Dipartimento di Informatica

Corso di laurea in Informatica

Caso di Studio per l'esame di Ingegneria della Conoscenza

Agente predittivo nell'ecosistema competitivo di Pokémon

Progetto di:
Nicolò Resta (758440) n.resta6@studenti.uniba.it

Link Repository:
https://github.com/ashkihotah/icon

Anno Accademico **2023-2024**

Indice

1 Introduzione

Nell'ecosistema competitivo del gioco Pokémon, la composizione di una squadra è di fondamentale importanza per poter aumentare le chance di vittoria in un match. La combinazione di tipi all'interno di una squadra, oltre ad altre caratteristiche e proprietà di quest'ultima, gioca un ruolo cruciale nel determinare il risultato di uno scontro. Tuttavia, scegliere i giusti tipi di Pokémon per ottenere una copertura ottimale contro una vasta gamma di avversari può essere una sfida complessa e soggettiva. Un sesto Pokémon ben scelto può coprire le debolezze del resto della squadra e fornire un vantaggio strategico contro l'avversario.

In questo contesto, l'uso delle tecniche di **apprendimento supervisionato**, **deep learning** e **programmazione logica** emerge come un'opportunità promettente per ottimizzare la strategia di squadra. Questo caso di studio si propone di esplorare l'applicazione di tali tecniche per la previsione dei tipi del sesto Pokémon all'interno di una squadra composta da sei Pokémon, dati gli altri cinque in input.

1.1 Elenco argomenti di interesse

1. **Programmazione Logica**: è stato utilizzato il linguaggio Prolog per effettuare il parsing di un team pokemon a partire dalla sua rappresentazione a stringa standard di Pokèmon Showdown e per effettuare la verifica dell'integrità dei dati presenti in esso. Le sezioni di programma, del libro di riferimento [2], sono le seguenti: Rappresentazione e ragionamento proposizionale (capitolo 5 Proposizioni ed inferenze) e relazionale (capitolo 15 individui e relazioni).

2. **Apprendimento supervisionato**: Sono stati addestrati e valutati i classici modelli di apprendimento supervisionato per poter scegliere quello più performante. Le sezioni di programma, del libro di riferimento [2], sono le seguenti: Problemi di apprendimento, Fondamenti di Apprendimento Supervisionato, Valutare le predizioni, Stime puntuali senza Input Features, Tipi di errori, Modelli base di Apprendimento Supervisionato (Alberi di Decisione e Regressori Lineari), Overfitting (Pseudoconteggi, Regolarizzazione e Cross Validation), Modelli compositi (Boosting con Gradient-Boosted Trees e Bagging con Random Forests)

3. **Reti Neurali e Deep Learning**: Sono state addestrate e valutate diverse reti neurali con architetture e configurazioni diverse degli iperparametri per poter scegliere quella più performante. Le sezioni di programma, del libro di riferimento [2], sono le seguenti: Feedforward Neural Networks, Apprendimento di parametri

4. **Tecniche di Over/Under-sampling**: Sono state applicate tecniche di Over/Under-sampling per poter migliorare, gestire e valutare i modelli in presenza di un dataset fortemente sbilanciato sulle classi target. Le sezioni di tali argomenti extra, prese dalla guida di riferimento [1], sono le seguenti: Over-sampling, Under-sampling, tecniche miste di Over/Under-sampling, valutare le predizioni in presenza di sbilanciamento delle classi.

5. **Apprendimento Probabilistico**: è stata appresa una belief network sul dataset di riferimento per poter combattere la scarsità di dati e lo sbilanciamento del dataset attraverso la generazione di nuovi sample artificiali e sensati. Le sezioni di programma, del libro di riferimento [2], sono le seguenti: Ragionamento sotto incertezza (capitolo 9) e Apprendimento sotto incertezza (capitolo 10).

2 Analisi del dominio di interesse

Data la popolarità del gioco, esistono diverse modalità di gioco, chiamati **formati**, e **piattaforme** di online matching. I formati definiscono le regole, le restrizioni e le condizioni specifiche in cui avvengono i combattimenti tra giocatori. Questi formati sono progettati per fornire varietà e equilibrio al gioco competitivo, consentendo ai giocatori di partecipare a sfide con diverse sfumature strategiche e composizioni di squadra. Oltre ai formati ufficiali di Nintendo esistono formati definiti da Smogon, una community internazionale di Pokémon. Il formato analizzato in questo caso di studio è il formato OverUsed (**OU**) di Smogon. Questo formato è stato scelto per i seguenti principali motivi:

1. **Popolarità**: è il formato più popolare tra i videogiocatori competitivi di tutto il mondo.
2. **Disponibilità di dati**: la sua enorme e internazionale popolarità ed importanza è, anche, fonte di un'enorme quantità di dati a nostra disposizione che è possibile analizzare.

La piattaforma di online matching presa in considerazione, per il caso di studio, è Pokémon Showdown. Questa piattaforma è stata scelta per i seguenti principali motivi:

1. **Popolarità**: è la piattaforma dove i videogiocatori competitivi di Pokémon di tutto il mondo si riuniscono per competere fra di loro in battaglie con diversi formati disponibili.
2. **Disponibilità di dati**: data l'enorme popolarità internazionale, tale piattaforma è diventata, ormai, l'unico punto di riferimento da cui è possibile ottenere grandissime quantità di dati sui replay di partite giocate e salvate da altri giocatori che la piattaforma stessa mette pubblicamente a disposizione.
3. **Compatibilità con i formati di Smogon**: oltre ad essere compatibile con i formati ufficiali Nintendo, tale piattaforma è compatibile anche con i formati della community di Smogon tra cui il formato OU di nostro interesse.

Tutti i giochi Pokémon creati dalla Pokémon Company sono divisi in generazioni [4]. Attualmente, ad Aprile del 2024, esistono nove differenti generazioni, ognuna composta da un numero variabile di Pokémon. Ogni generazione, oltre ad aggiungere nuovi Pokémon, presenta una serie di caratteristiche che possono sostituire quelle dei precedenti titoli. Le generazioni sono divise cronologicamente per ordine di pubblicazione. La generazione analizzata in questo caso di studio è la **5° generazione**, introdotta a partire dal 2010 [4]. Questa generazione è stata scelta poichè lo scopo di questo caso di studio è quello di esplorare ed impostare, per la prima volta, un progetto sull'analisi del competitivo Pokémon. Per farlo, tuttavia, si è dovuto scendere a compromessi tra la complessità della generazione Pokémon, in termini di feature connesse, e la disponibilità di dati da raccogliere. La quinta generazione, dopo un'attenta esplorazione di features e di dati, risulta essere una buona base di partenza con un livello di complessità abbastanza alto da poter analizzare aspetti interessanti del problema di nostro interesse, senza complicare inutilmente fin dall'inizio il dominio da analizzare. Partire da una base semplice, con la complessità più adatta per l'impostazione di questo studio, permette una gestione più efficace del progetto e permette una più facile espansione modulare per obiettivi futuri. Il nostro dominio di interesse, dunque, si restringe a tutti quei dati ricavabili dalla **5° Generazione Pokemon OU** espressi nel formato standard di **Pokémon Showdown**. Fissare il dominio di interesse ad una generazione è molto importante poichè da generazione a generazione le tecniche di team building cambiano

poichè cambiano i Pokémon disponibili, le regole di gioco e tante altre meccaniche di gameplay. Mischiare i team costruiti dai giocatori in generazioni Pokémon differenti può essere controproducente dato che i modelli di apprendimento automatico potrebbero ritrovarsi dei pattern di team building contrastanti e contraddittori.

2.1 Features di un team di Pokémon

Nel formato OU, scelto, ogni giocatore ha a disposizione 6 pokemon nel team. Nella quinta generazione, ciascuno di questi sei Pokémon ha le seguenti caratteristiche o features (quelle con * sono obbligatorie e sempre presenti o valorizzate):

1. **Nome o ID***: nome (*stringa*) o identificativo (*intero*) del Pokémon.
2. **Tipo principale***: il principale tipo del Pokémon. Corrisponde ad una feature categorica con dominio [*Normal, Fighting, Flying, Poison, Ground, Rock, Bug, Ghost, Steel, Fire, Water, Grass, Electric, Psychic, Ice, Dragon, Dark*]
3. **Tipo Secondario***: è il tipo secondario del Pokémon. Corrisponde ad una feature categorica con dominio [*null_type, Normal, Fighting, Flying, Poison, Ground, Rock, Bug, Ghost, Steel, Fire, Water, Grass, Electric, Psychic, Ice, Dragon, Dark*]. Il valore *null_type* è stato introdotto per gestire tutti quei Pokémon senza il secondo tipo.
4. **Mosse***: lista di massimo 4 mosse che un Pokémon può avere e utilizzare in battaglia. Ciascuna mossa ha le seguenti sotto-caratteristiche:
   (a) **Nome***: *stringa* rappresentante il nome della mossa in questione.
   (b) **Tipo***: feature categorica con dominio [*Normal, Fighting, Flying, Poison, Ground, Rock, Bug, Ghost, Steel, Fire, Water, Grass, Electric, Psychic, Ice, Dragon, Dark*].
   (c) **PP***: *intero* che rappresenta il numero massimo di volte in cui quella mossa può essere utilizzata prima che si esauriscano i suoi utilizzi ovvero i suoi Punti Potenza (Power Points - PP).
   (d) **Damage Class***: feature categorica con dominio [*Special, Phisical, Status*] che rappresenta la classe della mossa.
   (e) **Effetto**: *stringa* rappresentante la descrizione di un eventuale effetto che una mossa potrebbe avere.
   (f) **Probabilità**: *float* rappresentante la probabilità con la quale l'eventuale effetto, se presente, si applica in battaglia.
   (g) **Potenza**: se la mossa non è di Damage Class *Status*, è un *intero* che indica il danno inflitto al Pokémon avversario.
   (h) **Accuratezza**: *float* rappresentante la probabilità con la quale la mossa può colpire il Pokémon avversario.
5. **Abilità***: un'abilità tra quelle che il Pokémon può avere. Le abilità conferiscono delle proprietà particolari al Pokémon che la possiede. Essa è composta da un nome e da una descrizione che ne esplica gli effetti. Può essere vista come una *feature categorica* di cui non si riportanto tutti i possibili valori per facilitare la lettura.
6. **Item**: oggetto che il Pokémon possiede ed eventualmente usa in battaglia. Gli items conferiscono delle proprietà particolari al Pokémon che la possiede. Può essere vista come una *feature categorica* di cui non si riportanto tutti i possibili valori per facilitare la lettura.

7. **Ruolo***: per ruolo si intende la combinazione delle **Statistiche***, degli **EV*** e della **Natura*** del Pokémon. Queste sono altre features che un Pokémon ha e che si è deciso di non descrivere in maniera approfondita per facilitare la lettura.

Le features prese in considerazione per il caso di studio sono quelle riportate in verde. Si è deciso di considerare solo quelle features per i seguenti principali motivi:

1. **Significatività**: tali features risultano essere le più significative per il task di nostro interesse. Questa è un'assunzione molto forte e stringente e verrà ripresa nella sezione della valutazione dei risultati.

2. **Facilità di rappresentazione**: tali features sono facilmente rappresentabili e veicolabili ai classici modelli di machine learning utilizzati nel caso di studio. Per rappresentare e veicolare le altre features sono necessarie rappresentazioni notevolmente più complesse che verranno discusse in seguito nelle sezioni.

3 Creazione del Dataset

Prima di poter eseguire qualsiasi tipo di esperimento di apprendimento automatico, vi è la necessità di costruire un dataset adatto al nostro scopo. Purtroppo, per il task prefissato in questo caso di studio, è stato molto difficile trovare una fonte da cui prendere un dataset già ben pulito, costruito e adatto al nostro task. La principale difficoltà incontrata è stata la mancanza di un numero adeguato di team di esempio costruiti da utenti, esperti nel competitivo Pokémon, o adirittura la totale mancanza di sorgenti di dati. Una possibile soluzione a questo problema, esplorata inizialmente, è stata quella di raccogliere quanti più replay possibile da Pokémon Showdown ed effettuare il parsing di ciascun file di log raccolto in modo da ricavare quante informazioni possibili riguardanti i team dei due player coinvolti nei replay singoli e salvarle in un dataset sperando di riuscire a raccogliere abbastanza dati. Questa soluzione è stata implementata tuttavia, dopo un'attenta analisi dei dati raccolti, è stato possibile notare che più del 90% dei dati raccolti erano incompleti e, dunque, inutilizzabili per il nostro scopo. Si è scelto, dunque di continuare la ricerca di eventuali fonti di dati, anche grezze da dover ripulire, per poter valutare la fattibilità del caso di studio. Fortunatamente è stato possibile trovare due dataset che, combinati, avrebbero permesso la costruzione di un dataset affidabile e robusto. Nelle sezioni di seguito vi è la descrizione di come è stato costruito il dataset utilizzato.

