A Multi-Language Content-Based Music Recommendation System Using Spotify Audio Features

Vaibhav Garg, Saher Dev, Swayam, Arnav Kataria, Tanisha Sonkar

Indian Institute of Technology Jodhpur

{b23cm1046, b23cs1059, b23ee1073, b23ee1083, b23ee1074}@iitj.ac.in

Abstract

This paper presents a content-based multilingual song recommendation system developed using unsupervised machine learning techniques.Designed to support five languages—Hindi ,Tamil ,Korean, English ,and a miscellaneous type—the system takes a user's input of a language and a specific song title, and identifies similar songs within that language to ensure linguistic and cultural relevance. To build the recommendation engine, we curated a dataset of 61,711 songs across the supported languages using Kaggle as our source. Audio features such as tempo, valence, energy, acousticness, and danceability were extracted and standardised. We then applied Kernel Principal Component Analysis (Kernel PCA) with an RBF kernel to reduce dimensionality while preserving non-linear feature relationships.

The reduced feature set was clustered using K-Means, which produced more distinct and meaningful groupings than DBSCAN. K-Means was fine-tuned by using K-Means++ for centroid initialization and the Elbow Method by analyzing the Within-Cluster Sum of Squares (WCSS) to determine the optimal number of clusters per language, while the best epsilon values were used for DBSCAN for comparison.Once clusters were formed, cosine similarity was employed within the relevant cluster to retrieve songs most similar to the user's input track. This hybrid approach ensures that recommendations are both musically coherent and computationally efficient. The system consistently delivered results aligned with user preferences in mood, genre, and rhythm. Future improvements may include integrating lyrics-based features, deep learning models for better embeddings, and user feedback mechanisms to enhance personalization further.

1. Introduction

Music plays an important role in the daily lives of students, serving as a companion during study sessions and leisure activities. Identifying songs that match personal interests can be time-consuming and inefficient on streaming platforms with millions of tracks available. This project aims to simplify the music discovery process by employing fundamental machine learning techniques to produce personalized song recommendations. Through the analysis of a user-specified track's acoustics and various musical elements, our algorithm finds and recommends seven songs that are highly compatible with the user's taste in music, improving their listening experience with no effort.

2. Dataset

2.1. About the Dataset

The dataset for this project is sourced from ***Kaggle's Spotify Tracks Dataset*** , which includes details such as artist names, release years, and various audio features used to train the model.

2.2. Data Preprocessing

To prepare the data for model training and clustering, the following preprocessing steps were applied:

- Removed columns other than audio features that were not required for model training.
- Re-indexed the data after removing languages such as Telugu and Malayalam that did not contribute significantly to ensure easy access.
- Handled duplicates and missing values by dropping them to maintain data consistency.
- Categorical features (`key`, `time_signature`) were processed using OneHotEncoder to convert them into binary vectors to make sure model does not capture any irrelevant relationship between these features.
- Standardized numerical features (`year`, `popularity`, `duration_ms`, `loudness`, `tempo`) using `StandardScaler` to achieve zero mean and unit variance to make sure one feature does not dominate others because of its magnitude.
- Left pass-through features (e.g., `acousticness`, `danceability`, `energy`, etc.) unchanged, as they are already normalized between 0 and 1.
- Songs belonging to the same language were stored in separate files along with their audio features after scaling .

Table 1. Dataset Features and Their Incorporation in Predictive Modeling

Feature	**Feature Description**	**DataType**	**Incorporation in Predictive Modeling**
track id	Spotify's unique identifier for the track	object	No
track name	Name of the track	object	No
artist name	Names of the artists who performed the track	object	No
year	Year of release	int64	Yes
popularity	Popularity score of the track	int64	Yes
artwork url	URL of the album or track's artwork	object	No
album name	Name of the album	object	No
acousticness	Confidence measure of whether the track is acoustic (0.0 to 1.0)	float64	Yes
danceability	Suitability for dancing (0.0 = least, 1.0 = most)	float64	Yes
duration ms	Duration of the track in milliseconds	float64	Yes
energy	Intensity and activity level (0.0 to 1.0)	float64	Yes
instrumentalness	Likelihood of the track containing no vocals	float64	Yes
key	Musical key using standard pitch class notation (e.g., 0 = C)	float64	Yes
liveness	Likelihood the track was recorded live	float64	Yes
loudness	Overall loudness in decibels (dB)	float64	Yes
mode	Modality (1 = major, 0 = minor)	float64	Yes
speechiness	Presence of spoken words (closer to 1.0 = more speech)	float64	Yes
tempo	Speed in beats per minute (BPM)	float64	Yes
time signature	Beats per measure (typically 3 to 7)	float64	Yes
valence	Positivity or musical happiness of the track	float64	Yes
track url	Spotify URL link to the track	object	No
language	Language of the lyrics (English, Tamil, Hindi, Telugu, Korean)	object	Yes

2.3. Dataset Features

Each song in the dataset contains audio features that are essential for training clustering algorithms. The key features include the above.

