa la completezza.

2.4 Analisi descrittiva del dataset integrato

Dato il tipo di informazione che desideriamo ottenere dal dataset integrato, abbiamo valutato una serie di statistiche che caratterizzano fortemente i dati, ottenuti con lo script `count`.

Media di tiri in stagione	455.72
Media di blocchi	271.31
Media dei tiri con esito positivo	206.05
Media dei tiri con esito negativo	249.67

Tabella 2.4: Statistiche *per giocatore* del dataset integrato

La relazione tra numero di giocatori e tiri *made* è mostrata nella figura 2.4, mentre quella con i tiri *missed* nella figura 2.5.

Figura 2.4: Frequenza di tiri con esito positivo

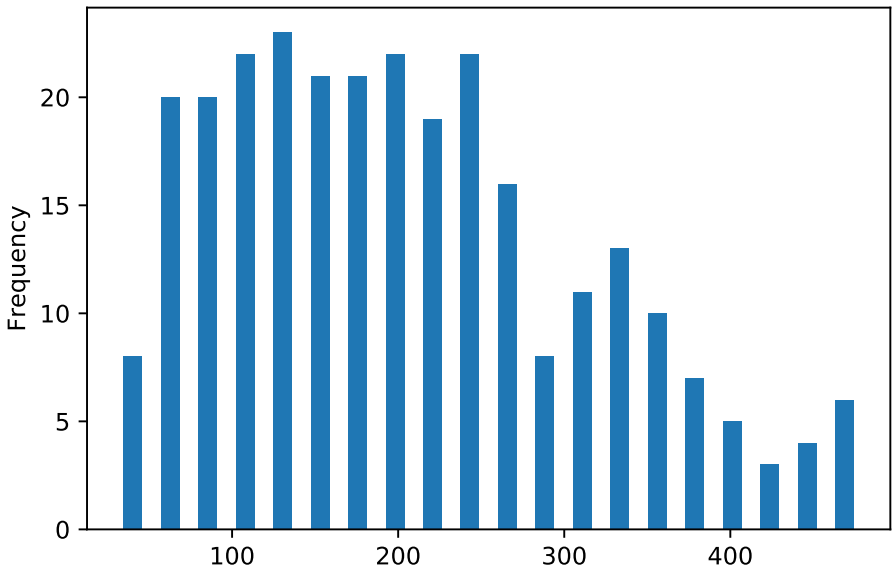
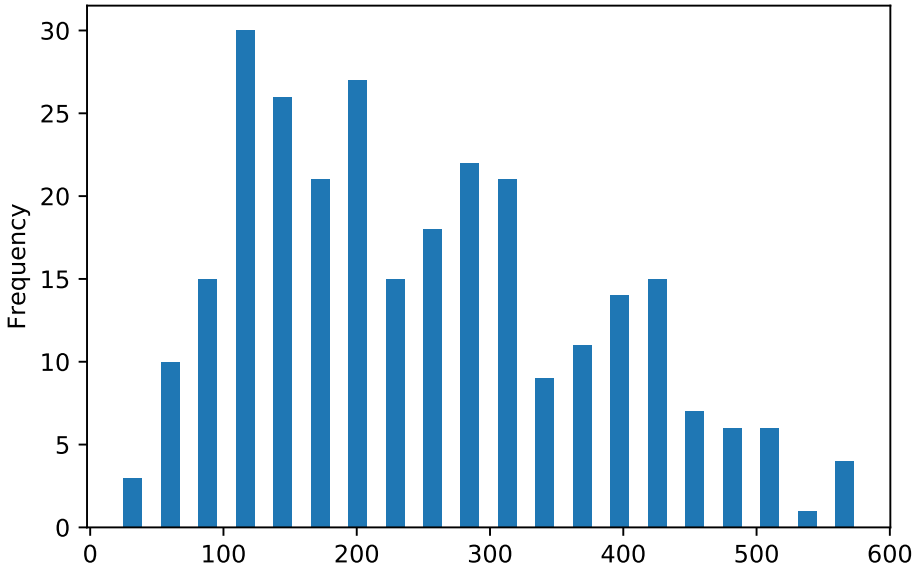


Figura 2.5: Frequenza di tiri con esito negativo



3 | Machine Learning

La scelta del modello di machine learning da utilizzare è in gran parte determinata dalle caratteristiche dei dati a disposizione.

Iniziamo dunque con un analisi del nostro dataset per individuare le caratteristiche che descrivono gli attributi e in che modo i valori che assumono sono distribuiti nelle nostre istanze.

