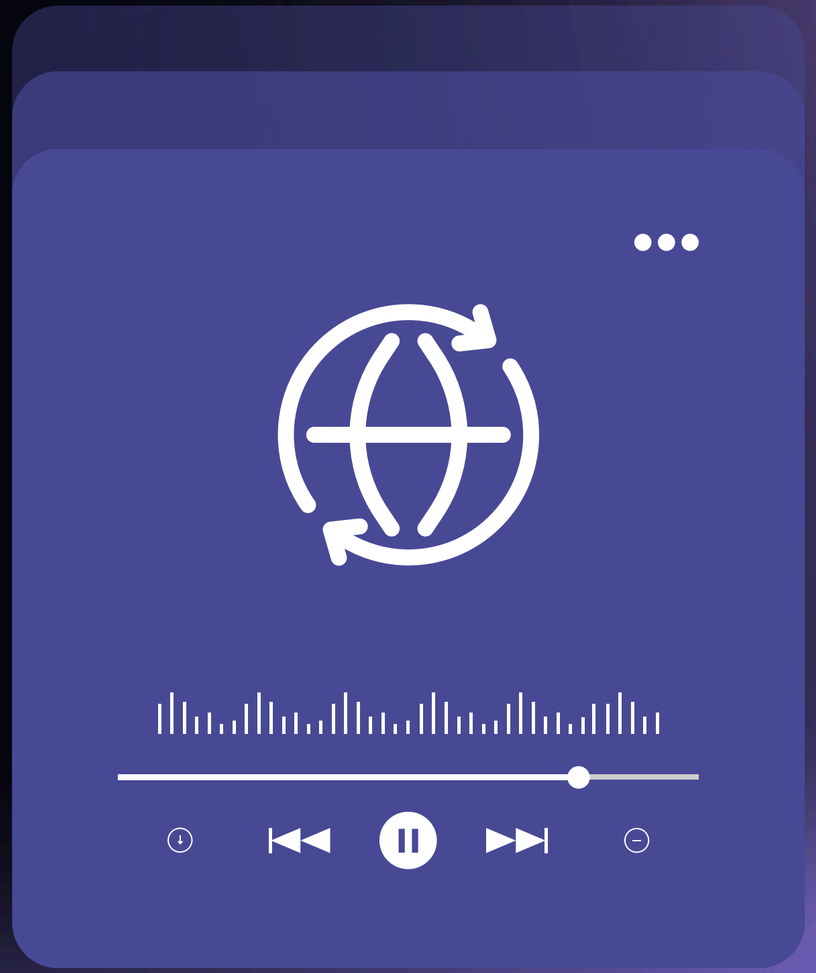
TEAM
NEUROBYTES

Music Recommendation Application

Implementation of Large Language Models and MLOps



Project Overview

The proliferation of digital music platforms has escalated the demand for sophisticated music recommendation systems that not only understand user preferences but also appreciate the complex musicality of songs.

We address this need by deploying a state-of-the-art large-language model that processes a fusion of user interaction metrics and intrinsic song features to deliver personalized music recommendations through MLOp practices.

Approach

Data

The cloud database chosen has data related to:

1. User-to-Song rating/playback information
2. Song-to-Song similarity information based on musicality.

DATASET 1

millionsong_dataset.csv

DATASET 2

music_data.csv

DATASET 3

user_preferences.csv

Approach

Methods

01 **Model Preprocessing & Training**

Get million song subset data song list Get metadata and join the data use genre similarity and genre to train the model on similarity

02 **Exploratory Analysis & Prototyping**

Exploring track and user preference related data

03 **ML Production Cloud Pipline**

AWS RDBS, My SQL

Approach

Methods

06 **Re-training & CI/CD** — Production of reccomnedations

02 **Evaluation with Colab** — Evaluating Model and Accuracy

03 **UI/UX Using Gradio** — Implementing model with user experience using gradio and huggingface as a tool

Experiments

1 Spotify API	2 LastFM API	3 EDA
Testing data related to Spotify API	Connecting User Preferences to the Million Song Dataset and LastFM API	Exploratory Data Analysis of the Synthetic User Preferences

