Final_Project_Notebook

June 29, 2025

1 Explaining Music Recommendations through Audio Features and Spectrogram Analysis

Team 4: Anna-Mariia Chornohuz (h12014626) and Iana Schnattler (h11831531)

2 1. Literature Review

3 Project Overview: Spectrogram-Based Explainability in Music Recommender Systems

3.1 1. Introduction

Music recommender systems have become essential components of streaming platforms such as **Spotify**, **Apple Music**, and **Deezer**. These systems aim to increase user satisfaction and engagement by providing **personalized track suggestions** based on past listening behavior, inferred preferences, and contextual variables. Despite their success, a critical challenge remains: **explainability**—that is, the ability to convey to users why a particular song was recommended.

Explainable recommender systems (XRS) are especially relevant in domains like music, where preferences are **subjective, emotional, and context-sensitive**. Prior work has demonstrated that when users understand the reasoning behind recommendations, their trust and adoption rates increase (Zhang & Chen, 2018). However, most platforms today provide little to no insight into the underlying logic behind their recommendations.

3.2 2. Limitations of Traditional Systems

Most existing systems are based on two fundamental approaches: - **Collaborative Filtering (CF)**: Recommends items liked by users with similar behavior. - **Content-Based Filtering (CBF)**: Recommends items based on the features of previously liked items (e.g., tempo, energy, valence, etc.).

These features, while powerful, are typically **invisible to users** and highly technical, which makes human-centered explanations difficult. A user is unlikely to find it meaningful to hear that a song was recommended because it has a high “zero-crossing rate” or “spectral centroid” (Nam et al., 2022).

3.3 3. Post-Hoc Explainability Techniques

Recent advances in explainable AI (XAI) have led to the adoption of **model-agnostic techniques** such as: - **LIME** (Local Interpretable Model-Agnostic Explanations) - **SHAP** (SHapley Additive exPlanations)

These methods explain the model's decision by identifying which input features most contributed to the output. However, their reliance on low-level numeric features **limits user interpretability**, especially in music, where emotional and perceptual aspects dominate (Lundberg & Lee, 2017; Ribeiro et al., 2016; Marconi et al., 2023).

3.4 4. Spectrograms as Perceptual Explanation Tools

A **spectrogram** is a visual representation of an audio signal's frequency content over time. It captures structural elements of sound such as: - Rhythm - Timbre - Pitch contours - Instrumentation

Unlike numeric features, **spectrograms preserve the sonic character** of a track, allowing for intuitive, human-interpretable comparisons between songs. Spectrograms can effectively "show" the texture of music, acting as a bridge between signal-level data and human perception (Müller, 2015).

3.5 5. Proposed Method

This project proposes a **hybrid recommendation-explanation system** that: 1. Uses standard audio features (e.g., tempo, RMS, spectral centroid) for similarity-based recommendation. 2. Generates **spectrograms** from raw audio and uses them to both: - **Inform** deep learning models trained on image-based representations. - **Explain** recommendations visually to users.

By comparing spectrograms of a liked track and its recommendation, users can **visually perceive similarities** in structure, rhythm, or tonal content. This bridges the gap between black-box recommendation logic and perceptual user understanding.

3.6 6. Comparative Evaluation

We will compare: - **Feature-based similarity** models (e.g., cosine distance on tempo, RMS, ZCR). - **Spectrogram-based CNN models** (trained to predict similarity based on visual data). - **User feedback** on perceived clarity and trustworthiness of explanations.

Relevant previous work includes deep learning-based music classification using spectrograms and CNNs (e.g., ResNet-50, VGG-16), and feature-based interpretability studies (Nam et al., 2022).

3.7 7. Contribution to Explainable AI (XAI)

This work contributes to the XAI and **human-centered AI design** communities by: - Introducing perceptually aligned explanations in music recommendation. - Exploring **spectrograms as both**

input and explanation tools. - Highlighting how users can interact with visual data to enhance trust and satisfaction.

