**Documentazione DNA-GymCraft**

Giuseppe Pio Sorrentino Antonio Albanese Marco Greco


LAURA TRIENNALE IN INFORMATICA UNIVERSITÀ DI SALERNO
CORSO DI FONDAMENTI DI INTELLIGENZA ARTIFICIALE
PROFESSORE FABIO PALOMBA


Link repository GitHub: https://github.com/Gmarco2706/DNAGYMCraft

# Indice

# 1 Definizione del problema

## 1.1 Introduzione

DNA-GymCraft è un algoritmo di intelligenza artificiale che nasce dall'idea di voler unire una passione di noi studenti con gli argomenti affrontati a lezione. Questo algoritmo va incontro ai coach di una palestra perchè permette di creare una scheda di allenamento personalizzata a seconda del cliente basandosi su una serie di esercizi presi in input da un dataset contente circa 3000 esercizi.
Questo algoritmo si basa molto anche sulle preferenze del cliente, esso infatti permette di modellare la scheda in base alle esigenze di quest'ultimo cercando di soddisfarlo il piu possibile utilizzando anche un sistema di feedback che permette di riprogrammare la scheda nel caso in cui il cliente non sia soddisfatto Per poter realizzare questo algoritmo è stato utilizzato un algoritmo genetico

## 1.2 Obiettivo

L'Obiettivo del progetto è quello di realizzare una scheda che aiuti il coach a consegnarle per tutti i clienti della palestra in breve tempo e trovare la scheda perfetta per un cliente. La scheda deve essere quanto più bilanciata possibile, deve cercare di assegnare, in una scheda di sette esercizi, la giusta combinazione di esercizi per una determinata categoria (gambe, braccia e petto) comprendendo sempre un esercizio di streatching all'inizio e diversificando gli esercizi seguenti, privilegiando l'assegnazione di esercizi con attrezzi se il cliente deve svolgere esercizi in palestra e in caso contrario assegnargli esercizi senza attrezzi per potergli permettere di svolgere la scheda assegnata senza problemi. Il feedback finale dell'utente si rivelerà poi molto importanto in quanto, in caso di feedback negativo, la scheda viene riformulata fin quando non riceverà un feedback positivo.

# Tool utilizzati

I tool da utilizzare per sviluppare il progetto sono:

Colab per l'implementazione dell'algoritmo Python come linguaggio di programmazione GitHub per condividere i lavori Kaggle per trovare il dataset adatto al problema Overleaf per la scrittura del documento del progetto Canva per la realizzazione della presentazione

## 2 Specifica PEAS

| **PEAS** | |
|---|---|
| **Performance** | La misura di performance dell'agente è la sua capacità di massimizzare l'efficacia delle schede di allenamento generate in base alle preferenze dell'utente. |
| **Environment** | L'ambiente in cui opera il nostro agente riguarda l'ambito ginnico. L'ambiente è:<br>**Dinamico** la dinamicità deriva dal fatto che l'agente (algoritmo genetico) interagisce con un ambiente in evoluzione, generando schede di allenamento in risposta alle preferenze e alle modifiche apportate dall'utente nel corso del tempo. L'ambiente è influenzato dalle scelte dell'utente, e l'agente si adatta continuamente per massimizzare le prestazioni delle schede di allenamento in base a tali preferenze e feedback.<br>**Episodico** in quanto un'azione intrapresa dall'agente in un dato istante non è influenzata dall'azione effettuata precedentemente.<br>**Totalmente osservabile** in quanto l'agente in ogni istante può avere una visione completa dell'ambiente in cui è calato.<br>**Noto** in quanto abbiamo informazioni sulle dinamiche e le regole che guidano il processo di generazione delle schede di allenamento.<br>**Singolo** in quanto l'ambiente prevede al proprio interno l'introduzione di un singolo agente.<br>Gli elementi dell'ambiente sono le schede di programmazione. |
| **Actuators** | L'agente agisce sull'ambiente tramite lo stream di output del nostro computer nel quale andrà generare una scheda di programmazione per una determinata persona |
| **Sensors** | L'agente percepirá l'ambiente tramite lo stream di input del nostro computer. |

