Università degli Studi di Torino
Dipartimento di Informatica
Corso di Laurea in Informatica

Tesi di Laurea Triennale

Raccomandazione di contenuti musicali: un sistema intelligente basato sulla combinazione di concetti

RELATORE
Prof. Gian Luca Pozzato

CORRELATORE

CANDIDATO
Alberto Marocco
947841

Anno Accademico 2024/2025

Dichiarazione di Originalità

Dichiaro di essere responsabile del contenuto dell'elaborato che presento al fine del conseguimento del titolo, di non avere plagiato in tutto o in parte il lavoro prodotto da altri e di aver citato le fonti originali in modo congruente alle normative vigenti in materia di plagio e di diritto d'autore. Sono inoltre consapevole che nel caso la mia dichiarazione risultasse mendace, potrei incorrere nelle sanzioni previste dalla legge e la mia ammissione alla prova finale potrebbe essere negata.

Abstract

Questo lavoro presenta un sistema intelligente di raccomandazione musicale basato sulla combinazione di concetti. Il sistema utilizza testi e caratteristiche stilistiche dei brani, acquisiti e arricchiti tramite un crawler automatico di Genius, per costruire prototipi di genere e ibridi cross-genere. La pipeline implementata comprende moduli di analisi delle ripetizioni, generazione di prototipi concettuali e un classificatore che sfrutta "anchors" e soglie adattive per selezionare i contenuti più rilevanti. L'approccio proposto coniuga trasparenza e interpretabilità, fornendo raccomandazioni spiegabili e adattabili a diversi scenari musicali.

Indice

1 Introduzione

Questa tesi propone DEGARI-Music, un sistema di raccomandazione musicale spiegabile basato su prototipi di genere e combinazione di concetti. L'obiettivo è produrre suggerimenti trasparenti, con spiegazioni legate a proprietà tipiche e rigide dei generi (e loro ibridi).

1.a Contesto e motivazioni

Le piattaforme raccomandano bene ma spiegano poco. La trasparenza incide su fiducia, controllo e scoperta consapevole.

1.b Problema e idea

Problema: raccomandazioni "black-box". Idea: prototipi + combinazione (head/modifier) + soglie/anchors → spiegazioni white-box.

1.c Contributi

- Implementazione end-to-end (crawler → prototipi → combinazioni → classificatore).
- Spiegazioni leggibili basate su match proprietà↔brano.
- Valutazione: riclassificazione/copertura, ablation, esempi qualitativi.

1.d Risultati in sintesi

Breve teaser dei risultati più significativi.

1.e Organizzazione del manoscritto

Breve guida ai capitoli 4–13.

2 Fondamenti teorici: tipicalità e combinazione di concetti (TCL)

2.a Tipicità e chiusura razionale

Le Logiche Descrittive (DL) forniscono un formalismo per rappresentare concetti, ruoli e individui. Nelle DL classiche (ad es. ALC) il ragionamento è monotòno: aggiungere nuova informazione non invalida ciò che è già derivabile. Questo è inadeguato per conoscenza "di buon senso" ricca di eccezioni.

Per modellare regolarità con eccezioni si introduce l'operatore di tipicità $T(c\cdot)$: un'assioma del tipo $T(C) \sqsubseteq D$ indica che "tipicamente i C sono D". Le inclusioni **rigide** conservano la forma classica $C \sqsubseteq D$.

La **rational closure** estende alle DL la chiusura razionale: i concetti vengono ordinati per grado di eccezionalità e si adottano **modelli minimi** che minimizzano i ranghi di anomalia. In questo modo le inferenze su $T(c\cdot)$ sono conservative, ma consentono di sospendere premesse tipiche quando emergono informazioni più specifiche (ereditarietà difettibile).

2.a.i Notazione di base

- Inclusioni rigide: $C \sqsubseteq D$.
- Inclusioni tipiche: $T(C) \sqsubseteq D$.
- Preferenza semantica: si privilegiano interpretazioni che minimizzano l'eccezionalità (modelli minimi) e rispettano la gerarchia di specificità.

2.b TCL: Typicality-based Compositional Logic

TCL combina tipicità, probabilità e combinazione concettuale in stile cognitivo. Le proprietà tipiche sono annotate con un grado:

$$p :: T(C) \sqsubseteq D,, p \in (0.5, 1] \tag{1}$$

che si legge: "con grado p, i C tipici sono D".

Dato un **HEAD** C_H e un **MODIFIER** C_M, la combinazione produce un concetto composto che non è la semplice intersezione, ma il risultato di una selezione coerente delle proprietà:

1. **Generazione di scenari**: si costruiscono insiemi compatibili di inclusioni tipiche (rispettando tutte le rigide).
2. **Valutazione**: ad ogni scenario si associa un punteggio/probabilità ottenuto combinando le annotazioni delle tipicità considerate attive (ipotesi d'indipendenza sui pesi tipici).
3. **Selezione**: si scelgono gli scenari ammessi e meglio valutati; le proprietà del risultato sono quelle presenti nello/gli scenario/i selezionato/i.