3.1 Analisi esplorativa

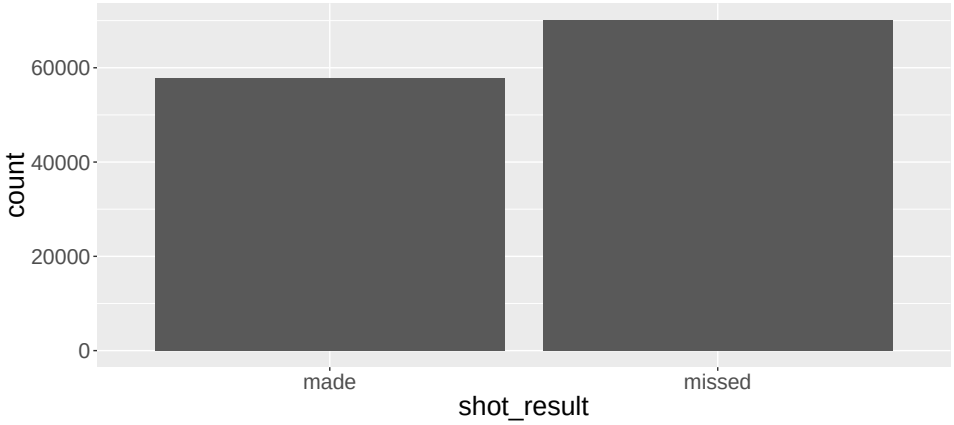
Il dataset integrato risultante possiede per ciascuno dei 128 000 tiri informazioni relative al tiro e dati relativi agli atleti coinvolti nello scontro che ha portato allo stesso.

In Shot logs sono stati esclusi alcuni attributi (descritti nella tabella 2.1) riguardanti la partita come `W`, `Final margin` e `Matchup`, ritenuti non rilevanti per le sorti di un tiro a canestro. Anche `Period` e `Game clock` sono stati ritenuti superflui, in quanto è già presente l'attributo `Shot number`, così come non significativo è stato valutato l'attributo `Pts type` (deducibile dall'attributo `Shot distance`).

Analogamente, in Seasons stats sono state escluse decine di indici statistici non determinanti, oltre agli attributi (descritti in tabella 2.2) come `Games` e `MP`.

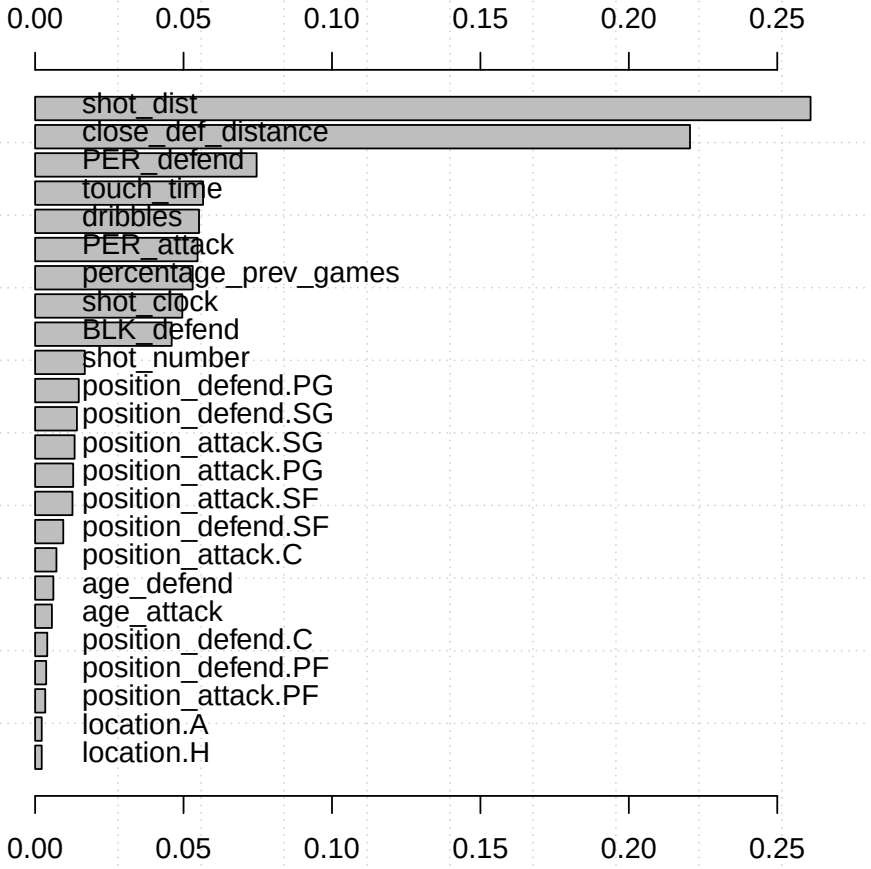
Nell'analisi di questi dati, è emerso lampante che avere a disposizione altri fattori più circostanziali al tiro, che sono reperibili grazie al *visual tracking*, come la parabola del tiro stesso, la velocità dell'attaccante e del difensore, le coordinate sul rettangolo di gioco dei due giocatori e così via, avrebbe aiutato a creare modelli più approfonditi in modo da fornire predizioni più fedeli. Riprendiamo la possibilità di usare altri dati nella sezione 4.2.

