Abstract

Analyzing the emotional tone of songs' texts can give insights into societal trends and this information can be useful especially for recommendation algorithms. This study focuses on developing four Machine Learning models to classify emotions conveyed in English song lyrics at the stanza level. The classification employs Plutchik's eight primary emotions, offering a nuanced understanding of emotional expression in lyrical content. The selected model architectures are:

- **Random Forest**
- **Support Vector Machine (SVM)**
- **One-Dimensional Convolutional Neural Network (1D-CNN)**
- **Recurrent Neural Network (RNN)**

These models were chosen for their proven effectiveness across various domains and their diverse approaches, providing a thorough investigation of different techniques for emotion classification in text.

DA FINIRE CON CONCLUSIONI

Introduction

Songs have the unique ability to engage people in ways that few other mediums can match. Lyrics serve as one of the main foundations of songs, playing a crucial role in expressing feelings in many different ways. The emotional tone of songs can serve various purposes, such as automatized playlist creation or songs' organization, offering an alternative to the more traditional genre-based classification.

The goal of this project is the development of 4 Machine Learning models that perform emotion detection on songs' stanzas. To obtain a deeper understanding of emotional fluctuations within the texts, the models assign emotion labels to individual stanzas instead of full songs. The emotion

labels correspond to Robert Plutchik's eight primary emotions (shown in figure 1), providing a comprehensive range for representing various emotional states.

Figure 1: Plutchik's eight primary emotions

This report aims to clearly cover and illustrate various aspects of the work. The *Methods* section provides a detailed explanation of the data and procedures used in the project, with a particular focus on the pipeline implemented to construct the models. Following this, the *Results* chapter presents an overview of the obtained outcomes, using plots and figures to highlight significant findings. These results are further explored in the final sections, *Discussion* and *Conclusions*, which interpret the general findings, recap the primary objectives of the work, and discuss the importance or potential applications of the results.

Methods

The dataset used in this project is a sampled subset of English-language songs derived from the *Genius Song Lyrics Dataset*[1]. The original dataset contained numerous attributes; the ones considered relevant for model training are:

- **title:** the song's title;
- **lemmatized_stanzas:** lyrics of the single stanza;
- **stanza_number:** identifies the position of the stanza in the song;
- **is_chorus:** boolean variable that attests whether the stanza is a chorus or not;
- **tag:** represents the genre of the song. For easier handling, this attribute of the original

dataset has been one-hot encoded into various boolean variables (is_country, is_pop, is_rap, is_rb, is_rock);

- **label:** represents the emotional classification of the stanza, assigned by Albert Base v2[2] model.

All of these attributes, except for `title`, were the result of the preprocessing phase, as described in section . Due to limited computational power, the labeling process was time-intensive, ultimately resulting in a limited dataset consisting of **(QUANTE? AGGIUNGEREI NUMERO STROFE)**.

Preprocessing

The initial preprocessing step involved sampling from the original dataset while maintaining the proportional distribution of genres. This approach ensured that the genre representation in the sampled subset accurately reflected that of the full dataset.
The preliminary text cleaning process focused on `lyrics`, which contained the complete lyrics of each song in string format. Initially, a regular expression (RegEx) was built to remove noise from the lyrics, specifically targeting words enclosed in square brackets that were irrelevant to the stanza splitting process. Many keywords marking different stanzas were written within square brackets, and removing non-keyword items inside brackets was crucial to avoid potential issues.
The next critical step was stanza splitting. After cleaning texts from noisy square-bracketed items, lyrics were split based on various keywords used to denote stanzas (such as ”chorus”, ”verse”, ”intro”, ”outro”, ”refrain”, ”hook”, etc.). The RegEx developed accounted for the different formats in which these keywords appeared, including square brackets, parentheses, or no brackets at all, as well as stanzas separated only by double newline characters. The output of this step was, for each song record, a list of strings representing individual stanzas (each stanza has also a header with the corresponding keyword; this aspect will be discussed in the next paragraph). Next, uninformative strings—such as empty strings or those with fewer than 20 characters—were removed, as they were too short to provide meaningful content. As a result, the output of this preliminary preprocessing phase was a dataset where the records were no longer whole songs but individual stanzas, each numbered according to its position within the song.

A further and more detailed cleaning process on the stanzas involved the creation of the boolean feature `is_chorus`, which was assigned `true` for repeated stanzas within the same song or for stanzas with headers such as ”hook”, ”chorus”, ”refrain”, or ”bridge”. Next, stanza headers and newline characters between verses were removed to obtain cleaner stanzas. Since choruses, hooks, bridges, and refrains often repeat throughout songs, duplicate stanzas were discarded to avoid redundant data. This resulted in a dataset of cleaned, non-duplicate stanzas, which served as the checkpoint for the labeling step and the starting point for the text lemmatization process.
The subsequent step involved lemmatizing the stanzas using the `spaCy`[3] library. A list of lemmatized tokens was created by filtering out punctuation and empty words. Lemmatization was chosen over stemming because it produces more accurate and meaningful results, particularly for tasks requiring semantic understanding, such as the one at hand.
Since the dataset was not pre-labeled at the stanza level, ALBERT Base v2 was employed for this task. This transformer model is specifically designed to be fine-tuned on tasks that require an understanding of the entire sentence, such as sequence classification.