2.b.I Ruolo HEAD / MODIFIER

- L'HEAD fornisce la struttura concettuale di base; in caso di conflitti tra tipicità, hanno priorità le rigide e, tra le tipiche, si privilegiano scelte coerenti con la "fisionomia" dell'HEAD.
- Il MODIFIER introduce restrizioni/aggiustamenti che possono soppiantare tratti non essenziali dell'HEAD se questo aumenta la coerenza globale dello scenario.

2.b.II Perché serve TCL (il "pet–fish")

La combinazione concettuale umana non è additiva: esistono congiunzioni in cui la tipicità dell'insieme composto supera quella dei componenti (es. **pet fish**). Approcci puramente fuzzy o intersezioni rigide non catturano questi effetti; l'uso di tipicità difettibili, modelli minimi e scenari pesati consente invece di selezionare combinazioni plausibili senza contraddizioni.

2.c Semantica probabilistica delle tipicità

Le annotazioni p sulle inclusioni tipiche possono essere interpretate in modo "distribuzionale": uno scenario eredita un punteggio combinando (sotto ipotesi d'indipendenza) i pesi delle tipicità attive e penalizzando quelle escluse o in conflitto. Questa lettura consente di ordinare gli scenari e di spiegare perché alcune combinazioni risultano più plausibili di altre dal punto di vista qualitativo e quantitativo.

2.d Il sistema CoCoS (cenni)

CoCoS implementa il calcolo di scenari di TCL. Dati HEAD e MODIFIER con le rispettive proprietà rigide e tipiche, costruisce gli scenari ammissibili, ne calcola il punteggio e restituisce il/i prototipo/i del concetto composto come mappa proprietà $\rightarrow$ grado, insieme alle informazioni sullo scenario selezionato. (Questa sezione introduce solo l'idea operativa; i dettagli implementativi saranno trattati nei capitoli successivi.)

3 Probabilità e strumenti per la combinazione concettuale (TCL e CoCoS)

In questo capitolo approfondiamo (i) la componente probabilistica di TCL e (ii) il tool CoCoS che implementa la combinazione concettuale basata su tipicità, presentando anche sistemi affini riutilizzatori di CoCoS.

3.a Inclusioni probabilistiche e scenari in TCL

In TCL le proprietà tipiche sono annotate con un grado di credenza:

$$p :: T(C) \sqsubseteq D, \quad p \in (0.5, 1] \tag{2}$$

. L'idea è che le tipicità valgano "per i C normali", ma possano essere sospese di fronte a informazione più specifica. La semantica probabilistica (stile DISPONTE) assume indipendenza fra tipicità e definisce una distribuzione sugli **scenari**, cioè sulle scelte di quali inclusioni tipiche sono attive.

Definizione operativa. Data una base con tipicità $p_i :: T(C_i) \sqsubseteq D_i$:

- uno **scenario** è un sottoinsieme consistente delle tipicità;
- allo scenario si associa un punteggio/probabilità ottenuto combinando i p_i (attive) e $(1 - p_i)$ (escluse);
- si selezionano gli scenari **ammessi** dalle rigide e **coerenti** con l'euristica HEAD/MODIFIER; il/i prototipo/i del concetto composto eredita/no le proprietà presenti nello/gli scenario/i selezionato/i.

Questa lettura si integra con la **rational closure** di ALC+TR: gli scenari sono valutati solo sui modelli che minimizzano l'eccezionalità, preservando specificità e irrilevanza.

3.b CoCoS: input, algoritmo, output

Scopo. CoCoS implementa TCL: dato un concetto HEAD C_H e un MODIFIER C_M, costruisce gli scenari ammissibili e restituisce il prototipo del concetto composto $C_H l \wedge C_M$.

Input.

- **Proprietà rigide** di C_H e C_M (vincoli non derogabili).
- **Proprietà tipiche** con grado p per ciascun concetto.

- (Opzionale) limiti al numero massimo di proprietà da ereditare.

Algoritmo (sketch).

1. **Genera scenari**: combina le tipicità in insiemi consistenti con le rigide.
2. **Valuta**: assegna a ogni scenario un punteggio/probabilità (ipotesi di indipendenza).
3. **Filtra HEAD/MODIFIER**: scarta scenari incompatibili con la dominanza semantica dell'HEAD.
4. **Seleziona**: sceglie lo/gli scenario/i migliore/i; produce una mappa **proprietà → grado** per il prototipo composito e annota la probabilità/score di scenario.

Output. Un file **Result** con la mappa delle proprietà ereditate (e.g., `{rock: 0.95, high_repetition: 0.8, ...}`) e le info di scenario; opzionale **Scenario** con la maschera delle tipicità attive.

