Univeristà degli Studi Roma Tre

Utilizzo di Word2Vec per un Sistema di Raccomandazione Musicale

Hermann Tamilia (558851), Matteo Vitale (546622) and Irene Lontano (560298)

Questo lavoro è stato svolto per l'esame di Machine Learning

Abstract

La continuazione automatica delle playlist è una sfida significativa nella raccomandazione musicale, poiché la creazione manuale delle playlist è dispendiosa e complessa a causa dei gusti diversificati e dell'ampio contenuto musicale. La sfida ACM RecSys 2018 [6], organizzata da Spotify, ha mirato a migliorare la continuazione automatica delle playlist utilizzando un dataset su larga scala. L'obiettivo era raccomandare fino a 500 brani che corrispondessero alle caratteristiche di una playlist data.

1. Introduzione

Negli ultimi anni, insieme allo sviluppo tecnologico, vi è stata anche una grande evoluzione dell'industria musicale: dall'acquisto di vinili e cd si è passati all'utilizzo di piattaforme di streaming musicale, quali Spotify, Apple Music o Youtube, che permettono dei servizi molto più comodi a coloro che ascoltano musica. Mentre in precedenza i sistemi di raccomandazione si concentravano sulla presentazione di contenuti che potresti voler acquistare per un consumo successivo, le moderne piattaforme di streaming devono invece concentrarsi sulla raccomandazione di contenuti che puoi e vorrai goderti in quel preciso momento. In particolare, le playlist sono diventate una parte significativa della nostra esperienza di ascolto e proprio per questo consigliare playlist è fondamentale per i servizi musicali odierni. A questo scopo, si è cercato il più possibile di sviluppare modelli di streaming in cui l'attenzione è ora rivolta maggiormente alla generazione di sequenze di brani simili che vanno bene insieme. Il progetto qui presentato consiste proprio in un sistema di Raccomandazione Musicale per aggiungere canzoni idonee ad una playlist Spotify cercando di prevedere la probabilità che un utente apprezzi una canzone partendo da dati riguardanti le playlist. Si vuole quindi generare, a partire da una playlist creata da un utente, una lista di 500 canzoni raccomandate, ordinate per rilevanza in ordine decrescente. Difatti per tale progetto è stata presa ispirazione dalla *ACM RecSys 2018 Challenge* organizzata da Spotify, ha mirato a migliorare la continuazione automatica delle playlist utilizzando un dataset su larga scala disponibile su *Kaggle* [7], dove l'obiettivo era proprio raccomandare fino a 500 brani che corrispondessero alle caratteristiche di una playlist data.

Table 1. Statistiche del Dataset

Proprietà	Valore
Numero di playlist	1,000,000
Numero di tracce	66,346,428
Numero di tracce uniche	2,262,292
Numero di album unici	734,684
Numero di artisti unici	295,860
Numero di nomi di playlist unici	92,944
Numero di nomi di playlist normalizzati unici	17,381
Durata media di una playlist (tracce)	66.35

2. Analisi del Dataset

Ogni playlist all'interno del **Million Playlist Dataset (MPD)** contiene un titolo, una lista di brani (compresi gli ID dei brani e i metadati associati) e altri campi di metadati, come l'ultima data di modifica, il numero di modifiche alla playlist e altro ancora. Tutti i dati sono anonimizzati per proteggere la privacy degli utenti. Le playlist sono campionate con un certo grado di randomizzazione, filtrate manualmente per garantire la qualità e per rimuovere contenuti offensivi, e contengono alcune alterazioni e tracce fittizie. Delle statistiche base del dataset sono riportate nella tabella 1 Di conseguenza, il dataset non rappresenta in maniera fedele la reale distribuzione delle playlist presenti sulla piattaforma Spotify e non deve essere interpretato come tale in alcun tipo di ricerca o analisi eseguita sul dataset. Ecco un esempio di una tipica playlist presente nel dataset in formato JSON.

```
{
  "name": "musical",
  "collaborative": "false",
  "pid": 5,
  "modified_at": 1493424000,
  "num_albums": 7,
  "num_tracks": 12,
  "num_followers": 1,
  "num_edits": 2,
  "duration_ms": 2657366,
  "num_artists": 6,
  "tracks": [
  {
    "pos": 0,
    "artist_name": "Degiheugi",
    "track_uri": "spotify:track:7
    vqa3sDmtEaVJ2gcvxtRID",
    "artist_uri": "spotify:artist:3
    V2paBXEoZIAhfZRJmo2jL",
    "track_name": "Finalement",
    "album_uri": "spotify:album:2
    KrRMJ9z7Xjoz1Az4O6UML",
    "duration_ms": 166264,
    "album_name": "Dancing Chords and Fireflies"
  },
  {
    "pos": 1,
    "artist_name": "Degiheugi",
    "track_uri": "spotify:track:23
    EOmJivOZ88WJPUbIPjh6",
    "artist_uri": "spotify:artist:3
    V2paBXEoZIAhfZRJmo2jL",
    "track_name": "Betty",
    "album_uri": "spotify:album:3
    lUSlvjUoHNA8IkNTqURqd",
    "duration_ms": 235534,
    "album_name": "Endless Smile"
  },],
}
```

Code 1. Singolo file in MPD.

