Music Genre Classification from Lyrics (English & Hindi)

A Project Report

Submitted in partial fulfilment of the

Requirements for the award of the Degree of

MASTER OF SCIENCE (DATA ANALYTICS)

Name of the Student: Ashwin Suresh

Seat Number: 6855

Under the esteemed guidance of

Mr. OMKAR SHERKHANE

Designation: Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE

MAHATMA EDUCATION SOCIETY'S

PILLAI COLLEGE OF ARTS, COMMERCE AND SCIENCE (Autonomous),

NEW PANVEL, 410206, MAHARASHTRA

(Affiliated to University of Mumbai)

2024-2025

MAHATMA EDUCATION SOCIETY'S

PILLAI COLLEGE OF ARTS, COMMERCE AND SCIENCE (Autonomous) NEW PANVEL, 410206, MAHARASHTRA

(Affiliated to University of Mumbai)

DEPARTMENT OF COMPUTER SCIENCE

CERTIFICATE

This is to certify that the project entitled **"Music Genre Classification from Lyrics (English & Hindi)"** is Bonafide work of **Ashwin Suresh** bearing Roll. No **6855** submitted in fulfilment for the completion of MSc. degree in Data Analytics of University of Mumbai.

[illegible] [illegible]

Internal Guide **Co-Ordinator**

Date: **College seal** **External Examiner**

ACKNOWLEDGEMENT

I Mr. Ashwin Suresh student of PILLAI COLLEGE OF ARTS, COMMERCE & SCIENCE (AUTONOMOUS), NEW PANVEL would like to express my sincere gratitude towards our college's Computer Science Department.

I would like to thank our Coordinator Mr. Omkar Sherkhane for his constant support during this project. The project would have not been completed without the dedication, creativity and the enthusiasm my family provided me.

Yours faithfully,

Ashwin Suresh

(Final Year Data Analytics)

DECLARATION

I hereby, declare that the project entitled, "**Music Genre Classification from Lyrics (English & Hindi)**" done at Pillai College of Arts, Commerce & Science (Autonomous), New Panvel, has not been in any case duplicated to submit to any other university for the award of any degree. To the best of my knowledge other than me, no one has submitted to any other university.

The project is done in fulfillment of the requirements for the award of degree of MASTER OF SCIENCE (DATA ANALYTICS) to be submitted as a final semester project as part of our curriculum.

Signature of the Student

ABSTRACT

This project, titled "**Music Genre Classification from Lyrics (English & Hindi)**", explores the application of deep learning techniques for categorizing music genres based solely on song lyrics in two languages: English and Hindi. The approach involves two distinct modules, each designed to handle language-specific preprocessing and model training. For the English module, text is cleaned by removing punctuation, stopwords, and extraneous characters, then tokenized and padded before training a bidirectional LSTM model using pre-trained GloVe embeddings. Similarly, the Hindi module employs tailored text cleaning—incorporating removal of Hindi-specific punctuation and stopwords—followed by tokenization and training of an LSTM model with dedicated Hindi word embeddings.

Both models are integrated into a unified Flask-based web interface that provides an intuitive and responsive experience for users. AJAX is used to facilitate asynchronous predictions, ensuring a smooth user interaction without full-page reloads.

The system demonstrates promising results in accurately classifying music genres and highlights the potential of deep learning in natural language processing for music analytics.

Index

Chapter 1

Introduction

Music is an integral part of modern **culture**, and with the exponential growth of **digital music libraries**, the ability to automatically classify songs into **genres** has become increasingly valuable. This project, titled **"Music Genre Classification from Lyrics (English & Hindi)"**, addresses the challenge of categorizing music based solely on its **lyrical content**—a task that combines the fields of **natural language processing (NLP)** and **deep learning**.

The primary goal of this project is to develop a robust system that accurately predicts the **genre** of a song from its **lyrics**. The system is designed to handle two languages—**English** and **Hindi**—by employing language-specific **preprocessing pipelines** and dedicated **deep learning models**. This **bilingual approach** not only highlights the versatility of modern NLP techniques but also caters to diverse music collections spanning different linguistic traditions.

For the **English module**, the project utilizes extensive **data cleaning** methods that involve converting text to **lowercase**, removing **punctuation** and extraneous text (such as content within **brackets**), and filtering out common **stopwords** using **NLTK**'s predefined lists. Cleaned lyrics are then **tokenized** and **padded** to a fixed length. To capture the **semantic meaning** of words, the model leverages pre-trained **GloVe embeddings** (100-dimensional vectors), and a **bidirectional LSTM network** is trained on this processed data to classify songs into genres.

The **Hindi module** presents additional challenges, such as handling unique punctuation marks (e.g., the Hindi **danda** "।" and **double danda** "॥") and incorporating language-specific **stopwords**. A custom preprocessing pipeline cleans the raw Hindi lyrics, and a **tokenizer** is used to convert these into sequences of integers. For word representations, a set of pre-trained **Hindi embeddings** (**cc.hi.300.vec**, 300-dimensional vectors) is utilized. The model architecture, built with two stacked **bidirectional LSTM layers** and **dropout** for regularization, is optimized to learn complex patterns in Hindi text for accurate genre prediction.

To ensure usability, the project integrates both models into a unified **web interface** built with **Flask**, **Bootstrap**, and **jQuery**. This interface features two distinct sections for **English** and **Hindi** predictions, providing an intuitive user experience. **AJAX** is employed to handle form submissions **asynchronously**, which means predictions are delivered instantly without the need to reload the entire page.

In summary, this project demonstrates a comprehensive approach to **music genre classification** by combining advanced deep learning techniques with practical web development. It not only addresses the technical challenges of processing **multilingual text data** but also provides a scalable and interactive tool that can serve as the backbone for more sophisticated **music recommendation systems**. Future enhancements may include experimenting with more complex neural architectures, integrating personalized recommendation engines, and expanding the system to include additional languages or music metadata.

Chapter 2

Review of Literature

2.1 Background

The advent of digital music streaming and the explosive growth of music libraries have dramatically transformed the way people discover and consume music. With millions of songs available on platforms like Spotify, Apple Music, and YouTube, the need for automated methods to organize, classify, and recommend music has never been more critical. **Music genre classification**—categorizing songs based on attributes such as lyrics, melody, and rhythm—is essential for creating personalized user experiences, streamlining digital library management, and enabling data-driven decision-making for the music industry.

Recent advances in **Natural Language Processing (NLP)** and **deep learning**, particularly through Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) architectures, have significantly improved the ability to analyze sequential data like song lyrics. Pre-trained word embeddings (such as GloVe for English and domain-specific embeddings for Hindi) allow models to capture the semantic and syntactic nuances of language. This project leverages these technologies to build an automated system that classifies music genres from lyrics in both **English** and **Hindi**, addressing the increasing demand for multilingual solutions in global digital music platforms.

2.2 Objectives

The primary objectives of this project are as follows:

1. **Develop Robust Preprocessing Pipelines:**
    1. Clean and normalize song lyrics for English and Hindi.
    2. Remove noise such as punctuation, extraneous text (e.g., content within brackets), and stopwords.
2. **Build and Train Deep Learning Models:**
    1. Train a bidirectional LSTM model for English lyrics using GloVe embeddings.
    2. Train a separate bidirectional LSTM model for Hindi lyrics using pre-trained Hindi embeddings (cc.hi.300.vec).
3. **Integrate Models into a Unified Interface:**
    1. Create a user-friendly web interface with Flask and AJAX to allow real-time predictions.
    2. Display predictions for both English and Hindi lyrics in a visually appealing manner.
4. **Demonstrate the Feasibility of Multilingual Genre Classification:**
    1. Evaluate model performance using standard metrics.
    2. Provide insights into how deep learning can bridge language barriers in music classification.

2.3 Purpose

- **Automating Genre Classification:**

By developing deep learning models that classify songs based solely on their lyrics, the project eliminates the need for manual tagging, ensuring a scalable solution for managing vast music libraries.