Figura 3.1: Distribuzione degli esiti nel dataset integrato



I valori da predire sono distribuiti in modo non uniforme. Il 55% (70 157 record) sono tiri *missed*, mentre il 45% (57 900 record) sono tiri *made* (figura 3.1).

Figura 3.2: Importanza delle feature, secondo il modello prodotto



Importance, mostrata nella figura 3.2, rappresenta il contributo informativo delle feature considerate. Le principali risultano essere `shot_dist` e `close_def_distance`.

Figura 3.4: Distanza media del tiro rispetto al ruolo del giocatore

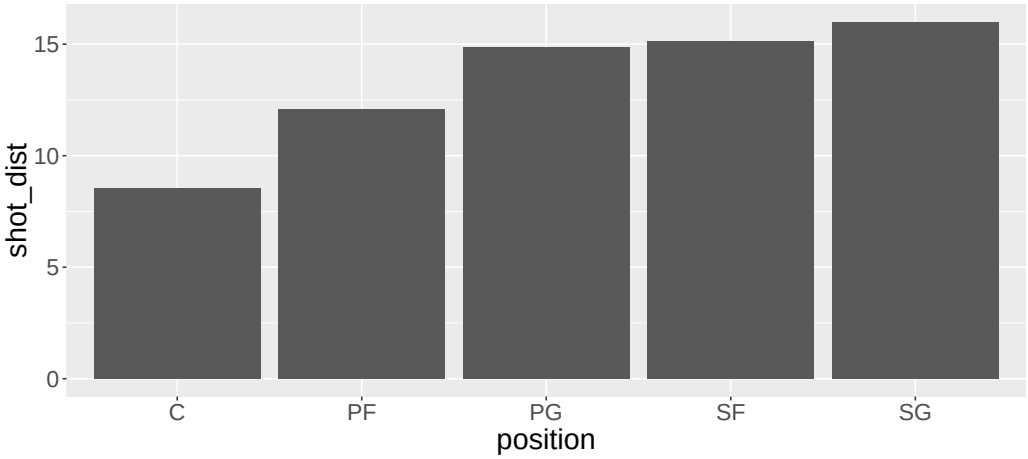
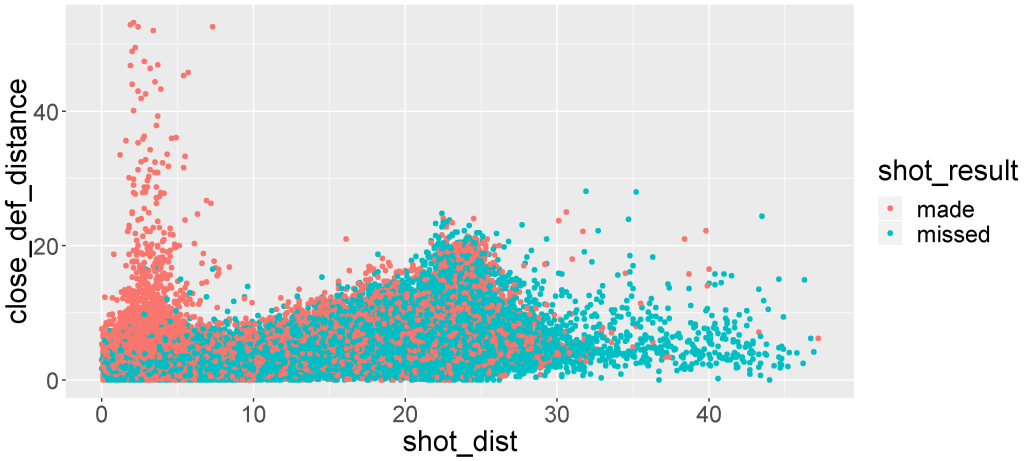


