Documentazione ShowAdvisor

Grano Giuseppe, Tortorelli Gioacchino

UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA
CORSO DI FONDAMENTI DI INTELLIGENZA ARTIFICIALE ANNO 2022/2023
PROGETTO: SHOWADVISOR REPOSITORY GITHUB
https://github.com/SalvatoreDL01/ShowAdvisor_FIA.git

1 INTRODUZIONE

1.1 Panoramica del progetto

ShowAdvisor è un progetto realizzato nell'ambito universitario riguardante l'esame di Fondamenti di intelligenza artificiale. Infatti, ShowAdvisor implementa un agente intelligente capace di suggerire dei film e serie tv in base ai gusti dell'utente. L'idea nasce per gli appassionati di film e serie tv che si ritrovano spesso nel tempo libero a sfogliare le tante pagine dei cataloghi di piattaforme di streaming alla ricerca del giusto film da guardare nelle prossime ore. Il nome ci fa capire che è uno strumento capace di dare dei suggerimenti che in questo contesto particolare riguarda appunto degli show intesi come film e serie tv.

1.2 Obiettivi del sistema

Una situazione comune a tante persone è quella di non riuscire mai a trovare un titolo che soddisfi i loro gusti nelle prime pagine dei cataloghi di piattaforme di streaming.

Questa ricerca ci porta via molto tempo, portando la maggior parte delle persone a rinunciarci. Uno strumento come ShowAdvisor in questi contesti potrebbe tornare utile in quanto il suo obiettivo è quello di suggerire dei titoli in base ai nostri gusti.

2 AMBIENTE

2.1 Specifiche PEAS

La formulazione PEAS ci permette di definire gli aspetti fondamentali dell'agente in relazione all'ambiente dove opera. Qui di seguito troviamo la formulazione PEAS del progetto in considerazione:

- P(Performance) = indica la misura prestazionale scelta che ci permette di quantificare quanto è "desiderabile" il lavoro svolto dall'agente.
- E(Environment) = esso è la descrizione dell'ambiente che nel nostro contesto è un insieme di titoli di alcune piattaforme di streaming, e un insieme di dati forniti dall'utente.
- A(Actuators) = è l'insieme degli attuatori a disposizione dell'agente che è rappresentato da una interfaccia disponibile alla visione dell'utente dove l'agente stampa i risultati della sua elaborazione.
- S(Sensors) = è l'insieme dei sensori a disposizione dell'agente per ricevere gli input e questi sono dei campi dove l'utente può inserire dei parametri su cui si baserà il lavoro dell'agente.

2.2 Caratteristiche dell'ambiente

L'ambiente dove lavorerà ShowAdvisor è:

- Completamente osservabile: i sensori danno accesso completo all'ambiente rappresentato dall'insieme dei titoli e le loro caratteristiche;
- Statico: l'ambiente rimane invariato a seguito delle azione dell'agente;
- Deterministico: in quanto lo stato successivo dell'ambiente è determinato dall'azione dell'agente e dallo stato corrente;
- Singolo utente: siccome l'ambiente ammette la presenza di un solo agente che agisce su di esso;
- Discreto: all'agente vengono rese disponibili un numero limitato di percezioni rappresentate dai gusti dell'utente;
- Episodico: l'esperienza dell'utente è divisa in episodi atomici non influenzati dalla storia percettiva;

3 FORMULAZIONE DEL PROBLEMA

3.1 Scelta dell'agente

L'agente intelligente scelto per la realizzazione del progetto è un agente basato su obiettivi: questi agenti hanno bisogno di informazioni fornite dall'utente utili ad arrivare all'obiettivo che in questo caso è l'ottimizzazione di una soluzione che soddisfi il più possibile i gusti dell'utente.

3.2 Descrizione del problema

Una volta definito l'ambiente e scelto l'agente che agirà su di esso possiamo passare alla formulazione del problema che consiste nello andare a definire il problema e definire il modello e l'algoritmo che ci porterà alla soluzione. Il problema da risolvere è quello di trovare una combinazione di film e serie tv che massimizzino un certo valore restituito da una funzione che rappresenta quando la soluzione si avvicina ai

gusti dell'utente. La soluzione al problema è realizzata mediante un modello che implementa un algoritmo di ricerca locale, in particolare un algoritmo genetico, che mira a ottimizzare una popolazione di possibili soluzioni restituendo le migliori.

3.3 Analisi del dataset

Il dataset che è stato utilizzato è stato reperito su una piattaforma Kaggle che offre tra i vari servizi quello di fornire dei dataset in formato .csv con i dati organizzati in base ad alcune categorie. In particolare il dataset scelto per la realizzazione di ShowAdvisor è una raccolta di titoli della piattaforma HBO che contiene 3166 titoli. Ogni record è composto dai seguenti attributi:

