Teambuilder Pokemon

Gruppo di lavoro

- Francesco_Laporta, 789422, f.laporta14@studenti.uniba.it

https://github.com/Fvvancesco/Teambuilder

AA 2024-25

Introduzione

Il dominio scelto è quello del mondo pokèmon un videogioco di lotta a turni uscito verso la fine degli anni '90. Il motivo per il quale ho scelto questo dominio, oltre che per interesse personale, è perché si presta molto bene a un progetto di una materia come Ingegneria della conoscenza: dopo solo il primo turno le combinazioni di risultati possibili sono stimate essere oltre 10^350; questo non solo rende il gioco interessante e imprevedibile all'utente medio ma costituisce anche un enorme spazio di ricerca da esplorare tramite le metodologie viste durante il corso.
La popolarità della proprietà intellettuale unita a questa sua caratteristica ha portato alla disponibilità di un certo numero di strumenti e collezioni di dati che hanno reso possibile concentrarsi prevalentemente sulla realizzazione del progetto piuttosto che sulla fase di ottenimento e un eventuale preprocessing invasivo (sebbene vi sia stato) dei dati.

Breve spiegazione del funzionamento del gioco e del dominio specifico e delle regole di dominio

La parte che sono andato ad analizzare nello specifico è quella della lotta, in particolare la lotta in singolo che è il formato più popolare e il secondo formato più competitivo del dominio considerato.

Prima di cominciare la lotta i giocatori devono avere una squadra composta da al più ma generalmente non meno di 6 membri e la lotta in singolo inizia quando entrambi i giocatori fanno scendere in campo il loro "lead" cioè il primo pokèmon che hanno intenzione di utilizzare.

I giocatori selezionano la mossa desiderata considerando tutte le variabili possibili (in particolare oltre alla loro situazione complessiva valutano anche il lead o il pokèmon scelto dal rivale o gli eventi dei turni precedente). Gli aspetti cruciali del gioco, prima dell'inizio della partita, sono tutti legati esclusivamente alle caratteristiche che può assumere il singolo pokémon e per estensione la squadra.
Un pokémon infatti, durante una lotta, è definito da questi fattori:

- Statistiche (Salute HP, Attacco fisico ATK, Difesa fisica DEF, Attacco speciale SPATK, Difesa speciale SPDEF e Velocità SPD)
- Tipi: a ciascuna specie, è assegnato un tipo o due tipi (Elementali come acqua fuoco… O caratteristiche peculiari come "Volante" e via discorrendo). Determinano le debolezze e resistenze più evidenti del pokémon. Sistema sasso carta forbice, fuoco batte erba che batte acqua.
- Abilità: ogni pokémon ne ha una in battaglia ma ogni specie ne ha fino a tre
- Mosse: ogni pokémon ne può avere fino a quattro e ogni specie ne può imparare un insieme predeterminato, a ciascuna mossa è assegnato un tipo, gli stessi che possono essere assegnati al pokémon e permettono di fare danno o possono essere di status, cambiando la situazione del gioco. Ogni mossa ha un valore di danno e di precisione
- Oggetti: qualunque pokémon può equipaggiare qualunque oggetto, ciascuno ha effetti diversi.

Queste sono solo alcune delle caratteristiche e meccaniche che governano il dominio scelto, la lista completa da sola potrebbe occupare tutto il documento pertanto ho deciso di concentrarmi su quelle fondamentali.

Sommario

Il progetto verte sulla costruzione di uno strumento che sia in grado di aiutare un giocatore alle prime armi nella creazione della sua prima squadra pokémon

Elenco argomenti di interesse

- Clustering per la scoperta dei ruoli (Cap: 10)
- Apprendimento supervisionato per assegnare punteggi individuali (Cap 7)
- Teambuilding come CSP: Greedy ascent (Cap 3)

Clustering per la scoperta dei ruoli

Sommario

Come spiegato, una delle parti fondamentali nella costruzione di una squadra Pokémon sono le caratteristiche dei singoli mostri tascabili. Le più importanti sono le statistiche base, che determinano tre fattori chiave: forza complessiva, orientamento (offensivo o difensivo) e specializzazione. Basandosi su queste metriche, nel panorama competitivo si formano generalmente 5 o 6 ruoli principali:

- Sweeper fisici e speciali
- Wall fisici e speciali
- Tank e support

Poiché non esiste un dataset classificato (vista una certa soggettività nelle classificazioni e nei ruoli individuati nonostante una certa unanimità nelle etichette) ho scelto di basare l'individuazione dei ruoli dei Pokémon un modello di apprendimento non supervisionato, in modo da far emergere queste categorie matematicamente dai dati grezzi.

Strumenti utilizzati

- Pandas e NumPy: Utilizzati per il caricamento, la manipolazione del dataset CSV strutturato e la gestione degli array vettoriali.
- Scikit-learn: Utilizzata per l'implementazione della pipeline di apprendimento. Nello specifico, è stata utilizzata un'ampia gamma di Scaler (`StandardScaler`, `MinMaxScaler`, `RobustScaler`, `MaxAbsScaler`) nella fase di preprocessing , e l'algoritmo `KMeans` nella fase di clustering. È stata inoltre impiegata la PCA per la riduzione della dimensionalità a scopo visivo.
- Matplotlib: Impiegata per la visualizzazione grafica delle metriche di valutazione, fondamentale per l'analisi eversiva della dimensionalità e la scelta degli iperparametri.

