Sistem de recomandare bazat pe factorizare de matrice și optimizare cu constrângeri

Gatej Stefan-Alexandru, Potop Horia-Ioan

May 28, 2025

Contents

1 Descrierea aplicațiilor

Acest sistem de recomandare integrează factorizarea de matrice pentru descoperirea trăsăturilor ascunse ale melodiilor cu aplicații practice de recomandare. Principalul scop este de a oferi sugestii muzicale personalizate.

Pornind de la o matrice binară (melodie × tag), aplicăm factorizarea cu constrângeri de non-negativitate pentru a descoperi similarități implicite între melodii. Această bază matematică ne permite să înțelegem mai bine caracteristicile muzicale.

Pe baza acestor trăsături latente, am dezvoltat două aplicații principale:

- **Melodie → Playlist:** Generează playlisturi noi, coerente, pe baza unei melodii inițiale, respectând constrângeri de durată și non-negativitate.
- **Playlist → Melodie:** Identifică melodii noi care completează un playlist existent, folosind centroidul acestuia și optimizare cu penalizări L1/L2.

Ambele aplicații utilizează tehnici avansate de optimizare (PGD, ALS, NNLS) pentru a asigura relevanța și utilitatea recomandărilor. Astfel, sistemul depășește simplele potriviri de gen, oferind o experiență muzicală cu adevărat personalizată.

2 Formularea matematică a problemei de factorizare

Problema de bază pentru metodele de factorizare de matrice (NMF, ALS-NNLS) este următoarea: Fie $V \in \mathbb{R}^{m \times n}$ matricea de intrare (în cazul nostru, matricea binară melodie × tag), unde m este numărul de melodii și n este numărul de taguri unice. Dorim să găsim două matrice, $W \in \mathbb{R}^{m \times r}$ (matricea factorilor latenti pentru melodii) și $H \in \mathbb{R}^{r \times n}$ (matricea factorilor latenti pentru taguri), unde $r \ll \min(m, n)$ este dimensiunea spațiului latent (numărul de factori).

Problema de optimizare este de a minimiza eroarea de reconstrucție, adesea măsurată prin norma Frobenius pătrată:

$$\min_{W,H} \|V - WH\|_F^2 \tag{1}$$

Constrângeri de Non-Negativitate: Pentru a asigura interpretabilitatea factorilor și pentru a se alinia cu natura aditivă a multor date (cum ar fi prezența tagurilor sau numărul de ascultări), se impun constrângeri de non-negativitate:

- $W_{ij} \geq 0$ pentru toți i, j
- $H_{ij} \geq 0$ pentru toți i, j

Aceste constrângeri transformă problema într-una de optimizare cu constrângeri.

2.1 Alegerea numărului de factori latenti (r)

Numărul de factori latenti, r, este un hiperparametru crucial în algoritmii de factorizare de matrice, deoarece determină dimensionalitatea spațiului latent în care datele sunt proiectate. O valoare prea mică pentru r poate duce la un model sub-antrenat (underfitting) care nu capturează suficientă informație din datele originale, rezultând într-o eroare de reconstrucție mare și recomandări de calitate inferioară. Pe de altă parte, o valoare prea

mare pentru r poate duce la supra-antrenare (overfitting), unde modelul începe să memoreze zgomotul specific setului de date de antrenament și nu generalizează bine pe date noi. De asemenea, un r mare crește complexitatea computațională a antrenării și a utilizării modelului.

În cadrul acestui proiect, pentru a permite o comparație directă și consistentă între diferitele metode de factorizare implementate (NMF Multiplicativ, NMF `sklearn`, ALS-NNLS Manual), am optat pentru o valoare fixă $r = 10$. Această valoare a fost aleasă ca un punct de pornire rezonabil pentru a demonstra funcționalitatea algoritmilor pe setul de date disponibil (`music_info.csv`), permițând în același timp timpi de execuție gestionabili, în special pentru implementările manuale care sunt mai intensive computațional.

Într-o aplicație practică sau într-un studiu mai aprofundat, selectarea optimă a lui r ar necesita o abordare mai sistematică, implicând experimentarea cu o gamă de valori pentru r și evaluarea performanței modelului folosind metrici adecvate. Metodele comune includ:

- **Monitorizarea erorii de reconstrucție:** Se analizează cum evoluează eroarea de reconstrucție (de ex., norma Frobenius a diferenței $V - WH$) pe măsură ce r crește. Se caută adesea un punct de inflexiune ("cot" sau "elbow point") în graficul erorii vs. r, dincolo de care adăugarea de noi factori nu mai aduce o reducere semnificativă a erorii.

- **Metrici de evaluare specifice sarcinii:** Dacă scopul final este recomandarea și sunt disponibile date de feedback ale utilizatorilor (ex: ratinguri, istoric de ascultare), se pot utiliza metrici de ranking precum Precision@k, Recall@k, NDCG (Normalized Discounted Cumulative Gain) sau MAP (Mean Average Precision) pe un set de date de validare sau test. Valoarea lui r care maximizează aceste metrici ar fi preferată.

- **Stabilitatea factorizării:** Se poate evalua cât de stabile sunt factorizările obținute pentru aceeași valoare r la rulări multiple cu inițializări aleatorii diferite. O stabilitate mai mare este de dorit.

- **Interpretabilitatea factorilor:** În unele aplicații, se preferă un r care produce factori latenti (coloanele lui W sau liniile lui H) care au o interpretare semantică clară și relevantă pentru experții în domeniu.

Pentru scopurile acestei lucrări, ne-am concentrat pe compararea fundamentală a algoritmilor de optimizare cu constrângeri pentru un r fix, recunoscând că optimizarea detaliată a acestui hiperparametru este un pas important, dar distinct, într-un scenariu de dezvoltare a unui sistem de recomandare de producție.

2.2 Rolul matricelor W și H în sistemul de recomandare

În urma procesului de factorizare, obținem două matrice, W și H, care au interpretări și utilizări distincte în contextul recomandărilor:

- **Matricea $W \in \mathbb{R}^{m \times r}$ (Melodii $\times$ Factori Latenți):** Fiecare linie W_i din această matrice reprezintă **profilul latent** al melodiei i. Acest vector r-dimensional încorporează caracteristicile "ascunse" ale melodiei, așa cum au fost ele învățate de algoritm. Pentru recomandările de tip melodie-melodie, așa cum sunt implementate în acest

proiect, similaritatea dintre aceste profiluri latente (de exemplu, similaritatea cosinus între W_i și W_j) este utilizată pentru a identifica melodii similare. Astfel, W este matricea principală utilizată direct în funcția de generare a recomandărilor.