- **Enhancing Music Discovery:**

With accurate genre predictions for both English and Hindi lyrics, the system enables personalized music recommendations. This allows streaming platforms and digital libraries to offer tailored playlists that match individual listener preferences, thereby improving the overall user experience.

- **Advancing Multilingual NLP Applications:**

By supporting two languages, the project demonstrates how advanced natural language processing techniques—such as bidirectional LSTM networks and pre-trained word embeddings—can be applied to multilingual datasets. This not only fills a gap in current research but also sets the foundation for future systems in diverse linguistic markets.

- **Providing Industry Insights:**

The automated classification system offers valuable insights into music trends and listener preferences. These insights can inform marketing strategies, playlist curation, and content creation for record labels, thereby bridging the gap between data analytics and music production.

2.4 Scope of the Project

- **Digital Music Streaming:**

Integration into music streaming services to automate playlist curation and recommendation systems based on genre predictions.

- **Music Library Management:**

Assisting music distributors and digital libraries in organizing and categorizing extensive catalogs, leading to improved searchability and user engagement.

- **Content Analytics:**

Providing analytical insights for record labels and artists by understanding genre trends, which can influence promotional strategies and new releases.

- **Multilingual Expansion:**

Serving as a foundation for developing similar classification systems in other languages, thereby supporting global and culturally diverse music markets.

- **Research and Development:**

Laying the groundwork for further exploration into advanced neural network architectures and the incorporation of additional modalities (such as audio features) for comprehensive music analysis.

2.5 Applicability

- **Digital Music Platforms:**

The system can be integrated into music streaming services and digital libraries to automatically classify and organize songs by genre.

- **Enhanced Music Recommendation:**

By accurately predicting genres from lyrics, the project supports the development of personalized recommendation systems.

- **Academic Research:**

It serves as a practical example of applying deep learning and NLP techniques to multilingual text classification, providing insights for further research.

- **Industry Utility:**

The solution assists in efficient cataloging and analysis of music content, benefiting record labels, curators, and marketing teams.

Chapter 3

System Planning: Survey of Technologies

3.1 System Architecture

The system architecture is designed to ensure modularity, scalability, and ease of maintenance. The architecture is divided into three layers:

3.1.1. Presentation Layer (Frontend)

The presentation layer is responsible for interacting with the user. It gathers user input (lyrics), displays the predictions, and provides an intuitive interface. Key technologies used in this layer include:

- **HTML** for structuring the web pages.
- **CSS and Bootstrap** for responsive, modern styling.
- **JavaScript/jQuery** for interactivity and AJAX-based asynchronous communication.

Main Functions:

- Collecting user input (lyrics) for both English and Hindi.
- Displaying predictions with dynamic updates (via AJAX).
- Offering a clean, intuitive UI that enhances user experience.

3.1.2. Application Layer (Backend)

The application layer handles business logic, model inference, and data processing. It is implemented using Python with the Flask framework. This layer serves as the bridge between the user interface and the pre-trained deep learning models.

Technologies and Libraries:

- **Flask:** For developing the web application and APIs.
- **TensorFlow/Keras:** For loading and running the LSTM models.
- **NumPy and Pickle:** For handling model artifacts and numerical computations.

Key Responsibilities:

- Loading pre-trained models, tokenizers, and label maps from file storage.
- Processing the input lyrics (tokenization and padding).
- Generating predictions by invoking the respective deep learning model (for English or Hindi).
- Sending JSON responses back to the frontend through AJAX calls.

3.1.3. Data Layer

This project uses file-based storage for its data:

- **Model Artifacts:** Pre-trained models, tokenizers, and label maps for both English and Hindi are stored as files (e.g., `.h5` and `.pkl`).

- **Embedding Files:** Pre-trained embeddings (GloVe for English and cc.hi.300.vec for Hindi) are stored in plain text files.

While no dedicated database is used, all data is managed effectively through the file system, ensuring fast access and minimal overhead.

3.2 Data Flow

Data flows through the system in the following process:

1. **User Input:**
   - The user enters lyrics into the web interface and selects the language (English or Hindi).
2. **Request Transmission:**
   - The input is sent to the backend asynchronously via AJAX POST requests.
3. **Data Processing:**
   - The backend processes the lyrics (converts to lowercase, tokenizes, and pads) and passes them to the appropriate LSTM model.
   - The model generates a genre prediction.
4. **Response Delivery:**
   - The backend sends the predicted genre back to the frontend as a JSON response.
   - The frontend displays the result with appropriate styling and color coding.

3.3 Technologies Used

The system leverages a combination of modern technologies to ensure robustness and efficiency:

3.3.1. Frontend Technologies:

- **HTML/CSS:** For creating and styling the user interface.

- **Bootstrap:** To ensure the interface is responsive and visually appealing.

- **JavaScript/jQuery:** To handle asynchronous requests and dynamic content updates.

3.3.2. Backend Technologies:

- **Python with Flask:** Provides the web framework to build APIs and handle backend logic.

- **TensorFlow/Keras:** Used for loading and running the LSTM models for genre classification.

- **NumPy and Pickle:** Employed for numerical operations and managing model artifacts.

3.3.3. Data Storage:

File-based Storage:
Model artifacts (e.g., .h5 for models, .pkl for tokenizers and label maps) and embedding files (e.g., GloVe and cc.hi.300.vec) are stored locally, providing a lightweight alternative to a full-fledged database.

3.4 Integration and Deployment

3.4.1. Integration:

- **AJAX-based Communication:**
The frontend uses AJAX to communicate with the Flask backend, allowing for real-time predictions without page reloads.

- **Model Loading:**
Pre-trained models and related artifacts are loaded once at application startup, ensuring efficient inference.

3.4.2. Deployment:

Local Deployment:

The application is designed to run locally using Flask’s development server.

3.5 Security and User Experience

3.5.1. Security:

- **Secure Communication:**
When deployed, the application can be configured to run over HTTPS to ensure secure data transmission.

- **Input Validation:**
The backend validates user inputs to prevent potential injection attacks and other security vulnerabilities.

3.5.2. User Experience:

- **Responsive Design:**
The frontend is built using Bootstrap to ensure it is accessible on both desktop and mobile devices.

- **Real-Time Feedback:**
AJAX is used to provide instant feedback, improving the overall responsiveness of the application.

- **Intuitive Interface:**
The design emphasizes simplicity and ease of use, allowing users to quickly input lyrics and view predictions.

Chapter 4

Methodology

The methodology adopted for developing the AI-enabled fitness web app follows a systematic and structured approach to ensure accuracy, reliability, and user satisfaction. The development process is divided into the following key phases:

1. Research and Analysis Phase

Objective:
To understand existing resources, identify gaps, and establish a clear plan for developing a music genre classification system that handles both English and Hindi lyrics.

Activities:

- **Dataset Survey:**
    - **English Lyrics:**
      Researched and collected various publicly available datasets containing English lyrics with genre labels. These datasets were merged to create a larger, more comprehensive corpus.
    - **Hindi Lyrics:**
      Conducted an extensive search for dedicated Hindi lyric datasets. Due to limited availability, we extracted Hindi song lyrics directly (since APIs for copyrighted content were unavailable) and created our own dataset.
- **Resource Identification:**
    - Identified **GloVe embeddings (6B 100d)** for English to capture semantic relationships.
    - For Hindi, researched online and obtained a list of Hindi stopwords, compiling them into a file (`stopwords-hi.txt`), and found pre-trained **cc.hi.300.vec** embeddings (300-dimensional) tailored for Hindi.
- **Gap Analysis:**
    - Analyzed existing literature and solutions in music genre classification, identifying the need for multilingual support and robust deep learning models.

2. Data Collection and Preprocessing Phase

Objective:
To gather and clean relevant datasets for both English and Hindi lyrics, preparing them for model training.