Decisioni di Progetto

- Vista l'assenza di dati classificati e conoscenza certa delle classi ho deciso di seguire un approccio empirico per raffinamenti successivi. Partendo da un semplice clustering per poi eseguire una PCA per decidere come proseguire la ricerca dei valori ottimali, ho successivamente impostato un'ambiente di testing che mi permettesse di eseguire sia esperimenti mirati tramite una pipeline di aggregazione, scaling e plotting così da scoprire e decidere i valori ottimali. Infine ho implementato la grid search per comparare i risultati scelti con eventuali risultati migliori ottenuti

Valutazione

Il primo passaggio è decidere come trattare le statistiche.

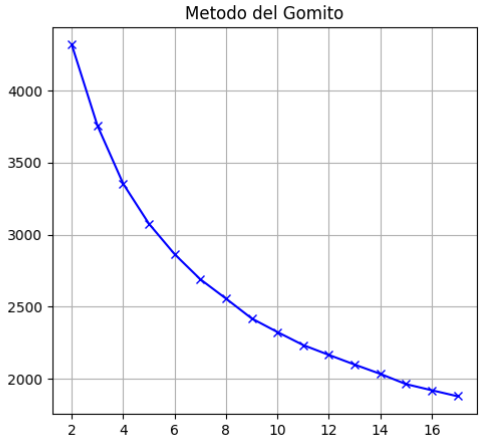
Analizziamo i problemi principali legati al clustering e al nostro dataset:

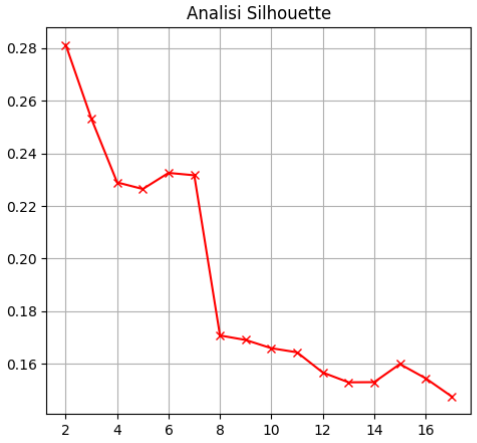
- Outliers: esistono pokémon con caratteristiche molto particolari come shedinja con 1 solo punto hp oppure pokémon il cui ruolo e le statistiche cambiano drasticamente durante la partita (aegislash che scambia attacco e difesa ricoprendo entrambi i ruoli)
- Distribuzione dei pokémon: non tutti i pokémon hanno una distribuzione delle statistiche pensata per il gioco competitivo, molti sono semplicemente bilanciati; questa situazione porta facilmente a pessimi risultati nell'apprendimento non supervisionato come k-means che può far fatica a trovare le etichette
- Curse of dimensionality:

Il primo problema ci suggerisce di utilizzare un algoritmo di clustering che gestisca bene gli outlier (come dbscan) tuttavia ho deciso di dargli poco peso poiché shedinja è il principale caso che potrebbe dare problemi a un normale algoritmo di clustering, al più lo potremmo rimuovere dal dataset.

Il secondo problema invece è stato trattato, nelle fasi più avanzate, con l'assunzione che pokemon con statistiche più alte tendano a specializzarsi in alcune aree e pertanto è stato applicato un filtro prima di cominciare l'addestramento.

Il primo approccio, quello "naive" prevedeva l'utilizzo delle 6 statistiche pure normalizzate tramite Z-Score. Poiché il K-Means si basa sulla distanza euclidea, la normalizzazione era strettamente necessaria per evitare che feature con un range più ampio (come gli HP) dominassero l'algoritmo. Il K-Means generalmente sfrutta un k prefissato. Invece in questo caso ho utilizzato le metriche ricavate dal metodo del gomito e l'analisi della silhouette per determinare un k ottimale.



