Babeș Bolyai University Cluj-Napoca

Lyrics Sentiment Analysis

Group Members:

Papuc Maria

Pargea Larisa

Șoaita Monica

Milosteanu Andrei

Nemeș Călin

2024

Table of Contents

1. Introduction

In this project, we explore the fascinating field of sentiment analysis on song lyrics with the goal of interpreting the emotional undertones present in a variety of musical compositions from various genres and eras. We categorize lyrics into (very) positive, (very) negative, and neutral sentiments in order to provide a more in-depth analysis of the emotional nuances contained in the linguistic fabric of music.

We chose this interesting subject because we have found that there is unrealized potential in large-scale music lyrics. With the use of data mining, we can carefully identify, evaluate, and extrapolate trends, revealing the emotional stories beneath the surface that could otherwise go unnoticed. Motivated by a natural curiosity, this research explores the rich tapestry of emotions entwined with the poetic legacy of music in addition to exploring the technical aspects of sentiment analysis. The attraction of data mining, as we traverse this fusion of technology and creativity, is its ability to convert unprocessed data into a sophisticated comprehension of the deep feelings enshrined in the poetic lines of our beloved songs.

2. Description

When we first started this research, we used the Naive Bayes approach to create our early models and explore the topic of sentiment analysis on song lyrics. This project involved creating two different Python and Java implementations. The Multinomial Naive Bayes classifier was used in the Python implementation, which has a thorough preparation pipeline. To improve the precision of sentiment classification, this involved removing tags, getting rid of stopwords, lemmatizing, and stemming. The Java implementation used stemming algorithms, a stopwords remover, and a Lucene index all at the same time.

We conducted a thorough evaluation process after the initial implementations to determine the efficacy of our models. We used NDCG (Normalized Discounted Cumulative Gain) as our evaluation metric since it gave us a numerical indicator of how well our sentiment predictions ranked.

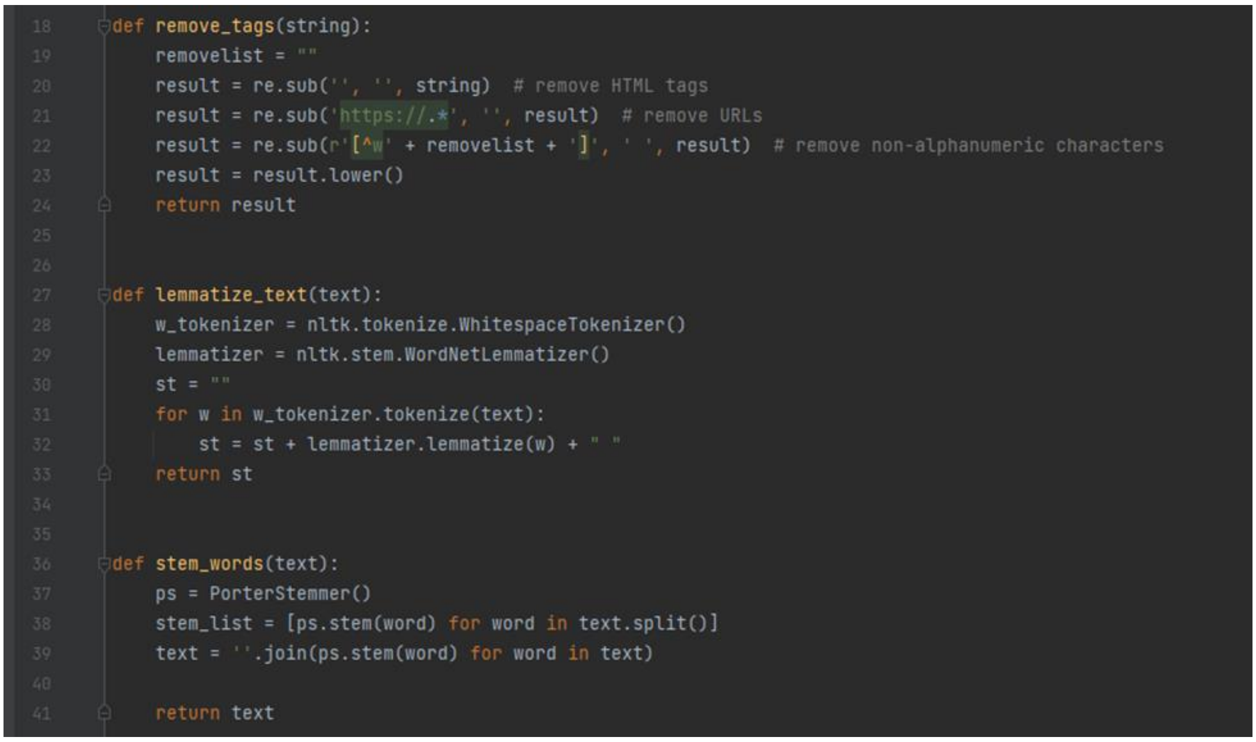
We used the StanfordNLP library to create a new and enhanced model as part of our quest for improvement. Thanks to this library's natural language processing features, our

sentiment analysis has become much more sophisticated. The updated version demonstrated improved precision and effectiveness in identifying emotions in song lyrics.
Our dedication to user accessibility and engagement extended beyond the domain of model development, resulting in the construction of an interactive frontend using React. This frontend not only smoothly connects with our latest sentiment analysis technology but also lets users to dynamically explore and interpret the emotional feelings of their favorite music.

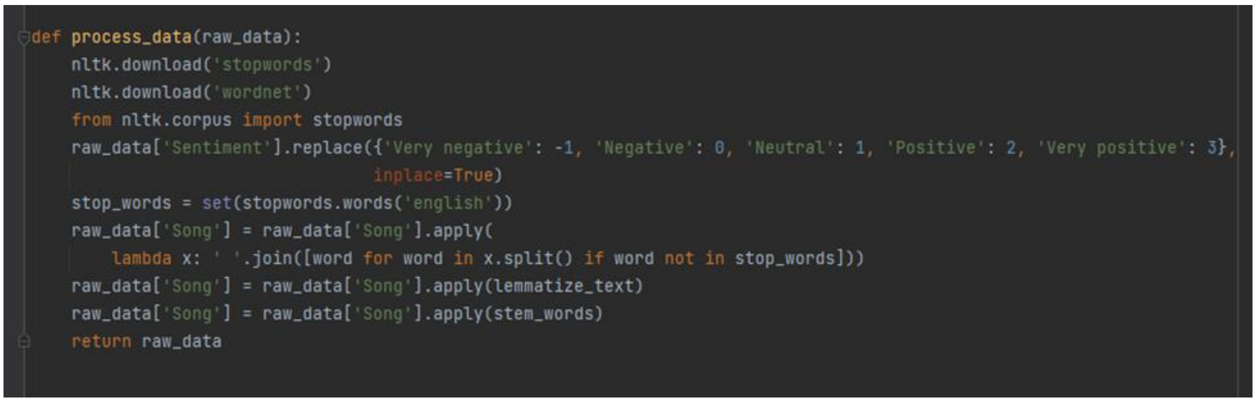
3. Indexing & Retrieval/Initial implementation

When we first started this project, we took a two-pronged approach, using models for sentiment analysis in both Python and Java.