Activities:

- **Data Collection:**

- **English:**
  Combined multiple publicly available datasets to build a large corpus of English song lyrics with genre labels.
- **Hindi:**
  Created a custom dataset by manually extracting Hindi song lyrics due to the unavailability of dedicated datasets.

- **Data Preprocessing:**
  - **English Preprocessing:**
    - Convert text to lowercase.
    - Remove punctuation, bracketed text, and common English stopwords (using NLTK).
    - Save the cleaned data to a new CSV file.
  - **Hindi Preprocessing:**
    - Load a custom Hindi stopwords file (`stopwords-hi.txt`).
    - Remove punctuation including Hindi-specific symbols like the danda "।" and double danda "॥".
    - Convert text to lowercase and filter out stopwords.
    - Save the processed lyrics into a cleaned dataset CSV file.

Outcome:
Two structured, cleaned datasets ready for model training—one for English lyrics (e.g., `lyrics_cleaneddd.csv`) and one for Hindi lyrics (e.g., `songs_happysadcleaned.csv`).

3. Model Development Phase

Objective:
To build and train deep learning models that accurately classify music genres from lyrics in both languages.

Steps:

1. **Model Selection:**
   - **English Model:**
     A bidirectional LSTM model using pre-trained **GloVe 6B 100d** embeddings.
   - **Hindi Model:**
     A bidirectional LSTM model using pre-trained **cc.hi.300.vec** embeddings (300-dimensional).
2. **Model Training:**
   - **Tokenization & Padding:**
     Convert the cleaned lyrics into sequences using Keras's `Tokenizer` and pad the sequences to fixed lengths (427 tokens for English; 224 tokens for Hindi based on dataset statistics).
   - **Label Encoding:**
     Map genres to numerical values and one-hot encode them.
   - **Embedding Matrix Construction:**
     Build embedding matrices from the respective embedding files to initialize the embedding layers.
   - **Architecture:**

        - For English:
          A single bidirectional LSTM layer followed by dropout and dense layers.
        - For Hindi:
          Two stacked bidirectional LSTM layers to capture more complex patterns, along with dropout and dense layers.
    - **Training Strategy:**
      Use early stopping and learning rate reduction to optimize training, and evaluate models on a validation set.
3. **Model Integration:**
    - Save the trained models, tokenizers, and label maps as artifacts for later use in the web interface.

Outcome:
Two optimized models—one for English and one for Hindi—that are capable of classifying song genres accurately from lyrics.

4. System Design Phase

Objective:
To design a scalable and user-friendly architecture that integrates the deep learning models with a web-based interface.

Components:

- **Frontend:**
    - **Technologies:** HTML, CSS, Bootstrap, JavaScript, and jQuery.
    - **Functionality:**
        - Provides separate sections for English and Hindi predictions.
        - Uses AJAX for asynchronous communication to deliver real-time genre predictions.
- **Backend:**
    - **Technologies:** Python with Flask, TensorFlow/Keras, NumPy, and Pickle.
    - **Functionality:**
        - Loads pre-trained models, tokenizers, and label maps.
        - Processes input lyrics (tokenization, padding) and invokes the respective model for predictions.
        - Returns results as JSON responses to the frontend.
- **Data Layer:**
    - **File-Based Storage:**
        - Stores model artifacts (e.g., `.h5` and `.pkl` files) and embedding files.
        - Contains cleaned datasets for both languages.

Outcome:
A well-defined architecture that supports seamless integration of multilingual deep learning models with a responsive and interactive web interface.

5. Implementation Phase

Objective:
To develop the core functionalities and integrate all components into a functioning system.

Activities:

- **Frontend Development:**
  - Build a responsive UI using Bootstrap.
  - Implement two separate prediction forms for English and Hindi lyrics.
  - Use AJAX for submitting data and displaying predictions without full page reloads.
- **Backend Development:**
  - Develop Flask endpoints to handle form submissions.
  - Load and invoke the appropriate deep learning model based on the language selection.
  - Return predictions as JSON responses.
- **Integration:**
  - Connect the frontend and backend to ensure smooth data flow and real-time feedback.

Outcome:
A fully functional web application that provides music genre classification from lyrics in English and Hindi, with a polished and responsive user interface.

6. Testing and Validation Phase

Objective:
To ensure the accuracy, reliability, and usability of the system.

Activities:

- **Unit Testing:**
  Test individual components (e.g., preprocessing functions, model inference) for correct behavior.
- **Integration Testing:**
  Verify that the entire pipeline—from data input to prediction display—works seamlessly.
- **Performance Testing:**
  Measure response times for AJAX calls and overall system throughput.
- **User Testing:**
  Conduct user acceptance testing (UAT) to gather feedback on interface usability and prediction accuracy.

Outcome:
A validated system with verified performance and user satisfaction, ensuring readiness for deployment.

7. Deployment Phase

Objective:
To make the system accessible to users in a production environment.

Activities:

- **Local Deployment:**
  Initially deploy the Flask application on a local development server.

Outcome:
A live, accessible web application that reliably classifies music genres from lyrics in both English and Hindi.

8. Maintenance and Upgradation Phase

Objective:
To keep the application up-to-date and improve its functionality.

Activities:

- Monitor application performance and user feedback.
- Fix bugs and improve system efficiency based on performance data.
- Upgrade models periodically to ensure accuracy.
- Add new features and improvements as needed.

Outcome:
An up-to-date application with enhanced performance and new features based on user feedback.

Chapter 5

Problem Definition and Proposed Solution

5.1.Problem Definition

In today's digital era, the explosion of music content poses significant challenges for music organization and discovery. Despite the availability of numerous music streaming platforms, users and industry professionals face several issues:

- **Manual Tagging and Inconsistencies:**
    - **Labor-Intensive Process:**
      Manually tagging songs by genre is time-consuming and prone to errors.
    - **Subjectivity:**
      Human-generated labels may vary, leading to inconsistencies in genre classification.
- **Limited Multilingual Support:**
    - **Focus on Single Language:**
      Most existing systems concentrate on English lyrics, neglecting other languages such as Hindi.
    - **Cultural Nuances:**
      Different languages and cultural contexts affect the interpretation of lyrics, complicating accurate genre classification.
- **Semantic Complexity of Lyrics:**
    - **Ambiguity in Language:**
      Lyrics often use metaphors, idioms, and ambiguous expressions that are challenging for traditional text processing methods.
    - **Contextual Dependencies:**
      The meaning of a word or phrase can vary depending on its context within the lyrics, making classification difficult.
- **Scalability and Automation:**
    - **Large-Scale Music Libraries:**
      With millions of songs available digitally, there is a need for automated systems that can efficiently and accurately classify genres.
    - **Real-Time Processing:**
      Users expect immediate feedback and recommendations, which current manual or semi-automated systems struggle to provide.

5.2. Proposed Solution

To address the challenges identified, this project proposes the development of an automated music genre classification system that leverages deep learning techniques and supports both English and Hindi lyrics. The key features of the proposed solution include:

1. **Automated Genre Classification:**
   - **Deep Learning Models:**
     Utilize bidirectional LSTM networks trained on preprocessed lyrics to automatically predict the genre of a song.
   - **Multilingual Capability:**
     Develop separate models for English and Hindi to cater to the diverse nature of music content, using **GloVe embeddings** for English and **cc.hi.300.vec** embeddings for Hindi.
2. **Comprehensive Data Processing Pipeline:**
   - **Data Collection and Integration:**
     Combine multiple publicly available datasets for English lyrics and create a custom dataset for Hindi lyrics due to limited availability.
   - **Robust Preprocessing:**
     Implement language-specific cleaning methods (removing punctuation, stopwords, etc.) to prepare data for model training.
3. **User-Friendly Web Interface:**
   - **Responsive Design:**
     Build a unified web interface using Flask, Bootstrap, and jQuery that offers separate sections for English and Hindi predictions.
   - **Asynchronous Processing:**
     Use AJAX to provide real-time predictions without page reloads, enhancing the user experience.
4. **Scalable and Maintainable Architecture:**
   - **Modular Design:**
     Organize the system into distinct layers—frontend, backend, and data storage—to ensure ease of maintenance and scalability.
   - **File-Based Storage:**
     Manage model artifacts (e.g., `.h5` and `.pkl` files) and embedding resources efficiently without the need for a complex database.

