SPOTIFY CASE STUDY

Adithya | Romaisa | Suranjana | Andre

KEDGE BUSINESS SCHOOL

OBJECTIVE AND METHODOLOGY OVERVIEW

Task 1: Song Recommendation Algorithm for Spotify.

Goal:
Analyze existing user data to predict unknown user values and create personalized playlists.

Methodology Overview:
1. User Profile Analysis: Analyze user listening habits.
2. Model Training: Train classifiers using musical features.
3. Making Predictions: Use trained models to fill in missing data.
4. Playlist Reconstruction: Generate and save personalized playlists.

USER PROFILE ANALYSIS AND MODEL TRAINING

STEP 1: USER PROFILE ANALYSIS

Analyze Listening Habits: Extract key features such as average values for various musical attributes.

Example User Data:

- User: Delta
- Average Length: 233,046.60 ms
- Average Popularity: 31.78
- Number of Songs: 696

```
# Analyze user profiles
user_profiles = labeled_data.groupby('user').agg({
    'length': 'mean',
    'popularity': 'mean',
    'song_id': 'count'
}).reset_index()

print(user_profiles)
```

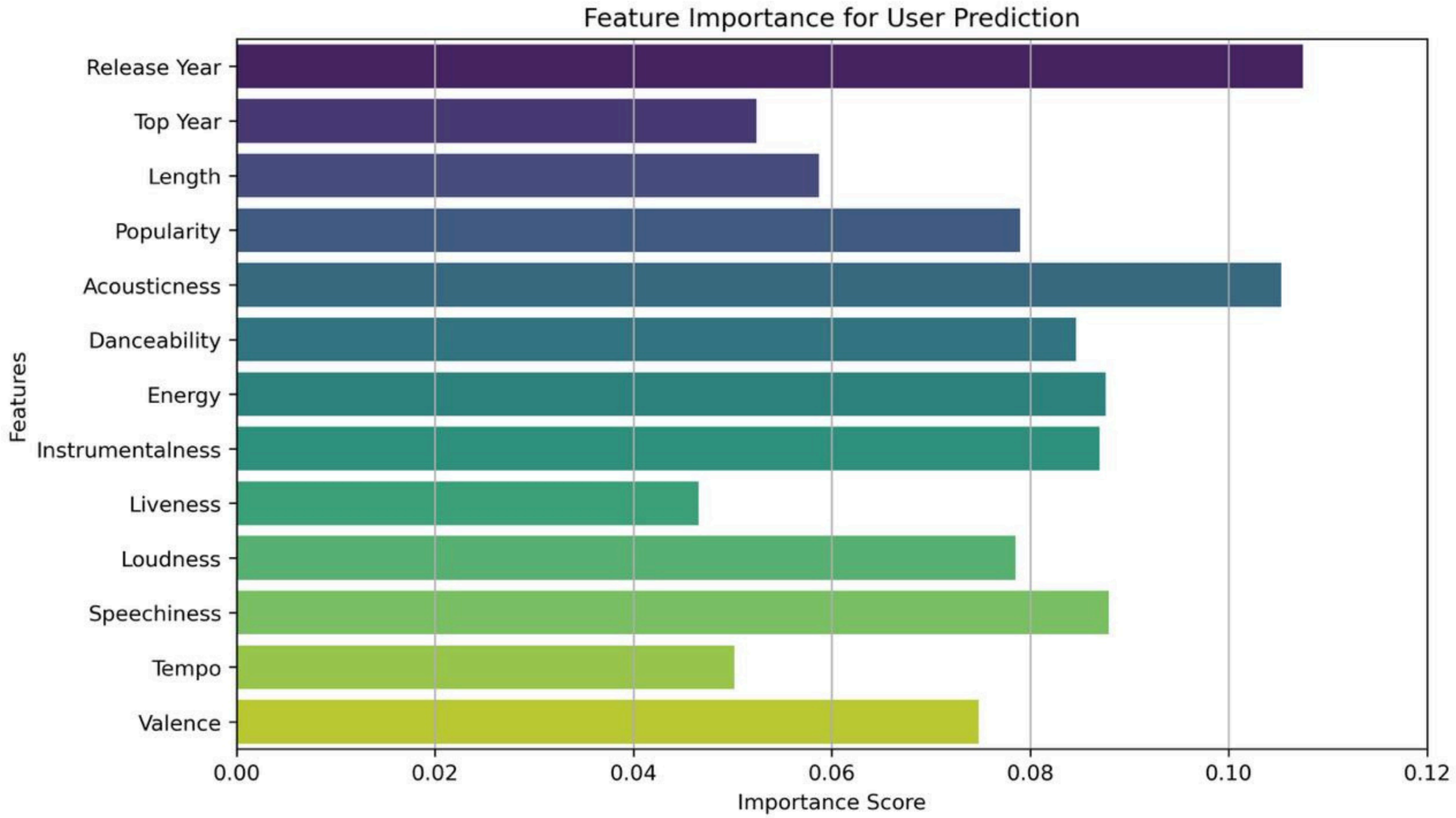
STEP 2: MODEL TRAINING

Expanded Features: Include additional musical attributes for better predictions

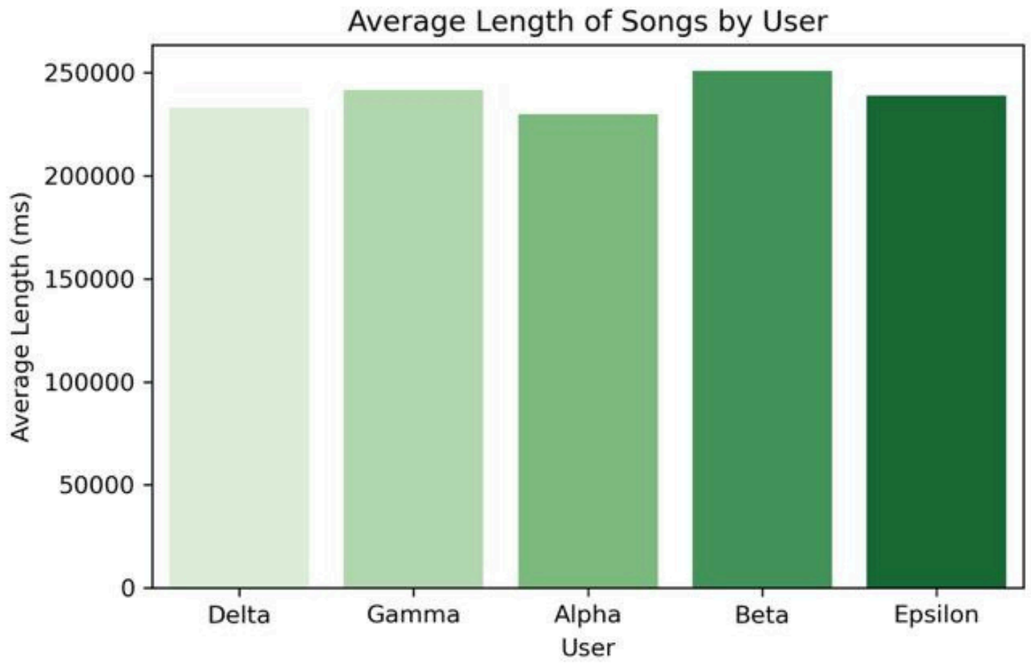
- Length, Popularity, Acousticness, Danceability, Energy, Instrumentalness, Liveness, Loudness, Speechiness, Tempo, Valence.