Figura 3.3: Correlazione tra gli attributi più rilevanti



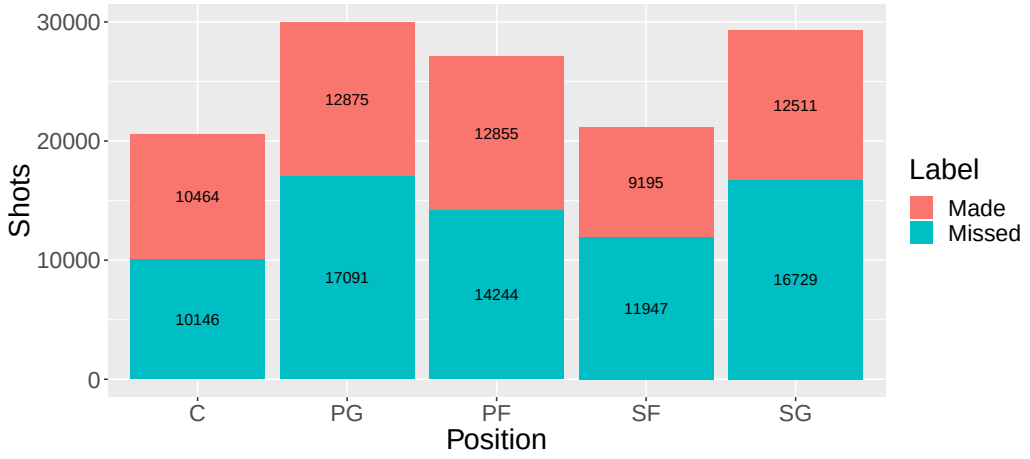
Il grafico in figura 3.3 è stato interpretato considerando due regioni: la regione con prevalenza `made`, ossia quella compresa tra $[0, 10]$ nell'asse delle ascisse e $[5, 60]$ nell'asse delle ordinate, e la regione con prevalenza `missed`, ossia quella compresa tra $[25, 40]$ nell'asse delle ascisse e $[0, 20]$ nell'asse delle ordinate. Abbiamo intepretato le distribuzioni di queste due regioni rintracciando quello che succede sul campo: i tiri effettuati dal pitturato (area sottostante al canestro), seppur con il difensore molto vicino, hanno una probabilità di essere messi a segno molto alta. I tiri effettuati dalla distanza, invece, sono notoriamente più difficili ed in questo caso è la probabilità di fallimento ad essere più elevata.

Il grafico in figura 3.4 mostra che C, il componente della squadra che dom-

ina il pitturato, tira in media da una distanza inferiore ai 10 piedi, seguito da PF. I giocatori in questo ruolo prendono quindi tiri ad indice di difficoltà più basso rispetto alla media. Non abbiamo a disposizione informazioni relative a peso e altezza, ma i dati [5] mostrano che in media in questi due ruoli giocano gli atleti più fisicamente dotati: soffrendo meno il contatto con l'avversario riescono a raggiungere più facilmente il canestro.

Al contrario, gli altri tre ruoli (PG, SF e SF) tirano da una distanza che si aggira intorno ai 15 piedi. Solitamente sono meno prestanti dal punto di vista fisico ma con un'abilità al tiro superiore, che li porta a prendere anche tiri di notevole difficoltà dalla distanza.

Figura 3.5: Successo dei tiri rispetto al ruolo del giocatore



La figura 3.5 conferma quest'ultima intepretazione: PF ha una percentuale di missed pari a 0.53, mentre addirittura inferiore (0.49) è la la percentuale di missed per C, per di più l'unico ruolo ad avere un numero di tiri made maggiori rispetto ai missed ma anche quello con il minor numero di tiri effettuati. Gli altri tre ruoli (PG, SF, SG) hanno una percentuale di missed pari a 0.57.

3.2 Scelta del modello

Il nostro dataset integrato ha un buon numero di attributi, sia numerici che categorici. Abbiamo quindi deciso di utilizzare SVM, un algoritmo di apprendimento automatico supervisionato ampiamente utilizzato per problemi di classificazione. Il suo punto di forza è l'utilizzo del cosiddetto *kernel trick*, strumento matematico per mappare l'input in uno spazio multi-dimensionale. SVM performa bene con dataset composti da tanti attributi e numerose osservazioni, adeguato sotto questo punto di vista al nostro caso. La presenza di attributi categorici porterebbe a preferire gli alberi di decisione, ma esistono alcune tecniche che possono trasformare queste tipologie di campi affinchè siano compatibili anche con le SVM.

3.3 Preparazione del dataset

La fase di Data integration è stata impostata in modo tale da produrre un dataset pulito e direttamente sottoponibile ad un modello di Machine Learning.

Ricordiamo che gli attributi relativi ai nomi degli atleti, resi necessari dalla mancanza di un ID univoco, vengono rimossi dopo aver effettuato l'integrazione.