3.c Sistemi affini e riuso di CoCoS

La stessa combinazione basata su TCL è stata riusata in diversi prototipi per creatività computazionale e XAI (ad es. supporto alla generazione di personaggi/ruoli, raccomandazione culturale). Esempi recenti includono estensioni di DEGARI e strumenti come **EDIFICA** e **GOCCIOLA**, che si appoggiano a CoCoS per la fase di combinazione.

3.d Collocazione nella pipeline della tesi

Nei capitoli successivi useremo TCL/CoCoS come "motore" di combinazione: dopo la costruzione dei prototipi di base (proprietà rigide/tipiche), combineremo HEAD/MODIFIER per ottenere concetti ibridi; questi prototipi composti alimenteranno classificazione e ranking.

4 Estrazione e pre-processing dei dati (Genius)

In questo capitolo descriviamo come sono stati raccolti e preparati i dati testuali (brani e metadati) da Genius.com, usando la libreria Python `lyricsgenius` per l'accesso all'API e uno script di enrichment per stimare l'indice di ripetizione e derivare tag ausiliari. :contentReference[oaicite:0]{index=0}

4.a Accesso all'API Genius e gestione del token

Per il recupero (o integrazione) dei testi utilizziamo `lyricsgenius`, un client Python che interfaccia sia la **Developer API** (`api.genius.com`) sia la **Public API** di Genius. L'accesso autenticato richiede un **access token** impostato come variabile d'ambiente `GENIUS_TOKEN` (token non versionato nel codice). Installazione: `pip install lyricsgenius`. Uso tipico: istanza `Genius(...)` con opzioni di rate limiting / timeout. :contentReference[oaicite:1]{index=1}

4.b Raccolta dei brani: criteri e formato di output

La raccolta è guidata da una lista di brani per genere (rap, metal, rock, pop, trap, reggae, rnb, country). Per ciascun brano, quando disponibile, persistiamo un record nel JSON unico (`descr_music_GENIUS.json`) con i campi principali: { «ID»: «rap_eminem_rap-god_2013_235729», «genius_id»: 235729, «source»: «genius», «source_url»: «https://genius.com/Eminem-rap-god-lyrics», «title»: «Rap God», «artist»: «Eminem», «album»: «Curtain Call 2», «year»: «2013», «lyrics»: «Testo integrale …», «tags»: [«high_repetition», «rap»], «moods»: [], «instruments»: [], «subgenres»: [«rap»], «contexts»: [], «repetition»: { «rep_ratio»: 0.576, «has_chorus_like»: 0, «has_hook_like»: 0 } }

I campi title/artist/album/year provengono da Genius; lyrics contiene il testo integrale (se presente o recuperabile). Il canale tags è usato anche per i segnali di ripetizione derivati nel pre-processing (sezione seguente). lyricsgenius.readthedocs.io

4.c Enrichment della ripetizione e tag derivati

Lo script enrich_repetition.py elabora il testo e aggiunge al record un blocco repetition con:

Indice di ripetizione

$$\text{rep\_ratio} = 1 - \frac{\text{vocab\_size}}{\text{token\_count}} in [0, 1] \tag{3}$$

su testo normalizzato;

n-gram più frequenti (top_terms, top_bigrams, top_trigrams);

Flag euristici has_chorus_like / has_hook_like (densità n-gram, eventuali etichette [Chorus]/[Hook]). In base a questi indici arricchiamo tags con: high_repetition (se

$$\text{rep\_ratio } ge 0.25 \tag{4}$$

), catchy_chorus e/o hook_repetition. L'arricchimento è idempotente e il JSON viene riscritto con commit atomico. (Questi segnali alimentano le proprietà tipiche usate da TCL/CoCoS nei capitoli successivi.)

4.d Nota sulla variante "extended"

È stata sperimentata una variante extended che, a partire dai prototipi per brano e dai profili typical/rigid, genera per ogni canzone un JSON dedicato (piano di struttura/strumentazione, focus lirico, lyrics da Genius). Per semplicità di integrazione nella pipeline, nella versione finale è stato adottato il JSON unico; la variante è mantenuta come utilità per analisi qualitative.

5 Creazione dei prototipi (Modulo 1)

In questo capitolo descriviamo come, a partire dal JSON `descr_music_GENIUS.json`, vengono generati i prototipi testuali per brano. Il modulo produce un file `.txt` per artwork/istanza con coppie `parola: punteggio` normalizzate in [0.6, 0.9].

5.a Input e obiettivo

Input. Lista di record con campi minimi: `ID`, `title`, `artist`, `album`, `year`, `lyrics`, `tags`, `moods`, `instruments`, `subgenres`, `contexts`. **Output.** Per ogni istanza `ID`, un file `<ID>.txt` con feature lessicali pesate, destinato ai moduli successivi (CoCoS e Recommender).

5.b Pre-processing linguistico

Tokenizzazione: con `nltk.tokenize.word_tokenize`, il tokenizer raccomandato da NLTK (Treebank+Punkt).

Stopword: rimozione delle stopword inglesi dal corpus NLTK.