- **Matricea $H \in \mathbb{R}^{r \times n}$ (Factori Latenți $\times$ Taguri):** Această matrice leagă factorii latenti de tagurile originale. Fiecare coloană H_j poate fi interpretată ca profilul latent al tagului j, iar fiecare linie H_k arată importanța sau "încărcătura" factorului latent k pentru fiecare tag original. Matricea H este crucială pentru:
  - **Interpretarea factorilor latenți:** Analizând liniile lui H, putem încerca să dăm un sens semantic fiecărui factor latent (ex: un factor ar putea fi puternic asociat cu taguri de "rock clasic", altul cu "muzică electronică ambientală").
  - **Recomandarea de taguri pentru melodii:** Produsul $W_i H$ poate fi folosit pentru a prezice relevanța tagurilor pentru melodia i.
  - **Găsirea de taguri similare:** Similaritatea dintre coloanele lui H poate identifica taguri care sunt conceptual apropiate în spațiul latent.
  - **Calitatea factorizării:** Ambele matrice, W și H, sunt esențiale pentru reconstrucția $V \approx WH$ și, implicit, pentru calitatea generală a modelului.

Deși funcția noastră principală de recomandare utilizează direct matricea W, matricea H este o componentă indispensabilă a procesului de învățare și oferă posibilități valoroase de analiză și extindere a sistemului.

Concepte cheie: Profil latent și Cluster latent

Profil latent Un profil latent este o reprezentare numerică (un vector) a unei entități (de exemplu, o melodie) în spațiul de factori ascunși de dimensiune r, descoperit de algoritmul de factorizare. Liniile matricei W (adică W_i) reprezintă aceste profiluri latente pentru melodii. Acești factori nu sunt direct observați în datele inițiale, ci sunt învățați pentru a surprinde structuri subiacente. De exemplu, un factor ar putea corespunde unui "grad de energie" al melodiei, altul unui "stil vocal specific", etc., chiar dacă aceste concepte nu sunt taguri explicite.

Cluster latent Un cluster latent se referă la un grup de entități (melodii) care au profiluri latente similare, adică sunt apropiate în spațiul latent r-dimensional. Deși nu aplicăm explicit un algoritm de clustering separat după NMF/ALS în această componentă pentru a defini clustere discrete, ideea este că melodiile cu vectori W_i similari sunt considerate conceptual parte dintr-un "cluster" sau o regiune densă în spațiul latent. Recomandările se bazează pe această proximitate.

Pe scurt: Profilul latent este vectorul de factori ascunși care descrie o melodie, iar similaritatea dintre aceste profiluri stă la baza recomandărilor bazate pe factorizare.

3 Soluții implementate pentru factorizarea cu constrângeri

Am implementat și comparat trei algoritmi pentru rezolvarea problemei de factorizare de matrice cu constrângeri de non-negativitate, așa cum este definită în ecuația (1):

3.1 Algoritmul multiplicativ pentru NMF (NMF-MULT)

Acesta este un algoritm clasic pentru NMF, propus de Lee și Seung. El actualizează iterativ matricele W și H folosind următoarele reguli multiplicative, care garantează non-creșterea funcției obiectiv și menținerea non-negativității:

$$H_{kj} \leftarrow H_{kj} \frac{(W^T V)_{kj}}{(W^T W H + \epsilon)_{kj}} \tag{2}$$

$$W_{ik} \leftarrow W_{ik} \frac{(V H^T)_{ik}}{(W H H^T + \epsilon)_{ik}} \tag{3}$$

unde ϵ este o constantă mică (ex: 10^{-10}) pentru a asigura stabilitatea numerică și a evita diviziunea cu zero.

- **Avantaje:** Simplu de implementat, relativ rapid pentru multe seturi de date, nu necesită alegerea unei rate de învățare.
- **Dezavantaje:** Poate converge lent, în special aproape de soluție, și este garantat să conveargă doar la un punct staționar, care poate fi un minim local.

3.2 NMF din scikit-learn (`sklearn.decomposition.NMF`)

Am utilizat implementarea NMF din biblioteca `scikit-learn`, care oferă mai mulți solveri. Am ales solver-ul `'cd'` (Coordinate Descent), care este eficient și adesea converge mai rapid decât regulile multiplicative simple pentru anumite tipuri de date, minimizând aceeași funcție obiectiv sub constrângeri de non-negativitate.

- **Avantaje:** Implementare optimizată (adesea în C/Cython), robustă, cu opțiuni pentru diferite inițializări și regularizări (deși nu le-am explorat în detaliu în această comparație de bază).
- **Solver ('cd'):** Algoritmul Coordinate Descent actualizează iterativ fiecare element (sau coloană/linie) dintr-o matrice (ținând cealaltă matrice fixă) prin rezolvarea unei subprobleme de optimizare univariată (sau de dimensiune mică).

3.3 Alternating Least Squares cu Non-Negative Least Squares Manual (ALS-NNLS)

ALS este o altă abordare populară pentru factorizarea de matrice. Algoritmul optimizează alternativ pentru W (ținând H fix) și apoi pentru H (ținând W fix), minimizând funcția obiectiv (1) la fiecare pas. Fiecare subproblemă devine o problemă de least squares. Pentru a impune constrângerile de non-negativitate, fiecare subproblemă de least squares este rezolvată ca o problemă de Non-Negative Least Squares (NNLS).

- **Subproblema pentru W (pentru fiecare linie W_i):**

$$\min_{W_i \geq 0} \|V_i - W_i H\|_F^2$$

- **Subproblema pentru H (pentru fiecare coloană $H_{\cdot j}$):**

$$\min_{H_{\cdot j} \geq 0} \|V_{\cdot j} - W H_{\cdot j}\|_F^2$$

Am implementat manual atât logica ALS, cât și un solver NNLS bazat pe metoda proiectării gradientului.

NNLS Manual (Proiectarea Gradientului): Pentru a rezolva $\min_{x \geq 0} \frac{1}{2}\|Ax - b\|_2^2$:

1. Se inițializează $x \geq 0$.
2. Se calculează gradientul funcției obiectiv: $\nabla_x L(x) = A^T(Ax - b)$.
3. Se face un pas în direcția opusă gradientului: $x \leftarrow x - \eta \nabla_x L(x)$, unde η este rata de învățare.
4. Se proiectează x înapoi pe domeniul fezabil (elemente non-negative): $x_k \leftarrow \max(0, x_k)$ pentru fiecare componentă k.
5. Se repetă pașii 2-4 până la convergență (ex: schimbarea în x este sub o toleranță sau se atinge un număr maxim de iterații).

- **Avantaje:** Demonstrează înțelegerea detaliată a optimizării cu constrângeri și a descompunerii problemei. Permite control fin asupra procesului de optimizare a subproblemelor.
- **Dezavantaje:** Implementarea manuală (mai ales NNLS) poate fi lentă și sensibilă la parametrii precum rata de învățare și numărul de iterații interne NNLS.