Static Models

The development of static models was straight forward, emphasizing simplicity. Static models were mainly developed to provide a performance comparison for the more complex Neural Networks. The two models share the same architectural framework: a preprocessing layer to handle the inputs, followed by a classifier for prediction.
The preprocessing layer processes both `title` and `lemmatized_stanzas` through Term Frequency-Inverse Document Frequency, to quantify term-wise importance within the dataset. The boolean attributes are converted to integers for compatibility with the model, while `stanza_number` is scaled to ensure uniformity and prevent bias due to its magnitude.
To optimize model performance, hyperparameter space exploration was conducted using Random Search. Cross-validation is also used in order to provide a more accurate estimate of model performance, ensuring that the results are less influenced by dataset splits. Additionally, cross-validation helps prevent overfitting, especially in cases of limited data availability or class imbalance.

Neural Networks

The Neural Network models can be trained with the `neural_networks.py` script. This script offers the possibility of configuring various parameters for training, which was used to test different configurations:

- `type`: specifies which type of Neural Network is to be trained. `1` is default, indicating One-Dimensional Convolutional Neural Networks training. Setting it to `2` trains a Recurrent Neural Network.
- `epochs`: Defines the number of epochs for training, allowing control over the duration of the learning process.
- `semisupervised`: Specifies the number of epochs dedicated to semi-supervised learning. If this parameter is omitted, the model is trained using only labeled data.
- `reset`: When included, the model is reset before starting training on newly pseudo-labeled data. This approach repurposes semi-supervised learning as a form of data augmentation rather than as transfer learning, providing an opportunity to enhance performance with pseudo-labeled examples.
- `even-labels`: When included, ensures even class representation in the dataset by downsampling overrepresented classes.

The architectures were developed and tuned through empirical, reiterated testing. These parameters helped with the process, and can be used for further experimentation.
Both the Recurrent Neural Network and the One-Dimensional Convolutional Neural Network share the same preprocessing architecture. Most attributes are processed in the same manner as in the Static Models; specific steps are applied to `lemmatized_stanzas` and `title`. Non-Negative Matrix Factorization is applied to `title` in addition to Term Frequency-Inverse Document Frequency to extract latent topics, providing a richer representation of the textual data. `lemmatized_stanzas` are handled by Convolutional and Recurrent pipelines of the two Networks. Elements are first tokenized, and then padded in order to get an input with consistent shape, which is essential for both types of recurrent layers.

One-Dimensional Convolutional Neural Network

The Convolutional part of the architecture is designed to extract and learn local patterns in `embedding_lyrics`. Its structure consists of three convolutional layers, each applying filters of varying sizes. This allows to detect patterns at different granularities. These layers are followed by Global Max Pooling, to reduce the previous output's dimension to a fixed-length vector, as well as retaining focus on the most informative patterns. A dropout layer is then applied, to introduce regularization and prevent overfitting.

Recurrent Neural Network

The Recurrent part of the architecture is designed to extract and learn local patterns in `embedding_lyrics`. Its structure consists of three gated recurrent units (`GRU`) to model temporal relationships. These are progressively formed by smaller numbers of units, in order to capture patterns. All three layers in the architecture use the `tanh` activation function to compute the hidden state and the `sigmoid` activation function for the recurrent gate. The first and second layers return the full sequence of hidden states for each time step in the input sequence, enabling richer learning of patterns over time. Dropout is applied on every layer, to prevent overfitting and add regularization.

Shared Components

The remaining features are handled by a simple pipeline, which concatenates their Input layers. The inputs are passed them through a dense layer to create a compact representation. The output of the lyrics-processing branch is concatenated with the processed additional features. The combined representation is passed through a Dense layer with 32 units (ReLU activation), followed by a Dropout layer with a rate of 0.3. Finally, the output layer uses 8 units with a softmax activation, corresponding to the classification into 8 emotion categories.

Evaluation

Discussion and Conclusions

Bibliography

[1] *Genius Song Lyrics*. URL: https://www.kaggle.com/datasets/carlosgdcj/genius-song-lyrics-with-language-information?select=song_lyrics.csv.

[2] *Albert Base v2*. URL: https://huggingface.co/albert/albert-base-v2.

[3] *spaCy*. URL: https://spacy.io.

List of figures

1 Plutchik's eight primary emotions 2