Analisi Performancee e Predizione degli Esiti delle Partite in League of Legends

Componenti del gruppo

Bondanese Vito, [MAT. 697321], v.bondanese@studenti.uniba.it

GitHub: https://github.com/strykl4stel/iCon/

Anno Accademico: 2024-2025

Indice

Il progetto è stato realizzato in Python [3.13.0] in quanto offre diverse librerie già impostate per la manipolazione dei dati e la loro visualizzazione.

Come ambiente di sviluppo e' stato scelto VSCode (e' stato scartato l'utilizzo di altre IDE, Es. PyCharm, sia un po' per gusto personale che per la ridotta "pesantezza" di VSCode in confronto)

Lista delle librerie coinvolte:

pandas = per l'importazione dei dataset .csv

matplotlib = per l'importazione dei dataset .csv

scikit-learn = per l'apprendimento

Capitolo 0 – Introduzione

L'obiettivo di questo progetto è quello di analizzare e comprendere le dinamiche che influenzano l'esito delle partite e analizzare le performance dei giocatori nel videogioco competitivo League of Legends, sfruttando approcci di machine learning supervisionato, non supervisionato e reti bayesiane.

L'intero lavoro si articola in tre moduli principali:

1. **Apprendimento non supervisionato***: individuazione di* **cluster** *di giocatori simili tra loro, utile per rivelare stili di gioco o strategie ricorrenti.*

2. **Apprendimento supervisionato***: creazione di modelli predittivi in grado di stimare la probabilità di vittoria in base alle statistiche individuali del giocatore.*
3. **Classificazione Probabilistica***: utilizzo del modello Naive Bayes per prevedere vittorie e sconfitte*

Questo approccio consente non solo di ottenere modelli predittivi accurati, ma anche di spiegare e interpretare in modo trasparente le strategie vincenti, offrendo uno strumento utile per coach, analyst e appassionati di e-sport competitivi.

Capitolo 1 – Dataset e preprocessing

Il dataset è stato ottenuto separatamente attraverso l'apposita API messa a disposizione dalla società proprietaria del gioco **Riot Games,** *esso riporta 500 partite di 500 giocatori (Ad alti livelli) contenente tutte le features riguardanti il giocatore e le sue performance durante la partita, inclusa di esito finale*

Le **features** *utilizzate sono praticamente quasi tutte utili allo scopo del sistema in quanto rappresentano le performance dei giocatori in modo piuttosto completo:*

champion -> *personaggio giocabile (Campione) utilizzato dal giocatore in tale partita*

side -> *lato della mappa in cui ha giocato il giocatore*

role-> *ruolo coperto dal giocatore durante la partita*

assists -> *numero di assist del giocatore*

damage_objectives -> *danno totale del giocatore agli obiettivi sulla mappa*

damage_turrets -> *danno totale del giocatore alle torri*

deaths -> *numero di morti del giocatore*

gold_earned -> *oro totale guadagnato durante la partita*

(tramite kills, assists, turret_kills, total_minions_killed….)

kills -> *numero di uccisioni avversari di un giocatore*

time_cc -> *tempo totale di effetti di "Crowd Control" (rallentamento, stun,…) inflitti dal giocatore agli avversari*

damage_total -> *danno totale inflitto dal giocatore*

(agli avversari, alle strutture, agli obiettivi)