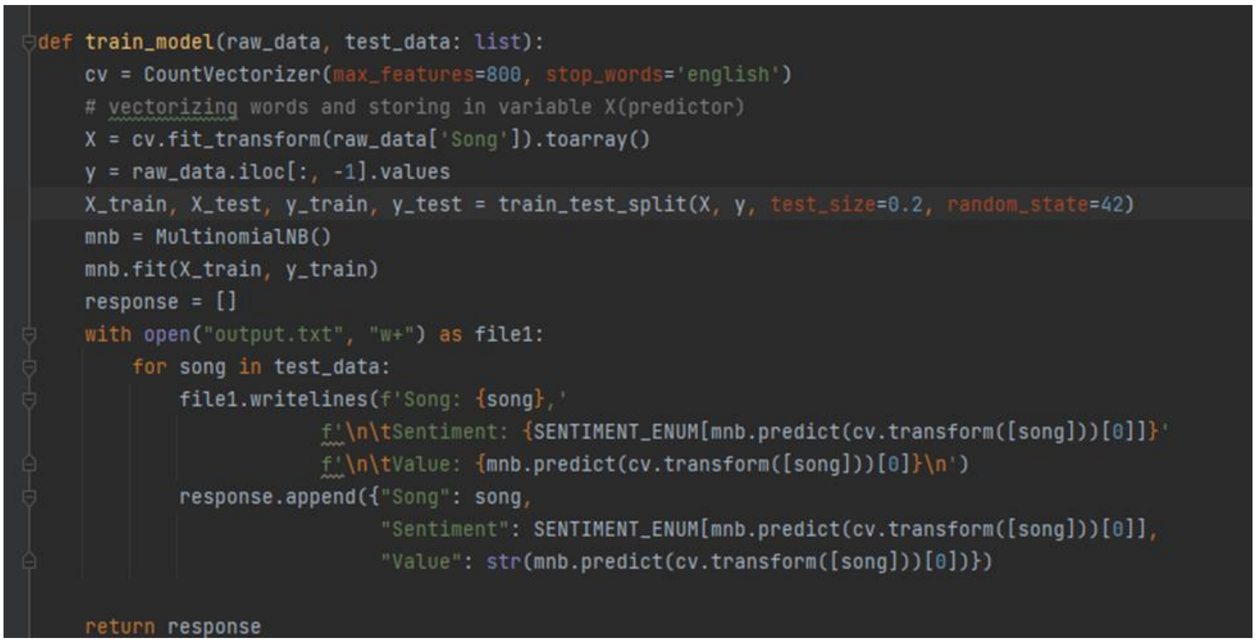
In the Python implementation, the training set's careful preprocessing was the main priority. In order to create a cleaner dataset, this required removing HTML tags, URLs, and non-alphanumeric characters. Furthermore, sentiment labels were substituted with equivalent numerical representations and stopwords were removed. The preprocessed data was then used to train a Multinomial Naive Bayes classifier, and testing was done to evaluate the model's performance. Sentiment predictions from the final output were saved in a text file for more thorough examination.



```python
def remove_tags(string):
    removelist = ""
    result = re.sub('', '', string)  # remove HTML tags
    result = re.sub('https://.*', '', result)  # remove URLs
    result = re.sub(r'[^w' + removelist + ']', ' ', result)  # remove non-alphanumeric characters
    result = result.lower()
    return result


def lemmatize_text(text):
    w_tokenizer = nltk.tokenize.WhitespaceTokenizer()
    lemmatizer = nltk.stem.WordNetLemmatizer()
    st = ""
    for w in w_tokenizer.tokenize(text):
        st = st + lemmatizer.lemmatize(w) + " "
    return st


def stem_words(text):
    ps = PorterStemmer()
    stem_list = [ps.stem(word) for word in text.split()]
    text = ''.join(ps.stem(word) for word in text)

    return text
```

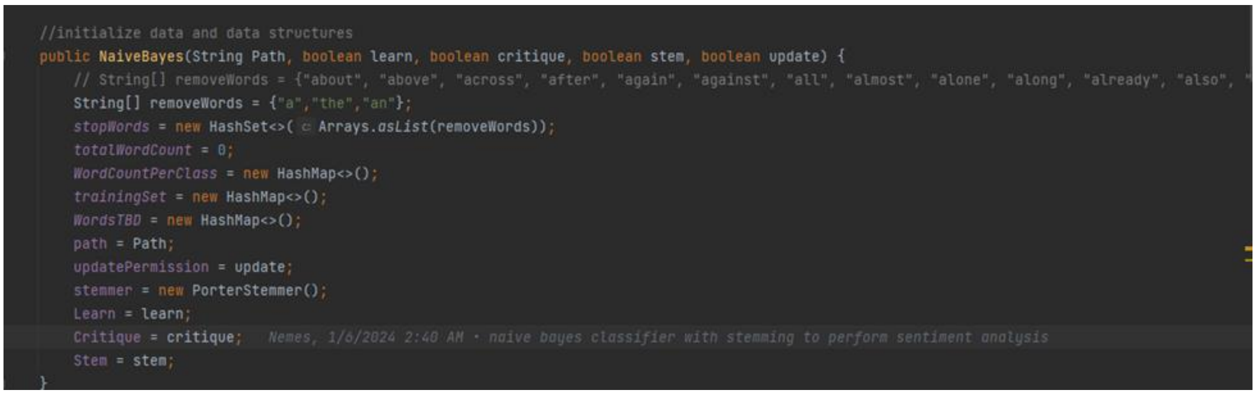


```python
def process_data(raw_data):
    nltk.download('stopwords')
    nltk.download('wordnet')
    from nltk.corpus import stopwords
    raw_data['Sentiment'].replace({'Very negative': -1, 'Negative': 0, 'Neutral': 1, 'Positive': 2, 'Very positive': 3},
                                  inplace=True)
    stop_words = set(stopwords.words('english'))
    raw_data['Song'] = raw_data['Song'].apply(
        lambda x: ' '.join([word for word in x.split() if word not in stop_words]))
    raw_data['Song'] = raw_data['Song'].apply(lemmatize_text)
    raw_data['Song'] = raw_data['Song'].apply(stem_words)
    return raw_data
```

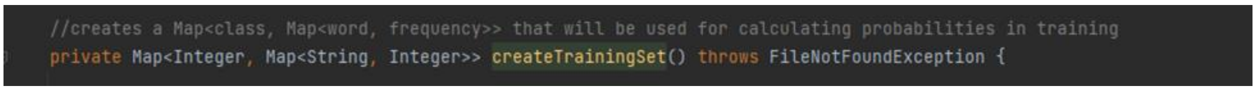


```python
def train_model(raw_data, test_data: list):
    cv = CountVectorizer(max_features=800, stop_words='english')
    # vectorizing words and storing in variable X(predictor)
    X = cv.fit_transform(raw_data['Song']).toarray()
    y = raw_data.iloc[:, -1].values
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
    mnb = MultinomialNB()
    mnb.fit(X_train, y_train)
    response = []
    with open("output.txt", "w+") as file1:
        for song in test_data:
            file1.writelines(f'Song: {song},'
                             f'\n\tSentiment: {SENTIMENT_ENUM[mnb.predict(cv.transform([song]))[0]]}'
                             f'\n\tValue: {mnb.predict(cv.transform([song]))[0]}\n')
            response.append({"Song": song,
                             "Sentiment": SENTIMENT_ENUM[mnb.predict(cv.transform([song]))[0]],
                             "Value": str(mnb.predict(cv.transform([song]))[0])})

    return response
```