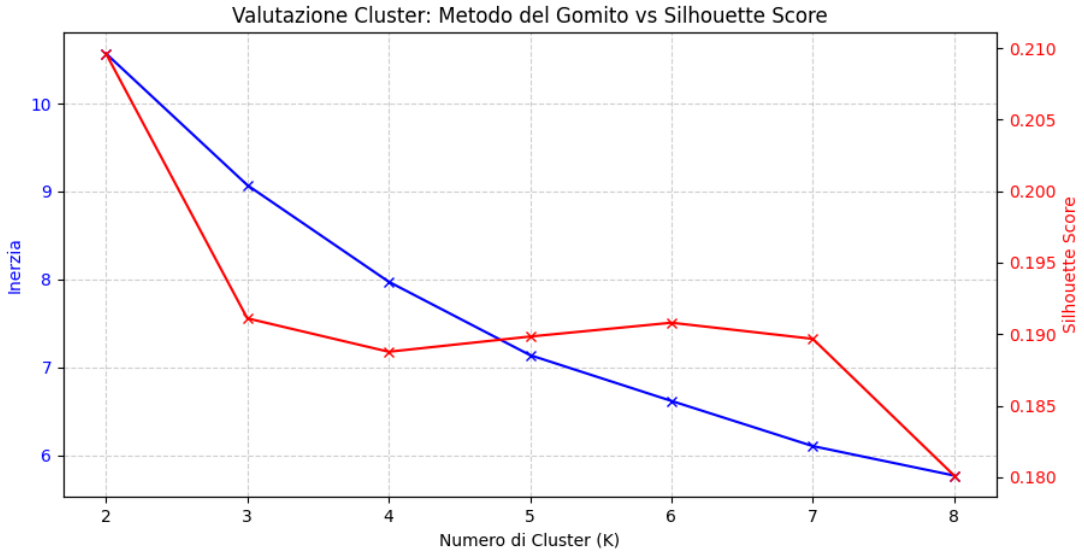


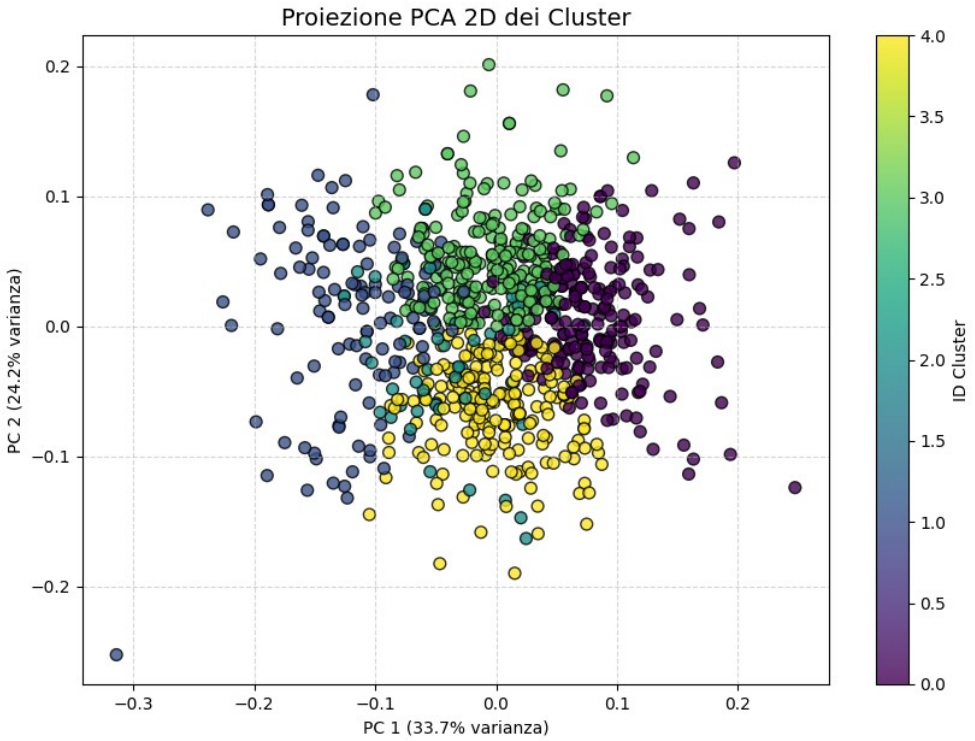
I valori più promettenti sono quelli inferiori a 7.
Tuttavia sia il metodo del gomito che la silhouette rivelano un clustering di basso livello, poiché nel primo caso la curva è molto dolce e quindi non c'è un valore di clustering netto che suggerisca un vero e proprio cutoff, nel secondo caso invece il valore, anche di picco è solo 0.28 che indica una correlazione debole tra le classi ottenute.

Per migliorare questo risultato ho iniziato un'analisi dimensionale, prima automatizzata e dopo manuale.

Nel primo caso ho chiesto al modello di scomporre in due dimensioni tramite una combinazione lineare delle statistiche, questo approccio ha rivelato alcune informazioni che già immaginavamo come una correlazione negativa tra difesa e velocità e piccole altre relazioni di poco conto.

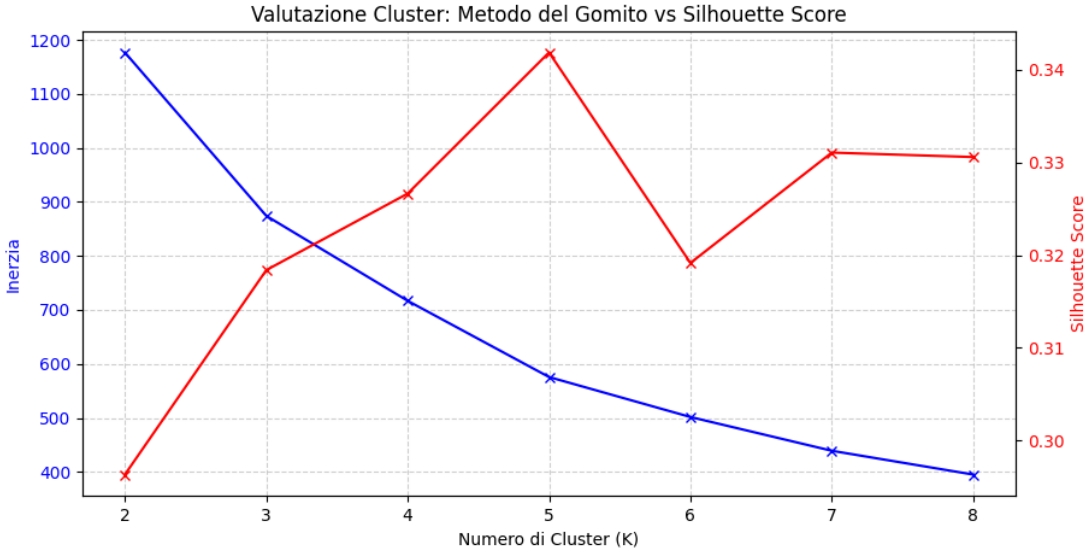
Tuttavia il plot dell'analisi automatica ha rivelato la presenza di un outlier non considerato, il punto in alto a sinistra che è shuckle. Questi outlier possono in alcuni casi spostare il centroide e peggiorare il clustering pertanto successivamente verrà rimosso. Tolte queste informazioni la PCA automatica non ha riportato un buon clustering