damage_taken -> *danno totale del giocatore subito dagli avversari*

total_minions_killed -> *numero di minions farmati*

turret_kills -> *numero di torri distrutte da un giocatore durante la partita*

vision_score -> *punteggio visione del giocatore*

result -> *rappresenta il risultato della partita:*

"1" = Vittoria "0" = Sconfitta

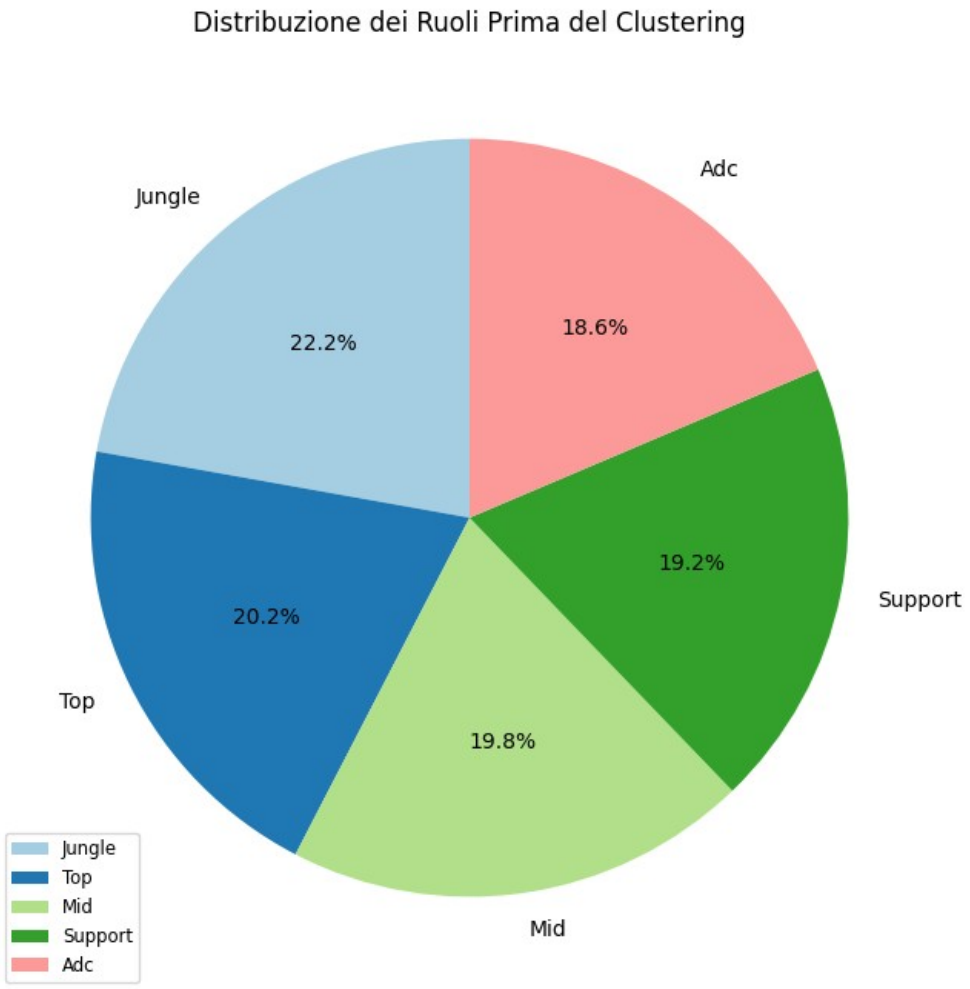
id -> *identificativo del giocatore/partita*

d_spell -> *incantesimo scelto dal giocatore sulla lettera 'D'*

f_spell -> *incantesimo scelto dal giocatore sulla lettera 'F'*

- *Nella fase di* **preprocessing** *e' stata rimossa la feature 'id', inutile al fine della classificazione degli stili di gioco e alla predizione del risultato…*

  …e normalizzate tutte e sole le features numeriche, ciò è necessario in quanto l'algoritmo K-Means per l'apprendimento non supervisionato calcola la **distanza euclidea** *tra i punti per creare i cluster…*
  …la distanza sarà imprecisa se le feature non sono sulla stessa scala, le feature del dataset hanno scale molto diverse: ad esempio, damage_total può superare i **100.000**, *mentre vision_score raramente va oltre* **50**.



Capitolo 2 - Apprendimento non supervisionato

L'apprendimento non supervisionato è una sottoarea dell'apprendimento automatico in cui l'agente è addestrato utilizzando un insieme di dati privi di etichette. Esistono due principali tecniche di apprendimento non supervisionato:

- **Clustering**: *Lo scopo è organizzare gli elementi del dataset in gruppi basati su somiglianze tra di essi.*

- **Riduzione della dimensionalità**: *L'obiettivo è ridurre il numero di variabili (o feature) nel dataset, preservando però le informazioni essenziali. Un esempio di questa tecnica è l'analisi delle componenti principali (PCA).*

Nel caso di questo progetto e' stata utilizzata la tecnica del **clustering** *per andare a raggruppare i giocatori in gruppi (cluster) che rappresentano i diversi stili di gioco individuati*