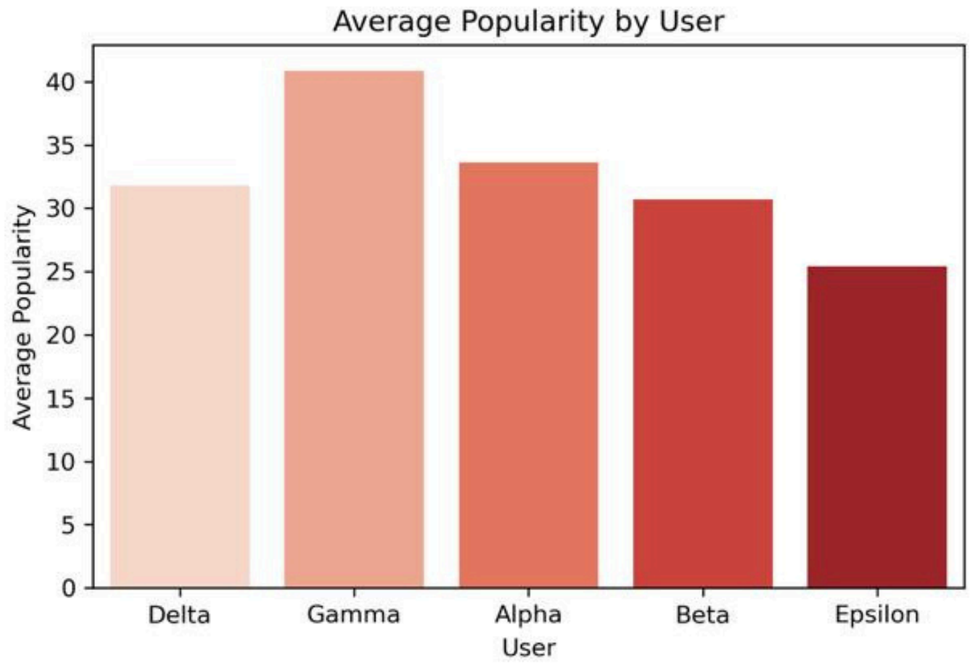
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

# Define the parameter grid for hyperparameter tuning
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt', 'log2']
}
```

USER PROFILE ANALYSIS



USER PROFILE ANALYSIS





PREDICTION AND RESULTS

STEP 3: CHECKING THE ACCURACY OF THE MODEL

Using the Best Estimator: After hyperparameter tuning, using the best model to make predictions.
Validation Accuracy: Evaluating the model's performance on the validation set.

Here, we can see the accuracy of the model after performing Accuracy, Recall and F1-score

```
Validation Accuracy: 0.6862
Recall: 0.6862
F1 Score: 0.6847

Classification Report:
              precision    recall  f1-score   support

       alpha       0.63      0.58      0.60       137
        beta       0.65      0.64      0.65       135
       delta       0.73      0.68      0.70       141
     epsilon       0.68      0.72      0.70       135
       gamma       0.73      0.80      0.76       150

    accuracy                           0.69       698
   macro avg       0.68      0.68      0.68       698
weighted avg       0.68      0.69      0.68       698
```

PLAYLIST RECONSTRUCTION

STEP 4: GENERATING TOP OF THE YEAR PLAYLISTS

Reconstruct each user's Top-of-the-Year playlists for each year

Achievements:

- Successfully reconstructed personalized "Top of the Year" playlists for each user.

- Enhanced user experience by providing tailored playlists based on predictions.

