A Mini Project Report

On

“Music Recommendation System”

Submitted to the

Pune Institute of Computer Technology, Pune

In partial fulfillment for the award of the Degree of

Bachelor of Engineering

in

Information Technology

by

Tanish Charthankar	33116
Soham Kottawar	33147
Piyush Kinekar	33164
Tabish Ansari	33180

Under the guidance of

Prof. Mr. Ravindra Murumkar

Department of Information Technology

Pune Institute of Computer Technology College of Engineering

Sr. No 27, Pune-Satara Road, Dhankawadi, Pune - 411 043.

2024-2025

SCTR's PUNE INSTITUTE OF COMPUTER TECHNOLOGY
DEPARTMENT OF INFORMATION TECHNOLOGY

CERTIFICATE

This is to certify that the project report entitled

MUSIC RECOMMENDATION SYSTEM

Submitted by

Tanish Charthankar	33116
Soham Kottawar	33147
Piyush Kinekar	33164
Tabish Ansari	33180

is a bonafide word carried out by them under the supervision of Prof. Ravindra Murumkar and it is approved for the partial fulfillment of the requirement of **Laboratory Practice I** for the award of the Degree of Bachelor of Engineering (Information Technology)

Prof. Ravindra Murumkar	**Dr. A. S. Ghotkar**	**Dr. S. T. Gandhe**
Project Guide	HOD, IT	Principal
PICT, Pune	PICT, Pune	PICT, Pune

Place:
Date:

ACKNOWLEDGEMENT

We thank everyone who has helped and provided valuable suggestions for successfully developing a wonderful project. We are very grateful to our guide Prof. Ravindra Murumkar, Head of Department Dr. A. S. Ghotkar and our Principal Dr. S. T. Gandhe. They have been very supportive and have ensured that all facilities remained available for smooth progress of the project.

ABSTRACT

This project focuses on developing a music recommendation system using collaborative filtering techniques in machine learning. The primary aim is to provide users with personalized song recommendations based on their preferences, specifically by analyzing the features of songs in a given dataset. The system leverages various song attributes such as danceability, energy, and loudness to identify similar tracks. This report outlines the project lifecycle, including data acquisition, cleaning, exploration, modeling, and visualization of results, culminating in an effective recommendation engine.

CONTENTS

CERTIFICATE	2
ACKNOWLEDGMENT	3
ABSTRACT	4
CONTENTS	5
LIST OF TABLES & FIGURES	6
INTRODUCTION	7
LITERATURE SURVEY	8
SYSTEM ARCHITECTURE & DESIGN	10
EXPERIMENTATION & RESULTS	13
CONCLUSION & FUTURE SCOPE	17
REFERENCES	18
ANNEXURE	19

LIST OF TABLES

Table Number	Table Title	Page Number
3.2.1	Dataset Overview	11
4.6.1	Comparative Analysis	15

LIST OF FIGURES

Figure Number	Figure Title	Page Number
3.1.1	System Architecture	10
4.5.1	Recommendations Visualization	14

1. INTRODUCTION

1.1 PURPOSE, PROBLEM STATEMENT

The purpose of this project is to design and implement a music recommendation system that enhances the user experience by providing personalized music suggestions. The problem statement focuses on identifying user preferences and suggesting songs that align with those preferences.

1.2 SCOPE, OBJECTIVE

- **Scope:** The project explores collaborative filtering to recommend songs based on user input. The dataset contains various musical attributes that influence the similarity between songs.
- **Objective:** To develop a system that enables users to input their favorite song and receive recommendations for similar songs based on their features.

1.3 DEFINITIONS, ACRONYMS, AND ABBREVIATIONS

- **KNN:** K-Nearest Neighbors
- **Cosine Similarity:** A measure of similarity between two non-zero vectors.
- **DataFrame:** A data structure used in pandas to store and manipulate tabular data.