Chapter 6

System Design

Program Code

English Lyrics Preprocessing
preprocessing.py

```
import pandas as pd
import re
import string
import nltk
from nltk.corpus import stopwords

# Download stopwords if not already downloaded
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))

def clean_text(text):
    if pd.isna(text):
        return ""
    text = text.lower()  # Convert to lowercase
    text = re.sub(r'\[.*?\]', '', text)  # Remove text inside square brackets
    text = re.sub(f"[{re.escape(string.punctuation)}]", '', text)  # Remove punctuation
    words = [word for word in text.split() if word not in stop_words]  # Remove stopwords
    return " ".join(words)

def preprocess_data(input_csv='/content/drive/MyDrive/data_genre.csv',
output_csv='/content/drive/MyDrive/lyrics_cleaneddd.csv'):
    # Load the dataset (expects columns 'Genre' and 'Lyrics')
    df = pd.read_csv(input_csv)
    # Drop rows missing either Lyrics or Genre
    df.dropna(subset=['Lyrics', 'Genre'], inplace=True)
    # Clean the Lyrics column
    df['cleaned_lyrics'] = df['Lyrics'].apply(clean_text)
    # Save the cleaned data
    df.to_csv(output_csv, index=False)
    print(f"Preprocessed data saved to {output_csv}")

if __name__ == '__main__':
    preprocess_data()
```

English LSTM Model Training
lstm_model_training.py

```
import pandas as pd
import numpy as np
import pickle
import os
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout, Bidirectional
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from sklearn.model_selection import train_test_split

# Hyperparameters
MAX_NUM_WORDS = 30000
MAX_SEQUENCE_LENGTH = 427  # 95th percentile of token counts
EMBEDDING_DIM = 100        # GloVe 6B 100d
VALIDATION_SPLIT = 0.2
EPOCHS = 30
BATCH_SIZE = 128

def load_data(input_csv='/content/drive/MyDrive/lyrics_cleaneddd.csv'):
    df = pd.read_csv(input_csv)
    df['cleaned_lyrics'] = df['cleaned_lyrics'].astype(str)
    texts = df['cleaned_lyrics'].tolist()
    labels = df['Genre'].tolist()
    return texts, labels

def prepare_labels(labels):
    genres = sorted(list(set(labels)))
    label_map = {genre: idx for idx, genre in enumerate(genres)}
    y = [label_map[label] for label in labels]
    y = to_categorical(np.asarray(y))
    return y, label_map

def load_glove_embeddings(glove_dir='/content/drive/MyDrive/glove',
glove_file='glove.6B.100d.txt'):
    embeddings_index = {}
    glove_path = os.path.join(glove_dir, glove_file)
    with open(glove_path, encoding='utf8') as f:
        for line in f:
            values = line.split()
            word = values[0]
            coefs = np.asarray(values[1:], dtype='float32')
        embeddings_index[word] = coefs
    print(f"Found {len(embeddings_index)} word vectors in GloVe.")
    return embeddings_index

def create_embedding_matrix(word_index, embeddings_index,
max_num_words=MAX_NUM_WORDS, embedding_dim=EMBEDDING_DIM):
    num_words = min(max_num_words, len(word_index) + 1)
    embedding_matrix = np.zeros((num_words, embedding_dim))
    for word, i in word_index.items():
        if i >= max_num_words:
            continue
        embedding_vector = embeddings_index.get(word)
        if embedding_vector is not None:
            embedding_matrix[i] = embedding_vector
    return embedding_matrix

def build_lstm_model(embedding_matrix, num_classes):
    model = Sequential()
    model.add(Embedding(input_dim=MAX_NUM_WORDS,
                        output_dim=EMBEDDING_DIM,
                        weights=[embedding_matrix],
                        input_length=MAX_SEQUENCE_LENGTH,
                        trainable=True))
    model.add(Bidirectional(LSTM(64, dropout=0.5, recurrent_dropout=0.5)))
    model.add(Dropout(0.5))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(num_classes, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    model.summary()
    return model

def train_lstm_model():
    texts, labels = load_data()
    y, label_map = prepare_labels(labels)
    tokenizer = Tokenizer(num_words=MAX_NUM_WORDS)
    tokenizer.fit_on_texts(texts)
    sequences = tokenizer.texts_to_sequences(texts)
    word_index = tokenizer.word_index
    print(f"Found {len(word_index)} unique tokens.")
    data = pad_sequences(sequences, maxlen=MAX_SEQUENCE_LENGTH)
    X_train, X_val, y_train, y_val = train_test_split(data, y, test_size=VALIDATION_SPLIT,
random_state=42)
    num_classes = y.shape[1]
    embeddings_index = load_glove_embeddings()
    embedding_matrix = create_embedding_matrix(word_index, embeddings_index)
    model = build_lstm_model(embedding_matrix, num_classes)
    early_stop = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
    reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=2, min_lr=1e-5)
    model.fit(X_train, y_train, batch_size=BATCH_SIZE, epochs=EPOCHS,
              validation_data=(X_val, y_val), callbacks=[early_stop, reduce_lr])
    loss, accuracy = model.evaluate(X_val, y_val, verbose=0)
    print(f"LSTM Model Validation Accuracy: {accuracy:.4f}")
    model.save('/content/drive/MyDrive/lstm_genre_modelbest.h5')
    with open('/content/drive/MyDrive/tokenizerbest.pkl', 'wb') as f:
        pickle.dump(tokenizer, f)
    with open('/content/drive/MyDrive/label_mapbest.pkl', 'wb') as f:
        pickle.dump(label_map, f)
    print("Model, tokenizer, and label map saved.")

if __name__ == '__main__':
    train_lstm_model()
```

Hindi Lyrics Preprocessing
preprocessing_hindi.py

```
import pandas as pd
import re
import string

def load_hindi_stopwords(filepath='/content/drive/MyDrive/stopwords-hi.txt'):
    with open(filepath, encoding='utf-8') as f:
        stopwords = f.read().splitlines()
    return set(stopwords)

hindi_stop_words = load_hindi_stopwords()

def clean_text(text):
    if pd.isna(text):
        return ""
    text = text.lower()  # Convert to lowercase
    text = re.sub(r'\[.*?\]', '', text)  # Remove text inside square brackets
    # Remove punctuation: combine Python's punctuation with Hindi danda "।" and double
danda "॥"
    punctuation = string.punctuation + "।" + "॥"
    text = re.sub(f"[{re.escape(punctuation)}]", '', text)
    # Remove stopwords
    words = [word for word in text.split() if word not in hindi_stop_words]
    return " ".join(words)
```

```
def preprocess_data(input_csv='/content/drive/MyDrive/songshappy_sad.csv',
output_csv='/content/drive/MyDrive/songs_happysadcleaned.csv'):
    df = pd.read_csv(input_csv)
    df.dropna(subset=['Lyrics', 'Genre'], inplace=True)
    df['cleaned_lyrics'] = df['Lyrics'].apply(clean_text)
    df.to_csv(output_csv, index=False)
    print(f"Preprocessed data saved to {output_csv}")

if __name__ == '__main__':
    preprocess_data()
```