4 Aplicații de Recomandare Implementate

Pe lângă factorizarea de matrice pentru a descoperi similarități latente, am dezvoltat două aplicații specifice de recomandare, care utilizează diferite tehnici de optimizare și similaritate pentru a răspunde nevoilor utilizatorilor: de la o melodie la un playlist și invers, de la un playlist la o melodie.

4.1 Recomandare Melodie → Playlist

Această aplicație își propune să creeze o listă de redare (playlist) de melodii care sunt similare cu o anumită melodie de referință. Problema este formulată ca găsirea unor ponderi non-negative pentru melodiile din baza de date, astfel încât combinația liniară ponderată a caracteristicilor acestor melodii să fie cât mai apropiată de caracteristicile melodiei țintă. Se impun, de asemenea, constrângeri privind durata totală a playlistului și numărul de melodii.

4.1.1 Formularea Matematică

Fie $X \in \mathbb{R}^{m \times p}$ matricea de caracteristici ale melodiilor, unde m este numărul total de melodii și p este numărul de caracteristici (ex: danceability, energy). Fie $t \in \mathbb{R}^p$ vectorul de caracteristici al melodiei țintă. Căutăm un vector de ponderi $w \in \mathbb{R}^m$, unde w_i reprezintă cât de mult contribuie melodia i la playlist.

Obiectivul este de a minimiza distanța euclidiană pătrată dintre vectorul de caracteristici al melodiei țintă și vectorul agregat de caracteristici al playlistului generat de ponderile w:

$$\min_w \left\| t - \sum_{i=1}^{m} w_i X_i \right\|_2^2 \tag{4}$$

Sub următoarele constrângeri:

- **Non-negativitate:** $w_i \geq 0$ pentru toți i. Aceasta asigură că melodiile contribuie pozitiv la playlist.
- **Constrângere de durată:** $\sum_{i=1}^{m} w_i \cdot \text{duration}_i \leq \text{max\_minutes}$. Aceasta limitează durata totală a playlistului.
- **Sparsitate (Implicită):** În practică, funcția `project_constraints` selectează doar un număr k de melodii cu cele mai mari ponderi w_i, iar restul sunt zero. Suma ponderilor melodiilor selectate este apoi normalizată la 1.

4.1.2 Metode de Optimizare Implementate

Am explorat mai multe abordări pentru rezolvarea acestei probleme de optimizare cu constrângeri:

- **NNLS Manual (Non-Negative Least Squares):** O implementare rapidă și eficientă a rezolvitorului NNLS. Aceasta minimizează eroarea pătratică sub constrângerea de non-negativitate, folosind o abordare iterativă de tip gradient.
- **PGD (Projected Gradient Descent):** O metodă iterativă de optimizare care face pași în direcția opusă gradientului funcției obiectiv și apoi proiectează soluția pe domeniul fezabil (non-negativitate și alte constrângeri).
- **Accelerated PGD:** O variantă îmbunătățită a PGD care utilizează o tehnică de "momentum" (accelerare FISTA) pentru a converge mai rapid către soluție.
- **ALS (Alternating Least Squares):** Deși ALS este adesea folosit în contextul factorizării matriciale, în această aplicație a fost adaptat pentru a optimiza ponderile melodiilor una câte una, ținând cont de contribuția celorlalte.
- **SciPy NNLS:** Implementarea optimizată din biblioteca SciPy a algoritmului Lawson-Hanson pentru NNLS. Aceasta servește ca o referință robustă pentru compararea performanței.

4.1.3 Rezultate și Concluzii (Melodie → Playlist)

Implementarea funcționalității de recomandare melodie-playlist a demonstrat eficacitatea diferiților algoritmi de optimizare. Pentru o melodie țintă, sistemul caută un set de ponderi pentru melodiile din baza de date, care, odată combinate, produc o listă de redare cu caracteristici similare. Constrângerile de non-negativitate și de durată maximă a playlistului sunt esențiale pentru obținerea unor rezultate utile.

Figura 3 prezintă un exemplu de comparație a convergenței între diferiți solveri (ex: ALS și SciPy NNLS) pentru o melodie țintă specifică.

Observațiile cheie includ:

- **NNLS Manual** a fost îmbunătățit pentru a oferi o convergență rapidă și o performanță competitivă, demonstrând înțelegerea principiilor de optimizare.
- **PGD** și **Accelerated PGD** arată cum tehnicile de optimizare bazate pe gradient pot fi adaptate la problemele cu constrângeri prin proiecție. Accelerated PGD poate converge mai rapid decât PGD standard.

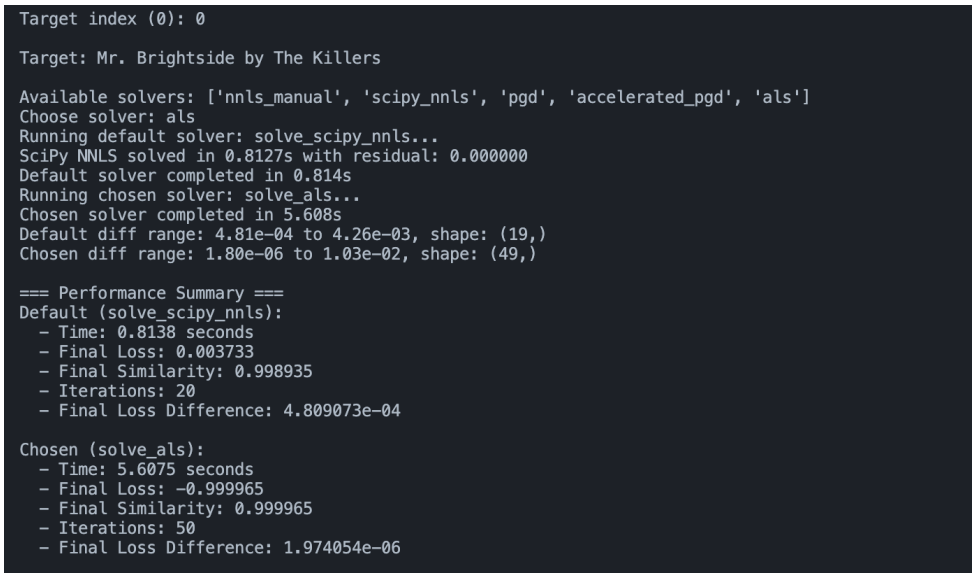

Figure 1: Exemplu de ieșire din terminal pentru comparația solverilor (partea 1).