# 3 Analisi dei dati

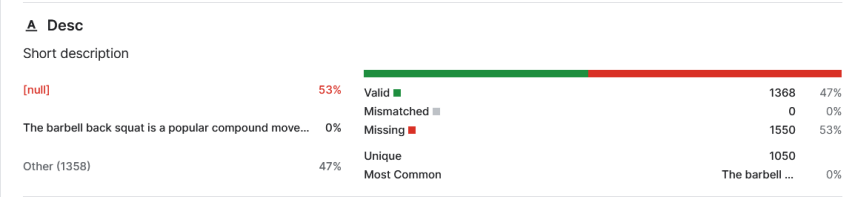
L'insieme degli esercizi è stato preso da un dataset su presente su Kaggle al seguente link https://www.kaggle.com/datasets/niharika41298/gym-exercise-data contenente circa 3000 esercizi al suo interno e per ogni esercizio memorizza

- titolo ovvero il nome dell'esercizio da svolgere
- descrizione ovvero una breve spiegazione dell'esercizio
- tipo di esercizio indica la tipologia di esercizio che viene svolto
- parte del corpo allenata
- equipaggiamento indica lo strumento da utilizzare per svolgere l'esercizio se l'esercizio ne necessita
- il livello di complessità per indicare quanto è complesso l'esercizio
- il rating dell'esercizio, una valutazione media che viene data all'esercizio
- la descrizione del rating, motivazione del voto assegnato

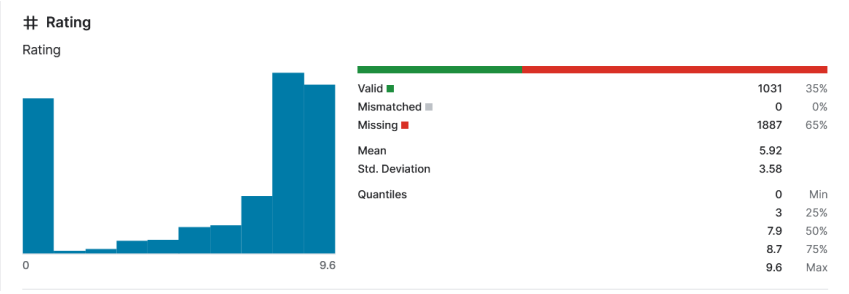
# 4 Pulizia dei dati

## 4.1 Rimozione colonne:

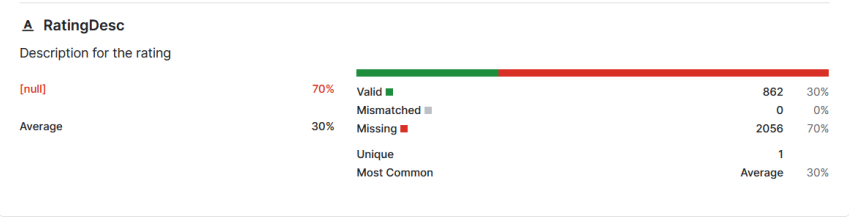
La colonna **Descrizione** viene rimossa in quanto la maggior parte dei campi in quella colonna sono null, risulta inoltre essere inutile per l'algoritmo in quanto non va a influire sulla creazione della scheda



La colonna **Rating** viene rimossa anche in questo caso perche più della meta dei campi sono vuoti e essendo ogni esercizio indipendente dagli altri abbiamo deciso di non applicare tecniche di data imputation come la media in quanto basarci su rating di altri esercizi avrebbe potuto sopravvalutare o sottovalutare esercizi




La colonna **RatingDesc** viene rimossa per un ragionamento analogo a quello della colonna descrizione e soprattuto perche eliminando la colonna Rating questa diventa inutile oltre a non essere influente e contenere per lo più valori nulli




- ## 4.2 Rimozione righe:

Le righe contenenti campi vuoti nelle colonne di : titolo, tipo di esercizio, parte del corpo allenata, equipaggiamento necessario per svolgere l'esercizio e il livello di complessità vengono eliminate in quanto poche e essendo informazioni importanti non posso mancare

