UNIVERSITÀ DEGLI STUDI DI CAGLIARI

FACOLTÀ DI INGEGNERIA ED ARCHITETTURA

CORSO DI LAUREA IN INGEGNERIA ELETTRICA, ELETTRONICA ED INFORMATICA

PORTAMENTO

MACHINE LEARNING PER LA CATALOGAZIONE MUSICALE,
INTERFACCIA TRIDIMENSIONALE PER EVIDENZIARNE LE INERENTI SFUMATURE

Relatore:
Giorgio Giacinto

Tesi di laurea di:
Nicolò Loddo

ANNO ACCADEMICO 2019/2020

Abstract

Portamento è un programma di catalogazione musicale che si pone come obbiettivo l'esplorazione interattiva di un database di tracce.
Il nome "Portamento" nel mondo della musica si riferisce all'esecuzione, nel passaggio da una nota all'altra, dei suoni compresi tra le due. In analogia a questo concetto, **Portamento**, vuole rispettare le sfumature implicite presenti nella metodologia classica dei criteri di catalogazione musicale.
Il software ottiene le informazioni sulle tracce da Spotify a partire da una semplice lista di playlist.
La catalogazione avviene attraverso algoritmi di Machine Learning su Python, forniti dalla libreria scikit-learn.
L'interfaccia è sviluppata in C# utilizzando il motore grafico *Unity*.

Indice

1 Introduzione **3**
1.1 Lo streaming musicale . . . 3
1.2 Motivazioni . . . 4
1.3 Progetti con presupposti simili . . . 4

2 Il problema della catalogazione musicale **5**
2.1 Sulla ambiguità e soggettività . . . 5
2.2 Approccio proposto . . . 6

3 Il Software **8**
3.1 Dati e metodologie . . . 8
3.1.1 Algoritmi . . . 8
3.1.2 Parametri di Clusterizzazione . . . 10
3.2 Implementazione . . . 14
3.2.1 Clustering . . . 14
3.2.2 Popolamento e gestione del Database . . . 15
3.2.3 Il motore grafico e gli asset utilizzati . . . 15
3.3 Struttura schematica del software . . . 17

4 Utilizzo **21**
4.1 Funzionamento e risultati . . . 21
4.2 Altri possibili utilizzi . . . 23

5 Possibili miglioramenti e sviluppi futuri **24**

Capitolo 1

Introduzione

1.1 Lo streaming musicale

L'avvento di piattaforme di streaming musicale come Spotify o Apple Music hanno rivoluzionato interamente il modo di ascoltare la musica.
Sono innumerevoli i benefici che questi hanno portato ai fan e all'industria musicale, che sta vedendo una nuova crescita dopo anni di recessione dovuti alla pirateria.

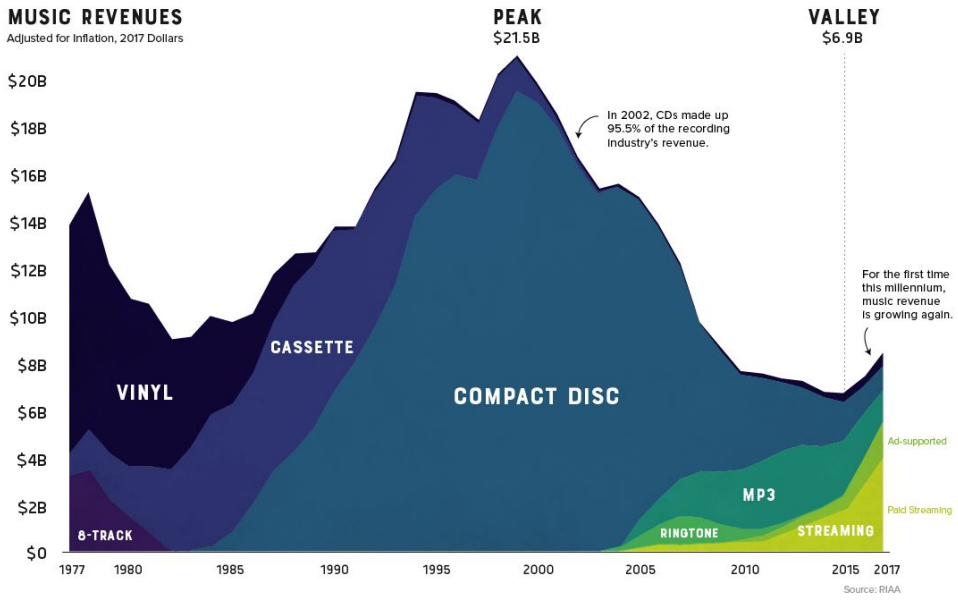


Figura 1.1: Ricavi dell'industria musicale in America. Fonte: RIAA.

Questo tipo di piattaforme, permettono di avere un'immensa selezione di tracce a pochi click di distanza, comodamente divisibile in playlist adatte ad ogni momento o umore.
Al giorno d'oggi hanno sotto il proprio controllo non solo un database da milioni di tracce musicali, ma un altrettanto vasto database riguardante le tendenze di ascolto di milioni di fan distribuiti in tutto il mondo; infinite dunque le possibilità di utilizzo. Tuttavia da grandi potenzialità, come noto, derivano grandi responsabilità.

Tra i più famosi servizi di streaming Spotify è probabilmente quello che sfrutta al meglio il proprio tesoro di informazioni e anche per questo gode del bacino d'utenza più importante. Il sistema di raccomandazione su cui principalmente si basa il colosso Svedese è denominato *Collaborative Filtering* e ruota attorno ai pattern comportamentali collezionati implicitamente dai

propri utenti: se due persone hanno vari interessi in comune è infatti probabile che ne abbiano anche altri [8].
Spotify, oltre a consultare i pattern comportamentali del proprio bacino d'utenza, esegue anche analisi tecniche delle tracce, estrapolando varie caratteristiche per ogni canzone. Può dunque calcolare che tipo di musica sarebbe più appropriato consigliarci basandosi semplicemente sui nostri soli ascolti passati.
Un altro modo intelligente con cui ha saputo sfruttare il proprio Database è stato costruirci attorno un'interessante e ricca API messa a disposizione degli interessati. Trovo questa decisione positiva in quanto promuove l'avanzamento scientifico e l'istruzione, incitando e aiutando la ricerca indipendente di qualunque sviluppatore di terze parti.
È proprio grazie alla API di Spotify che questo progetto è potuto nascere.

1.2 Motivazioni

Il sistema di raccomandazioni di Spotify è un sistema estremamente sofisticato, frutto di anni di lavoro e investimenti nel campo della scienza dei dati. Evidentemente, il software da me sviluppato non è comparabile a tale sistema. Esso non vuole porre rimedio ad una ipotetica mancanza di accuratezza nelle raccomandazioni, ma, invece, soddisfare un bisogno di **ricerca più consapevole e interattiva**.
La maggior parte dei servizi che utilizziamo raccolgono costantemente dati di utilizzo per fornirci un'esperienza estremamente personalizzata rispetto alle nostre esigenze. I suggerimenti forniti da Spotify testimoniano quanto la piattaforma conosca bene i nostri gusti, tanto da essere addirittura distopico per alcuni.
Lungi da me analizzare l'etica della raccolta dati: ciò di cui mi preoccupa parlare è invece il fatto che, un tale metodo, **si sostituisce a noi stessi** nella ricerca, quando invece spesso il tragitto è importante tanto quanto la meta anche in questo tipo di "viaggi".
Il software Portamento cerca dunque di soddisfare in qualche misura questo bisogno di esplorazione e ricerca individuali nell'ambito dell'infinito universo musicale e costituisce uno dei possibili approcci che si possono utilizzare.

1.3 Progetti con presupposti simili

Esistono vari altri **metodi** e **servizi** con cui si può effettuare una ricerca più consapevole.
Tra i primi, i numerevoli forum in cui si parla attivamente di musica o la navigazione tra gli artisti con cui collaborano quelli da noi preferiti.
Tra i secondi, ritengo valido ad esempio il servizio *Discover Quickly* [6], basato anch'esso proprio sull'API di Spotify.
Il sito *Every Noise at Once* [7] merita invece una citazione particolare in quanto è stato di ispirazione per questo progetto. Esso fornisce una immensa mappa con i più svariati generi musicali organizzati posizionalmente.

Capitolo 2

Il problema della catalogazione musicale

2.1 Sulla ambiguità e soggettività

Per navigare efficientemente attraverso grossi database di musica è indispensabile la preventiva individuazione di un criterio di categorizzazione.
La nozione di "genere musicale" è spesso usata a questo scopo, ma presenta grossi problemi di soggettività e ambiguità.
In un testo di ricerca riguardante l'estrazione di caratteristiche di timbro dalle tracce, lo studioso F. Morchen sostiene che: "*Le esistenti tassonomie dei generi musicali risultano in qualche modo arbitrarie e difficili da comparare. Spesso i generi non corrispondono nemmeno a come suona la musica, ma al tempo e luogo di provenienza o al bagaglio culturale degli autori*".
"*Alcuni ricercatori spiegano le basse prestazioni di algoritmi di catalogazione musicale basati sull'estrazione di caratteristiche acustiche, attraverso appunto la natura ambigua dei generi stessi*" [12].