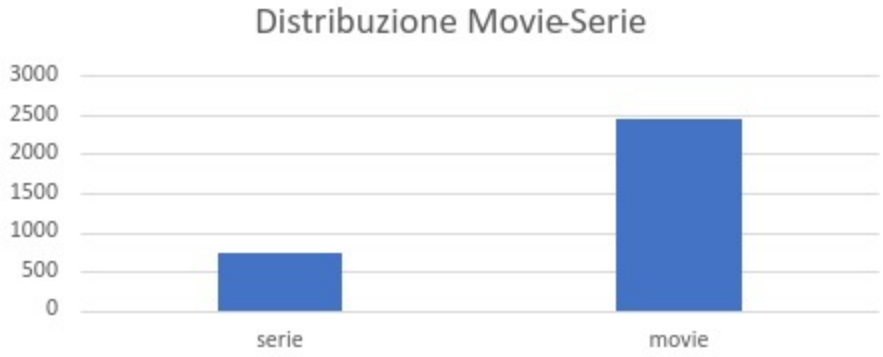
- id
- title
- type
- description
- release year
- runtime
- genres
- seasons
- score

Il dataset originale conteneva 3256 elementi. I 90 elementi eliminati dal dataset contenevano valori nulli relativi a score e/o generi, i quali non erano stimabili. È stato pertanto necessario rimuovere tutte le righe che non erano utilizzabili. Inoltre vi erano presenti due tipi di score relativi al due siti di recensione diversi. Siccome alcuni titoli non avevano uno dei due score, abbiamo deciso di accomunarli entrambi sotto l'unica denominazione di score.

3.3.1 Dimensioni del dataset

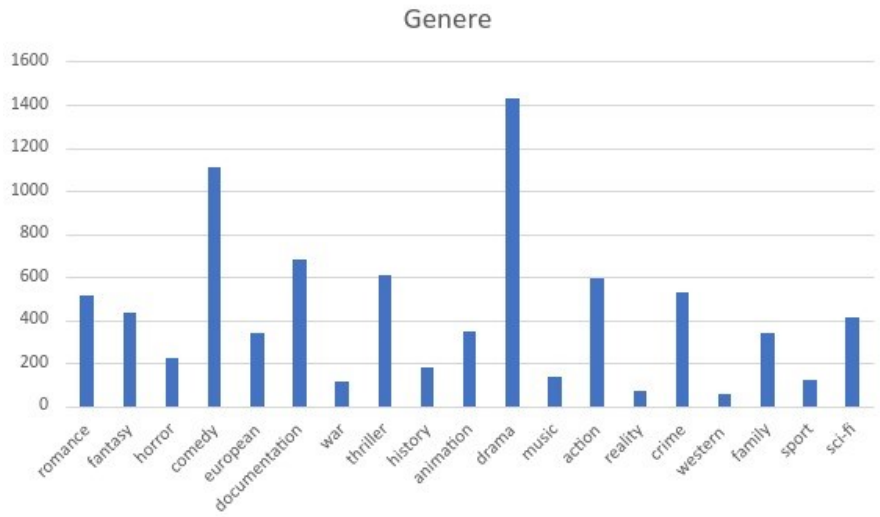
Abbiamo eseguito inoltre delle analisi sulla distribuzione dei dati all'interno del dataset : quest'ultimo risulta abbastanza limitato in relazione alla varietà dei dati che contiene. I grafici di seguito aiutano a comprendere al meglio la situazione:

- Distribuzione SerieTV-Film



  Inizialmente abbiamo notato che all'interno del dataset, che presenta 3166 entry, ci sono prevalentemente film (2444) rispetto alle serieTv (722). Questo porta ovviamente ad un'accuratezza dei risultati inferiore rispetto a quella dei film.

- Genere

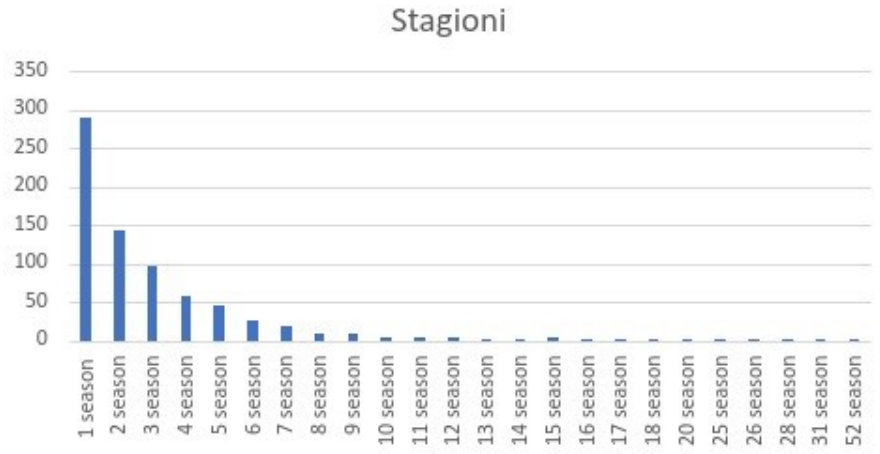


Il primo grafico che abbiamo voluto ispezionare è quello relativo al genere degli show: questa infatti è la caratteristica che ha più peso per quanto riguarda il calcolo del valore di fitness di ogni show. A questo proposito abbiamo notato che il numero di occorrenze dei vari generi è abbastanza simile (350-600 occorrenze) per la gran parte dei generi, con qualche eccezione in cui il numero di occorrenze eccede (fino ad arrivare a 1400) oppure è addirittura al di sotto di 100 (ad esempio western, con 55 occorrenze). Per bilanciare questo gap presente all'interno del dataset abbiamo voluto premiare in uno show la presenza di un genere con poche occorrenze rispetto ad uno con un numero più alto. Per fare questo abbiamo assegnato il valore di fitness relativo ai generi tramite la formula:

$$\text{valoreFitness} = \text{mediaGeneri} : \text{occorrenzeGenere}$$

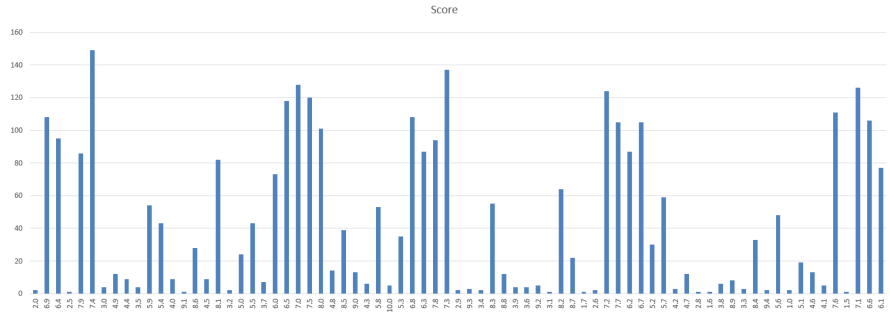
dove mediaGeneri è la media delle occorrenze di tutti i generi, mentre occorrenzeGenere è il numero di occorrenze di quel genere all'interno del dataset.

- Stagioni



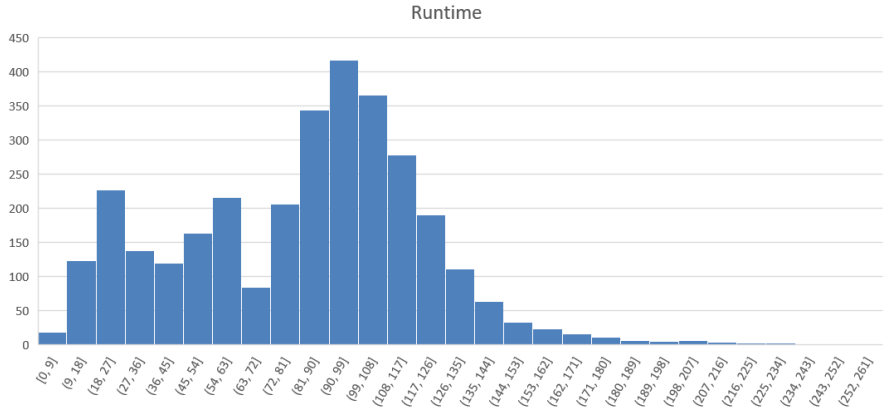
Questo dato influenza solo gli show che sono delle Serie Tv. Come si evince dal grafico la quasi totalità delle Serie Tv presenti hanno un numero di stagioni compreso tra 1 e 4, con un forte sbilanciamento verso 1 stagione. L'utente può scegliere il numero di stagioni che deve avere la serie desiderata, quindi immetterà un numero minimo e un numero massimo. Nel caso in cui il numero di stagioni di una serie Tv non combaci con l'intervallo scelto dall'utente l'algoritmo effettua la differenza tra il numero di stagioni desiderate e il numero effettivo: questo sarà sottratto dal valore di fitness dello show. Analoga è la situazione nel caso in cui il numero di stagioni ecceda rispetto a quello desiderato. Nel caso in cui il numero di stagioni effettive combaci con la scelta dell'utente verrà aggiunto un valore positivo alla fitness dello show.

- Score



Dal grafico precedentemente riportato si può notare come i valori relativi ai voti degli show presenti nel dataset siano abbastanza sbilanciati: avremo quindi delle occorrenze molto differenti tra loro. Questo influisce però in maniera relativa in quanto, nonostante l'algoritmo seleziona degli individui con la fitness più alta, l'aggiunta di punti per il valore di fitness in base allo score non pesa molto nel calcolo della fitness totale. Questo permette all'algoritmo di andare a preferire comunque show con score più alto, senza però escludere show che hanno uno score relativamente basso ma con altre caratteristiche combacianti.

- Runtime



In questo caso per quanto riguarda il Runtime degli show abbiamo uno sbilanciamento in eccesso del numero di occorrenze per la fascia di minutaggio compresa tra 72'-126'. Questo dipende dal fatto che sono presenti molti più film che serie Tv (come si nota dal grafico successivo). Per colmare questo sbilanciamento abbiamo deciso di togliere al valore di fitness la differenza tra minutaggio desiderato e minutaggio effettivo nel caso in cui il risultato di questa operazione sia al massimo 15, mentre di togliere un valore arbitrario in caso contrario. Nel caso in cui il minutaggio effettivo si trovi nel range specificato dall'utente, aggiungiamo un valore arbitrario alla fitness dello show. I valori scelti hanno un peso non molto incisivo per il valore totale della fitness, in modo da cercare di equilibrare e considerare show che hanno altre caratteristiche combacianti con quelle desiderate, tranne che il runtime.