- ## 4.3 Riformattazione colonne:

Le colonne contenenti dati categorici sono state eliminate per lasciare spazio a valori binari che indicano l'appartenza di un esercizio a una determinata caratteristica. Per poter effettuare questa operazione abbiamo utilizzato la one hot encoding che scompone le varie categorie in colonne, se l'esercizio appartiene a una determinata categoria il valore corrispondente a quella colonna sarà 1 altrimenti 0

I vantaggi dell'utilizzo di una codifica one hot :

Può migliorare le prestazioni del modello fornendo maggiori informazioni al modello sulle variabili categoriche.
Può aiutare a evitare il problema dell'ordinalità, che può verificarsi quando una variabile categorica ha un ordinamento naturale

Gli svantaggi dell'utilizzo di una codifica one hot includono:

Può portare a una maggiore dimensionalità, poiché viene creata una colonna separata per ciascuna categoria nella variabile. Ciò può rendere il modello più complesso e lento da addestrare.
Può portare a dati sparsi, poiché la maggior parte delle osservazioni avrà un valore pari a 0 nella maggior parte delle colonne.

```
from sklearn.preprocessing import OneHotEncoder

# Preparazione dell'encoder one-hot
encoder = OneHotEncoder(sparse=False)

# Selezione delle colonne da codificare
columns_to_encode = ['Type', 'BodyPart', 'Equipment', 'Level']

# Applicazione della codifica one-hot
encoded_data = encoder.fit_transform(gym_dataset[columns_to_encode])

# Creazione di un DataFrame con i dati codificati
encoded_columns = encoder.get_feature_names_out(columns_to_encode)
encoded_df = pd.DataFrame(encoded_data, columns=encoded_columns)

# Eliminazione delle colonne originali dal dataset
gym_dataset.drop(columns=columns_to_encode, inplace=True)

# Concatenazione delle colonne codificate con il dataset originale
gym_dataset = pd.concat([gym_dataset, encoded_df], axis=1)

# Visualizzazione del dataset risultante con la codifica one-hot
gym_dataset
```

# 5 Descrizione dell'Algoritmo

## 5.1 Rappresentazione degli individui

L'output che l'algoritmo deve restituire è una scheda di allenamento contenente 7 esercizi, possiamo quindi rappresentare la scheda come una array di 7 elementi contenente per ogni cella il nome dell'esercizio da svolgere.

| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| Allungamento petto | Panca Piana | Panca inclinata | Croci 30° | Pectoral Machine | Flessioni | Dietrocollo |

## 5.2 Inizializzazione della popolazione

La popolazione iniziale è generata pseudocasualmente attraverso una funzione che colloca al primo elemento della scheda un esercizio appartenente alla categoria di streatching per poi riempire casualmente le altre celle basandosi sul tipo di allenamento scelto per quel giorno

```python
#creazione della popolazione
def generate_random_workout(data, max_exercises=7, include_stretching=True):

    workout = []
    #estrazione gli esercizi di stretching dal dataset
    stretching_exercises = data[data['Type_Stretching'] == 1]

    #aggiunge un esercizio di stretching random (vincolo)
    if include_stretching and not stretching_exercises.empty:
        workout.append(stretching_exercises.sample(n=1).iloc[0])

    remaining_slots = max_exercises - len(workout)
    #estrazione degli esercizi che non sono di stretching
    other_exercises = data[data['Type_Stretching'] == 0]
    #filla tutti gli slot vuoti rimanenti (vincolo max 7 es)
    workout += other_exercises.sample(n=min(len(other_exercises), remaining_slots)).to_dict('records')

    return workout
#genera la prima generazione di schede di allenamento
def generate_initial_population(data, population_size=10, days_per_week=3):
    population = []
    #per ogni individuo screa schede di allenamento
    for _ in range(population_size):
        weekly_schedule = {}
        for day in range(days_per_week):
            daily_workout = generate_random_workout(data)
            weekly_schedule[f"Day {day + 1}"] = daily_workout
        population.append(weekly_schedule)

    return population

# Generiamo una popolazione iniziale di schede di allenamento
initial_population = generate_initial_population(gym_dataset2)
initial_population[0]  # Visualizziamo la scheda
```