3.1 Dataset utilizzati

I dataset utilizzati, per la costruzione del dataset finale, sono i seguenti:

1. **Dataset delle features dei Pokémon**: scaricato da kaggle, contiene tutte le informazioni riguardanti il gioco Pokémon completo. Questo dataset contiene, anche, informazioni in più riguardanti le statistiche di gioco del formato VGC che, però, non sono state prese in considerazione poichè fuori dal nostro dominio di interesse.

2. **Dataset dei team**, preso da un tool sviluppato da fulllifegames, un membro della community di Smogon, chiamato Smogon Team Dump. Questo tool, in base a dei filtri, permette di restituire tutti i team pubblicati sui forum di Smogon in un formato stringa, standard di Pokémon Showdown. In particolare, in un suo post, ha condiviso un link ad un suo drive Google in cui sono presenti tutti i dati da lui raccolti. Per il caso di studio è stato utilizzato il file `gen5ou.json` come dataset contenente circa 1131 squadre di Pokémon pubblicati da giocatori competitivi.

Attraverso il parsing dei team in formato stringa, nel secondo dataset, e attraverso le informazioni contenute nel primo dataset è stato possibile riuscire a ricavare tutte le features di un team Pokémon di cui necessitiamo ed è stato possibile riuscire a costruire un dataset abbastanza grande, robusto e affidabile.

3.2 Pre-processing dei dati

Poichè il dataset delle features dei Pokémon fa riferimento al formato VGC dell'ultima generazione, mentre il dominio di interesse del caso di studio si limita alla quinta generazione del formato OU, e poichè i dataset utilizzati provengono da due fonti diverse, è stata dedicata molta attenzione al pre-processing dei dati. In particolare è stato necessario:

1. Normalizzare i valori dei tipi dei Pokémon e delle Mosse di questi ultimi all'insieme dei valori [*Normal, Fighting, Flying, Poison, Ground, Rock, Bug, Ghost, Steel, Fire, Water, Grass, Electric, Psychic, Ice, Dragon, Dark*].

2. Normalizzare i nomi delle Mosse. Poichè il dataset dei team di Pokémon proviene da uno web scraping dei post pubblicati nei forum di Smogon, molti utenti hanno utilizzato diverse stringhe equivalenti, ma diverse per sintassi (un esempio è la mossa U-Turn scritta in varianti quali U Turn, U-turn ecc.), per rappresentare una stessa mossa.

3. Tutte le informazioni riguardanti le generazioni successive alla quinta sono state ignorate o, eventualmente, convertite per aderire al contenuto presente nella quinta generazione (Ad esempio il Pokémon Togekiss in generazioni successive alla quinta ha anche il tipo *Fairy*, non esistente nella quinta generazione, come secondo tipo. In tal caso è bastato convertire i tipi di Togekiss a quelli che aveva in quinta generazione).

Altri problemi sorti durante il caricamento dei dati sono la **mancanza**, l'**ambiguità** e l'**illegibilità** delle informazioni per alcuni team nel dataset dei team Pokémon. Per questo caso di studio le condizioni minime necessarie affinchè i dati possano essere utilizzati correttamente sono le seguenti:

1. Il team dev'essere completo: dev'essere composto da 6 Pokémon.

2. Ciascun Pokémon nel team deve avere 4 mosse, ovvero il suo "moveset" dev'essere completo.

3. Il team non deve contenere informazioni ambigue. Spesso gli utenti nel forum da cui il dataset è stato raccolto hanno pubblicato più mosse per uno stesso slot per dare più opzioni di scelta e per discuterne con altri utenti nel forum. Questa corrisponde ad un'informazione ambigua dato che, per lo stesso slot, non si può sapere qual'è la mossa giusta da considerare.

4. Il team dev'essere leggibile e, dunque, descritto nel formato standard di Pokémon Showdown così da poter permettere il parsing e l'estrazione delle informazioni. Spesso, nel dataset, si possono trovare team descritti in maniera non strutturata, tramite il linguaggio naturale, o non nel formato standard strutturato di Pokémon Showdown.

Tutti quei team che non verificano congiuntamente tutte le precedenti condizioni sono stati ignorati. Effettuato tale pre-processing il numero di team disponibili sono scesi da **1131 in totale nel dataset** a **928 caricati correttamente**. Una perdita di 203 team, rispetto alle problematiche che essi avrebbero portato, è più che tollerabile.

3.3 Scelta del Pokémon target e Data Augmentation

Un primo passo per iniziare a costruire una rappresentazione degli esempi da inserire nel training set è quello di scegliere, per ciascun team, il Pokémon target di cui si vorrà prevedere il tipo primario e secondario. Ciascun esempio, all'interno del dataset, infatti, dovrà tenere conto di quali sono i Pokémon di input e di qual'è il Pokémon target. Invece di utilizzare un algoritmo che designa arbitrariamente un Pokémon come target e gli altri come input, si è deciso di scrivere un algoritmo che costruisce, per ciascun team originale caricato correttamente dal dataset, 6 esempi "artificiali", uno per ogni Pokémon presente nel team, designando, a turno, ciascun Pokémon come target e gli altri come Pokémon di input. L'algoritmo 1 mostra l'idea appena spiegata. Questa funzione prende in input una lista di Pokémon, rappresentante un team, e restituisce una coppia di liste: `input_teams` è una lista contenente sei liste con cinque Pokémon ciascuna che rappresentano l'insieme dei Pokémon di input mentre `target_pks` è una lista, parallela alla prima, contenente i rispettivi Pokémon target scelti a turno. Ciò ci permetterà di costruire le input features, a partire

Algorithm 1 Algoritmo di generazione artificiale di esempi

```
def getSamples(pokemons):
        input_teams = []
        target_pks = []
        target = 0
        for target in range(0, len(pokemons)):
            target_pks.append(pokemons[target])
            subset = []
            for pk in pokemons:
                if pk != pokemons[target]:
                    subset.append(pk)
            input_teams.append(subset)
        return input_teams, target_pks
```

da `input_teams`, e le target features, a partire da `target_pks`, degli esempi del dataset. Questa operazione ha portanto ai seguenti vantaggi:

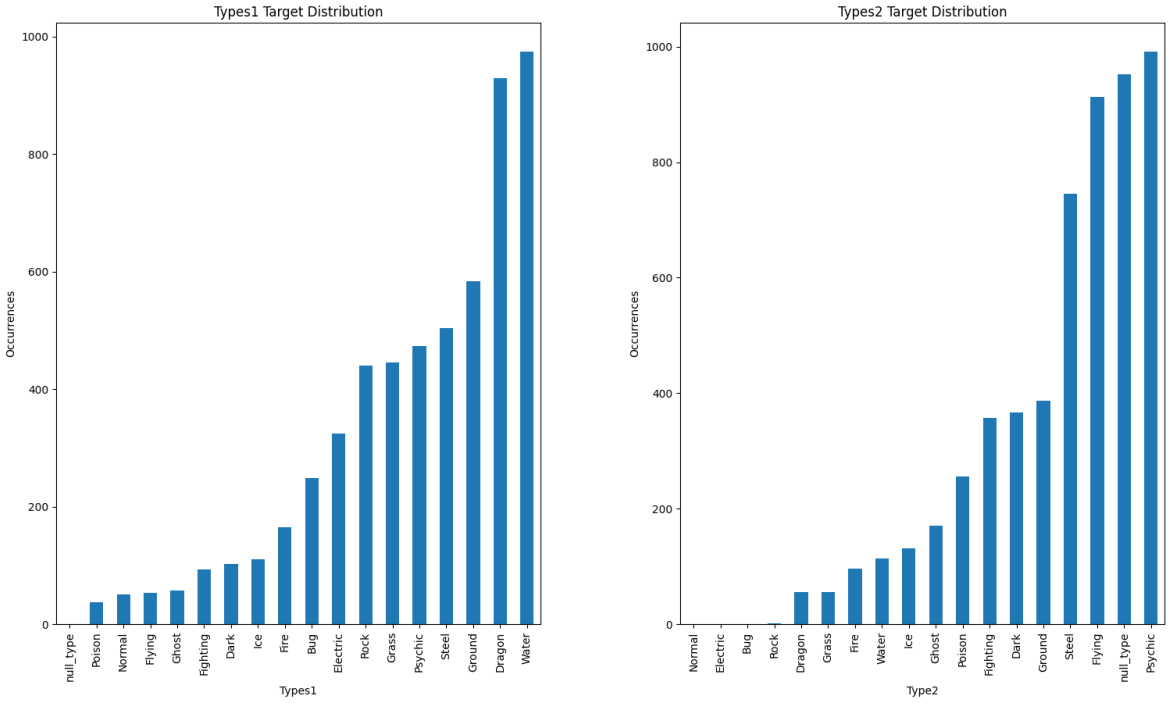
1. **Data Augmentation:** attraverso questa operazione è stato possibile aumentare artificialmente il numero degli esempi a nostra disposizione da **928** a **928 * 6 = 5527**. Più esempi riusciamo a raccogliere, più i modelli avranno una maggiore evidenza su cui ragionare.

2. **Sfruttamento efficiente dei dati**: ciò ci permette di sfruttare al meglio i pochi dati a nostra disposizione. In questo modo, infatti, i modelli di apprendimento potranno analizzare più a fondo uno stesso team da "punti di vista differenti" riuscendo, dunque, a catturare meglio i pattern, di team building, che si vuole loro insegnare.

3.4 Features Engineering di un team Pokémon

Durante lo sviluppo del progetto sono state sperimentate diverse rappresentazioni delle features in modo tale da scegliere quella più adatta e che potesse veicolare le informazioni nel modo più giusto possibile. Nella rappresentazione finale adottata, ogni esempio nel dataset è un array di dimensione 306 in cui ciascuna posizione è associata ad una particolare informazione. Le prime 289 posizioni sono mappate biunivocamente a tutte le possibili combinazioni di tipi primari e secondari di cui un Pokémon può essere, pari a 17 x 17 = 289. Le ultime 17 posizioni, invece, sono mappate biunivocamente a tutti i possibili tipi di cui una mossa può essere. In questo modo, nelle prime 289 posizioni andremmo a salvare il numero di Pokèmon presenti nel team di quella combinazione di tipi mentre nelle ultime 17 posizioni andremmo a salvare il numero di mosse presenti nel team di quel tipo. Se, ad esempio, in un team ci sono 2 Pokèmon di tipo Fuoco, 3 di tipo Acqua Volante e 1 di tipo Normale allora nelle prime 289 posizioni dell'array soltanto le posizioni associate al tipo Fuoco, Acqua Volante e Normale saranno valorizzate rispettivamente a 2, 3 e 1 mentre tutte le altre saranno impostate a 0. Continuando l'esempio, se nello stesso team ci sono 4 mosse di tipo Fuoco, 8 di tipo Acqua e 12 di tipo Volante allora nelle ultime 17 posizioni soltanto le posizioni associate al tipo Fuoco, Acqua e Volante saranno valorizzate rispettivamente a 4, 8 e 12 e il resto delle posizioni in quel range sarà impostato a 0. Questa rappresentazione è stata ideata per evitare che i modelli apprendessero dei pattern errati anche in base alle posizioni, dell'array, in cui i valori comparivano.

Poichè tutte le possibili combinazioni di tipi primari e secondari dei Pokémon sono 289, non è possibile visualizzare graficamente in maniera chiara qual'è la loro distribuzione. Per avere un'idea della loro distribuzione in Figura 1 sono mostrate le distribuzioni dei tipi primari e secondari, dei Pokèmon, separatamente: in Figura 1a dei tipi primari e in Figura 1b



(a) Distribuzione dei Tipi primari

(b) Distribuzione dei Tipi secondari

Figura 1: Distribuzioni dei tipi primari e secondari dei Pokémon target

dei tipi secondari. Da queste due figure possiamo notare come il dataset creato è fortemente sbilanciato a causa di un maggiore utilizzo, dei giocatori, dei tipi più popolari. Questo comporta, inevitabilmente, che, anche, la distribuzione delle loro combinazioni sarà fortemente sbilanciata. Le classi *null_type* nella distribuzione dei tipi primari e *Normal* nella distribuzione dei tipi secondari è normale che siano poste a 0 dato che il tipo primario non può essere nullo e non esistono Pokémon di tipo secondario *Normal* nella quinta generazione. Soltanto le classi *Electric* e *Bug* nella distribuzione dei tipi secondari sono problematiche in quanto non potranno essere previste dal alcun modello come tipi secondari. In questo caso di studio, purtroppo, questo problema non verrà affrontato tuttavia saranno proposte delle soluzioni per risolvere il problema. Quello che si sta per affrontare, dunque, è un problema di classificazione multiclasse con classi target sbilanciate.