3. Dimensionality Reduction

To reduce the dimensionality of our dataset in order to visualize clusters and understand data distribution, we applied both **Principal Component Analysis (PCA)** and **Kernel PCA**. Below, we compare the results and their effectiveness:

3.1. Principal Component Analysis (PCA)

PCA is a linear dimensionality reduction technique that projects data along directions (principal components) that maximize variance. It assumes that data lies on a linear subspace, which may not always be true in real-world high-dimensional datasets like music features where data distribution is non linear.

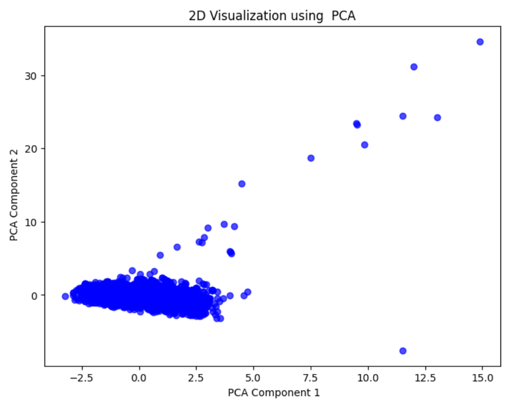

Figure 1. PCA visualization of audio features. The clumping near origin shows limited separation due to linearity.

As shown in the PCA visualization, data points have dense distribution near origin and variance along one component is significantly higher than other one.This indicates that while PCA captures some variance, it might not be fully effective in separating complex patterns in the data due to its non linear nature.

3.2. Kernel PCA with RBF Kernel

Kernel PCA extends PCA by using kernel functions (in our case, the RBF kernel) to map the data into a higher-dimensional space where it may become linearly separable. This allows Kernel PCA to capture **non-linear** relationships between features, which are common in audio data.

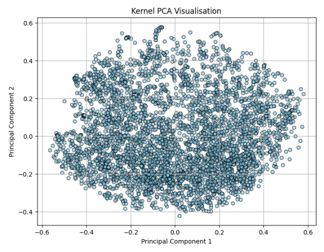

Figure 2. Kernel PCA visualization of audio features. Figure shows that data distribution is dense and evenly spread out.

The Kernel PCA plot shows a much more even and roughly a circular spread where along the two components variance is comparable suggesting that the technique captures the non-linear relationship of audio features better in two dimensions. This structure is likely to enhance the performance of clustering algorithms applied afterward.

3.3. Comparison

In conclusion, although **PCA** is simpler and computationally faster, **Kernel PCA** proved to be more effective for our dataset due to its ability to uncover complex patterns in non-linear data and giving us a more dense and uniformly spread distribution, making it a more suitable and effective choice for clustering techniques.

4. Clustering Approaches

We tested two clustering algorithms that come under Unsupervised Learning: **DBSCAN** and **K-Means**.

4.1. K-Means Clustering

K-Means was applied on the Kernel PCA-reduced data.

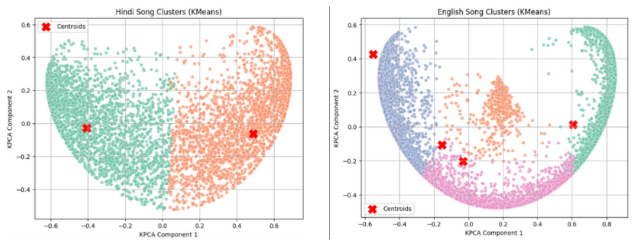

Figure 3. Kmeans with KPCA reduced data

We realized that reducing dimensionality might lead to loss of important characteristics required to determine similar songs, Hence model was applied on scaled models only. The results for PCA were visualized using a 2D scatter plot, showing well-separated groupings of songs.We implemented K-Means from scratch, with automatic k selection for each language subset of the dataset.

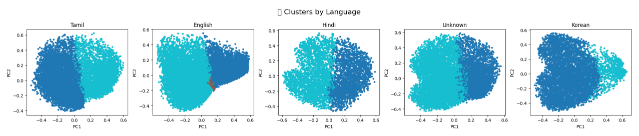

Figure 4. K-Means visualization of clusters by language.Each subplot shows how songs are grouped in the feature space for a specific language,indicating effective separation.