Lemmatizzazione & POS: quando disponibile, si usa TreeTagger via `treetaggerwrapper` (tag POS e lemma); in fallback, si abbassa a lowercase senza lemma.

Regola leggera di co-occorrenza. Se compare un verbo `lemma` seguito da un sostantivo/parola "contenuto", oltre al lemma del sostantivo si incrementa anche il conteggio del verbo (cattura associazioni verbo→tema).

5.c Estrazione e pesatura delle feature

Per ogni istanza si concatena una "descrizione" dai campi configurati (titolo, artista, lyrics, tags, …), si tokenizza/lemmatizza e si contano le occorrenze grezze. I conteggi sono trasformati in punteggi tramite normalizzazione lineare sull'intervallo [0.6, 0.9]:

$$\text{score} = 0.6 + (0.9 - 0.6) * \frac{\text{freq} - \text{minFreq}}{\text{maxFreq} - \text{minFreq}} \tag{5}$$

dove

$$\text{freq} = \frac{\text{count}(w)}{\text{totWords}} \tag{6}$$

; se

$$\text{maxFreq} = \text{minFreq} \tag{7}$$

, si assegna

$$\text{score} = 0.9 \tag{8}$$

.

5.d Formato dell'output

Un file per istanza, una riga per parola: `word: 0.66` Allineamento spaziato per leggibilità; l'`ID` dell'istanza determina il filename (sanificato per Windows).

5.e Esempio sintetico

Estratto (rap **Rap God**): `rap: 0.645 · lookin: 0.627 · god: 0.624 · --: 0.618 · feel: 0.615 …` (La lista completa è nel file generato in `music_for_cocos`.)

5.f Considerazioni implementative

Robustezza: se TreeTagger non è disponibile, il modulo prosegue senza lemma/POS (solo tokenizzazione+stopword).

Strumenti citati: TreeTagger (POS/lemma multilingue) e `treetaggerwrapper` (wrapper Python); NLTK per tokenizzazione e stopword.

6 Combinazione di generi musicali con CoCoS (Modulo 2)

In questo capitolo descriviamo come, a partire dalle proprietà rigide e tipiche per macro-genere, CoCoS costruisce scenari di combinazione HEAD/MODIFIER e seleziona i prototipi dei concetti ibridi (es. pop–rap).

6.a Input, preprocessing e formato dei file

Input. Le directory typical/ e rigid/ prodotte dal Modulo 2, più la cartella dei .txt per coppia H_M in prototipi_music (creata da cocos_preprocessing.py). Ogni file H_M.txt contiene:

intestazione con Head Concept Name e Modifier Concept Name,

elenco delle rigide di head e modifier,

inclusioni tipiche annotate come

$$p :: T(C) \sqsubseteq D \tag{9}$$

(pesi in

$$(0.5, 1] \tag{10}$$

).

Esempio sintetico (pop–rap).

Head Concept Name: pop Modifier Concept Name: rap

head, pop head, high_repetition

modifier, high_repetition

T(head), pop, 0.95 T(head), high_repetition, 0.629 T(head), hook_repetition, 0.629 T(head), catchy_chorus, 0.6

T(modifier), high_repetition, 0.8

6.b Vincoli e ruolo HEAD/MODIFIER

Le rigide di entrambi i concetti vanno sempre rispettate.

Tra le tipiche, CoCoS privilegia scenari coerenti con la fisionomia dell'HEAD; il MODIFIER introduce aggiustamenti compatibili (cfr. TCL nel cap. 2–3).

Un limite al numero di tipiche selezionabili (parametro MAX_ATTRS in cocos_config.py, default 2) evita combinazioni "sovraccariche".

6.c Generazione e valutazione degli scenari

Dalle tipicità disponibili si costruiscono sottoinsiemi ammissibili (rispetto a rigide e al limite `MAX_ATTRS`). A ciascuno scenario si associa uno **score** in stile DISPONTE (indipendenza dei pesi tipici):

$$\text{score}(S) = \prod_{\{i \in S\}} p_i \tag{11}$$

Gli scenari sono ordinati per score; in caso di parità si mantengono più best.

6.d Selezione e output

Per ciascun file H_M.txt:

si stampano a console gli scenari raccomandati;

si appendono al file due righe:

Result: `{ ... proprietà → peso ..., "@scenario_probability": score }`

Scenario: `[ ... bitmask ..., score ]`

Opzionale: con `-o` si salva un JSON pulito in `prototipi_music/scenarios_json/` con tutti i **best**.

Esempio di risultato (**pop–rap**):

```
Result: {"pop": 0.95, "high_repetition": 0.8, "hook_repetition": 0.629,
"@scenario_probability": 7.0941136, "@scenario_score": 7.0941136}
Scenario: [1, 1, 0, 1, 0, 7.0941136]
```

6.e Parametri rilevanti

MAX_ATTRS in cocos_config.py (default 2): massimo numero di tipiche ereditabili.