Il primo passo effettivo, quindi, è stato il preprocessing dei dati. I valori nulli di ciascun attributo erano stati rimossi durante l'integrazione. È da verificare, invece, la presenza di valori inconsistenti nel dominio dell'attributo preso in considerazione. Per `touch_time` sono stati trovati valori non positivi. Considerando che questo attributo rappresenta periodi di tempo non nulli e minori di 24 secondi, valori non negativi non sono ammessi e sono stati rimossi. Non sono stati trovati altri attributi con valori anomali.

È stata applicata una normalizzazione del dataset per gli attributi numerici. È un procedimento che viene applicato frequentemente perché gli attributi sono calcolati con unità di misura differenti. In questo modo si evita che un attributo abbia un peso maggiore di un altro. È stato utilizzato per ciascun attributo il metodo

$$\text{(min max)} \quad x = \frac{x - min(x)}{max(x) - min(x)} \tag{3.1}$$

molto utilizzato in letteratura in alternativa a

$$\text{(z score)} \quad x = \frac{x - \mu}{\sigma} \tag{3.2}$$

in quanto gli attributi sono limitati in un range. In questo modo i valori di ciascun attributo vengono posti tra 0 e 1.

Non essendo SVM una tecnica che gestisce attributi categorici, è stata applicata una tecnica di *one-hot encoding* per gestirli. In questo modo vengono creati tanti nuovi attributi quante sono le categorie di ciascun attributo. Il nuovo attributo è 1 se l'attributo originale aveva quel determinato valore, 0 altrimenti.

3.4 Implementazione di SVM

Inizialmente abbiamo provato ad implementare SVM con il package R `e1071`, ma il dataset utilizzato contiene un numero eccessivo di istanze e la computazione risultava troppo onerosa (`cannot allocate vector in R of size xx Gb`). Abbiamo sperimentato anche con il package `liquidSVM` ma non erano presenti di default funzionalità utili allo sviluppo del progetto come il calcolo della *ROC* e *AUC*, quindi abbiamo optato per il package `rminer` che implementa l'algoritmo della SVM di Kernlabs, basato sul lavoro di John Platt [12] in cui viene descritto il metodo ad oggi più efficiente per ottenere stime probabilistiche sulla classificazione del test set con una SVM.

Dividiamo l'intero dataset in training e test set con questa funzione R:

```
# Split dataset helper function
split.data = function(data, p = 0.8, s = 1) {
  set.seed(s)
  index = sample(1:dim(data)[1])
  train = data[index[1:floor(dim(data)[1] * p)], ]
  test = data[index[((ceiling(dim(data)[1] * p)) + 1):dim(data)[1]], ]
  return(list(train=train, test=test))
}
```

La seguente riga produce e allena il modello:

```
model=fit(shot_result~., trainset, model="ksvm", task="prob",
          kernel="rbfdot", C=1)
```

Ricordiamo che SVM ha un valore $f(z)$ per ogni istanza di test z

$$f(\mathbf{z}) = \mathbf{w}^T \phi(\mathbf{z}) + b, \tag{3.3}$$

$$= \sum_{i \in SV} \alpha_i y_i \kappa(\mathbf{x}_i, \mathbf{z}) + b. \tag{3.4}$$

Il risultato è un numero reale

$$f(\cdot) : \mathbb{R}^d \mapsto \mathbb{R}$$

Per ottenere etichette binarie utilizzando la soglia predefinita, viene applicata la funzione scalino (o Heaviside) su quest'ultimo:

$$\hat{y}(z) = \Theta(f(z))$$

Per ottenere $f(z)$ e non direttamente $\Theta(f(z))$, imposteremo il modello per eseguire il task *prob* invece del predefinito *class*, anche se il nostro è un problema di classificazione.

In questo modo, $f(z)$ da numero reale viene scalato in un intervallo $[0, 1]$ con la tecnica chiamata *Platt scaling* [12], producendo quindi una stima della probabilità di classificazione.

Aggiungendo alla funzione fit il parametro `search="heuristic10"` consentiamo la ricerca semiautomatica degli iperparametri ottimali su 10 range diversi per massimizzare la predizione del modello.

Dopo una ricerca euristica su un sample di 25 000 istanze, le metriche più accurate sono state ottenute con il valore C della SVM uguale a 1 e kernel *RBFDOT* ossia Radial Basis Function Kernel.