Training


huggingface.co
Training
In [ ]:
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
import numpy as np
# Define the neural network model with Dropout and Batch Normalization
class ImprovedSongRecommender(nn.Module):
def __init__(self, input_size, num_titles):
super(ImprovedSongRecommender, self).__init__()
self.fc1 = nn.Linear(input_size, 128)
self.bn1 = nn.BatchNorm1d(128)
self.fc2 = nn.Linear(128, 256)
self.bn2 = nn.BatchNorm1d(256)
self.fc3 = nn.Linear(256, 128)
self.bn3 = nn.BatchNorm1d(128)
self.output = nn.Linear(128, num_titles)
self.dropout = nn.Dropout(0.5)
def forward(self, x):
x = torch.relu(self.bn1(self.fc1(x)))
x = self.dropout(x)
x = torch.relu(self.bn2(self.fc2(x)))
x = self.dropout(x)
x = torch.relu(self.bn3(self.fc3(x)))
x = self.dropout(x)
x = self.output(x)
return x
# Adjusting input size for the model
input_size = X_train.shape[1] # Number of features in the input
num_unique_titles = len(label_encoders['title'].classes_) # Number of unique
# Initialize the model with the correct input size and output size
model = ImprovedSongRecommender(input_size, num_unique_titles)
# Initialize the optimizer and loss function
optimizer = optim.AdamW(model.parameters(), lr=0.001, weight_decay=0.01)
criterion = nn.CrossEntropyLoss()
# Use a learning rate scheduler
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=3, gamma=0.1)
# Early stopping parameters
patience = 3
min_delta = 0.01
best_val_loss = np.inf
patience_counter = 0
# Function to train the model
def train_model(model, X_train, y_train, X_test, y_test):
global best_val_loss, patience_counter
train_loader = DataLoader(list(zip(X_train.values.astype(float), y_train)),
test_loader = DataLoader(list(zip(X_test.values.astype(float), y_test)),
model.train()
for epoch in range(20): # Increase the number of epochs

Testing


huggingface.co
Testing
In [ ]:
import torch
from joblib import load
# Define the same neural network model
class ImprovedSongRecommender(nn.Module):
def __init__(self, input_size, num_titles):
super(ImprovedSongRecommender, self).__init__()
self.fc1 = nn.Linear(input_size, 128)
self.bn1 = nn.BatchNorm1d(128)
self.fc2 = nn.Linear(128, 256)
self.bn2 = nn.BatchNorm1d(256)
self.fc3 = nn.Linear(256, 128)
self.bn3 = nn.BatchNorm1d(128)
self.output = nn.Linear(128, num_titles)
self.dropout = nn.Dropout(0.5)
def forward(self, x):
x = torch.relu(self.bn1(self.fc1(x)))
x = self.dropout(x)
x = torch.relu(self.bn2(self.fc2(x)))
x = self.dropout(x)
x = torch.relu(self.bn3(self.fc3(x)))
x = self.dropout(x)
x = self.output(x)
return x
# Load the trained model
model_path = '/content/improved_model.pth'
num_unique_titles = 4855 # Update this to match your dataset
model = ImprovedSongRecommender(input_size=2, num_titles=num_unique_titles)
model.load_state_dict(torch.load(model_path, map_location=torch.device('cpu')))
model.eval()
# Load the label encoders and scaler
label_encoders_path = '/content/new_label_encoders.joblib'
scaler_path = '/content/new_scaler.joblib'
label_encoders = load(label_encoders_path)
scaler = load(scaler_path)
# Create a mapping from encoded indices to actual song titles
index_to_song_title = {index: title for index, title in enumerate(label_encode
def encode_input(tags, artist_name):
tags = tags.strip().replace('\n', '')
artist_name = artist_name.strip().replace('\n', '')
try:
encoded_tags = label_encoders['tags'].transform([tags])[0]
except ValueError:
encoded_tags = label_encoders['tags'].transform(['unknown'])[0]
try:
encoded_artist = label_encoders['artist_name'].transform([artist_name]
except ValueError:
encoded_artist = label_encoders['artist_name'].transform(['unknown'])