CLI (cocos.py):

esecuzione su tutta la cartella configurata: python cocos.py

esecuzione su una coppia: python cocos.py pathH_M.txt 3 -o pathscenarios_json

6.f Casi senza scenario

Se nessuno scenario supera i vincoli (rigide, coerenza, limite di attributi), CoCoS segnala "NO recommended scenarios!" e non modifica il file H_M.txt. Questo

comportamento è utile per evidenziare coppie poco informative o profili troppo scarsi.

6.g Collegamento ai moduli successivi

I prototipi compositi (proprietà → grado + scenario score) alimentano il classificatore e il recommender: sono usati per spiegazioni e ranking, mantenendo trasparenza sui tratti ereditati e sullo scenario scelto.

7 Preprocessing CoCoS: costruzione dei file H_M (Modulo 3a)

In questo capitolo descriviamo il preprocessing che, a partire dai profili per genere (cartelle typical/ e rigid/ prodotte dal Modulo 2), costruisce i file di input per CoCoS in prototipi_music/, uno per ogni coppia Head–Modifier (H_M.txt).

7.a Scopo, input e percorsi

Script. Sistema di raccomandazione/cocos_preprocessing.py Config. cocos_config.py (percorsi TYPICAL_PROP_DIR, RIGID_PROP_DIR, COCOS_DIR).

Input.

typical/.txt: righe prop: peso (es. high_repetition: 0.80).

rigid/.txt: una proprietà per riga (ancore non derogabili).

Output.

prototipi_music/H_M.txt con:

intestazione (titoli Head/Modifier),

rigide di head e modifier,

tipiche annotate come T(head), prop, p e T(modifier), prop, p.

7.b Algoritmo (cocos_preprocessing.py)

Legge rigid e typical per head e modifier.

Scrive H_M.txt in COCOS_DIR con il seguente ordine:

intestazione Title, Head Concept Name, Modifier Concept Name;

blocco rigid dell'head (head,), poi del modifier (modifier,);

blocco typical del modifier (T(modifier), ,);

blocco typical dell'head (T(head), ,).

L'ordine rigid → T(modifier) → T(head) è coerente con il ruolo Head/ Modifier usato da CoCoS nel modulo successivo.

7.c Esecuzione

Su una coppia specifica: `python cocos_preprocessing.py <HEAD> <MODIFIER>`

Batch su tutte le coppie (H ≠ M). Vedi snippet PowerShell già usato:

SNIPPET CHE NON IRESCO AD INCOCLLARE

7.d Formato del file H_M.txt (esempi)

country–metal:

Title: country-metal Head Concept Name: country Modifier Concept Name: metal

head, country head, high_repetition

modifier, metal modifier, high_repetition

T(modifier), metal, 0.95 T(modifier), high_repetition, 0.6

T(head), country, 0.95 T(head), high_repetition, 0.6

metal–country (simmetrico ma invertiti i ruoli):

Title: metal-country Head Concept Name: metal Modifier Concept Name: country

head, metal head, high_repetition

modifier, country modifier, high_repetition

T(modifier), country, 0.95 T(modifier), high_repetition, 0.6

T(head), metal, 0.95 T(head), high_repetition, 0.6

7.e Note pratiche

Le rigid sono riportate come vincoli duri e saranno sempre rispettate da CoCoS.

I pesi delle typical vengono copiati dai file di genere (range tipico [0.6, 0.95]).

È utile generare sia H_M.txt sia M_H.txt: l'esito della combinazione dipende dal ruolo Head/Modifier.

I percorsi sono centralizzati in cocos_config.py (ad es. COCOS_DIR per la destinazione dei file).

7.f Collegamento al Modulo 3b

I file H_M.txt prodotti qui sono consumati da cocos.py, che costruisce gli scenari di combinazione, seleziona i best e li appende al file (oltre a generare, se richiesto, un JSON con gli scenari raccomandati).

8 Sistema di raccomandazione (Modulo 4)

Il modulo finale prende i prototipi ibridi generati da CoCoS e filtra/ordina i brani del dataset originale, producendo per ogni coppia HEAD_MODIFIER un file JSON con gli item consigliati e l'evidenza delle proprietà soddisfatte.

8.a Input e panoramica

Input.

I file H_M.txt arricchiti da CoCoS con le rigide, le tipiche e le due righe finali Result: e Scenario:.

Il JSON sorgente dei brani configurato in Recommender_config.py (stessi campi usati nei moduli precedenti).

Output. Per ogni H_M.txt viene salvato un H_M_recommendations.json con: categoria, prototipo attivo, ancore, lista dei risultati id–titolo–artista–matches–anchors_hit, numero di brani classificati su totale. Gli output vengono poi spostati in prototipi_music/recommender_out/.

8.b Dal file CoCoS al prototipo "pulito"

Il classificatore legge il Result: e l'array Scenario: del file H_M.txt e costruisce il prototipo attivo:

seleziona le tipiche con bit 1 nello scenario;

include le ancore (le rigide del file e, più in generale, le proprietà marcate obbligate nella sezione iniziale);

rimuove duplicati e normalizza i nomi.