Hindi LSTM Model Training
lstm_model_hindi.py

```
import pandas as pd
import numpy as np
import pickle
import os
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout, Bidirectional
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from sklearn.model_selection import train_test_split

MAX_NUM_WORDS = 3317        # Vocabulary size from the Hindi dataset
MAX_SEQUENCE_LENGTH = 224   # 95th percentile of sequence lengths
EMBEDDING_DIM = 300         # Using cc.hi.300.vec embeddings
VALIDATION_SPLIT = 0.2
EPOCHS = 30
BATCH_SIZE = 128

def load_data(input_csv='/content/drive/MyDrive/songs_happysadcleaned.csv'):
    df = pd.read_csv(input_csv)
    df['cleaned_lyrics'] = df['cleaned_lyrics'].astype(str)
    texts = df['cleaned_lyrics'].tolist()
    labels = df['Genre'].tolist()
    return texts, labels

def prepare_labels(labels):
    genres = sorted(list(set(labels)))
    label_map = {genre: idx for idx, genre in enumerate(genres)}
    y = [label_map[label] for label in labels]
    y = to_categorical(np.asarray(y))
    return y, label_map

def
load_hindi_embeddings(embedding_file='/content/drive/MyDrive/cc.hi.300.vec/cc.hi.300.ve
c'):
    embeddings_index = {}
    with open(embedding_file, encoding='utf8') as f:
        first_line = f.readline().strip().split()
        if len(first_line) == 2:
            # Header found; skip processing this line.
            pass
        else:
            word = first_line[0]
            coefs = np.asarray(first_line[1:], dtype='float32')
            embeddings_index[word] = coefs
        for line in f:
            values = line.rstrip().split()
            word = values[0]
            coefs = np.asarray(values[1:], dtype='float32')
            embeddings_index[word] = coefs
    print(f"Found {len(embeddings_index)} word vectors in the Hindi embeddings file.")
    return embeddings_index

def create_embedding_matrix(word_index, embeddings_index,
max_num_words=MAX_NUM_WORDS, embedding_dim=EMBEDDING_DIM):
    num_words = min(max_num_words, len(word_index) + 1)
    embedding_matrix = np.zeros((num_words, embedding_dim))
    for word, i in word_index.items():
        if i >= max_num_words:
            continue
        embedding_vector = embeddings_index.get(word)
        if embedding_vector is not None:
            embedding_matrix[i] = embedding_vector
    return embedding_matrix

def build_lstm_model(embedding_matrix, num_classes):
    model = Sequential()
    model.add(Embedding(input_dim=MAX_NUM_WORDS,
                        output_dim=EMBEDDING_DIM,
                        weights=[embedding_matrix],
                        input_length=MAX_SEQUENCE_LENGTH,
                        trainable=True))
    model.add(Bidirectional(LSTM(64, dropout=0.5, recurrent_dropout=0.5,
return_sequences=True)))
    model.add(Bidirectional(LSTM(32, dropout=0.5, recurrent_dropout=0.5)))
    model.add(Dropout(0.5))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(num_classes, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    model.summary()
    return model

def train_lstm_model():
    texts, labels = load_data()
    y, label_map = prepare_labels(labels)
    tokenizer = Tokenizer(num_words=MAX_NUM_WORDS)
    tokenizer.fit_on_texts(texts)
    sequences = tokenizer.texts_to_sequences(texts)
    word_index = tokenizer.word_index
    print(f"Found {len(word_index)} unique tokens.")
    data = pad_sequences(sequences, maxlen=MAX_SEQUENCE_LENGTH)
    X_train, X_val, y_train, y_val = train_test_split(data, y, test_size=VALIDATION_SPLIT,
random_state=42)
    num_classes = y.shape[1]
    embeddings_index = load_hindi_embeddings()
    embedding_matrix = create_embedding_matrix(word_index, embeddings_index)
    model = build_lstm_model(embedding_matrix, num_classes)
    early_stop = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
    reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=2, min_lr=1e-5)
    model.fit(X_train, y_train, batch_size=BATCH_SIZE, epochs=EPOCHS,
              validation_data=(X_val, y_val), callbacks=[early_stop, reduce_lr])
    loss, accuracy = model.evaluate(X_val, y_val, verbose=0)
    print(f"LSTM Model Validation Accuracy: {accuracy:.4f}")
    model.save('/content/drive/MyDrive/lstm_genre_model_hindi.h5')
    with open('/content/drive/MyDrive/tokenizer_hindi.pkl', 'wb') as f:
        pickle.dump(tokenizer, f)
    with open('/content/drive/MyDrive/label_map_hindi.pkl', 'wb') as f:
        pickle.dump(label_map, f)
    print("Model, tokenizer, and label map saved.")

if __name__ == '__main__':
    train_lstm_model()
```

Web Interface for Genre Classification
app.py

```
from flask import Flask, render_template, request, jsonify
import pickle
import numpy as np
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

```
app = Flask(__name__)

MAX_SEQUENCE_LENGTH = 427

def load_model_artifacts(model_path, tokenizer_path, label_map_path):
    model = load_model(model_path)
    with open(tokenizer_path, 'rb') as f:
        tokenizer = pickle.load(f)
    with open(label_map_path, 'rb') as f:
        label_map = pickle.load(f)
    reverse_label_map = {v: k for k, v in label_map.items()}
    return model, tokenizer, reverse_label_map

# Load Hindi model artifacts
hindi_model, hindi_tokenizer, hindi_reverse_label_map = load_model_artifacts(
    'lstm_genre_model_hindi.h5',
    'tokenizer_hindi.pkl',
    'label_map_hindi.pkl'
)

# Load English model artifacts
english_model, english_tokenizer, english_reverse_label_map = load_model_artifacts(
    'lstm_genre_modelbest.h5',
    'tokenizerbest.pkl',
    'label_mapbest.pkl'
)

def predict_genre(lyrics, model, tokenizer, reverse_label_map):
    lyrics = lyrics.lower()
    sequence = tokenizer.texts_to_sequences([lyrics])
    padded_seq = pad_sequences(sequence, maxlen=MAX_SEQUENCE_LENGTH)
    pred = model.predict(padded_seq)[0]
    top_index = np.argmax(pred)
    genre = reverse_label_map.get(top_index, "Unknown")
    return genre

@app.route('/', methods=['GET', 'POST'])
def index():
    if request.method == 'POST':
        language = request.form.get('language')
        lyrics = request.form.get('lyrics', '').strip()
        if not lyrics:
            result = "Please enter some lyrics."
        else:
            if language == 'english':
```

```
            result = predict_genre(lyrics, english_model, english_tokenizer,
english_reverse_label_map)
        elif language == 'hindi':
            result = predict_genre(lyrics, hindi_model, hindi_tokenizer,
hindi_reverse_label_map)
        else:
            result = "Invalid language selection."
        if request.headers.get('X-Requested-With') == 'XMLHttpRequest':
            return jsonify({'result': result})
    return render_template('index.html')

if __name__ == '__main__':
    app.run(debug=True)
```

Web Interface (Frontend)
templates/index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Music Genre Classification</title>
    <!-- Bootstrap CSS for a modern look -->
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
    <!-- Google Fonts: Merriweather for headlines, Roboto for body -->
    <link
href="https://fonts.googleapis.com/css2?family=Merriweather:wght@700&family=Roboto:w
ght@400;700&display=swap" rel="stylesheet">
    <style>
        body {
            background: linear-gradient(135deg, #f8f9fa, #e9ecef);
            font-family: 'Roboto', sans-serif;
            color: #333;
        }
        .hero {
            background-image: url('https://images.unsplash.com/photo-1511671782779-
c97d3d27a1d4?ixlib=rb-1.2.1&auto=format&fit=crop&w=1950&q=80');
            background-size: cover;
            background-position: center;
            padding: 100px 0;
            color: #fff;
            text-align: center;
            box-shadow: inset 0 0 10px rgba(0,0,0,0.5);
```

```
   transition: background 0.5s ease-in-out;
}
.hero:hover {
   filter: brightness(90%);
}
.hero h1 {
   font-family: 'Merriweather', serif;
   font-size: 4rem;
   font-weight: bold;
   text-shadow: 2px 2px 5px #000;
}
.hero p.lead {
   font-size: 1.5rem;
   font-weight: bold;
   text-shadow: 1px 1px 3px #000;
}
.card {
   margin-bottom: 30px;
   border: none;
   border-radius: 15px;
   box-shadow: 0 8px 16px rgba(0,0,0,0.15);
   transition: transform 0.3s ease;
}
.card:hover {
   transform: translateY(-5px);
}
.card-header {
   font-family: 'Merriweather', serif;
   font-size: 1.8rem;
   font-weight: bold;
   background-color: transparent;
   border-bottom: none;
   padding: 1rem 1.5rem;
}
.card-body label {
   font-weight: bold;
}
.btn-primary, .btn-success, .btn-secondary {
   font-weight: bold;
   letter-spacing: 0.5px;
}
.spinner-border {
   width: 3rem;
   height: 3rem;
}
footer {
```