Glenn McDonald, fondatore del sito *Every Noise at Once* citato nell'introduzione, effettua la catalogazione utilizzando un'analisi di dati stratificata insieme ad opportuno lavoro editoriale (di contesto).
Lui e il suo team di ricerca non si interrogano riguardo la coerenza nella tassonomia dei generi: a titolo d'esempio, il "Rock Sloveno" è definito da caratteristiche geografiche e culturali, mentre il "Christian Hip-Hop" si distingue per orientamento filosofico. L'obbiettivo ultimo è la ricerca, e i generi da loro definiti hanno come unica funzione l'orientamento: "*il punto non è risolvere dispute, ma invitarti ad esplorare la musica.*" [10].

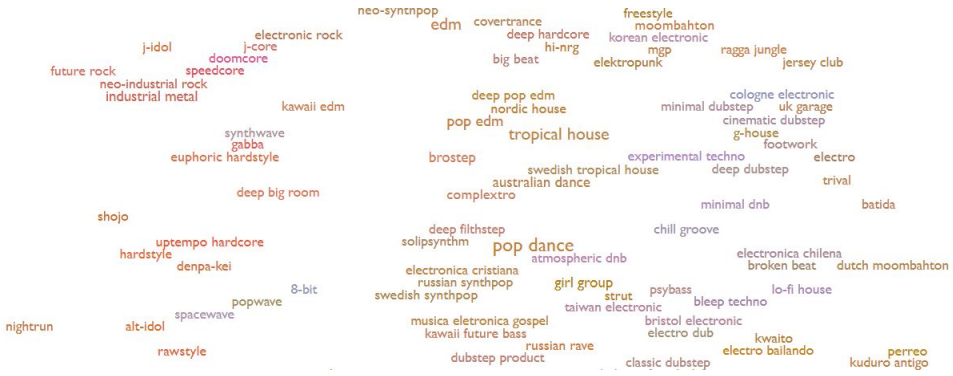


Figura 2.1: Piccolo estratto della mappa di Every Noise at Once.

2.2 Approccio proposto

Il progetto da me realizzato vuole avere lo stesso identico obbiettivo, ma utilizza un approccio diametralmente opposto.

Per catturare al meglio le nuance della catalogazione è stato scelto di progettare e creare per il software un'interfaccia tridimensionale dentro la quale muoversi comandando un personaggio fittizio.
L'orientamento non avviene utilizzando una preventiva denominazione dei generi, ma si basa sui quattro punti seguenti:

- la definizione da parte dell'utente del significato dei tre assi cartesiani tra le varie caratteristiche dei pezzi;
- il raggruppamento dei pezzi in "**cluster**" di canzoni simili e la rappresentazione di tali cluster, e non delle canzoni, all'interno dello spazio tridimensionale;
- l'utilizzo di una mappa bidimensionale che fornisce subito informazioni sulla posizione dei cluster;
- l'inserimento di un massimo di dieci pezzi familiari all'utente, il cui cluster di appartenenza sarà segnalato sulla mappa.

Non dovendo dunque etichettare con un nome i raggruppamenti, la definizione contestuale dei cluster è data dalle stesse canzoni che lo compongono.
Il progetto si basa sulla concezione della musica come un universo raggruppabile in infiniti modi e a diversi livelli di profondità, come in una **struttura ad albero** la cui **radice** è il **database intero**, mentre le **foglie** - raggruppamenti all'ultimo livello di profondità - sono cluster definiti da una **unica canzone** del database.
Partendo da questi presupposti è stato scelto di utilizzare un algoritmo di clustering di tipo gerarchico.
Una clusterizzazione di questo genere riesce, grazie alla sua struttura capillare, a non sminuire le **inerenti sfumature** di cui si è parlato prima e, allo stesso tempo, a fornire uno dei presupposti citati per l'orientamento dell'utente nell'interfaccia.
La scelta dell'algoritmo verrà trattata più approfonditamente nel prossimo capitolo.

La clusterizzazione, prendendo in considerazione tutte le coordinate dei nostri oggetti, raggiunge anche un altro scopo: limita il problema della rappresentazione scarna, in sole tre dimensioni, di oggetti così complessi. Tre dimensioni non sono infatti sufficienti per rappresentare oggetti con più di tre caratteristiche costituenti, per cui ciò che vedremo attraverso l'interfaccia è da considerarsi la proiezione di un oggetto multidimensionale - la canzone - in un universo tridimensionale.
Il raggruppamento consente di non dover distinguere le canzoni dell'intero database basandosi su sole tre coordinate, infatti, si potrà navigare attraverso la struttura dell'albero di raggruppamento. Muoversi tra i livelli di profondità ha funzione analoga al muoversi rispetto a tutte le caratteristiche censite delle canzoni.

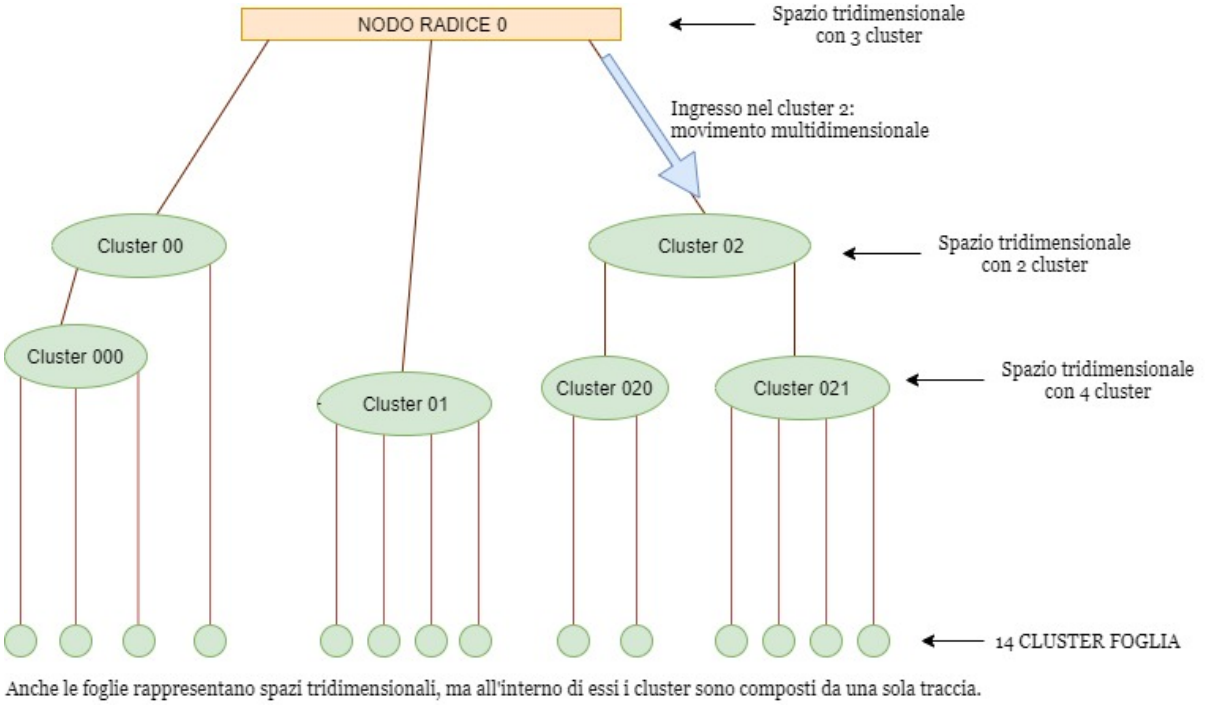


Figura 2.2: La struttura ricorda un albero rovesciato.

Il diagramma rappresenta la struttura ad albero e il modo in cui ci si sposta all'interno. L'ingresso in un cluster è considerabile un movimento multidimensionale in quanto fatto lungo tutte le caratteristiche delle canzoni, mentre negli spazi tridimensionali costituiti dai nodi ci si sposta lungo i tre assi classici.

Per assecondare infine la soggettività della catalogazione, il software consente all'utente di decidere il **peso** che ogni caratteristica deve avere nella formazione dei raggruppamenti. La definizione dei pesi è utile anche per esigenze di raggruppamento con criteri differenti da quelli generalmente utilizzati: ogni set di pesi costituirà effettivamente un **diverso significato semantico** e porterà a risultati completamente diversi, per esempio un set può conferire un significato di "generi" ai raggruppamenti, un altro, invece, il significato di "umori".
Il set da me utilizzato e scelto come predefinito per il software, cerca di ottenere una suddivisione simile al modo in cui solitamente si suddivide per generi. A questo scopo, ho ottenuto risultati a mio parere migliori attribuendo **al tempo** e **all'energia** un peso maggiore rispetto a parametri quali la ballabilità e la "liveness". L'attributo "speechiness" è di grande importanza in quanto riesce ad identificare opportunamente tracce esclusivamente discorsive come ad esempio interludi o poesie. Trovo tuttavia la costruzione del set dei pesi e le considerazioni che ne derivano puramente soggettive.