Ultimately, this approach seeks to make music recommendation not only accurate but also **transparent, interpretable, and emotionally resonant**.

3.8 8. Next Steps

- Implement spectrogram extraction and visualization.
- Train CNN or similarity-based models on spectrograms.
- Compare outputs with traditional feature-based models.
- Conduct user testing to evaluate **interpretability and trust**.

3.9 References

- Zhang, Y., & Chen, X. (2020). *Explainable Recommendation: A Survey and New Perspectives*. Foundations and Trends® in Information Retrieval, 14(1), 1–101.
- Lundberg, S. M., & Lee, S.-I. (2017). *A Unified Approach to Interpreting Model Predictions*. Advances in Neural Information Processing Systems, 30.
- Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). *"Why Should I Trust You?": Explaining the Predictions of Any Classifier*. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 1135–1144.
- Marconi, L., Matamoros Aragón, R., & Epifania, F. (2023). *A Short Review on Explainability for Recommender Systems*. CEUR Workshop Proceedings, Vol. 3463.
- Müller, M. (2015). *Fundamentals of Music Processing: Audio, Analysis, Algorithms, Applications*. Springer.
- Nam, J., Herrera, J., Slaney, M., & Smith III, J. O. (2012, October). Learning Sparse Feature Representations for Music Annotation and Retrieval. In ISMIR (pp. 565-570).

4 2. Preparation

4.1 Step 1: Load and Prepare Data

[19]:
```
import pandas as pd

# Load datasets
df_deezer = pd.read_csv("Deezer Data/metadata_DEEZER_active.csv")
df_spotify = pd.read_csv("Spotify Data/train.csv")

# Clean Deezer data
df_deezer_clean = df_deezer.dropna(subset=["country"]).copy()
```

```
df_deezer_clean = df_deezer_clean.rename(columns={"item_id": "deezer_item_id"})

# Clean Spotify data
df_spotify_clean = df_spotify[[
    "track_id", "track_name", "artists", "danceability", "energy", "valence",
    "tempo", "acousticness", "instrumentalness", "speechiness", "track_genre"
]].dropna()
df_spotify_clean["track_name_lc"] = df_spotify_clean["track_name"].str.lower().
 ↪str.strip()
df_spotify_clean["artist_lc"] = df_spotify_clean["artists"].str.lower().str.
 ↪strip()

# Simulate matched tracks (random pairing)
sample_matched_spotify = df_spotify_clean.sample(n=100, random_state=1).copy()
sample_matched_spotify["deezer_item_id"] = df_deezer_clean["deezer_item_id"].
 ↪sample(n=100, random_state=2).values

# Merge to add country info
df_matched = pd.merge(sample_matched_spotify, df_deezer_clean,␣
 ↪on="deezer_item_id")
df_matched.head()
```

[19]:

	track_id	track_name	artists	danceability	energy	valence	tempo	acousticness	instrumentalness	speechiness	track_genre	track_name_lc	artist_lc	deezer_item_id	country
0	1JiR4RJaZlbZ5b3HG8jkeL	Longtime (feat. Skepta)	Wizkid;Skepta	0.850	0.660	0.622	101.947	0.4170	0.000051	0.1690	dancehall	longtime (feat. skepta)	wizkid;skepta	58552	DE
1	6cfNBFSKFB59wO8xIFZOqI	Dom na wiślanym brzegu	Ivan komarenko	0.640	0.758	0.849	106.978	0.4970	0.000000	0.0675	disco	dom na wiślanym brzegu	ivan komarenko	15195	DE
2	5VcFzH97JEHgXgedErp4cP	Idea 686	Jayla Darden	0.712	0.347	0.500	135.975	0.3780	0.021200	0.0410	chill	idea 686	jayla darden	340815	FR
3	7oMBY3VmIxqNVeZ7yneYuN	Me Voy Enamorando	Chino & Nacho	0.686	0.912	0.511	99.952	0.0533	0.000000	0.0685	electro	me voy enamorando	chino & nacho		
4	7g8U2TPh6JPFJK5LgqsNeE	What I've Done	Linkin Park	0.623	0.930	0.287	120.119	0.0141	0.000002	0.0324	grunge	what i've done	linkin park		

```
3          265475      FR
4          327431      US
```