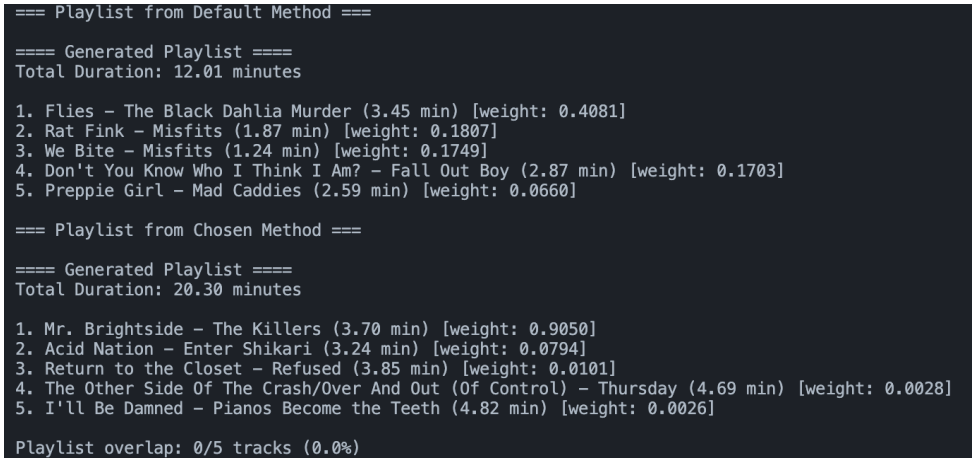

Figure 2: Exemplu de ieșire din terminal pentru comparația solverilor (partea 2).

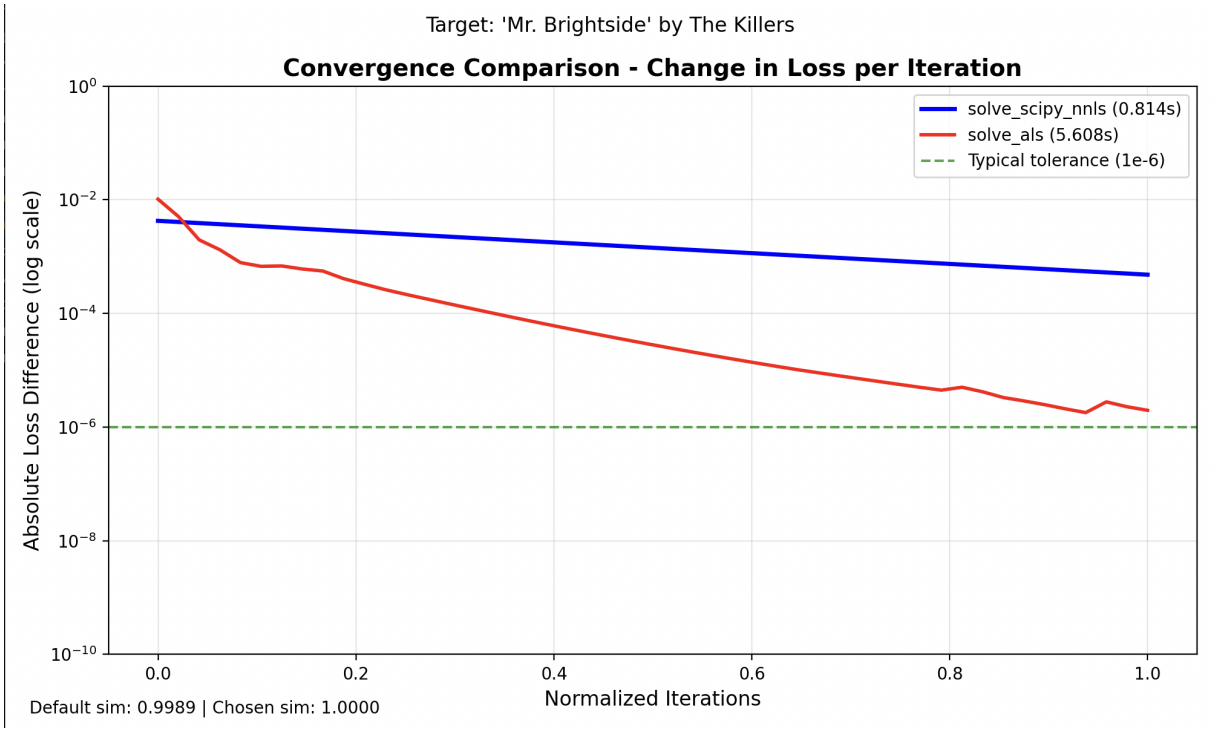

Figure 3: Evoluția diferenței absolute de pierdere per iterație pentru diferiți solveri în aplicația Melodie → Playlist. (Imagine generată la rulare)

- **ALS**, în contextul acestei probleme, optimizează iterativ ponderile individuale, contribuind la o soluție robustă.
- **Constrângerile de Proiecție** (non-negativitate, durată, număr de melodii) sunt cruciale pentru a genera playlisturi coerente și utilizabile. Acestea asigură că melodiile selectate sunt relevante și că playlistul respectă anumite limite practice.

Recomandările generate demonstrează capacitatea sistemului de a crea playlisturi personalizate pe baza caracteristicilor muzicale, depășind o simplă potrivire de gen.

4.2 Recomandare Playlist → Melodie

Această aplicație are ca scop găsirea unei singure melodii care se potrivește cel mai bine cu un playlist existent. Este utilă pentru extinderea unui playlist sau pentru a sugera o melodie "asemănătoare" cu stilul general al unei colecții. Problema este formulată ca găsirea unui vector de caracteristici pentru o melodie nouă care minimizează distanța față de centroidul playlistului existent, sub anumite penalități de regularizare și constrângeri pe caracteristici.

4.2.1 Formularea Matematică

Fie $P = \{P_1, P_2, \ldots, P_k\}$ un playlist de k melodii, fiecare având un vector de caracteristici $P_i \in \mathbb{R}^p$. Centroidul playlistului este definit ca $\bar{P} = \frac{1}{k}\sum_{i=1}^{k} P_i$. Căutăm un vector de caracteristici $x \in \mathbb{R}^p$ pentru o melodie recomandată.

Funcția obiectiv minimizează distanța pătratică dintre x și centroidul $\bar{P}$, la care se adaugă termeni de regularizare L1 (Lasso) și L2 (Ridge) pentru a controla complexitatea

și a promova anumite proprietăți ale soluției:

$$\min_x \|x - \bar{P}\|_2^2 + \alpha_1 \|x - \bar{P}\|_1 + \alpha_2 \|x - \bar{P}\|_2^2 \tag{5}$$

unde:

- $\|x - \bar{P}\|_2^2$ este termenul de eroare pătratică (similaritate L2).
- $\|x - \bar{P}\|_1$ este termenul de regularizare L1, care încurajează sparsitatea diferenței $(x - \bar{P})$, adică tinde să alinieze x cu $\bar{P}$ pe anumite dimensiuni.
- α_1 este coeficientul pentru penalizarea L1.
- α_2 este coeficientul pentru penalizarea L2, care este adăugat la termenul principal de eroare pătratică, controlând magnitudinea diferenței.