La funzione di discriminazione di tale Kernel è:

$$y(\mathbf{x}) = \sum_{i=1}^{N} w_i \, \phi(\|\mathbf{x} - \mathbf{x}_i\|) \tag{3.5}$$

Infine eseguiamo una 10-fold cross validation con

```
valdata = crossvaldata(shot_result~., dataset, fit, predict,
                       ngroup = 10, task="prob", model="ksvm",
                       kernel="rbfdot", C=1)
```

3.5 Esperimenti e analisi dei risultati

Il comando *mmetric* ci permette di ottenere metriche e stime di performance sul nostro modello:

```
metrics=mmetric(testset$shot_result,prediction,metric=c("ALL"))
```

Accuracy complessiva	61.16
Precision per la classe *made*	61.32
Precision per la classe *missed*	61.09
Recall per la classe *made*	38.49
Recall per la classe *missed*	79.90
F-measure per la classe *made*	47.29
F-measure per la classe *missed*	69.24
Area Under Curve complessiva	0.62

Tabella 3.1: Metriche risultate dell'esecuzione della cross validation su SVM

		Prediction outcome		
		p	**n**	**total**
Actual value	**p′**	4 357	6 962	P′
	n′	2 749	10 932	N′
	total	P	N	

Tabella 3.2: Confusion Matrix di SVM

Il risultato (su un sample di 25 000) intuitivamente un po' basso è in realtà in linea con le aspettative: i fattori che influenzano un tiro a canestro sono molteplici, mentre le informazioni in nostro possesso sono limitate. Il fatto che sia comunque superiore al *random guessing* dimostra che esistono dei fattori nel dataset che tendono effettivamente ad influenzare il successo o il fallimento del tiro.

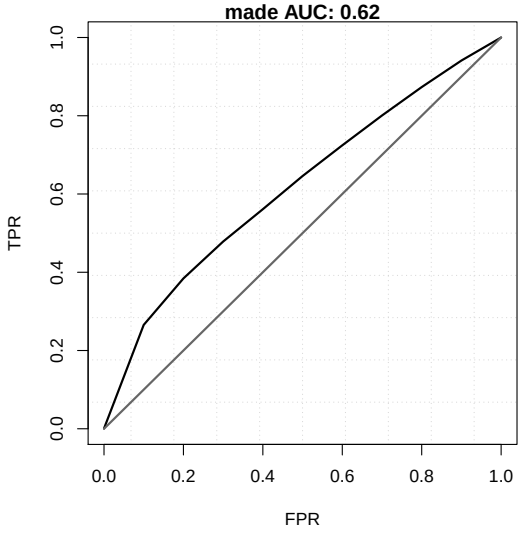
Considerando la classe *made* quella positiva e la classe *missed* quella negativa, dalle metriche ottenute possiamo fare alcune valutazioni: per entrambe le classi il valore di precision è simile ed è sul 60%, da cui deduciamo una proporzionalità nelle classi di falsi positivi e falsi negativi rispetto ai valori target. Guardando le recall, che invece considera i valori da predire, notiamo che per la classe *missed* tendono ad esserci pochi falsi negativi, mentre la vera criticità è il numero eccessivo di falsi positivi per *made*. In altre parole, un tiro legittimo tende ad essere predetto come un fallimento.

Le F-measure sono valori che permettono di confrontare direttamente le due classi. Precision e recall sono spesso diverse e quindi è complicato determinare quale classe abbia effettivamente meno errori: la F-measure, ottenuta computando la media armonica di queste due metriche, permette di quantificare le performance delle due classi in maniera comparabile. Nel nostro caso era deducibile che la f-measure di *missed* fosse molto più alta di quella di *made*: la disparità nelle recall influisce molto sul calcolo complessivo.

Essendo un problema di apprendimento automatico binario, la Area Under Curve, intesa come AUROC (Area Under Receiver Operating Characteristic), è solo una. Tale metrica corrisponde all'area sottostante la ROC, ottenuta muovendo gradualmente la threshold della SVM e rappresentando sul grafo ogni volta i valori di TPR (*True Positive Rate*) e FPR (*False Positive Rate*). Più grande (vicino a 1) è il valore della AUROC, più è grande la distinzione tra True Positive e True Negative: nel nostro caso vale 0.64.

Sia eseguendo SVM sull'intero dataset, diviso opportunamente in training set (80%) e trainset (il restante 20%), sia eseguendolo con una 10-fold cross validation, il valore di accuracy complessivo tra le due esecuzioni non si discosta troppo. Ciò porta a concludere che il dataset è formato da istanze poco sofferenti di bias e difficilmente rischia di andare in overfitting.