4 Programmazione Logica

Per il parsing del team Pokèmon a partire dal suo formato stringa standard, è stato utilizzato il Prolog. Questa decisione è stata presa in quanto i dati, provenienti dal dataset, non erano tutti facilmente elaborabili bensì presentavano molte irregolarità, come specificato nella sezione 3.1. Grazie alla possibilità di definire fatti e regole logiche da cui poi inferire eventuali informazioni e validare l'integrità dei dati, è stato possibile gestire efficacemente e facilmente i dati utilizzando delle basi di conoscenza. Si è scelto di usare la libreria `swiplserver` per l'integrazione tra il linguaggio Python e Prolog. Un esempio di stringa standard di cui effettuare il parsing e la validazione è la seguente:

```
Politoed @ Chesto Berry
Ability: Drizzle
EVs: 252 HP / 156 Def / 56 SpD
   / 44 Spe
Bold Nature
IVs: 0 Atk
- Scald
- Toxic
- Encore
- Rest

Thundurus Therian Forme @
   Leftovers
Ability: Volt Absorb
EVs: 32 HP / 176 SpA / 48 SpD
   / 252 Spe
Timid Nature
IVs: 2 Atk / 30 Def
- Thunder
- Hidden Power [Ice]
- Focus Blast
- Protect

Tentacruel @ Black Sludge
Ability: Rain Dish
EVs: 252 HP / 68 SpD / 188 Spe
Timid Nature
IVs: 0 Atk
- Scald
- Rapid Spin
- Toxic
- Protect

Ferrothorn @ Chople Berry
Ability: Iron Barbs
EVs: 252 HP / 72 Def / 184 SpD
Sassy Nature
IVs: 0 Spe
- Stealth Rock
- Spikes
- Power Whip
- Gyro Ball

Latios @ Choice Scarf
Ability: Levitate
EVs: 4 Def / 252 SpA / 252 Spe
Timid Nature
IVs: 0 Atk
- Draco Meteor
- Trick
- Surf
- Dragon Pulse

Scizor @ Sitrus Berry
Ability: Technician
EVs: 248 HP / 216 Atk / 8 Def
   / 36 SpD
Adamant Nature
- Bullet Punch
- Bug Bite
- Pursuit
- U Turn
```

Le basi di conoscenza implementate sono le seguenti:

1. `types.pl`: definisce quali sono tutti i possibili tipi validi e qual'è il loro nome standard. Esempi di regole presenti in tale base di conoscenza sono mostrate in figura 2.

Algorithm 2 Base di conoscenza types.pl

```
type("Normal").
type("Fighting").
type("Flying").
. . .
```

2. `pokemons.pl`: definisce quali sono tutti i possibili pokemon validi e qual'è il loro nome standard. Esempi di regole presenti in tale base di conoscenza sono mostrate in figura 3.

Algorithm 3 Base di conoscenza pokemons.pl

```
pokemon("Bulbasaur").
pokemon("Ivysaur").
pokemon("Venusaur").
. . .
```

3. `moves.pl`: definisce quali sono tutte le possibili mosse valide dei pokemon e qual'è il loro nome standard. Esempi di regole presenti in tale base di conoscenza sono mostrate in figura 4. In figura 4 la regola `move(input)` è una prima regola che permette di fare inferenza e di riconoscere mosse in formati tipo Hidden Power [Ice] e di validare sia il nome della mossa (Hidden Power) sia il tipo di riferimento (Ice).

Algorithm 4 Base di conoscenza moves.pl

```
move(Input) :-
    sub_string(Input, Before, _, After, ' [').
    NewAfter is After - 1,
    sub_string(Input, 0, Before, _, Move),
    sub_string(Input, _, NewAfter, 1, Type),
    type(Type),
    move(Move).

move("Absorb").
move("Accelerock").
move("Acid").
. . .
```

4. `parser.pl`: definisce le regole ed i fatti per effettuare il parsing del team Pokèmon a partire dal loro formato stringa. Le regole presenti in tale base di conoscenza sono mostrate in figura nelle figure seguenti. Le prime tre regole in figura 5 permettono di

Algorithm 5 Base di conoscenza parser.pl (importazione delle dipendenze)

```
:- ensure_loaded(pokemons).
:- ensure_loaded(moves).
:- ensure_loaded(types).
```

usare regole e fatti definite in altre basi di conoscenze. In questo caso permettono di usare quelle definite in `pokemons.pl`, `moves.pl` e `types.pl`.

In figura 6 vengono mostrate le regole che effettuano il parsing del team. La prima regola permette di effettuare uno split in base al doppio carattere newline in modo tale da riconoscere ed effettuare il parsing delle build di tutti i pokemon presenti nel team. La regola `parse_pokemons` permette di effettuare il parsing della singola build pokemon e di validarla. La validazione di tale regola permette di validare anche la regola `parse_team`. La regola `parse_pokemons` effettua uno split in base al singolo carattere newline per ottenere tutte le stringhe che compongono la build pokemon. L'array di linee ottenuto viene utilizzato dalle regole `parse_name_and_item` e `parse_`

moves per effettuare i propri parsing. Infine la regola parse_pokemons ottiene i risultati delle due regole precedenti e restituisce il fatto pokemon_build(Name,Item,Moves) che indica la validità della build pokemon.

Algorithm 6 Base di conoscenza parser.pl (parsing del team)

```
list_of_six_elements([_, _, _, _, _, _]).
list_of_four_elements([_, _, _, _]).

% Parse the entire team description
parse_team(TeamString, Output) :-
    % Split the team description by line breaks
    atomic_list_concat(PokemonAtomStrings, '\n\n', TeamString),
    maplist(atom_string, PokemonAtomStrings, RawPokemonStrings),
    list_of_six_elements(Output),
    parse_pokemons(RawPokemonStrings, Output).

parse_pokemons([], []). % Base case: empty list, return empty list
parse_pokemons([PokemonString | RestPks], [ParsedPk | ParsedRest]) :-
    split_string(PokemonString, "\n", "", Lines),
    parse_name_and_item(Lines, Name, Item),
    parse_moves(Lines, Moves),
    list_of_four_elements(Moves),
    ParsedPk = pokemon_build(Name, Item, Moves),
    parse_pokemons(RestPks, ParsedRest).
```

In figura 7 vengono mostrate le regole che effettuano il parsing e verificano la validità dei nomi dei Pokèmon. La regola che ha la testa parse_name_and_item permette di effettuare il parsing, dalla prima riga della stringa, del nome e dell'item in modo da poterli validare successivamente. La regola con la testa trim_spaces permette di normalizzare gli spazi nella stringa in input. Le regole che hanno come testa parse_name permettono di riconoscere e validare, con pokemon(ParsedName), il nome del Pokèmon nei seguenti esempi di formati: Sloth (Hippowdon) (F) (Nickname (Nome standard) (sesso)), Thundurus-Therian (M) (Nome standard (sesso)) e Pikachu (Nome standard).

In figura 8 vengono mostrate le due regole per effettuare il parsing delle mosse. Tali regole iterano per tutte le linee presenti e se trovano il carattere iniziale '-' riconoscono la mossa, effettuano il parsing e la validano con move(Move).

Durante il processo di creazione del dataset, alla base di conoscenza parser.pl viene effettuata la query parse_team(TeamString,ParsedTeam), tramite l'utilizzo della libreria swiplserver, in cui TeamString rappresenta la stringa in formato standard del team pokemon, e pertanto rappresenta l'input, e ParsedTeam è una lista di pokemon_build(Name, Item,Moves) che rappresenta le build validate dei pokemon nel team. Questa lista, nel caso in cui la query abbia dato esito positivo, viene poi utilizzata in python per creare gli oggetti PokemonTeam e Pokemon necessari per poter creare successivamente il dataset. Così facendo è stato possibile gestire la difficoltà del parsing e di mantenere la logica di controllo dell'integrità dei dati al di fuori del codice python permettendo una più chiara manutenzione e uno sviluppo più semplice del progetto.

Algorithm 7 Base di conoscenza parser.pl (parsing del nome e dell'item)

```
parse_name_and_item(Lines, Pokemon, Item) :-
    nth0(0, Lines, Line),
    split_string(Line, "@", "both", [PokemonPart, ItemPart]),
    atom_string(PokemonPart, PokemonStr),
    trim_spaces(PokemonStr, PokemonClean),
    parse_name(PokemonClean, Pokemon),
    atom_string(ItemPart, ItemStr),
    trim_spaces(ItemStr, Item).

trim_spaces(Input, Output) :-
    atom_string(Input, InputStr),
    normalize_space(atom(TrimmedAtom), InputStr),
    atom_string(TrimmedAtom, Output).

parse_name(Name, ParsedName) :-
    split_string(Name, "(|)", " ", Parts),
    length(Parts, 5),
    nth0(1, Parts, ParsedName),
    pokemon(ParsedName).

parse_name(Name, ParsedName) :-
    split_string(Name, "(|)", " ", Parts),
    length(Parts, 3),
    nth0(0, Parts, ParsedName),
    pokemon(ParsedName).

parse_name(Name, ParsedName) :-
    split_string(Name, "(|)", " ", Parts),
    length(Parts, 3),
    nth0(1, Parts, ParsedName),
    pokemon(ParsedName).

parse_name(Name, ParsedName) :-
    pokemon(Name),
    ParsedName = Name.
```

Algorithm 8 Base di conoscenza parser.pl (parsing delle mosse)

```
parse_moves([H|T], Result) :-
    (   string_chars(H, ['-'|RestChars])
    ->  Result = [Move|Rest],
        string_chars(MoveStr, RestChars),
        trim_spaces(MoveStr, Move),
        move(Move),
        parse_moves(T, Rest)
    ;   parse_moves(T, Result)
    ).
parse_moves([], []).
```

5 Apprendimento Supervisionato

Il task di Apprendimento Supervisionato analizzato nel caso di studio è un task di classificazione multiclasse. A partire da un insieme di cinque Pokémon di un team, si cerca di predire quale sarà la giusta combinazione del tipo primario e secondario del sesto Pokémon dello stesso team. Per fare ciò è stato necessario ingegnerizzare e veicolare correttamente, ai modelli di apprendimento automatico supervisionato, tutte le features significative che i cinque Pokémon in input possono offrire ai modelli per permettere loro di apprendere come sfruttare tali informazioni per prevedere il giusto valore della target feature.

5.1 Modelli e metodologie utilizzate

Metodologia utilizzata: Al fine di poter addestrare il modello più adatto al nostro caso di studio, è stato necessario:

1. Scegliere, per ciascun modello, i valori migliori degli iper-parametri.
2. Addestrare ciascun modello con i valori migliori per i suoi iperparametri.
3. Valutare ciascun modello con i valori migliori per i suoi iperparametri.

Questa iterazione è stata ripetuta per tutti i modelli di seguito riportati. Maggiori informazioni su ciascuna delle precedenti fasi sono riportate nei paragrafi successivi.

Modelli adottati e iperparametri: I modelli adottati per i seguenti esperimenti, con i loro iperparametri e corrispettivi range di valori, sono:

1. **Regressori Lineari** (Linear Regressors), con la libreria `scikit-learn`.
   - **penalty**: Fattore di penalizzazione, o di regolarizzazione, da applicare all'errore per mitigare l'overfitting.
     I valori sperimentati sono: ["l1", "l2", "elasticnet"],
   - **class_weight**: I pesi associati a ciascuna classe. La modalità "bilanciata" utilizza i valori della target feature per regolare automaticamente i pesi in modo inversamente proporzionale alle frequenze delle classi nei dati di input.
     I valori sperimentati sono: [None, "balanced"],
   - **max_iter**: Numero massimo di iterazioni da effettuare nell'apprendimento.
     I valori sperimentati sono: [100, 1000].
2. **Alberi di Decisione** (Decision Trees), con la libreria `scikit-learn`.
   - **criterion**: Criterio di suddivisione.
     I valori sperimentati sono: ["gini", "entropy", "log_loss"].
   - **max_depth**: Massima profondità dell'albero.
     I valori sperimentati sono: [10, 20, 40].
   - **min_samples_split**: Numero minimo di campioni per suddividere un nodo.
     I valori sperimentati sono: [2, 5, 10, 15].
   - **min_samples_leaf**: Numero minimo di campioni per essere foglia.
     I valori sperimentati sono: [1, 2, 5, 10, 15].