4.1.1 K-Means++ Initialization

To avoid poor clustering due to random initialization, we employed the *K-Means++* algorithm for centroid initialization.This technique selects the initial centroids randomly and all the subsequent centroids with a probability proportional to their squared distance from the nearest already chosen centroid. This significantly improves clustering quality compared to naive random initialization.

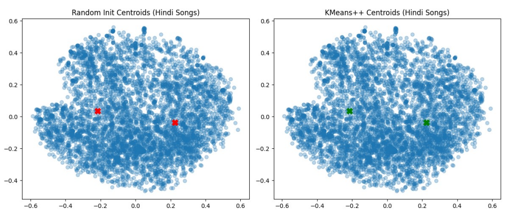

Figure 5. Random Vs KMeans++ Centroid initialisation

4.1.2 Custom K-Means Implementation

Our implementation iteratively assigns each data point to the nearest centroid using Euclidean distance. After assigning points, new centroids are computed as the mean of all points in each cluster. This process repeats until the centroids *converge* or a defined *maximum number* of iterations is reached. A small convergence threshold (tolerance) is used to determine early stopping when centroid movements become negligible. `(tol=1e-4)`

4.1.3 Selecting Optimal Number of Clusters

Rather than predefining the number of clusters, we automated the selection of k using the Elbow Method.For values of k ranging from 1 to a maximum(in our case 6), the algorithm computes the Within-Cluster Sum of Squares (WCSS), also known as inertia, which quantifies the compactness of clusters.

$$WCSS = \sum_{j=1}^{k} \sum_{x_i \in C_j} \|x_i - \mu_j\|^2 \tag{1}$$

We then identify the "elbow point" on the WCSS vs. k plot by finding the point where the decrease in WCSS starts to slow down noticeably which looks like an "elbow" in the curve.It represents the optimum k number of clusters because adding more clusters beyond this does not improve the grouping much.

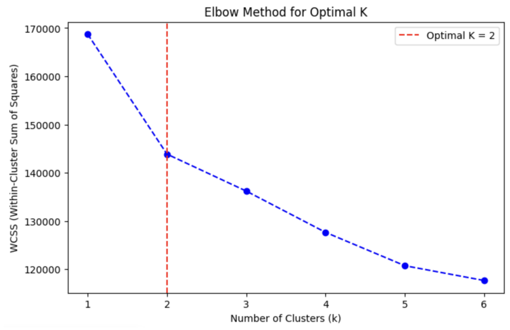

Figure 6. Elbow curve used to find optimal number of clusters for Hindi songs.

This procedure is repeated for each language-specific subset to derive language-specific k values, enhancing recommendation relevance within linguistic contexts.

4.1.4 Dunn Index for Cluster Validation

In addition to WCSS, we used the Dunn Index to validate cluster quality. The Dunn Index is defined as the ratio between the minimum intercluster distance and the maximum intracluster distance.

$$DunnIndex = \frac{\min_{1 \le i < j \le k} \delta(C_i, C_j)}{\max_{1 \le l \le k} \Delta(C_l)} \tag{2}$$

A higher Dunn Index suggests well-separated and compact clusters. We evaluated the Dunn Index over a range of k values (from 2 to 6) using our KMeans model.

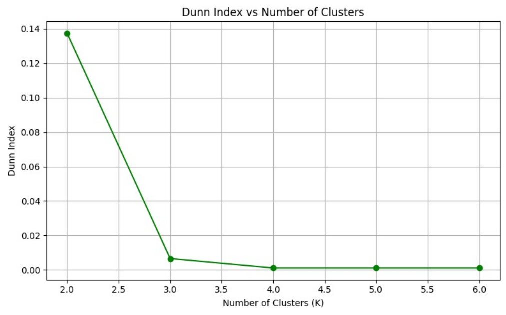

Figure 7. Dunn Index vs. Number of Clusters k for Hindi songs.

Though computationally expensive, the Dunn Index reinforced the elbow method's selection in most language datasets, validating our cluster count choices.

4.1.5 Language-Based Clustering Models

Finally, for each language (Hindi, Tamil, English, Korean, and Unknown), the scaled feature set was passed through the custom KMeans model with automatic k selection. The model stored the resulting centroids and cluster labels for use in the recommendation phase. These models were later used to find the input song's cluster and perform intra-cluster similarity comparisons.