1.4 REFERENCES

- https://www.javatpoint.com/recommendation-system-machine-learning
- https://www.geeksforgeeks.org/recommendation-system-in-python/
- https://medium.com/@ashu19/music-recommendation-system-using-machine-learning-8bc8cc52d143

2. LITERATURE SURVEY

2.1 INTRODUCTION

Music recommendation systems are essential in today's digital era, where users face vast libraries of songs. These systems harness advanced machine learning and data mining techniques to personalize music choices based on a user's listening patterns and preferences. This survey explores key methodologies, particularly focusing on content-based filtering and clustering techniques used in music recommendation systems. The research aims to compare the effectiveness of these methods in enhancing user satisfaction and system accuracy, with a primary focus on studies from recent years that investigate different approaches to tackling recommendation challenges like data sparsity and cold start problems.

2.2 DETAILED LITERATURE SURVEY

- **Content-Based Filtering**: This approach analyzes song characteristics such as Danceability, Energy, and Valence to suggest similar tracks. It tailors recommendations by focusing on song features that match the user's historical preferences. Content-based filtering has shown effectiveness in scenarios where user data is limited but depends heavily on feature extraction techniques like Mel-spectrograms and Convolutional Neural Networks (CNN) for high precision.
- **Clustering Techniques (K-Means)**: K-means clustering segments songs based on their acoustic similarities, grouping them into clusters. While effective in pattern recognition, this method doesn't consider individual user preferences. Studies applying this technique on large datasets such as Spotify have revealed promising results, with high clustering efficiency, particularly when evaluated using the Silhouette Index to assess accuracy.
- **Personalization and Hybrid Models**: Personalizing recommendations remains a significant challenge. Hybrid models, which combine collaborative filtering and content-based methods, have been proposed to better understand user preferences by integrating both song attributes and user-item interactions. These models outperform traditional methods by balancing personalization with discovery of new music.

2.3 FINDINGS OF LITERATURE SURVEY

- The comparative studies suggest that while content-based filtering excels in personalizing music based on audio features, clustering techniques offer a broader perspective in recognizing patterns across large datasets. Hybrid approaches, integrating multiple techniques, are increasingly necessary for enhancing user experience by addressing limitations like cold starts and sparse data.

3. SYSTEM ARCHITECTURE AND DESIGN

3.1 DETAILED ARCHITECTURE

The architecture of the music recommendation system consists of data ingestion, processing, modeling, and output stages.

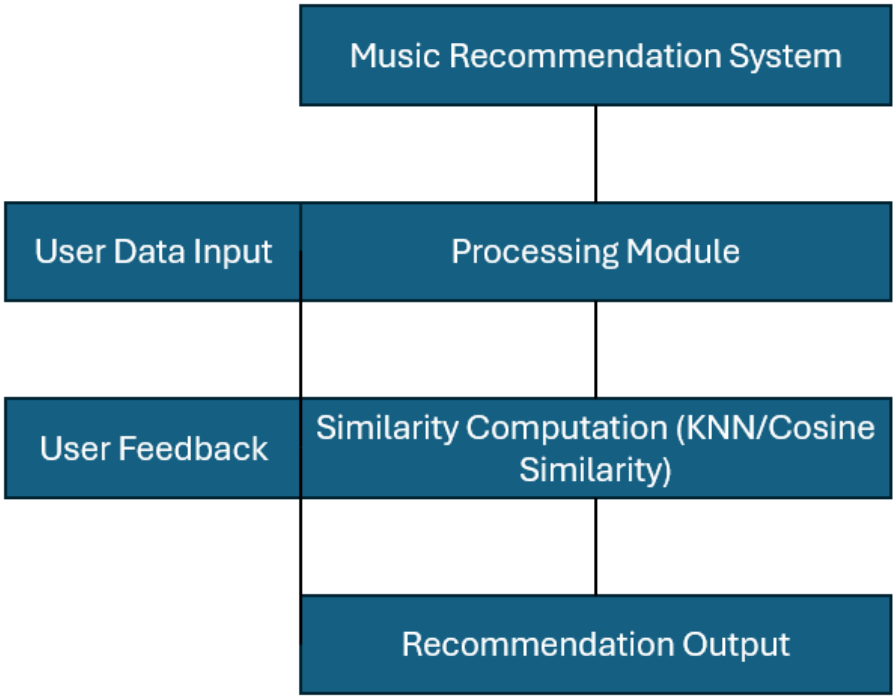

Figure 3.1.1 System Architecture

3.2 DATASET DESCRIPTION

The dataset used for this project includes song attributes such as:

- **track_id:** Unique identifier for the song
- **name:** Song title
- **artist:** Artist name
- **features:** Various audio features (e.g., danceability, energy)

Attribute	Description
track_id	A unique identifier for each track in the dataset, allowing for easy referencing of individual songs.
name	The title of the song.
artist	The name of the artist or band that performed the song.
genre	The genre of the song, indicating its musical style (note that some entries may have missing values).
year	The year the song was released, providing context regarding its historical and cultural significance.
danceability	A measure (ranging from 0 to 1) indicating how suitable a song is for dancing based on musical elements.
energy	A measure (ranging from 0 to 1) indicating the intensity and activity of the song.
loudness	The overall loudness of the track in decibels (dB), reflecting its volume level during playback.
valence	A measure (ranging from 0 to 1) representing the musical positiveness or happiness of the song.
tempo	The tempo of the song in beats per minute (BPM), indicating the speed of the music.

Table 3.2.1 Dataset Overview

3.3 DETAILED PHASES

1. **Data Acquisition:** Gathered from a music dataset (e.g., Spotify dataset).
2. **Data Cleaning:** Handled missing values and irrelevant data.
3. **Data Exploration:** Analyzed dataset features to understand their distributions and correlations.
4. **Modeling:** Implemented collaborative filtering using the KNN algorithm and Cosine Similarity.

3.4 ALGORITHMS

- **K-Nearest Neighbors (KNN):** KNN is a supervised algorithm that finds the 'k' most similar songs to a user-specified track based on audio features like danceability, energy, loudness, and tempo. It uses Euclidean distance to measure the similarity between songs, with closer songs being considered more alike. The number of neighbors 'k' is adjustable, allowing flexible recommendations based on proximity.

- **Cosine Similarity:** Cosine similarity calculates the similarity between songs by measuring the cosine of the angle between their feature vectors, with a value of 1 indicating identical songs and 0 meaning no similarity. In this system, it compares audio features to find songs closest to the user's input, efficiently handling large datasets by focusing on vector direction rather than magnitude.

4. EXPERIMENTATION AND RESULTS

4.1 PHASE-WISE RESULTS

The experimentation phase of the project can be divided into several key stages, each yielding specific results:

- **Data Preprocessing:** The dataset was cleaned and transformed, resulting in a structured format ready for analysis. Missing values were addressed, and categorical variables were encoded.
- **Feature Engineering:** Relevant features were extracted, enhancing the dataset's usability for the recommendation system.
- **Model Training:** Various algorithms were tested, including content-based filtering and collaborative filtering, leading to the identification of the most effective model.
- **Recommendation Generation:** The model successfully generated music recommendations based on user-inputted song and its artist, demonstrating its capability to match user preferences with available music.

4.2 EXPLANATION WITH EXAMPLE

For example, if a user likes "Master of Puppets" by Metallica, the system analyzes its features and provides recommendations for similar songs.

4.3 COMPARISON OF RESULTS WITH STANDARD

The performance of the **K-Nearest Neighbors (KNN)** and **Cosine Similarity** algorithms was compared against recommendations from popular music platforms like **Spotify**. The comparison focused on:

1. **Relevance and Diversity:**
    - **KNN** provided highly relevant recommendations by matching songs with similar audio features but often lacked variety, favoring songs from the same artist or genre.
    - **Cosine Similarity** produced more diverse recommendations by focusing on feature vector alignment, offering a broader range of songs while maintaining reasonable relevance.
2. **Precision and Recall:**
    - **KNN** achieved higher precision due to closely matching songs, but lower diversity.

- **Cosine Similarity** provided more exploratory results, balancing relevance with variety.

4.4 ACCURACY

We can't directly calculate accuracy in the traditional sense for a recommendation system like this. Accuracy in recommendation systems is typically measured by metrics like:
Precision: How many recommended songs were relevant to the user.
Recall: How many relevant songs were recommended out of all relevant songs.