Ogni file contiene informazioni annidate relative alle tracce presenti nelle playlist, oltre a dati associati alle playlist stesse. L'intero dataset è suddiviso in 1000 file (o *slice*), ognuno dei quali contiene 1000 playlist. Dopo una prima fase di preprocessamento, i file `.json` sono stati suddivisi in quattro diverse categorie. `.csv`:

- `artists.csv`: `artist_id, artist_uri, artist_name`
- `playlists.csv`: `playlist_id, name, num_artists, num_albums, num_followers, num_edits, durations_ms, modified_at, collaborative, description`

	name	collaborative	pid	modified_at	num_tracks	num_albums	num_followers	tracks	num_edits	duration_ms	num_artists	description
0	disney	false	1000	1457827200	189	16	1	[{'pos': 0, 'artist_name': 'Original Broadway ...	4	31428282	65	NaN
1	Indie Electro	false	1001	1417824000	165	18	2	[{'pos': 0, 'artist_name': 'The Octopus Projec...	2	38241566	8	NaN
2	jack & jack	false	1002	1465430400	17	14	1	[{'pos': 0, 'artist_name': 'Jack & Jack', 'tra...	3	3549358	3	NaN
3	vibes	false	1003	1498435200	225	195	2	[{'pos': 0, 'artist_name': 'LANY', 'track_uri'...	91	51242585	157	NaN
4	Indie	false	1004	1498608000	165	118	1	[{'pos': 0, 'artist_name': 'Youth Lagoon', 'tr...	74	42601098	92	NaN

(a) Playlist level

	pos	artist_name	track_uri	artist_uri	track_name	album_uri	duration_ms	album_name	playlist_name	pid
0	0	Original Broadway Cast - The Little Mermaid	spotify:track:5IbCV9Icebx8rR6wAp5hhP	spotify:artist:3TymzPhJTMyupk7P5xkahM	Fathoms Below - Broadway Cast Recording	spotify:album:3ULJeOMgroG27dpn27MDfS	154506	The Little Mermaid: Original Broadway Cast Rec...	disney	1000
1	1	Original Broadway Cast - The Little Mermaid	spotify:track:6rKVAvjHcxAzZ1BHtwh5yC	spotify:artist:3TymzPhJTMyupk7P5xkahM	Daughters Of Triton - Broadway Cast Recording	spotify:album:3ULJeOMgroG27dpn27MDfS	79066	The Little Mermaid: Original Broadway Cast Rec...	disney	1000
2	2	Original Broadway Cast - The Little Mermaid	spotify:track:6Jlkb1Wh08RYHstWScsTvg	spotify:artist:3TymzPhJTMyupk7P5xkahM	The World Above - Broadway Cast Recording	spotify:album:3ULJeOMgroG27dpn27MDfS	94600	The Little Mermaid: Original Broadway Cast Rec...	disney	1000
3	3	Original Broadway Cast - The Little Mermaid	spotify:track:0XhC8bfStML9ygBmfOt1JJ	spotify:artist:3TymzPhJTMyupk7P5xkahM	Human Stuff - Broadway Cast Recording	spotify:album:3ULJeOMgroG27dpn27MDfS	151480	The Little Mermaid: Original Broadway Cast Rec...	disney	1000
4	4	Original Broadway Cast - The Little Mermaid	spotify:track:0ABxAcsRWlqckkyONsfP67	spotify:artist:3TymzPhJTMyupk7P5xkahM	I Want the Good Times Back - Broadway Cast Rec...	spotify:album:3ULJeOMgroG27dpn27MDfS	297920	The Little Mermaid: Original Broadway Cast Rec...	disney	1000

(b) Track level

- `tracks.csv`: `track_id, track_uri, track_name, artist_id, album_uri, durations_ms_track, album_name`
- `playlist_tracks.csv`: `playlist_id, track_id, artist_id, pos`

ognuno dei quali rappresenta le informazioni in modo strutturato e relazionale suggerito dal loro nome, anche se alla fine sono stati di interesse solo le informazioni relative all'appartenenza di una singola traccia in una determinata playlist, descritta nel file `playlist_tracks.csv`.

Sinteticamente possiamo riassumere le informazioni presenti nel dataset in due livelli di dettaglio, pari al grado di nidificazione dei dati in esso.

- **Playlist level**: Tutte le informazioni globali delle playlist. Figura 1a
- **Track level**: Tutte le informazioni relative alle tracce. Figura 1b

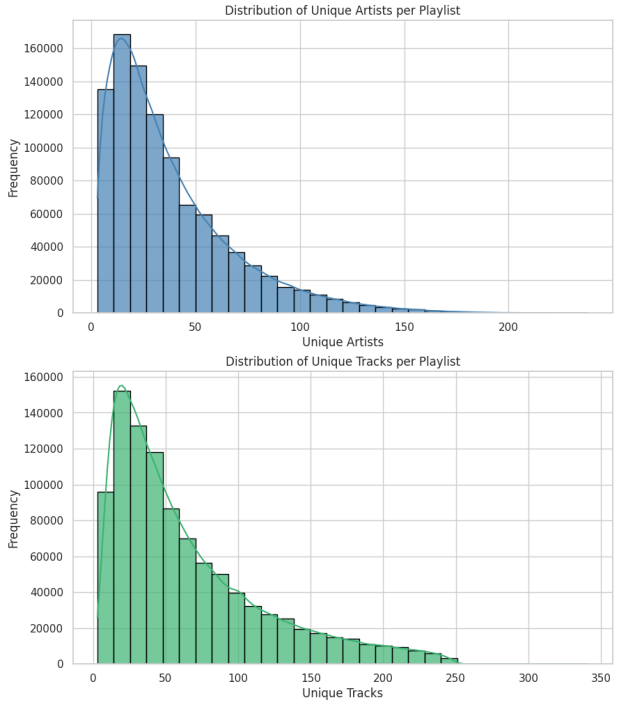

Figure 2. Distribuzioni del numero di artisti e di tracce uniche per playlist

Dall'analisi preliminare dei dati riportati in Figura 2, emerge che le distribuzioni di tracce e artisti all'interno delle playlist suggeriscono una correlazione tra la frequenza delle playlist e la diversità degli artisti e delle tracce. In particolare, le playlist più comuni tendono a presentare un elevato livello di eterogeneità sia in termini di artisti che di tracce, indicando una varietà significativa nel contenuto musicale.