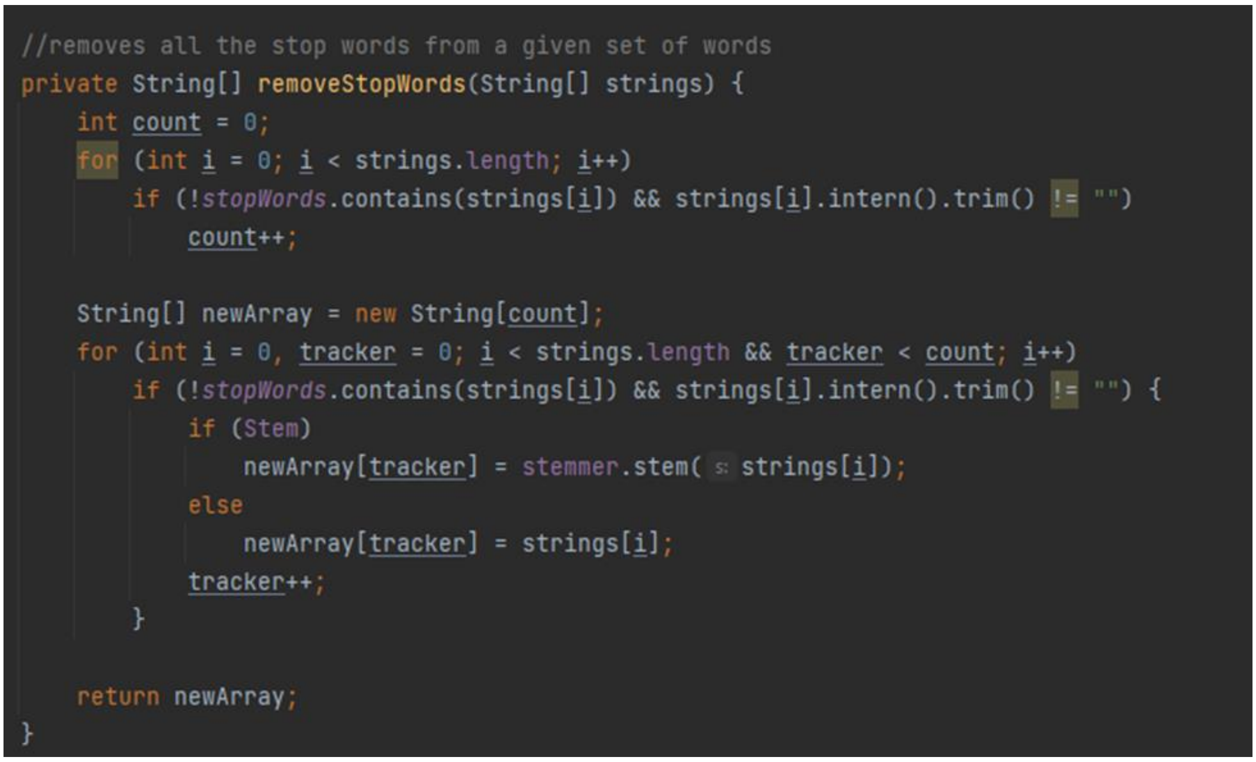
Concurrently, the Java implementation took a similar path, emphasizing configurability to accommodate different tastes. A constructor in the code let users set up the model by selecting between stemming and non-stemming options. The method efficiently indexed the dataset by stemming it using a Porter Stemmer that was based on Lucene. The preparation pipeline included eliminating stopwords and other superfluous components, just like in its counterpart in Python. The accuracy of the model was carefully assessed, offering insights into how well it performed in various setups.



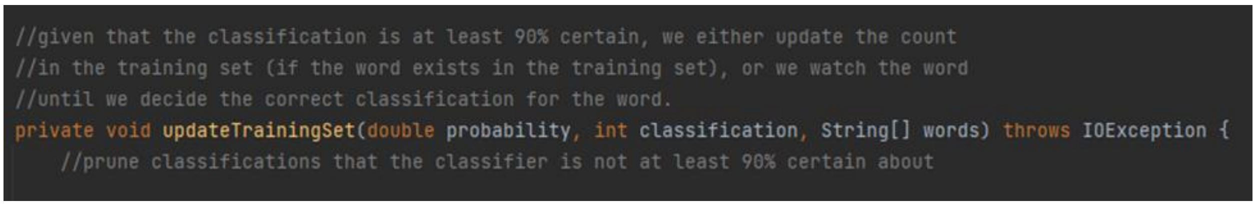
```java
//initialize data and data structures
public NaiveBayes(String Path, boolean learn, boolean critique, boolean stem, boolean update) {
    // String[] removeWords = {"about", "above", "across", "after", "again", "against", "all", "almost", "alone", "along", "already", "also", "
    String[] removeWords = {"a","the","an"};
    stopWords = new HashSet<>( c Arrays.asList(removeWords));
    totalWordCount = 0;
    WordCountPerClass = new HashMap<>();
    trainingSet = new HashMap<>();
    WordsTBD = new HashMap<>();
    path = Path;
    updatePermission = update;
    stemmer = new PorterStemmer();
    Learn = learn;
    Critique = critique;   Nemes, 1/6/2024 2:40 AM • naive bayes classifier with stemming to perform sentiment analysis
    Stem = stem;
}
```



```java
//creates a Map<class, Map<word, frequency>> that will be used for calculating probabilities in training
private Map<Integer, Map<String, Integer>> createTrainingSet() throws FileNotFoundException {
```