## 5.3 Vincoli

I vincoli imposti dal progetto sono i seguenti:

- I giorni di allenamento sono 3
- Gli esercizi per ogni scheda sono 7

- Deve esserci un esercizio di streatching

## 5.4 Selezione degli individui

Per la selezione degli individui viene utilizzata la tecnica **K-Tournament Selection**, viene impostato il valore di k a 3 questo comporta che ad ogni selezione vengono casualmente scelti 3 individui dalla popolazione. Il miglior individuo (miglior valore di fitness) viene selezionato come genitore per la creazione della prossima generazione. E' stato scelto questo particolare algoritmo di selezione per evitare la problematica del super individuo e per la sua semplicità di implementazione e la possibilità da diminuire la diversità genetica o di diminuire la possibilità di una convergenza prematura.

```python
#selezione tramite K-Tournament selection
def selezione_tornei(popolazione, valutazioni_fitness, numero_selezionati):
    selezionati = []
    for _ in range(numero_selezionati):
        #selezioniamo K partecipanti casuali dove k= 3 insieme al loro valore di fitness
        partecipanti_torneo = random.sample(list(zip(popolazione, valutazioni_fitness)), k=3)
        #prendiamo il migliore di loro e lo selezioniamo
        migliore = max(partecipanti_torneo, key=lambda individuo: individuo[1])[0]
        selezionati.append(migliore)

    return selezionati
```

## 5.5 Elitismo

viene applicato elitismo su due membri della popolazione, i due individui vengono scelti in base al loro valore di fitness. L'elitismo è stato applicato per stabilizzare l'algoritmo evitando ampie fluttuazioni tra i valori di fitness delle varie generazioni.

## 5.6 Crossover

Viene Utilizzata la tecnica di **One-Point Crossover**, viene scelto un punto di crossover sulla lunghezza del genitore in modo randomico. Venegono combinati i due genitori in modo da creare un figlio con il corredo genetico costituito dalla zona prima del punto di crossover del genitore 1 unita alla zona dopo il punto di crossover del genitore2. La scelta è data dal fatto che il One-Point Crossover è molto efficace per problemi di ottimizzazioni semplici, inoltre ha un buon equilibrio tra esplorazione e integrità dei genitori.

```python
#Funzione per eseguire la crossover di due genitori
def crossover(genitore1, genitore2):
    figlio = {}
    punto_crossover = random.randint(1, len(genitore1))

    for i, (day, workout) in enumerate(genitore1.items()):
        if i < punto_crossover:
            #copiamo tutto quello che c'è prima del punto di crossover di entrambi i genitori
            figlio[day] = workout
        else:
            figlio[day] = genitore2[day]

    return figlio
```

## 5.7 Mutazione

Per la mutazione è stata scelta una tecnica di **Random Resetting**, la randomizzazione è data dalla scelta di un numero casuale confrontato poi con un tasso di mutazione e dallo scambio di un esercizio casuale della scheda con uno casuale del dataset. Questa strategia di mutazione è stata scelta per via della varietà casuale che evita la convergenza prematura.

```python
#Funzione per eseguire una mutazione su una scheda di allenamento
def mutazione(scheda, gym_dataset2, tasso_di_mutazione=0.1):
    #Creaiamo una copia della scheda originale
    nuova_scheda = scheda.copy()
    #iteriamo i giorni, il tasso di mutazione definisce la probabilità di mutare l'allenamento e viene scelto a caso un esercizio da sostituire
    for day, allenamento in nuova_scheda.items():
        if random.random() < tasso_di_mutazione:
            indice_esercizio = random.randint(0, len(allenamento) - 1)
            nuovo_esercizio = gym_dataset2.sample(n=1).iloc[0].to_dict()
            nuova_scheda[day][indice_esercizio] = nuovo_esercizio
    return nuova_scheda
```

