Predicting Hit Songs using an LSTM Network

Project Overview

A music streaming company wants to predict whether a newly released song will become a hit. Early prediction helps the marketing team decide which songs to promote heavily.

This notebook builds an LSTM-based prediction model to forecast if a song will reach the **Top 50** in the charts within the next month based on its initial streaming data. We will use the Spotify Charts dataset from Kaggle or, if unavailable, a synthetically generated dataset that mimics streaming patterns.

1. Imports

First, we import all the necessary libraries. We need `pandas` and `numpy` for data manipulation, `tensorflow` and `keras` for building the LSTM model, `scikit-learn` for data preprocessing and evaluation, and `plotly` for creating interactive visualizations.

In [1]:
```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import confusion_matrix
import plotly.graph_objects as go
import os
import zipfile
import requests
from io import BytesIO
import sys
```

```
2025-08-12 11:10:22.298683: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may se
e slightly different numerical results due to floating-point round-off errors from different computation or
ders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2025-08-12 11:10:22.384305: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:467] Unable to regist
er cuFFT factory: Attempting to register factory for plugin cuFFT when one has already been registered
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
E0000 00:00:1754977222.422966    6044 cuda_dnn.cc:8579] Unable to register cuDNN factory: Attempting to reg
ister factory for plugin cuDNN when one has already been registered
E0000 00:00:1754977222.433108    6044 cuda_blas.cc:1407] Unable to register cuBLAS factory: Attempting to r
egister factory for plugin cuBLAS when one has already been registered
W0000 00:00:1754977222.498038    6044 computation_placer.cc:177] computation placer already registered. Ple
ase check linkage and avoid linking the same target more than once.
W0000 00:00:1754977222.498059    6044 computation_placer.cc:177] computation placer already registered. Ple
ase check linkage and avoid linking the same target more than once.
W0000 00:00:1754977222.498060    6044 computation_placer.cc:177] computation placer already registered. Ple
ase check linkage and avoid linking the same target more than once.
W0000 00:00:1754977222.498061    6044 computation_placer.cc:177] computation placer already registered. Ple
ase check linkage and avoid linking the same target more than once.
2025-08-12 11:10:22.506056: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is
optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 AVX_VNNI FMA, in other operations, rebuild TensorFlow with the a
ppropriate compiler flags.
```

2. Configuration

We define some global parameters for our model and data processing. This makes it easy to tweak the model's behavior later.

- `LOOK_BACK`: The number of past days of streaming data the model will use to make a prediction (e.g., 14 days).
- `PREDICT_HORIZON`: The number of future days the model will look into to determine if a song became a hit (e.g., 7 days).
- `EPOCHS`: The number of times the model will iterate over the entire training dataset.
- `BATCH_SIZE`: The number of samples processed before the model is updated.

In [2]:
```python
# --- Configuration ---
LOOK_BACK = 14
PREDICT_HORIZON = 7
EPOCHS = 10
BATCH_SIZE = 32
MODEL_PATH = 'hit_song_predictor.h5'
DATA_FILE_PATH = 'charts.csv'
```

3. Data Loading and Preprocessing

These helper functions handle downloading the data from Kaggle (if needed) and loading it into a pandas DataFrame. If the primary dataset (`charts.csv`) is not found, a synthetic dataset is created to allow the notebook to run. The `load_data` function also performs initial preprocessing, like converting the 'date' column and handling missing 'streams' values.

In [3]:
```python
def load_data(file_path):
    """Loads and preprocesses the dataset."""
    try:
        data = pd.read_csv(file_path, on_bad_lines='skip')
        # Basic preprocessing
        data['date'] = pd.to_datetime(data['date'])
        if 'streams' not in data.columns:
            print("'streams' column not found. Simulating stream count from 'rank'.")
            data['streams'] = 201 - data['rank'] # Invert rank to simulate stream count
        data['streams'].fillna(0, inplace=True)
        return data
    except FileNotFoundError:
        print(f"Error: Dataset file not found at {file_path}.")
        return None

# --- Data Loading and Preparation ---
if not os.path.exists(DATA_FILE_PATH):
    print(f"Warning: '{DATA_FILE_PATH}' not found. Creating a synthetic dataset for demonstration.")
    dates = pd.to_datetime(pd.date_range(start='2023-01-01', periods=100))
    songs = []
    for i in range(1, 41):
        if i % 2 == 0: # Hit song
            streams = np.linspace(10000, 100000, 50) + np.random.randint(-5000, 5000, 50)
            streams = np.append(streams, np.linspace(100000, 500000, 50) + np.random.randint(-10000, 10000
            ranks = 201 - (streams / streams.max() * 200)
```

```
        else: # Non-hit song
            streams = np.linspace(5000, 20000, 100) + np.random.randint(-2000, 2000, 100)
            ranks = 201 - (streams / streams.max() * 100)
        for j in range(100):
            songs.append({'date': dates[j], 'rank': int(ranks[j]), 'title': f'Song {i}', 'artist': f'Artis
    pd.DataFrame(songs).to_csv(DATA_FILE_PATH, index=False)
    print("Synthetic dataset created.")

df = load_data(DATA_FILE_PATH)
if df is not None:
    print("Data loaded successfully.")
    display(df.head())
```

```
Warning: 'charts.csv' not found. Creating a synthetic dataset for demonstration.
Synthetic dataset created.
Data loaded successfully.
```

```
/tmp/ipykernel_6044/1244481109.py:10: FutureWarning: A value is trying to be set on a copy of a DataFrame o
r Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object
on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=T
rue)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  data['streams'].fillna(0, inplace=True)
```