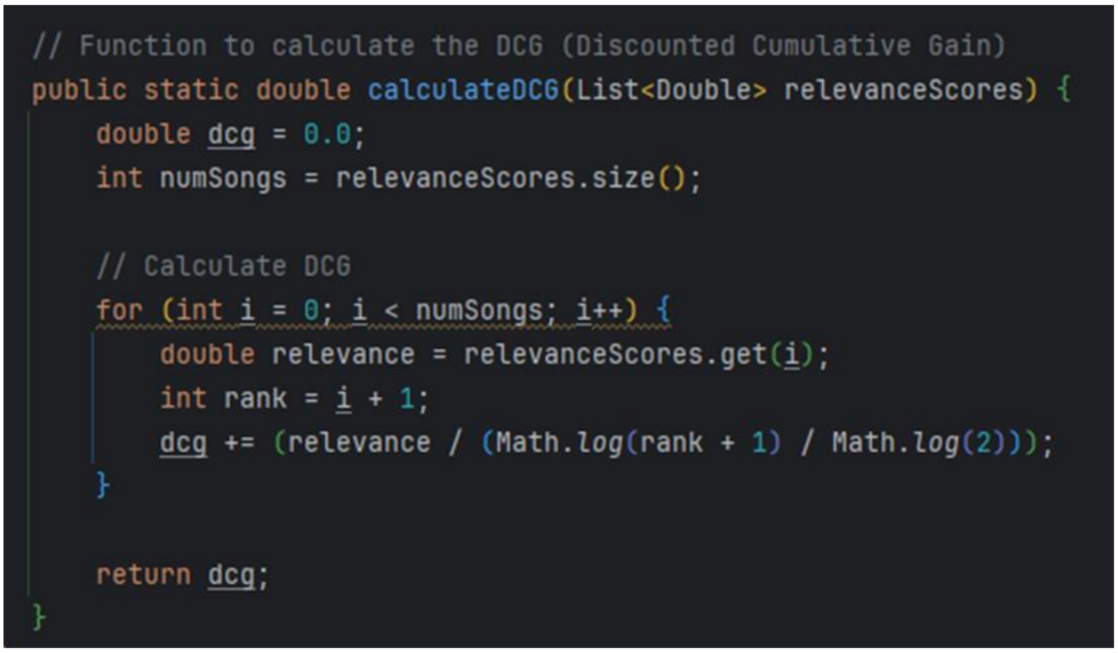
```java
//removes all the stop words from a given set of words
private String[] removeStopWords(String[] strings) {
    int count = 0;
    for (int i = 0; i < strings.length; i++)
        if (!stopWords.contains(strings[i]) && strings[i].intern().trim() != "")
            count++;

    String[] newArray = new String[count];
    for (int i = 0, tracker = 0; i < strings.length && tracker < count; i++)
        if (!stopWords.contains(strings[i]) && strings[i].intern().trim() != "") {
            if (Stem)
                newArray[tracker] = stemmer.stem( s: strings[i]);
            else
                newArray[tracker] = strings[i];
            tracker++;
        }

    return newArray;
}
```



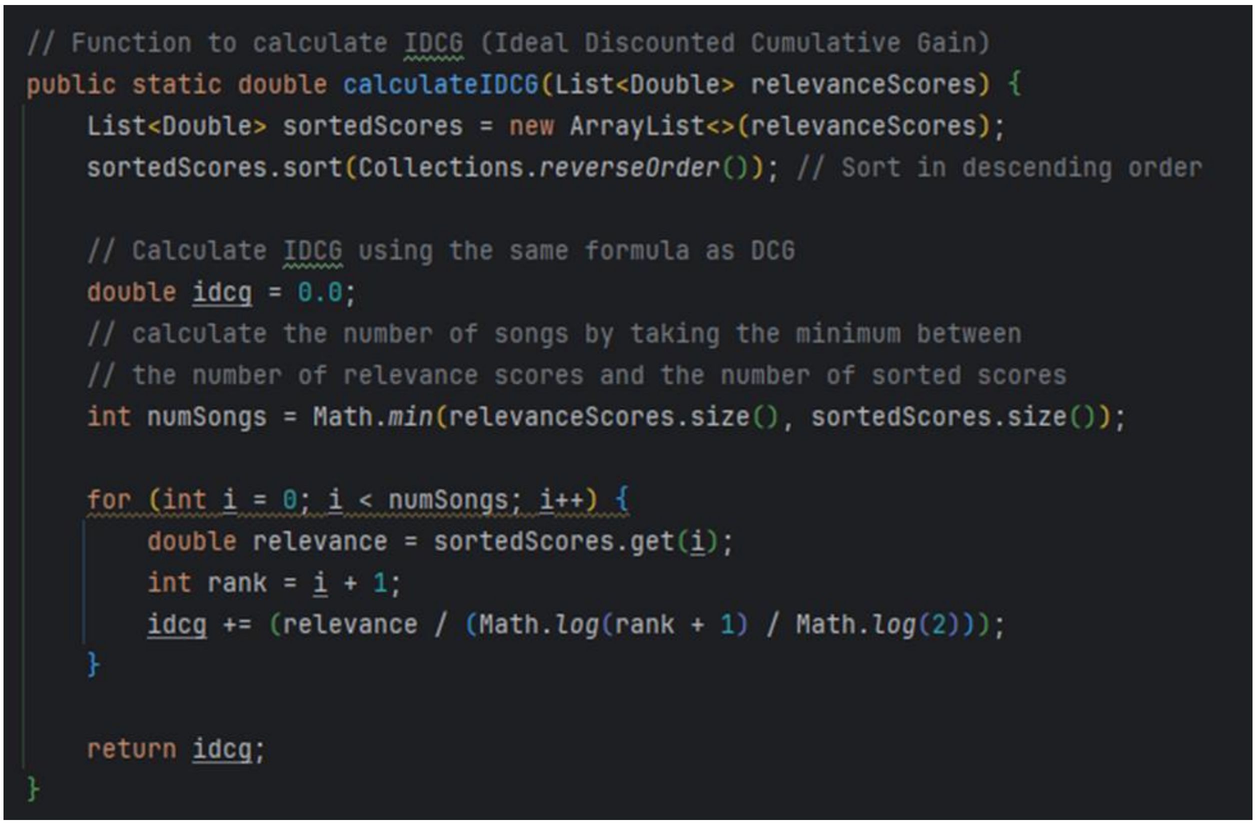
```java
//given that the classification is at least 90% certain, we either update the count
//in the training set (if the word exists in the training set), or we watch the word
//until we decide the correct classification for the word.
private void updateTrainingSet(double probability, int classification, String[] words) throws IOException {
    //prune classifications that the classifier is not at least 90% certain about
```

4. Measuring Performance/Evaluation

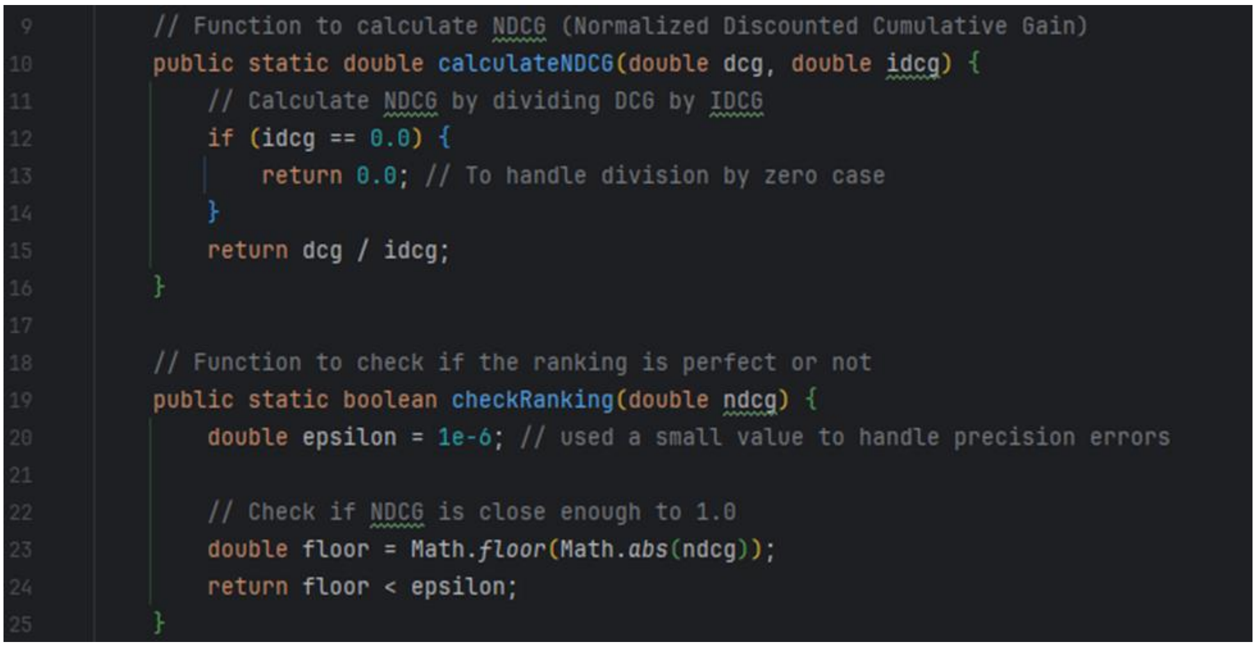
We used three primary measures in our evaluation process: Normalized Discounted Cumulative Gain (NDCG), Ideal Discounted Cumulative Gain (IDCG), and Discounted Cumulative Gain (DCG). All implementations, including both Python and Java versions, underwent rigorous evaluation using these metrics. NDCG, in particular, served as a comprehensive measure of the ranking quality of our sentiment predictions, considering the ideal scenario and normalizing the results for meaningful comparison across diverse models and configurations.