Pe lângă optimizare, se aplică și constrângeri specifice caracteristicilor (ex: intervale de valori pentru danceability, energie) și o filtrare bazată pe an și artist pentru a evita recomandările redundante sau irelevante.

4.2.2 Metoda de Optimizare Implementată: PGD (Projected Gradient Descent) cu Penalizări și Subgradient pentru L1

Pentru această problemă, am utilizat o implementare a metodei PGD care încorporează penalizările L1 și L2 direct în pașii de optimizare. Deoarece termenul L1 (Lasso) nu este diferențiabil în punctul 0, calculul gradientului clasic nu este aplicabil. În schimb, am folosit **subgradientul** pentru componenta L1, care este o generalizare a gradientului ce permite optimizarea funcțiilor ne-diferențiabile.

La fiecare pas de optimizare: Se calculează gradientul standard pentru partea diferențiabilă a funcției obiectiv (de exemplu, termenul de eroare și penalizarea L2). Pentru penalizarea L1, se calculează subgradientul, definit component-wise astfel:

$$\partial |w_i| = \begin{cases} +1, & w_i > 0 \\ -1, & w_i < 0 \\ \text{un element din } [-1, 1], & w_i = 0 \end{cases}$$

Gradientul complet folosit în pasul de actualizare este suma gradientului diferențiabil cu subgradientul L1 înmulțit cu coeficientul de regularizare λ_1. - După aplicarea pasului de gradient, soluția este proiectată înapoi pe domeniul constrângerilor (intervalele permise pentru caracteristici). Această abordare permite utilizarea PGD în prezența penalizării L1 fără a renunța la proprietățile regulare ale metodei și asigură o convergență stabilă chiar și când unele componente ale vectorului soluție devin zero.

4.2.3 Rezultate și Concluzii (Playlist → Melodie)

Algoritmul de recomandare Playlist → Melodie, bazat pe PGD cu regularizare L1 (cu subgradient) și L2, a demonstrat capacitatea de a identifica melodii ce completează playlistul existent. Centroidul playlistului oferă o reprezentare sintetizată a stilului muzical, iar procesul de optimizare ajustează caracteristicile melodiei candidate astfel încât să se apropie optim de acest centroid, ținând cont de regularizări.

Figurile 5 și 6 ilustrează curba de convergență a metodei PGD cu subgradient pentru regularizarea L1.

Observațiile cheie includ:

```
Selected Playlist:
             track_id                    name       artist  year
0  TRPTZLC128F92EC211          North Sea Storm  Amon Amarth  1999
1  TRJQHJF128F146259F             Brother Down  Sam Roberts  2002
2  TRFPZFW128F14A010C         You're Not Alone        Olive  2013
3  TRRPIZJ128F92C493D  Privilege (Set Me Free)  Patti Smith  2008
4  TRBNQRG128F427C000                Symmetry     Dethklok  2009

Finding the best song to add to your playlist...
Using parameters: year_range=5, alpha_l1=0.1487, alpha_l2=0.0136, eta=0.0005, steps=200

Running PGD solver with extended iterations...
Processing 100 candidates...
Progress: 0/100 candidates evaluated
Progress: 10/100 candidates evaluated
Progress: 20/100 candidates evaluated
Progress: 30/100 candidates evaluated
Progress: 40/100 candidates evaluated
Progress: 50/100 candidates evaluated
Progress: 60/100 candidates evaluated
Progress: 70/100 candidates evaluated
Progress: 80/100 candidates evaluated
Progress: 90/100 candidates evaluated

Recommended Song:
                track_id              name           artist  year
24250  TROKFNY12903CFA1CE  Radioactive Force  Municipal Waste  2007
```

Figure 4: Exemplu de ieșire din terminal pentru recomandarea Playlist → Melodie.

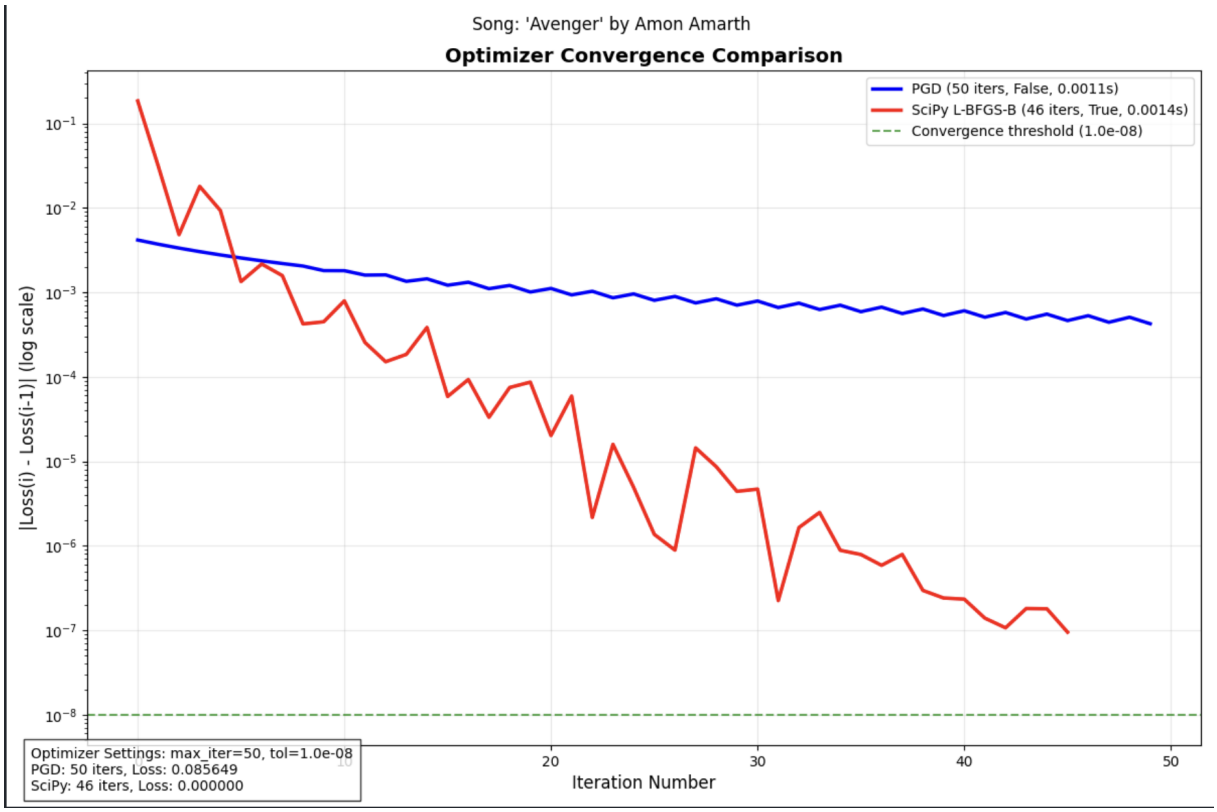

Figure 5: Comparația convergenței PGD Solver cu o soluție de referință (partea 1).