Nel dominio delle raccomandazioni musicali, le playlist create dagli utenti spesso presentano una struttura comune influenzata dal modo in cui vengono compilate. Gli utenti tendono a raggruppare i brani provenienti dallo stesso album, a creare playlist monogenere arricchite nel tempo con nuove canzoni dello stesso artista, e a includere collaborazioni di un unico artista in modo sequenziale. Inoltre, è frequente trovare playlist in cui inizialmente sono presenti artisti diversi, seguiti da brani dello stesso artista riaggiunti in posizioni

successive, come illustrato nella Figura 3.

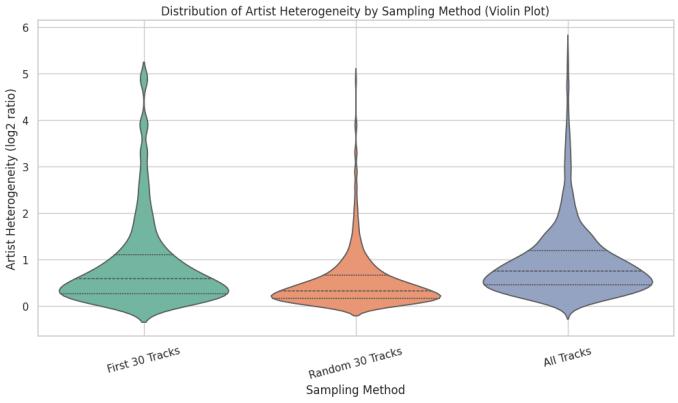

Figure 3. Varietà di artisti per 5000 playlist campionata in maniera casuale calcolata con tre diversi metodi di campionamento per le tracce al loro interno

Per sfruttare queste modalità, definiamo una nuova misura per stimare la diversità degli artisti presenti nelle playlist 1.

$$ArtistHeterogeneity_p = \log_2\left(\frac{|uniqueTrack_p|}{|uniqueArtists_p|}\right) \quad (1)$$

Dove p rappresenta la playlist. Un valore di $ArtistHeterogeneity = 0$ indica una playlist con un'ampia varietà di artisti, simile a quelle contenenti i 100 brani più popolari di un determinato genere. Al contrario, un valore elevato di $ArtistHeterogeneity$ indica una playlist molto ripetitiva [5].

2.1. Preprocessamento

Dopo la lettura dei dati, questi vengono preprocessati per ottimizzare le successive operazioni computazionali. Le operazioni eseguite comprendono la gestione dei valori mancanti, attuata mediante l'utilizzo dell'API di Spotify per il recupero delle informazioni ove possibile; la rimozione dei duplicati; la normalizzazione dei testi, che migliora la qualità delle informazioni testuali; e il dowcasting dei tipi di dati, finalizzato a ottimizzare l'utilizzo delle risorse di sistema. Infine, si procede al merging dei file CSV precedentemente analizzati, selezionando esclusivamente le colonne ritenute significative per l'addestramento del modello. Questo processo culmina nella creazione di un *DataFrame* finale, denominato `dataframe.feather`, che rappresenta un formato ad alte prestazioni e consente una rapida lettura e scrittura del dataset sul disco. La fase conclusiva prevede la separazione del *training set* e del *test set*, con quest'ultimo composto esclusivamente da 1000 playlist. Poiché l'apprendimento non è supervisionato, la metodologia di separazione tra i due set non riveste un'importanza fondamentale.

3. Il Modello

L'idea di ricondurre il problema di raccomandazione di canzoni a un problema di NLP è suggerita dalla natura sequenziale e testuale delle playlist, dalle tecniche di embedding applicabili, e dalla possibilità di utilizzare metodi di analisi del linguaggio per migliorare la comprensione e la raccomandazione dei brani.

Nell'ambito dell'elaborazione del linguaggio naturale, i modelli **Word2Vec** e **Doc2Vec** si basano su tecniche di apprendimento non supervisionato per catturare le relazioni semantiche tra le parole e il contesto in cui appaiono, consentendo di trasformare il linguaggio naturale in una forma numerica utilizzabile per varie applicazioni, tra cui il recupero delle informazioni, la classificazione dei testi e i sistemi di raccomandazione.

Word2Vec è un modello che genera vettori per singole parole, mantenendo la loro relazione semantica. Attraverso due architetture principali: **Continuous Bag of Words (CBOW)** e **Skip-gram** riesce a prevedere una parola in base al contesto o viceversa, producendo vettori che riflettono le similitudini semantiche tra le parole. Ad esempio, parole con significati simili tendono ad avere rappresentazioni vettoriali vicine nello spazio di rappresentazione.

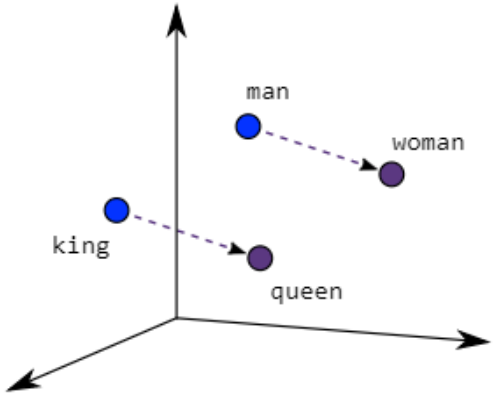

Figure 4. Le parole con significato simile si trovano più vicine nello spazio, indicando la loro somiglianza semantica.

Doc2Vec, d'altra parte, estende il concetto di Word2Vec per gestire interi documenti. Introducendo un vettore di documento unico che viene appreso insieme ai vettori delle parole, Doc2Vec consente di catturare le informazioni contestuali e tematiche su un livello più ampio. Questa rappresentazione è particolarmente utile quando si desidera analizzare testi complessi o lunghi, come articoli, recensioni o intere playlist musicali.