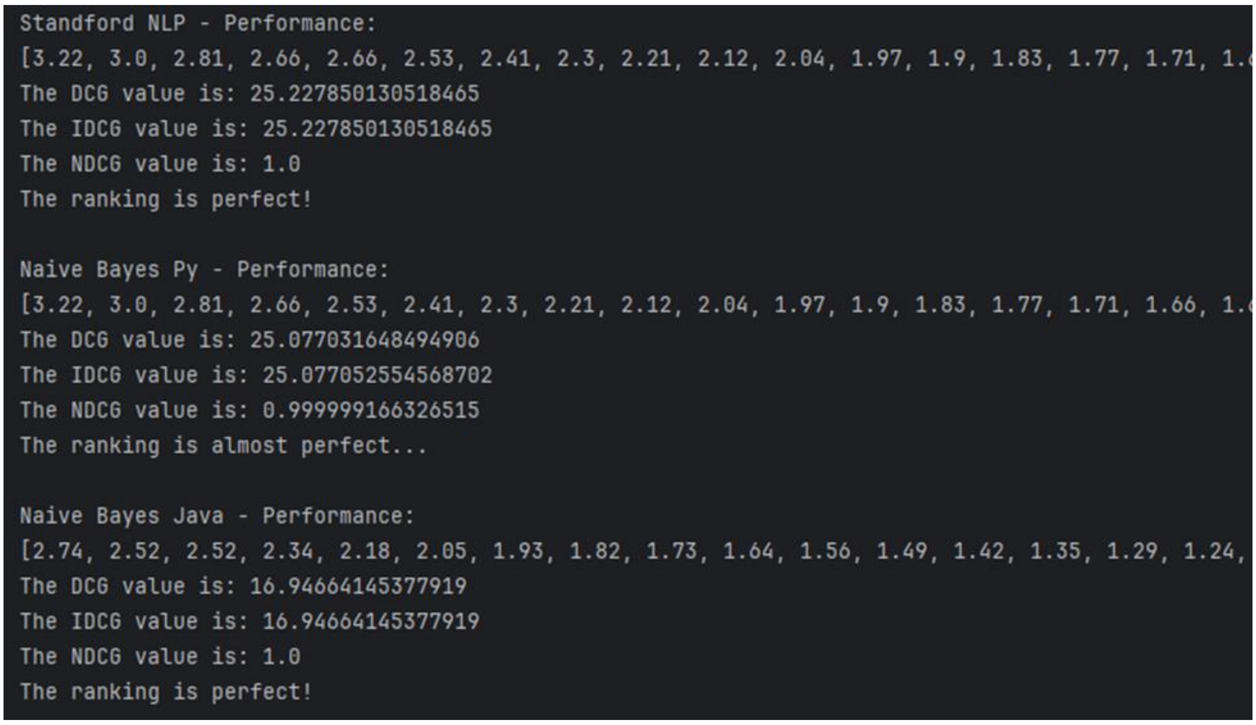
```java
// Function to calculate the DCG (Discounted Cumulative Gain)
public static double calculateDCG(List<Double> relevanceScores) {
    double dcg = 0.0;
    int numSongs = relevanceScores.size();

    // Calculate DCG
    for (int i = 0; i < numSongs; i++) {
        double relevance = relevanceScores.get(i);
        int rank = i + 1;
        dcg += (relevance / (Math.log(rank + 1) / Math.log(2)));
    }

    return dcg;
}
```



```java
// Function to calculate IDCG (Ideal Discounted Cumulative Gain)
public static double calculateIDCG(List<Double> relevanceScores) {
    List<Double> sortedScores = new ArrayList<>(relevanceScores);
    sortedScores.sort(Collections.reverseOrder()); // Sort in descending order

    // Calculate IDCG using the same formula as DCG
    double idcg = 0.0;
    // calculate the number of songs by taking the minimum between
    // the number of relevance scores and the number of sorted scores
    int numSongs = Math.min(relevanceScores.size(), sortedScores.size());

    for (int i = 0; i < numSongs; i++) {
        double relevance = sortedScores.get(i);
        int rank = i + 1;
        idcg += (relevance / (Math.log(rank + 1) / Math.log(2)));
    }

    return idcg;
}
```



```java
    // Function to calculate NDCG (Normalized Discounted Cumulative Gain)
    public static double calculateNDCG(double dcg, double idcg) {
        // Calculate NDCG by dividing DCG by IDCG
        if (idcg == 0.0) {
            return 0.0; // To handle division by zero case
        }
        return dcg / idcg;
    }

    // Function to check if the ranking is perfect or not
    public static boolean checkRanking(double ndcg) {
        double epsilon = 1e-6; // used a small value to handle precision errors

        // Check if NDCG is close enough to 1.0
        double floor = Math.floor(Math.abs(ndcg));
        return floor < epsilon;
    }
```



```
Standford NLP - Performance:
[3.22, 3.0, 2.81, 2.66, 2.66, 2.53, 2.41, 2.3, 2.21, 2.12, 2.04, 1.97, 1.9, 1.83, 1.77, 1.71, 1.
The DCG value is: 25.227850130518465
The IDCG value is: 25.227850130518465
The NDCG value is: 1.0
The ranking is perfect!

Naive Bayes Py - Performance:
[3.22, 3.0, 2.81, 2.66, 2.53, 2.41, 2.3, 2.21, 2.12, 2.04, 1.97, 1.9, 1.83, 1.77, 1.71, 1.66, 1.
The DCG value is: 25.077031648494906
The IDCG value is: 25.077052554568702
The NDCG value is: 0.999999166326515
The ranking is almost perfect...

Naive Bayes Java - Performance:
[2.74, 2.52, 2.52, 2.34, 2.18, 2.05, 1.93, 1.82, 1.73, 1.64, 1.56, 1.49, 1.42, 1.35, 1.29, 1.24,
The DCG value is: 16.94664145377919
The IDCG value is: 16.94664145377919
The NDCG value is: 1.0
The ranking is perfect!
```