- **class_weight**: I pesi associati a ciascuna classe. La modalità "bilanciata" utilizza i valori della target feature per regolare automaticamente i pesi in modo inversamente proporzionale alle frequenze delle classi nei dati di input.
  I valori sperimentati sono: [None, "balanced"],

3. **Random Forest**, con la libreria `scikit-learn`.

   - **n_estimators**: Numero di stimatori (alberi) nella foresta.
     I valori sperimentati sono: [100, 200],
   - **criterion**: Criterio di suddivisione.
     I valori sperimentati sono: ["gini", "entropy", "log_loss"].
   - **max_depth**: Massima profondità degli alberi.
     I valori sperimentati sono: [None, 10, 20, 40],
   - **class_weight**: I pesi associati a ciascuna classe. La modalità "bilanciata" utilizza i valori della target feature per regolare automaticamente i pesi in modo inversamente proporzionale alle frequenze delle classi nei dati di input.
     I valori sperimentati sono: [None, "balanced"],

4. **Boosting di Alberi di Decisione** (Gradient Boosted Trees), con la libreria `lightgbm`.

   - **learning_rate**: Velocità con cui il modello impara dai dati durante il processo di addestramento. Un learning rate più basso richiede più iterazioni per raggiungere la convergenza, ma può portare a una migliore generalizzazione e a modelli più stabili. D'altra parte, un learning rate più alto potrebbe accelerare il processo di addestramento.
     I valori sperimentati sono: [0.01, 0.05, 0.1, 1.0],
   - **max_depth**: Massima profondità degli alberi appresi.
     I valori sperimentati sono: [10, 20, 40],
   - **n_estimators**: Numero di stimatori (alberi) utilizzati.
     I valori sperimentati sono: [50, 100, 200],
   - **class_weight**: I pesi associati a ciascuna classe. La modalità "bilanciata" utilizza i valori della target feature per regolare automaticamente i pesi in modo inversamente proporzionale alle frequenze delle classi nei dati di input.
     I valori sperimentati sono: [None, "balanced"]

5. **Reti Neurali** (Multi-Layer Perceptrons), con la libreria `scikit-learn`. Queste sono approfondite nella sezione dedicata.

Addestramento e ricerca dei migliori iperparametri: Per l'addestramento, di ciascun modello, è stata utilizzata una **Stratified K-Fold Cross-Validation** in cui il dataset viene partizionato in k = 5 fold con una divisione stratificata, con l'utilizzo della classe `StratifiedKFold` della libreria `scikit-learn`, per permettere una buona distribuzione nei fold delle classi target. In combinazione con la cross-validation, è stata utilizzata una **ricerca esaustiva** per trovare i migliori iperparametri, di ciascun modello, attraverso la classe `GridSearchCV` di `scikit-learn`. Il tutto è stato impostato per massimizzare l'**accuratezza** dei modelli.

Valutazione: Per la valutazione è stata utilizzata, anche qui, una **Stratified K-Fold Cross-Validation**, in combinazione con il metodo `cross_validate` di `scikit-learn`, con il quale è stato possibile calcolare e valutare le performance dei modelli con gli iperparametri migliori per k = 5 iterazioni di training e testing secondo la normale e standard procedura della K-Fold Cross Validation. Ciò ci ha permesso, per ogni metrica, di calcolare lo score medio, la sua varianza e deviazione standard in modo tale da valutare, anche, la stabilità del modello e dei risultati ottenuti. Le metriche adottate, con l'utilizzo di `scikit-learn`, per misurare le performance del modello sono le seguenti: **Accuracy**, **Average Precision**, **Precision**, **Recall/Balanced Accuracy**, **F1**. Tutte le precedenti metriche, tranne l'accuratezza, rappresentano la macro-media delle stesse metriche calcolate per ciascuna classe target, così come specificato dalla libreria `scikit-learn`[3]. Poichè è stata utilizzata la macro-media, Recall e Balanced Accuracy coincidono, per come è stata definita la Balanced Accuracy in `scikit-learn`[3]. In più è stata utilizzata un'altra metrica, la **Geometric Mean** di `imbalanced-learn`[1], per valutare le performance anche sotto sbilanciamento delle classi target. La Geometric Mean è la radice del prodotto della sensibilità per classe. Questa misura cerca di massimizzare l'accuratezza su ciascuna delle classi, mantenendo queste accuratezze bilanciate. Per i problemi multiclasse è la radice n-esima del prodotto della sensibilità per ciascuna classe. Se almeno una classe non viene riconosciuta dal classificatore, la geometric mean sarà 0. Questa metrica è utile per tenere sotto controllo le performance dei modelli sotto sbilanciamento delle classi target.

Riproducibilità: Per garantire la riproducibilità dei risultati si è scelto di fissare il `seed` per controllare la casualità a 42.

5.2 Risultati dell'addestramento e della valutazione dei modelli

In una prima fase si è deciso di allenare i modelli, proposti in precedenza, senza alcuna manipolazione del dataset per analizzare un loro comportamento ed eventuale adattamento di base ai dati. Nella Tabella 1 sono mostrati i migliori iperparametri trovati, con la `GridSearchCV`, per ciascun modello.

Iperparametro	Valore
LR__class_weight	None
LR__max_iter	1000
LR__penalty	l2

(a) Logistic Regression Hyperparameters

Iperparametro	Valore
DT__class_weight	None
DT__criterion	entropy
DT__max_depth	20
DT__min_samples_leaf	1
DT__min_samples_split	2

(b) Decision Tree Hyperparameters

Iperparametro	Valore
RF__class_weight	None
RF__criterion	gini
RF__max_depth	20
RF__n_estimators	100
RF__random_state	42

(c) Random Forest Hyperparameters

Iperparametro	Valore
LGBM__class_weight	balanced
LGBM__learning_rate	0.1
LGBM__max_depth	40
LGBM__n_estimators	200

(d) LGBM Hyperparameters

Tabella 1: Migliori iperparametri per ciascun modello

Nelle tabelle dalla 2 alla 5 sono mostrati i risultati per ciascuna metrica, con relativa varianza e deviazione standard, di ciascun modello.

Metric	Score Mean	Score Variance	Score Std
Accuracy	0.5582815578855184	0.00019698633707695306	0.014035182117698119
Geometric Mean	0.6998499463976634	5.528394484918481e-05	0.007435317400702193
Average Precision	0.5307393833727256	0.0004014792486956356	0.020036947090204028
Precision	0.5271740464683647	0.00044969164307711603	0.02120593414771243
Recall	0.49421452006283656	0.00010805823784555419	0.010395106437432673
F1	0.49506103222307046	0.00028244078514465894	0.016805974685946034

Tabella 2: Risultati delle metriche di valutazione per la Logistic Regression

Metric	Score Mean	Score Variance	Score Std
Accuracy	0.745989409751786	0.0001481699256049562	0.012172506956455446
Geometric Mean	0.8340733283836046	0.0001966428733034402	0.014022940964841869
Average Precision	0.5553623826489003	0.0012001937137470374	0.03464381205564765
Precision	0.7115411837850149	0.0015285287059522853	0.03909640272393722
Recall	0.699413098382785	0.0005442486121477631	0.023329136549554576
F1	0.6897860667627167	0.0009840094546075767	0.0313689249832948

Tabella 3: Risultati delle metriche di valutazione per il Decision Tree

Metric	Score Mean	Score Variance	Score Std
Accuracy	0.7821988144760421	0.0002115512030966191	0.01454479986444018
Geometric Mean	0.8393930347765428	0.00022049643877127483	0.014849122491624709
Average Precision	0.7542473587243925	0.0005279814186908127	0.022977846258751333
Precision	0.7518365942455467	0.0018055183449363719	0.04249139142151469
Recall	0.7078883284112887	0.0006186280234974608	0.024872233986866978
F1	0.7164040162846305	0.0010830355663766728	0.032909505714560236

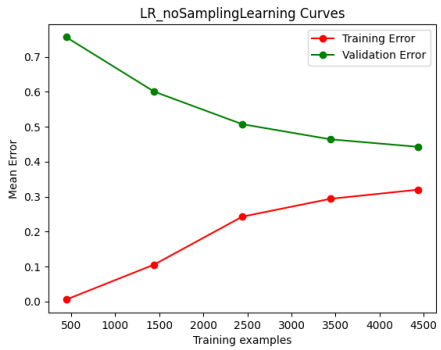
Tabella 4: Risultati delle metriche di valutazione per il Random Forest

Metric	Score Mean	Score Variance	Score Std
Accuracy	0.8108400029192108	0.00024928852714826315	0.015788873523727497
Geometric Mean	0.8634207521324646	0.0002941409485590793	0.017150537850431375
Average Precision	0.8004364491961369	0.0009928903084429248	0.03151016198693565
Precision	0.7733667765918677	0.0010159828684040203	0.031874486166901896
Recall	0.7486188371292599	0.0008790795445305328	0.029649275615612144
F1	0.7481786732699027	0.001060794182190986	0.032569835464598004

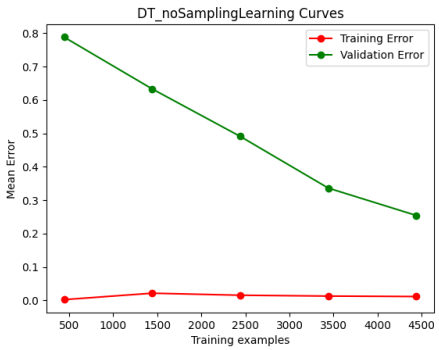
Tabella 5: Risultati delle metriche di valutazione per il Gradient Boosted Tree

Nella Figura 2, invece, sono mostrate le curve di apprendimento per ogni modello.

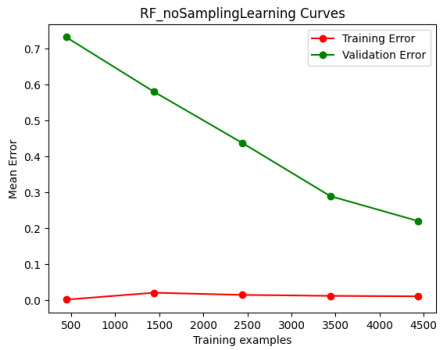
Valutazione: Già dai primi risultati è possibile osservare un paio di aspetti interessanti. Dalle pessime performance della **Logistic Regression** è possibile capire che il fenomeno analizzato, e che i modelli dovrebbero apprendere, **non segue un andamento lineare in funzione delle input features che abbiamo predisposto**. Sia la curva di apprendimento che i risultati delle metriche di valutazione ci permettono di capire che la Logistic Regression è fortemente in **underfitting**. Questo comporta che il modello non è abbastanza espressivo da catturare a pieno il comportamento che noi vogliamo che imparasse. Questo