La tecnica del clustering è a sua volta divisa in due tipologie

- **hard clustering**: *associa ogni esempio ad uno specifico cluster (identificazione di uno "stile")*
- **soft clustering**: *associa a ogni esempio la probabilità di appartenenza a ogni cluster*

...in questo caso come tipologia di clustering e' stato utilizzato l'hard clustering sfruttando l'algoritmo **K-Means**, *uno dei metodi di clustering più conosciuti: l'obiettivo è dividere i dati in* **K** *cluster distinti, in modo che la varianza all'interno di ciascun cluster sia minima.*

La problematica principale nell'utilizzo dell'algoritmo K-Means è la scelta di una valore K ideale, tale problematica può essere risolta mediante l'utilizzo del **metodo del gomito:**

L'obiettivo è identificare il valore di **K** *(numero di cluster) che minimizza la varianza all'interno dei cluster, evitando di scegliere un valore troppo alto che possa portare a un eccesso di clustering,* **Overfitting**

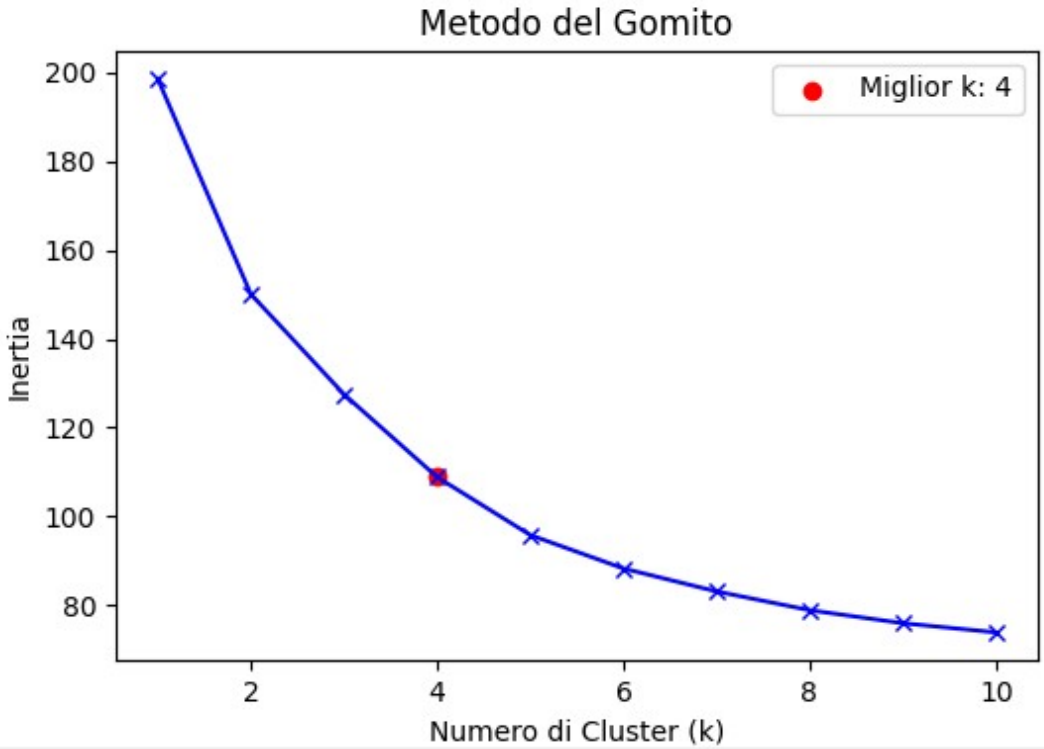
```python
def elbowMethod(dataSet):
    inertia = []
    maxK = 10  # Numero massimo di cluster da testare

    for i in range(1, maxK + 1):
        kmeans = KMeans(n_clusters=i, n_init=10, init='random')
        kmeans.fit(dataSet)
        inertia.append(kmeans.inertia_)
```

il funzionamento si esprime calcolando l'inertia (la somma delle distanze quadrate tra i punti e i centroidi) per vari valori di **K** *(numero di cluster). Viene tracciato un grafico che mostra come l'inertia diminuisce al crescere di K. Il punto in cui la riduzione dell'inertia diventa meno pronunciata (formando un "gomito") indica il numero ottimale di cluster, poiché oltre a quel punto l'aggiunta di cluster non migliora significativamente la separazione dei dati*

Parametri:

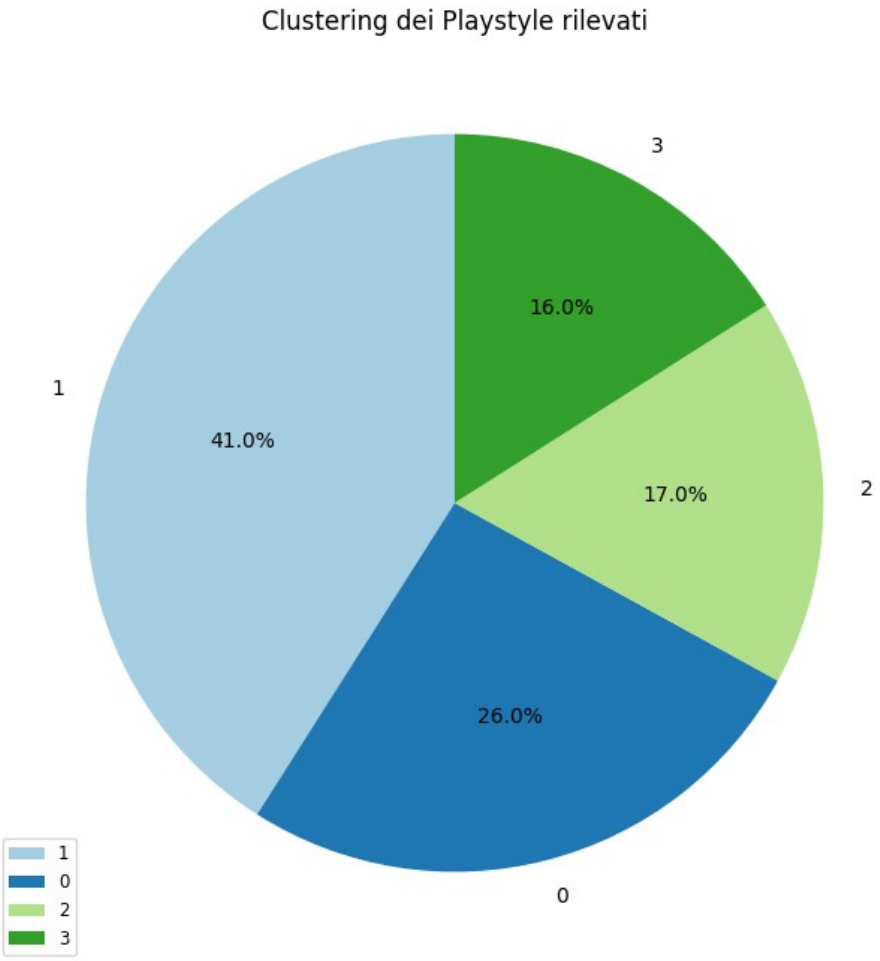
- **n_clusters***: appresenta il numero di cluster che stai testando, in questo caso al più 10 cluster in quanto è improbabile in questo "mondo" trovare più di 10 stili di gioco totalmente differenti tra loro (aumentare questo valore porterebbe ad un aumento di risorsi computazionali per poi produrre risultate potenzialmente errati)*
- **n_init***: indica il numero di volte che K-Means eseguirà l'algoritmo con inizializzazioni randomizzate dei centroidi.*

  L'algoritmo K-Means eseguirà quindi 10 tentativi di inizializzazione per garantire stabilità nei risultati e restituire il clustering migliore
- **init***: indica il tipo di inizializzazione dei centroidi, in questo caso le inizializzazioni avvengono in modo casuale*



Nel caso considerato il numero di cluster K ottimale rilevato dall'elbow method è **k = 4***...*

…valore con cui e' stato sucessivamente eseguito l'algoritmo **k**-*Means creando le seguente suddivisione:*



Il grafico rappresente quindi la suddivisione in k=4 cluster ognuno rappresentante il "Playstyle" rilevato con la relativa percentuale di giocatori associati ad esso

Capitolo 3 – Apprendimento supervisionato

L'apprendimento supervisionato è una tecnica di machine learning in cui un modello viene addestrato su un dataset contenente esempi con input noti e output attesi (etichettati), con l'obiettivo di apprendere una funzione che mappi correttamente nuovi dati.