Un'idea promettente era quella di combinare questi due modelli attraverso un approccio di *ensemble learning*. Questo approccio sfrutta la forza di entrambi i modelli: Word2Vec per ottenere rappresentazioni dettagliate e precise delle singole parole e Doc2Vec per fornire un contesto più ampio e significativo a livello di documento. Integrando le informazioni provenienti da entrambe le fonti, possiamo migliorare le prestazioni complessive del sistema, ottenendo una rappresentazione più robusta e informativa dei dati. L'ensemble learning consente, infatti, di mitigare le limitazioni di ciascun modello, sfruttando i punti di forza di entrambi per raggiungere risultati più accurati e affidabili nelle applicazioni di raccomandazione e analisi testuale.

Purtroppo durante lo svolgimento del progetto abbiamo deciso di ridurre la complessità del modello al solo uso di Word2Vec.

3.1. Word2Vec

Word2Vec, sviluppata da Tomas Mikolov nel 2013 presso Google. E' una delle tecniche più comuni per eseguire incorporamenti di parole in diversi casi di elaborazione del linguaggio naturale (NLP) utilizzando reti neurali superficiali. Tale metodo di incorporamenti di parole può essere generalizzato ad altri incorporamenti di elementi, che associano qualsiasi prodotto su un sito di e-commerce, qualsiasi video su Youtube, film su Netflix con un vettore. Naturalmente, in questo caso, anche le canzoni possono essere un vettore.

Come già menzionato, Word2Vec è una delle tecniche più diffuse per apprendere le rappresentazioni vettoriali delle parole, utilizzando una rete neurale superficiale. Una rete neurale, come altri algoritmi di apprendimento supervisionato, richiede dati etichettati per essere addestrata. Tuttavia, come possiamo addestrare una rete neurale se i dati si presentano sotto forma di sequenze di parole (ossia parole) o sequenze di brani (ossia playlist) senza alcun obiettivo o etichetta di dati?

Per affrontare questa sfida, si crea una cosiddetta "task fittizia". In questo contesto, non ci interessano tanto gli input e gli output della

rete, quanto piuttosto l'addestramento dei pesi tra il livello di input e il livello nascosto, che vengono successivamente estratti come vettori. In termini semplici, l'obiettivo delle rappresentazioni vettoriali può essere classificato come apprendimento non supervisionato, ma il processo di ottenimento delle rappresentazioni in Word2Vec viene realizzato attraverso l'apprendimento supervisionato mediante una rete neurale.

Di seguito viene illustrata la generale architettura di Word2Vec:

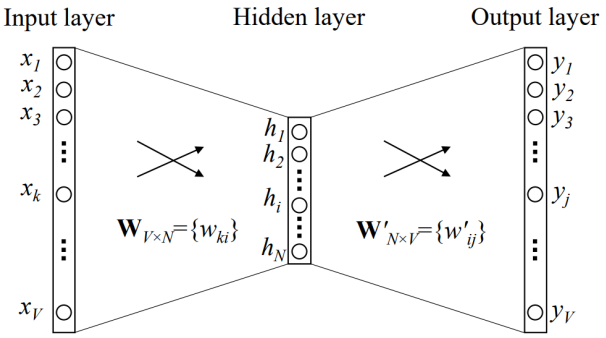

Figure 5. Un semplice modello W2V con architettura CBOW con una sola parola nel contesto [3]

- Lo strato di input è un vettore codificato tramite la tecnica *One-Hot* di dimensione V (dimensione del vocabolario).
- $W_{V\times N}$ è la matrice dei pesi che proietta l'input x allo strato nascosto. **Questi valori sono gli embedded vectors**.
- Lo strato nascosto contiene N neuroni, il suo scopo è di copiare la somma pesata degli input al prossimo strato senza nessuna funzione d'attivazione.
- $W'_{N\times V}$ è la matrice dei pesi che mappa l'output dello strato nascosto allo strato finale di output.
- Lo strato di output è ancora un vettore lungo V, con una funzione d'attivazione di tipo *softmax*.

Precedentemente abbiamo detto che Word2Vec può utilizzare due tipi di approcci mantenendo praticamente la stessa architettura:

- Skip-gram: Il task fittizio è di predirre il contesto data una parola target.
- Continuos Bag-Of-Words (CBOW): Il task fittizio è di predirre una parola target dato il contesto.

Nel corso di questo lavoro abbiamo adottato l'approccio CBOW che molto di più rappresenta il task di predirre la canzone corretta dato il contesto, cioè le altre tracce presenti nella playlist come mostrato in Figura 6.

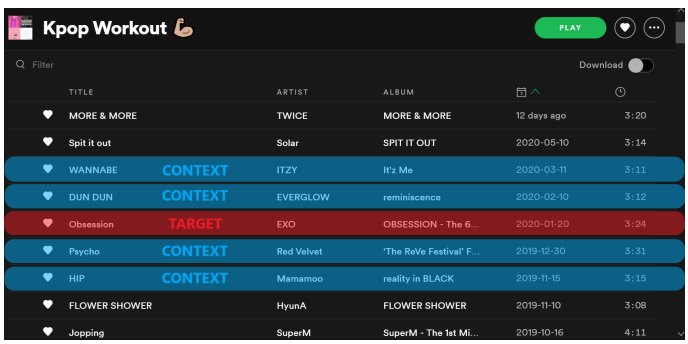

Figure 6. Applicazione dell'approccio CBOW su un playlist

3.2. Training

L'addestramento del modello è dunque effettuato su un training set contenente 99000 playlist per un totale di 6553507 tracce (anche ripetute). Usando `gensim`, possiamo suddividere la fase di addestramento in tre step principali:

In primo luogo creiamo un'istanza di `Word2Vec` e impostiamo gli iperparametri del modello:

- `vector_size = 300`: Ogni brano sarà rappresentato da un vettore di 300 dimensioni.
- `window=5`: La distanza massima in cui consideriamo il contesto di una canzone
- `min_count = 5`: Vengono selezionate solo le parole (canzoni) che appaiono almeno 5 volte, per ridurre il rumore e outliers dai dati
- `workers=multiprocessing.cpu_count()`: Utilizza tutti i processori disponibili per velocizzare l'addestramento.
- `epochs=400`: Il numero di epoche (cicli completi di addestramento) da eseguire.
- `sg=0`: Utilizza l'algoritmo CBOW (Continuous Bag of Words) per addestrare il modello.
- `negative=5`: Usa campioni negativi per migliorare la qualità dei vettori.
- `alpha=0.1`: Tasso di apprendimento iniziale.