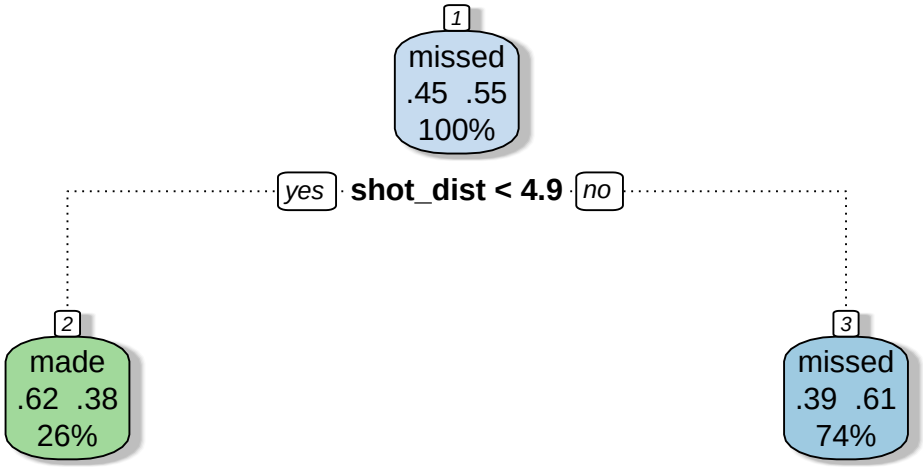
Figura 3.6: ROC di SVM per made



3.5.1 Alberi di Decisione

Per verificare come si comportassero altri modelli diversi dalla SVM, abbiamo dato in input il nostro dataset e abbiamo allenato un modello utilizzando gli alberi di decisione. Innanzitutto è stata eseguita un'euristica per individuare il valore migliore di CP, ossia quello con *xerror* minimo. Questo parametro indica di quanto debba migliorare il relative error complessivo per poter eseguire uno split nel nodo dell'albero. Il risultato però è stato un albero con un solo split:

Figura 3.7: Alberi di decisioni



Un'analisi delle metriche e del contributo informativo delle feature considerate ci mostra il perchè di un albero così corto:

shot_dist	77
shot_clock	8
touch_time	8
close_def_distance	7
percentage_prev_games	1

Tabella 3.3: Importance in DT

L'importance di *shot_dist* è nettamente più alta di qualsiasi altro attributo. Questi valori dimostrano che il decision tree non è adeguato per il nostro problema in quanto non coinvolge attributi che per noi sono influenti e probabilmente generalizzerà male con istanze nuove, nonostante siano state ottenute metriche simili a SVM.

Accuracy complessiva	61.06
Precision per la classe *made*	62.12
Precision per la classe *missed*	60.68
Recall per la classe *made*	35.71
Recall per la classe *missed*	82.01
F-measure per la classe *made*	45.35
F-measure per la classe *missed*	69.75

Tabella 3.4: Metriche risultate dell'esecuzione della cross validation su Decision Tree

		Prediction outcome		
		p	**n**	**total**
Actual value	**p′**	20 639	37 162	P′
	n′	12 587	57 357	N′
	total	P	N	

Tabella 3.5: Confusion Matrix di DT

4 | Conclusioni

L'intero progetto, dalla scelta dei dataset all'analisi dei risultati dell'apprendimento automatico, si è rivelato essere molto ampio, coinvolgendo non solo i temi più comuni riguardo la Data Technology e Machine Learning ma anche finezze statistiche, matematiche e informatiche. Ciò è sicuramente una dimostrazione di quante aree di studio possano essere coinvolte in questi due settori fortemente in crescita, sia nell'ambito della ricerca scientifica che nelle applicazioni commerciali. Andiamo ora a riepilogare il lavoro fatto e ricercarne i possibili margini di miglioramento e ulteriore sviluppo.

4.1 Riepilogo

È difficile determinare se SVM sia stata la tecnica ideale per allenare il nostro dataset: non abbiamo individuato un fattore discriminante che porti a preferirla rispetto ad altri modelli come le reti neurali oppure gli alberi di decisione.

Le metriche mostrano che la nostra SVM predice l'esito del tiro in maniera poco più che sufficiente ed è importante notare che predice meglio un tiro *missed* da uno *made*, come dimostrato dai valori delle due recall.

Abbiamo cercato lavori analoghi al nostro per confrontare e verificare la possibile presenza di errori nella realizzazione del modello. [13] fa un percorso analogo al nostro partendo dallo stesso datasets *Shot logs*, confrontando più algoritmi e raggiungendo un'accuratezza 68% utilizzando *XGBoost*, un'implementazione di *Boosting*. È un approccio che coinvolge diversi classificatori *weak learners* e infine, ne combina le predizioni, pesate differentemente, con un ulteriore algoritmo di apprendimento artificiale (*strong learner*).