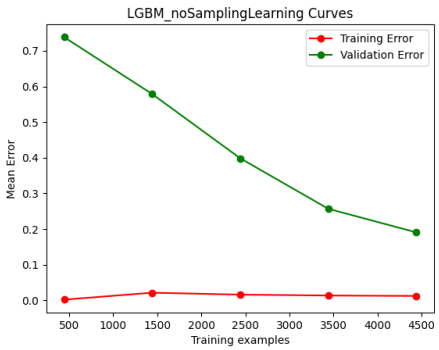
(a) Logistic Regression Learning Curve



(b) Decision Tree Learning Curve



(c) Random Forest Learning Curve



(d) LGBM Learning Curve

Figura 2: Curve di apprendimento per ogni modello

è principalmente dovuto alla **non linearità** del fenomeno in base alle sole input features predisposte. Probabilmente la Logistic Regression potrebbe performare meglio se le input features vengono estese con delle features la cui introduzione potrebbe rendere il fenomeno linearmente dipendente da esse. Questa operazione di **feature augmentation**, tuttavia, non è stata effettuata poichè, in seguito, si è deciso di utilizzare un altro modello, una rete neurale, per cercare di catturare la non linearità del fenomeno. Al contrario della Logistic Regression, tutti gli altri modelli sembrano comportarsi piuttosto bene con i dati e le input features a nostra disposizione. Questo è dovuto al fatto che si basano sugli alberi di decisione i quali sono piuttosto adatti ed efficaci ad approssimare anche comportamenti non lineari in base alle input features predisposte. Non ci sono evidenti segnali di **overfitting** e, inoltre, possiamo facilmente notare, dai valori della **Geometric Mean**, come un po' tutti i modelli si comportano abbastanza bene in presenza di sbilanciamento delle classi target. Dai valori medi, per ciascun valore della target feature, delle metriche di **Average Precision**, **Precision**, **Recall** e **F1** possiamo notare come un po' tutti i modelli non riescono a catturare un pattern che riesca a generalizzare molto bene per ciascun valore della target feature. Da quest'ultima osservazione possiamo notare che esisteranno dei valori delle target feature per i quali, i modelli, faranno fatica a fare delle previsioni affidabili. Se combiniamo tali informazioni anche con dei valori abbastanza alti della Geometric Mean possiamo ipotizzare che questo problema sia dovuto di più alla mancanza di un numero adeguato di dati nel dataset, più che allo sbilanciamento in esso. Probabilmente il numero di dati a nostra disposizione non è abbastanza alto, e anche bilanciato, da rendere il dataset abbastanza rappresentativo da cui poter catturare il fenomeno analizzato. Questo problema potrebbe essere risolto eventualmente raccogliendo altri dati e aggiungendoli al dataset ma a causa della scarsità di dati di questo tipo si dovrebbe trovare un'altra soluzione. Una possibile soluzione potrebbe consistere nella generazione artificiale di esempi, come l'**Over-Sampling**. Da queste prime

valutazioni si può già prevedere che i risultati dell'applicazione di una tecnica di **Under-Sampling** potrebbero essere catastrofici. Nelle sezioni di seguito sono state sperimentate tecniche standard di Over/Under-Sampling e nella sezione di sviluppi futuri sarà descritto un modo per effettuare tali tecniche, forse, in modo intelligente e più adatto al nostro caso. Da questi primi risultati possiamo notare che, in generale, i modelli classici di apprendimento automatico non riescono a superare la precisione massima di circa 0.81. Un altro motivo per il quale i modelli non riescono a superare questa accuratezza massima è la forte assunzione fatta nella sezione di "Analisi del dominio di interesse". Come già accennato in tale sezione, infatti, le features di un team Pokémon prese, in considerazione, non sono le uniche che influenzano la scelta del tipo del Pokémon rimanente e, quindi, non sono le uniche che influenzano in generale il team building. Alcune feature mancanti che potrebbero cambiare radicalmente le sorti della predizione sono:

1. Gli item posseduti dai Pokémon nella squadra e utilizzati in battaglia
2. Le abilità possedute dai Pokémon e utilizzate in battaglia

Queste sono quelle che più influenzerebbero il risultato della predizione. Le altre features, volutamente ignorate, nella fase di analisi del dominio di interesse si può prevedere che influenzano molto marginalmente i risultati delle predizioni. La decisione di ignorare determinate features è stata presa a causa della notevole difficoltà e del notevole sforzo necessario per integrare tali informazioni in una rappresentazione delle features adatta e non estremamente pesante. Per integrare le informazioni relative agli item e alle abilità avremmo dovuto implementare moduli di Natural Language Processing che in automatico avrebbero estratto, dalla sola descrizione degli item e delle abilità, tutte gli effetti in formato strutturato. Tutto sommato, comunque, anche facendo a meno di tali informazioni, i modelli si sono comportati piuttosto bene. Questo ci permette di avere una visione ottimistica, suggerendoci che un eventuale introduzione di tali informazioni mancanti potrebbe migliorare notevolmente le performance dei modelli. Questo task, tuttavia, è stato lasciato come un possibile sviluppo futuro. Complessivamente possiamo, facilmente, dire che il modello che si è comportato meglio fra tutti è il **Gradient Boosted Tree di LightGBM**. I valori di varianza e deviazione standard ci suggeriscono che risultati sono piuttosto affidabili.

6 Neural Networks e Deep Learning

Oltre ai modelli classici di apprendimento supervisionato, è stata addestrata, anche, una rete neurale con le stesse metodologie di addestramento e valutazione utilizzate per i precedenti modelli di apprendimento automatico. Gli iperparametri sperimentati, con i relativi range di valori e relativa descrizione, sono i seguenti:

1. **solver**: [*lbfgs, adam, sgd*]. Il solutore per l'ottimizzazione dei pesi.
2. **activation**: [*relu, logistic, tanh*]. Funzione di attivazione utilizzata negli hidden layers.
3. **early_stopping**: True. Indica se usare l'arresto anticipato per terminare l'addestramento quando il punteggio di validazione non migliora. Se impostata su True, verrà automaticamente accantonato il 10% dei dati di addestramento come validazione e si terminerà l'addestramento quando il punteggio di validazione non migliorerà di almeno tol per n_iter_no_change epoche consecutive.
4. **validation_fraction**: [0.1, 0.2]. Frazione del dataset predisposta per il validation set se è stato impostato l'early stopping.
5. **tol**: [0.1, 0.01, 0.001]. Tolleranza per l'ottimizzazione.
6. **alpha**: [1.0, 0.1, 0.01]. Costante moltiplicativa del termine di regolarizzazione L2.
7. **hidden_layer_sizes**: (300, 296). L'i-esimo elemento della lista rappresenta la dimensione dell'i-esimo hidden layer.
8. **max_iter**: 1000. Numero massimo di iterazioni effettuabili durante la fase di addestramento.

I migliori iperparametri trovati sono mostrati nella Tabella 6. I risultati delle metriche di valutazione sono mostrati nella Tabella 7. In Figura 3 è mostrata la curva di apprendimento della rete neurale.

Iperparametro	**Valore**
MLP__activation	relu
MLP__alpha	0.01
MLP__early_stopping	True
MLP__hidden_layer_sizes	(300, 296)
MLP__max_iter	1000
MLP__random_state	42
MLP__solver	lbfgs
MLP__tol	0.001
MLP__validation_fraction	0.1

Tabella 6: Iperparametri migliori per la Rete Neurale

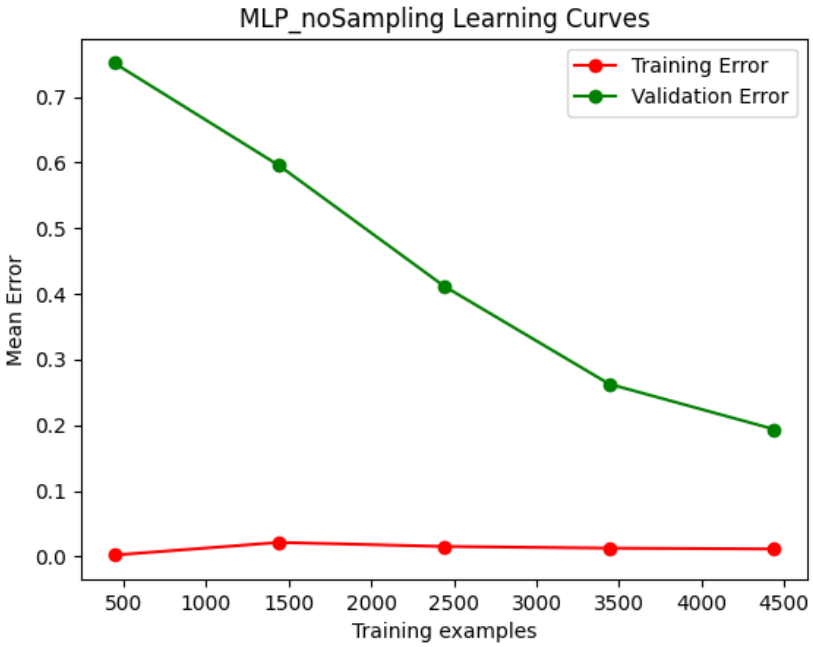


Figura 3: Curva di apprendimento della Rete Neurale

Metric	Score Mean	Score Variance	Score Std
Accuracy	0.8090417690417689	5.7408134066611094e-05	0.007576815562398961
Balanced Accuracy	0.7446144860950096	0.0005797130288754881	0.024077230506756545
Geometric Mean	0.8611580558229853	0.0001940552992131004	0.013930373261801006
Average Precision	0.7670920432699965	0.0010496185389689747	0.03239781688584857
Precision	0.7472201956325595	0.0012646471904866198	0.03556187833181228
Recall	0.7446144860950096	0.0005797130288754881	0.024077230506756545
F1	0.7348407416616047	0.000895144259314987	0.02991896153470215

Tabella 7: Risultati delle metriche di valutazione per la rete Neurale

Valutazione: I risultati della Rete Neurale sono quelli che più si avvicinano alle performance migliori del modello di LightGBM. Dalla curva di apprendimento sembra che non ci siano situazioni di over-fitting o under-fitting. In generale, anche qui, dalla curva di apprendimento si può notare che il modello abbia bisogno di ancora più dati per stabilizzarsi definitivamente e che l'aumento di essi potrebbe quasi sicuramente portare ad un notevole aumento delle performance. Le performance sono molto simili al Gradient Boosted Tree di LightGBM tuttavia sembrano essere leggermente più affidabili poichè hanno una varianza e deviazione standard leggermente inferiore. Le differenze, per ora, tra i due modelli sono impercettibili. Anche la rete neurale risulta comportarsi bene in presenza di sbilanciamento delle classi, confrontando la Geometric Mean e l'Accuracy. Anch'essa, tuttavia, sarà più debole nelle predizioni di determinate classi target così come le macro medie delle metriche di Average Precision, Precision, Recall e F1 per ciascuna classe target ci mostrano. Anche in questo caso la generazione artificiale di esempi potrebbe aumentare notevolmente le performance del modello e, pertanto, avrebbe senso sperimentare l'Over-Sampling.

7 Tecniche di Over-Sampling

Quando una delle classi è sottorappresentata rispetto alle altre, un modello addestrato su un dataset sbilanciato potrebbe essere parzialmente o completamente incline verso la classe maggioritaria, trascurando la classe minoritaria. Le tecniche di Over Sampling mirano a risolvere questo problema generando nuovi esempi dalla classe minoritaria, in modo che le classi siano più bilanciate. Come abbiamo già osservato, un po' tutti i modelli classici di apprendimento supervisionato non riescono a superare il limite massimo di accuratezza di 0.81 e, questo, può essere dovuto, anche, alla poca disponibilità di dati nel dataset. In questa sezione verrano esplorate le principali tecniche standard di Over-Sampling per cercare di generare artificialmente esempi di training con il quale poter combattere sia lo sbilanciamento delle classi sia la scarsità di dati. In generale, si è prestata molta attenzione all'implementazione di tali tecniche in modo da non "barare" nei risultati finali di valutazione. Le tecniche di Over-Sampling, proposte di seguito, sono state applicate solo al training set e non, anche, al test set in modo tale da evitare improvvisi picchi di performance dovuti principalmente al fatto che i modelli avrebbero facilmente predetto gli esempi generati artificialmente. Applicando tali tecniche al solo training set è possibile valutare se effettivamente, tali tecniche, migliorano le performance sugli esempi di nostro interesse e non su quelli artificiali creati. Le tecniche di Under-Sampling non sono state esplorate per i seguenti principali motivi:

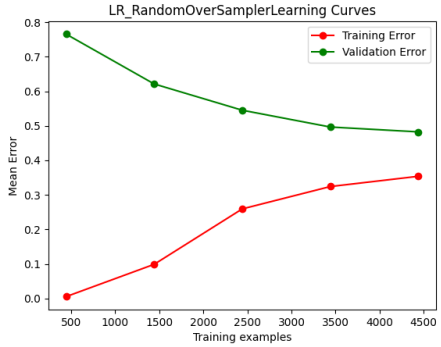
1. Effettuare Under Sampling, in questa situazione, con già un numero irrisorio di esempi di base per ogni classe target porterebbe ad un notevole peggioramento delle performance dei modelli. Questo è dovuto dal fatto che gli esempi a disposizione per ciascuna target class sono già molto pochi per molte classi target.

2. Avere anche un solo esempio in più, in questo caso, è molto più importante che averne di meno. Poichè, in generale, i team costruiti da giocatori competitivi Pokémon sono gli unici veri dati che possono permettere ai modelli di apprendere pattern di team building interessanti, avere anche un solo altro esempio in più di team costruito da un giocatore competitivo risulta avere un valore molto maggiore rispetto al bilanciamento delle classi in sè anche perchè, come abbiamo già visto e accennato, i modelli si comportano piuttosto bene anche in presenza di sbilanciamento.