Esso e' diviso in:

Classificazione: *In cui le etichette rappresentano un inseime finito di valori che la feature di output Y può assumere, nel caso di questo progetto è un caso di classificazione booleana (risultato = 0/1)*

Regressione: *la feature di output può assumere qualsiasi valore numerico (dominio continuo)*

Nel contesto di questo progetto, il task supervisionato consiste nella predizione dell'esito di una partita (result), etichettato come 0 per sconfitta e 1 per vittoria.
A partire da un insieme di feature derivate dalle performance in-game (statistiche numeriche come kills, assists, damage_total, ecc.) e variabili categoriche (role, champion, side, ecc.), il modello supervisionato ha il compito di classificare correttamente il risultato della partita.

Per l'addestramento e la valutazione del modello sono stati adottati tre algoritmi supervisionati:

- **Alberi decisionali**
  Un Decision Tree è una struttura ad albero che prende decisioni dividendo i dati in base a caratteristiche chiave.
  I nodi non foglia sono etichettati con delle condizioni basati sui valori delle feature negli esempi collegati da archi etichettati con "vero" e "falso" .
  I nodi foglia rappresentano le "decisioni" (Y)

- **Random Forest:**
  E' un algoritmo basato su un insieme di alberi decisionali e il valore finale si ottiene facendo la media delle predizioni di ogni albero

- **Regressione Logistica**
  E' un modello di apprendimento utilizzato per la classificazione binaria
  Il modello costruisce una funzione lineare delle variabili di input e applica la funzione sigmoide, che mappa l'output tra 0 e 1, rappresentando la probabilità della classe positiva.

  La previsione finale viene fatta confrontando questa probabilità con una soglia

Iperparametri, Addestramento e Test

Gli iperparametri sono parametri del modello che non vengono appresi durante l'allenamento, ma che devono essere definiti prima dell'addestramento.

Gli iperparametri influiscono direttamente sulle prestazioni del modello.

Alberi decisionali

max_depth*: Limita la profondità massima dell'albero. Un valore maggiore permette all'albero di crescere più profondo, ma potrebbe portare a un overfitting*

min_samples_split*: definisce il numero minimo di campioni richiesti per dividere un nodo interno*
Con valori bassi l'albero può crescere in profondità, aumentando il rischio di overfitting.
Con valori altri l'albero sarà meno profondo e più generalizzabile, riducendo l'overfitting ma se troppo profondo potrebbe generalizzare male non catturando correttamente la complessità del modello

min_samples_leaf*: Il numero minimo di esempi per poter creare una foglia*

criterion*: Determina il criterio per misurare la qualità di una divisione.*

Random Forest

n_estimators*: Indica il numero di alberi da costruire nel modello.*
Un valore troppo alto può portare a tempi di addestramento elevati senza miglioramenti significativi

Regressione Logistica

C*: È il parametro di regolarizzazione. Un valore più piccolo di C significa una maggiore regolarizzazione (penalizzazione dei coefficienti del modello), mentre un valore più grande di C riduce la regolarizzazione,*

lasciando che il modello si adatti meglio ai dati. Un buon valore di C aiuta a evitare l'overfitting e a migliorare la generalizzazione.

solver*: Specifica l'algoritmo utilizzato per ottimizzare i coefficienti della regressione logistica*

penalty*: Tipo di regolarizzazione da applicare ai coefficienti della regressione logistica per prevenire l'overfitting*

Nel progetto è stata utilizzata la tecnica **Grid Search** *in combinazione con la* **K-Fold** *CrossValidation: la Grid Search esplora tutte le combinazioni di iperparametri specificate nella "griglia" per trovare quella che ottimizza la performance del modello considerato*

La K-Fold Cross Validation (K=5) allena il modello e valuta ciascuna combinazione, restituendo dati sulle performance generali e la media delle performance sui fold

K-Fold CV

- *I dati vengono suddivisi in k sottoinsiemi (folds)*
- *Per ogni iterazione, un fold viene utilizzato per la validazione, mentre i restanti k-1 folds servono per l'addestramento*
- *Questo processo si ripete k volte, e alla fine si calcola la media delle metriche per stimare le prestazioni del modello*

Grid Search

- *È una ricerca esaustiva di combinazioni di iperparametri predefiniti*
- *Prova tutte le possibili configurazioni e sceglie quella con la miglior performance (in genere basata sull'accuracy o un'altra metrica)*
- *Ogni combinazione di iperparametri viene valutata usando Cross-Validation k-Fold per garantire che il modello sia robusto*