4 RISOLUZIONE DEL PROBLEMA

4.1 Algoritmi Genetici

La risoluzione del problema come stabilito nella fase di definizione del progetto avverrà mediante l'uso di un algoritmo genetico. Il funzionamento di questo tipo di algoritmo è come quello di un algoritmo locale ovvero si parte da una possibile soluzione e si cerca di ottimizzarla con la differenza che la fase di ottimizzazione si basa sulle teorie dell'evoluzione di Darwin. Esso parte generando una popolazione iniziale e poi grazie a tre fasi e ad una funzione di fitness si tenta di far evolvere la popolazione, il meccanismo di creazione di nuove generazioni continua fino al verificarsi di una determinata condizione. Le tre fasi dell'evoluzione a cui facciamo riferimento nelle

precedenti righe sono la fase di selezione, crossover e mutazione. L'algoritmo genetico è stata la nostra scelta per il semplice motivo che esso riesce a trovare una soluzione sub ottimale in tempi ragionevoli. In questo contesto trovare una soluzione sub ottimale è il miglior compromesso in quanto anche se esiste una soluzione ottima rappresentata da un ottimo globale non è certo che questo soddisfi totalmente l'utente la cui scelta può anche essere influenzata da fattori non quantificabili da un AI.

4.2 Definizione delle specifiche dell'algoritmo

La progettazione di un algoritmo genetico comprende la decisione della configurazione di alcuni parametri dell'algoritmo, questa fase è molto importante in quanto una buona configurazione può migliorare o peggiorare la ricerca. I parametri migliori della configurazione possono essere stabiliti tramite una sperimentazione empirica pertanto attraverso l'osservazione dei risultati ottenuti con diverse configurazioni e dall'esperienza.

4.2.1 Population size

La population size indica il numero di individui che compongono la popolazione nel nostro caso una popolazione è costituita da 500 individui differenti con preferenza di trovare o tutti show di tipo film o type serie tv.

4.2.2 Rappresentazione degli individui

Ogni individui è codificato come un array di 5 elementi, gli elementi presenti nell'array (individuo) sono delle istanze della classe Show. La classe Show è una rappresentazione degli elementi nel dataset quindi presenta gli stessi attributi, in parole semplici rappresenta i film e le serie tv.

4.2.3 Criterio di inizializzazione della popolazione

L'inizializzazione della popolazione viene fatta in maniera casuale, ogni individuo viene creato scegliendo 5 titoli dal dataset in maniera casuale. Qui di seguito è inserito il codice relativo alla creazione della popolazione iniziale:

```
public void inizializza(String tipo) throws IOException {
    for(int i=0; i<numIndividui;){
        Individuo individuo = new Individuo(numShow, fitnessTotale: 0);
        individuo.crea(tipo);
        if(!lista.contains(individuo)){
            lista.add(individuo);
            i++;
        }
    }
}
```

Il metodo inizializza() permette di iterare gli individui all'interno della popolazione su cui è chiamato e per ogni individuo invoca il metodo crea() che permette di scegliere in maniera casuale degli Show da inserire nell'individuo.

4.2.4 Funzione di fitness

La funzione di fitness analizza i singoli show all'interno dell'individuo per ogni individuo va a valutare il suo score, in base al suo valore l'algoritmo assegna un valore positivo, negativo o nullo alla fitness. Il secondo controllo che fa la funzione di fitness è quello di controllare il numero di generi scelti dall'utente presenti nello show, per genere presente nello show viene aggiunto un valore di +25 alla fitness più un valore di peso in base alla distribuzione dei generi nel dataset (meno generi di un tipo sono presenti nel dataset più alto sarà il loro valore di peso). Il valore di peso è calcolato per singolo genere ed è uguale al rapporto tra una media dei valori dei generi nel dataset e il numero di occorrenze di quel genere. L'altro controllo della funzione di fitness riguarda la durata degli show se questo rientra nell'intervallo specificato dall'utente viene assegnato un punteggio di +10 alla funzione di fitness altrimenti se la durata va

oltre il limite inferiore o superiore scelto dall'utente viene tolto alla fitness un valore pari a quanti minuti eccede la durata rispetto il limite (inferiore o minore) fino ad un massimo di -15 infatti oltre i 15 viene tolto un valore fisso di -20. Per quanto riguarda le serie tv viene fatto un controllo in più sul numero di stagioni simile a quello della durata, se il numero di generi rientra nell'intervallo specificato dall'utente allora viene aggiunto un valore di +10 altrimenti la fitness è decrementata di tanti punti quante sono le stagioni che eccedono dal valore massimo o minimo specificato.

4.2.5 Algoritmo di selezione

L'algoritmo usato nella fase di selezione è il Truncation: esso va ad ordinare gli individui in base alla fitness calcolata; una volta ordinata la popolazione si procede con il selezionare i k migliori individui dove k è la taglia della size mating pool e solo i k elementi scelti parteciperanno alle fasi successive.

```
public void selezione(Popolazione popolazione){
    fitness.calcolaFitnessPopolazione(popolazione);
    Collections.sort(popolazione.getLista(), (i1, i2) -> {
        return Double.compare(i1.getFitnessTotale(), i2.getFitnessTotale());
    });
    int i;
    for(i=sizeMatingPool; i<popolazione.getLista().size();){ //prendiamo i migliori mille
        popolazione.getLista().remove(i);
    }
}
```