## 5.8 Funzione di Fitness

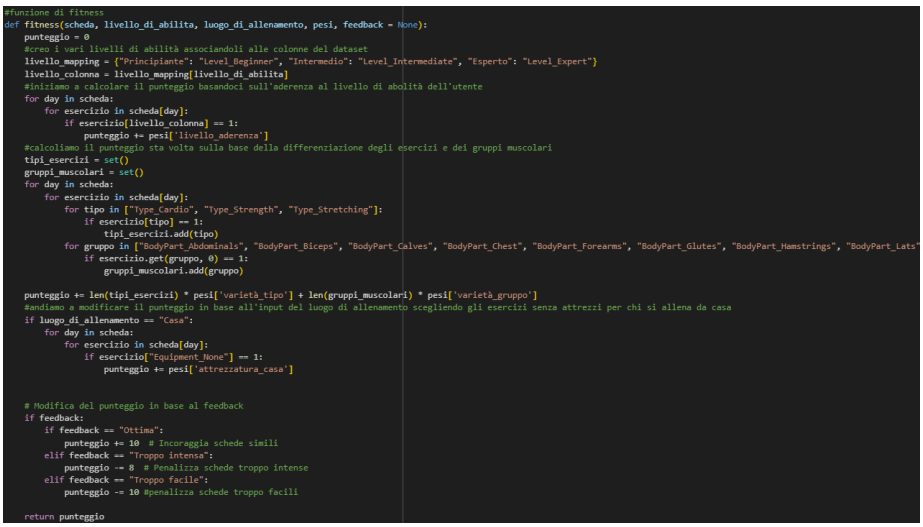
La fitness di ogni scheda di allenamento generata è valutata tramite diversi criteri:

- **livello di abilità.** indica il livello di abilità selezionato in input dall'utente che può essere Principiante Intermedio Esperto, l'algoritmo in risposta all'input digitato preferirà esercizi del livello selezionato dall'utente.
- **varietà dei tipi di esercizi** indica il grado di varietà degli esercizi per evitare di eseguire sempre gli stessi esercizi nei vari giorni di allenamento, di conseguenza l'algoritmo preferirà schede con tipi di esercizi diversi.
- **varietà dei gruppi muscolari** indica il grado di varietà dei gruppi muscolari allenati nei vari giorni della settimana per affaticare un determinato gruppo muscolare, di conseguenza l'algoritmo preferirà le schede che forniscono ampia varietà tra i gruppi muscolari.
- **luogo di allenamento** indica il luogo di allenamento selezionato in input dall'utente che può scegliere tra Casa e Palestra, per Casa l'algoritmo preferirà allenamenti senza strumenti o macchine viceversa se l'utente si

allena in palestra gli verrà fornita una scheda con esercizi che sfruttano macchine o attrezzi.

- **Feedback** indica il feedback selezionato dall'utente dopo la stampa della prima scheda, l'utente può inserire tre feedback Troppo facile, Troppo intensa, Ottima. Di conseguenza l'algoritmo attribuirà dei punti in più o in meno alla scheda stampata in base al feedback dell'utente fornendo una scheda con delle modifiche.