	date	rank	title	artist	streams
0	2023-01-01	178	Song 1	Artist 1	4801
1	2023-01-02	174	Song 1	Artist 1	5819
2	2023-01-03	185	Song 1	Artist 1	3377
3	2023-01-04	175	Song 1	Artist 1	5567
4	2023-01-05	172	Song 1	Artist 1	6131

4. Creating Sequences for LSTM

LSTMs require input data to be in the form of sequences. This function transforms our time-series data (daily streams for each song) into input-output pairs.

- **Input (X)**: A sequence of stream counts from the past `LOOK_BACK` days.
- **Output (y)**: A binary label (1 for 'Hit', 0 for 'Not a Hit'). A song is labeled a 'Hit' if its rank is 50 or less at any point within the future `PREDICT_HORIZON` days.

In [4]:
```
def create_sequences(data, look_back, predict_horizon):
    """Creates sequences for the LSTM model."""
    X, y, song_titles = [], [], []
    unique_songs = data['title'].unique()

    print(f"Creating sequences with look_back={look_back} and predict_horizon={predict_horizon}...")
    for song in unique_songs:
        song_df = data[data['title'] == song].sort_values('date')
        streams = song_df['streams'].values

        if len(streams) >= look_back + predict_horizon:
            for i in range(len(streams) - look_back - predict_horizon + 1):
                X.append(streams[i:(i + look_back)])
                future_ranks = data[
                    (data['title'] == song) &
                    (data['date'] > song_df['date'].iloc[i + look_back - 1]) &
                    (data['date'] <= song_df['date'].iloc[i + look_back - 1] + pd.Timedelta(days=predict_h
                ]['rank']

                if not future_ranks.empty and (future_ranks <= 50).any():
                    y.append(1)
                else:
                    y.append(0)
                song_titles.append(song)

    print(f"Created {len(X)} sequences.")
    return np.array(X), np.array(y), song_titles

X, y, _ = create_sequences(df, LOOK_BACK, PREDICT_HORIZON)

# Scale and reshape data for the model
scaler = MinMaxScaler(feature_range=(0, 1))
X_scaled = scaler.fit_transform(X)
```

```
X_scaled = X_scaled.reshape((X_scaled.shape[0], X_scaled.shape[1], 1))

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42, stratify=y
print(f"Training data shape: {X_train.shape}")
print(f"Testing data shape: {X_test.shape}")
```

```
Creating sequences with look_back=14 and predict_horizon=7...
Created 3200 sequences.
Training data shape: (2560, 14, 1)
Testing data shape: (640, 14, 1)
```

5. Building the LSTM Model

Here we define the architecture of our neural network. It consists of:

1. An **LSTM layer** with 64 units to process the input sequences.
2. A **Dropout layer** to prevent overfitting.
3. A second **LSTM layer** with 32 units.
4. Another **Dropout layer**.
5. A **Dense (fully connected) layer** with 16 units and a ReLU activation function.
6. The **output layer**, which is a single Dense neuron with a **sigmoid activation function**. This outputs a probability between 0 and 1, which is perfect for binary classification.

In [5]:
```
def build_model(input_shape):
    """Builds and compiles the LSTM model."""
    print("Building LSTM model...")
    model = Sequential([
        LSTM(64, input_shape=input_shape, return_sequences=True),
        Dropout(0.2),
        LSTM(32),
        Dropout(0.2),
        Dense(16, activation='relu'),
        Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    model.summary()
    return model
```

6. Model Training

We now train the model on our prepared dataset. If a pre-trained model file (`hit_song_predictor.h5`) exists, we'll skip this step to save time. Otherwise, the model is trained for the specified number of `EPOCHS` .