Capitolo 3

Il Software

3.1 Dati e metodologie

3.1.1 Algoritmi

Per quanto detto nel capitolo precedente, la scelta dell'algoritmo deve assecondare la struttura ad albero di cui abbiamo bisogno.
Il clustering gerarchico si presenta come scelta iniziale più naturale in questa prospettiva; è stata però valutata anche una eventuale clusterizzazione a un livello di profondità eseguita più volte con criteri sempre più restrittivi, ad esempio utilizzando l'algoritmo *kmeans*. Quest'ultima opzione permetterebbe anche di cambiare i pesi delle features ad ogni clusterizzazione, ottenendo una migliore flessibilità e dunque, potenzialmente, un miglior comportamento rispetto alle esigenze dell'utente.
Il problema di tale metodo è l'esponenziale costo computazionale e per questo la scelta è ricaduta sul Clustering Gerarchico.

Clustering Gerarchico

In inglese chiamato *Hierarchical Clustering*, costituisce una famiglia di algoritmi che crea raggruppamenti innestati unendoli o dividendoli in successione e costruendo dunque una *gerarchia di cluster*. I suoi risultati sono rappresentabili appropriatamente attraverso un dendogramma, struttura analoga appunto alla struttura ad albero precedentemente visualizzata.

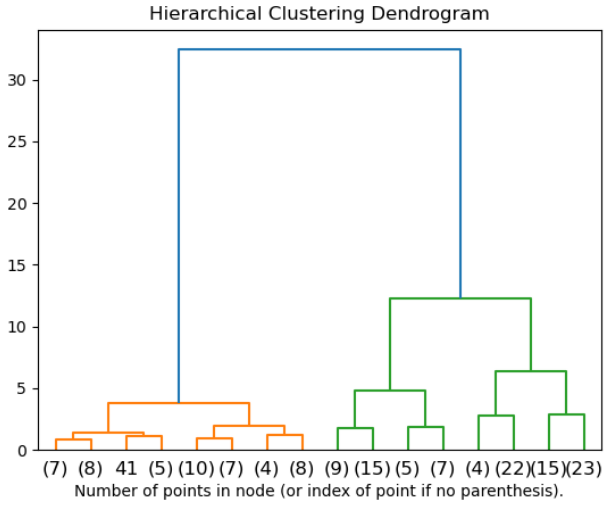


Figura 3.1: Esempio di dendogramma [14].

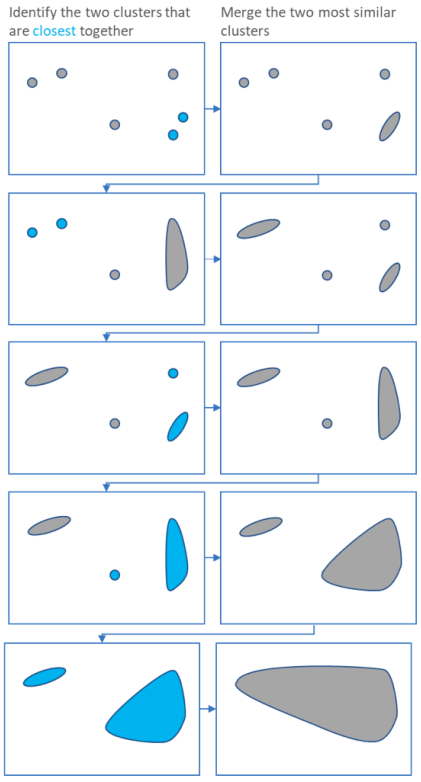
La famiglia si può dividere in due tipologie in base all'approccio utilizzato: dall'alto verso il basso o dal basso verso l'alto. Questa caratteristica definisce se si tratta, rispettivamente, di un algoritmo *agglomerativo (AGNES)* o *divisivo (DIANA)*.

Gli algoritmi agglomerativi inizialmente considerano ogni campione come un cluster. In seguito uniscono ad ogni iterazione i due cluster più vicini fino ad arrivare ad un unico gruppo, considerabile la radice del nostro albero gerarchico.
Meno impiegati rispetto ai primi principalmente per la loro difficoltà di realizzazione, gli algoritmi divisivi considerano invece l'insieme dei campioni come un unico cluster e si pongono l'obbiettivo di suddividerlo fino ad arrivare ai singoli campioni [3].

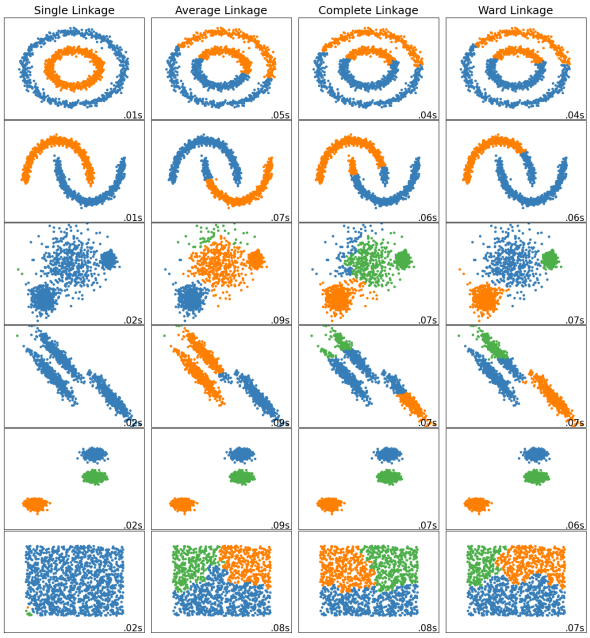
Oltre all'approccio utilizzato, è possibile decidere in che modo calcolare la distanza tra i cluster:

- Che tipo di distanza utilizzare, per esempio la Euclidea è quella prevalentemente utilizzata.
- Da che punto del cluster calcolare la distanza: anche detto "*Linkage Criteria*".

I *Linkage Criteria* proposti nell'implementazione dell'*Agglomerative Clustering* della libreria scikit-learn [14] sono:
Single Linkage: cerca la distanza minima tra i punti più vicini delle coppie.
Average Linkage: cerca la minima distanza media tra i punti delle coppie.
Complete Linkage: cerca la distanza minima tra i punti più lontani delle coppie.
Ward Linkage: sceglie i due cluster la cui unione minimizza la somma totale delle varianze interne di ogni cluster.



(a) Metodologia della clusterizzazione gerarchica agglomerativa [3].



(b) Visualizzazione dei risultati in base al tipo di *Linkage Criteria* [14].

La scelta di questo criterio va effettuata in base a considerazioni teoriche sul dominio dei campioni. Se non vi sono motivi chiari per la preferenza, solitamente la scelta ricade sul metodo di Ward [3].

BIRCH

Un tipo particolare di algoritmo appartenente alla famiglia da noi esaminata è il "*BIRCH*".
Il BIRCH crea una struttura ad albero con un approccio agglomerativo, ma è notevolmente migliore per quanto riguarda l'efficienza su grandi dataset.
I criteri di divisione e unione dei subcluster si basano principalmente su due parametri: il *threshold* e il *branching factor*, il primo limitante il raggio massimo dei subcluster, il secondo limitante il massimo numero di subcluster in un nodo.
La distanza dei subcluster viene valutata tra i rispettivi centroidi e l'algoritmo conserva nel subcluster tutti i parametri necessari a valutarne gli sviluppi, senza dunque dover mantenere in memoria tutti gli effettivi campioni appartenenti al cluster e le rispettive features. Questo permette di non avere sovraccarichi della memoria anche utilizzando una imponente raccolta di dati.
In seguito alla costruzione del "Clustering Features Tree", ossia dell'albero in questione, può essere effettuata una seconda clusterizzazione. Questo ulteriore passaggio può essere utile per ottenere un numero preciso di cluster senza utilizzare i parametri di *threshold* e *branching factor*. La scelta dell'algoritmo, spiegata più dettagliatamente in seguito, ricadrà proprio su quest'ultimo.

3.1.2 Parametri di Clusterizzazione

Al fine del raggruppamento delle tracce è stato scelto di utilizzare parametri numerici derivanti dall'analisi delle canzoni. Successivi sviluppi del software potrebbero comprendere anche l'utilizzo dei metadati delle canzoni. I parametri numerici sono offerti dalla API di Spotify e calcolati su server di loro proprietà mediante strumenti non condivisi col pubblico. Non si ha dunque alcun controllo sulla metodologia di calcolo utilizzata per tali parametri, nè si hanno informazioni a riguardo, ossia sulla effettiva definizione degli attributi data implicitamente dalla implementazione. Riporterò dunque tali parametri come descritti nella documentazione della API stessa traducendoli dall'inglese [1].

Attributi interi

Vi sono attributi interi che variano tra valori ben definiti e differenti per via della loro natura:

- key (int) [-1, 11]:
  La chiave stimata della traccia. È espressa da un numero intero usando la Classe di Altezze standard (standard Pitch Class notation). E.g. 0 = C, 2 = D, e così via. Se non è stata rilevata una chiave, il valore è pari a -1.

- mode (int - ma effettivamente è un bool) [0, 1]:
  La modalità della traccia, ossia se è in maggiore o minore: il tipo di scala da cui derivano i contenuti melodici della traccia. Maggiore è rappresentato da un 1, minore da uno 0.

- time signature (int) [1, ∞]:
  La stimata "time signature" della traccia. Questa è una convenzione notazionale che specifica quanti battiti ci sono in ogni barra.