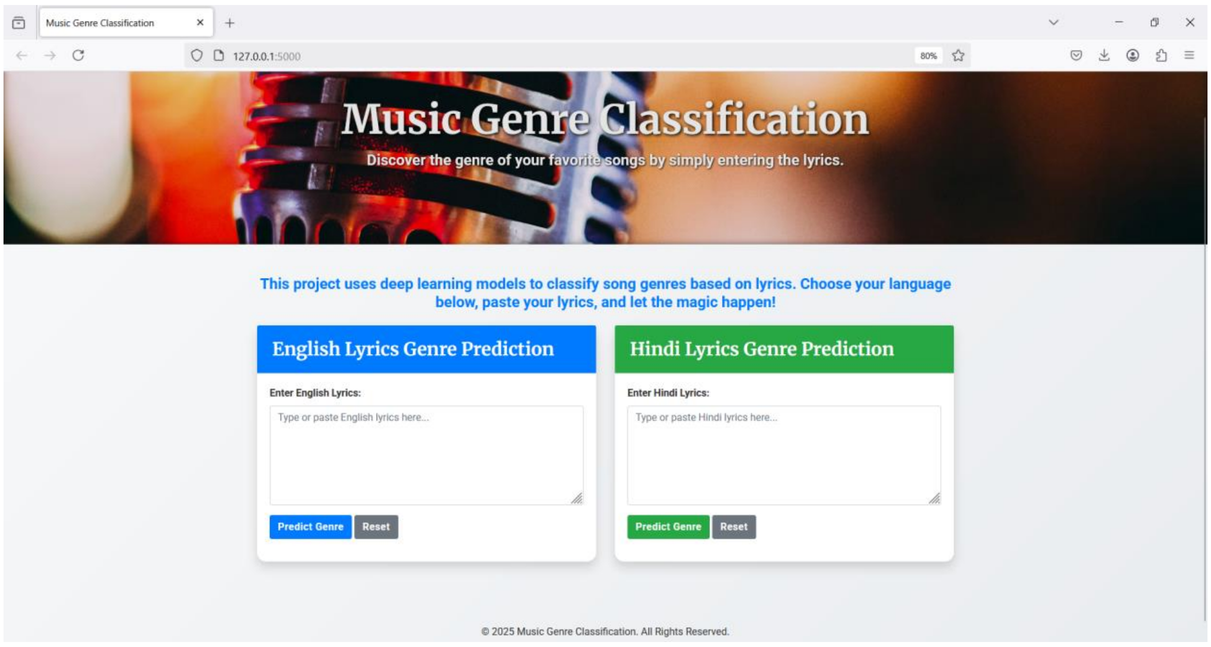
```
            margin-top: 50px;
            padding: 20px;
            text-align: center;
            background-color: transparent;
            color: #333;
        }
        /* Styling for predicted genre text */
        .predicted-genre {
            text-transform: capitalize;
            font-size: 1.8rem;
            font-weight: bold;
        }
        .predicted-genre.sad { color: #007bff; }
        .predicted-genre.happy { color: #28a745; }
        .predicted-genre.metal { color: #dc3545; }
        .predicted-genre.jazz { color: #6f42c1; }
    </style>
</head>
<body>
    <div class="hero">
        <div class="container">
            <h1 class="display-4">Music Genre Classification</h1>
            <p class="lead">Discover the genre of your favorite songs by simply entering the
lyrics.</p>
        </div>
    </div>

    <div class="container mt-5">
        <div class="mb-4">
            <p class="h4 text-center" style="font-weight: bold; color: #007bff;">
                This project uses deep learning models to classify song genres based on lyrics.
                Choose your language below, paste your lyrics, and let the magic happen!
            </p>
        </div>

        <div class="row">
            <!-- English Prediction Card -->
            <div class="col-md-6">
                <div class="card shadow">
                    <div class="card-header bg-primary text-white">
                        English Lyrics Genre Prediction
                    </div>
                    <div class="card-body">
                        <form id="english_form">
                            <input type="hidden" name="language" value="english">
                            <div class="form-group">
                                <label for="english_lyrics">Enter English Lyrics:</label>
                                <textarea class="form-control" name="lyrics" id="english_lyrics"
rows="6" placeholder="Type or paste English lyrics here..."></textarea>
                            </div>
                            <button type="submit" class="btn btn-primary">Predict Genre</button>
                            <button type="reset" id="english_reset" class="btn btn-
secondary">Reset</button>
                        </form>
                        <div id="english_result" class="mt-3"></div>
                    </div>
                </div>
            </div>
            <!-- Hindi Prediction Card -->
            <div class="col-md-6">
                <div class="card shadow">
                    <div class="card-header bg-success text-white">
                        Hindi Lyrics Genre Prediction
                    </div>
                    <div class="card-body">
                        <form id="hindi_form">
                            <input type="hidden" name="language" value="hindi">
                            <div class="form-group">
                                <label for="hindi_lyrics">Enter Hindi Lyrics:</label>
                                <textarea class="form-control" name="lyrics" id="hindi_lyrics"
rows="6" placeholder="Type or paste Hindi lyrics here..."></textarea>
                            </div>
                            <button type="submit" class="btn btn-success">Predict Genre</button>
                            <button type="reset" id="hindi_reset" class="btn btn-
secondary">Reset</button>
                        </form>
                        <div id="hindi_result" class="mt-3"></div>
                    </div>
                </div>
            </div>
        </div>

        <footer>
            <p>&copy; 2025 Music Genre Classification. All Rights Reserved.</p>
        </footer>
    </div>

    <!-- jQuery and Bootstrap JS -->
    <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@4.5.2/dist/js/bootstrap.bundle.min.js"></script>
    <script>
```

```
$(document).ready(function(){
    function getGenreClass(genre) {
        switch(genre.toLowerCase()) {
            case 'sad': return 'sad';
            case 'happy': return 'happy';
            case 'metal': return 'metal';
            case 'jazz': return 'jazz';
            default: return '';
        }
    }
    $('#english_form').submit(function(e){
        e.preventDefault();
        $('#english_result').html('<div class="text-center"><div class="spinner-border text-
primary" role="status"><span class="sr-only">Loading...</span></div></div>');
        $.ajax({
            type: "POST",
            url: '/',
            data: $(this).serialize(),
            dataType: "json",
            success: function(response){
                var genre = response.result;
                var genreClass = getGenreClass(genre);
                $('#english_result').html('<h4>Predicted Genre:</h4><p class="lead predicted-
genre ' + genreClass + '">' + genre + '</p>');
            },
            error: function(){
                $('#english_result').html('<p class="text-danger">Error processing your
request.</p>');
            }
        });
    });
    $('#english_reset').click(function(){
        $('#english_result').html('');
    });
    $('#hindi_form').submit(function(e){
        e.preventDefault();
        $('#hindi_result').html('<div class="text-center"><div class="spinner-border text-
success" role="status"><span class="sr-only">Loading...</span></div></div>');
        $.ajax({
            type: "POST",
            url: '/',
            data: $(this).serialize(),
            dataType: "json",
            success: function(response){
                var genre = response.result;
                var genreClass = getGenreClass(genre);
```

```
                    $('#hindi_result').html('<h4>Predicted Genre:</h4><p class="lead predicted-
genre ' + genreClass + '">' + genre + '</p>');
                },
                error: function(){
                    $('#hindi_result').html('<p class="text-danger">Error processing your
request.</p>');
                }
            });
        });
        $('#hindi_reset').click(function(){
            $('#hindi_result').html('');
        });
    });
    </script>
</body>
</html>
```

Chapter 7

User Interface



Music Genre Classification

Discover the genre of your favorite songs by simply entering the lyrics.