[14]:
```python
import re

# Helper to clean titles
def clean_title(title):
    if pd.isna(title):
        return None
    title = title.lower()
    title = re.sub(r"\(.*?\)", "", title)  # remove anything in ( )
    title = re.sub(r"/.*", "", title)      # remove anything after /
    title = re.sub(r"[^a-z0-9\s]", "", title)  # remove punctuation
    return title.strip()

# Helper to extract primary artist
def clean_artist(artist):
    if pd.isna(artist):
        return None
    artist = artist.lower()
    artist = artist.split(";")[0]  # only first artist
    artist = re.sub(r"(feat\.|with).*", "", artist)  # remove 'feat.', 'with'
    artist = re.sub(r"[^a-z0-9\s]", "", artist)  # remove punctuation
    return artist.strip()

# Apply to both datasets
df_deezer_enriched["track_name_clean"] = df_deezer_enriched["track_name"].
 ↪apply(clean_title)
df_deezer_enriched["artist_clean"] = df_deezer_enriched["artists"].
 ↪apply(clean_artist)

df_spotify_clean["track_name_clean"] = df_spotify_clean["track_name"].
 ↪apply(clean_title)
df_spotify_clean["artist_clean"] = df_spotify_clean["artists"].
 ↪apply(clean_artist)

# Now merge on cleaned fields
df_matched = pd.merge(
    df_deezer_enriched,
    df_spotify_clean,
    on=["track_name_clean", "artist_clean"],
    how="inner"
)

# Deduplicate
df_matched = (
    df_matched
```

```
    .sort_values("valence", ascending=False)
    .drop_duplicates(subset=["deezer_item_id"])
)

# Save
df_matched.to_csv("Matched Data/deezer_spotify_matched_200_cleaned.csv",␣
 ↪index=False)

print(f" Matched {len(df_matched)} of 200 with cleaned matching.")
display(df_matched[["track_name_x", "artists_x", "track_name_y", "artists_y"]].
 ↪head())
```

```
 Matched 1 of 200 with cleaned matching.
```

	track_name_x	artists_x	track_name_y	artists_y
0	Traumtänzerball (Album Version)	Michelle	Traumtänzerball	Michelle

[15]:
```
pip install rapidfuzz
```

```
Collecting rapidfuzz
  Downloading
rapidfuzz-3.9.7-cp38-cp38-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (3.4
MB)
     |                  | 3.4 MB 1.2 MB/s eta 0:00:01
Installing collected packages: rapidfuzz
Successfully installed rapidfuzz-3.9.7
Note: you may need to restart the kernel to use updated packages.
```

[13]:
```
print(" Sample from enriched Deezer data:")
print(df_deezer_enriched[["track_name", "artists"]].dropna().drop_duplicates().
 ↪head(10))

print("\n Sample from Spotify data:")
print(df_spotify_clean[["track_name", "artists"]].dropna().drop_duplicates().
 ↪head(10))
```

```
 Sample from enriched Deezer data:
```

	track_name	artists
4	Air (Johann Sebastian Bach)	Ed Staginsky
7	Grande Fratello (remix)	Roberto Delledonne
8	My Life	Mortimer Shuman
9	Carousel	Michael Jackson
10	Quincy Jones Interview #3	Quincy Jones
14	Goodbye (na na na) (Original Radio Version)	Rob M
16	Voice-Over Intro / Voice-Over Session from Thr…	Michael Jackson
17	Untha	Les Secrets De Morphee
20	A march for new european	Jack Or Jive
21	The Girl Is Mine (2008 with will.i.am) (with w…	Michael Jackson

```
 Sample from Spotify data:
                  track_name                                artists
0                     Comedy                            Gen Hoshino
1           Ghost - Acoustic                           Ben Woodward
2             To Begin Again                 Ingrid Michaelson;ZAYN
3  Can't Help Falling In Love                           Kina Grannis
4                    Hold On                        Chord Overstreet
5       Days I Will Remember                           Tyrone Wells
6              Say Something  A Great Big World;Christina Aguilera
7                  I'm Yours                               Jason Mraz
8                      Lucky              Jason Mraz;Colbie Caillat
9                     Hunger                          Ross Copperman
```