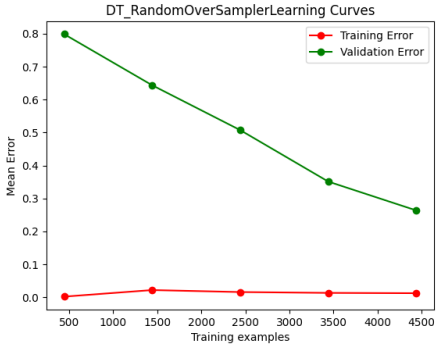
Il nostro problema maggiore, dunque, è combattere la scarsità dei dati con l'Over Sampling. Esistono principalmente due tecniche standard di Over-Sampling: **RandomOverSampling** e **SMOTE** (Synthetic Minority Oversampling Technique), implementate nella libreria di imbalanced-learn[1]. In tale libreria esistono, anche diverse varianti di SMOTE come **ADASYN** (Adaptive Synthetic), **BorderlineSMOTE**, **KMeansSMOTE** e **SVMSMOTE**. In fase di sviluppo sono state provate un po' tutte le tecniche precedentemente citate ma, per facilitare la lettura, si mostrano soltanto i risultati delle due principali tecniche che hanno performato meglio: RandomOverSampling e SMOTE. In ogni caso nella cartella `./logs` a partire dalla root directory del progetto ci sono tutti i file di log dei risultati degli esperimenti effettuati.

7.1 Random Over Sampling

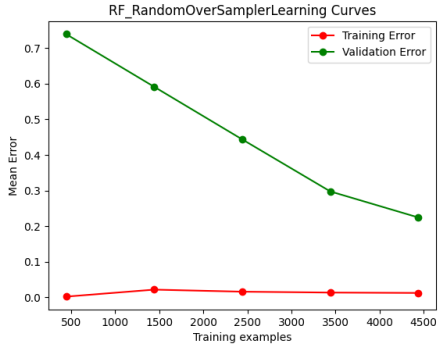
Il RandomOverSampling genera artificialmente nuovi esempi della target class minoritaria duplicando randomicamente degli esempi nell'insieme di esempi di quella classe target. Questa tecnica non è da sottovalutare poichè, in certi casi, potrebbe rafforzare i modelli nell'apprendere determinati pattern rari semplicemente duplicandone i loro esempi. Essa risulta essere anche piuttosto adatta al nostro dominio di interesse poichè, come vedremo, le tecniche di over-sampling basate su SMOTE non si dimostreranno abbastanza adatte al task



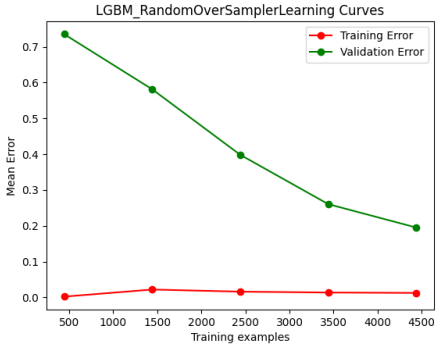
(a) Logistic Regression con Random Over Sampling



(b) Decision Tree con Random Over Sampling



(c) Random Forest con Random Over Sampling



(d) LGBM con Random Over Sampling

Figura 4: Curve di apprendimento per ogni modello

a causa della rappresentazione delle input features adottata. Dalla Tabella 8 alla Tabella 12 vengono mostrati i risultati dell'applicazione del **Random Over Sampling**. In Figura 4 e 5 sono mostrate le varie curve di apprendimento per ogni modello.

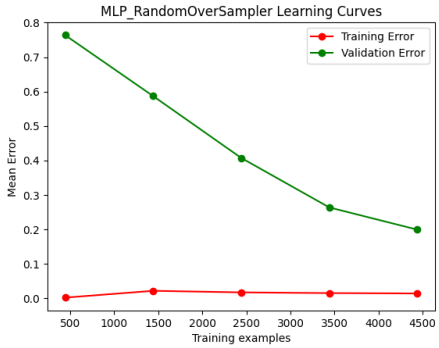


Figura 5: Rete Neurale con Random Over Sampling

Metric	Score Mean	Score Variance	Score Std
Accuracy	0.5195452518224796	8.141733175078919e-05	0.009023155310133434
Balanced Accuracy	0.5862236908357524	0.0005013475452549214	0.022390791528101934
Geometric Mean	0.7618804830232281	0.00021221246281766219	0.014567513954606743
Average Precision	0.5271394142303683	0.0007255622686200581	0.026936263078238192
Precision	0.4836040816598254	0.0013253849394297818	0.036405836612139295
Recall	0.5862236908357524	0.0005013475452549214	0.022390791528101934
F1	0.5023137077762609	0.0008871163581710605	0.029784498622119873

Tabella 8: Risultati delle metriche di valutazione per il Regressore Logistico con RandomOverSampling

Metric	Score Mean	Score Variance	Score Std
Accuracy	0.7341001127139741	7.833427289081163e-05	0.008850665110081368
Balanced Accuracy	0.6960561084974117	0.0007286716610123529	0.026993918963580537
Geometric Mean	0.8319308252222382	0.0002625466064999524	0.016203289989997476
Average Precision	0.5336429772866168	0.00031132414751154164	0.017644380054610637
Precision	0.682562326587832	0.0008177965733098464	0.02859714274730688
Recall	0.6960561084974117	0.0007286716610123529	0.026993918963580537
F1	0.6779924170784701	0.0008002737494446084	0.028289110085766365

Tabella 9: Risultati delle metriche di valutazione per i Decision Tree con RandomOverSampling

Metric	Score Mean	Score Variance	Score Std
Accuracy	0.778954922519279	0.00013514869208305777	0.011625346966136442
Balanced Accuracy	0.7271463068705598	0.0009280733079781147	0.030464295625832458
Geometric Mean	0.8506661731104807	0.00032009338925911726	0.017891153938723942
Average Precision	0.7560124009841257	0.0009004154237305641	0.03000692293006006
Precision	0.721669456209513	0.002373898298470422	0.04872266719372434
Recall	0.7271463068705598	0.0009280733079781147	0.030464295625832458
F1	0.711653169353727	0.0014761314474204708	0.038420456106356554

Tabella 10: Risultati delle metriche di valutazione per il Random Forest con RandomOverSampling

Metric	Score Mean	Score Variance	Score Std
Accuracy	0.807057192205707	0.00019905279229327763	0.014108607028806126
Balanced Accuracy	0.738686005939355	0.00084164559512101	0.029011128815008388
Geometric Mean	0.8576364948583028	0.0002869567124644109	0.01693979670670256
Average Precision	0.7978364971739234	0.0011492416983475656	0.033900467524026354
Precision	0.7873303274373469	0.001001135545965881	0.03164072606571918
Recall	0.738686005939355	0.00084164559512101	0.029011128815008388
F1	0.7466780264499906	0.0009763733052105671	0.031246972736739907

Tabella 11: Risultati delle metriche di valutazione per il Gradient Boosted Tree di LightGBM con RandomOverSampling

Metric	Score Mean	Score Variance	Score Std
Accuracy	0.8043569221787044	0.00012607546972800477	0.011228333345960333
Balanced Accuracy	0.7495648072776946	0.0008825208689713116	0.029707252800811313
Geometric Mean	0.8639129421033693	0.0002960518876782245	0.0172061584230247 2
Average Precision	0.7717808165168208	0.0006916426495069905	0.026299099785106535
Precision	0.7606274301986499	0.0021374874824044337	0.04623296964725967
Recall	0.7495648072776946	0.0008825208689713116	0.029707252800811313
F1	0.7431392278304604	0.0013617776929464765	0.03690227219218996

Tabella 12: Risultati delle metriche di valutazione per la Rete Neurale con Random Over Sampling

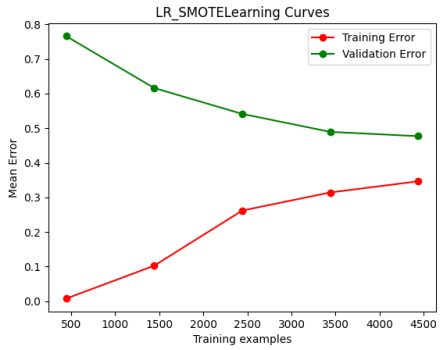
Valutazione: Da questi primi risultati possiamo già notare come il RandomOverSampling ha cambiato veramente poco i risultati. Alcune metriche sono migliorate di differenze quasi impercettibili mentre altre sono leggermente peggiorate. Il RandomOverSampling non ha portato ad alcun miglioramento e, per questo, è stato un fallimento totale ma da questo esperimento si può già intuire che, forse, un po' tutte le tecniche standard di Over-Sampling potrebbero non essere efficaci a causa della rappresentazione delle features adottata. Non sono stati riportati i valori degli iperparametri migliori poichè l'esperimento di per sè è fallito.

7.2 SMOTE, sue varianti e tecniche miste

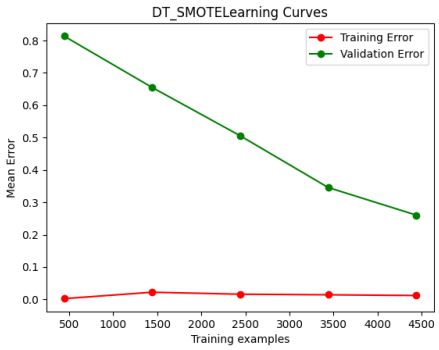
SMOTE genera nuovi esempi artificiali partendo da un esempio di base, della target class minoritaria, e variandone leggermente i suoi valori in base alla sua differenza rispetto ad un altro esempio della stessa classe target (Eventuali spiegazioni sul funzionamento di SMOTE si possono trovare nella guida di imbalanced-learn[1]). I risultati delle varianti non verrano mostrate tuttavia saranno, comunque, accessibili a partire della root directory del progetto al percorso relativo `./logs` in formato di file log testuale. Per poter applicare la tecnica di over sampling SMOTE è stato, prima, necessario implementare una piccola classe custom di "over sampling" in modo da poter permettere la corretta esecuzione di SMOTE. L'algoritmo 9 mostra il codice python dell'algoritmo di Random Over Sampling custom definito ed eseguito prima dell'esecuzione dell'over sampling di SMOTE. Il metodo `fit_resample()` semplicemente effettua un Random Over Sampling su tutte quelle classi che occorrono meno di `threshold` volte. In questo modo il Random Over Sampler andrà a generare esempi artificiali, per le classi target minoritarie, finchè non ci saranno almeno `threshold` esempi per ognuna di esse ignorando le altre classi target. Questa operazione è necessaria poichè il numero `k_neighbours` utilizzato, per l'applicazione di SMOTE, è pari a 5 e molte classi target occorrono meno di 25 volte il quale è il numero minimo di occorrenze affinchè, dopo aver effettuato la suddivisione in 5 fold per la Cross Validation, per ogni fold ci siano almeno 5 vicini da cui poter eseguire SMOTE. Le Tabelle dalla 13 alla 17 mostrano i risultati dell'applicazione di SMOTE. In Figura 6 e 7 sono mostrate le curve di apprendimento dei modelli con l'applicazione di SMOTE.