A causa dell'inefficienza computazionale della funzione *softmax* si utilizza un'alternativa chiamata negative sampling con funzione sigmoide, che riformula il problema come un insieme di task indipendenti di classificazione logistica binaria con complessità algoritmica pari a $O(K+1)$, dove K rappresenta il numero di campioni negativi e 1 è il campione positivo. Mikolov suggerisce di utilizzare K nell'intervallo $[5, 20]$ per vocabolari piccoli e $[2, 5]$ per vocabolari più ampi.

Il ***negative sampling*** viene dunque usato per aumentare l'efficienza, selezionando un piccolo numero di parole identificate come negative (o non target) per ogni coppia positiva contesto-target, aggiornando solo i pesi di queste parole campionate. Questo accelera il processo di addestramento e riduce il carico computazionale.

Per assicurarci che l'addestramento del modello procedesse correttamente abbiamo graficato l'andamento della loss function in Figura 7. Abbiamo notato che l'algoritmo di training converge molto velocemente per $\alpha = 0.1$, portando la loss a uno score sull'ordine delle centinaia.

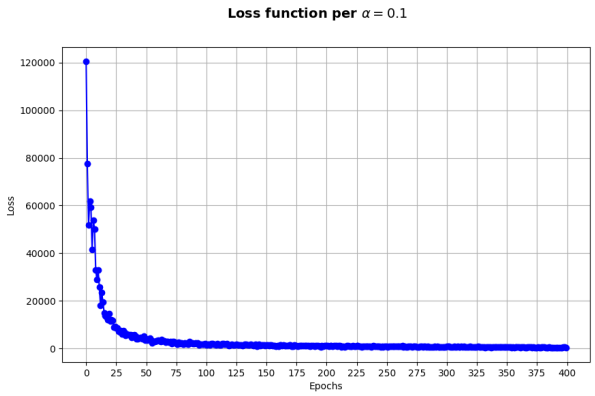

Figure 7. Plot dell'andamento della funzione di loss

Altri addestramenti con dataset maggiormente o minormente campionati a 50,000 playlists o 500,000 playlists presentano più o meno lo stesso andamento della loss function.

3.3. Visualizzazione dei Vettori

Le rappresentazioni vettoriali dei brani musicali possono essere visualizzate tramite un gradiente cromatico. Dato che il modello è stato addestrato su 300 dimensioni, ogni brano viene descritto da 300 barre colorate, che corrispondono ai valori dei singoli elementi del

vettore. La somiglianza tra i brani è calcolata utilizzando la **similarità coseno**, una metrica comunemente impiegata per confrontare vettori nello spazio multidimensionale.

$$simalirity(A, B) = \cos\theta = \frac{A \cdot B}{||A|| \cdot ||B||} \quad (2)$$

Matematicamente, la similarità coseno misura il coseno dell'angolo tra due vettori A e B, proiettati in uno spazio multidimensionale. I vettori che rappresentano brani musicali con contesti simili tendono ad occupare posizioni spaziali ravvicinate; in questo caso, il coseno tra tali vettori sarà prossimo a 1, indicando che l'angolo tra essi è vicino a 0. Più piccolo è l'angolo, maggiore sarà la similarità coseno, riflettendo una forte affinità tra i brani.

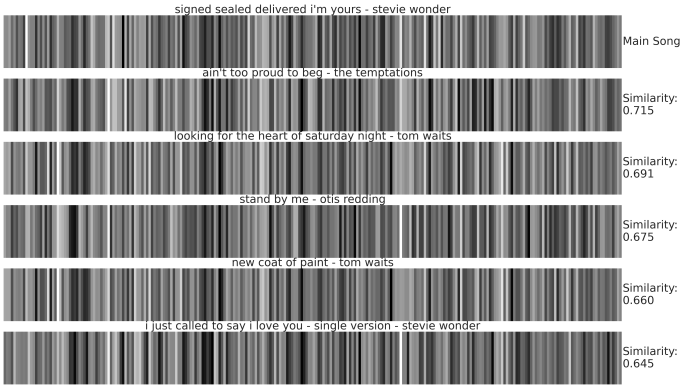

Figure 8. le 5 canzoni più simili a Signed Sealed Delivered I'm Yours di Stevie Wonder.

4. Clustering

Il clustering, applicato su questi embedding, consente di raggruppare brani con caratteristiche comuni in insiemi distinti, riducendo la complessità del problema di raccomandazione. Questa tecnica permette di segmentare l'intero spazio vettoriale in regioni coerenti, in cui le canzoni appartenenti allo stesso cluster condividono simili caratteristiche musicali, come genere, stile o struttura. Per effettuarlo possiamo usare l'algoritmo di clustering **K-Means** un tipo di algoritmo di apprendimento non supervisionato che suddivide un dataset in gruppi distinti (o cluster) in base alle somiglianze tra i punti dati. Ogni gruppo o cluster è rappresentato dal suo centroide, che costituisce essenzialmente il "centro" di quel gruppo. L'obiettivo del K-Means è minimizzare la varianza all'interno di ciascun cluster e massimizzare la varianza tra i diversi cluster. Però dobbiamo ricordare che la similarità tra vettori è calcolata mediante la **similarità coseno** e non la **distanza euclidea**.

Pertanto, dovrebbe essere preso in considerazione l'algoritmo K-Means con la distanza coseno, spesso denominato **Spherical K-Means Clustering** [2]. L'idea è di identificare il centroide in modo tale da uniformare e minimizzare l'angolo tra ciascun vettore all'interno di un cluster. L'intuizione è simile all'osservazione di un ammasso di stelle, dove ogni punto dovrebbe avere una distanza coerente rispetto agli altri. Questa distanza è definita dalla similarità coseno.