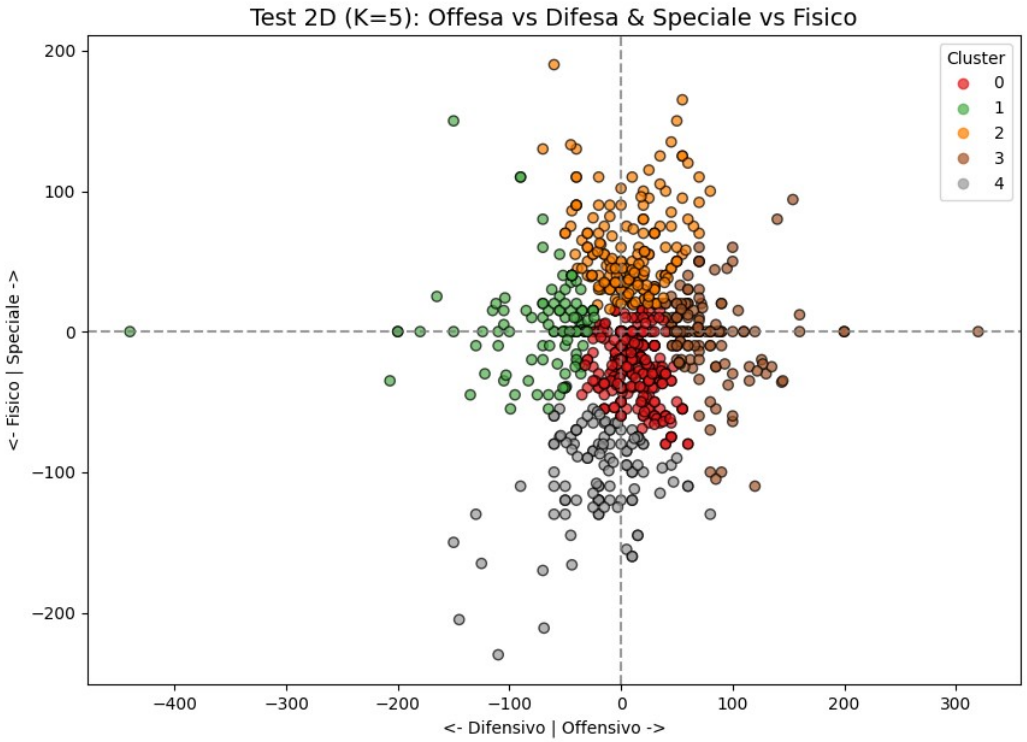




Nel caso manuale invece ho utilizzato l'intuizione legata alla BK cioè la suddivisione tra difesa e attacco e specializzazione: cioè fisico e speciale (è importante notare che questa strategia omette alcune delle classi più note come i Tank)

Scomponiamo manualmente in componenti offensive, difensive, fisiche e speciali



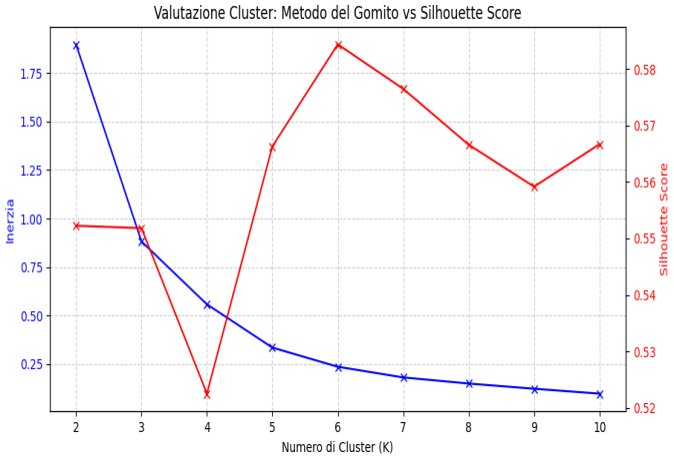


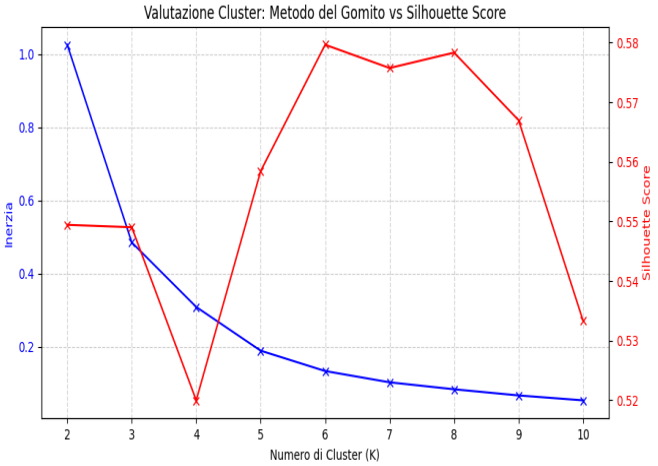
Dall'analisi PCA notiamo che c'è un cluster particolarmente addensato al centro che sarebbero i pokémon di cui abbiamo parlato prima, quelli bilanciati e non competitive oriented.