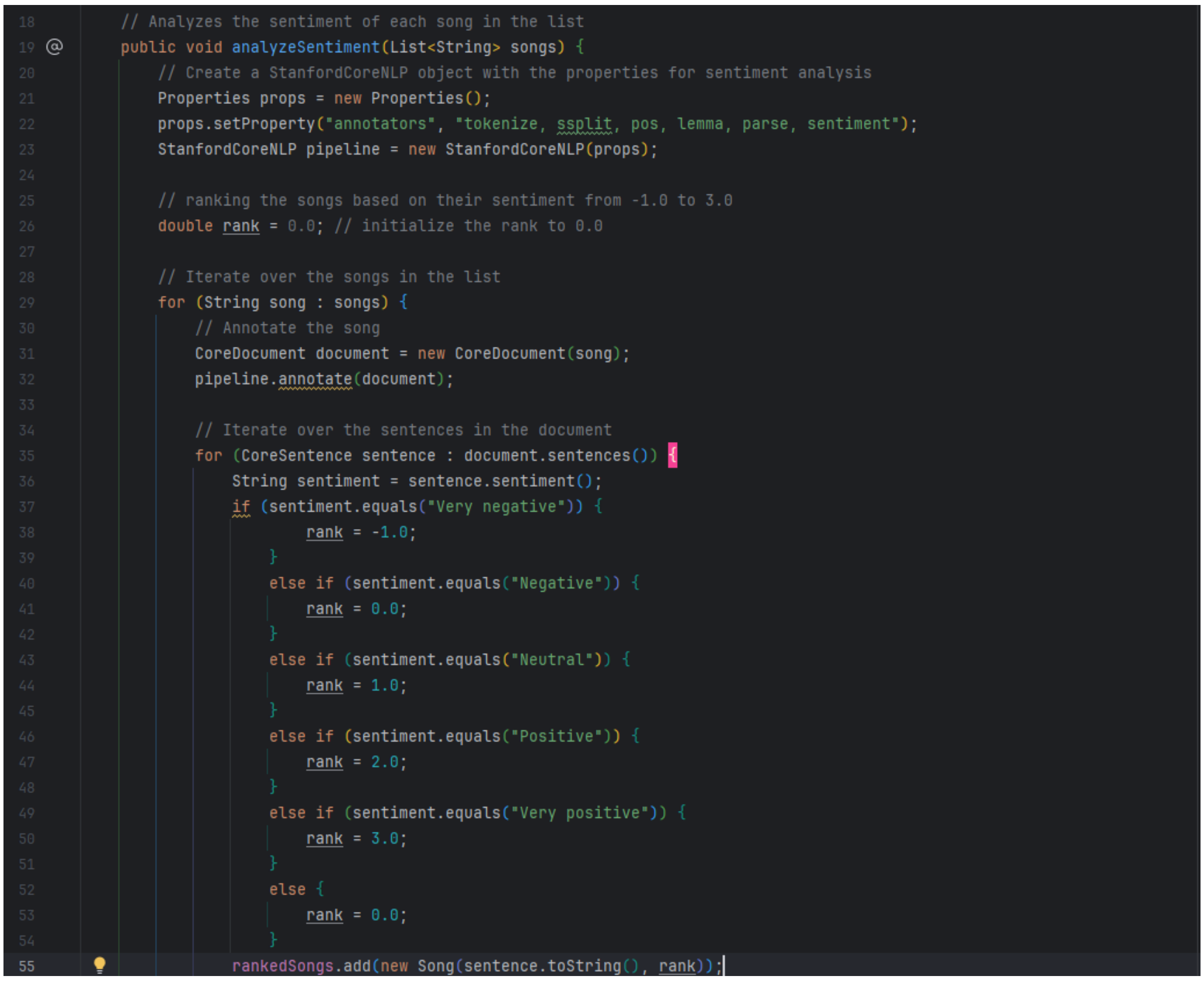
5. Error analysis

A thorough evaluation of our top sentiment analysis system's error analysis produced the following insights:

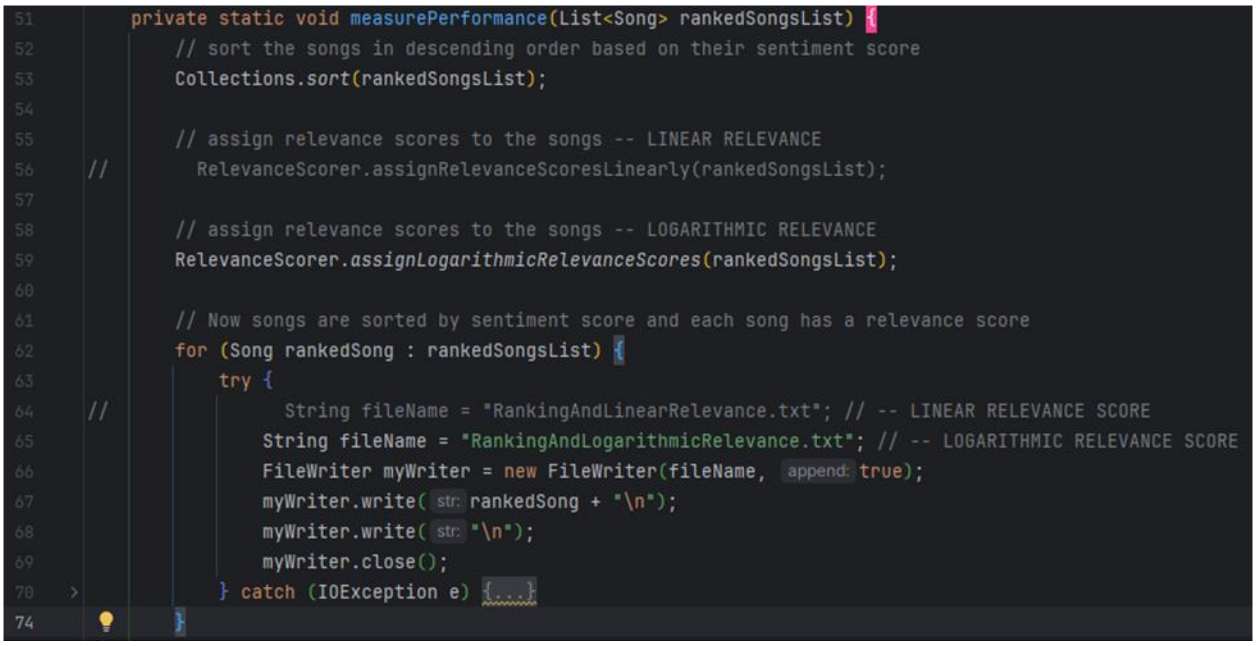
- Correct Sentiments: A considerable proportion of the lyrics were classified as having the correct sentiment, demonstrating how well the algorithm captures emotional subtleties.
- Semantic Ambiguity: Some errors can be attributed to the inherent semantic ambiguity within certain lyrics, where the sentiment may be subject to interpretation based on individual perspectives.
- Linguistic Nuances: Complex linguistic nuances and figurative language presented difficulties since the model might find it difficult to interpret delicate emotional cues concealed in poetic or metaphorical terms.
- Genre-specific Challenges: The efficiency of sentiment analysis differed throughout musical genres. Some genres posed difficulties for the model because of their unusual lyric structures or wide ranges of emotions.
- Data Imbalance: Imbalances in the training dataset, with certain sentiments being more prevalent than others, may lead to a bias in predictions, impacting the model's ability to generalize accurately.