4.1. Spherical K-Means

In assenza di un'implementazione diretta di Spherical K-Means, abbiamo optato per un approccio alternativo che normalizza i dati e applica un algoritmo di K-Means classico. Questo metodo si basa sull'idea che **normalizzando** i dati, è possibile ottenere risultati simili a quelli di Spherical K-Means [1].

L'idea di questo approccio si basa sul fatto che la normalizzazione dei dati implica scalare ogni punto in modo che abbia una norma unitaria. Dopo la normalizzazione, tutti i punti si trovano sulla superficie di una sfera unitaria, rendendo così le computazioni delle distanze intrinsecamente angolari. Nel K-Means tradizionale, i centri dei cluster vengono aggiornati in base alla media dei punti dati assegnati a ciascun cluster. Normalizzando i dati, i centri diventano anch'essi normalizzati, avvicinando l'algoritmo K-Means alla minimizzazione della distanza angolare tra i punti e i centri, come avviene in Spherical K-Means.

A questo punto viene eseguito l'algoritmo sui vettori delle canzoni iterando su un numero di cluster da 10 a 500 così da individuare l'ottimale numero di cluster che minimizza la **WCSS (Within Clusters Sum of Squares)** tramite l'*elbow method*.

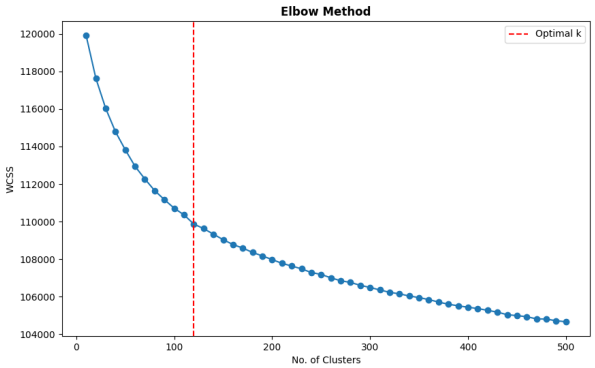

Figure 9. Elbow Method applicato sullo Spherical K-Means.

Per stimare automaticamente il numero ottimale di cluster, abbiamo utilizzato una funzione che traccia una retta tra i punti iniziali e finali del grafico. La funzione calcola quindi la distanza di ciascun punto dalla retta e seleziona infine il punto che massimizza tale distanza come mostrato in Figura 10. Tale numero ottimale di cluster individuato è pari a 120.

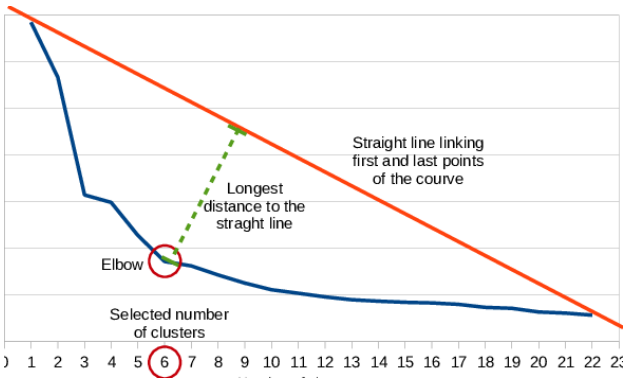

Figure 10. Ricerca del numero ottimale di cluster.[4]

4.2. t-SNE

È sempre utile visualizzare gli embedding creati. In questo contesto, abbiamo a disposizione vettori di canzoni con 300 dimensioni. Tuttavia, questi vettori ad alta dimensione non possono essere visualizzati nel nostro mondo tridimensionale. Pertanto, l'uso di algoritmi di riduzione della dimensionalità, come il **t-Distributed Stochastic Neighbor Embedding (t-SNE)**, ci consente di mappare i vettori in uno spazio di dimensione inferiore. Sebbene non verranno presentati i dettagli matematici del t-SNE, nella pratica questo approccio tende a generare visualizzazioni con cluster distintamente isolati.

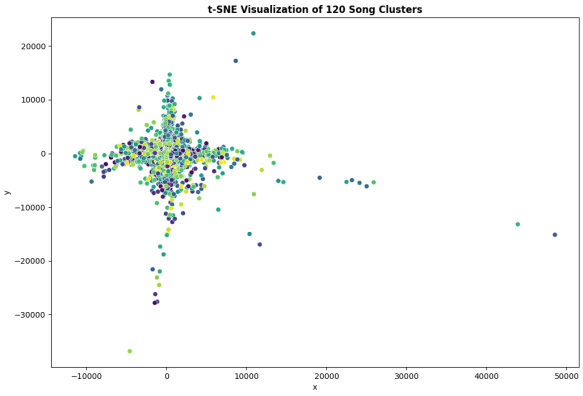

Figure 11. Visualizzazione dei 120 cluster tramite t-SNE

In Figura 11 è stato eseguito t-SNE sui 120 cluster con `perplexity = 40`. La maggior parte dei punti è concentrata verso il centro, con alcuni cluster che si disperdono più sporadicamente in diverse direzioni. Questa concentrazione centrale ci può suggerire che molti degli embedding sono strettamente correlati nello spazio latente.

Notiamo inoltre molti outliers, ovvero canzoni che hanno meno caratteristiche comuni con la maggior parte del dataset, o semplicemente clusters rappresentativi di generi o temi musicali molto di nicchia.

Il grafico dei cluster potrebbe apparire disordinato poiché vengono rappresentati contemporaneamente tutti i 110 cluster. In alternativa, possiamo applicare t-SNE su 10 cluster selezionati casualmente e visualizzare il risultato in Figura 12.