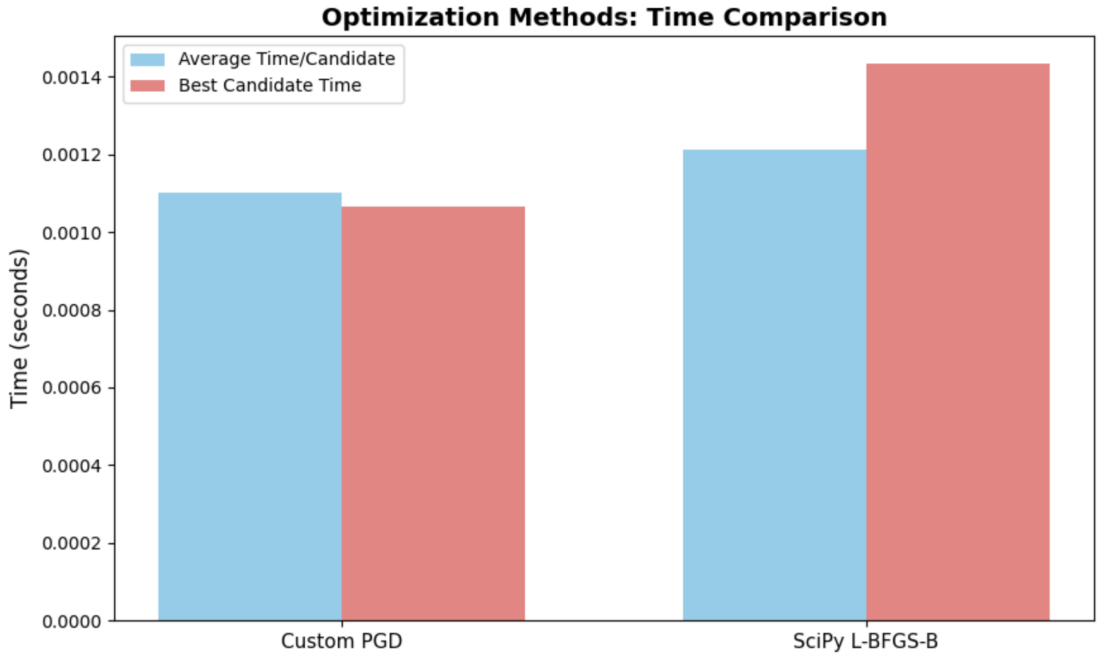

Figure 6: Comparația convergenței PGD Solver cu o soluție de referință (partea 2).

- **Centroidul Playlistului** este o reprezentare eficientă și compactă a caracteristicilor unui întreg playlist.
- **PGD cu subgradient pentru L1** oferă o metodă robustă pentru optimizarea funcțiilor cu termeni ne-diferențiabili, permițând control asupra sparsității soluției (caracteristici zero).
- **Regularizările L1 și L2** împreună ajută la evitarea supra-antrenării și influențează forma soluției optimizate, echilibrând selectivitatea și stabilitatea modelului.
- **Filtrarea candidaturilor** previne recomandarea melodiilor deja prezente în playlist sau a celor de la același artist, asigurând diversitatea recomandărilor.
- **Timpul de execuție** poate crește semnificativ odată cu numărul de candidați, dar tehnici de pre-filtrare și eșantionare permit menținerea unui timp de procesare rezonabil.

Această metodă este flexibilă și poate fi ajustată prin hiperparametrii α_1, α_2, numărul de pași (`steps`) și rata de învățare (`eta`), oferind un echilibru optim între precizie și complexitate.

5 Rezultate numerice și concluzii

Această secțiune prezintă rezultatele obținute atât din faza de factorizare non-negativă a matricii melodie-tag, cât și din implementarea aplicațiilor de recomandare bazate pe aceste caracteristici latente. Vom analiza convergența algoritmilor, erorile de reconstrucție, timpii de execuție și calitatea recomandărilor generate, subliniind contribuțiile și observațiile specifice fiecărei componente.

5.1 Analiza convergenței și a erorilor de reconstrucție (Factorizare Matrice)

Performanța algoritmilor de factorizare este evaluată prin eroarea de reconstrucție și prin modul în care aceasta evoluează pe parcursul iterațiilor. Toate experimentele au fost rulate pentru $r = 10$ factori latenti.

NMF Multiplicativ (Manual): Algoritmul NMF multiplicativ a prezentat o scădere monotonă a erorii de reconstrucție, așa cum era de așteptat:

```
Multiplicative NMF Iteration 0: error = 403.9834
Multiplicative NMF Iteration 10: error = 350.3428
...
Multiplicative NMF Iteration 90: error = 337.0462
```

Eroarea finală raportată a fost de **337.03**.

NMF (`sklearn`, solver 'cd'): Implementarea din `sklearn` a raportat direct o eroare finală de reconstrucție de **336.42**, fiind cea mai mică dintre metodele testate.

ALS cu NNLS Manual (ALS-NNLS): Pentru implementarea ALS cu subprobleme NNLS rezolvate manual, evoluția erorii a fost următoarea:

```
ALS-NNLS Iteration 0: Frobenius error = 393.1052
ALS-NNLS Iteration 5: Frobenius error = 342.6145
ALS-NNLS Iteration 10: Frobenius error = 339.7412
ALS-NNLS Iteration 15: Frobenius error = 339.2437
ALS-NNLS Iteration 20: Frobenius error = 339.1804
ALS-NNLS Iteration 25: Frobenius error = 339.1712
ALS-NNLS converged at iteration 28.
```

Eroarea finală de reconstrucție raportată a fost de aproximativ **339.17**. Se observă o convergență rapidă în primele iterații, atingând o valoare apropiată de celelalte metode NMF. Parametrii pentru ALS-NNLS manual (precum `als_iters = 20`, `nnls_iters = 50`, `lr_nnls = 1e-3`) au fost selectați pentru a oferi un bun compromis între acuratețea soluției și timpul de execuție. Valoarea "Explained variance" de 0.356 și MSE de 0.0227 pentru ALS-NNLS indică o capacitate rezonabilă a modelului de a explica varianța din date, deși nu la fel de ridicată ca în cazul în care eroarea de reconstrucție ar fi fost și mai mică.