Ogni criterio è stato pensato per contribuire al punteggio finale della fitness di ogni scheda di allenamento per soddisfare le esegenze di un utente andando a creare una scheda personalizzata secondo le sue esigenze, proprio a questo scopo è stato pensato un algoritmo genetico multiobiettivo .



```python
#funzione di fitness
def fitness(scheda, livello_di_abilita, luogo_di_allenamento, pesi, feedback = None):
    punteggio = 0
    #creo i vari livelli di abilità associandoli alle colonne del dataset
    livello_mapping = {"Principiante": "Level_Beginner", "Intermedio": "Level_Intermediate", "Esperto": "Level_Expert"}
    livello_colonna = livello_mapping[livello_di_abilita]
    #iniziamo a calcolare il punteggio basandoci sull'aderenza al livello di abolità dell'utente
    for day in scheda:
        for esercizio in scheda[day]:
            if esercizio[livello_colonna] == 1:
                punteggio += pesi['livello_aderenza']
    #calcoliamo il punteggio sta volta sulla base della differenziazione degli esercizi e dei gruppi muscolari
    tipi_esercizi = set()
    gruppi_muscolari = set()
    for day in scheda:
        for esercizio in scheda[day]:
            for tipo in ["Type_Cardio", "Type_Strength", "Type_Stretching"]:
                if esercizio[tipo] == 1:
                    tipi_esercizi.add(tipo)
            for gruppo in ["BodyPart_Abdominals", "BodyPart_Biceps", "BodyPart_Calves", "BodyPart_Chest", "BodyPart_Forearms", "BodyPart_Glutes", "BodyPart_Hamstrings", "BodyPart_Lats"
                if esercizio.get(gruppo, 0) == 1:
                    gruppi_muscolari.add(gruppo)

    punteggio += len(tipi_esercizi) * pesi['varietà_tipo'] + len(gruppi_muscolari) * pesi['varietà_gruppo']
    #andiamo a modificare il punteggio in base all'input del luogo di allenamento scegliendo gli esercizi senza attrezzi per chi si allena da casa
    if luogo_di_allenamento == "Casa":
        for day in scheda:
            for esercizio in scheda[day]:
                if esercizio["Equipment_None"] == 1:
                    punteggio += pesi['attrezzatura_casa']


    # Modifica del punteggio in base al feedback
    if feedback:
        if feedback == "Ottima":
            punteggio += 10  # Incoraggia schede simili
        elif feedback == "Troppo intensa":
            punteggio -= 8  # Penalizza schede troppo intense
        elif feedback == "Troppo facile":
            punteggio -= 10 #penalizza schede troppo facili

    return punteggio
```

### 5.9 Stopping Condition

La stopping condition scelta è quella di rendere fisso il numero di generazioni da creare tramite il parametro **numerogenerazioni**, questa scelta è stata fatta per tenere sotto controllo il tempo di esecuzione e il consumo di risorse dell'algoritmo.

## 6 Passi dell'algoritmo

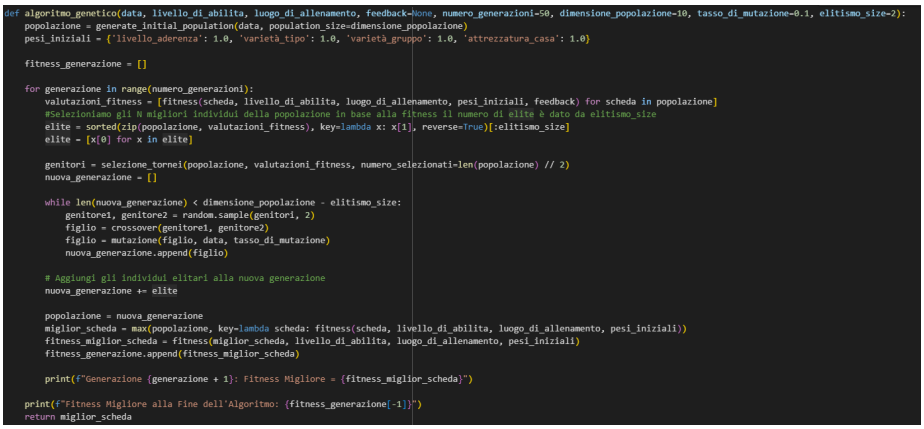
Inizialmente viene chiesto all'utente di inserire in input il livello di abilità e il luogo dell'allenamento. Successivamente viene generata una popolazione iniziale di schede di allenamento, per ogni generazione avremo:

1. Calcolo della fitnes per ogni scheda della popolazione in base ai criteri forniti

2. Selezione dei genitori per la creazione della next gen tramite K-Tournament Selection
3. Creazione della nuova generazione tramite crossover One-Point e mutazione Random Resetting
4. Selezione della miglior scheda sulla base della fitness.
5. Reiterazione su un numero fisso di generazioni e la ripetizione dei precedenti passi per ogni iterazione
6. Viene inserito un input di feedback da parte dell'utente
7. Seconda reiterazione con l'aggiunta del criterio inerente al feedback