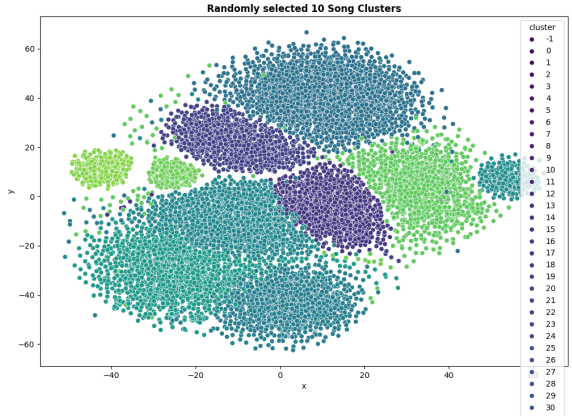

Figure 12. Visualizzazione di 10 cluster individuati casualmente

Come prima possiamo notare che tracce con contesti simili tendono ad essere visualizzate vicine, creando però cluster distinti con maggiore separazione visiva. La riduzione del numero di cluster facilita l'identificazione delle ragioni distinte, rendendo più chiaro il modo in cui i brani si raggruppano in base alle loro caratteristiche.

Sebbene i cluster siano separati visivamente, sono comunque vicini tra loro nella spazio t-SNE. Questo può indicare che i brani dei 10 cluster selezionati condividono ancora una certa somiglianza, anche se appartengono a gruppi distinti. Questo suggerisce che gli embedding, pur rappresentando canzoni diverse, condividono ancora caratteristiche musicali comuni che influenzano la vicinanza nel grafico in due dimensioni.

5. Valutazione del Modello

Per valutare il modello abbiamo bisogno di creare delle raccomandazioni musicali e valutarne l'accuratezza secondo delle determinate metriche.

5.1. Raccomandazione

Per suggerire brani simili basati su una determinata playlist possiamo calcolare il **vettore della playlist** per ciascuna playlist, facendo la media di tutti i vettori delle canzoni presenti in quella playlist. Questi vettori diventano quindi la query per trovare canzoni simili, basandosi sulla similarità coseno.

Per farlo viene utilizzata la funzione `meanVectors` che trasforma appunto ogni playlist presente nel test set in un vettore medio, ottenendo una lista di vettori medi.

```
def meanVectors(playlist):
    vec = []
    for song_id in playlist:
        try:
            vec.append(model.wv[song_id])
        except KeyError:
            continue
    if len(vec) > 0:
        return np.mean(vec, axis=0)
    else:
        return np.zeros(model.vector_size)
```

Code 2. Funzione meanVectors.

A questo punto, viene definita la funzione `similarSongsByVector(...)`, che, partendo dal vettore precedentemente costruito per rappresentare una playlist, raccomanda i brani più simili utilizzando la similarità coseno, una funzionalità nativa di Gensim. La funzione restituisce una lista di tuple, ciascuna contenente il nome del brano e il relativo punteggio di similarità. In questo modo, a partire dall'indice di una playlist, è possibile stampare le prime n canzoni raccomandate per quella playlist.

5.2. Valutazione

Per il processo di valutazione delle performance del sistema di raccomandazione musicale, è stata utilizzata la metrica del **Hit Rate (HR)**. Per ogni canzone nella playlist applichiamo la tecnica del **Leave-One-Out (LOO)**, l'obiettivo principale è misurare l'efficacia del sistema nel predire una canzone rimossa da una playlist. In particolare, ogni brano all'interno di una playlist viene temporaneamente rimosso, e il sistema di raccomandazione viene utilizzato per prevedere quale canzone è stata rimossa. Se la canzone rimossa (LOO) è inclusa tra le prime N canzoni raccomandate, si considera un HIT (successo), altrimenti un $MISS$ (errore). Al termine del processo di LOO su una playlist, calcoliamo HR come:

$$HR = \frac{HITs}{Playlist\ Length} \tag{3}$$

Questo processo viene dunque ripetuto per ogni canzone della playlist, andando a definire cosi un **Average Hit Rate at n (AHR@n)**.

L'ultimo passaggio consiste nella valutazione dei tre diversi sistemi di raccomandazione musicale, confrontando le loro performance in termini di **AHR@25** (Average Hit Rate con top 25 raccomandazioni).

Ecco i tre diversi sistemi messi a confronto:

- **Random Recommender**
- **Cluster-based Recommender**
- **Word2Vec Recommender**

Per implementare questa valutazione, si utilizza un sistema di raccomandazione casuale (Random Recommender) come baseline, ovvero un sistema che raccomanda canzoni selezionate in modo

completamente casuale, con lo scopo di confrontarne le prestazioni con metodi più avanzati.

Il *cluster-based recommender* è un sistema di raccomandazione che utilizza i cluster trovati in precedenza con Spherical K-Means per raggruppare elementi (in questo caso, canzoni) con caratteristiche simili e poi fare raccomandazioni basate su questi gruppi. IN particolare cerca di identificare quale cluster è il più frequente (tramite voto di maggioranza) tra le canzoni circostanti. La distanza massima tra la canzone rimossa (LOO) e le canzoni del contesto è definita da una finestra. Dalla lista delle canzoni appartenenti al cluster di maggioranza, vengono selezionati un insieme di brani in modo casuale.

```
def hitRateClustering(playlist, window, n_songs):
    hit = 0
    context_target_list = []
    for idx, target in enumerate(playlist):
        context = []
        for w in range(idx - window, idx + window + 1):
            if 0 <= w < len(playlist) and w != idx:
                context.append(playlist[w])
        context_target_list.append((context, target))

    for context, target in context_target_list:
        vectors = [model.wv[c].astype('float64') for c in context if c in model.wv.index_to_key]
        if len(vectors) == 0: continue
        km_opt.cluster_centers_ = km_opt.cluster_centers_.astype(float)
        cluster_numbers = km_opt.predict(vectors)
        majority_voting = stats.mode(cluster_numbers).mode[0]
        possible_songs_id = list(songs_cluster[songs_cluster['cluster'] == majority_voting].index)
        recommended_songs = random.sample(possible_songs_id, n_songs)
        songs_id = recommended_songs
        hit += int(target in songs_id)
    return hit/len(playlist)
```

Code 3. Cluster-based recommender

Infine, si procede alla valutazione del modello CBOW Word2Vec nel seguente modo: si calcolano i vettori medi delle canzoni di contesto circostanti utilizzando la funzione precedentemente definita `meanVectors()`, tenendo conto di una distanza massima definita dalla finestra temporale (window). Successivamente, si identificano le canzoni più simili basandosi sulla similarità coseno, avvalendosi della funzione `similarSongsByVector()`. Questo approccio consente di ottenere raccomandazioni musicali pertinenti e basate su un'analisi contestuale.