6. Improved approach

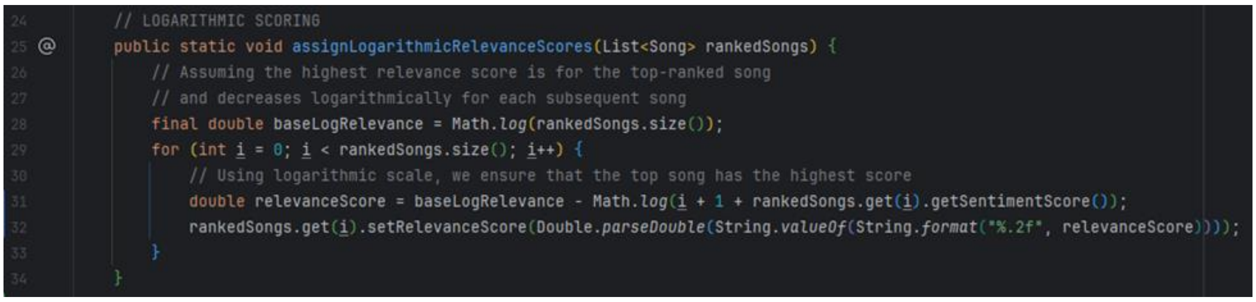
We included the StanfordNLP Java dependency into our system in our search for a better sentiment analysis method. With the use of this advanced natural language processing technology, our model was able to better capture complex linguistic elements and predict sentiment for song lyrics with greater accuracy. Crucially, we assigned a sentiment score to each song using the StanfordNLP analysis in order to rank them. We maintained the relationship between every music and its accompanying sentiment score by saving the outcomes of the sentiment ranking in a text file. Next, we combined the sentiment prediction with the ranking determined by the StanfordNLP study to create a relevance score for every song. In order to evaluate our improved method in its entirety, we utilized the NDCG metric.



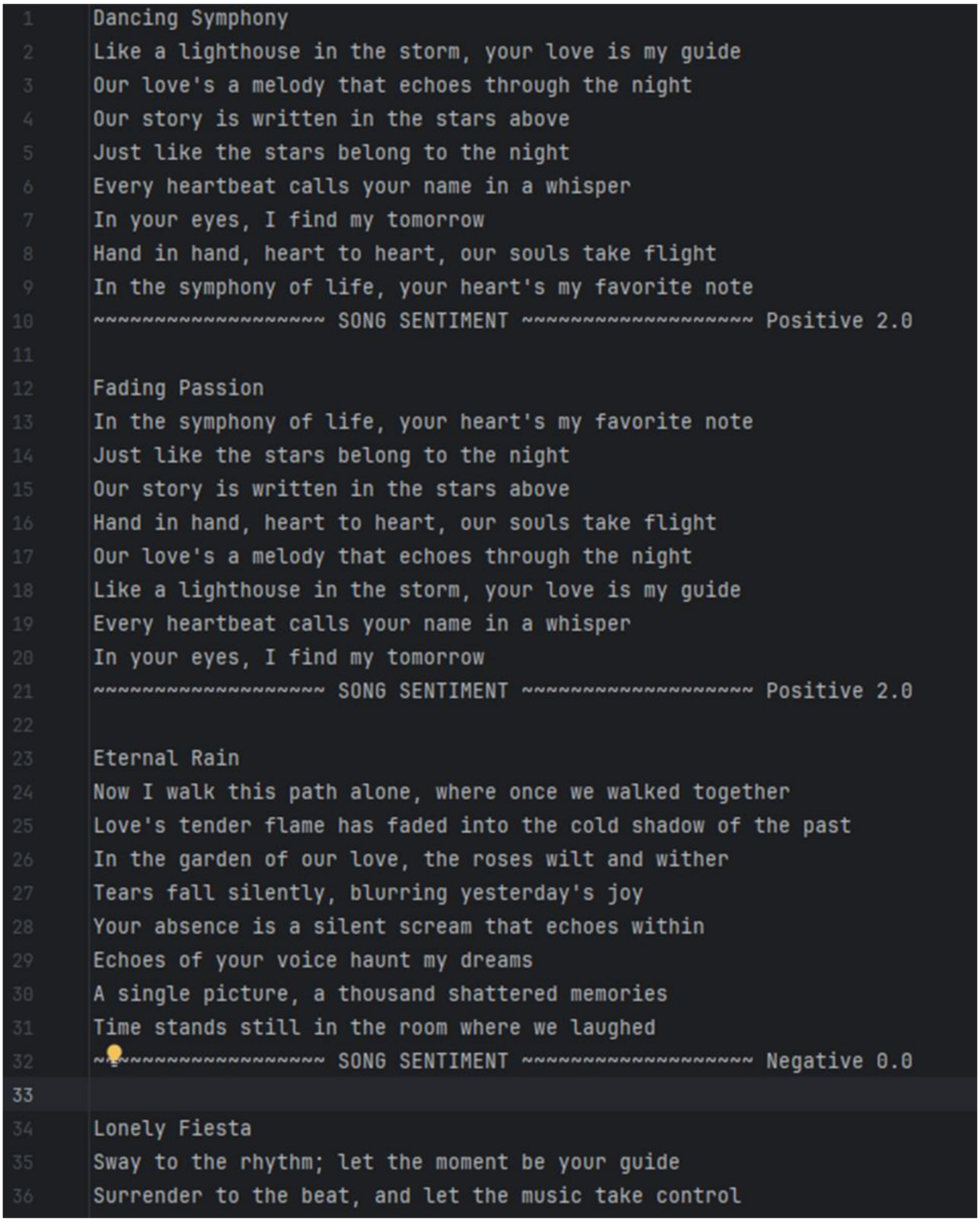
```java
    // Analyzes the sentiment of each song in the list
    public void analyzeSentiment(List<String> songs) {
        // Create a StanfordCoreNLP object with the properties for sentiment analysis
        Properties props = new Properties();
        props.setProperty("annotators", "tokenize, ssplit, pos, lemma, parse, sentiment");
        StanfordCoreNLP pipeline = new StanfordCoreNLP(props);

        // ranking the songs based on their sentiment from -1.0 to 3.0
        double rank = 0.0; // initialize the rank to 0.0

        // Iterate over the songs in the list
        for (String song : songs) {
            // Annotate the song
            CoreDocument document = new CoreDocument(song);
            pipeline.annotate(document);

            // Iterate over the sentences in the document
            for (CoreSentence sentence : document.sentences()) {
                String sentiment = sentence.sentiment();
                if (sentiment.equals("Very negative")) {
                        rank = -1.0;
                    }
                    else if (sentiment.equals("Negative")) {
                        rank = 0.0;
                    }
                    else if (sentiment.equals("Neutral")) {
                        rank = 1.0;
                    }
                    else if (sentiment.equals("Positive")) {
                        rank = 2.0;
                    }
                    else if (sentiment.equals("Very positive")) {
                        rank = 3.0;
                    }
                    else {
                        rank = 0.0;
                    }
                rankedSongs.add(new Song(sentence.toString(), rank));
```



```java
    private static void measurePerformance(List<Song> rankedSongsList) {
        // sort the songs in descending order based on their sentiment score
        Collections.sort(rankedSongsList);

        // assign relevance scores to the songs -- LINEAR RELEVANCE
//          RelevanceScorer.assignRelevanceScoresLinearly(rankedSongsList);

        // assign relevance scores to the songs -- LOGARITHMIC RELEVANCE
        RelevanceScorer.assignLogarithmicRelevanceScores(rankedSongsList);

        // Now songs are sorted by sentiment score and each song has a relevance score
        for (Song rankedSong : rankedSongsList) {
            try {
//                  String fileName = "RankingAndLinearRelevance.txt"; // -- LINEAR RELEVANCE SCORE
                String fileName = "RankingAndLogarithmicRelevance.txt"; // -- LOGARITHMIC RELEVANCE SCORE
                FileWriter myWriter = new FileWriter(fileName,  append: true);
                myWriter.write( str: rankedSong + "\n");
                myWriter.write( str: "\n");
                myWriter.close();
            } catch (IOException e) {...}
        }
```