[18]:
```python
# Combine title and artist for matching
df_deezer_enriched["match_key"] = df_deezer_enriched["track_name_clean"] + " -␣
 ↪" + df_deezer_enriched["artist_clean"]
df_spotify_clean["match_key"] = df_spotify_clean["track_name_clean"] + " - " +␣
 ↪df_spotify_clean["artist_clean"]

spotify_keys = df_spotify_clean["match_key"].dropna().unique().tolist()

# Updated safe function
def find_best_match(deezer_key):
    if pd.isna(deezer_key) or not isinstance(deezer_key, str) or deezer_key.
 ↪strip() == "":
        return pd.Series([None, 0])

    result = process.extractOne(deezer_key, spotify_keys, scorer=fuzz.
 ↪token_sort_ratio)
    if result is None:
        return pd.Series([None, 0])

    match, score, _ = result
    return pd.Series([match, score])

# Apply to 200 entries
df_sample = df_deezer_enriched.head(200).copy()
df_sample[["best_match", "match_score"]] = df_sample["match_key"].
 ↪apply(find_best_match)

# Filter and merge
df_matches_fuzzy = df_sample[df_sample["match_score"] >= 90].copy()
df_spotify_subset = df_spotify_clean[["match_key", "track_id", "track_name",␣
 ↪"artists", "danceability", "energy", "valence",
    "tempo", "acousticness", "instrumentalness", "speechiness", "track_genre"]]

df_final_fuzzy = pd.merge(
```

```
    df_matches_fuzzy,
    df_spotify_subset,
    left_on="best_match",
    right_on="match_key",
    how="left"
)

df_final_fuzzy.to_csv("Matched Data/deezer_spotify_matched_200_fuzzy.csv",␣
 ↪index=False)

print(f" Fuzzy-matched {len(df_final_fuzzy)} Deezer tracks (score  90)")
display(df_final_fuzzy[["track_name_x", "artists_x", "track_name_y",␣
 ↪"artists_y", "match_score"]].head())
```

```
 Fuzzy-matched 1 Deezer tracks (score  90)
```

	track_name_x	artists_x	track_name_y	artists_y	match_score
0	Traumtänzerball (Album Version)	Michelle	Traumtänzerball	Michelle	100.0

4.2 Step 2: Spectrogram Preparation and Feature Extraction

[2]:
```
# Generate spectrogram placeholder paths
df_matched["spectrogram_path"] = df_matched.apply(
    lambda row: f"spectrograms/{row['track_name'].replace('/',␣
 ↪'-')}_{row['artists'].split(';')[0]}.png", axis=1
)

# Extract audio features
feature_cols = ['danceability', 'energy', 'valence', 'tempo', 'acousticness',␣
 ↪'instrumentalness', 'speechiness']
df_features = df_matched[['track_id'] + feature_cols].set_index('track_id')
df_features.head()
```

[2]:

track_id	danceability	energy	valence	tempo	acousticness	instrumentalness	speechiness
1JiR4RJaZlbZ5b3HG8jkeL	0.850	0.660	0.622	101.947	0.4170	0.000051	0.1690
6cfNBFSKFB59wO8xIFZOqI	0.640	0.758	0.849	106.978	0.4970	0.000000	0.0675
5VcFzH97JEHgXgedErp4cP	0.712	0.347	0.500	135.975	0.3780		
7oMBY3VmIxqNVeZ7yneYuN	0.686	0.912	0.511	99.952	0.0533		
7g8U2TPh6JPFJK5LgqsNeE	0.623	0.930	0.287	120.119	0.0141		

```
5VcFzH97JEHgXgedErp4cP          0.021200       0.0410
7oMBY3VmIxqNVeZ7yneYuN          0.000000       0.0685
7g8U2TPh6JPFJK5LgqsNeE          0.000002       0.0324
```

