Tag-Based Music Recommender

Hill Zhang

December 13, 2025

GitHub repository: `https://github.com/HillZhang2004/tag-based-music-recommender/tree/main`

1 Introduction and problem setup

This report describes a small tag-based music recommender that models my Spotify taste using metadata from Spotify and Last.fm. The core idea is simple: tags encode high-level attributes (genre, mood, style), and songs with similar tags often share a similar "vibe." The goal is to use tags to estimate how likely I am to like a new track.

The task is framed as binary classification. Positive examples come from my Spotify listening history (medium-term top tracks and saved tracks). Negative examples are constructed from tracks whose tag sets do not overlap with my positive tag profile. A logistic regression classifier is trained on bag-of-tags features to predict the probability that a song is a "like" (label 1) versus a constructed "not like" (label 0).

At training time, the model trains on a labeled dataset `api_labeled_tracks.csv` that contains:

- `track_id`, `track_name`, `artist_name`,
- a comma-separated `tags` string,
- a binary `label` in $\{0, 1\}$.

At scoring time, candidate songs from `top_tracks_lastfm.csv` are ranked using the model's predicted probability of label 1. The model is exposed through a command-line scorer (`recommend_from_csv.py`), a small terminal demo (`interactive_demo.py`), and a Streamlit app (`streamlit_app.py`).

2 Data and experimental setup

2.1 Building the labeled dataset

Spotify data collection uses the Spotify Web API to retrieve my medium-term top tracks and saved tracks. For each track, the pipeline stores the Spotify ID, track name, artist name, and a simple weight (top tracks weight 2, saved-only tracks weight 1). All of these tracks are treated as positive examples (label 1).

Last.fm tags are collected via the Last.fm API. For each positive track, both track-level tags (`track.getInfo`) and artist-level tags (`artist.getTopTags`) are retrieved and combined. Tag names look like `pop`, `rnb`, `female vocalists`. A weighted counter over positive tags (weight 2 or 1) defines the "positive tag profile."

To expand coverage, additional tags are collected by looping over frequent positive tags and calling `tag.getTopTracks`. Tags from those returned tracks are added to an expanded tag dictionary, stopping once the tag vocabulary reaches roughly 1500 unique tags. This step helps surface tags that are adjacent to my interests, not only the tags already present in my listening history.

Negative examples are constructed from tags that never appear in the positive tag set. For each such tag, top tracks are fetched from Last.fm. Any candidate whose (track name, artist) matches an existing positive is skipped. Candidates with empty tags are dropped. Candidates that share any tag with the positive tag set are also dropped. The remaining tracks, whose tags live entirely outside the positive tag space, are labeled 0.

After cleaning (dropping rows with missing labels or empty tag strings), the final labeled dataset contains 529 tracks:

- 329 positives (label 1),
- 200 negatives (label 0).

The `tags` column stores a comma-separated string like `pop,rnb,female vocalists,soul`.

2.2 Features and model

Each song is represented with a bag-of-tags vector. A tokenizer cleans the tag string by removing list-style punctuation, splitting on commas, trimming whitespace, and dropping empty entries. This tokenizer is passed into scikit-learn's `CountVectorizer` with `binary=True`, so each feature indicates the presence or absence of a tag.

On the cleaned dataset, this produces a sparse feature matrix X with shape 529×1123, where each column corresponds to a tag and each row corresponds to a track. The target vector y contains the 0/1 labels.

The classifier is scikit-learn `LogisticRegression` with the `liblinear` solver. Regularization is controlled by the parameter C.

2.3 Training and evaluation protocol

The dataset is split into train, dev, and test using a 60/20/20 stratified split. Concretely, `train_test_split` is applied twice to produce:

- 317 training examples,
- 106 dev examples,
- 106 test examples.

On the dev set, C is tuned over $\{0.01, 0.1, 1.0, 10.0\}$. For each C, the model is trained on the training split and evaluated on dev using F1. In this setup, the best dev F1 is 1.00 at $C = 0.01$.

Using the tuned value of C, performance is reported on the held-out test set. A final model is then refit on all 529 labeled songs and saved as `tag_model.joblib`, along with the fitted `CountVectorizer` saved as `tag_vectorizer.joblib`. These saved artifacts are used by the CLI scorer and the demos.

3 Results

On the held-out test set of 106 tracks, the tuned logistic regression model with $C = 0.01$ achieves an accuracy of 0.991. Detailed test metrics are:

- Label 0: precision $= 0.976$, recall $= 1.000$, F1 $= 0.988$.
- Label 1: precision $= 1.000$, recall $= 0.985$, F1 $= 0.992$.

These very high scores are expected given how negatives are constructed. By design, negative songs have tag sets with zero overlap with the positive tag set, making the classes close to linearly separable in tag space.

A separate sanity check evaluates behavior on candidate songs. Tracks from `top_tracks_lastfm.csv` are scored and ranked. High-scoring candidates tend to carry tags like `rnb`, `soul`, `female vocalists`, and mainstream `pop`, which matches the intended preference profile and makes the ranking behavior easier to interpret.

4 Limitations and future work

This tag-based approach separates constructed positives and negatives reliably, and the qualitative ranking behavior aligns with the intended taste profile. However, the 0.99-level metrics should not be over-interpreted. The main reason performance is so high is the negative construction procedure: negatives are deliberately far from positives (disjoint tag space), which makes the classification problem easier than realistic recommendation settings where many items are borderline.

Several extensions would make the setup more realistic:

- **Harder negatives.** Include borderline items (for example: tracks with partial tag overlap, or tracks that are similar in tags but not saved).
- **Richer features.** Add simple audio features (tempo, energy, loudness) or recency signals on top of tags.
- **Generalization beyond one profile.** Extend to multiple user profiles or learn shared representations with lightweight personalization.

Even with these limitations, the project provides a compact end-to-end pipeline for API data collection, cleaning, feature construction, model training, evaluation, and deployment into small scripts and demos.