Il risultato è una lista di proprietà da soddisfare più l'insieme delle ancore richieste.

8.c Regole di matching

Per ogni brano del JSON sorgente il sistema:

cerca ogni proprietà del prototipo nei campi configurati in Recommender_config.py (titolo + campi descrittivi);

opzionalmente scarta il brano se contiene proprietà negate (prefisso - nel file prototipo);

accetta il brano se sono vere entrambe le soglie:

match-rate minimo (default 0.15 delle proprietà del prototipo),

minimo numero di ancore colpite (default 1).

Le soglie sono modificabili da CLI: –min-match-rate, –min-anchors (e –max-print per limitare le righe stampate a console).

8.d Formato dell'output

Ogni JSON contiene:

category: il nome del file prototipo (head_modifier.txt);

prototype: lista piatta delle proprietà del prototipo attivo;

anchors: ancore richieste;

results: array di oggetti { id, title, artist, matches, anchors_hit };

classified e total.

Esempio sintetico (rock_pop):

{ «category»: «rock_pop.txt», «prototype»: [«rock»,«hook_repetition»,«catchy_chorus»,«pop»,«high_repetition»], «anchors»: [«catchy_chorus»,«rock»,«high_repetition»,«pop»,«hook_repetition»], «results»: [{ «id»: «…*rap-god*…», «title»: «Rap God», «artist»: «Eminem», «matches»: [«high_repetition»], «anchors_hit»: [«high_repetition»] }, …], «classified»: 48, «total»: 48 }

8.e Uso a riga di comando e integrazione nella pipeline

Singolo prototipo. python Sistema di raccomandazione/Classificatore/ Recommender.py prototipi_musicH_M.txt

Batch su tutte le coppie. lo script PowerShell del progetto itera su tutti i H_M.txt, invoca il classificatore e sposta gli output nella cartella recommender_out/.

8.f Considerazioni e limiti

Spiegabilità. Ogni suggerimento riporta le proprietà che hanno fatto match e quali ancore sono state colpite.

Ranking. L'attuale versione filtra per soglie e non ordina con uno score continuo; in prospettiva si può introdurre un ranking per numero/"peso" di proprietà soddisfatte (es. pesi tipici di CoCoS).

Parametri. Le soglie di copertura e ancore permettono di rendere il sistema più "severo" o "inclusivo" senza modificare i prototipi di partenza.

Coerenza. Se CoCoS non aveva prodotto scenari per una coppia, non esiste file Result: valido e non viene generato alcun JSON di raccomandazioni per quella categoria.

8.g Collegamento ai capitoli successivi

I JSON di raccomandazione alimentano sia i Risultati sia le Spiegazioni, dove analizziamo copertura, varietà e casi tipici di match per le combinazioni testate.

9 Risultati

In questo capitolo sintetizziamo gli esiti della pipeline: prototipi compositi → classificatore/raccomandatore per ciascuna coppia HEAD/MODIFIER. Gli output sono file **recommendations.json che, per ogni coppia, riportano: categoria, prototipo (proprietà selezionate), ancore rigide/forti, lista dei brani consigliati con le proprietà effettivamente colpite, e contatori classified/total.**

9.a Metriche e formato

Ogni file espone:

prototype: proprietà del concetto ibrido (es. [«trap»,«high_repetition»,«metal»]).

anchors: sottoinsieme di proprietà chiave usate come vincoli/ancore.

results: brani con matches (proprietà colpite) e anchors_hit (ancore soddisfatte).

classified / total: copertura della classificazione per la coppia.

Esempio (estratto trap–metal): prototype = [«trap»,«high_repetition»,«metal»], classified = 48, total = 48.

9.b Copertura

Sulle coppie analizzate, la copertura è piena in molti casi (classified = total = 48, es. trap–metal, rap–rnb). In alcuni accoppiamenti con profili più scarsi, la copertura può ridursi (casi 79% osservati a log su rock–reggae), coerente con l'assenza di scenari o proprietà insufficienti nel prototipo.

9.c Coerenza dei suggerimenti con il prototipo

Trap–Metal. Il prototipo ibrido punta su trap, metal, high_repetition. Nei risultati:

tracce trap (Travis Scott, Future) colpiscono trap + high_repetition e soddisfano entrambe le ancore quando presenti (anchors_hit: trap, high_repetition);

classici metal (Metallica, Judas Priest) colpiscono metal + high_repetition;

brani di altri generi appaiono quando soddisfano comunque l'ancora di ripetizione (es. Weeknd, AC/DC, Marley), a conferma del ruolo trasversale di high_repetition.

Rap–R&B. Il prototipo enfatizza rnb + high_repetition. I suggerimenti includono Beyoncé e TLC (hit di rnb + high_repetition) e, per la componente di ripetizione, anche rap/pop/rock con forte hook (Eminem, Weeknd, AC/DC) quando soddisfano l'ancora.