4.3 Step 3: Build Content-Based Recommender

[3]:
```
from sklearn.metrics.pairwise import cosine_similarity

def get_recommendations(track_id, df_features, top_n=3):
    if track_id not in df_features.index:
        return pd.DataFrame()
    track_vec = df_features.loc[[track_id]]
    rest = df_features.drop(index=track_id)
    similarities = cosine_similarity(track_vec, rest)[0]
    rest = rest.copy()
    rest['similarity'] = similarities
    return rest.sort_values('similarity', ascending=False).head(top_n)

# Pick one reference track
reference_id = df_features.index[0]
recommended_df = get_recommendations(reference_id, df_features, top_n=3).
 ↪reset_index()
recommended_df
```

[3]:
```
                track_id  danceability  energy  valence    tempo  \
0  0CyiCor3YLnPcBkZ04cLz2         0.823   0.782    0.939  109.892
1  24SDeYAeTFda8OUzVI1VR6         0.571   0.747    0.635  106.253
2  6cfNBFSKFB59wO8xIFZ0qI         0.640   0.758    0.849  106.978

   acousticness  instrumentalness  speechiness  similarity
0         0.391               0.0       0.1980    0.999996
1         0.478               0.0       0.1700    0.999995
2         0.497               0.0       0.0675    0.999995
```

4.4 Step 4: Visual Comparison Setup

[7]:
```
# Prepare combined view of reference + recommended tracks
reference_row = df_matched[df_matched['track_id'] == reference_id]
recommend_rows = df_matched[df_matched['track_id'].
 ↪isin(recommended_df['track_id'])]
visual_compare = pd.concat([reference_row, recommend_rows], axis=0).
 ↪reset_index(drop=True)
visual_compare[["track_name", "artists", "track_genre", "spectrogram_path"]]
```

[7]:
```
                                track_name                    artists  \
0                    Longtime (feat. Skepta)              Wizkid;Skepta
1                     Dom na wiślanym brzegu             Ivan komarenko
```

```
2                                Weh Di Time  Voicemail;Delly Ranks;Bogle
3  TWIST & TURN (feat. Drake & PARTYNEXTDOOR)  Popcaan;Drake;PARTYNEXTDOOR

  track_genre                                    spectrogram_path
0   dancehall    spectrograms/Longtime (feat. Skepta)_Wizkid.png
1       disco  spectrograms/Dom na wiślanym brzegu_Ivan komar…
2     j-dance             spectrograms/Weh Di Time_Voicemail.png
3   dancehall  spectrograms/TWIST & TURN (feat. Drake & PARTY…
```

5 3. Spectrogram Generation and Explanation

5.1 Step 1: Setup and Imports

[8]:
```
import librosa
import librosa.display
import matplotlib.pyplot as plt
import numpy as np
import os
from pathlib import Path

# Create output folder if it doesn't exist
output_dir = Path("spectrograms")
output_dir.mkdir(exist_ok=True)
```

5.2 Step 2: Spectrogram Generator

[9]:
```
# Alternative spectrogram generation using scipy to avoid librosa/numba issues
import matplotlib.pyplot as plt
from scipy.io import wavfile
from scipy.signal import spectrogram

def generate_spectrogram(audio_path, save_path):
    sr, y = wavfile.read(audio_path)
    f, t, Sxx = spectrogram(y, sr)
    plt.figure(figsize=(10, 4))
    plt.pcolormesh(t, f, 10 * np.log10(Sxx + 1e-10), shading='gouraud',␣
 ↪cmap='viridis')
    plt.ylabel('Frequency [Hz]')
    plt.xlabel('Time [sec]')
    plt.title('Spectrogram (Scipy)')
    plt.colorbar(label='Intensity [dB]')
    plt.tight_layout()
    plt.savefig(save_path)
    plt.close()
```