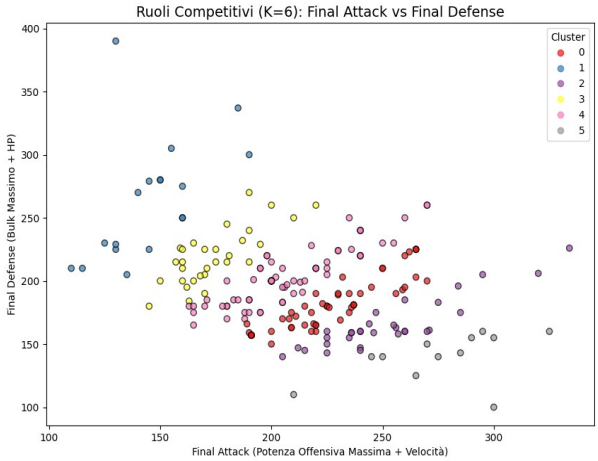
Sorprende vedere quegli outlier, che sono shuckle e Deoxys forma attacco di cui non avevamo considerato la disparità tra attacco e difesa.

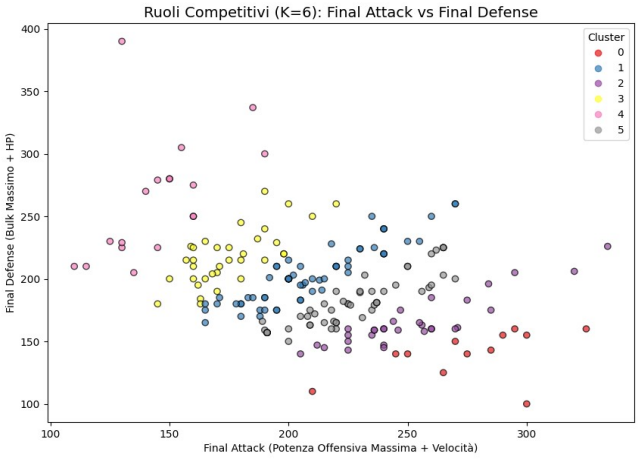
Ho rieseguito i test rimuovendo gli outlier ma il punteggio della silhouette non è aumentato di molto, nonostante ciò li rimuoveremo per tutti itest successivi. Pertanto passiamo al prossimo approccio, cioè sfruttiamo la conoscenza nota (tipi più usati nel meta) per indirizzare la ricerca.

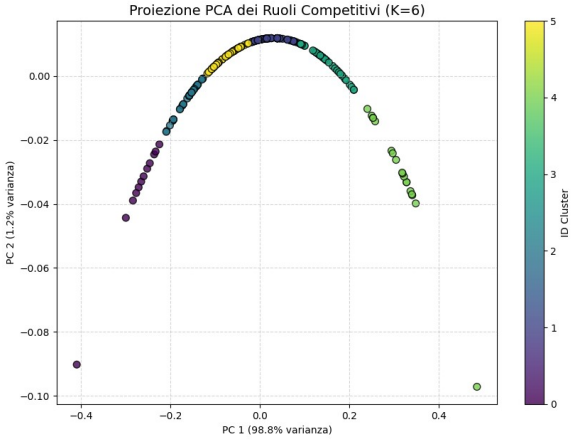
Seguendo le informazioni che sappiamo sul meta, quindi la suddivisione in Sweeper... e modificando la classe `PokemonRoleAnalyzer` e creo un sistema di pipeline (scelta delle feature, eventuale aggregazione e infine scaling tramite formato pseudo-json) e testing automatico del dataset. Dopo diversi esperimenti trovo due combinazioni che meglio catturano l'essenza della suddivisione nota al competitivo.

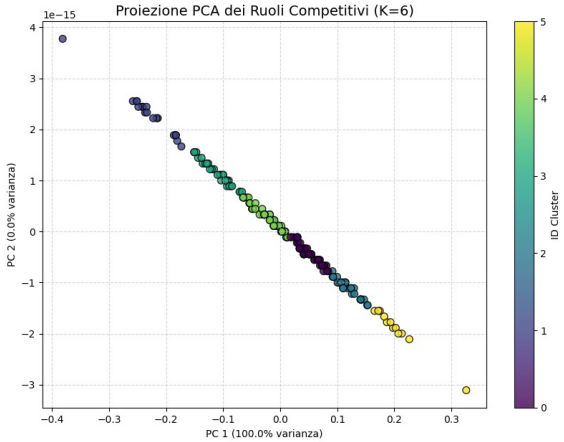












È interessante notare alcune cose: Il picco per entrambi è attorno a 6 che sono sia le categorie che abbiamo trovato che i membri di una squadra che le statistiche, questo ci semplifica il compito successivamente per la costruzione del team.

L'aggregazione delle statistiche è la seguente: Final atk = Max(Atk, Atk speciale) + Vel. Simmetricamente per difese e hp. Infine nel primo caso usiamo una regolarizzazione (o meglio normalizzazione) come L2 quindi somma dei quadrati, dall'altra parte invece utilizziamo il rapporto. Quello che succede nel secondo caso è evidente essendo le statistiche complementari una sola è sufficiente a spiegare l'intero dataset.

Dimostra che c'è una buona correlazione tra il bilancio offesa e difesa come sospettavamo! Sarebbe interessante trovare un'alternativa a L2 che dia più carattere alla varianza.