- duration ms (int) [0, ∞]:
  La durata in millisecondi della traccia

- popularity (int) [0, 100]:
  La popolarità della traccia in una scala da 0 a 100, con 100 che indica la massima popolarità. Questo parametro è calcolato basandosi principalmente sul numero totale di ascolti della traccia e quanto questi sono recenti. Generalmente, canzoni che vengono riprodotte spesso recentemente avranno un valore maggiore rispetto a canzoni che venivano riprodotte molto in passato.

Attributi in virgola mobile

Vi sono i seguenti attributi *float*, che variano da 0 a 1:

- acousticness (float):
  Una misura riguardo la probabilità che una traccia sia acustica. 1.0 rappresenta una alta probabilità.

- danceability (float):
  La ballabilità descrive quanto una traccia è ottimale per ballare basandosi su una combinazione di elementi del pezzo tra cui il tempo, la stabilità del ritmo, la forza dei battiti e la regolarità generale. Un valore pari a 0.0 è il meno ballabile e 1.0 è il più ballabile.

- energy (float):
  L'energia è una misura percettiva di intensità e attività all'interno della traccia. Tipicamente, tracce energiche suonano veloci, a volume alto e rumorose. Ad esempio, death metal ha una alta energia, mentre un preludio di Bach ha un punteggio basso in questa scala. Parametri percettivi che contribuiscono a questo attributo includono il dynamic range, il volume percepito, il timbro, l'onset rate e l'entropia generale.

- instrumentalness (float):
  Predice se una traccia non contiene vocali. "Ooh" e "aah" sono suoni trattati come strumentali in questo contesto. Rap o parole parlate sono invece chiaramente vocali. Più il valore si avvicina a 1.0, più è grande la probabilità che la traccia non contenga vocali. Valori sopra lo 0.5 intendono rappresentare tracce strumentali, ma la accuratezza aumenta con l'approcciare del valore a 1.0.

- liveness (float):
  Rileva la presenza di un audience nella registrazione. Un valore prossimo all'1 di "liveness" rappresenta una probabilità alta che la traccia sia stata registrata a un concerto, invece un valore superiore allo 0.8 ne indica una buona probabilità.

- speechiness (float):
  La discorsività rileva la presenza di parole parlate in una traccia. Più questa è parlata (e.g. talk show, audio book, poesia), più il valore è vicino a 1.0. Valori superiodi a 0.66 descrivono tracce probabilmente fatte interamente da parole parlate. Valordi tra lo 0.33 e 0.66 descrivono tracce che possono contenere sia musica che parlato, sia in sezioni diverse che insieme, includendo dunque casi simili al rap. Valori sotto il 0.33 rappresentano musice probabilmente priva di parole parlate.

- valence (float):
  Descrive la positività musicale espressa da una traccia. Tracce ad alta valenza suonano più positive (e.g. felici, euforiche), mentre tracce a bassa valenza suonano più negative (e.g. tristi, depresse, arrabbiate).

Vi sono inoltre il volume (loudness), *float* che varia tipicamente tra -60 e 0 dB, e il tempo, *float* che varia tra 0 e infinito con valori tipici tra gli 80 e i 200 BPM, parametri dunque da normalizzare prima di essere usati nella clusterizzazione:

- loudness (float):
  Il volume medio della traccia in decibels (dB). La media è effettuata sull'intera traccia e può essere utile per comparare i relativi volumi di più tracce. Il volume è fortemente correlato alla forza fisica del suono (ampiezza).
- tempo (float):
  Stima del tempo della traccia in battiti al minuto (BPM). Seguendo la terminologia musicale, il tempo è la velocità di un pezzo e deriva direttamente dalla durata media dei battiti.

Attributi di identificazione

Vi sono infine parametri volti alla identificazione della traccia e alla sua denominazione:

- name, il nome della canzone;
- disc number, se un album è costituito da più dischi indica il numero del disco;
- track number, il numero della canzone dentro all'album;
- id, l'ID della canzone nel database di Spotify (e anche nel dataset che costruiremo noi).

Per l'utilizzo dei dati sopraelencati è stato progettato il modello ER seguente.

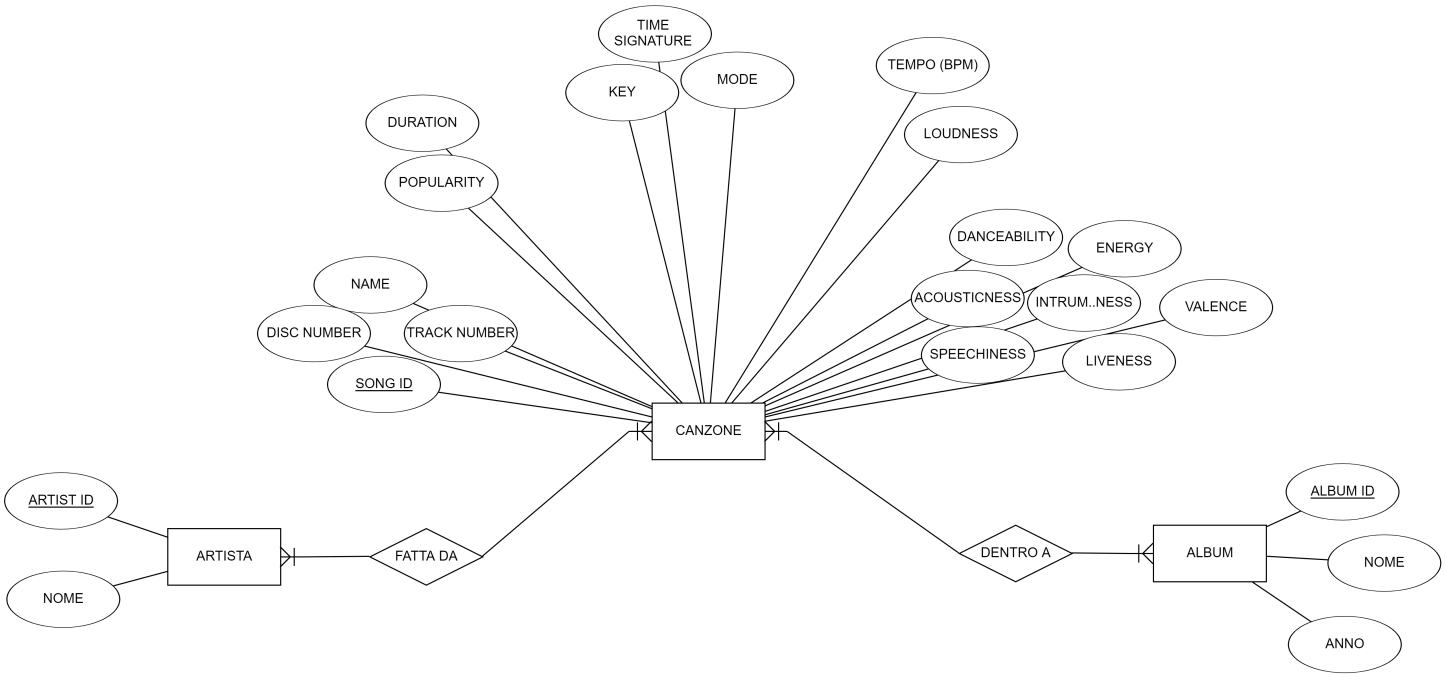


Figura 3.3: Modello ER costruito.

Ulteriori dati reperibili

La API di Spotify, oltre alle *feature* a cui ci siamo riferiti sopra, permette di ottenere un'analisi più dettagliata tramite la frammentazione della canzone in sezioni più piccole.

Un primo tipo di frammentazione fornisce informazioni attraverso due tipi di oggetti singolari: i "segments" e le "sections".
I **segments** forniscono una descrizione dei suoni all'interno della canzone: ogni segmento è caratterizzato da un array di 12 pitch e 12 timbri a identificarne la valenza strumentale.
Le **sections** sono invece suddivisioni della canzone in tempi più larghi, caratterizzate da key, mode, time signature e loudness propri. Con le sezioni riusciamo a identificare cambi di ritmo o timbro, cosa frequente quando si passa dai versi al ritornello o a un assolo di chitarra.

Oltre a questo tipo di suddivisione, abbiamo a disposizione un'ulteriore frammentazione più semplice, consistente di tre diversi "time interval object", intervalli di tempo caratterizzati solo dal secondo di inizio e dalla durata:

- tatums:
  Un "tatum" rappresenta il più piccolo impulso regolare a cui un ascoltatore inferisce intuitivamente grazie al tempismo degli eventi musicali percepiti.
- beats:
  L'unità di misura basica di un pezzo; ad esempio, ogni tick di un metronomo. I battiti sono tipicamente multipli dei "tatum".
- bars (or measure):
  Segmenti di tempo definiti da un dato numero di battiti. L'offset delle barre indicano implicitamente i downbeat, ossia i primi beat della barra.

I metodi da me implementati nel software prevedono già la possibilità di reperire anche queste ulteriori informazioni, ma non ne è stato progettato l'utilizzo.
Successivi sviluppi del software potrebbero far uso anche di queste informazioni.

Figura 3.4: Schema di frammentazione dell'API di Spotify [1].