9.d Interpretazione

Ancore forti funzionano bene. Dove il prototipo contiene proprietà altamente distintive (es. trap, metal, rnb) la lista privilegia brani emblematici dei generi coinvolti, mantenendo diversità sul terzo asse (high_repetition).

Segnali trasversali spiegano intersezioni ampie. high_repetition porta nella top anche brani di altri generi con forte chorus/hook: spiegazione coerente con i tag generati nell'enrichment.

9.e Esempi puntuali

Trap–Metal → "SICKO MODE" (Travis Scott). matches = [«high_repetition»,«trap»], anchors_hit = [«high_repetition»,«trap»]. Metal → "Enter Sandman" (Metallica). matches = [«high_repetition»,«metal»], anchors_hit = [«high_repetition»,«metal»].

Rap–R&B → "Drunk in Love" (Beyoncé). matches = [«high_repetition»,«rnb»], anchors_hit = [«high_repetition»,«rnb»]. Rap–R&B → "Rap God" (Eminem). matches = [«high_repetition»], richiamato dalla forte ripetizione anche se non rnb.

9.f Limiti osservati

Dipendenza da high_repetition. Essendo un segnale orizzontale, può allargare troppo la platea se il prototipo non contiene altre tipiche/rigide selettive; l'effetto è voluto per esplorare cross-over, ma va bilanciato in fase di presentazione.

Copertura non uniforme. Dove i profili di genere sono scarsi (poche tipiche/rigide) la selezione può risultare vuota o parziale (casi "NO scenario" in CoCoS e coperture $< 100\%$ a valle).

9.g Takeaway

Il raccomandatore preserva le scelte di CoCoS: le ancore del prototipo ibrido guidano effettivamente i suggerimenti.

I tag di ripetizione arricchiti su Genius si riflettono nei risultati, favorendo brani con hook/chorus marcati anche fuori dal macro-genere dell'HEAD/ MODIFIER.

Con profili più ricchi (più tipiche non trasversali) ci si attende una maggiore precisione semantica e minore dipendenza da high_repetition.

10 Discussione

Inquadriamo i risultati alla luce della pipeline (prototipi → CoCoS → raccomandatore), discutendo criticità, confronto con approcci affini e implicazioni d'uso.

10.a Dove il sistema fallisce (e perché)

Propagazione degli errori. Ogni modulo erede i vincoli del precedente: se i profili typical/rigid sono scarsi o sbilanciati, CoCoS genera pochi scenari o nessuno; il raccomandatore, a cascata, ha copertura ridotta o suggerimenti troppo generici.

Dominanza di segnali trasversali. Tag orizzontali (es. high_repetition) migliorano la copertura ma, se gli altri tratti sono deboli, allargano troppo la platea e "schiacciano" la specificità di genere. Rimedi pratici:

aumentare topk_typical solo se esistono tratti non-trasversali;

alzare la penalità dei "globaloni" (COMMON_PENALTY) o innalzare le soglie typical_thr;

usare MAX_ATTRS ≥ 2 ma con almeno una tipica fortemente distintiva dell'HEAD.

Assunzioni di indipendenza. Lo score degli scenari moltiplica pesi tipici come se fossero indipendenti. Quando due proprietà sono correlate (es. trap e 808), lo scenario può sovrastimarle. Possibili fix: regole di mutua esclusione/coesione o "gruppi" di feature con peso congiunto.

Heuristics Head/Modifier. La priorità semantica all'HEAD è utile ma rigida: alcuni mix (es. pop↔rnb) potrebbero richiedere simmetria o switch dinamico del ruolo. Si può introdurre un parametro di "plasticità" o una scelta H/M guidata dai punteggi.

Rumore testuale. Tokenizzazione/stopword inglesi e qualità variabile dei testi di Genius introducono:

sinonimia superficiale (manca normalizzazione a lemmi se TreeTagger non è disponibile);

lessico di dominio non catturato se fuori whitelist;

mappe SUB2MACRO incomplete su tag rari. Mitigazioni: ampliare DOMAIN_WHITELIST, raffinando SUB2MACRO e (se disponibile) usare lemmatizzatori robusti.

10.b Confronto con approcci affini

Baselines "bag-of-words" o tf–idf. Offrono buone similitudini ma non distinguono tra tipicità (difettibile) e vincoli rigidi; mancano ruoli HEAD/MODIFIER e non motivano perché certe proprietà "saltano" o vengono sospese.

Collaborative filtering / embedding neurali. Predicono bene preferenze e vicinanze, ma la spiegabilità è bassa e la combinazione concettuale richiede hack (intersezione di vicini). Qui, invece, il prototipo ibrido è esplicito e auditabile, con scenari e pesi tracciabili.

TCL/CoCoS. Rispetto a semplici regole logiche o fuzzy, aggiunge: (i) priorità gerarchica e sospendibilità delle tipiche, (ii) selezione per scenari con punteggi, (iii) ruolo H/M. Il trade-off è una maggiore sensibilità alla qualità dei profili e all'ipotesi d'indipendenza dei pesi.