Grid Search

Per assicurarmi che non ci siano risultati migliori ho applicato una grid-search che mi ha fatto ottenere ottimi coefficienti di clustering (silhouette) ma l'ha fatto concentrandosi su quella che era l'astrazione della potenza (tramite media delle statistiche totali) del pokemon piuttosto che il ruolo in un team.

Questo dimostra che i pokémon hanno tier di potenza diversi e che sono ben classificabili da questi. (Probabilmente è legato alla quantizzazione dei dati dovuta al meccanismo di evoluzione nel gioco esempio: un pokémon ha 3 forme, un altro 2, le forme lasciano grandi spazi di statistiche totali tra loro permettendo una facile classificazione; c'è però da considerare che questi dati sono stati ricavati considerati statistiche totali > 500)

Apprendimento supervisionato per assegnare punteggi individuali

Sommario

Nella prima parte abbiamo parlato del teambuilding dal punto di vista della suddivisione dei ruoli per creare un team che possa gestire più minacce durante la ricerca abbiamo notato come il dataset si prestasse bene, in relazione esclusivamente alle statistiche a una suddivisione gerarchica.

Come abbiamo già detto però la modalità della lotta si presta bene a ulteriori complicanze derivanti da interazioni tra tipi, forza sistematica dei pokémon, abilità e altro ancora…
Poiché l'obbiettivo di questo progetto è costruire un teambuilder per le lotte 6v6 in singolo ho utilizzato il seguente dataset:

https://www.kaggle.com/datasets/terminus7/pokemon-challenge

Che funge da base di conoscenza sui pokémon e ci permette di applicare la tecnica dell'apprendimento supervisionato per studiare la forza individuale tramite una lotta 1v1 in singolo contro altri pokèmon.

Il dataset è composto nel seguente modo:
Feature sono: chi ha attaccato per primo e chi per secondo al primo turno

Target: vincitore

Sfruttiamo questa informazione per ricavare la forza intriseca dei pokémon non solo legata alle statistiche ma anche ad altri fattori come tipi, molto importanti, e se sono o meno leggendari.

Strumenti utilizzati

LightGBM (`LGBMRanker`): Scelto per la sua efficienza, il suo supporto nativo alle funzioni obiettivo di ranking (tramite *lambdarank* e ottimizzazione della metrica NDCG) e alla sua compatibilità nativa agli istogrammi che permettono di gestire in modo naturale il typing dei pokémon, a differanza di un pesante 1hot enconding.

Scikit-learn: Utilizzato per la validazione incrociata (`GroupKFold`) e per la ricerca stocastica degli iperparametri (`ParameterSampler`).

Pandas e NumPy: Impiegati per la manipolazione vettoriale dei dati, la gestione degli array e la creazione della pipeline di feature engineering.

Decisioni di Progetto

Ho deciso di includere solamente, rispetto al dataset originario, una feature "is_first" che è legata all'ordine di esecuzinoe della mossa, tutte le altre feature che avremmo potuto ottenere espandendo il dataset originario sono state ignorate dati due fattori:

dimensione del dataset, non sufficientemente grande per includere, mantenendo un alto livello, tutte le feature

la specificità della situazione analizzata: 1v1 invece di lotta in doppio dove le interazioni esplodono a livello combinatoriale e fattori come le abilità introdurrebbero solo rumore.

In futuro con un dataset più ampio come pokéchamp o smogon sarà possibile espandere il progetto.

Per mitigare eventuali data leakage e overfitting o metriche sballate ho implementato la Kfold Cross Validation: separa in k partizioni e allena il modello considerando k-1 partizioni come di training e una di test o validation.

Il tuning degli iperparametri è stato realizzato con una random search, implementata personalmente perché **Scikit-learn** entrava in conflitto con LightGBM, per evitare un calcolo computazionale elevato come con la gridsearch.

Valutazione

Per valutare un modello di Learning-to-Rank a gruppi di dimensione 2, le metriche classiche di accuratezza non sono adeguate. Ho quindi implementato una metrica personalizzata di Pairwise Accuracy (Accuratezza a coppie) per la validazione incrociata. Questa metrica valuta semplicemente se la differenza tra lo score predetto per il primo Pokémon e quello del secondo ha lo stesso segno (similarmente a una loss tradizionale) della differenza reale (chi ha vinto effettivamente). Parallelamente, durante l'addestramento, l'algoritmo ottimizzava internamente l'NDCG (Normalized Discounted Cumulative Gain metrica studiata durante MRI per i valutare proprio i ranking), che penalizza pesantemente gli errori di ordinamento.

Sfruttando le capacità dei Gradient Boosted Trees, ho calcolato l'importanza delle feature basata sul numero di split (quante volte una feature è stata usata per dividere i dati nei nodi dell'albero). I risultati confermano in modo sorprendente la Background Knowledge del metagioco competitivo:

- La Velocità (Speed) domina in modo assoluto le decisioni del modello (1868 split). Nelle lotte 1v1, colpire per primi è il fattore più critico, poiché permette di eliminare l'avversario prima di subire danni.
- Seguono i Tipi (Type 2: 434, Type 1: 353), che confermano l'importanza delle sinergie elementali e dei moltiplicatori di danno (debolezze e resistenze).
- Le statistiche di Bulk (HP: 333) e forza offensiva (Attack: 324), unite al vantaggio di turno (`is_first`: 293), formano il nucleo secondario delle decisioni.

```
========================================
FEATURE IMPORTANCE (Split)
========================================
- Speed          : 1868
- Type 2         : 434
- Type 1         : 353
- HP             : 333
- Attack         : 324
- is_first       : 293
- Sp. Atk        : 246
- Defense        : 147
- Sp. Def        : 143
- Generation     : 41
- Legendary      : 18
```

Estrazione della Classifica Generale (Power Ranking) Calcolando lo Strength Score globale (ottenuto mediando i punteggi predetti di ciascun Pokémon sia come primo che come secondo attaccante), è stata stilata la classifica assoluta della forza intrinseca.