3.2 Implementazione

3.2.1 Clustering

Considerando possibili sviluppi futuri del progetto il BIRCH è sembrata la scelta ottimale per quanto riguarda l'algoritmo. Permette infatti di ottenere una buona scalabilità, l'utilizzo di grandi dataset, fattore necessario per il software in questione e permette inoltre di rieffettuare la clusterizzazione in seguito a un inserimento di nuovi campioni, senza dover riutilizzare i vecchi.

L'implementazione dell'algoritmo utilizzata è quella fornita dalla libreria scikit-learn, scritta in Python [9].
Questa prevede l'utilizzo dei classici parametri di *threshold* e *branching factor* e utilizza un algoritmo standard di clustering agglomerativo per il passaggio finale utilizzabile nel caso in cui si voglia specificare solo il numero di cluster finali. A seguito della clusterizzazione, questa implementazione fornisce le etichette di appartenenza di ogni campione ai subcluster foglia, ossia al livello di suddivisione maggiore.
È risultata necessaria una sua modifica ai fini dell'utilizzo nel software:

- è stato aggiunto alla classe dei subcluster un campo denominato "*samples*" in cui vengono salvati gli indici dei campioni che gli appartengono. Questo riduce leggermente l'efficienza sulla memoria dell'algoritmo, di base comunque molto alta. È importante notare però che l'unica cosa che viene salvata è un numero intero riferito al campione e non l'intero set di features;
- il metodo di predizione è stato estrapolato dalla classe al fine di essere utilizzato a qualunque livello della struttura ad albero e non solo sulle foglie, come previsto precedentemente dall'implementazione.

Grazie alla prima modifica è possibile navigare all'interno della struttura gerarchica a partire dalla radice, ottenendo per ogni nodo i rispettivi subcluster e la loro composizione.
L'estrapolazione ed adattamento al contesto del metodo di predizione ha invece lo scopo di collocamento nei cluster delle tracce scelte dall'utente per l'orientamento.

Per quanto riguarda i parametri di *threshold* e *branching factor* sono stati scelti come valori rispettivamente 0.15 e 15 allo scopo di ottenere una buona clusterizzazione e una struttura navigabile dell'albero.

Preliminarmente all'utilizzo dell'algoritmo di machine learning vengono effettuate varie operazioni:

- prima di tutto viene effettuato il filtraggio di parametri attraverso l'utilizzo di una blacklist da parte dell'utente, questo risulta utile per escludere quelli non utilizzabili o poco interessanti per la clusterizzazione, come il numero cardinale assegnato alla canzone nel disco di appartenenza o la durata della canzone. Questa funzionalità può comunque essere ottenuta anche semplicemente impostando il peso del parametro da ignorare sullo 0;
- la normalizzazione di dati al fine di ottenere una scala omogenea che va da 0 a 1. Un esempio di utilizzo è riferibile al parametro "tempo", normalizzato in un range variante tra i 50 e i 250 BPM;
- la applicazione dei pesi dati ai parametri, effettuata moltiplicando il parametro per un fattore di peso anch'esso definito tra 0 e 1. La moltiplicazione del parametro allarga infatti la scala di definizione del parametro, permettendogli di avere un maggior peso rispetto a quelli con una scala standard. Nel mio caso, la moltiplicazione per un numero tra 0 e 1 porta in realtà a un restringimento proporzionale della scala dei parametri meno importanti.

3.2.2 Popolamento e gestione del Database

Come già affermato, il progetto ottiene i dati attraverso la "Application Programming Interface" di Spotify.
Il popolamento del Database è scritto in Python e utilizza la libreria *requests* [16].
Acquisite le informazioni, queste vengono incapsulate in oggetti di tipo Dataset, poi formattati e salvati in *dump* appositi. Il software permette tuttavia, tramite l'impostazione di un semplice valore booleano, di salvare il Dataset in vari file di formato ".csv" etichettati e catalogati in cartelle nel caso in cui dovesse essere utile. Per la gestione del database e per la realizzazione dello schema relazionale progettato, si utilizza la libreria di funzioni *pandas* [17].

Index	acousticness	album_id	artists_id	danceability	disc_number	duration_ms	energy	id	instrumentalness	key
0	0.00168	7JggdVIipgSS…	1wNYYqNhLKSd…	0.68	1	234653	0.813	1VfhO1GZsZmV…	5.74e-05	1
1	0.366	5NKTuBLCYhN0…	1aSxMhuvixZ8…	0.481	1	269067	0.392	5YSI1311X8t3…	0.0452	1
2	0.439	0T5C5nZUcMAq…	0bwaKyQeHXTy…	0.378	1	192453	0.425	30XCZtcaWi6P…	0.00824	0

liveness	loudness	mode	name	playlist	popularity	speechiness	tempo	time_signature	track_number	valence
0.081	-5.672	1	Me Me Me - Trentemøller…	dataset	25	0.0379	120.03	4	9	0.915
0.124	-11.381	1	Wuthering Heights	dataset	69	0.0312	124.572	4	6	0.52
0.256	-12.747	1	Gambale Twist	dataset	13	0.0783	167.149	4	1	0.443

Figura 3.5: Screenshot dell'entità canzone all'interno del dataset

3.2.3 Il motore grafico e gli asset utilizzati

Nel progetto è stata data particolare importanza al design dell'interfaccia e al suo funzionamento utilizzando un motore grafico, software sviluppato al fine di gestire la grafica in tempo reale di videogiochi e contenuti interattivi.
Solitamente per lo sviluppo di applicazioni di questo tipo, è necessario risolvere problemi simili tra loro, quali ad esempio il rendering e la fisica degli oggetti; questi problemi vengono risolti in maniera modulare e riutilizzabile dai motori grafici in commercio, capita tuttavia nell'industria videoludica di avere la necessità di sviluppare un motore grafico appositamente progettato per un solo gioco.
Tipicamente comprendono un motore di rendering per la resa visuale degli oggetti all'interno di una ambientazione, un motore fisico che gestisce le collisioni e le forze applicate sugli oggetti della scena, supporto per scripting, networking, intelligenze artificiali.

Il motore grafico da me utilizzato è stato sviluppato da Unity Technologies ed è particolarmente noto in questo ambito. Come punti di forza di quest'ultimo si possono elencare la vasta gamma di classi e funzioni di cui sono dotate le sue librerie e la ottima documentazione annessa che favorisce l'apprendimento del framework. Inoltre, permette la compilazione per l'esecuzione in una ampia selezione di piattaforme [20].

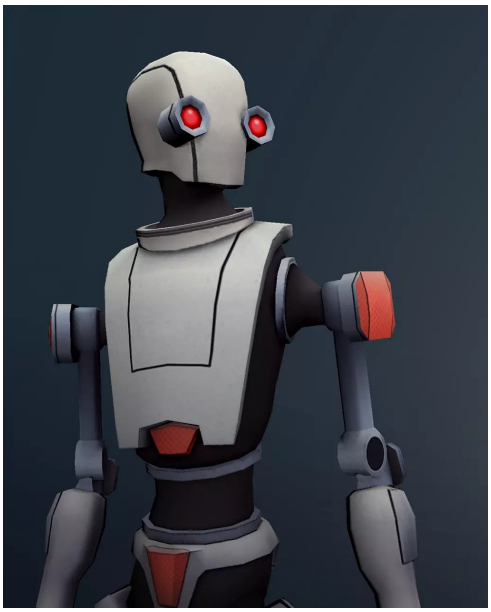
Lo sviluppo delle **dinamiche dell'interfaccia** in Unity avviene attraverso il linguaggio C#.

Per gli scopi del progetto in questione si è dovuto progettare: il movimento e controllo del personaggio insieme alla gestione delle sue animazioni, tra le quali a mio parere non poteva mancare una animazione di ballo; il reperimento dei risultati dagli script eseguiti in python per il clustering; l'istanziamento a tempo di esecuzione dei cluster nelle loro rispettive posizioni; la gestione e il collegamento con i dati degli elementi di UI, tra i quali la mappa, il menù dei cluster e il display delle coordinate.

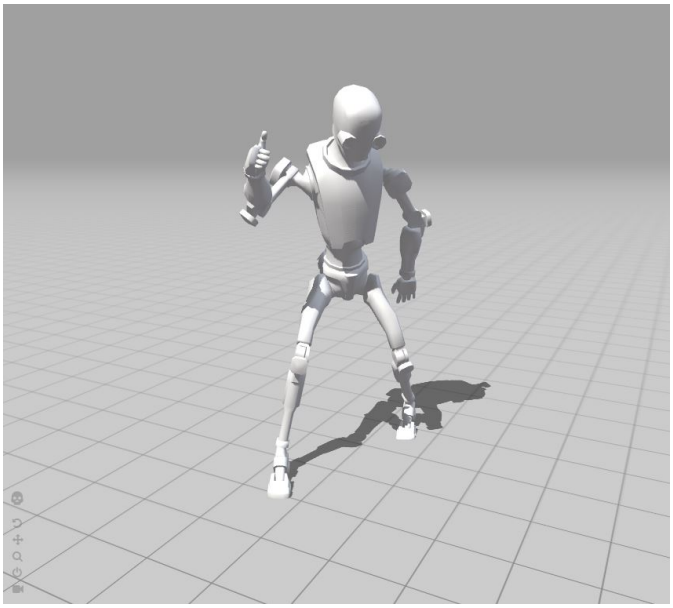
L'ambientazione scelta per "Portamento" vuole rievocare l'estetica dello Spazio e gli asset utilizzati per la composizione della scena sono stati scelti appositamente conformi a questa ambientazione.
Protagonista dell'interfaccia è *Kyle*, robot dotato della possibilità di volare. L'utente prenderà il comando di Kyle allo scopo di vagare per lo Spazio alla ricerca di nuove canzoni.