## 6.1 Reiterazione

Dopo diverse esecuzioni abbiamo notato che l'algoritmo performi meglio se eseguito su più istanze, abbiamo notato quanto impatta il feedback sull'evoluzione della scheda. Si è notata anche una fluttuazione nei valori di fitness tra le varie generazioni per poi mostrare una convergenza verso la fine dell'esecuzione



```
def algoritmo_genetico(data, livello_di_abilita, luogo_di_allenamento, feedback=None, numero_generazioni=50, dimensione_popolazione=10, tasso_di_mutazione=0.1, elitismo_size=2):
    popolazione = generate_initial_population(data, population_size=dimensione_popolazione)
    pesi_iniziali = {'livello_aderenza': 1.0, 'varietà_tipo': 1.0, 'varietà_gruppo': 1.0, 'attrezzatura_casa': 1.0}

    fitness_generazione = []

    for generazione in range(numero_generazioni):
        valutazioni_fitness = [fitness(scheda, livello_di_abilita, luogo_di_allenamento, pesi_iniziali, feedback) for scheda in popolazione]
        #Selezioniamo gli N migliori individui della popolazione in base alla fitness il numero di elite è dato da elitismo_size
        elite = sorted(zip(popolazione, valutazioni_fitness), key=lambda x: x[1], reverse=True)[:elitismo_size]
        elite = [x[0] for x in elite]

        genitori = selezione_tornei(popolazione, valutazioni_fitness, numero_selezionati=len(popolazione) // 2)
        nuova_generazione = []

        while len(nuova_generazione) < dimensione_popolazione - elitismo_size:
            genitore1, genitore2 = random.sample(genitori, 2)
            figlio = crossover(genitore1, genitore2)
            figlio = mutazione(figlio, data, tasso_di_mutazione)
            nuova_generazione.append(figlio)

        # Aggiungi gli individui elitari alla nuova generazione
        nuova_generazione += elite

        popolazione = nuova_generazione
        miglior_scheda = max(popolazione, key=lambda scheda: fitness(scheda, livello_di_abilita, luogo_di_allenamento, pesi_iniziali))
        fitness_miglior_scheda = fitness(miglior_scheda, livello_di_abilita, luogo_di_allenamento, pesi_iniziali)
        fitness_generazione.append(fitness_miglior_scheda)

        print(f"Generazione {generazione + 1}: Fitness Migliore = {fitness_miglior_scheda}")

    print(f"Fitness Migliore alla Fine dell'Algoritmo: {fitness_generazione[-1]}")
    return miglior_scheda
```

# 7 Testing

Per la parte relativa al testing abbiamo inserito una funzione di testing che testasse l'algoritmo con tutte le combinazioni di input disponibili salvando i dati in un dataset

## 7.1 Numero di iterazioni

La fitness migliore non aumenta in modo significativo con il numero di iterazioni quindi un numero eccessivo potrebbe solo portare alla peggiorazione dell'efficienza dell'algoritmo.

### 7.2 durata delle iterazioni

La durata delle iterazioni rimane costante, da questo si evince che qualsiasi sia l'input inserito l'algoritmo è ottimo nel trovare una soluzione

### 7.3 Size della popolazione

Una popolazione troppo grande potrebbe rallentare l'intero algoritmo senza portare a miglioramenti significativi.

### 7.4 Dimensione dell'individuo

La dimensione sembra adeguata dato che l'algoritmo riesce a generare buone varietà di esercizi e si adatta bene al valore del feedback

## 8 Considerazioni finali

L'algoritmo mostra efficacia nel trovare buona soluzioni, più in generale l'algoritmo restituisce ottimi risultati per gli input di difficoltà intermedia ma ha difficoltà su l'input Esperto e su principiante e risponde bene al feedback dato dall'utente.