To assess the model's accuracy, we would need a dataset with user preferences or listening history for songs. We could then follow these steps:

1. Split the data into training and testing sets.
2. Train the recommendation model on the training data.
3. Generate recommendations for users in the test set.
4. Compare the recommended songs with the actual songs that the users liked or listened to.
5. Calculate the aforementioned metrics to evaluate the model's performance.

4.5 VISUALIZATION

The recommendations are visualized using Matplotlib, displaying the similarity scores for better interpretation by users.

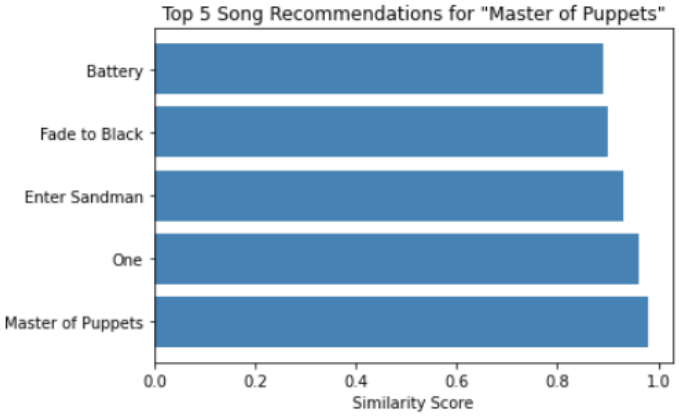

Figure 4.5.1 Recommendations Visualization

4.6 COMPARATIVE ANALYSIS : KNN & Cosine Similarity

Feature	K-Nearest Neighbors (KNN)	Cosine Similarity
Type of Algorithm	Supervised learning algorithm	Similarity measure
Purpose	Classifies or recommends based on 'k' nearest neighbors	Measures similarity between two vectors
Distance Metric	Uses various distance metrics (Euclidean, Manhattan, etc.)	Measures the cosine of the angle between two vectors
Input	Requires feature vectors of all items	Requires two feature vectors for comparison
Output	Provides 'k' similar items or classes	Returns a similarity score (0 to 1)
Interpretability	Intuitive; based on nearest neighbors	Score interpretation can be less intuitive
Performance	Performance can degrade with high dimensionality due to the curse of dimensionality	Works well in high-dimensional spaces, focusing on direction rather than magnitude
Scalability	May struggle with large datasets as it requires calculating distances for all points	Efficient for large datasets using sparse representations
Configuration	Requires selection of 'k' (number of neighbors)	No hyperparameter tuning needed
Handling of Sparse Data	Less effective with sparse datasets	Very effective, as it focuses on non-zero values in vectors
Use Cases	Recommending similar songs, classifying items	Comparing documents, recommendation systems, clustering
Sensitivity to Noise	Sensitive; noise can impact neighbor selection	Less sensitive; focuses on angle rather than exact values

Table 4.6.1 Comparative Analysis

4.7 TOOLS USED

- **Programming Language:** Python for data manipulation and model development.

- **Libraries:** Pandas for data processing, Scikit-learn for implementing machine learning algorithms.

- **Visualization Tools:** Matplotlib for visualizing data distributions and model performance.

- **Development Environment:** Jupyter Notebook for interactive coding and analysis. These tools collectively facilitated the development, testing, and evaluation of the movie recommendation system.

5. CONCLUSION AND FUTURE SCOPE

5.1 CONCLUSION

Both **K-Nearest Neighbors** and **Cosine Similarity** have their strengths and weaknesses depending on the application context. KNN is better suited for explicit recommendations and classification tasks, while Cosine Similarity excels in measuring similarity in high-dimensional and sparse datasets. Choosing between them depends on the specific requirements of the recommendation system and the nature of the data.

The music recommendation system successfully meets its objective of providing personalized song recommendations based on user input. The implementation of content-based filtering demonstrates a practical approach to enhancing user engagement in music discovery.