This project uses deep learning models to classify song genres based on lyrics. Choose your language below, paste your lyrics, and let the magic happen!

English Lyrics Genre Prediction

Enter English Lyrics:

Type or paste English lyrics here...

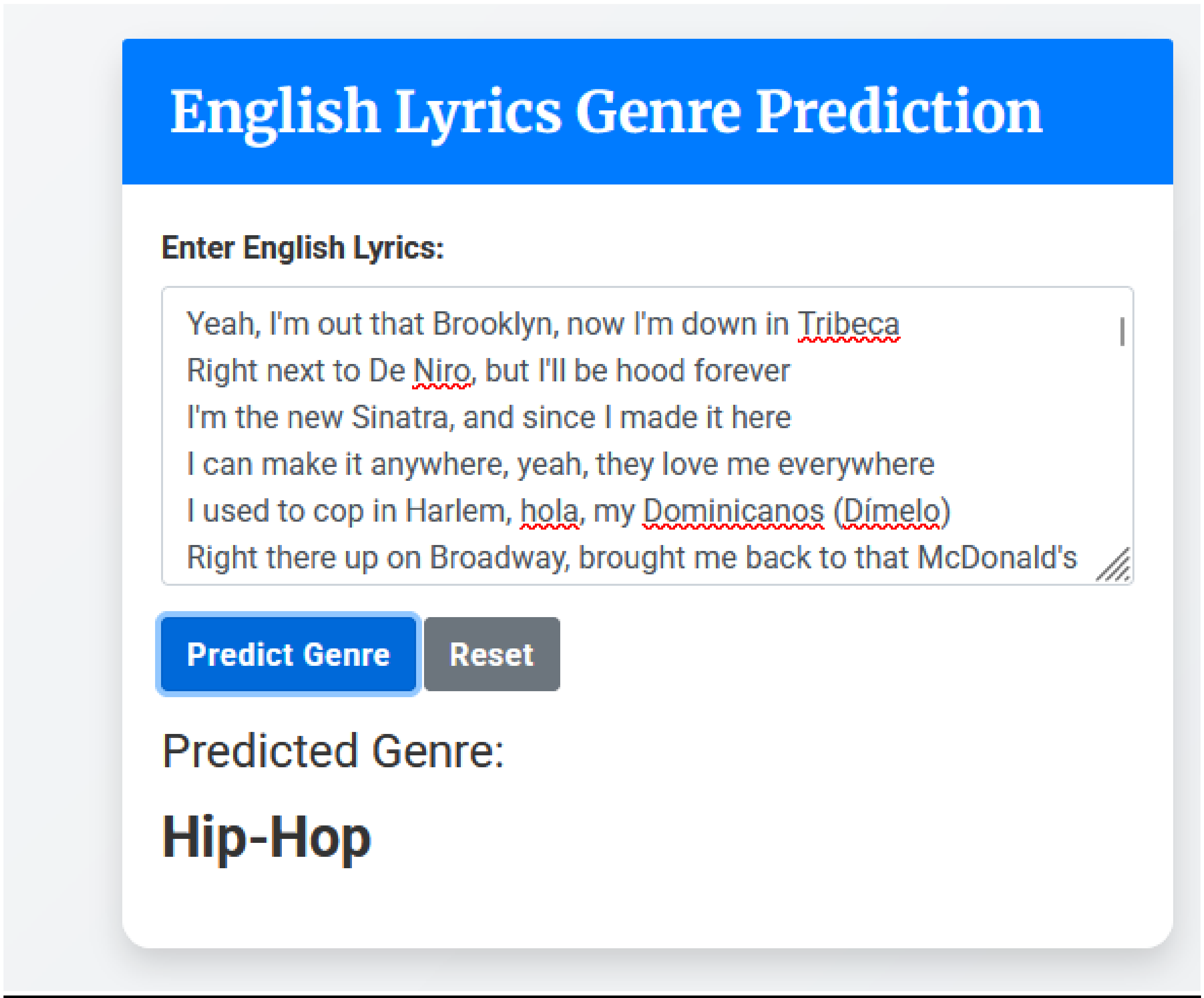
Hindi Lyrics Genre Prediction

Enter Hindi Lyrics:

Type or paste Hindi lyrics here...

© 2025 Music Genre Classification. All Rights Reserved.

We will provide hip hop song lyrics to check if it predicts the genre correctly



English Lyrics Genre Prediction

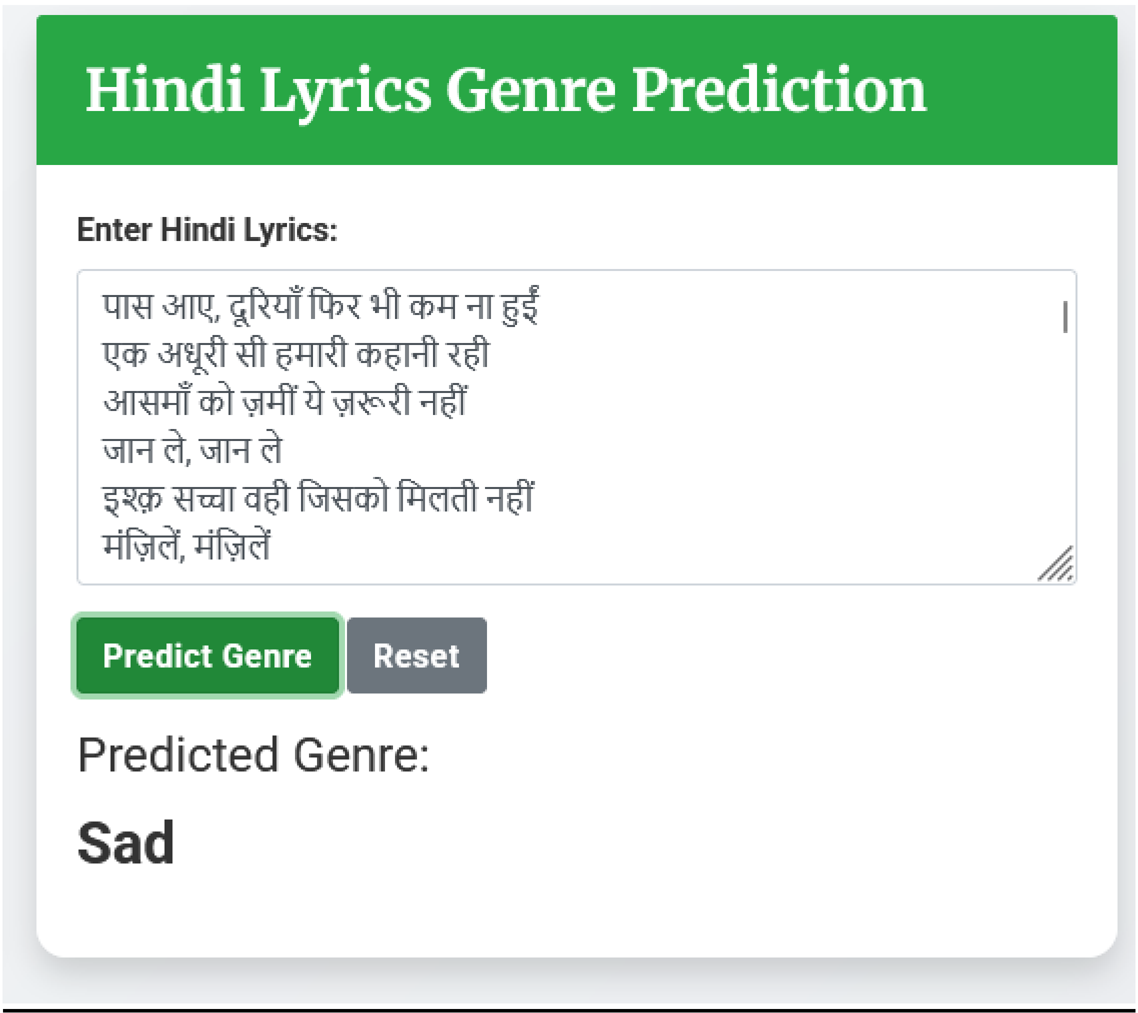
Enter English Lyrics:

Yeah, I'm out that Brooklyn, now I'm down in Tribeca
Right next to De Niro, but I'll be hood forever
I'm the new Sinatra, and since I made it here
I can make it anywhere, yeah, they love me everywhere
I used to cop in Harlem, hola, my Dominicanos (Dímelo)
Right there up on Broadway, brought me back to that McDonald's

Predicted Genre:

Hip-Hop

We will provide Sad Hindi song lyrics to check if it predicts the genre correctly



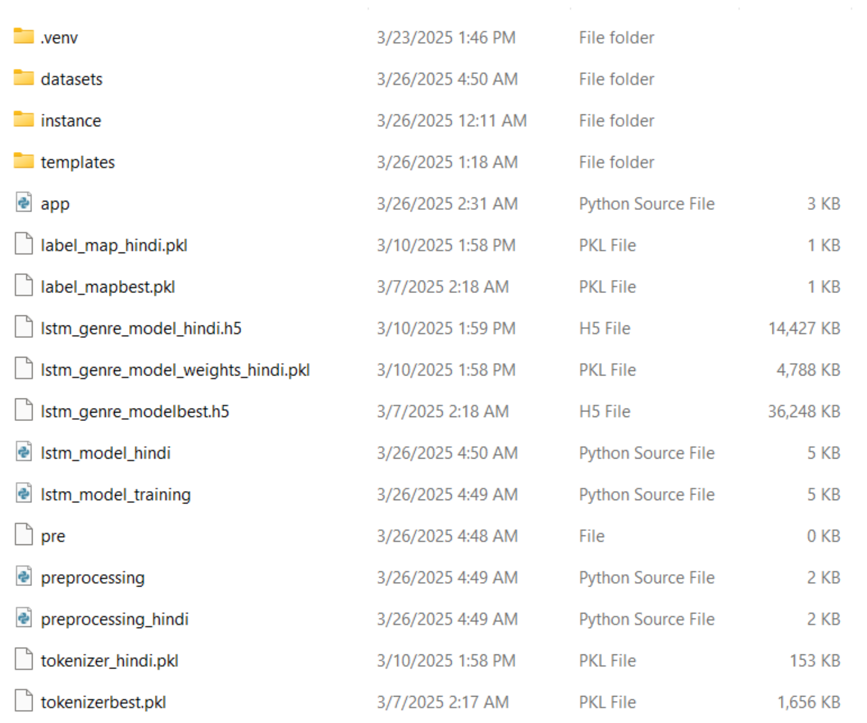
Hindi Lyrics Genre Prediction

Enter Hindi Lyrics:

पास आए, दूरियाँ फिर भी कम ना हुईं
एक अधूरी सी हमारी कहानी रही
आसमाँ को ज़मीं ये ज़रूरी नहीं
जान ले, जान ले
इश्क़ सच्चा वही जिसको मिलती नहीं
मंज़िलें, मंज़िलें

Predicted Genre:

Sad



.venv	3/23/2025 1:46 PM	File folder	
datasets	3/26/2025 4:50 AM	File folder	
instance	3/26/2025 12:11 AM	File folder	
templates	3/26/2025 1:18 AM	File folder	
app	3/26/2025 2:31 AM	Python Source File	3 KB
label_map_hindi.pkl	3/10/2025 1:58 PM	PKL File	1 KB
label_mapbest.pkl	3/7/2025 2:18 AM	PKL File	1 KB
lstm_genre_model_hindi.h5	3/10/2025 1:59 PM	H5 File	14,427 KB
lstm_genre_model_weights_hindi.pkl	3/10/2025 1:58 PM	PKL File	4,788 KB
lstm_genre_modelbest.h5	3/7/2025 2:18 AM	H5 File	36,248 KB
lstm_model_hindi	3/26/2025 4:50 AM	Python Source File	5 KB
lstm_model_training	3/26/2025 4:49 AM	Python Source File	5 KB
pre	3/26/2025 4:48 AM	File	0 KB
preprocessing	3/26/2025 4:49 AM	Python Source File	2 KB
preprocessing_hindi	3/26/2025 4:49 AM	Python Source File	2 KB
tokenizer_hindi.pkl	3/10/2025 1:58 PM	PKL File	153 KB
tokenizerbest.pkl	3/7/2025 2:17 AM	PKL File	1,656 KB

Chapter 8

Conclusion

This project has successfully demonstrated the feasibility and effectiveness of using deep learning techniques for music genre classification based solely on song lyrics. Our approach tackled the inherent challenges of processing textual data from two different languages—English and Hindi—by developing separate, language-specific preprocessing pipelines and dedicated LSTM models.

For the English module, we collected and merged multiple publicly available datasets to create a comprehensive corpus of song lyrics. The preprocessing pipeline involved cleaning the text by converting it to lowercase, removing punctuation and extraneous characters, and eliminating stopwords using NLTK. The cleaned data was then tokenized and padded to a fixed length, allowing us to effectively train a bidirectional LSTM model using pre-trained GloVe 6B 100d embeddings. This setup enabled the model to capture semantic nuances and contextual information from the lyrics, resulting in robust genre predictions.

In contrast, the Hindi module presented unique challenges due to the limited availability of dedicated datasets and resources. To overcome this, we manually created our own dataset by extracting Hindi song lyrics, and we compiled a comprehensive stopwords file from available online sources. The preprocessing of Hindi lyrics was tailored to handle language-specific punctuation and stopwords. We then trained a separate bidirectional LSTM model using pre-trained Hindi embeddings from the **cc.hi.300.vec** file, which provides 300-dimensional word vectors. Despite the constraints, the Hindi model achieved promising results in classifying songs into genres such as "happy" and "sad."

A critical component of the project was the integration of both models into a unified, user-friendly web interface built with Flask, Bootstrap, and jQuery. The interface features separate sections for English and Hindi genre predictions and uses AJAX to submit user input asynchronously. This design choice ensures that users receive real-time feedback without the need for page reloads, significantly enhancing the overall user experience. The interface also employs responsive design techniques, ensuring compatibility across different devices and screen sizes.

Throughout the project, careful attention was given to model evaluation and performance optimization. Techniques such as early stopping and learning rate reduction were employed during training to prevent overfitting and ensure that the models generalize well on unseen data. The system was rigorously tested to confirm that it meets the required accuracy and responsiveness standards, making it a viable tool for practical applications.

In summary, this project not only provides an effective solution for automatic music genre classification from lyrics but also highlights the potential of deep learning in handling multilingual text data. The methodology and system design presented here lay a solid foundation for further enhancements, such as integrating additional languages, incorporating more advanced neural network architectures, and extending the system to include personalized music recommendations. The successful implementation of this project demonstrates its relevance and applicability in real-world digital music platforms, offering both academic and commercial value in the evolving landscape of music analytics and recommendation systems.