Metriche di Valutazione e Valori Restituiti

Le metriche per la valutazione in questo caso sono:

Accuracy: È la percentuale di previsioni corrette sul totale delle osservazioni

F1-score Macro: L'F1-score è una metrica che bilancia Precision e Recall attraverso la loro media armonica.
La variante Macro calcola l'F1-score per ogni classe e poi ne fa la media non pesata

Precision Macro: La Precision misura quanti degli esempi predetti come positivi sono effettivamente positivi

Recall Macro: Misura quanti dei veri esempi positivi sono stati effettivamente identificati

Alberi Decisionali

	Accuracy	*F1-score Macro*	*Precision Macro*	*Recall Macro*
1	*0.7300*	*0.7287*	*0.7321*	*0.7289*
2	*0.8200*	*0.8193*	*0.8222*	*0.8191*
3	*0.7000*	*0.7000*	*0.7003*	*0.7003*
4	*0.6800*	*0.6799*	*0.6803*	*0.6800*
5	*0.6300*	*0.6297*	*0.6305*	*0.6300*

1 = {'criterion': 'gini', 'max_depth': 10, 'min_samples_leaf': 5, 'min_samples_split': 2}

Fold 1: Accuracy = 0.7700

Fold 2: Accuracy = 0.8300

Fold 3: Accuracy = 0.8800

Fold 4: Accuracy = 0.7200

Fold 5: Accuracy = 0.8300

2 = {'criterion': 'entropy', 'max_depth': 10, 'min_samples_leaf': 10, 'min_samples_split': 2}

Fold 1: Accuracy = 0.6800

Fold 2: Accuracy = 0.8200

Fold 3: Accuracy = 0.7800

Fold 4: Accuracy = 0.6700

Fold 5: Accuracy = 0.7700

3 = {'criterion': 'gini', 'max_depth': 15, 'min_samples_leaf': 2, 'min_samples_split': 2}

Fold 1: Accuracy = 0.7300

Fold 2: Accuracy = 0.7400

Fold 3: Accuracy = 0.7000

Fold 4: Accuracy = 0.7100

Fold 5: Accuracy = 0.7100

4 = {'criterion': 'gini', 'max_depth': 10, 'min_samples_leaf': 4, 'min_samples_split': 2}

Fold 1: Accuracy = 0.7400

Fold 2: Accuracy = 0.7500

Fold 3: Accuracy = 0.7300

Fold 4: Accuracy = 0.6800

Fold 5: Accuracy = 0.7200

5 = {'criterion': 'gini', 'max_depth': 15, 'min_samples_leaf': 1, 'min_samples_split': 5}

Fold 1: Accuracy = 0.6700

Fold 2: Accuracy = 0.7700

Fold 3: Accuracy = 0.6700

Fold 4: Accuracy = 0.6800

Fold 5: Accuracy = 0.6300

Random Forest

	Accuracy	F1-score Macro	Precision Macro	Recall Macro
1	0.7700	0.7698	0.7700	0.7697
2	0.8200	0.8199	0.8199	0.8199
3	0.8400	0.8394	0.8424	0.8391
4	0.7200	0.7196	0.7214	0.7200
5	0.8400	0.8399	0.8405	0.8400

1 = {'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_split': 10, 'n_estimators': 100}

Fold 1: Accuracy = 0.7700

Fold 2: Accuracy = 0.8300

Fold 3: Accuracy = 0.8800

Fold 4: Accuracy = 0.7200

Fold 5: Accuracy = 0.8300

2 = {'max_depth': 15, 'min_samples_leaf': 1, 'min_samples_split': 10, 'n_estimators': 100}

Fold 1: Accuracy = 0.7500

Fold 2: Accuracy = 0.8200

Fold 3: Accuracy = 0.9000

Fold 4: Accuracy = 0.7300

Fold 5: Accuracy = 0.8300

3 = {'max_depth': 15, 'min_samples_leaf': 2, 'min_samples_split': 5, 'n_estimators': 50}

Fold 1: Accuracy = 0.7700

Fold 2: Accuracy = 0.8300

Fold 3: Accuracy = 0.8400

Fold 4: Accuracy = 0.7200

Fold 5: Accuracy = 0.8500

4 = {'max_depth': 15, 'min_samples_leaf': 2, 'min_samples_split': 5, 'n_estimators': 200}