- I 5 Pokémon più forti: La vetta della classifica (guidata da Deoxys Speed Forme con uno score di 5.2853, seguito da Ninjask, Mega Aerodactyl, Mega Lopunny e Mega Beedrill) è interamente occupata da Pokémon che eccellono in un attributo specifico: la Velocità estrema unita a un alto potenziale offensivo. Questo output è perfettamente coerente con la Feature Importance calcolata in precedenza e identifica i cosiddetti "Sweeper" o "Glass Cannon" (Pokémon con altissima capacità offensiva ma altrettanto fragili).

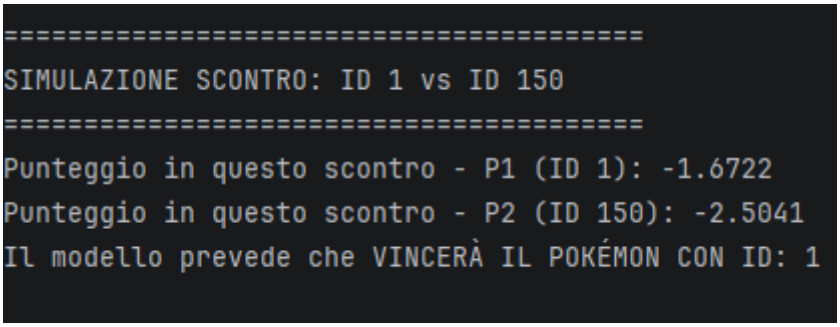
```
========================================
ANALISI CLASSIFICA GENERALE
========================================

I 5 Pokémon più forti in assoluto:
POS   | ID   | NOME                      | STRENGTH SCORE
-------------------------------------------------------------
1     | 432  | Deoxys Speed Forme        | 5.2853
2     | 316  | Ninjask                   | 5.1740
3     | 155  | Mega Aerodactyl           | 5.1373
4     | 477  | Mega Lopunny              | 5.1249
5     | 20   | Mega Beedrill             | 5.0758
```

- I 5 Pokémon più deboli: Nelle posizioni più basse troviamo stadi base o "baby" Pokémon (Cleffa, Igglybuff, Wooper, Magikarp). All'ultimo posto assoluto si posiziona Shuckle (-5.1304): nonostante possieda le difese più alte del gioco, la sua Velocità e il suo Attacco quasi nulli lo rendono matematicamente incapace di vincere uno scontro 1v1 diretto, dimostrando che il modello ha compreso le penalità di un design eccessivamente passivo.

```
I 5 Pokémon più deboli in assoluto:
POS   | ID   | NOME                      | STRENGTH SCORE
-------------------------------------------------------------
1     | 188  | Cleffa                    | -4.1598
2     | 189  | Igglybuff                 | -4.1953
3     | 210  | Wooper                    | -4.2477
4     | 140  | Magikarp                  | -4.2519
5     | 231  | Shuckle                   | -5.1304
```

Simulazione Empirica degli Scontri Per dimostrare l'applicabilità del modello all'interno del futuro Teambuilder, è stata implementata una funzione di simulazione predittiva. Interrogando il modello con gli ID di due Pokémon specifici (nel test: ID 1 vs ID 150), la pipeline ricostruisce le feature di

entrambi, simula i due scenari di vantaggio di turno e restituisce lo score relativo. Nel test effettuato, lo scontro si è risolto con uno score di -1.6722 per il Pokémon 1 e -2.5041 per il Pokémon 2, portando il modello a prevedere la vittoria del Pokémon 1. Questa simulazione empirica fornisce lo strumento valutativo algoritmico ("funzione di utilità") che verrà sfruttato dal Constraint Satisfaction Problem (CSP) nella fase successiva per ottimizzare e selezionare i membri della squadra finale.



Teambuilding come CSP: Greedy ascent (Cap 3)

Sommario

Dopo aver ottenuto un modello in grado di determinare la forza di un pokémon e una classificazione dei loro ruoli possiamo finalmente unirli in un unico progetto: la costruzione di un team equilibrato per affrontare il gioco competitivo. Per fare ciò ho trasformato questo problema in un problema di CSP con vincoli hard (6 membri) e soft cioè la massimizzazione di una funzione.

L'algoritmo che ho scelto per la risoluzione di questo problema è l'algoritmo di Hill Climbing o Greedy Ascent per la sua semplicità nell'implementazione

Strumenti utilizzati

Pandas e librerie standard di Python: Li ho usati per unire i dati dei tre step precedenti (statistiche, ruoli del K-Means e punteggi del LightGBM caricandoli da file) e per gestire le liste e i dizionari durante la ricerca.

Algoritmo di Ricerca Locale: Ho scritto una mia implementazione dell'algoritmo Hill Climbing con "Random Restarts". È una tecnica standard dell'Intelligenza Artificiale, perfetta per i casi in cui non si conosce lo stato finale ma si può calcolare quanto sia "buona" una certa mossa.