Tutti gli asset sono stati scaricati dal negozio apposito ufficiale di Unity. Le animazioni del personaggio principale provengono invece dal sito Mixamo, di proprietà di Adobe [11].
Riporto l'elenco degli asset e le rispettive fonti di seguito:

- Milky Way Skybox [2];
- Space Robot Kyle [18];
- Unity Samples: UI [19];
- Vast Outer Space [4];



(a) Robot Kyle.



(b) Interfaccia di Mixamo per le animazioni.

3.3 Struttura schematica del software

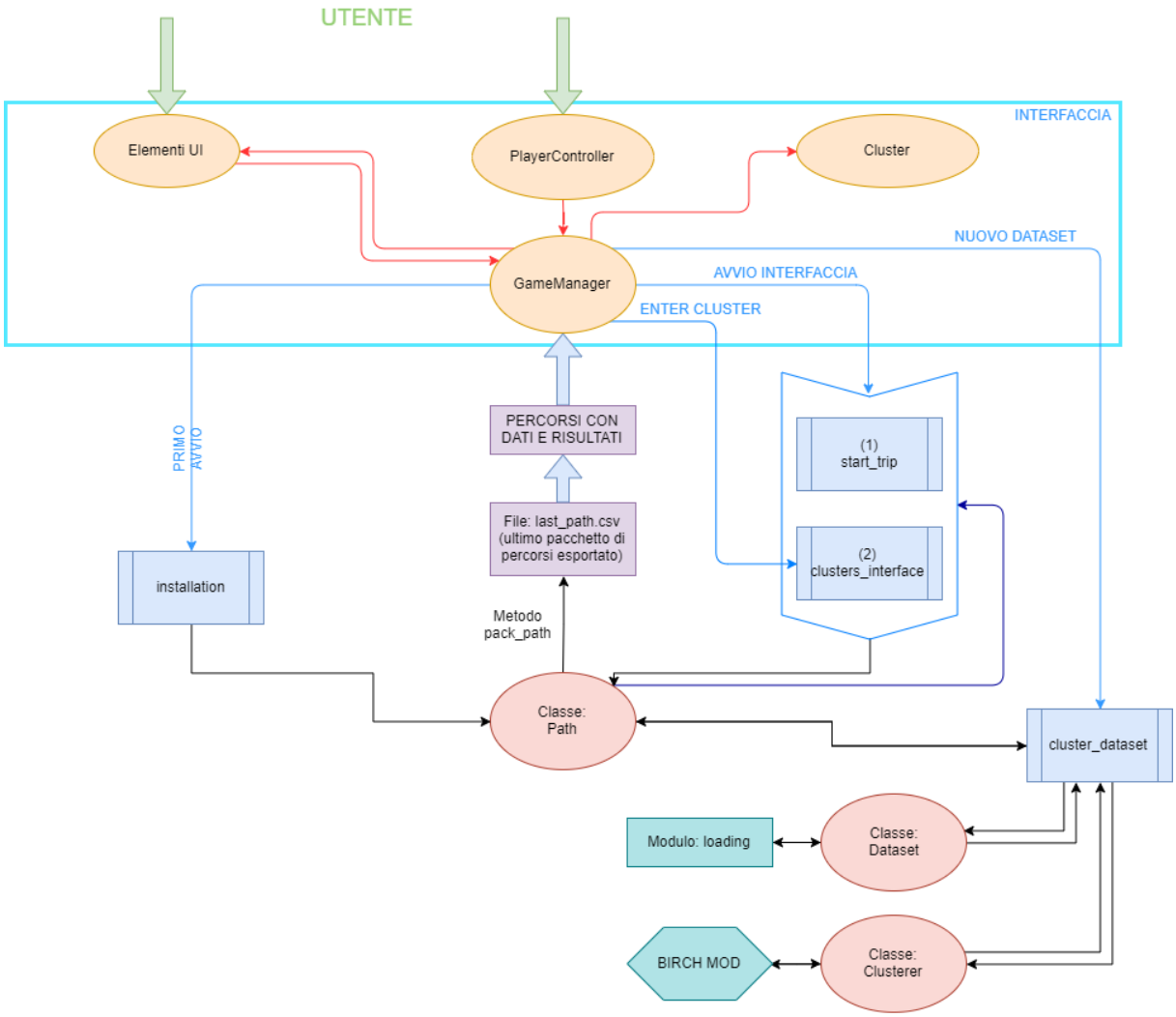


Figura 3.7: Schema indicativo sul funzionamento del software.

La struttura del software è divisa in **backend** e **interfaccia (o frontend)**. Quest'ultima è a diretto contatto con l'utente, gli fornisce le informazioni e riceve gli input da esso. Il GameManager si occupa della gestione degli elementi visualizzati dall'interfaccia e fa dunque da tramite tra l'utente e il backend: in seguito ad opportuni comandi avvia gli script di questo, evidenziati nel diagramma dal colore blu di sfondo. Gli script utilizzano le classi e i moduli rappresentati nello schema per scrivere in file i propri risultati. La classe Path si occupa della gestione dei percorsi di restituzione.

Gli script in questione sono i seguenti:

- *installation*, che si occupa della creazione dei primi percorsi importanti al primo avvio;
- *cluster dataset*, che si occupa della clusterizzazione dei dataset nuovi o di quello predefinito;
- *start trip*, che si occupa del selezionamento del dataset e delle canzoni radar da utilizzare prima dell'avvio dell'interfaccia;
- *clusters interface*, che si occupa di restituire i cluster da visualizzare nell'interfaccia ottenendo in ingresso il codice corrispondente alla posizione nell'albero. Per farlo, utilizza la classe Clusterer per ottenere la struttura ad albero preliminarmente creata. All'interno di questo script vi è inoltre la funzione predict estrapolata dalla classe Birch e utilizzata in questo contesto per assegnare le canzoni radar ai cluster da mostrare nella scena.

Questi fanno uso delle seguenti **classi** e **moduli** da me progettati:

CLASSI:

- *Path*, classe usata estensivamente in tutte le altre classi e script. Incapsula tutti i percorsi utilizzati dal software, creandoli se inesistenti (ad esempio all'inserimento di un nuovo dataset da clusterizzare) o collegandoli ai propri campi se invece già esistenti.
  Elencare i campi di questa classe sarebbe ridondante in quanto corrispondono a tutti i singoli percorsi necessari al funzionamento del software.
  Di seguito i metodi più importanti per dare un'idea del funzionamento della classe:
  - `initialize default files`;
  - `link database`;
  - `link radar`;
  - `delete saved clusters`;
  - `pack paths`, metodo particolarmente importante per la interfaccia in quanto impacchetta i percorsi del dataset a cui è collegata l'istanza di questa classe, permettendo all'interfaccia di ottenere dunque i percorsi giusti del dataset da visualizzare.
- *Dataset*, classe che si occupa dell'ottenimento e gestione del Dataset, composta dai campi:
  - `dataset`, dizionario composto dai campi track (l'effettivo set di features), albums e artists;
  - `is radar`, booleano che indica se abbiamo un dataset composto da canzoni radar, ossia quelle che servono all'orientamento dell'utente. Questo perchè queste devono essere salvate in altri percorsi del software per essere distinguibili dai normali dataset;
  - `save dataset`, booleano che indica se salvare in ".csv" il dataset per una visualizzazione esplicita attraverso i file. Se falso, il dataset viene comunque salvato in un file di dump.

  E dai metodi:
  - `get and save dataset`;
  - `format playlist`;
  - `format features`;
  - `save dataset key`, che salva un campo del dizionario del dataset, fornitagli la chiave del campo;
  - `format analysis`;

  - `save analysis`, metodo che salva l'analisi delle canzoni ricavata attraverso i dati descritti al capitolo 3.1.2, non ancora utilizzati dal software;
  - `save n_songs`;
  - `save n_playlists`.

- *Clusterer*, classe che si occupa del clustering composta dai campi:
  - `return clusters`, booleano che decide se restituire la composizione dei cluster;
  - `save final clusters`, booleano che decide se salvare i cluster finali su file;
  - `detaset`, il dataset da clusterizzare;
  - `audio relevant columns`, lista di nomi delle colonne del dataset da utilizzare;
  - `model`, il modello utilizzato, salvato in seguito alla clusterizzazione;
  - `weights`, il set dei pesi.

  E dai metodi:
  - `cluster new dataset`;
  - `get blacklist`, ottiene le colonne da eliminare dalla blacklist;
  - `get relevant columns`, ottiene le colonne da utilizzare a partire da quelle da eliminare;
  - `save clusters`;
  - `save centroids`;
  - `format dataset`, diverso dai format presenti nella classe Dataset: esegue la formattazione volta al clustering, comprendente normalizzazione e l'applicazione dei pesi;
  - `normalize column`;
  - `filter weights`, elimina i parametri da non utilizzare dal set dei pesi;
  - `weight`, la funzione che esegue la effettiva moltiplicazione per applicare il peso a una colonna del dataset;
  - `weight features`, utilizza la funzione `weight su ogni colonna`.