5.3 Step 3: Example – Generate Spectrogram for One File

[11]:
```
# Install required library (run only once)
!pip install pydub
```

```
Collecting pydub
  Downloading pydub-0.25.1-py2.py3-none-any.whl (32 kB)
Installing collected packages: pydub
Successfully installed pydub-0.25.1
```

[12]:
```
# Import required modules
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import wavfile
from scipy.signal import spectrogram
from pydub import AudioSegment
import os

# Create output directory
output_dir = "spectrograms"
os.makedirs(output_dir, exist_ok=True)

# Define the MP3 input and paths
mp3_path = "JJ - Wasted Love (Official Audio).mp3"  # Your uploaded file
wav_path = "converted_audio.wav"
spectrogram_path = os.path.join(output_dir, "wasted_love_spectrogram.png")

# Convert MP3 to WAV
audio = AudioSegment.from_mp3(mp3_path)
audio.export(wav_path, format="wav")

# Define spectrogram generation function
def generate_spectrogram(audio_path, save_path):
    sr, y = wavfile.read(audio_path)
    if y.ndim > 1:  # Convert stereo to mono if needed
        y = y.mean(axis=1)
    f, t, Sxx = spectrogram(y, sr)

    plt.figure(figsize=(10, 4))
    plt.pcolormesh(t, f, 10 * np.log10(Sxx + 1e-10), shading='gouraud',␣
 ↪cmap='viridis')
    plt.ylabel('Frequency [Hz]')
    plt.xlabel('Time [sec]')
    plt.title('Spectrogram')
    plt.colorbar(label='Intensity [dB]')
    plt.tight_layout()
    plt.savefig(save_path)
    plt.close()
```

```
# Generate and save the spectrogram
generate_spectrogram(wav_path, spectrogram_path)
print(f"Spectrogram saved to {spectrogram_path}")
```

```
Spectrogram saved to spectrograms/wasted_love_spectrogram.png
```

5.4 Step 4: Optional – Prepare for Clustering or SHAP Explanation

[ ]:
```
# For SHAP or clustering, use the feature dataframe from the earlier notebook
# Example: df_features.to_csv('features.csv')

# Then you can import into this notebook and use SHAP or clustering libraries
  ↪like sklearn or seaborn
```

6 4. Download the MP3 from preview_url, Convert, and Generate Spectrogram

[ ]:
```
import requests
from pydub import AudioSegment
import librosa
import librosa.display
import matplotlib.pyplot as plt
import numpy as np
import os

# === CONFIG ===
preview_url = "https://p.scdn.co/mp3-preview/..."  # Replace with actual
  ↪preview_url
output_dir = "spotify_previews"
os.makedirs(output_dir, exist_ok=True)
mp3_path = os.path.join(output_dir, "track_preview.mp3")
wav_path = os.path.join(output_dir, "track_preview.wav")
spectrogram_path = os.path.join(output_dir, "track_spectrogram.png")

# === 1. Download the preview MP3 ===
response = requests.get(preview_url)
with open(mp3_path, "wb") as f:
    f.write(response.content)

# === 2. Convert MP3 to WAV ===
audio = AudioSegment.from_mp3(mp3_path)
audio.export(wav_path, format="wav")

# === 3. Generate Spectrogram from WAV ===
y, sr = librosa.load(wav_path, sr=None)
```

```
S = librosa.feature.melspectrogram(y=y, sr=sr, n_mels=128)
S_DB = librosa.power_to_db(S, ref=np.max)

plt.figure(figsize=(10, 4))
librosa.display.specshow(S_DB, sr=sr, x_axis='time', y_axis='mel',␣
 ↪cmap='viridis')
plt.colorbar(format='%+2.0f dB')
plt.title('Spectrogram of Spotify Preview')
plt.tight_layout()
plt.savefig(spectrogram_path)
plt.close()

print(f"Spectrogram saved to: {spectrogram_path}")
```