Decisioni di Progetto

Spazio degli Stati e Vincoli Hard: Uno stato valido è semplicemente una squadra di 6 Pokémon tutti diversi tra loro (non si possono usare doppioni). L'algoritmo genera e valuta solo squadre che rispettano questa regola fissa.

All'utente inoltre è data la possibilità di scegliere pokèmon predefiniti così da poter giocare con

il team che preferisce.

Funzione Obiettivo (Vincoli Soft): Questa è la funzione che l'algoritmo cerca di massimizzare ed è il vero cuore del Teambuilder. Valuta quanto è forte una squadra bilanciando tre cose:

Forza Bruta: È la somma dei punteggi di forza calcolati dal modello LightGBM. Mi assicura di inserire Pokémon che sono validi in lotta 1v1.

Sinergia di Ruolo (Bonus Diversità): Ho aggiunto un bonus positivo per ogni ruolo diverso presente nel team, basandomi sui cluster trovati col K-Means. In questo modo "costringo" l'algoritmo a non scegliere solo attaccanti veloci, ma a creare un team sensato (ad esempio, inserendo anche dei difensori). Per aggiungere varietà ho utilizzato una funzione di distanza che premia se i pokémon sono molto diversi tra loro per rendere più resistente il team una funzione di punizione che favorisce tipi distinti.

Esplorazione e Random Restarts: Per passare da uno stato all'altro, l'algoritmo prende un Pokémon della squadra e prova a scambiarlo con tutti gli altri Pokémon disponibili nel dataset. Alla fine sceglie

lo scambio che fa aumentare di più il punteggio totale del team (Greedy Ascent). Dato che l'Hill Climbing puro rischia spesso di bloccarsi in un "ottimo locale" (una soluzione buona ma non la migliore in assoluto), ho aggiunto dei riavvii casuali (Random Restarts). Praticamente faccio ripartire l'algoritmo da capo più volte creando squadre iniziali completamente casuali, e alla fine mi faccio restituire la migliore in assoluto tra tutte le esecuzioni.

Coefficienti per aumentare la sinergia: per impedire che

Valutazione

A differenza dei modelli di machine learning dove si guarda l'accuratezza, per valutare questo algoritmo di ricerca locale ho osservato quanto velocemente arriva a una soluzione e che punteggio riesce a raggiungere.

Ho fatto girare l'algoritmo impostando 10 riavvii casuali (Random Restarts), dando a ciascuno un limite massimo di 1000 iterazioni per non farlo girare all'infinito.

Ho notato che sia con che senza il random restart l'algoritmo convergeva sempre allo stesso risultato a parità di vincoli

Il sistema non sceglie solo i migliori pokémon ma anche quelli che funzionano meglio (eccezion fatta per i primi tre che erano parte del vincolo hard):

```
Inizio la ricerca con Greedy Ascent...

==================================================
=== SQUADRA OTTIMA (LOCALE) TROVATA ===
==================================================
Punteggio Totale (Forza + Sinergia): 14.89

1. Butterfree           (ID:  16) | Score:   0.498 | Ruolo: Cluster 4
2. Ekans                (ID:  29) | Score:  -0.932 | Ruolo: Cluster 5
3. Dragonair            (ID: 161) | Score:   0.452 | Ruolo: Cluster 5
4. Meloetta Pirouette Forme (ID: 717) | Score:   5.187 | Ruolo: Cluster 2
5. Deoxys Speed Forme   (ID: 432) | Score:   5.962 | Ruolo: Cluster 1
6. Electrode            (ID: 110) | Score:   5.727 | Ruolo: Cluster 5
```

Conclusioni e possibili estensioni

Per quanto riguarda la parte di apprendimento non supervisionato è possibile espandendo la grid search e lavorando con algoritmi più flessibili (dbscan) trovare classi migliori, nonostante ciò il risultato trovato è soddisfacente e indica un clustering adeguato per il nostro obiettivo: il bilanciamento dei ruoli

Per quanto riguarda il modello di apprendimento supervisionato cioè il GBDT è riuscito a risolvere correttamente il problema del ranking e con l'analisi delle feature abbiamo visto che molto probabilmente il problema sarebbe potuto essere risolto anche da un modello lineare.
Per sfruttare al meglio le sue potenzialità sarebbe necessario trovare un dataset con più feature che modella situazioni più complicate.

Infine l'algoritmo di Greedy Ascent riesce a trovare sempre un ottimo locale (che data la semplicità del contesto, i vincoli imposti sono una semplificazione del dominio scelto, è stato visto che combacia sempre a un ottimo globale) abbastanza in fretta (di solito in meno di 100 iterazioni per ogni riavvio).

Il risultato della "Run Ottimale" dimostra che unire tutti i pezzi del progetto ha funzionato bene. Se l'algoritmo avesse guardato solo la forza bruta, mi avrebbe restituito una squadra formata semplicemente dai primi 6 Pokémon della classifica del LightGBM (che però sarebbero stati quasi tutti Sweeper fragilissimi e con debolezze sovrapposte). Invece, grazie alla funzione obiettivo che ho disegnato, l'algoritmo "sacrifica" volentieri un po' di pura potenza individuale per inserire pokémon estratti da cluster diversi (come i Wall difensivi) e con tipi differenti.

Il risultato è uno strumento in grado di indirizzare un novizio verso squadra che iè matematicamente forte e getta le basi per uno strumento più avanzato in futuro integrando le considerazioni fatte.