Final Project

Music Recommendation Engine

Northeastern University: College of Professional Studies

EAI6010: Applications of Artificial Intelligence

Professor: Sergiy Shevchenko

Team: Husky 1

Ayush Anand

Anjal Mohammed

Mohit

Saurav Dibakar

Shashikant Nallan chakravartula

Sangayya Hiremath

June 24th, 2025

Introduction

We address the problem of recommending "similar-sounding" songs in the absence of any user listening history. Our solution is a **content‑based** approach that represents each track by its audio features, tempo, danceability, valence, and so on, standardizes those features, and then employs a cosine-distance k-Nearest Neighbors model to find the closest tracks in feature space. The model is exposed via a **FastAPI** service so clients can retrieve recommendations by supplying one or more track IDs.

Background

Recommender systems typically employ **collaborative filtering** (based on user–item interactions), **content-based filtering** (based on item attributes), or **hybrid** methods combining both. Content-based approaches are particularly suited when user data is absent or sparse, since they leverage only item properties to make recommendations.

Data & Preprocessing

- **Dataset:** SpotifyFeatures.csv (≈232,725 tracks, 18 columns)
- **Cleaning & Encoding:**
    - Empty track_name values (< 0.1% of rows) → "N/A"
    - Categorical fields (key, mode, time_signature) → numeric codes
- **Feature Set:** 14 numeric features (13 audio metrics + popularity)
- **Standardization:** Applied scikit-learn's StandardScaler to each feature, ensuring zero mean and unit variance so no single feature dominates the cosine-distance metric.

Feature	Range	Description
acousticness	0 – 1	Confidence, the track is acoustic
danceability	0 – 1	Suitability for dancing
energy	0 – 1	Perceived intensity and activity
valence	0 – 1	Musical "positiveness"
tempo	bpm	Beats per minute
popularity	0 – 100	Spotify popularity score

Table 1. Selected Audio Features

Model Design & Implementation

Feature Scaling:

We fit a StandardScaler on the 14-dimensional feature matrix.

k-NN Indexing:

Initialized NearestNeighbors(n_neighbors=10, metric='cosine') and fit it on the scaled data. This builds an efficient in-memory index for nearest-neighbor queries.

Recommendation Logic:

- **Input:** one or more Spotify track IDs and a desired top_n.
- **Process:**
  1. Map each track ID to its scaled feature vector.
  2. Query the k-NN index for n_neighbors = top_n + buffer to allow excluding the query tracks.
  3. Convert distances to similarity percentages via (1 – distance) × 100.

4. Return the highest-scoring top_n tracks, excluding any input IDs.

Hyperparameter Selection:

We experimented with k = 5, 10, and 15; **k = 10** offered the best mix of response diversity and relevance in our informal listening tests.

API & Demonstration

A **FastAPI** service wraps the model and exposes two endpoints:

- **GET /songs?page={page}&limit={limit}**

  Returns a paginated catalog of all tracks.

- **POST /recommendations**

Request body:

json

```
{
  "track_ids": ["0xq4ZTcmwBfkPGo4RRKmMe", "61uyGDPJ06MkxJtHgPmuyO"],
  "top_n": 5
}
```

Response:

json

```
[
  {
    "track_id": "45OfR7ugJMgbFDuNOVpIq3",
    "track_name": "Party On The West Coast (feat. Snoop Dogg)",
```

```
    "artist_name": "Matoma",
    "genre": "Dance",
    "similarity": 97.8
  },
  // …
]
```


Sample Recommendations

For a sample query combining **"In the End"** by Linkin Park, the API returned:

Track ID	Track Name	Artist	Genre	Similarity
45OfR7ugJMgbFDuNOVpIq3	Party On The West Coast (ft. Snoop Dogg)	Matoma	Dance	97.8 %
0xq4ZTcmwBfkPGo4RRKmMe	Gotta Go	CHUNG HA	Pop	97.4 %
61uyGDPJ06MkxJtHgPmuyO	Company	Justin Bieber	Pop	96.1 %
0g5EKLgdKvNlln7TNqBByK	Middle	DJ Snake	Pop	95.4 %
4P5KoWXOxwuobLmHXLMobV	Come As You Are	Nirvana	Rock	95.3 %

Table 2. Sample Recommendations (top 5)

What Worked

- **Data Pipeline:** Loading, cleaning, and encoding using pandas proceeded without issues.
- **Scaling:** StandardScaler reliably balanced feature ranges.
- **k-NN Queries:** sub-100 ms response times for single-track requests.

- **API Integration:** FastAPI's automatic Swagger UI allowed instant endpoint testing.
- **Reproducibility:** Persisted artifacts (knn_model.pkl, scaler.pkl, songs.csv) enable others to reload the exact environment.

What Didn't & Why

- **Scalability:** Brute-force k-NN on 230 K+ tracks will slow significantly at production scale; approximate methods (e.g., FAISS) were not integrated.
- **Lack of Personalization:** Recommendations remain the same for every user given the same track ID.
- **Popularity Bias:** Including popularity equally with audio features sometimes surfaced globally popular but sonically dissimilar tracks.
- **Edge Cases:** Very short or niche-genre tracks occasionally produced less coherent neighbors.

Future Work

- **Approximate Nearest Neighbors:** Integrate libraries like FAISS or Annoy for sublinear-time lookups.
- **Hybrid Filtering:** Combine audio-based similarity with collaborative signals (user–item interactions) to personalize recommendations.
- **Feature Enrichment:** Incorporate learned embeddings (autoencoders, Siamese networks) or lyric-based NLP features.
- **Genre Pre-Filtering:** Allow optional constraints to recommend only within the same genre or mood profile.

Reproduction Steps

1. **Clone** the repository.
2. **Install** dependencies:

   pip install -r requirements.txt

3. **Run** the server:

   uvicorn app: app-- reload

4. **Explore** via http://localhost:8000/docs (Swagger UI):

   GET /songs?page=1&limit=10

   POST /recommendations with JSON { "track_ids": […], "top_n": 5 }

Conclusion

This project successfully demonstrates how a straightforward content-based approach can power music recommendations by leveraging descriptive audio features. By transforming raw song metadata into standardized vectors and applying cosine-distance k-NN, we achieved fast, interpretable "percent-similar" outputs that align well with human intuition. The use of FastAPI to expose the model ensures easy integration and immediate feedback via a simple REST interface.

While the proof of concept meets our initial goals, delivering meaningful recommendations without user data, it also highlights areas for growth, such as scalability and personalization. The groundwork laid here sets the stage for integrating approximate nearest-neighbor libraries, hybrid filtering strategies, and enriched feature embeddings in future iterations.

Overall, this engine underlines the value of content-based methods in recommendation tasks, offering transparent logic and reproducible pipelines. It provides a robust framework that teams can build upon to develop more sophisticated, user-centric music discovery experiences.

References

Recommender system. (n.d.). In *Wikipedia*. Retrieved June 22, 2025, from https://en.wikipedia.org/wiki/Recommender_system

Audio Features API – Spotify for Developers. (n.d.). Retrieved June 22, 2025, from https://developer.spotify.com/documentation/web-api/reference/#endpoint-get-audio-features

NearestNeighbors — scikit-learn 1.7.0 documentation. (n.d.). Retrieved June 22, 2025, from https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.NearestNeighbors.html

FastAPI. (n.d.). Retrieved June 22, 2025, from https://fastapi.tiangolo.com/