Fold 1: Accuracy = 0.7400

Fold 2: Accuracy = 0.8400

Fold 3: Accuracy = 0.8500

Fold 4: Accuracy = 0.7200

Fold 5: Accuracy = 0.8500

5 = {'max_depth': 15, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}

Fold 1: Accuracy = 0.7800

Fold 2: Accuracy = 0.8300

Fold 3: Accuracy = 0.8600

Fold 4: Accuracy = 0.7200

Fold 5: Accuracy = 0.8400

Regressione Logistica

	Accuracy	F1-score Macro	Precision Macro	Recall Macro
1	0.8000	0.7999	0.8017	0.8007
2	0.8600	0.8591	0.8658	0.8587
3	0.8600	0.8595	0.8626	0.8591
4	0.7100	0.7097	0.7108	0.7100
5	0.8200	0.8197	0.8221	0.8200

1 = {'C': 5, 'penalty': 'l2', 'solver': 'saga'}

Fold 1: Accuracy = 0.8000

Fold 2: Accuracy = 0.8600

Fold 3: Accuracy = 0.8600

Fold 4: Accuracy = 0.7100

Fold 5: Accuracy = 0.8200

2 = {'C': 5, 'penalty': 'l2', 'solver': 'liblinear'}

Fold 1: Accuracy = 0.8000

Fold 2: Accuracy = 0.8600

Fold 3: Accuracy = 0.8500

Fold 4: Accuracy = 0.7100

Fold 5: Accuracy = 0.8200

3 = {'C': 10, 'penalty': 'l2', 'solver': 'liblinear'}

Fold 1: Accuracy = 0.7900

Fold 2: Accuracy = 0.8400

Fold 3: Accuracy = 0.8600

Fold 4: Accuracy = 0.7200

Fold 5: Accuracy = 0.8000

4 = {'C': 5, 'penalty': 'l2', 'solver': 'liblinear'}

Fold 1: Accuracy = 0.8000

Fold 2: Accuracy = 0.8600

Fold 3: Accuracy = 0.8500

Fold 4: Accuracy = 0.7100

Fold 5: Accuracy = 0.8200

5 = {'C': 5, 'penalty': 'l2', 'solver': 'liblinear'}

Fold 1: Accuracy = 0.8000

Fold 2: Accuracy = 0.8600

Fold 3: Accuracy = 0.8500

Fold 4: Accuracy = 0.7100

Fold 5: Accuracy = 0.8200

Deviazione standard

Alberi Decisionali *Tra 0.0147 e 0.0589*

Random Forest *Tra 0.0496 e 0.0609*

Regressione Logistica *Tra 0.0483 e 0.0551*

Analisi dei Risultati

Alberi Decisionali

Nel caso degli alberi decisionali la miglior combinazione di iperparametri sembra essere *{'C': 5, 'penalty': 'l2', 'solver': 'liblinear'}*

Accuracy media: Variabile tra 0.63 e 0.82, con oscillazioni dovute alla scelta degli iperparametri

F1-score Macro: In linea con l'accuracy, indicando che il modello non sbilancia eccessivamente precision e recall

Deviazione standard: Tra 0.0147 e 0.0589, segno che il modello è sensibile ai dati di training

Conclusioni:

Il Decision Tree ha prestazioni instabili tra le varie configurazioni, mostrando variazioni anche significative tra i fold in più
il modello soffre probabilmente di overfitting, specialmente con profondità elevate ('max_depth') per non parlare dell'accuracy, la peggiora tra i 3 modelli

Random Forest

Sicuramente nel caso del modello random forest, i valori risultano ovviamente essere più solidi rispetto ad utilizzare un solo albero decisionale, la combinazione di iperparametri che genera i migliori valori sembra essere

{'max_depth': 15, 'min_samples_leaf': 2, 'min_samples_split': 5, 'n_estimators': 50}

Accuracy media: Tra 0.72 e 0.84, risultando uno dei migliori modelli.

F1-score Macro: Molto vicino all'accuracy, suggerendo un buon bilanciamento

Deviazione standard: Tra 0.0496 e 0.0609, mostrando una discreta robustezza

Conclusioni:

Il Random Forest è sicuramente uno dei modelli più stabili e robusti tra quelli testati in quanto è meno incline all'overfitting rispetto al Decision Tree, in più l'uso di più alberi migliora la generalizzazione.