- *Tab*, struttura in cui sono elencati i preset utilizzabili dal software, utilizzata per i preset di weights, datasets e axis. È composta dai campi:
  - `table type`;
  - `table`.

  E dai metodi:
  - `create`;
  - `save`;
  - `get`;
  - `table path`, metodo che contiene uno switch restituente il percorso a seconda del `table type`.

- *Tables*, classe contenente tutti i tab utili al software collegati ai loro percorsi. Composta dai campi:
  - `weights`;
  - `datasets`;
  - `axis`.

MODULI:

- *loading*, modulo contente le funzioni utilizzate all'interno della classe Dataset per l'acquisizione delle informazioni ed elencate di seguito:
  - `get song analysis`, che incapsula la richiesta e la restituisce, esattamente come le altre `get` che seguono;
  - `get playlist`;
  - `get features`, questa funzione incapsula la richiesta delle features e gestisce il limite di 100 ID imposto da Spotify utilizzando un parametro di offset da inserire nella richiesta: viene eseguita ricorsivamente fino ad aver ottenuto il numero di canzoni totali da ottenere;
  - `get oauth`, con la funzione di ottenere il token di autorizzazione per utilizzare la API di Spotify;
  - `get url request`, crea l'URL a cui fare la richiesta in base al tipo di richiesta data in ingresso;
  - `request thing`, esegue la effettiva richiesta e viene incapsulata dalle funzioni `request` che seguono;
  - `request features`;
  - `request audio analysis`;
  - `request playlist`;
  - `get items`, come per le features, la richiesta della playlist è limitata a 100 canzoni acquisibili, questa funzione gestisce l'offset di questo tipo di richiesta. È stata creata una funzione apposita per rispondere ad esigenze di funzionamento diverse da quella del `get features`;
  - `request items`.
- *birch mod*, modulo contente la classe `BIRCH` da me modificata e tutte le classi di cui essa era composta nella libreria scikit.

Per il **frontend**, scritto in C#, sono state progettate le seguenti classi:

- *GameManager*, si occupa dell'acquisizione dei risultati del **backend**, della traduzione dei file ".csv" in Liste di Dizionari utilizzabili nell'interfaccia e del comando degli elementi dell'interfaccia;
- *PlayerController*, si occupa del movimento del personaggio e della risposta ai comandi;
- *Cluster*, si occupa del posizionamento e comportamento del cluster; fornisce inoltre le giuste informazioni al menù apposito.

Oltre alle tre descritte, sono state create ulteriori classi per ogni elemento dell'UI da comandare per gestire la visualizzazione delle informazioni.

Capitolo 4

Utilizzo

4.1 Funzionamento e risultati

All'avvio del programma l'utente potrà subito programmare e iniziare una esplorazione.

Per la creazione del viaggio serve innanzitutto inserire una playlist di Spotify.
La prima della lista sarà la **canzone di partenza**, che costituirà appunto la **posizione** iniziale. Tutte le canzoni della lista saranno sempre presenti nell'interfaccia all'interno di una mappa con la funzione di orientare l'utente; una volta selezionate all'interno nella mappa verrà infatti evidenziato il cluster a cui appartengono tra quelli presenti nella scena. Per ora, non più di 11 canzoni saranno accettate per la costituzione di tale funzione "*radar*".

Altri parametri controllabili dall'utente, ma di cui sono presenti valori predefiniti, sono:

- il Database dentro al quale navigare, il cui valore predefinito è una playlist da più di 5000 canzoni dai generi più disparati prelevata dal servizio *Every Noise At Once* [7];
- le tre caratteristiche corrispondenti ai tre assi di movimento dello spazio tridimenzionale;
- i pesi della clusterizzazione.

Nel caso in cui si voglia cambiare il Database, verrà avviato uno script scritto in python per la clusterizzazione dello stesso attraverso il modello BIRCH modificato.

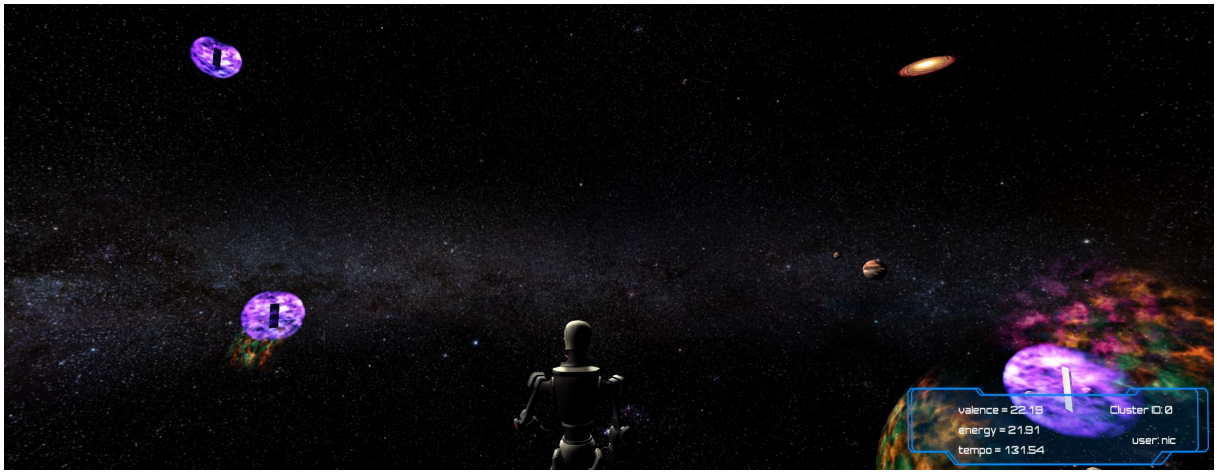


Figura 4.1: I primi passi dentro all'interfaccia.

Ideato il viaggio, l'utente può ora catapultarsi nell'universo clusterizzato, con la possibilità immediata di ascoltare musica e di navigare verso svariati generi musicali.
All'angolo dello schermo viene evidenziata la posizione attraverso le coordinate tridimensionali e un codice associato al cluster, identificante la posizione nell'albero. Il primo numero del codice, sempre pari a 0, indica il cluster radice, i seguenti corrispondono alle scelte dell'utente.

Utilizzando il comando designato, si può aprire la **mappa**. Premendo su una delle canzoni-radar verrà evidenziato con un verde acceso il cluster di appartenenza all'interno della mappa; questo diventerà inoltre della stessa tonalità anche all'interno dello spazio, facilitandone l'individuazione.

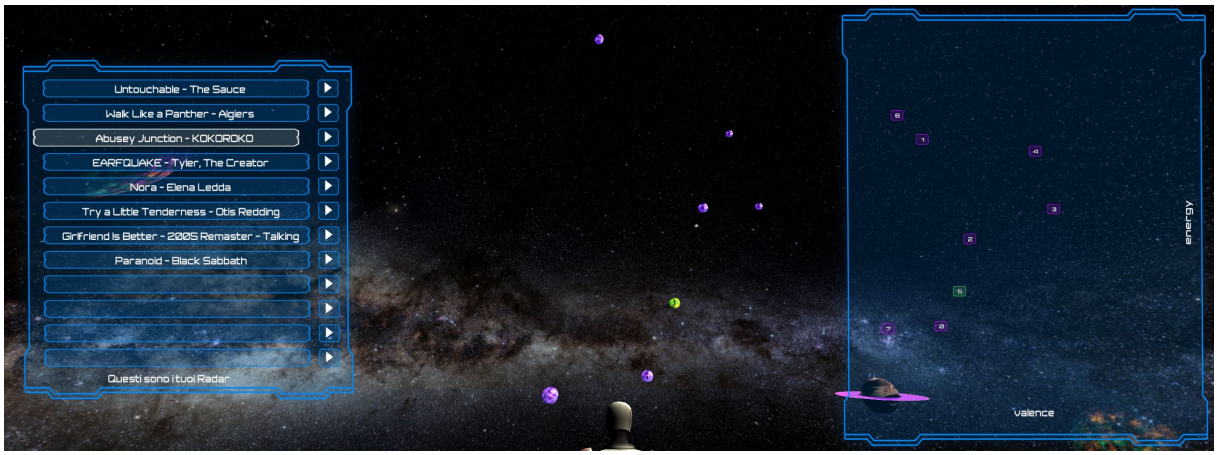


Figura 4.2: La lista dei radar composta da canzoni di vario genere e la mappa di fianco.

I cluster sono facilmente individuabili per la loro conformazione e luminanza. Avvicinandosi si può aprire il rispettivo menù, che ci permette di ascoltare le canzoni appartenenti al cluster e ottenere informazioni su esse che Spotify fornisce esclusivamente attraverso la sua API. Da qui, la funzione di spostamento all'interno dell'albero attraverso il tasto "*Enter Cluster*": in seguito all'azionamento di questa funzione, si entra all'interno di un universo composto dalle sole canzoni del cluster selezionato, anch'esse divise in ulteriori raggruppamenti.

Figura 4.3: Il menù di un cluster.