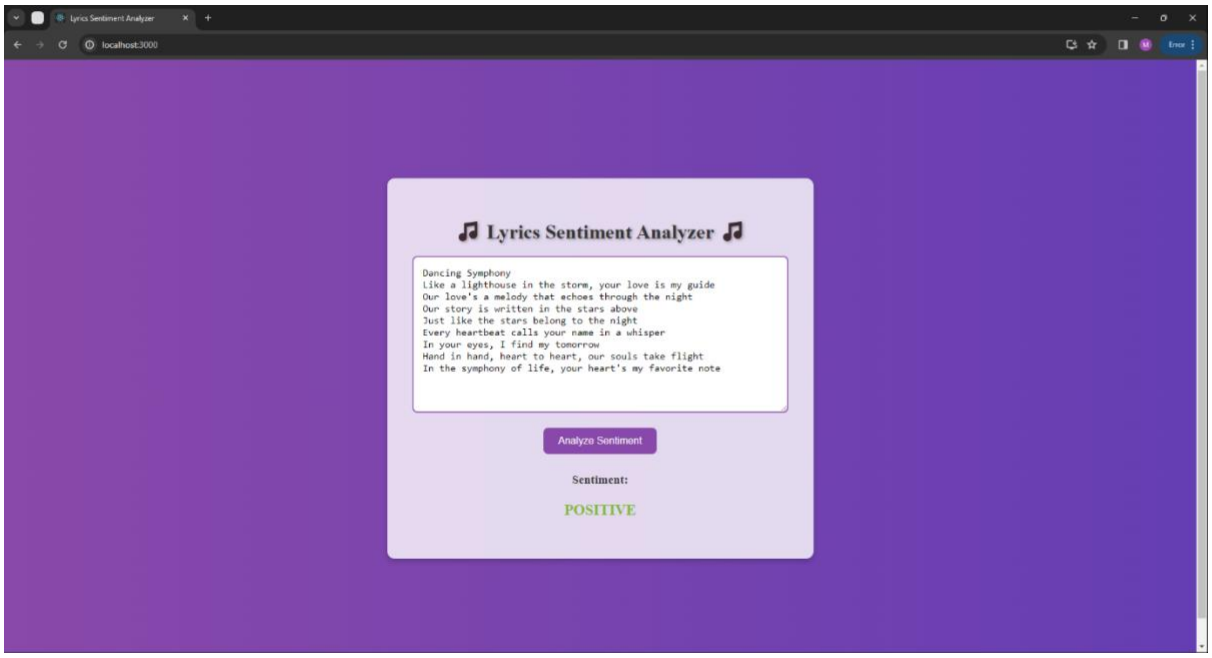
```java
// LOGARITHMIC SCORING
public static void assignLogarithmicRelevanceScores(List<Song> rankedSongs) {
    // Assuming the highest relevance score is for the top-ranked song
    // and decreases logarithmically for each subsequent song
    final double baseLogRelevance = Math.log(rankedSongs.size());
    for (int i = 0; i < rankedSongs.size(); i++) {
        // Using logarithmic scale, we ensure that the top song has the highest score
        double relevanceScore = baseLogRelevance - Math.log(i + 1 + rankedSongs.get(i).getSentimentScore());
        rankedSongs.get(i).setRelevanceScore(Double.parseDouble(String.valueOf(String.format("%.2f", relevanceScore))));
    }
}
```



```
Dancing Symphony
Like a lighthouse in the storm, your love is my guide
Our love's a melody that echoes through the night
Our story is written in the stars above
Just like the stars belong to the night
Every heartbeat calls your name in a whisper
In your eyes, I find my tomorrow
Hand in hand, heart to heart, our souls take flight
In the symphony of life, your heart's my favorite note
~~~~~~~~~~~~~~~~~~~ SONG SENTIMENT ~~~~~~~~~~~~~~~~~~~ Positive 2.0

Fading Passion
In the symphony of life, your heart's my favorite note
Just like the stars belong to the night
Our story is written in the stars above
Hand in hand, heart to heart, our souls take flight
Our love's a melody that echoes through the night
Like a lighthouse in the storm, your love is my guide
Every heartbeat calls your name in a whisper
In your eyes, I find my tomorrow
~~~~~~~~~~~~~~~~~~~ SONG SENTIMENT ~~~~~~~~~~~~~~~~~~~ Positive 2.0

Eternal Rain
Now I walk this path alone, where once we walked together
Love's tender flame has faded into the cold shadow of the past
In the garden of our love, the roses wilt and wither
Tears fall silently, blurring yesterday's joy
Your absence is a silent scream that echoes within
Echoes of your voice haunt my dreams
A single picture, a thousand shattered memories
Time stands still in the room where we laughed
~~~~~~~~~~~~~~~~~~~ SONG SENTIMENT ~~~~~~~~~~~~~~~~~~~ Negative 0.0

Lonely Fiesta
Sway to the rhythm; let the moment be your guide
Surrender to the beat, and let the music take control
```

7. Frontend

In addition to refining our sentiment analysis model, we've introduced an interactive React frontend that allows users to seamlessly explore the emotional sentiment of songs using our latest and most enhanced implementation.



```
Dancing Symphony
Like a lighthouse in the storm, your love is my guide
Our love's a melody that echoes through the night
Our story is written in the stars above
Just like the stars belong to the night
Every heartbeat calls your name in a whisper
In your eyes, I find my tomorrow
Hand in hand, heart to heart, our souls take flight
In the symphony of life, your heart's my favorite note
```

Sentiment:

POSITIVE

8. Conclusions

In concise terms, our project, driven by data mining methodologies, embarked on a nuanced journey through Python and Java implementations, StanfordNLP integration, and the development of a user-friendly React frontend. By dissecting the emotional threads within song lyrics, our endeavor aimed to elevate sentiment analysis, offering a sophisticated yet accessible lens into the intricate emotional landscapes encapsulated in musical compositions.

9. Contributions

- Initial implementation / Indexing and retrieval: Milosteanu Andrei & Nemeș Călin
- Measuring Performance / Evaluation: Șoaita Monica
- Error analysis: all members
- Improved approach: Papuc Maria, Pargea Larisa & Șoaita Monica
- Frontend: Pargea Larisa
- README file: all members
- Project report: all members
- Presentation: Papuc Maria
- Video: all members