L'unica pecca è il costo computazionale, il più costoso tra i 3

Regressione Logistica

Insieme al modello random forest risulta essere tra i modelli più solidi, ancha grazie al fatto di aver ottenuto la migliore accuracy tra i 3, migliori iperparametri

{'C': 5, 'penalty': 'l2', 'solver': 'liblinear'} *(Anche con C = 10)*

Accuracy media: 0.80-0.86, la più alta tra i modelli testati

F1-score Macro: Molto vicino all'accuracy, indicando un buon bilanciamento

Deviazione standard: Tra 0.0483 e 0.0551, segno di stabilità

Conclusioni:

La Logistic Regression ha ottenuto le migliori performance in termini di accuracy generale ed è il modello più semplice e interpretabile

È meno flessibile di un random forest ma meno incline all'overfitting rispetto al decision tree.

Valutazione Finale

Il modello di regressione logistica e il modello random forest risultano essere i più adatti in questo contesto

Random Forest premia stabilità e generalizzazione, evitando problemi di overfitting

Regressione Logistica premia invece e la semplicità e l'interpretabilità con in più la migliore accuracy

A livello di potenza predittiva e alla luce dei dati mostrati, con l'unica pecca del costo computazionale, il modello migliore risulta essere il Random Forest (Seguito dal modello Logistic Regression)

Capitolo 4 – Classificazione Probabilistica

Il Naive Bayes è un algoritmo di classificazione basato sul teorema di Bayes, con l'assunzione che le caratteristiche siano indipendenti l'una dall'altra (da cui il termine "naive", cioè ingenuo). È uno degli algoritmi

più semplici e rapidi, ma nonostante la sua semplicità, è molto efficace in molti casi

I risultato ottenuti sono i seguenti:

Accuracy: 0.7600				
	precision	*recall*	*F1-score*	*support*
0	*0.71*	*0.88*	*0.79*	*51*
1	*0.84*	*0.63*	*072*	*49*
accuracy			*0.76*	*100*
macro avg	*0.78*	*0.76*	*0.76*	*100*
weighted avg	*0.77*	*0.76*	*0.76*	*100*

L'accuratezza complessiva è **76%**, *il che significa che il modello classifica correttamente il risultato della partita (win/loss) nel 76% dei casi*

Classe 0:

Precisione 0.71 = Quando il modello predice una sconfitta, ha il 71% di probabilità di aver ragione

Recall 0.88 = L'88% delle partite effettivamente perse sono state classificate correttamente come perse

F1-score 0.79 = Media armonica tra precision e recall, indica un buon bilanciamento tra falsi positivi e falsi negativi

Support 51 = Il dataset contiene 51 partite perse nel test set

Classe 1:
Precisione 0.84 = Quando il modello predice una vittoria, ha l'84% di probabilità di aver ragione

Recall 0.63 = Solo il 63% delle partite vinte sono state classificate correttamente

F1-score 0.72 = Indica che il modello ha difficoltà nel riconoscere tutte le partite vinte rispetto alle perse

Support 49 = Il dataset contiene 49 partite vinte nel test set

Macro Avg = 0.78 (precision), 0.76 (recall), 0.76 (f1-score)

(media aritmetica tra le metriche delle due classi dando quindi uguale peso per entrambe)

Weighted Avg = 0.77 (precision), 0.76 (recall), 0.76 (f1-score)

(Una media ponderata che tiene conto del numero di istanze per classe, valore 'support')

Il problema principale risulta essere il basso valore di recall per la classe 1(0.63), quella delle vittorie.
Il modello fatica a riconoscere tutti i casi di vittoria

Capitolo 5 – Sviluppi Futuri

Con adeguati miglioramenti e adattamenti il progetto potrebbe essere utilizzato nell'ambito dell'e-sport del gioco strategico League of Legends da parte dei coach delle diverse squadre per monitorare, magari in tempo reale durante le partite, le varie statistiche attuali e avvenimenti all'interno della partita per adeguarsi di conseguenza e proporre le scelte migliori al team

Capitolo 6 – Riferimenti

Dataset e Features: Riot Developer Portal