4.2.6 Algoritmo di crossover

L'algoritmo utilizzato per la seconda fase, ovvero quella di crossover, è un single-point crossover. La scelta è ricaduta sul single-point crossover in quanto ci permette di non andare a creare troppa diversità, situazione che si sarebbe verificata in caso di utilizzo di un crossover a più punti. L'algoritmo è stato realizzato in maniera tale da andare a selezionare due individui in modo casuale tra quelli rimasti nella popolazione a seguito della fase di selezione, ed andarli ad incrociare tra di loro a seguito della scelta di un punto singolo di crossover in modo casuale. In particolare il crossover avverrà tra due individui generando un sono figlio per via del fatto che creare due individui con gli stessi elementi ma con show in posizioni differenti non cambia la fitness dell'individuo.

```
public void crossOver(Popolazione popolazione){
    Random random = new Random();
    int i;
    Individuo iCross1 = null;
    Individuo iCross2 = null;
    while(popolazione.getLista().size() < popolazione.getNumIndividui()){
        Individuo i1 = new Individuo(popolazione.getNumShow(), fitnessTotale: 0);
        iCross1 = popolazione.getLista().get(random.nextInt( bound: sizeMatingPool - 1));
        iCross2 = popolazione.getLista().get(random.nextInt( bound: sizeMatingPool - 1));
        int singlePoint = random.nextInt(popolazione.getLista().get(0).size());
        int j;
        for(j=0; j<popolazione.getLista().get(0).size(); j++){
            if(j < singlePoint)
                i1.add(iCross1.get(j));
            else
                i1.add(iCross2.get(j));
        }
        popolazione.getLista().add(i1);
    }
    fitness.calcolaFitnessPopolazione(popolazione);
}
```

4.2.7 Algoritmo di mutazione

L'algoritmo scelto per la fase di mutazione è il random resetting. Il random resetting consiste nell'andare a selezionare un gene dell'individuo, nel nostro caso è una delle istanze Show nell'array, e cambiarlo con un'altra istanza casuale che non sia già presente nell'individuo. Questo algoritmo è stato scelto per evitare di modificare troppo gli individui che è una possibile conseguenza dell'uso di algoritmi che potrebbero cambiare più di un gene, mentre l'altro motivo che ci porta all'uso del random resetting

è che algoritmi come swap, scramble o inversion lavorano sulle posizione dei geni nell'individuo ma nel nostro caso la posizione degli show nell'array sono irrilevanti al calcolo di una euristica ispirata alla funzione di fitness. L'algoritmo è stato reso meno casuale grazie al calcolo della fitness degli elementi che lo compongono trovando così il peggiore che sarà proprio il gene che verrà sostituito. Qui di seguito è mostrato il codice per l'implementazione del random resetting:

```
public void mutazione(Individuo i) throws IOException {
    int n, posizionePeggiore = 0;
    double valorePeggiore = fitness.fitnessShow(i.get(0));
    //ricerca dell'elemmento peggiore e salvataggio della sua posizione nell'individuo
    for(n=0; n<i.size(); n++){
        //System.out.println(fitness.fitnessShow(i.get(n)));
        if(fitness.fitnessShow(i.get(n)) < valorePeggiore){
            valorePeggiore = fitness.fitnessShow(i.get(n));
            posizionePeggiore = n;
        }
    }
    Random random = new Random();
    List<Show> showList = Parser.getInstance(tipo);
    i.set(posizionePeggiore, showList.get(random.nextInt(elementi)));
    fitness.calcolaFitness(i);
}
```

5 TESTING

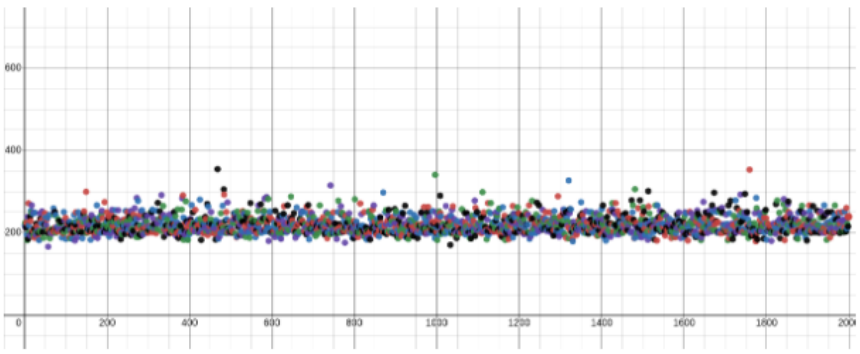
La fase di testing del progetto riguarda la scelta dei valori per la configurazione dell'algoritmo. In particolare i test riguardano la scelta della size mating pool, il numero di iterazioni per la stopping condition ed il numero di individui che appartengono ad ogni popolazione. I test sono stati effettuati tramite la classe di test TestClass presente all'interno del progetto. In seguito vengono mostrati solo alcuni dei test eseguiti considerati più rilevanti per la definizione dei parametri di configurazione. I test sono stati eseguiti sulla ricerca di un film in quanto rappresenta la maggior parte dei dati contenuti nel dataset con le seguenti preferenze:

```
listaGeneri.add("reality");

listaGeneri.add("music");

listaGeneri.add("sport");

Fitness fitness = new Fitness("MOVIE", 0, 0, listaGeneri, 80,
100);
```

Ovvero la ricerca è stata fatta su dei film di genere reality, music e sport, mentre la durata minima scelta è di 80 min e quella massima è di 100.