```
# Save reconstructed playlists
for user in data['user_complete'].unique():
    playlist = data[data['user_complete'] == user]
    playlist.to_csv(f"{user}_top_playlist.csv", index=False)
```

TASK 2: RECOMMENDATION ALGORITHM

1. Retrieve lost user playlist data
2. Generate a song recommendation algorithm and suggest several song features as an algorithm input

Pipeline Overview

Data Overview: Looking on features

Features Selection and scaling

Model Training and Prediction: Selecting Model / Approaches

Evaluation

Dataset Overview

Dataset have 22 columns which we will group to 3 categories

Song Metadata(6)

Name
Album
Artist
Release_date
Uri
Release_year

Users Data (2)

Top_year (user's top tracks year)
user

Song Technical

Features (14)

Length
Acousticness
Danceability
Energy
Instrumentalness
Liveness
Loudness
Speechiness
Tempo
Valence
Time_signature
Key
Mode
Popularity (0-100, track popularity based on number of play and recent play)

Numerical values with different scale

DATASET OVERVIEW

Create User-Item Matrix and Check Sparsity, we get the sparsity is 20% in this dataset

Users	Song	Popularity
Alpha	1	80
Beta	2	50
Gamma	3	50

	Song 1	Song 2	Song 3
Alpha	80		
Beta		50	
Gamma	50		

Calculate Matrix Sparsity

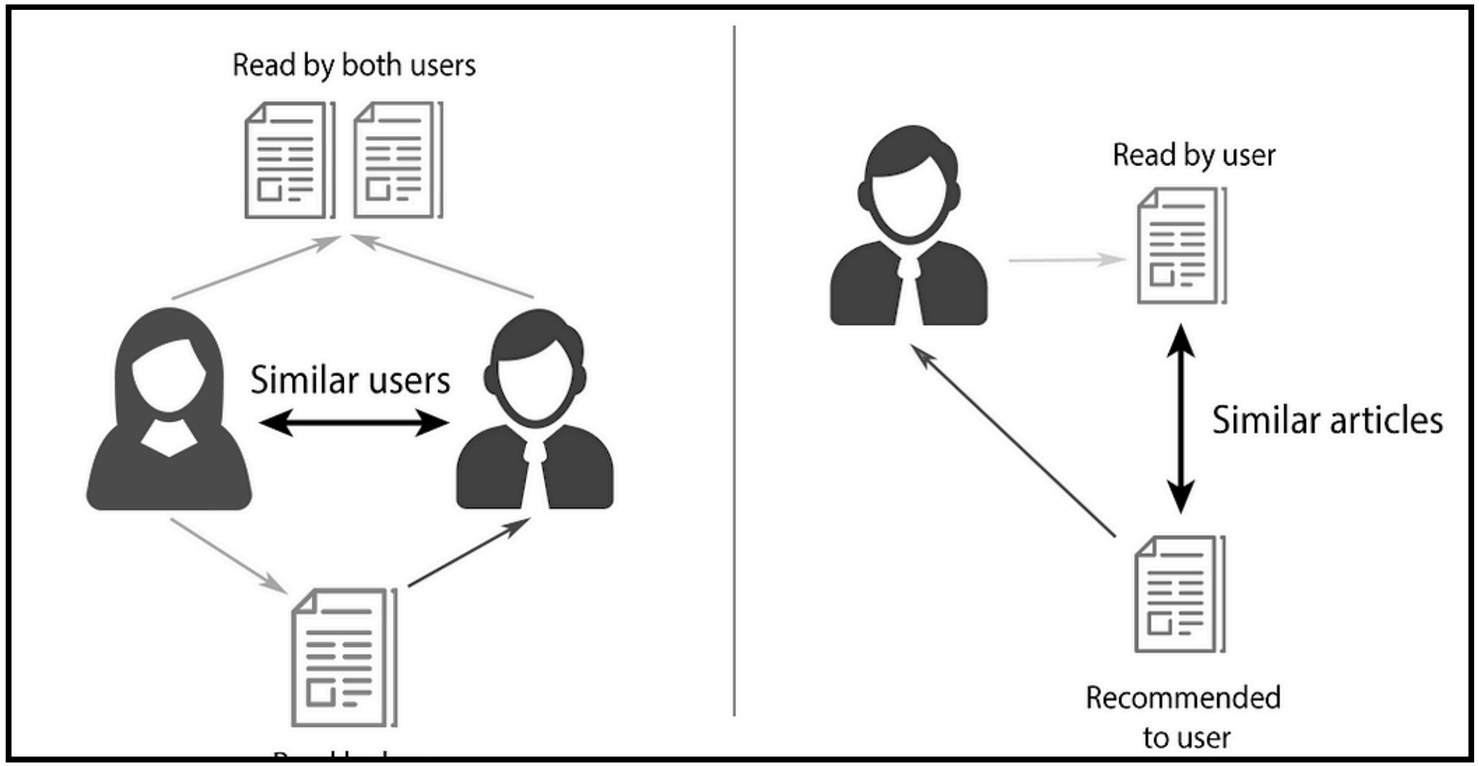
$$\text{sparsity} = \frac{\text{\# ratings}}{\text{total \# elements}}$$

FEATURES SELECTION AND SCALING

Since we have 14 audio features, we will do feature selection to get the most relevant audio features with popularity as the target label

The workflow is as follows:
1. Create a pipeline consisting of scaling (MinMaxScaler) and feature selection (SelectKBest) with score function regression since we wanted features that positively correlated with popularity
2. Add Hyperparameter tuning for parameter K in SelectKBest using Grid Search with cross-validation and apply Negative MSE as scoring metric since the feature selection is regression
3. Apply the best parameter and check the selected features

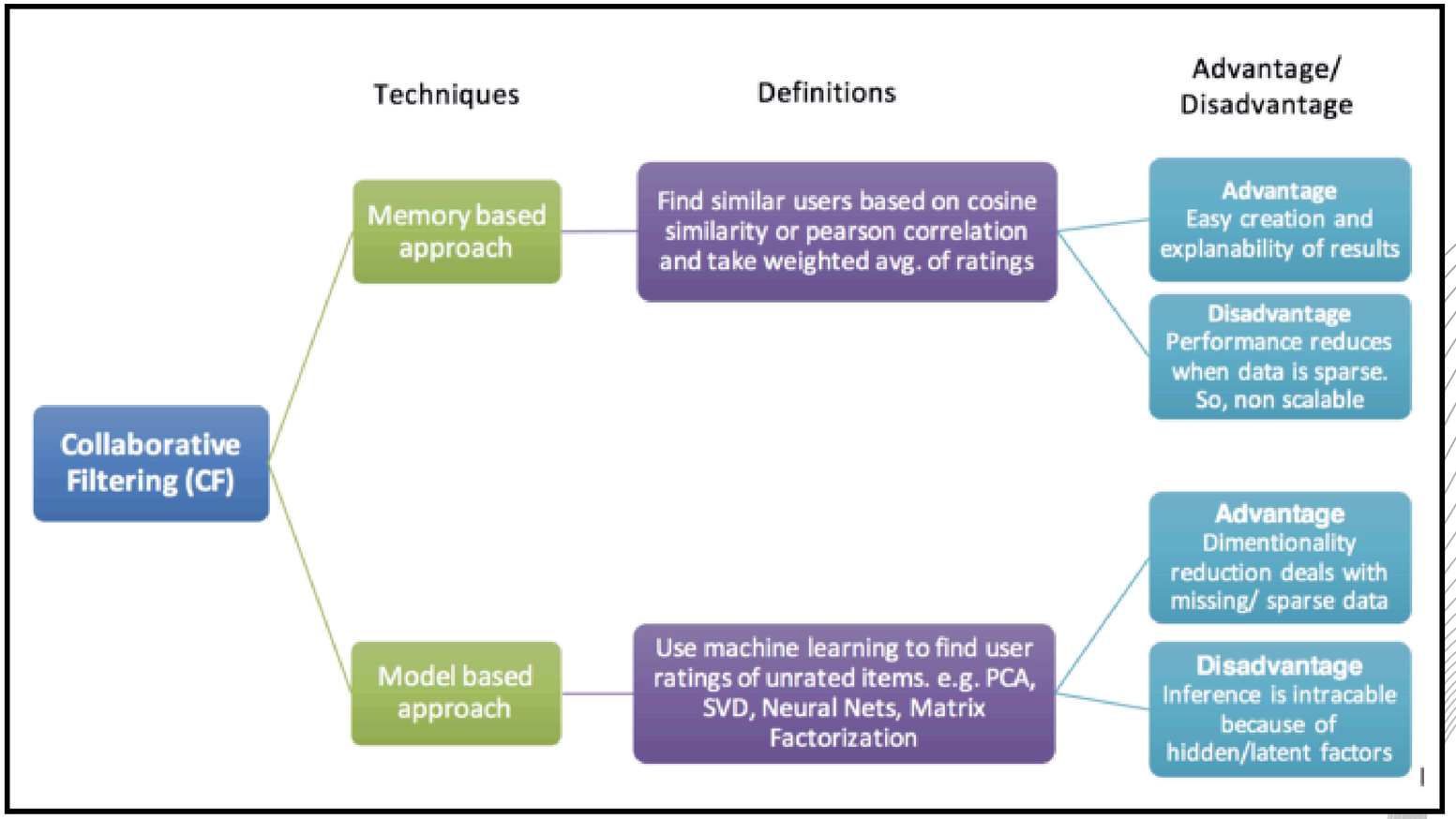
MODEL TRAINING: MODEL SELECTION



After we have selected the best features, we need to choose the algorithm of our recommender system. We will use the hybrid of Collaborative filtering and content-based filtering to taking account of User-Item interaction(CF) and Item-Item Similarity (CBF). To give recommendation for specific users and also with similar songs