| | Livello | Luogo | Feedback | Fitness | Tempo |
|---|---|---|---|---|---|
| 0 | Principiante | Casa | Ottima | 25.0 | 0.348694 |
| 1 | Principiante | Casa | Troppo intensa | 5.0 | 0.310416 |
| 2 | Principiante | Casa | Troppo facile | 4.0 | 0.322271 |
| 3 | Principiante | Palestra | Ottima | 21.0 | 0.328057 |
| 4 | Principiante | Palestra | Troppo intensa | 7.0 | 0.334636 |
| 5 | Principiante | Palestra | Troppo facile | 3.0 | 0.322433 |
| 6 | Intermedio | Casa | Ottima | 40.0 | 0.326314 |
| 7 | Intermedio | Casa | Troppo intensa | 20.0 | 0.340727 |
| 8 | Intermedio | Casa | Troppo facile | 18.0 | 0.331525 |
| 9 | Intermedio | Palestra | Ottima | 38.0 | 0.348791 |
| 10 | Intermedio | Palestra | Troppo intensa | 19.0 | 0.358228 |
| 11 | Intermedio | Palestra | Troppo facile | 20.0 | 0.335713 |
| 12 | Esperto | Casa | Ottima | 24.0 | 0.355741 |
| 13 | Esperto | Casa | Troppo intensa | 2.0 | 0.329176 |
| 14 | Esperto | Casa | Troppo facile | 2.0 | 0.359884 |
| 15 | Esperto | Palestra | Ottima | 21.0 | 0.303852 |
| 16 | Esperto | Palestra | Troppo intensa | 0.0 | 0.348467 |
| 17 | Esperto | Palestra | Troppo facile | 1.0 | 0.340614 |

# 9 Conclusioni

L'implementazione di questo algoritmo è stata molto interessante, unire due delle nostre più grandi passioni ci ha permesso di lavorare in modo leggero cercando sempre di migliorare l'esecuzione con l'obiettivo di provare nella vita reale gli output forniti. Durante la fase di esecuzione abbiamo notato che l'algoritmo potrebbe generare errori su l'input Esperto e su principiante in quanto il dataset utilizzato presentava una grande variazione di esercizi di difficoltà intermedia e un pò meno dell'altro tipo di difficoltà. Siamo soddisfatti del lavoro svolto in quanto dall'inizio del corso non vedevamo l'ora di poter implementare algoritmi di intelligenza artificiale

**Glossario**

- Dataset : Insieme di dati organizzati in forma relazionale. Ha una struttura tabellare, dove di solito ogni colonna rappresenta una variabile e ogni riga corrisponde a una osservazione.
- Algoritmo Genetico : Procedura di alto livello (meta-euristica) ispirata alla genetica per definire un algoritmo di ricerca.
- Specifica P.E.A.S. : Performance Environment Actuators Sensors, sintesi in una parola degli aspetti da prendere in considerazione nello studio dell'intelligenza Artificiale.
- One Hot Encoding: ecnica che utilizziamo per rappresentare le variabili categoriali come valori numerici in un modello di machine learning
- Data imputation: Insieme di tecniche che possono stimare il valore di dati mancanti sulla base dei dati disponibili oppure mitigare il problema dei dati mancanti
- Selezione. Una copia di alcuni individui nella successiva generazione.
- Crossover. Accoppiamento di individui (parents) per crearne di nuovi (offsprings), da aggiungere alla nuova generazione.
- Mutazione. Una modifica casuale di alcune parti del corredo genetico.
- Convergenza: Capacità di un GA di migliorare iterativamente le soluzioni candidate verso il punto di ottimo
- Funzione di fitness: funzione in grado di associare un valore a ogni soluzione.
- K-way tournament: K ($<$ n) individui sono selezionati casualmente (formando il torneo) e il migliore tra questi K passa la selezione definitivamente. Si ripete finché non si arriva ad M tornei (e quindi M vincitori). Si possono avere selezioni ripetute.
- Single Point: si seleziona un punto del patrimonio genetico dei genitori e si procede alla generazione di due figli tramite scambio di cromosomi.
- Stopping Condition: Con quale criterio decidiamo di terminare l'evoluzione oppure proseguire
- Elitismo : Operazione per la quale salviamo il migliore (o i migliori k) individuo e lo copiamo nella generazione successiva.