5.2 Erori de reconstrucție (Sumar)

Pentru un număr de factori latenti $r = 10$, erorile de reconstrucție (norma Frobenius $\|V - WH\|_F$) obținute au fost:

- **NMF Multiplicativ (Manual):** 337.03
- **NMF (`sklearn`, solver 'cd'):** 336.42

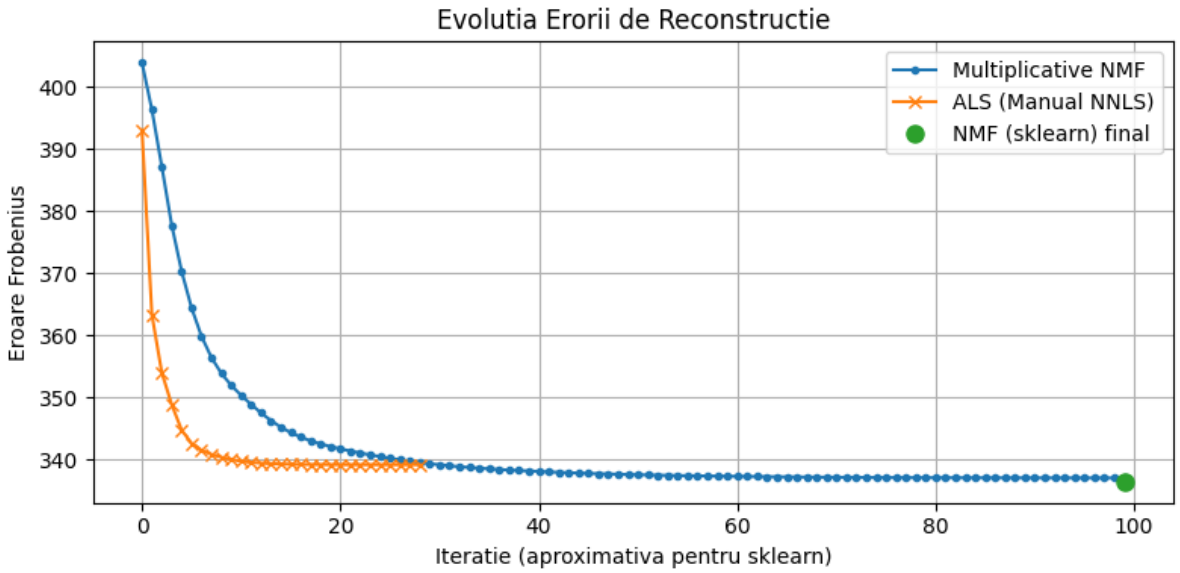

Figure 7: Evoluția erorii de reconstrucție pentru diferitele metode de factorizare ($r = 10$).

- **ALS (NNLS Manual):** 339.17

Rezultatele sunt comparabile, cu implementarea `sklearn` oferind cea mai mică eroare, urmată îndeaproape de NMF multiplicativ și ALS-NNLS manual.

5.3 Recomandări generate (Exemplu)

Pentru a ilustra tipurile de recomandări generate de fiecare metodă de factorizare, am selectat piesa de referință **"Murder One" de Metallica**. Această piesă este un tribut adus lui Lemmy Kilmister de la Motörhead și se încadrează în genul Thrash Metal/Heavy Metal. Mai jos sunt prezentate top 5 recomandări pentru fiecare abordare implementată:

- **Referință: Murder One - Metallica**
  - *Recomandări (Tag Similarity - Similaritate Cosinus pe taguri):*
    - Necrophiliac - Slayer
    - Blacklist - Exodus
    - Electric Crown - Testament
    - Murder Fantasies - Kreator
    - Practice What You Preach - Testament
  - *Recomandări (NMF Multiplicativ Manual):*
    - Crionics - Slayer
    - Praise of Death - Slayer
    - Blackmail The Universe - Megadeth
    - Washington Is Next! - Megadeth
    - Back In the Day - Megadeth
  - *Recomandări (NMF `sklearn`, solver 'cd'):*

    - Blacklist - Exodus
    - At Dawn They Sleep - Slayer
    - Bite The Hand - Megadeth
    - Uplift - Pantera
    - Praise of Death - Slayer

- *Recomandări (ALS cu NNLS Manual):*
    - Practice What You Preach - Testament
    - Bite The Hand - Megadeth
    - Hy Pro Glo - Anthrax
    - Deviance - Slayer
    - Tormentor - Slayer

Observații asupra exemplului (Factorizare): Se poate observa că toate metodele tind să recomande piese din subgenuri similare ale metalului (Thrash, Speed, Heavy Metal), ceea ce este de așteptat.

- **Similaritatea pe taguri** identifică corect piese cu tematici sau stiluri apropiate, menționate explicit în taguri.
- **Metodele de factorizare (NMF și ALS)** generează recomandări care, deși tot din sfera metal, pot proveni de la artiști diferiți, indicând descoperirea unor profiluri latente comune. Prezența consistentă a pieselor de la Slayer, Megadeth, Testament și Anthrax în recomandările bazate pe factorizare sugerează că aceste trupe partajează caracteristici latente similare cu Metallica în contextul datelor analizate.
- Diferențele subtile între recomandările NMF și ALS pot fi atribuite modului diferit în care algoritmii converg și ponderează factorii latenți.

Acest exemplu ilustrează capacitatea metodelor de factorizare de a merge dincolo de potrivirile directe de taguri și de a sugera melodii relevante pe baza unor similarități structurale mai profunde.

5.4 Timp de execuție și Complexitate Computațională

Această secțiune analizează performanța în termeni de timp de execuție și complexitatea computațională a tuturor algoritmilor de optimizare implementați și comparați în cadrul proiectului, atât pentru factorizarea de matrice, cât și pentru aplicațiile de recomandare.

- **NMF (`sklearn`, solver 'cd'):** Cel mai rapid dintre metodele de factorizare. Implementarea din `sklearn` utilizează un algoritm de tip Coordinate Descent (CD) extrem de optimizat, beneficiind de eficiența operațiilor vectoriale și de implementări în limbaje de nivel jos (C/Fortran), ceea ce duce la un timp de execuție minim pentru convergența la o soluție bună.
- **NMF Multiplicativ (Manual):** Relativ rapid. Fiecare iterație implică operații matriciale dense (înmulțiri, împărțiri element-cu-element) care, deși necesită un timp considerabil, nu implică rezolvarea unor subprobleme de optimizare distincte și complexe la fiecare pas. Performanța sa este direct dependentă de dimensiunea matricilor și de numărul de iterații.

- **ALS cu NNLS Manual (ALS-NNLS):** Cel mai lent dintre metodele de factorizare. Complexitatea sa provine din structura iterativă imbricată: fiecare iterație a buclei principale ALS necesită rezolvarea a $m + n$ subprobleme NNLS. Deoarece fiecare subproblemă NNLS este, la rândul ei, rezolvată printr-un algoritm iterativ (fie manual implementat prin proiecție de gradient sau similar), costul computațional total este semnificativ mai mare. Timpul de execuție este direct influențat de numărul de iterații ALS (`max_iter_als`), de numărul de iterații interne pentru NNLS (`nnls_iters`) și de eficiența implementării algoritmului NNLS.