5.1 1° test

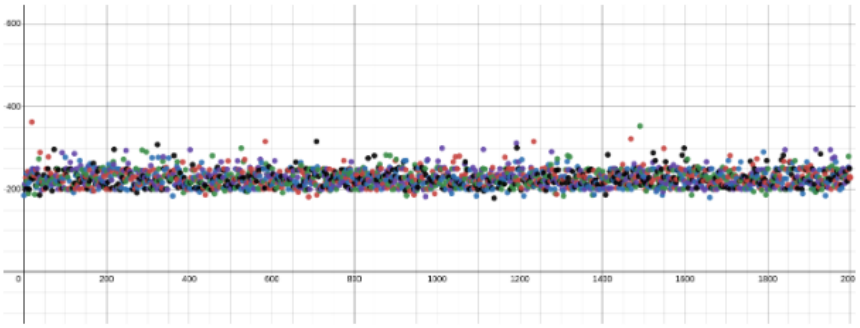
Configurazione:
Size mating pool: 500
Numero di individui per popolazione: 1000
Numero iterazioni: 2000



Da questo primo test si può notare come l'algoritmo non produce nessun miglioramento rispetto la popolazione iniziale creata casualmente all'inizio, pertanto si procede aumentando il numero di individui.

5.2 2° test

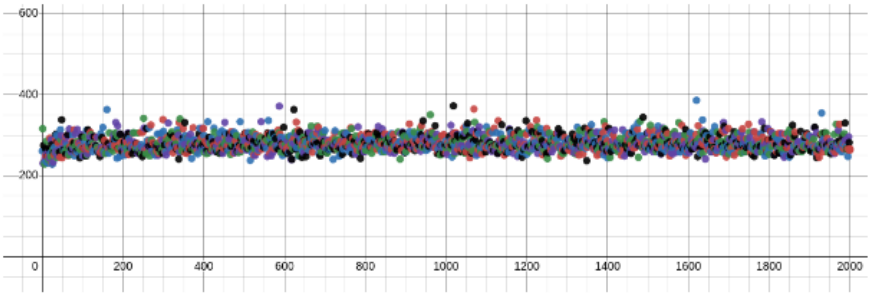
Configurazione:
Size mating pool: 500
Numero di individui per popolazione: 2000
Numero iterazioni: 2000



Dal secondo test dopo aver raddoppiato il numero di individui non ci sono stati miglioramenti rispetto al primo test pertanto si procede aumentando il valore della size mating pool.

5.3 3° test

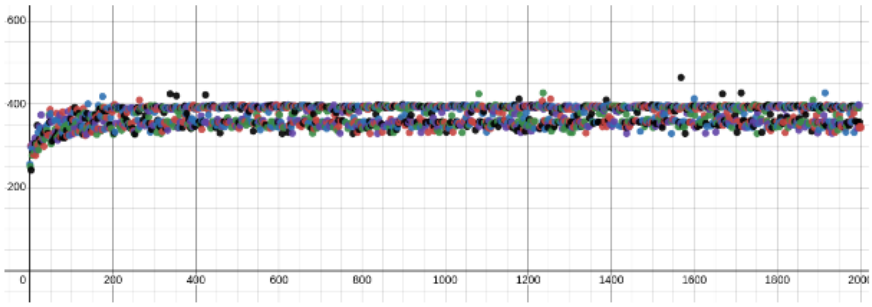
Configurazione:
Size mating pool: 1500
Numero di individui per popolazione: 2000
Numero iterazioni: 2000



Il valore della size mating pool è stato portato da 500 a 1500 notando un miglioramento della fitness degli individui soprattutto nelle prime 100 iterazioni. A seguito di questo test è stato aumentato ancora il valore della size mating pool.

5.4 4° test

Configurazione:
Size mating pool: 1800
Numero di individui per popolazione: 2000
Numero iterazioni: 2000

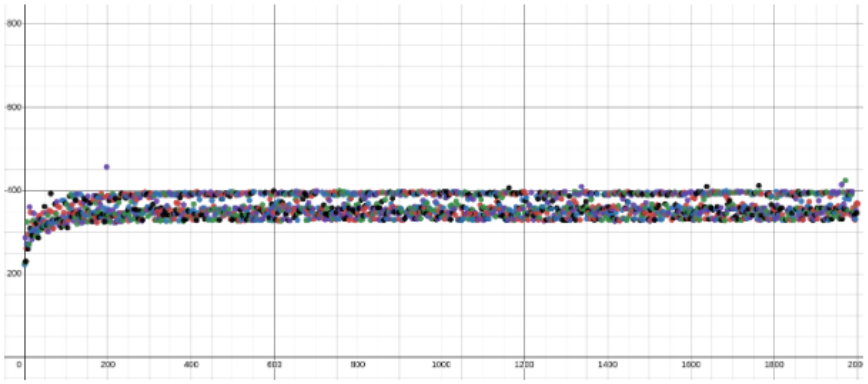


Il valore della size mating pool è aumentato a 1800 e si è notato ancora un ulteriore aumento ottenendo degli individui con fitness pari a circa 400.