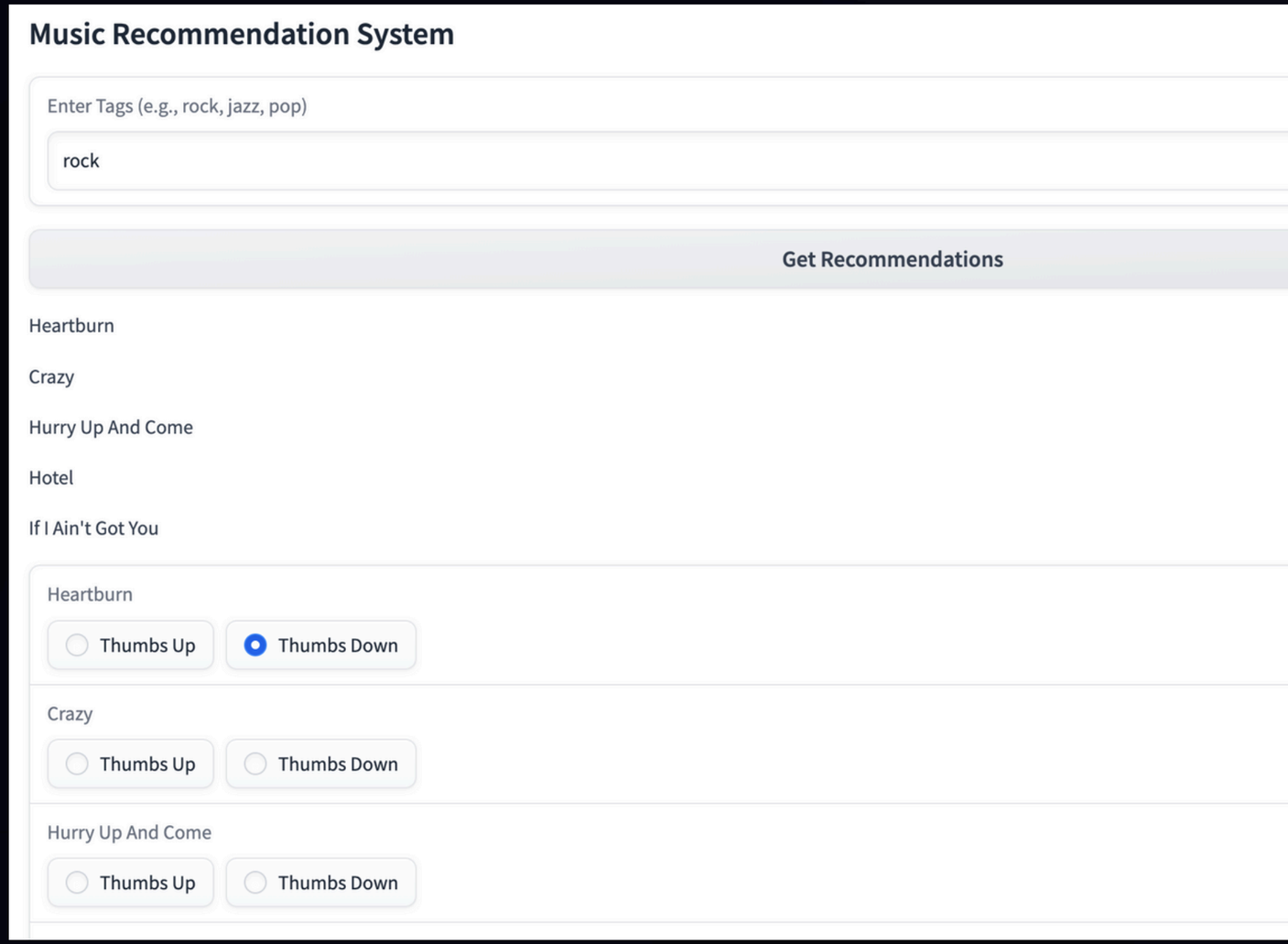
Evaluation


huggingface.co
In [312…
# Split data into features and target
X = df_scaled[['tags', 'artist']]
y = df_scaled['song']
# Split the dataset into training and testing sets
X_valid, X_test, y_valid, y_test = train_test_split(X, y, test_size=0.2, rando
print("Data split into validation and testing sets.")
Data split into validation and testing sets.
In [313…
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
import numpy as np
from sklearn.metrics import accuracy_score
In [314…
valid_loader = DataLoader(list(zip(X_valid.values.astype(float), y_valid)), ba
test_loader = DataLoader(list(zip(X_test.values.astype(float), y_test)), batch
In [315…
valid_accuracy = 0
test_accuracy = 0
for features, labels in valid_loader:
preds = model(features.float().detach())
# Get the predicted class (the one with the highest score)
_, predicted_class = torch.max(preds, 1)
# Convert to numpy arrays
predicted_class_np = predicted_class.numpy()
labels_np = labels.numpy()
# Calculate accuracy
accuracy = accuracy_score(labels_np, predicted_class_np)
valid_accuracy += accuracy
for features, labels in test_loader:
preds = model(features.float())
# Get the predicted class (the one with the highest score)
_, predicted_class = torch.max(preds, 1)
# Convert to numpy arrays
predicted_class_np = predicted_class.numpy()
labels_np = labels.numpy()
# Calculate accuracy
accuracy = accuracy_score(labels_np, predicted_class_np)
test_accuracy += accuracy
In [316…
print('The loss of the model on the unseen validation dataset is: ', valid_ac
print('The loss of the model on the unseen test dataset is: ', test_accuracy)
The loss of the model on the unseen validation dataset is: 2.0
The loss of the model on the unseen test dataset is: 0.0
In [316…

Implementation

Gradio Demo

link



Conclusion

Future work

We successfully produced a music recommendation system using traditional MLOp practices. For future work, we could potentially scale this larger and implement the system with existing music listening platforms.

TEAM
NEUROBYTES

Thank you for listening

For any questions feel free to contact our team.