
Project Report: Lyrics Sentiment Analysis

Date: April 4, 2025

Team Members:

- DEEPAK SHITOLE (642303019)
- SOMESHWAR GIRAM (642303009)
- OM BHUTKAR (642303005)

1. Executive Summary

This report details the "Lyrics Sentiment Analysis" project, designed to analyze the emotional tone (Positive, Negative, Neutral) within song lyrics using Natural Language Processing (NLP) techniques. The project successfully implements a system that processes a dataset of song lyrics, performs sentiment analysis using TextBlob, and presents the findings through an interactive web application built with Streamlit. Key deliverables include visualizations of sentiment trends, distributions, top songs/artists, word clouds, and the ability for users to analyze custom text snippets in real-time. The primary goal is to provide insights into the emotional landscape of music based on lyrical content.

2. Introduction

Music lyrics are a rich source of emotional expression. Understanding the sentiment conveyed in songs can offer valuable insights into cultural trends, artistic expression, and the potential relationship between lyrical emotion and musical characteristics like genre or artist popularity. This project aims to automate the process of sentiment analysis for large datasets of song lyrics. By applying NLP techniques, specifically using the TextBlob library, the project calculates sentiment polarity and subjectivity scores. An interactive Streamlit dashboard serves as the user interface, facilitating exploration and analysis of the results.

3. Project Features and Functionality

The project offers a comprehensive set of features for analyzing song lyrics sentiment:

- **Data Handling:** Loads lyrics data from a CSV file (data/song_lyrics.csv).

- **Preprocessing:** Includes language filtering (defaulting to English) and standard text cleaning necessary for NLP tasks (handled within src/preprocessing.py).
- **Sentiment Analysis:** Utilizes TextBlob (src/sentiment_analyzer.py) to compute:
    - **Polarity:** A score indicating the positivity or negativity (-1 to +1).
    - **Subjectivity:** A score indicating how objective or subjective the text is (0 to 1).
- **Data Visualization (src/visualization.py, outputs in plots/):**
    - Overall sentiment distribution (Positive/Negative/Neutral).
    - Average sentiment polarity for top N artists and genres/tags.
    - Sentiment polarity trends over the years available in the dataset.
    - Word cloud visualizing the most frequent terms in the lyrics.
    - Distribution of subjectivity scores via histogram.
- **Insights:** Lists the top 10 most positive and negative songs based on their polarity scores.
- **Interactive Dashboard (app.py):** A user-friendly web interface built with Streamlit.
- **Custom Analysis:** Allows users to input their own text (lyrics or other) for on-the-fly sentiment analysis.
- **Data Export:** Option to download the processed data, including lyrics and calculated sentiment scores, as a CSV file (data/processed_lyrics.csv).

4. Methodology and Implementation

The project follows a standard data science pipeline:

1. **Data Loading (src/data_loader.py):** Reads the raw song lyrics dataset (CSV format) into a Pandas DataFrame. Configuration for filenames and columns is managed (likely in main.py or config.py).
2. **Preprocessing (src/preprocessing.py):** Cleans the text data. This typically involves steps like lowercasing, removing punctuation, tokenization (splitting into words), removing stopwords (common words like 'the', 'is'), and potentially lemmatization (reducing words to their base form). NLTK library is used for these tasks. Language filtering is also applied here.

3. **Sentiment Analysis (src/sentiment_analyzer.py):** Applies the TextBlob library to the processed lyrics column. For each song, it calculates the polarity and subjectivity scores. These scores are added as new columns to the DataFrame.
4. **Analysis and Visualization (src/visualization.py, main.py):** Aggregates data to generate insights. Calculates overall distributions, groups data by artist, genre, and year to find trends and averages. Uses Matplotlib, Seaborn, Plotly, and WordCloud to create plots saved in the plots/ directory (e.g., sentiment_by_artist_top_10.png, sentiment_trends_over_time.png). Identifies top positive/negative songs.
5. **Web Application (app.py):** Uses Streamlit to build the interactive dashboard. It provides controls (like the "Run Full Analysis Pipeline" button) and displays the results (data tables, plots). It includes a text input area for custom analysis.
6. **Backend Logic (main.py):** Contains the core function (run_full_analysis) that orchestrates the loading, preprocessing, analysis, and visualization steps. Also likely holds configuration constants.