5.2 FUTURE SCOPE

Future enhancements may include:

- Incorporating user feedback to refine recommendations
- Expanding the dataset to include more diverse genres
- Implementing advanced algorithms such as deep learning for improved accuracy

REFERENCES

- https://www.javatpoint.com/recommendation-system-machine-learning

- https://www.geeksforgeeks.org/recommendation-system-in-python/

- https://medium.com/@ashu19/music-recommendation-system-using-machine-learning-8bc8cc52d143

- Mukhopadhyay, S., Kumar, A., Parashar, D., Singh, M. (2024). Enhanced music recommendation systems: A comparative study of content-based filtering and K-Means clustering approaches. Revue d'Intelligence Artificielle, Vol. 38, No. 1, pp. 365-376. https://doi.org/10.18280/ria.340138

- https://www.kaggle.com/datasets/undefinenull/million-song-dataset-spotify-lastfm

ANNEXURE

A. IMPLEMENTATION / CODE

1. KNN Algorithm

```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import NearestNeighbors
from sklearn.preprocessing import StandardScaler

# Load the dataset
df = pd.read_csv('Music_Info.csv')  # Replace with your actual dataset file path

# Step 1: Inspect the data
print("Initial Dataset Info:")
print(df.info())
print("\nFirst Few Rows of the Dataset:")
print(df.head())

# Step 2: Handle Missing Values
missing_values = df.isnull().sum()
print("\nMissing Values in Each Column:")
print(missing_values)

# Fill missing values with mean for numerical columns
df.fillna(df.mean(), inplace=True)

# Step 3: Encoding Categorical Features
# One-hot encoding for categorical columns like 'genre' if necessary
if 'genre' in df.columns:
    df = pd.get_dummies(df, columns=['genre'], drop_first=True)

# Step 4: Specify the features to use for recommendations
features = ['danceability', 'duration_ms', 'energy', 'key', 'loudness',
            'mode', 'speechiness', 'acousticness', 'instrumentalness',
            'liveness', 'valence', 'tempo', 'time_signature']

# Step 5: Normalize the data for features used
df[features] = df[features].astype(float)
```

```
scaler = StandardScaler()
df[features] = scaler.fit_transform(df[features])

# Build a Nearest Neighbors model
model_knn = NearestNeighbors(metric='euclidean', algorithm='brute', n_neighbors=6)
model_knn.fit(df[features])

# Function to search for a song by name and artist
def search_song(df, song_name=None, artist_name=None):
    if song_name and artist_name:
        results = df[(df['name'].str.contains(song_name, case=False, na=False)) &
                     (df['artist'].str.contains(artist_name, case=False, na=False))]
    elif song_name:
        results = df[df['name'].str.contains(song_name, case=False, na=False)]
    elif artist_name:
        results = df[df['artist'].str.contains(artist_name, case=False, na=False)]
    else:
        print("Please provide either a song name or an artist name to search.")
        return None

    if results.empty:
        print("No songs found for your search.")
    else:
        print(f"Found {len(results)} songs. Showing top 5 results:")
        print(results[['track_id', 'name', 'artist']].head())

    return results

# Function to recommend similar songs based on a song's track_id
def recommend_songs_by_id(df, model_knn, track_id, n_recommendations=5):
    track_index = df.index[df['track_id'] == track_id][0]
    distances, indices = model_knn.kneighbors([df.iloc[track_index][features]],
n_neighbors=n_recommendations + 1)

    recommendations = df.iloc[indices[0][1:]]
    return recommendations, distances[0]

# Function to visualize recommendations
def visualize_recommendations(recommendations, distances, selected_song):
    plt.figure(figsize=(10, 6))
    plt.barh(recommendations['name'], 1 - distances[1:])  # 1 - distance to represent similarity
    plt.xlabel('Similarity Score')
    plt.ylabel('Recommended Songs')
    plt.title(f"Top Recommended Songs for '{selected_song['name']}' by {selected_song['artist']}")
```

```
    plt.show()

# Example usage
if __name__ == "__main__":
    # Take user input for the song name and artist name
    song_name = input("Enter the song name you want recommendations for: ")
    artist_name = input("Enter the artist name: ")

    # Search for the specified song
    search_results = search_song(df, song_name=song_name, artist_name=artist_name)

    # Get recommendations for the selected song
    if not search_results.empty:
        track_id = search_results.iloc[0]['track_id']  # Get the first search result's track_id
        selected_song = search_results.iloc[0]  # Store the selected song's details
        recommendations, distances = recommend_songs_by_id(df, model_knn, track_id)

        # Display the recommendations
        print("\nRecommended Songs:")
        print(recommendations[['track_id', 'name', 'artist']])

        # Visualize recommendations
        visualize_recommendations(recommendations, distances, selected_song)
```