In [6]:
```python
if not os.path.exists(MODEL_PATH):
    print("\n--- Training Model ---")
    model = build_model((X_train.shape[1], 1))
    history = model.fit(X_train, y_train, epochs=EPOCHS, batch_size=BATCH_SIZE, validation_data=(X_test, y_
    model.save(MODEL_PATH)
    print(f"Model trained and saved to {MODEL_PATH}")
else:
    print(f"\nFound existing model at {MODEL_PATH}. Loading it.")
    model = tf.keras.models.load_model(MODEL_PATH)
    history = None # No history if we load the model
```

```
--- Training Model ---
Building LSTM model...
```

```
I0000 00:00:1754977226.614631    6044 gpu_device.cc:2019] Created device /job:localhost/replica:0/task:0/de
vice:GPU:0 with 2958 MB memory:  -> device: 0, name: NVIDIA GeForce RTX 4050 Laptop GPU, pci bus id: 0000:0
1:00.0, compute capability: 8.9
/home/abhijit-42/miniconda3/envs/tf-env/lib/python3.11/site-packages/keras/src/layers/rnn/rnn.py:199: UserW
arning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer
using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 14, 64)	16,896
dropout (Dropout)	(None, 14, 64)	0
lstm_1 (LSTM)	(None, 32)	12,416
dropout_1 (Dropout)	(None, 32)	0
dense (Dense)	(None, 16)	528
dense_1 (Dense)	(None, 1)	17

Total params: 29,857 (116.63 KB)

Trainable params: 29,857 (116.63 KB)

Non-trainable params: 0 (0.00 B)

```
Epoch 1/10
I0000 00:00:1754977228.488042    6134 cuda_dnn.cc:529] Loaded cuDNN version 90300
```

```
80/80 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.9391 - loss: 0.2577 - val_accuracy: 0.9812 - val_loss:
0.0453
Epoch 2/10
80/80 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9871 - loss: 0.0387 - val_accuracy: 0.9812 - val_loss:
0.0362
Epoch 3/10
80/80 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9887 - loss: 0.0258 - val_accuracy: 0.9844 - val_loss:
0.0404
Epoch 4/10
80/80 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9922 - loss: 0.0222 - val_accuracy: 0.9891 - val_loss:
0.0277
Epoch 5/10
80/80 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9941 - loss: 0.0154 - val_accuracy: 0.9937 - val_loss:
0.0154
Epoch 6/10
80/80 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9941 - loss: 0.0147 - val_accuracy: 0.9953 - val_loss:
0.0165
Epoch 7/10
80/80 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9930 - loss: 0.0187 - val_accuracy: 0.9906 - val_loss:
0.0213
Epoch 8/10
80/80 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9914 - loss: 0.0179 - val_accuracy: 0.9891 - val_loss:
0.0256
Epoch 9/10
80/80 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9906 - loss: 0.0179 - val_accuracy: 0.9922 - val_loss:
0.0144
Epoch 10/10
80/80 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9941 - loss: 0.0156 - val_accuracy: 0.9937 - val_loss:
0.0157
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(mode
l)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `mode
l.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
Model trained and saved to hit_song_predictor.h5
```

7. Visualizing Model Performance

These functions help us visualize the model's performance.

- `plot_history`: Shows the training and validation accuracy/loss over epochs. This helps diagnose overfitting or underfitting.

- `plot_confusion_matrix` : Shows how many predictions were correct (True Positives, True Negatives) and incorrect (False Positives, False Negatives) on the test data.

In [7]:
```
def plot_history(history):
    """Plots training & validation accuracy and loss."""
    if history is None:
        print("No training history to plot as the model was pre-loaded.")
        return

    print("\nGenerating training history plot...")
    fig = go.Figure()
    fig.add_trace(go.Scatter(y=history.history['accuracy'], name='Train Accuracy', line=dict(color='blue')
    fig.add_trace(go.Scatter(y=history.history['val_accuracy'], name='Validation Accuracy', line=dict(colo
    fig.add_trace(go.Scatter(y=history.history['loss'], name='Train Loss', yaxis='y2', line=dict(color='re
    fig.add_trace(go.Scatter(y=history.history['val_loss'], name='Validation Loss', yaxis='y2', line=dict(

    fig.update_layout(
        title_text="<b>Model Training History</b>",
        xaxis_title="Epoch",
        yaxis=dict(title="Accuracy"),
        yaxis2=dict(title="Loss", overlaying='y', side='right'),
        legend_title="Metrics"
    )
    fig.show()

def plot_confusion_matrix(y_true, y_pred_classes, labels):
    """Plots the confusion matrix."""
    print("\nGenerating confusion matrix plot...")
    cm = confusion_matrix(y_true, y_pred_classes)
    fig = go.Figure(data=go.Heatmap(
        z=cm,
        x=labels,
        y=labels,
        hoverongaps=False,
        colorscale='Blues',
        text=[[str(y) for y in x] for x in cm],
        texttemplate="%{text}"
    ))
    fig.update_layout(
        title='<b>Confusion Matrix on Test Data</b>',
        xaxis_title='Predicted Label',
```

```python
        yaxis_title='True Label'
    )
    fig.show()

# Plot training history if the model was just trained
plot_history(history)

# Evaluate and plot confusion matrix on the test set
y_pred_proba = model.predict(X_test)
y_pred_classes = (y_pred_proba > 0.5).astype("int32")
plot_confusion_matrix(y_test, y_pred_classes, labels=['Not a Hit', 'Hit'])
```

```
Generating training history plot...
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step

Generating confusion matrix plot...
```