5. Technical Stack

- **Programming Language:** Python 3.x
- **Data Manipulation:** Pandas
- **NLP & Sentiment Analysis:** NLTK, TextBlob
- **Web Application Framework:** Streamlit
- **Data Visualization:** Matplotlib, Seaborn, Plotly, WordCloud
- **Environment Management:** Virtual Environment (venv)

6. Project Structure

The project is organized into distinct modules for clarity and maintainability:

```
.
├── app.py              # Main Streamlit application script
├── config.py           # Configuration variables (optional, might be in main.py)
├── main.py             # Backend pipeline logic (run_full_analysis) & config
├── README.md           # Project documentation (this file)
├── requirements.txt    # Project dependencies
├── data/               # Data files directory
│   ├── song_lyrics.csv     # Example raw dataset
│   ├── processed_lyrics.csv # Output data with sentiment (after run)
```

```
│   └── ... (other data files like large versions)
├── notebooks/          # Jupyter notebooks for exploration/prototyping
│   └── preprocessing_exploration.ipynb # Example notebook
├── plots/              # Saved visualization outputs
│   ├── sentiment_by_artist_top_10.png
│   ├── sentiment_by_tag_top_10.png
│   ├── sentiment_tb_distribution.png
│   ├── sentiment_trends_over_time.png
│   └── ... (other generated plots)
├── src/                # Source code directory
│   ├── __init__.py
│   ├── data_loader.py    # Handles data loading
│   ├── preprocessing.py  # Text preprocessing functions
│   ├── sentiment_analyzer.py# Sentiment analysis functions
│   ├── utils.py          # Utility functions (e.g., saving data)
│   └── visualization.py  # Plotting functions
└── venv/               # Virtual environment directory (if created)
```

7. Data

The project utilizes a song lyrics dataset sourced from Kaggle: Genius Song Lyrics with Language Information. The primary input is expected in CSV format (e.g., song_lyrics.csv) located in the data/ directory. The analysis pipeline generates a processed CSV file (processed_lyrics.csv) containing the original data augmented with sentiment scores. The project structure indicates handling of potentially large datasets (e.g., processed_lyrics_large.csv).

8. Setup and Usage

(Prerequisites: Python 3.x, Git)

1. **Clone:** git clone https://github.com/Deepak22903/song_lyrics_sentiment_analysis.git
2. **Navigate:** cd song_lyrics_sentiment_analysis
3. **Create & Activate Virtual Environment:**
    - python -m venv venv
    - Windows: .\venv\Scripts\activate
    - macOS/Linux: source venv/bin/activate
4. **Install Dependencies:** pip install -r requirements.txt
5. **Download NLTK Data:** Run python and execute:

Python

```
import nltk
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('stopwords')
quit()
```

6. **Prepare Data:** Place the dataset CSV (e.g., song_lyrics.csv) in the data/ directory. Ensure the filename matches the configuration in main.py or config.py.
7. **Run Application:** streamlit run app.py
8. **Interact:** Access the application via the URL provided in the terminal. Use the sidebar options to run the analysis or analyze custom text.

9. Results and Visualizations

The application presents several key results:

- **Sentiment Distribution:** A bar chart or pie chart showing the proportion of Positive, Negative, and Neutral songs in the dataset.
- **Sentiment by Artist/Genre:** Bar charts displaying the average polarity score for the top N artists and genres, indicating which tend to have more positive or negative lyrics.
- **Sentiment Trends:** A line plot showing how the average sentiment polarity has changed over the years covered by the data.
- **Word Cloud:** A visual representation of the most frequent words in the lyrics corpus (after preprocessing).
- **Top Songs:** Lists of the songs with the highest (most positive) and lowest (most negative) polarity scores.
- **Subjectivity Distribution:** A histogram showing how subjective or objective the lyrics tend to be.
- **Custom Analysis Output:** Polarity and subjectivity scores for user-provided text.

10. Future Enhancements

Several potential improvements and extensions were identified:

- **Alternative Sentiment Models:** Integrate VADER for comparison.

- **Topic Modeling:** Use techniques like LDA to discover lyrical themes.
- **N-gram Analysis:** Identify common multi-word phrases.
- **Enhanced Filtering:** Allow users to filter data by artist, genre, or year *before* analysis.
- **Interactive Plots:** Increase the use of Plotly for more dynamic visualizations.
- **Performance Optimization:** Utilize Streamlit's caching features (@st.cache_data, @st.cache_resource).
- **Deployment:** Host the application online (e.g., Streamlit Community Cloud, Heroku).

11. Conclusion

The Lyrics Sentiment Analysis project successfully demonstrates the application of NLP techniques to extract emotional insights from song lyrics. The interactive Streamlit dashboard provides an accessible way to explore sentiment distributions, trends, and perform custom analyses. The modular code structure and clear documentation facilitate understanding and future development. The project serves as a valuable tool for anyone interested in the intersection of music, language, and emotion.