5.5 5° test

Configurazione:

Size mating pool: 900
Numero di individui per popolazione: 1000
Numero iterazioni: 2000

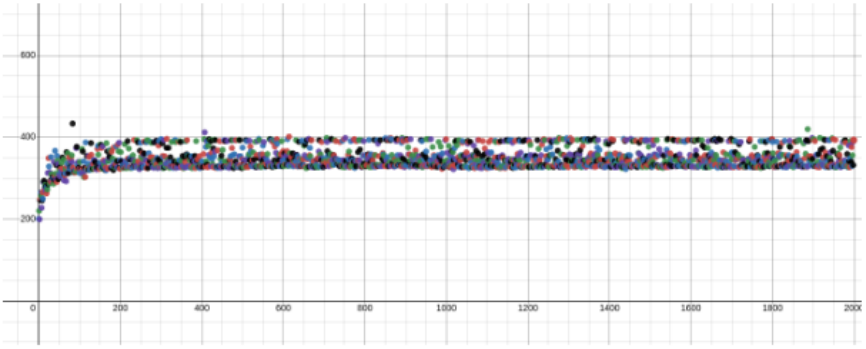


In questo test è stato dimezzato il valore della size mating pool e degli individui della popolazione, a seguito di cioè si è notato che il valore della fitness che si riesce a trovare si aggira sempre intorno ai 400 nonostante la diminuzione degli individui. questo test è stato utile a capire la relazione tra size mating pool e numero di individui, infatti la fitness aumenta quando la size mating pool si avvicina al numero di individui piuttosto che non l'aumentare di quest'ultimi, i successivi test pertanto manterranno un numero di individui basso andando a migliorare anche il tempo per trovare la soluzione.

5.6 6° test

Configurazione:

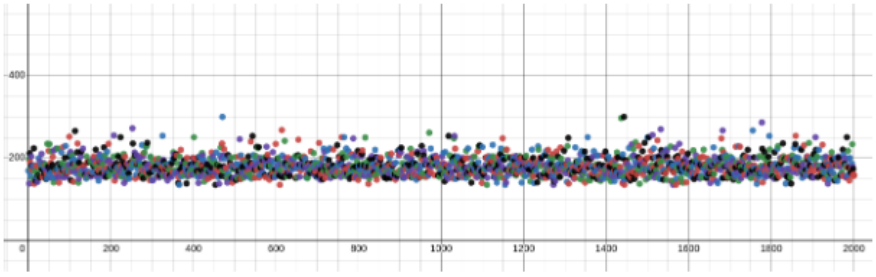
Size mating pool: 450
Numero di individui per popolazione: 500
Numero iterazioni: 2000



In questo test è stato diminuito ulteriormente il valore della size mating pool e quello del numero di individui lasciando il rapporto tra di loro uguale rispetto il test precedente.

5.7 7° test

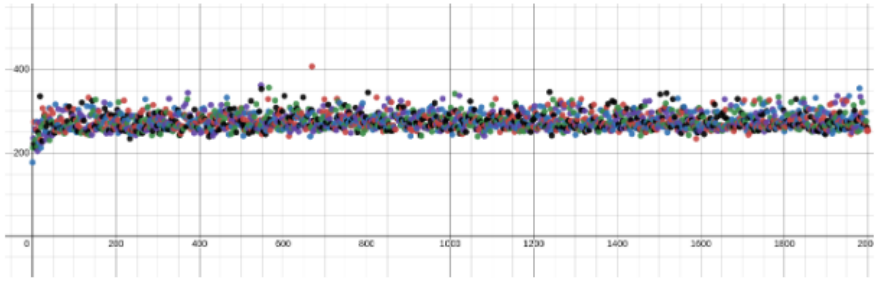
Configurazione:
Size mating pool: 85
Numero di individui per popolazione: 100
Numero iterazioni: 2000



In questo test si è diminuito ancora ulteriormente il numero di individui fino a 100 ottenendo una diminuzione della fitness, pertanto 100 individui sono pochi e creano poca diversità.

5.8 8° test

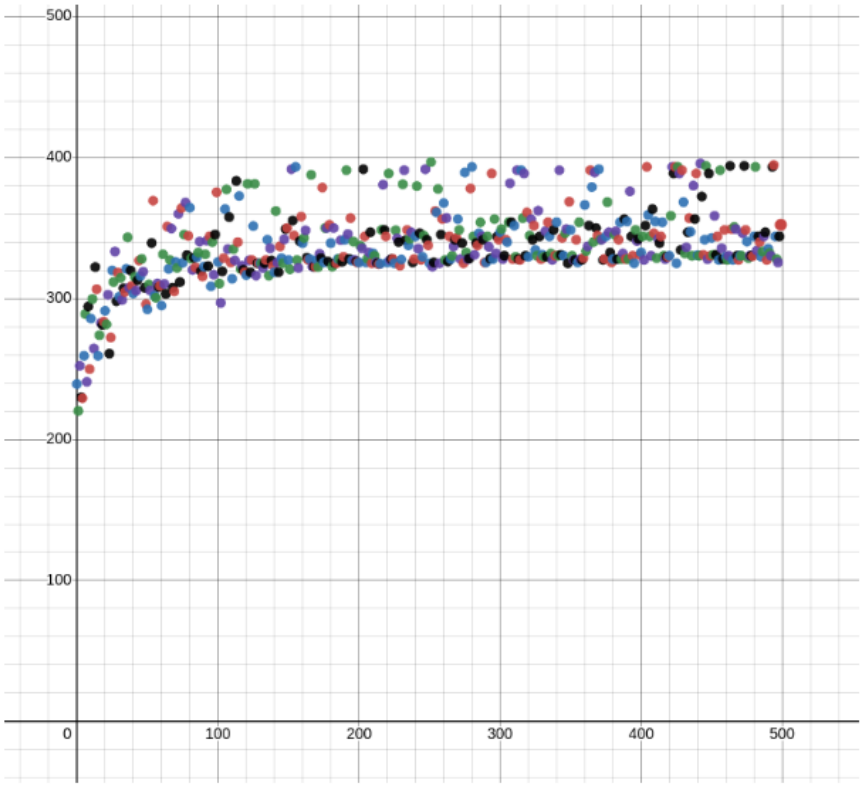
Configurazione:
Size mating pool: 400
Numero di individui per popolazione: 500
Numero iterazioni: 2000