- **Centroid Initialization:** KMeans++
- **Cluster Count Selection:** Elbow Method using WCSS
- **Convergence Criteria:** Tolerance threshold of 1×10^{-4}
- **Validation Metric:** Dunn Index

This K-Means clustering approach allowed the system to form coherent groupings of songs, serving as the backbone for meaningful, context-aware music recommendations.

4.2. DBSCAN Clustering

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) was also applied to the Kernel PCA-reduced dataset. It does not require the number of clusters to be specified beforehand and can identify outliers, but is sensitive to its hyperparameters — especially the 'epsilon' value.

To optimize DBSCAN, we used the Elbow Method to find an appropriate epsilon. However, DBSCAN struggled with the dataset. When applied to 2D Kernel PCA-reduced data, it failed to form meaningful clusters and mostly classified points as outliers.

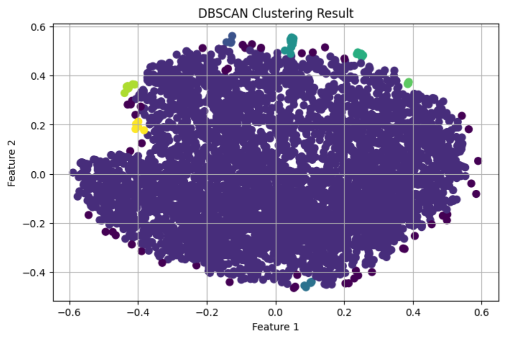

Figure 8. DBSCAN on 2D Kernel PCA-reduced data. Most of the points are labeled as outliers, with very few clusters formed.

We determined the optimal 'epsilon' by plotting the distances to the k-th nearest neighbor and locating the inflection point:

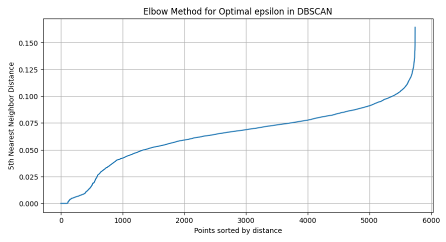

Figure 9. Elbow Method to determine optimal epsilon for DBSCAN. The ”knee” in the curve indicates the best choice.

Next, we applied DBSCAN on data reduced to 4 dimensions using Kernel PCA, expecting more separation power. While it formed some clusters, the results were still not as meaningful as K-Means.

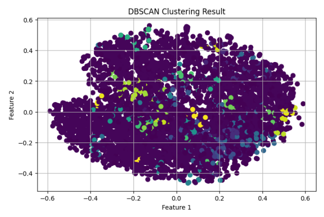

Figure 10. DBSCAN on 4D Kernel PCA-reduced data. Clusters formed are still sparse, and many points remain unclassified.

Another elbow plot was generated using the 4D-reduced data to find the optimal epsilon again:

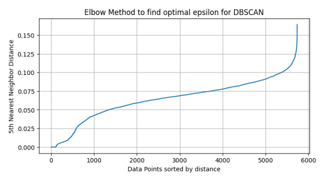

Figure 11. Optimal epsilon determination for 4D DBSCAN using Elbow Method.

Despite these efforts, DBSCAN did not yield meaningful clustering results for our dataset. The model often classified a majority of points as noise or formed just one or two clusters. Hence, we opted for K-Means, which performed significantly better in identifying meaningful groupings.

K-Means clearly outperformed DBSCAN in this context and was chosen for the final recommendation system.

5. Recommendation using Cosine Similarity

Once clusters were formed, cosine similarity was applied to find songs similar to a given input song within the same cluster. This method ensures that the recommendations are both contextually and sonically aligned with user preferences.

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \cdot \sqrt{\sum_{i=1}^{n} B_i^2}} \tag{3}$$

6. Why Cosine Similarity Over KNN

In music recommendation, **K-Nearest Neighbors (KNN)** with Euclidean distance is sensitive to magnitude-based differences in features like tempo or loudness, which may not align with perceptual similarity. **Cosine similarity**,measures the angle between vectors, capturing *directional similarity* in feature space. This makes it ideal for audio features such as energy, danceability, or valence, where *relative patterns* are more important than absolute values. Coupling **K-Means clustering** with **cosine similarity** allows for efficient, high-dimensional similarity matching within clusters, leading to more context-aware and musically coherent recommendations.

7. Song Recommender Function

The `SongRecommender` class is a custom recommendation system designed to suggest musically similar tracks based on an input song's audio features.