4.2 Metriche mancanti

Riteniamo che il limite principale sia quindi la mancanza di attributi più descrittivi, che conferirebbero al computatore una migliore rappresentazione della realtà del fenomeno che abbiamo analizzato. Attualmente, per stimare l'effettivo stato fisico e mentale del giocatore durante la stagione utilizziamo l'attributo `percentage_prev_games`. Supponendo di conoscere e poter utilizzare le statistiche delle stagioni precedenti, un indice ampiamente utilizzato per valutare il tiro di un giocatore è EFG (Effective Field Goal): consiste in una semplice statistica che classifica la bravura del giocatore separando i tiri secondo il loro valore potenziale (tiri da 2 e tiri da 3). Questa metrica era la più avanzata nell'era precedente al *visual tracking*: come già introdotto nella sezione 3.1, ad oggi sempre più fattori vengono analizzati per giudicare le prestazioni degli atleti, in qualsiasi ambito. Un'altra possibilità è utilizzare una metrica basata su QSQ (Quantified Shot Quality) [14]: essa misura quanto bene un giocatore tira in relazione alla difficoltà del tiro stesso e, a differenza di EFG, non si limita ad una statistica pura. Ogni giocatore e tiro corrispondente potrebbe avere quindi come attributo un valore che rappresenta la difficoltà del tiro tenendo in considerazioni i diversi fattori sopra elencati.

Dall'analisi effettuata è un dato di fatto che il modello attuale non faccia discriminazioni profonde: sia con SVM che con Decision Tree l'importance è sbilanciata in favore dell'attributo `shot_dist`. Questa interpretazione dei dati è però limitata in quanto non sempre essere vicini al canestro è sinonimo di efficacia: ad esempio può esserci una situazione in cui il difensore copre bene la zona e l'attaccante è sbilanciato. Attualmente il nostro modello non è in grado di riconoscere queste situazioni. Per lo stesso motivo, essere lontani dal canestro non è sinonimo di fallimento: ad esempio può esserci una situazione che rende agevole il canestro. In questo senso l'attributo `closest_def_dist` aiuta il modello e infatti, con SVM, è secondo per contributo informativo.

4.3 Sviluppi futuri

Per ottenere predizioni ancora più accurate, implementare sensori come misuratori di inerzia, di rotazione e accelerometri sui giocatori e/o sulla palla da basket fornirebbe tutt'altro livello di dettaglio.

Verrebbe quindi aggiunta una mole di dati continua da gestire con tecnologie IoT adeguate. Il basso livello di fattibilità riguardo a queste ultime considerazioni - difficilmente i giocatori di NBA giocherebbero con sensori

addosso - ci permette di intravedere una soglia di incertezza sull'esito dei tiri che, quantomeno nel prossimo futuro, sarà arduo superare.

Riferimenti

[1] NBA Statistics Glossary. https://web.archive.org/web/20130907075032/http://stats.nba.com/glossary.html.

[2] Basket Shots project Git repository. https://github.com/avivace/basket-shots.

[3] NBA Shot logs. https://www.kaggle.com/dansbecker/nba-shot-logs.

[4] NBA Players stats since 1950. https://www.kaggle.com/drgilermo/nba-players-stats#Seasons_Stats.csv.

[5] Basketball-Reference.com. https://www.basketball-reference.com/.

[6] Basketball Reference - PER Definition. https://www.basketball-reference.com/about/per.html.

[7] Tong Zhang, Junlong Chen, and Xinchao Zhao. Modeling and analysis of player efficiency rating for different positions: Case study with nba. In Mark Zhou and Honghua Tan, editors, *Advances in Computer Science and Education Applications*, pages 234–242, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[8] Basketball Reference Glossary. https://www.basketball-reference.com/about/glossary.html.

[9] Carlo Batini and Monica Scannapieco. *Data Quality: Concepts, Methodologies and Techniques*. 01 2006.

[10] Donald Ballou, Richard Wang, Harold Pazer, and Giri Kumar Tayi. Modeling information manufacturing systems to determine information product quality. *Management Science*, 44(4):462–484, 1998.

[11] NBA Schedule, Season 2018-2019. https://fixturedownload.com/results/nba-2018.

[12] John C. Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In *ADVANCES IN LARGE MARGIN CLASSIFIERS*, pages 61–74. MIT Press, 1999.

[13] Brett Meehan. Predicting NBA Shots. `http://cs229.stanford.edu/proj2017/final-reports/5132133.pdf`.

[14] Yu-Han Chang, Rajiv Maheswaran, Jeff Su, Sheldon Kwok, Tal Levy, Adam Wexler, and Kevin Squire. Quantifying shot quality in the nba.