In questo test dopo aver individuato un buon numero di individui per la popolazione ovvero 500 si è proceduti provando a far diminuire la size mating pool ottenendo però un peggioramento.

5.9 9° test

Configurazione:
Size mating pool: 450
Numero di individui per popolazione: 500
Numero iterazioni: 500

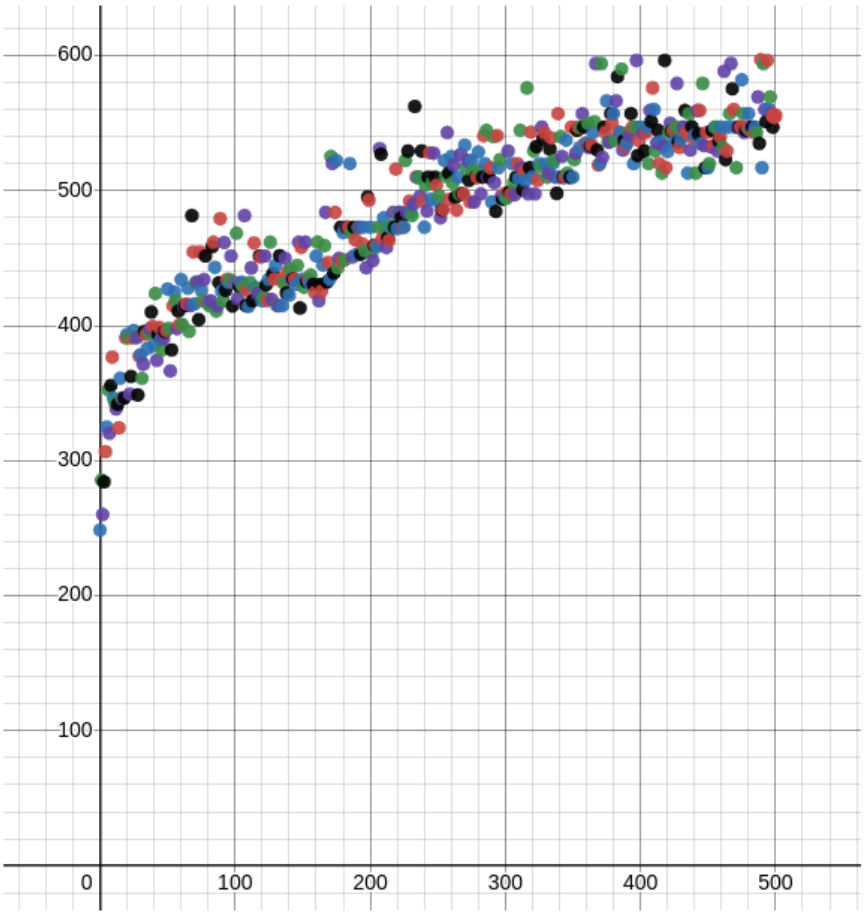


In questo test è stato diminuito il valore delle iterazioni dell'evoluzione fino a 500 in quanto l'evoluzione arriva a convergenza dopo le prime 200 iterazioni come osservato dai test precedenti. La diminuzione di questo valore ha portato ad un'ulteriore diminuzione del tempo di esecuzione dell'algoritmo.

5.10 10° test

Configurazione:

Size mating pool: 490
Numero di individui per popolazione: 500
Numero iterazioni: 500



Questo test svolge l'operazione inversa del test 8 dove andiamo a diminuire la size mating pool, in questo caso andiamo ad aumentarla a seguito della proprietà notata: avvicinando la size mating pool alla size della popolazione aumenta la fitness. Questa proprietà è verificata anche in questo caso ma comporta l'ottenimento di individui con show (con fitness alte) ripetuti come mostrato nel seguente risultato. che presenta tre show con lo stesso id. Risultati:

id=tm68758, id=tm68758, id=tm127046, id=tm68758, id=tm117765

Dai test eseguiti si è arrivati alla definizione dei parametri di configurazione usati per l'algoritmo ovvero:
Size mating pool: 450
Numero di individui per popolazione: 500
Numero iterazioni: 500