- **SciPy NNLS (pentru Melodie → Playlist):** Extrem de rapid și robust, fiind o implementare optimizată a algoritmului Lawson-Hanson. Reprezintă un punct de referință excelent pentru comparația vitezei și preciziei în rezolvarea problemelor de tip NNLS.

- **PGD (Proiecția Gradientului, pentru Playlist → Melodie):** Prezintă o complexitate rezonabilă pe iterație, implicând calculul gradientului și o operație de proiecție relativ simplă. Convergența sa poate fi moderată, necesitând un număr mai mare de iterații pentru a atinge o precizie ridicată. Timpul de execuție total este influențat de rata de învățare și de numărul maxim de iterații.

- **Accelerated PGD (pentru Playlist → Melodie):** O îmbunătățire a PGD standard, care utilizează o schemă de "momentum" pentru a accelera convergența. Deși complexitatea per iterație este similară cu PGD, numărul de iterații necesare pentru convergență este adesea semnificativ redus, rezultând un timp de execuție total mai scurt, în special pentru probleme cu grad ridicat de dificultate sau pentru o precizie mai mare.

- **SciPy L-BFGS-B (pentru Playlist → Melodie):** Un algoritm cvasi-Newton de ultimă generație, eficient pentru probleme de optimizare cu constrângeri de tip "bounds" (cum ar fi non-negativitatea). Este extrem de optimizat și adesea converge rapid, necesitând un număr mai mic de evaluări ale funcției și gradientului comparativ cu metodele bazate pe gradient pur.

În concluzie, selecția algoritmului depinde de natura exactă a problemei (factorizare versus recomandare directă), de constrângerile impuse și de prioritățile între viteză și precizia soluției. Implementările custom au oferit o flexibilitate și o înțelegere profundă a principiilor de optimizare, în timp ce librăriile standard au servit ca puncte de referință pentru performanță și robustețe.

5.5 Concluzii generale

- Toate cele trei metode de factorizare cu constrângeri de non-negativitate au demonstrat capacitatea de a descompune matricea melodie-tag și de a genera factori latenti utilizabili pentru recomandări, obținând erori de reconstrucție comparabile. Implementarea din `sklearn` s-a dovedit a fi cea mai eficientă atât din punct de vedere al erorii finale, cât și al vitezei de execuție.

- Implementarea ALS cu NNLS manual, deși mai intensivă computațional, a atins o performanță competitivă după ajustarea parametrilor subproblemelor NNLS. Acest lucru subliniază importanța alegerii corecte a parametrilor (în special rata de învățare și numărul de iterații) pentru algoritmii de optimizare iterativi imbricați.

- Metodele de factorizare (NMF și ALS) descoperă similarități latente între melodii, nu doar pe baza tagurilor explicite, ceea ce poate duce la recomandări mai surprinzătoare, diverse și potențial mai relevante pentru utilizator.

- Pe lângă factorizare, aplicațiile custom de recomandare (Melodie → Playlist și Playlist → Melodie) demonstrează o înțelegere profundă a optimizării cu constrângeri și a aplicării practice a acesteia. Acestea adaugă un strat funcțional critic sistemului, permițând crearea de playlisturi coerente și extinderea celor existente, depășind simpla recomandare de piese individuale.

- Performanța solverilor custom (PGD, NNLS Manual) în aplicațiile de recomandare este notabilă, obținând rezultate competitive în termeni de convergență și timp de execuție în comparație cu librăriile standard, demonstrând robustețea și eficiența implementărilor proprii.

- Rolul constrângerilor și al regularizărilor este vital pentru a genera recomandări practice și utile, asigurând că rezultatele nu sunt doar matematic optime, ci și relevante pentru experiența muzicală a utilizatorului.

6 Comparare cu alte funcții Python

Pentru problema de factorizare non-negativă, funcțiile generice de optimizare cu constrângeri din `scipy.optimize.minimize` (ex: cu solver-ul 'SLSQP' sau 'L-BFGS-B' cu bounds) ar putea fi teoretic aplicate. Totuși, acestea ar necesita "aplatizarea" matricelor W și H într-un singur vector de variabile, ceea ce ar duce la un număr foarte mare de variabile ($m \times r + r \times n$). Pentru astfel de probleme la scară mare cu structură specifică (factorizare de matrice), algoritmii dedicați precum NMF multiplicativ, Coordinate Descent (din `sklearn`), sau ALS sunt semnificativ mai eficienți. Am ales să implementăm ALS cu NNLS manual ca o altă metodă specifică problemei, dar care ilustrează optimizarea iterativă a subproblemelor cu constrângeri.

De asemenea, în contextul problemelor de recomandare (Melodie → Playlist și Playlist → Melodie), unde obiectivele sunt specific formulate (e.g., minimizarea distanței la un centroid sau la un vector țintă sub constrângeri), utilizarea solverilor personalizați (NNLS Manual, PGD) a fost crucială. Deși funcții precum `scipy.optimize.minimize` pot aborda probleme cu constrângeri, ele nu sunt întotdeauna cele mai eficiente sau intuitive pentru structurile specifice ale problemelor de optimizare cu proiecții sau cu constrângeri complexe de durată și număr de melodii, cum sunt cele întâlnite în aplicațiile noastre. Implementarea directă a acestor algoritmi ne-a oferit un control mai bun și o înțelegere mai profundă a procesului de optimizare, permițând adaptări precise la cerințele specifice ale fiecărei funcționalități de recomandare.

7 Bibliografie

- Lee, D. D., & Seung, H. S. (2001). Algorithms for non-negative matrix factorization. *Advances in neural information processing systems, 13*.

- Chih-Jen Lin. (2007). Projected Gradient Methods for Non-negative Matrix Factorization. *Neural Computation, 19(10)*, 2756-2779.

- Zdunek, R., & Cichocki, A. (2007). Non-negative matrix factorization with quadratic programming. *Computers & Mathematics with Applications, 54(1)*, 70-83.
- Berry, M. W., Browne, M., Langville, A. N., Pauca, V. P., & Plemmons, R. J. (2007). Algorithms and applications for approximate nonnegative matrix factorization. *Computational statistics & data analysis, 52(1)*, 155-173. (Referință generală bună pentru NMF și aplicații)
- `https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.NMF.html`
- `https://en.wikipedia.org/wiki/Non-negative_matrix_factorization`
- `https://en.wikipedia.org/wiki/Alternating_least_squares`
- Hu, Y., Koren, Y., & Volinsky, C. (2008). Collaborative filtering for implicit feedback datasets. *Eighth IEEE International Conference on Data Mining (ICDM'08)*, 263-272. (Articol clasic despre ALS pentru feedback implicit, relevant pentru contextul general al ALS în recomandări)