I risultati ottenuti dall'algoritmo di clusterizzazione sono accurati in particolare osservando i cluster foglia l'uno rispetto all'altro. Si possono visualizzare tuttavia risultati interessanti anche subito al nodo radice.
Un esempio è il cluster in Figura 4.3, ad elevata energia e medio-alta ballabilità, che risultava composto innanzitutto da ritmi simili a ciò che definiamo reggaeton ma comprendente anche electropop e musica indiana con un simile orientamento.
Un altro esempio potrebbe essere un particolare cluster trovato anch'esso al nodo radice, composto esclusivamente da discussioni parlate; la caratteristica principale risultava essere evidentemente la elevata "speechiness".

Le somiglianze dei pezzi appartenenti ai cluster sono evidenziate subito osservando il display delle caratteristiche delle canzoni che la compongono: i simboli, utilizzati come indici di intensità delle caratteristiche, forniscono infatti immediatamente una indicazione visiva in aggiunta al voto in centesimi.

4.2 Altri possibili utilizzi

La possibilità di scegliere un Database alternativo permette, a chi ne avesse bisogno, di utilizzare il software per organizzare set di musica che già conosce.
Nel mestiere del DJing, ad esempio, potrebbe essere utile questa sfaccettatura, in quanto è necessario riuscire a selezionare, tra le proprie conoscenze, insiemi di tracce con una certa coerenza tra loro.

L'interfaccia e il cuore del software sono progettati in maniera modulare e possono, con opportuni accorgimenti, essere utilizzati per scopi diversi anche singolarmente. Si potrebbe ad esempio utilizzare esclusivamente l'interfaccia per navigare attraverso una struttura ad albero generica, oppure, si potrebbe utilizzare esclusivamente il cuore del software per l'acquisizione dei dati da Spotify e per la clusterizzazione ma progettando una interfaccia totalmente diversa.

Capitolo 5

Possibili miglioramenti e sviluppi futuri

Le similarità tra brani considerate fino ad ora sono esclusivamente sonore o di ritmo.
Non si è considerato in alcun modo il contesto culturale di ideazione del pezzo, ossia ad esempio la città di origine dell'artista (o in modo più generale la nazionalità), l'anno di pubblicazione o la lingua del testo. Un primo sviluppo del software potrebbe dunque consistere nell'introduzione di queste informazioni come ulteriori parametri di clusterizzazione.
Se volessi, ad esempio, ascoltare strettamente musica proveniente dalla Sardegna, potrebbe essere utile applicare un filtro geografico che escluda in anticipo dall'interfaccia tutto ciò che non è sardo.
Una ipotetica implementazione di tali informazioni testuali potrebbe essere analoga a quella utilizzata dal Professor Giacinto e colleghi al fine di ottenere immagini rilevanti in ricerche riguardanti attrazioni turistiche [5]. A questo scopo è stata utilizzata la possibilità del secondo passaggio di clusterizzazione dell'algoritmo BIRCH, ma scambiando le informazioni estratte dalle immagini con i loro metadati.
Per l'ottenimento di dati utili di questo tipo l'enciclopedia **MusicBrainz** potrebbe essere la principale fonte di informazioni. Il servizio fornisce infatti al pubblico un ampio database gestito dai suoi utenti con ricchi e vari metadati riguardanti canzoni da ogni parte del mondo [13].

Un altro possibile miglioramento potrebbe consistere nell'utilizzo di un algoritmo che permetta di avere diversi pesi delle features ai differenti livelli della gerarchia. La musica è infatti notevolmente divisibile in generi e sottogeneri gerarchicamente, ma spesso viene data più importanza a una certa caratteristica quando si parla di "macrogeneri" e si passa ad altri tipi di distinzione quando si entra nel particolare.

I risultati forniti dal software non sono **sempre** condivisibili: i parametri prelevati da Spotify sono di fatto relativamente arbitrari e capita che non rispecchino il valore che ci aspetteremmo; tuttavia, vi sono sicuramente margini di miglioramento dipendenti da fattori da noi controllabili. I parametri di *threshold* e *branching factor*, ad esempio, sono stati scelti con il compromesso di facilitare la navigazione all'interno dell'albero, un primo miglioramento dei raggruppamenti lo si può avere dunque modificando questi parametri perdendo però navigabilità dell'interfaccia.

Infine, un ulteriore sviluppo consisterebbe nell'utilizzo dei parametri aggiuntivi di cui si parla al Capitolo 3.1.2, già prelevabili dal software. Questi sono interessanti tanto quanto quelli già utilizzati, ma presentano una maggiore difficoltà di utilizzo.

Ringraziamenti

Se ho raggiunto questo traguardo conservando una sufficiente sanità mentale (o così ritengo), lo devo anche a varie persone.
Mamma, babbo, grazie all'infinito, in particolare per il conforto nella debolezza e per spingermi a sentirmi inarrestabile nell'affrontare un'ambizione. Pino, grazie perchè so con sicurezza che ci sarai, se mai mi servisse sapere il nome di Goldaniga ad esempio. Sei la cosa migliore mi sia capitata. Al fianco tuo non ho paura.
Ringrazio la mia famiglia intera, dai luculliani Loddo agli strampalati Perrera, che ha allestito l'asfalto e l'iniziale guard-rail del tragitto percorso. Siete ciò di più puro che mi rimane, grattando fino all'osso. Tra questi, un grazie particolare a nonne e nonni, per la genuinità con cui mi avete cresciuto e amato.
Ringrazio Ste, zio non di sangue ma di cuore. Sei famiglia. Grazie per le favole e per avermi spinto a inventarne sempre di nuove.

Grazie ai Perperieri, alle Luiselle e tutti gli altri fratelli. Siete come rifugio e sebadas. Spero sappiate che, comunque vada la mia vita, perlomeno uno spazio sotto al cavalcavia potrò farvelo sempre.
Grazie a Nico per amorevolmente non accettarmi come sono; grazie a Giuli per amorevolmente accettarmi come sono.
Grazie alle delusioni e ai traguardi.
Grazie a Gianni per le favole al telefono.
Grazie a prof. Giacinto, prof. Nespolo, mister Saba, coach Sergio, Dr. House e l'Uomo Tigre.

Bibliografia

[1] Spotify AB. *Spotify Web API Documentation*. URL: https://developer.spotify.com/documentation/web-api/.

[2] Adam Bielecki. *Milky Way Skybox*. URL: https://assetstore.unity.com/packages/2d/textures-materials/milky-way-skybox-94001.

[3] Tim Bock. *What is Hierarchical Clustering?* URL: https://www.displayr.com/what-is-hierarchical-clustering/#:~:text=Hierarchical%20clustering%2C%20also%20known%20as,broadly%20similar%20to%20each%20other..

[4] Prodigious Creations. *Vast Outer Space*. URL: https://assetstore.unity.com/packages/3d/environments/sci-fi/vast-outer-space-38913.

[5] Duc-Tien Dang-Nguyen et al. "Multimodal Retrieval with Diversification and Relevance Feedback for Tourist Attraction Images". In: *ACM Trans. Multimedia Comput. Commun. Appl.* 13.4 (ago. 2017). ISSN: 1551-6857. DOI: 10.1145/3103613. URL: https://doi.org/10.1145/3103613.

[6] *discoverquickly.com*. URL: https://developer.spotify.com/community/showcase/discover-quickly/.

[7] *Every Noise at Once*. URL: http://everynoise.com/.

[8] Proma Huq. "Music To My Ears: De-Blackboxing Spotify's Recommendation Engine". In: *CCTP-607 Big Ideas in Tech: AI to the Cloud* (2019). URL: https://blogs.commons.georgetown.edu/cctp-607-spring2019/2019/05/06/music-to-my-ears-de-blackboxing-spotifys-recommendation-algorithm/.

[9] *Implementazione del BIRCH*. URL: https://github.com/scikit-learn/scikit-learn/blob/95119c13a/sklearn/cluster/_birch.py#L326.

[10] Glenn McDonald. "How We Understand Music Genres". In: (). URL: http://everynoise.com/EverynoiseIntro.pdf.

[11] *Mixamo*. URL: www.mixamo.com.

[12] F. Morchen et al. “Modeling timbre distance with temporal statistics from polyphonic music”. In: *IEEE Transactions on Audio, Speech, and Language Processing* 14.1 (2006), pp. 81–90. DOI: 10.1109/TSA.2005.860352.

[13] *MusicBrainz encyclopedia*. URL: https://musicbrainz.org/.

[14] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[15] *Portamento nel Vocabolario Treccani*. URL: https://www.treccani.it/vocabolario/portamento/.

[16] *Requests Python Library*. URL: https://requests.readthedocs.io/en/master/.

[17] The pandas development team. *pandas-dev/pandas: Pandas*. Ver. latest. Feb. 2020. DOI: 10.5281/zenodo.3509134. URL: https://doi.org/10.5281/zenodo.3509134.

[18] Unity Technologies. *Space Robot Kyle*. URL: https://assetstore.unity.com/packages/3d/characters/robots/space-robot-kyle-4696.

[19] Unity Technologies. *Unity Samples: UI*. URL: https://assetstore.unity.com/packages/essentials/unity-samples-ui-25468.

[20] *Unity Engine*. URL: unity.com.