10.c Implicazioni pratiche

Trasparenza e controllo. Ogni suggerimento eredita ancore e matches del prototipo: l'utente (o il curatore) può vedere quali tratti hanno governato il ranking e regolare soglie/whitelist per pilotare le proposte.

Discovery di crossover. La presenza di segnali trasversali (hook, ripetizione) fa emergere brani di generi "lontani" ma plausibili rispetto al concept ibrido. È utile per playlist tematiche, format radio e ideazione creativa (moodboard sonori).

Cura dei profili. Il sistema incentiva la manutenzione leggera dei dizionari: aggiungere 2–3 tipiche distintive per genere migliora molto gli scenari; la rigidità va usata con parsimonia (ancore poche ma forti).

10.d Minacce alla validità

Copertura dati. Il campione è limitato; pochi esempi per un genere riducono la stabilità dei typical/rigid.

Bias di sorgente. Testi/metadata di Genius riflettono cataloghi e pratiche editoriali specifiche.

Scelte di iperparametri. MIN_W/MAX_W, soglie e MAX_ATTRS influenzano direttamente gli scenari; risultati diversi con settaggi diversi.

10.e Cosa migliorare subito

Rinforzare la specificità. Aumentare DISTINCTIVE_BOOST o arricchire i profili con 2–3 tratti non-trasversali per genere (riduce l'effetto calamita di high_repetition).

Regole di coerenza. Piccolo set di vincoli: se trap allora preferisci 808; se reggae evita double_kick.

Selezione scenari soft. Oltre al best, tenere top-k scenari e far decidere al raccomandatore con un rimescolamento pesato (diversifica le playlist).

Diagnostica di copertura. Report automatico: brani non classificati per coppia, proprietà mai attivate, rigid che azzerano gli scenari.

Arricchimento linguistico. Estendere whitelist e mappature SUB2MACRO; dove possibile, lemmatizzazione stabile per ridurre la frammentazione del lessico.

10.f Takeaway

Il paradigma prototipi + combinazione fornisce spiegazioni locali e controllabilità globale con pochissimi iperparametri.

La qualità dei profili typical/rigid è la leva principale: quando sono ricchi, gli scenari sono sensati e le raccomandazioni coerenti; quando sono poveri, prevale il segnale trasversale.

Il sistema è adatto a discovery e curation di crossover, lasciando spazio a moduli neurali/CF come re-ranker, mantenendo però tracciabilità delle scelte.

11 Conclusioni e sviluppi futuri

Ricapitoliamo contributi e risultati e indichiamo le direzioni successive.

11.a Conclusioni

Che cosa abbiamo dimostrato e con quali limiti.

11.b Sviluppi futuri

- Integrazione di feature audio e metadata strutturati.
- Estensione multilingua.
- Lessici d'intensità (es. VAD) e aspetti temporali del brano.
- Studio utente più ampio / A/B test sulle spiegazioni.

Bibliografia / Sitografia

[1] *UniTO Typst Template*. (2024). [Online]. Disponibile su: https://github.com/eduardz1/UniTO-typst-template

[2] «Typst — A new markup-based typesetting system». [Online]. Disponibile su: https://typst.app/

[3] Alberto Marocco, *DEGARI-Music*. (2025). [Online]. Disponibile su: https://github.com/albymar01/DEGARI-Music

[4] Alberto Marocco, *Tesi-UniTO (manoscritto)*. (2025). [Online]. Disponibile su: https://github.com/albymar01/Tesi-UniTO

[5] «Genius». [Online]. Disponibile su: https://genius.com/

[6] «Scrapy». [Online]. Disponibile su: https://docs.scrapy.org/en/latest/

[7] «NLTK — Natural Language Toolkit». [Online]. Disponibile su: https://www.nltk.org/

[8] «TreeTaggerWrapper». [Online]. Disponibile su: https://treetaggerwrapper.readthedocs.io/

[9] Helmut Schmid, «TreeTagger». [Online]. Disponibile su: https://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/

[10] A. Valese, «CoCoS: uno strumento per la combinazione di concetti», 2020.

[11] «scikit-learn». [Online]. Disponibile su: https://scikit-learn.org/

[12] «pandas». [Online]. Disponibile su: https://pandas.pydata.org/

[13] «NumPy». [Online]. Disponibile su: https://numpy.org/

[14] «Matplotlib». [Online]. Disponibile su: https://matplotlib.org/

[15] Peter Gärdenfors, «Concept Combination and Prototypes», 2004.

[16] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, e Clifford Stein, *Introduction to Algorithms*, 3rd ed. MIT Press, 2009.

Ringraziamenti

Desidero esprimere la mia sincera gratitudine al Prof. Gian Luca Pozzato per la sua guida e supporto durante lo sviluppo di questa tesi. Un ringraziamento speciale va anche ai miei amici e familiari per il loro incoraggiamento costante.