MODEL TRAINING: MODEL SELECTION

For the collaborative filtering, looking at our dataset, since we have a smaller dataset (one family account) and high sparsity (20%) therefore we will use a model-based approach specifically the Alternating Least Square (ALS) algorithm because it can handle sparse dataset



MODEL TRAINING: COLLABORATIVE FILTERING

We will use User-Item interaction matrix which in our data is:

- user, popularity and top_year
- Create a matrix of users and songs with the value is interaction score

	Song 1	Song 2	Song 3
Alpha	0.8		
Beta		0.5	
Gamma	0.5		

Alternating Least Square

Parameter:
Factor
Regularization

Predicted value

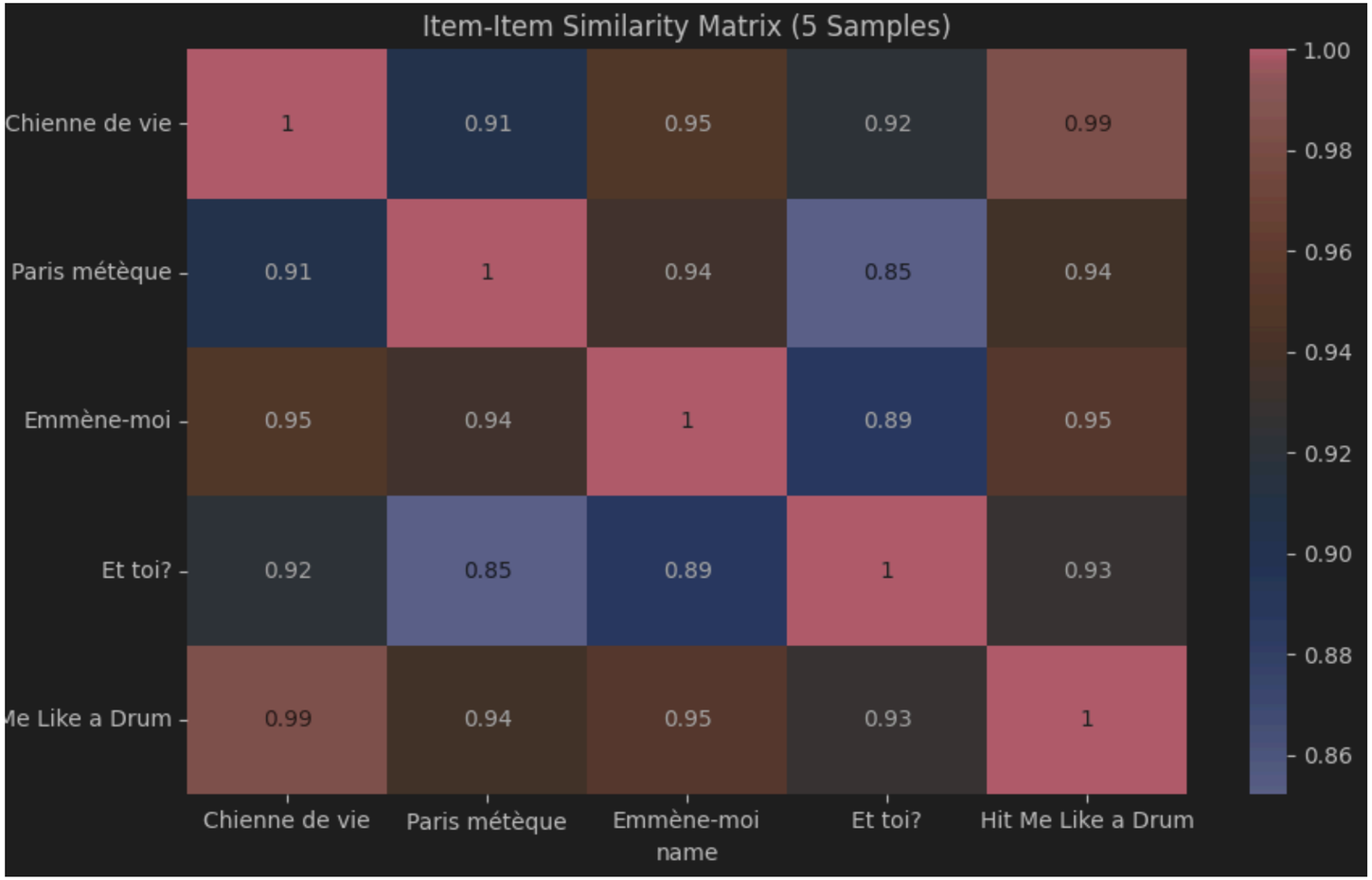
	Song 1	Song 2	Song 3
Alpha	0.8	0.6	0.5
Beta	0.7	0.5	0.4
Gamma	0.5	0.7	0.8

$$\text{Interaction Score} = \frac{\text{scaled popularity} \times \text{scaled top\_year}}{2}$$

- Popularity (assuming higher popularity is more favourable by users)
- And top_year which will be scaled from 0 to 1 as factor to popularity (more recent year give higher score)

MODEL TRAINING: CONTENT-BASED FILTERING

Create item-item similarity matrix using song technical features from feature selection results applying **cosine similarity**



MODEL TRAINING: CREATING HYBRID MODEL

Combining collaborative filtering and content-based filtering by giving a weight factor to each results

$$\text{Final Recommendation} = \alpha \times \text{scaled\_ALS\_score} + \beta \times \text{scaled\_CBF\_score}$$

User Related

Song Related

We try to set $\alpha = 0.4$ and $\beta = 0.6$ to emphasize more on song related factor since the user-item data is very sparse

EVALUATION

To evaluate the recommendation result we will use **Precision@K** and **Recall@K** where:

- **Precision@K is the proportion of the number of relevant items to the K recommendation**
- **Recall@K is the proportion of relevant items in K recommendation to the total relevant items**

To determine relevant items, we will use final score threshold of 0.8 to be considered as relevant items for users

APPENDIX

1.How to Design and Build a Recommendation System Pipeline in Python (Jill Cates)
2.Recommendation Systems — A walk through
3.Basics of Content Based and Collaborative Based Recommendation Engines

THANK YOU


Welcome
Follow
Play
Spotify
Theme Template
Gallery
STARBOY
30