E' possibile a questo punto riportare in un grafico a barre orizzontali come quello mostrato in figura 13 il confronto tra i vari sistemi, ponendo sull'asse delle ascisse il valore dell'AHR.

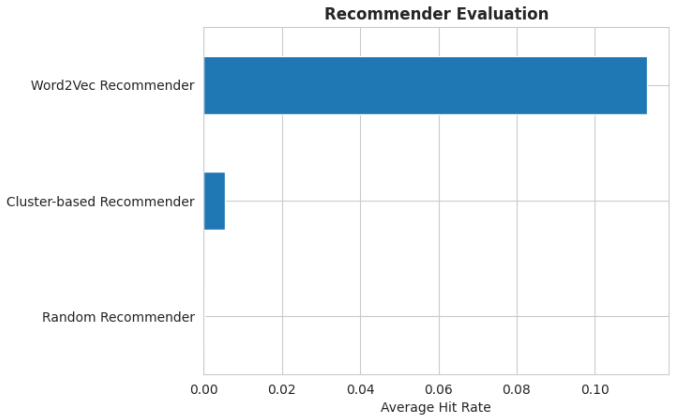

Figure 13. I tre sistemi messi a confronto su AHR@25

Effettivamente otteniamo un $AHR@25 = 0.1132686$ per il nostro modello addestrato, un valore nettamente maggiore agli altri sistemi messi a confronto, ma comunque abbastanza ridotto per un reale sistema di raccomandazione, aprendoci dunque una serie di valutazioni sul possibile sviluppo e miglioramento del modello.

6. Conclusioni e Sviluppi Futuri

In conclusione, il sistema di raccomandazione musicale basato su Word2Vec si è rivelato efficace, ottenendo risultati significativi rispetto ai metodi di raccomandazione casuale e basati su clustering. Tuttavia, l'accuratezza del modello, misurata tramite AHR@25, evidenzia che esistono ancora margini di miglioramento. Tra le possibili aree di sviluppo futuro, si potrebbe valutare l'integrazione di informazioni aggiuntive relative alle playlist, inizialmente scartate per adattarsi al funzionamento intrinseco del modello. Inoltre, il dataset fornisce gli *Spotify Uniform Resource Identifier (URI)*, che potrebbero essere utilizzati per interfacciarsi con le API di Spotify e ottenere metadati aggiuntivi sulle singole canzoni, come il genere musicale, le caratteristiche acustiche delle tracce e altri dati che potrebbero aiutare a individuare caratteristiche o pattern comuni tra le tracce. L'adozione di tecniche di *ensemble learning* con Doc2Vec, insieme alla valutazione di modelli alternativi come il *Collaborative Filtering* o il *Content-Based Filtering*, entrambi largamente impiegati nei sistemi di raccomandazione, potrebbe migliorare ulteriormente la precisione delle raccomandazioni. Infine, ulteriori analisi dei dati tramite tecniche avanzate di riduzione della dimensionalità e un incremento del dataset di addestramento potrebbero contribuire a potenziare le prestazioni complessive del sistema.

References

[1] S. Zhong, "Efficient online spherical k-means clustering", in *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, vol. 5, 2005, 3180–3185 vol. 5. DOI: 10.1109/IJCNN.2005.1556436.

[2] K. Hornik, I. Feinerer, M. Kober, and C. Buchta, "Spherical k-means clustering", *Journal of Statistical Software*, vol. 50, no. 10, pp. 1–22, 2012. DOI: 10.18637/jss.v050.i10. [Online]. Available: https://www.jstatsoft.org/index.php/jss/article/view/v050i10.

[3] X. Rong, "Word2vec parameter learning explained", Nov. 2014. [Online]. Available: https://doi.org/10.48550/arXiv.1411.2738.

[4] H. Delgado, X. Anguera, C. Fredouille, and J. Serrano, "Fast single- and cross-show speaker diarization using binary key speaker modeling", *IEEE Transactions on Audio Speech and Language Processing*, vol. 23, pp. 2286–2297, Dec. 2015. DOI: 10.1109/TASLP.2015.2479043.

[5] S. Antenucci, S. Boglio, E. Chioso, *et al.*, "Artist-driven layering and user's behaviour impact on recommendations in a playlist continuation scenario", in *Proceedings of the ACM Recommender Systems Challenge 2018*, ser. RecSys Challenge '18, Vancouver, BC, Canada: Association for Computing Machinery, 2018, ISBN: 9781450365864. DOI: 10.1145/3267471.3267475. [Online]. Available: https://doi.org/10.1145/3267471.3267475.

[6] C.-W. Chen, P. Lamere, M. Schedl, and H. Zamani, "Recsys challenge 2018: Automatic music playlist continuation", in *Proceedings of the 12th ACM Conference on Recommender Systems*, ser. RecSys '18, Vancouver, British Columbia, Canada: Association for Computing Machinery, 2018, pp. 527–528, ISBN: 9781450359016. DOI: 10.1145/3240323.3240342. [Online]. Available: https://doi.org/10.1145/3240323.3240342.

[7] Kaggle, *Spotify milion playlist dataset*, https://www.kaggle.com/datasets/himanshuwagh/spotify-million.