Algorithm 9 Algoritmo di generazione artificiale di esempi

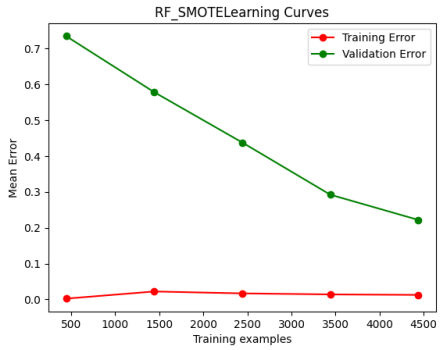
```
class CustomOverSampler():

    def __init__(self, threshold):
        self.threshold = threshold

    def fit_resample(self, X, y):
        y = list(y)
        seed = 42
        occurrences = {item: y.count(item) for item in y}
        min_samples = {}
        for target in occurrences.keys():
            if occurrences[target] < self.threshold:
                min_samples[target] = self.threshold
        ros = RandomOverSampler(sampling_strategy=min_samples,
                                random_state=seed, shrinkage=0)
        return ros.fit_resample(X, y)
```



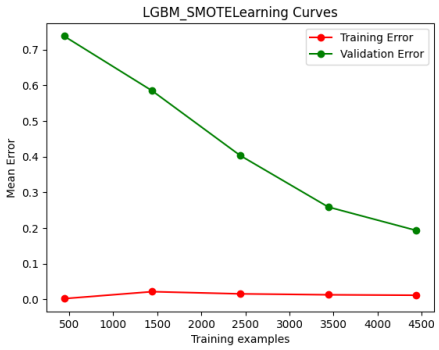
(a) Logistic Regression con SMOTE



(b) Decision Tree con SMOTE



(c) Random Forest con SMOTE



(d) LGBM con SMOTE

Figura 6: Curve di apprendimento per ogni modello

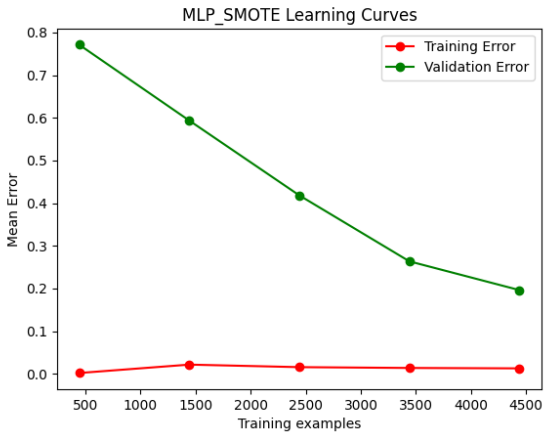


Figura 7: Rete Neurale con SMOTE

Metric	Score Mean	Score Variance	Score Std
Accuracy	0.522969323959423	3.638046266810086e-05	0.006031621893661842
Balanced Accuracy	0.5693139288337228	0.0006217003225104645	0.02493391911654613
Geometric Mean	0.7507848655318519	0.0002733606050399989	0.0165336204456253
Average Precision	0.5241544829096164	0.000920931693120244	0.030346856396013144
Precision	0.48521671417039364	0.001146114221819477	0.03385430876298432
Recall	0.5693139288337228	0.0006217003225104645	0.02493391911654613
F1	0.5013990183687582	0.0008038575010533301	0.028352380870983834

Tabella 13: Risultati delle metriche di valutazione per i Regressori Logistici con SMOTE

Metric	Score Mean	Score Variance	Score Std
Accuracy	0.7362601665571964	0.0002008737444684253	0.014172993490029736
Balanced Accuracy	0.6993216975347498	0.0005847863807694105	0.024182356807586198
Geometric Mean	0.8339329722722674	0.00020997467685596543	0.01449050298837019
Average Precision	0.5411525991703806	0.0011534184084401042	0.033962014198809
Precision	0.6969572053245029	0.002292457249517407	0.04787961204434939
Recall	0.6993216975347498	0.0005847863807694105	0.024182356807586198
F1	0.6848931460650626	0.0011918239637004797	0.03452280353187556

Tabella 14: Risultati delle metriche di valutazione per i Decision Tree con SMOTE

Metric	Score Mean	Score Variance	Score Std
Accuracy	0.7767929225354968	0.00020660465001370014	0.014373748641662694
Balanced Accuracy	0.724233312479102	0.0006639194717402333	0.025766634854792997
Geometric Mean	0.8489960975644198	0.00023166589301304384	0.015220574661064669
Average Precision	0.7496669299146138	0.0009788863237898185	0.031287159087872114
Precision	0.7095688523438787	0.0016097683987322687	0.04012191918057097
Recall	0.724233312479102	0.0006639194717402333	0.025766634854792997
F1	0.7011719745910334	0.000881149009081206	0.029684154174933232

Tabella 15: Risultati delle metriche di valutazione per i Random Forest con SMOTE

Metric	Score Mean	Score Variance	Score Std
Accuracy	0.8079580931066079	0.00020906255879220656	0.014458995773988128
Balanced Accuracy	0.7419484991726322	0.0006382477713167013	0.025263566084713796
Geometric Mean	0.8595769844993424	0.00021813470256788593	0.014769383960337883
Average Precision	0.7953008118638902	0.0009359367016475578	0.03059308257837967
Precision	0.7910581377813479	0.0008499627047711311	0.029154119859312013
Recall	0.7419484991726322	0.0006382477713167013	0.025263566084713796
F1	0.7484907118039815	0.0006691159247192105	0.025867275169975102

Tabella 16: Risultati delle metriche di valutazione per i Gradient Boosted Tree di LightGBM con SMOTE

Metric	Score Mean	Score Variance	Score Std
Accuracy	0.8054389763300656	2.717011721120031e-05	0.005212496255269668
Balanced Accuracy	0.7433673622742966	0.0004359614606871771	0.020879690148255962
Geometric Mean	0.860425367982662	0.0001471379590461264 6	0.012130043653925011
Average Precision	0.7623712454325229	0.0006997966303067769	0.026453669505510514
Precision	0.7641874380566683	0.0015763345695491225	0.039703080101537745
Recall	0.7433673622742966	0.0004359614606871771	0.020879690148255962
F1	0.7416088226490738	0.0009333661915181823	0.03055104239658906

Tabella 17: Risultati delle metriche di valutazione per la Rete Neurale con SMOTE

Valutazione: Anche qui, come avevamo già immaginato, purtroppo, non ci sono grandi cambiamenti. Questo ci permette di capire che anche le altre varianti dello SMOTE non funzionerebbero poichè inadatte alla rappresentazione delle features scelta. Tali tecniche, infatti, poichè si basano sulla generazione di nuovi esempi secondo la formula $x_{new} = x_i + \lambda \times (x_{zi} - x_i)$, dove X_i è l'esempio di partenza, X_{zi} è uno dei `k_neighbours` λ è una costante moltiplicativa tra 0 e 1, generano degli esempi artificiali incoerenti con la rappresentazione scelta e, soprattutto, non generano degli esempi significativi poichè le operazioni di somma e sottrazione tra esempi con la rappresentazione adottata non hanno un vero e proprio senso logico poichè potrebbero portare alla generazione di esempi artificiali che sono fuori dominio (ad esempio potrebbero essere generati degli esempi in cui c'è un numero di Pokémon negativo di un particolare tipo). Da questi risultati, dunque, è stato possibile capire che le normali e standard tecniche di Over-Sampling non sono adatte ai nostri scopi e che, pertanto, bisognerebbe implementare un meccanismo di Over Sampling, o in generale di generazione artificiale di esempi, che generi esempi significativi e rappresentativi per il task di nostro interesse. Tale meccanismo è discusso nella sezione di possibili sviluppi futuri e, forse, è l'unico modo per effettuare dell'Over Sampling intelligente e che possa portare dei miglioramenti significativi. Se le normali e standard tecniche di Over Sampling sono inadatte, di conseguenza, saranno inadatte anche tutte le varianti di SMOTE quali **ADASYN**, **BorderlineSMOTE**, **KMeansSMOTE** e **SVMSMOTE** insieme a tutte quelle altre tecniche di sampling misto (che effettuano sia Over Sampling sia Under Sampling) che si basano anche su SMOTE quali **SMOTEENN** e **SMOTETomek**. Anche se implementate e valutate, nel progetto, si è deciso di non inserirne i risultati in questa documentazione poichè sono coerenti con tutto ciò che si è detto e poichè non cambiano, se non peggiorano, le performance dei modelli. In ogni caso è sempre possibile visionare i file di log situati nella cartella `./logs` a partire dalla root directory del progetto. In conclusione si può affermare che è sconsigliato, se non adirittura inutile o controproducente, utilizzare tali tecniche per questo task.

8 Apprendimento probabilistico

Com'è stato osservato nella sezione 7, le normali e standard tecniche di over-sampling non sono adatte per il nostro dataset di riferimento o, in generale, non sono riuscite a portare miglioramenti significativi. Di seguito, in questa sezione, si è deciso di esplorare le tecniche di apprendimento probabilistico per apprendere una Belief Network (o rete bayesiana) in modo tale da estrarre dal dataset quante più informazioni possibili sulla distribuzione dei campioni in esso e cercare di generare esempi artificiali credibili e che possano migliorare le prestazioni dei modelli di apprendimento supervisionato. L'obiettivo, dunque, è quello di generare esempi artificiali credibili in modo da poter effettuare dell'over-sampling intelligente nella pipeline di addestramento dei modelli.

8.1 Apprendimento della Rete Bayesiana

Un problema principale della rete bayesiana è lo spazio richiesto per poter apprendere tutte le probabilità che servono per poterla utilizzare. Date le risorse a nostra disposizione, si è deciso di scendere a compromessi tra la corrispondenza della rete bayesiana con la distribuzione reale (ground truth) e la sua complessità in spazio. Se volessimo apprendere una rete bayesiana che simula perfettamente, o che perlomeno si avvicina, alla vera distribuzione di probabilità del campione, le risorse computazionali a nostra disposizione non ce lo permetterebbero dato che sarebbe richiesto uno spazio di memoria in RAM nel'ordine delle decine, se non delle centinaia, di GigaBytes. Oltre alla difficoltà intrinseca dell'apprendimento della Rete Bayesiana, vi è pure la natura complessa del dataset a complicare le cose ulteriormente. Si è deciso, pertanto, di dare alla rete bayesiana una struttura arbitraria che risulta essere un compromesso fra la complessità in spazio e la sua vicinanza alla distribuzione reale del campione. Tale struttura risulta essere un'approssimazione molto forte e stringente della distribuzione del campione tuttavia risulta essere sempre meglio delle tecniche di over-sampling discusse in precedenza e, pertanto, vale la pena sperimentarle.

Le feature prese in considerazione sono le stesse del dataset originale ma in formato diverso. Le feature sono: *Pk1_types*, *Pk2_types*, *Pk3_types*, *Pk4_types*, *Pk5_types*, *Target* (che indicano rispettivamente la combinazione di tipi primario e secondario del primo, secondo, terzo, ..., quinto pokemon e sesto pokemon denominato Target) e feature nel formato *Pk(i)_move(j)_type* indicano il tipo della j-esima mossa dell'i-esimo pokemon dove $1 <= i <= 5, 1 <= j <= 4$. Si è deciso di fissare la struttura della rete bayesiana nel seguente modo: tutte le feature *Pk1_types*, *Pk2_types*, *Pk3_types*, *Pk4_types*, *Pk5_types* riguardanti le combinazioni di tipi primari e secondari dei pokemon in input sono state rese dipendenti dalla combinazione di tipi del pokemon *Target*. Dopodichè le $1 <= j <= 4$ mosse che si riferiscono all'i-esimo pokemon sono state rese dipendenti dalla combinazione di tipi dell'i-esimo pokemon. Questa struttura ci permette di creare facilmente degli esempi a partire da un particolare valore della feature target di cui vogliamo aumentare la numerosità nel dataset. La libreria utilizzata per l'apprendimento della rete bayesiana è `pgmpy`. In figura 8 si riporta la struttura della rete bayesiana.

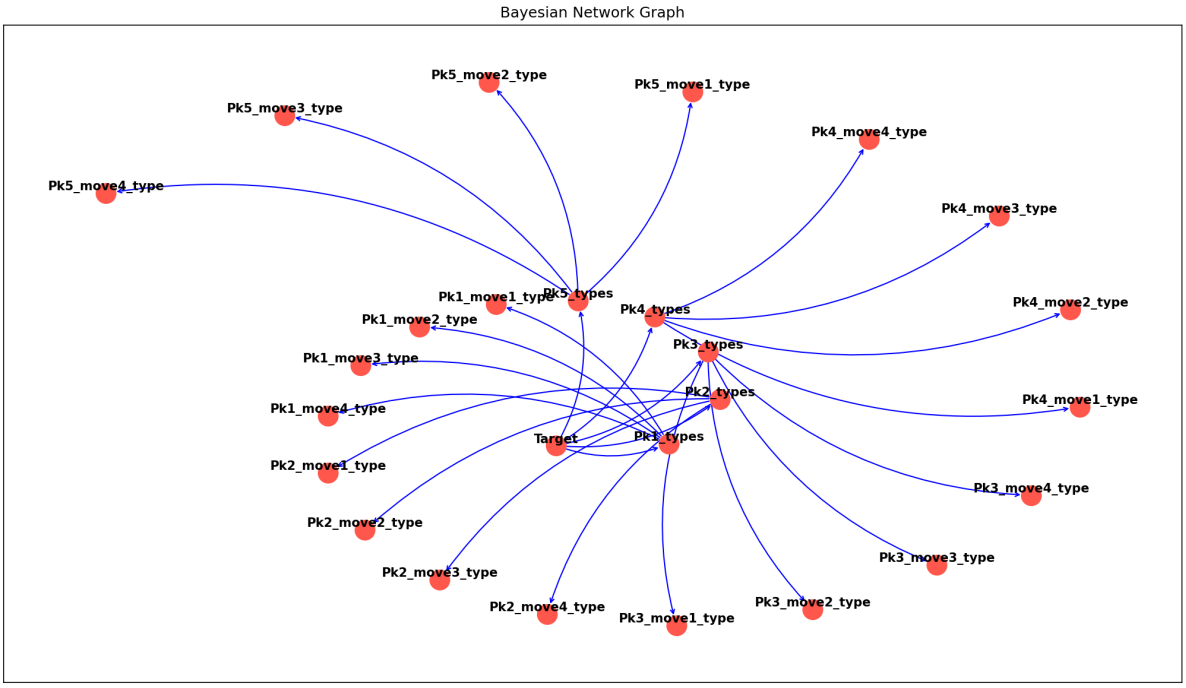


Figura 8: Struttura della rete bayesiana

A partire dalla root directory del progetto al percorso `./bn_results/cpds.txt` sono presenti i risultati delle Conditional Probabilities Distributions (o Tables). Di seguito si riportano soltanto alcuni esempi di CPD. Nella tabella 18 viene riportata la CPD della feature *Target* mentre nella tabella 19 la CPD della feature *Pk1_types*.