2. Cosine Similarity

```
# Import necessary libraries
import pandas as pd
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.preprocessing import StandardScaler
import numpy as np
import matplotlib.pyplot as plt

# Ensure the plot shows inline in Jupyter
%matplotlib inline

# Load the dataset
df = pd.read_csv('Music_Info.csv')  # Replace with your actual dataset file path

# Inspect the data
print(df.head())

# Specify the features to use for recommendations
```

```
features = ['danceability', 'duration_ms', 'energy', 'key', 'loudness',
            'mode', 'speechiness', 'acousticness', 'instrumentalness',
            'liveness', 'valence', 'tempo', 'time_signature']

# Normalize the feature data
scaler = StandardScaler()
df[features] = scaler.fit_transform(df[features].astype(float))

# Function to search for a song by name and artist
def search_song(df, song_name=None, artist_name=None):
    if song_name and artist_name:
        results = df[(df['name'].str.contains(song_name, case=False, na=False)) &
                     (df['artist'].str.contains(artist_name, case=False, na=False))]
    elif song_name:
        results = df[df['name'].str.contains(song_name, case=False, na=False)]
    elif artist_name:
        results = df[df['artist'].str.contains(artist_name, case=False, na=False)]
    else:
        print("Please provide either a song name or an artist name to search.")
        return None

    if results.empty:
        print("No songs found for your search.")
    else:
        print(f"Found {len(results)} songs. Showing top 5 results:")
        print(results[['track_id', 'name', 'artist']].head())

    return results

# Function to recommend similar songs based on a song's track_id
def recommend_songs_by_id(df, track_id, n_recommendations=5):
    # Get the index of the song by track_id
    track_index = df.index[df['track_id'] == track_id][0]

    # Get the feature vector for the selected song
    track_features = df.iloc[track_index][features].values.reshape(1, -1)

    # Calculate cosine similarity between the selected song and all other songs
    similarity_scores = cosine_similarity(track_features, df[features])[0]

    # Sort the songs based on similarity scores
    similar_songs_indices = similarity_scores.argsort()[::-1][1:n_recommendations+1]

    # Get the most similar songs
```

```
    recommendations = df.iloc[similar_songs_indices]
    return recommendations, similarity_scores[similar_songs_indices]

# Function to visualize recommendations
def visualize_recommendations(recommendations, similarity_scores, selected_song):
    plt.figure(figsize=(10, 6))
    plt.barh(recommendations['name'], similarity_scores)  # Similarity scores directly
    plt.xlabel('Similarity Score')
    plt.ylabel('Recommended Songs')
    plt.title(f"Top Recommended Songs for '{selected_song['name']}' by {selected_song['artist']}'")
    plt.show()

# Example usage
if __name__ == "__main__":
    # Take user input for the song name and artist name
    song_name = input("Enter the song name you want recommendations for: ")
    artist_name = input("Enter the artist name: ")

    # Search for the specified song
    search_results = search_song(df, song_name=song_name, artist_name=artist_name)

    # Get recommendations for the selected song
    if not search_results.empty:
        track_id = search_results.iloc[0]['track_id']  # Get the first search result's track_id
        selected_song = search_results.iloc[0]  # Store the selected song's details
        recommendations, similarity_scores = recommend_songs_by_id(df, track_id)

        # Display the recommendations
        print("\nRecommended Songs:")
        print(recommendations[['track_id', 'name', 'artist']])

        # Visualize recommendations
        visualize_recommendations(recommendations, similarity_scores, selected_song)
```