The system avoids recommending the input song itself (based on normalized name matching) and returns the top-n most similar tracks from the cluster. *For example, Kesariya and Kesariya from(”Brahmastra”) are the same exact songs.* Only the songs within the identified cluster are considered for recommendation using Cosine similarity is computed between the input vector and each song in the same cluster to rank them based on similarity. For each selected song, relevant metadata such as track name, artist, track URL, artwork URL, and language is retrieved using the original indices passed to the function.

8. Spotify API Integration and Access Limitations

The integration of the Spotify Web API was initially proposed to dynamically fetch metadata and audio features for a broader range of songs, with the aim of generalizing the recommendation system beyond the static dataset. Specifically, the approach intended to retrieve real-time information based on track names and artist identifiers by querying Spotify's endpoints for `audio-features` and `track` metadata.

However, during implementation,One major issue we faced is that although we are able to retrieve the track IDs successfully, repeated requests to the API, particularly for songs in Indian regional languages resulted in HTTP `403 Forbidden` errors. This status code, returned consistently despite valid authentication credentials, indicates that Spotify's licensing and content access policies explicitly restrict certain tracks, particularly independent and region-specific songs that are not globally licensed or fully indexed in Spotify's content database.

As a result, the system design was revised to operate solely on our dataset of pre-fetched tracks for which audio metadata was already available. This limitation confines the recommendation scope to a fixed set of songs but ensures reliable functionality without dependency on external API availability .

9. Results and Discussion

The use of Kernel PCA + K-Means provided clear cluster separation as shown in the figures. Within each cluster, cosine similarity was effective in finding similar songs, providing personalized recommendations based on a single input track.

The system's performance was evaluated qualitatively by testing different songs as input and comparing the recommended songs. The results consistently aligned well with musical features and genre preferences.

10. Future Work

While the current system performs well in identifying and recommending similar songs across multiple languages, there are several directions in which the system can be enhanced:

- **Deep learning for audio feature extraction:** Deep learning techniques like CNN can be used to extract audio features from audio files and these features are then passed to Clustering algorithms for song prediction.
- **Integration of user interaction data:** Incorporating user listening history will help us to give more personalized recommendations.
- **Genre classification:** Explicit genre labels and subgenre detection can guide recommendations and further improve cluster formation.

11. Conclusion

We successfully built a music recommendation engine tailored to multilingual songs using unsupervised learning. Kernel PCA followed by K-Means yielded distinct groupings, and cosine similarity refined recommendations within each group. Future work could include deep learning-based audio embeddings or user interaction data for even more personalized results.

Contributions

- **Saher Dev**: Implemented K-Means Clustering from scratch with K-Means++ for centroid initialization, Cosine Similarity logic, and the Song Recommender function. Contributed to the Report, Web Page, Spotlight Video, and Minutes of Meetings.
- **Vaibhav Garg**: Preprocessed the dataset, implemented dimensionality reduction techniques and DBSCAN clustering. Continuously evaluated model performance and contributed to the Report and Spotlight Video.
- **Tanisha Sonkar**: Designed the UI, implemented full stack frontend-backend integration using Flask, generated pickle files and handled user query logic via the Song Recommender Function. Contributed to the Presentation, Spotlight Video, and Web Demo.
- **Swayam**: Implemented K-Nearest Neighbors and DBSCAN algorithms from scratch. Contributed to the Presentation for the Spotlight Video.
- **Arnav Kataria**: Built and integrated the entire frontend. Contributed to PCA, Cosine Similarity algorithms from scratch, and Minutes of the Meetings.

Use of Google Cloud

Most Development such as clustering and recommendation logic, was done using **Google Colab**, which offered a collaborative and GPU-enabled environment. We intend to deploy our final web-based music recommendation system using **Google Cloud Platform (GCP)** via **Cloud Run** or **App Engine**, ensuring scalable & serverless hosting. Additionally, **Google Cloud Storage** may be used to store preprocessed datasets and model files. This integration supports modern deployment practices and aligns with the course's encouragement to leverage Google Cloud services.

References

[1] Scikit-learn Documentation. `https://scikit-learn.org/`

[2] Spotify Web API. *Official Developer Documentation.* `https://developer.spotify.com/documentation/web-api/`

[3] Vaibhav Garg, Saher Dev, Swayam, Arnav Kataria, Tanisha Sonkar. *PRML Project: Multilingual Music Recommendation System.* GitHub Repository. `https://github.com/philnumpy/PRML-PROJECT`

[4] Lance Galletti. *KMeans From Scratch.* Medium Article. `https://medium.com/@gallettilance/kmeans-from-scratch-24be6bee8021`

[5] IBM Cloud Education. *K-Means Clustering Explained.* IBM Think. `https://www.ibm.com/think/topics/k-means-clustering`