Tabella 18: Target CPD

Target	Probability
Target(Bug Fire)	0.014727
Target(Bug Ghost)	0.000359195
Target(Bug Poison)	0.000359195
Target(Bug Rock)	0.000179598
Target(Bug Steel)	0.0289152
Target(Dark Dragon)	0.00862069
Target(Dark Fighting)	0.000359195
Target(Dark Flying)	0.000179598
Target(Dark Ghost)	0.00323276
Target(Dark Ice)	0.00538793
Target(Dark Steel)	0.000179598
Target(Dark null_type)	0.000359195
...	...
Target(Water Poison)	0.0217313
Target(Water Psychic)	0.0466954
Target(Water Steel)	0.000897989
Target(Water null_type)	0.0427443

Tabella 19: CPD of Pk1_types

Target	...	Target(Water null_type)
Pk1_types(Bug_Fire)	...	0.03361344537815126
Pk1_types(Bug_Ghost)	...	0.008403361344537815
Pk1_types(Bug_Poison)	...	0.0
Pk1_types(Bug_Steel)	...	0.01680672268907563
Pk1_types(Dark_Dragon)	...	0.0
Pk1_types(Dark_Ghost)	...	0.0
Pk1_types(Dark_Ice)	...	0.0
Pk1_types(Dark_null_type)	...	0.0
Pk1_types(Dragon_Flying)	...	0.008403361344537815
Pk1_types(Dragon_Ground)	...	0.0546218487394958
Pk1_types(Dragon_Ice)	...	0.03361344537815126
Pk1_types(Dragon_Psychic)	...	0.06302521008403361
...	...	...
Pk1_types(Water_Ghost)	...	0.004201680672268907
Pk1_types(Water_Ground)	...	0.02100840336134454
Pk1_types(Water_Poison)	...	0.04201680672268908
Pk1_types(Water_Psychic)	...	0.058823529411764705
Pk1_types(Water_null_type)	...	0.07142857142857142

8.2 Generazione di esempi sintetici e inferenza probabilistica

La rete bayesiana appresa permette di generare esempi credibili del campione analizzato in modo tale da poter supportare la pipeline di addestramento supervisionato dei modelli. In tabella 20 si riporta un esempio generato dalla rete sfruttando le probabilità apprese.

Pk1_types	**Pk2_types**	**Pk3_types**	**Pk4_types**
Psychic_null_type	Ground_Flying	Dragon_Psychic	Fighting_null_type
Pk5_types	**Pk1_move1_type**	**Pk1_move2_type**	**Pk1_move3_type**
Electric_Flying	Rock	Ground	Dark
Pk1_move4_type	**Pk2_move1_type**	**Pk2_move2_type**	**Pk2_move3_type**
Bug	Normal	Rock	Ground
Pk2_move4_type	**Pk3_move1_type**	**Pk3_move2_type**	**Pk3_move3_type**
Normal	Dragon	Psychic	Dragon
Pk3_move4_type	**Pk4_move1_type**	**Pk4_move2_type**	**Pk4_move3_type**
Psychic	Fighting	Rock	Bug
Pk4_move4_type	**Pk5_move1_type**	**Pk5_move2_type**	**Pk5_move3_type**
Ice	Psychic	Electric	Ice
Pk5_move4_type	**Target**		
Fighting	Bug		

Tabella 20: Esempio artificiale generato

Generati esempi di questo tipo, successivamente, potranno essere convertiti in esempi nello stesso formato del dataset utilizzato nella pipeline dell'apprendimento supervisionato per poi essere inseriti in quest'ultimo in modo tale da effettuare un over-sampling intelligente del dataset per combattere lo sbilanciamento e la mancanza di dati.

La rete bayesiana permette, anche, di ottenere la distribuzione di probabilità di una certa variabile a partire da delle evidenze osservate. Questo può essere utile in caso di sviluppi futuri in cui la generazione intelligente di samples viene approfondita di più. Di seguito si riportano alcune query di esempio con l'output tagliato per chiarezza visiva:

1. **Query 1**: "Sapendo che il Pokemon Target è di tipo Bug Fire, qual'è la distribuzione di probabilità del tipo del Pokemon1 in squadra?".

   Codice:

   ```
   execute_and_print_query(inference, variables=['Pk1_types'],
      evidence={'Target': 'Bug Fire'})
   ```

   Risultato:

   ```
   +--------------------------------+------------------+
   | Pk1_types                      |   phi(Pk1_types) |
   +================================+==================+
    . . .
   +--------------------------------+------------------+
   | Pk1_types(Dark_Ice)            |           0.0122 |
   +--------------------------------+------------------+
   | Pk1_types(Dragon_Ground)       |           0.2317 |
   +--------------------------------+------------------+
   | Pk1_types(Dragon_Psychic)      |           0.0244 |
   +--------------------------------+------------------+
   | Pk1_types(Fire_null_type)      |           0.0854 |
   +--------------------------------+------------------+
   | Pk1_types(Ground_Flying)       |           0.0366 |
   +--------------------------------+------------------+
   | Pk1_types(Ice_Ghost)           |           0.1220 |
   +--------------------------------+------------------+
   | Pk1_types(Psychic_null_type)   |           0.0122 |
   +--------------------------------+------------------+
   | Pk1_types(Rock_Dark)           |           0.0488 |
   +--------------------------------+------------------+
   | Pk1_types(Steel_Flying)        |           0.0488 |
   +--------------------------------+------------------+
   | Pk1_types(Water_Psychic)       |           0.0488 |
   +--------------------------------+------------------+
   | Pk1_types(Water_null_type)     |           0.1341 |
   +--------------------------------+------------------+
   . . .
   ```

2. **Query 2**: "Sapendo che il tipo del Pokemon1 in squadra è Water_null_type qual'è la distribuzione di probabilità del tipo della sua prima mossa?".

   Codice:

   ```
   execute_and_print_query(inference, variables=['Pk1_move1_type'],
      evidence={'Pk1_types': 'Water_null_type'})
   ```

   Risultato:

Pk1_move1_type	phi(Pk1_move1_type)
Pk1_move1_type(Bug)	0.0000
Pk1_move1_type(Dark)	0.0000
Pk1_move1_type(Dragon)	0.0000
Pk1_move1_type(Electric)	0.0000
Pk1_move1_type(Fighting)	0.0000
Pk1_move1_type(Fire)	0.0000
Pk1_move1_type(Flying)	0.0000
Pk1_move1_type(Ghost)	0.0000
Pk1_move1_type(Grass)	0.0000
Pk1_move1_type(Ground)	0.0000
Pk1_move1_type(Ice)	0.0000
Pk1_move1_type(Normal)	0.0359
Pk1_move1_type(Poison)	0.0785
Pk1_move1_type(Psychic)	0.0224
Pk1_move1_type(Rock)	0.0000
Pk1_move1_type(Steel)	0.0000
Pk1_move1_type(Water)	0.8632

Possiamo, infine, concludere che la rete bayesiana appresa risulta essere una buona base di partenza per generare degli esempi più credibili e soprattutto può essere utilizzata come strumento di supporto per lo stesso scopo (grazie alla possibilità di effettuare query su di essa sulle distribuzioni di probabilità di determinate features con determinate evidenze).

9 Conclusioni

In questo caso di studio è stato possibile esplorare tecniche di programmazione logica per risolvere determinati sotto-task complessi in maniera più semplice. Inoltre è stato possibile esplorare le principali tecniche di apprendimento supervisionato e profondo per risolvere il task di classificazione prefissato: predire la combinazione di tipo primario e secondario del sesto Pokémon a partire dagli altri cinque nel team. Sia le reti neurali che gli alberi di decisione, soprattutto i Gradient Boosted Trees di LightGBM, forniscono buone performance di predizione. Nel caso in cui l'interpretabilità dei risultati sia più importante, così come in casi di reccomender systems o di applicativi di team building, i modelli basati sugli alberi di decisione, principalmente il Gradient Boosted Tree di LightGBM, sono da preferire in quanto le loro predizioni sarebbero più interpretabili e facilmente spiegabili all'utente finale. Le reti neurali, invece, potrebbero essere utilizzate in contesti in cui l'explainability non sia importante. Inoltre è stato, anche, dimostrato come le tecniche di over-sampling standard e classiche risultino essere inefficaci con questo tipo di rappresentazione e come questi modelli necessitino di più dati per dare delle performance migliori. Infine abbiamo discusso sull'utilizzo di tecniche di generazione intelligente di esempi di training attraverso l'utilizzo delle reti bayesiane in modo da migliorare le performance dei modelli.

9.1 Sviluppi futuri

I tools e i dataset citati nella sezione della creazione del dataset permettono di sperimentare molte idee nell'ecosistema competitivo Pokémon. Di seguito sono elencati, in ordine di importanza, alcuni degli sviluppi futuri che possono estendere e migliorare notevolmente questo progetto:

1. Generazione intelligente di esempi di team competitivi di Pokémon sfruttando le tecniche studiate nel capitolo di Ragionamento con vincoli e Ricerca di Soluzioni in spazi di stati del libro di riferimento[2] che permettano di generare un team competitivo attraverso una ricerca euristica o greedy nello spazio di tutti i possibili team competitivi. Si potrebbe pensare di utilizzare come funzione di valutazione euristica le statistiche, o probabilità, fornite da Smogon o effettuando query sulla rete bayesiana appresa in questo caso di studio. Questa è una possibile soluzione alla scarsità di dati e potrebbe essere usata come tecnica di Over-Sampling o Data Augmentation per poter migliorare notevolmente le performance dei modelli.

2. Sviluppo di AutoEncoders che riescano ad apprendere come rappresentare le features di un team di Pokémon in uno spazio vettoriale di dimensione inferiore a quelle previste sin'ora. Questo permetterebbe una notevole riduzione della dimensionalità delle features in input e darebbe la possibilità di rappresentare e veicolare correttamente anche le features di input dei team Pokémon, ignorate in fase di analisi del dominio di interesse.

3. Completare la predizione del tipo del sesto Pokémon con le predizioni delle altre features significative dello stesso. Questo potrebbe avviare un sistema di reccomending o building di team di Pokémon.

4. Sviluppo di reti bayesiane e di modelli di apprendimento automatico per la predizione delle mosse che un avversario potrebbe effettuare in battaglia attraverso l'utilizzo dei replay di Pokémon Showdown come dataset. Gli argomenti del libro di riferimento[2] che potrebbero essere utili sono Apprendimento Supervisionato, Modelli di conoscenza incerta e Apprendimento e incertezza.

Riferimenti bibliografici

[1] The imbalanced-learn developers. User guide. https://imbalanced-learn.org/stable/user_guide.html, 2014-2024. Online; accessed 09-April-2024.

[2] David Poole and Alan Mackworth. *Artificial Intelligence: Foundations of Computational Agents*. Cambridge University Press, Cambridge, UK, 3 edition, 2023.

[3] scikit learn. Metrics and scoring: quantifying the quality of predictions. https://scikit-learn.org/stable/modules/model_evaluation.html, 2024. Online; accessed 09-April-2024.

[4] Pokémon Central Wiki. Generazioni pokémon. https://wiki.pokemoncentral.it/Generazione, 2024. Online; accessed 09-April-2024.