Indian Institute of Information Technology Bhopal

Lab Assignment – Design and Analysis of Algorithms (DAA)

Social Media Sentiment Analysis

Project Report

Department- Information Technology

Submitted By:

Jatin Wadhwani(23U030**51**)

Yash Zinzala (23U030**60**)

Submitted To:

Dr Yatendra Sahu

Asst. Proffesor IIIT Bhopal

Title: Social Media Sentiment Analysis Tool

Abstract

This project introduces a Social Media Sentiment Analysis Tool designed to evaluate public sentiment on social media platforms, developed using C++. Social media has become a central platform for people to express their opinions and share experiences, creating a large pool of valuable data that can be analyzed for sentiment. This tool is built to address this need by offering a fast and efficient method of determining the emotional tone in social media posts through a rule-based keyword matching and scoring approach.

The tool utilizes a predefined dictionary of sentiment-laden keywords, each associated with a positive, negative, or neutral score. By matching words in each social media post to these keywords, the tool calculates an overall sentiment score that categorizes each post accordingly. Additionally, the tool is capable of tracking sentiment trends over time, providing insights into shifts in public opinion on specific topics or brands. This allows organizations, researchers, and businesses to keep up with changing sentiments, helping them respond to public needs and concerns more effectively. The lightweight, rule-based approach ensures that the tool is resource-efficient, adaptable, and suitable for real-time sentiment analysis applications. This makes it a practical choice for those needing an accessible yet reliable solution to monitor public sentiment in dynamic social media environments.

With this project, we hope to offer a dependable, accurate, and efficient approach to sentiment data analysis. Because of its scalable design, the tool can effectively handle large keyword matching using optimized failure connections inside a Trie structure. By creating this tool, we hope to close a big gap in real-time sentiment analysis, particularly in market research, crisis management, and brand monitoring, by giving stakeholders from a variety of industries useful, instant insights.

Introduction

Social media platforms have revolutionized the way people voice concerns, share experiences, and express opinions. These platforms, which have billions of active users, include a vast amount of data about public opinion on a wide range of subjects. Organizations can learn about public opinion, consumer feedback, and even societal trends by examining this attitude. The sheer volume and diversity of emotional communication on social media, however, makes it difficult to identify significant patterns in the massive amounts of unstructured text data via social media poses several difficulties, chiefly due to the vast amount and diversity of emotional expressions.

Opinion mining, also known as sentiment analysis, provides a way to evaluate public sentiments by classifying text data as neutral, negative, or positive. For fields where real-time feedback is crucial, like marketing, politics, and crisis management, this interpretation has important ramifications. To handle the enormous, constantly expanding amounts of social media content, conventional sentiment analysis techniques like manual tagging and basic keyword-based filters are no longer enough. Therefore, there is a need for sophisticated, scalable solutions that can quickly and precisely identify subtle emotion patterns.

As social media websites keep evolving and slowly become the source of all kinds of information, people have started posting their opinions on various topics, discussions, issues, and complaints, expressing negative, positive, or neutral emotions in response to the products they use or the situations they experience. Many brands and companies even conduct polls on these sites and blogs to understand the general public sentiment and demand for their various offerings. This trend has created a requirement for technology that can identify and summarize overall public sentiment.

Literature Review:

Approaches to solve the problem: -

The problem can be approached using various methods depending on the complexity, efficiency, and features you want to implement. Here are a few different approaches:

We use four approaches for your project on sentiment analysis. This includes the Lexicon-Based Keyword Scoring, Rule-Based Sentiment Analysis with Phrases, Pattern Matching with Context-Based Scoring, and a Dynamic Programming (DP) approach.

1. Lexicon-Based Keyword Scoring

Methodology:

This approach relies on a predefined dictionary or lexicon that assigns sentiment scores (positive, negative, or neutral) to individual words. By summing up the sentiment scores of the words present in a text, the overall sentiment score is calculated.

Algorithm:

1. Preprocess the text (convert to lowercase, remove punctuation).

2. Tokenize the text into words.

3. For each word:

• If it exists in the sentiment lexicon, add its score to a cumulative score.

4. Based on the cumulative score, classify the text sentiment (e.g., positive if the score is positive, negative if it's negative, and neutral if zero).

Advantages:

• Simple and Fast: It is easy to implement and computationally efficient.

• Transparent: The process is interpretable, as it relies on directly counting words.

Disadvantages:

• Limited Vocabulary: Only words in the lexicon are identified; other words are ignored.

• Lacks Context: Cannot account for negations, intensifiers, or the effect of surrounding words.

Time Complexity: O(n), where n is the number of words in the text.

2. Rule-Based Sentiment Analysis with Phrases

Methodology:

This approach extends the lexicon-based method by including multi-word phrases, negations, and intensifiers. It uses predefined rules to handle phrases (e.g., "very happy") and adjust scores accordingly.

Algorithm:

1. Preprocess the text and tokenize it.

2. Identify common sentiment phrases (multi-word expressions) and define scores.

3. For each phrase and word in the text:

• Check if a phrase exists and add its score to the cumulative score.

• For single words, use lexicon-based scoring.

4. Modify scores based on modifiers like negations and intensifiers (e.g., “not good” inverts “good” to negative).

Advantages:

• Improved Accuracy: Handles sentiment phrases and modifiers for more precise scoring.

• Greater Flexibility: Recognizes commonly used expressions and can handle certain complex wordings.

Disadvantages:

• Rule Complexity: Requires extensive lists of phrases and complex rules for each possible modifier.

• Limited Scalability: Hard to cover every possible phrase or expression.

Time Complexity: O(n + k \cdot m), where n is the number of words, k is the number of phrases, and m is the average phrase length.

3. Pattern Matching with Context-Based Scoring

Methodology:

This approach combines lexicon-based scoring with pattern matching to handle context better. Words are assigned scores based on their presence in a sentiment lexicon, but modifiers like “not,” “very,” and “but” are also considered to adjust the score. This approach captures local context around sentiment-bearing words.

Algorithm:

1. Preprocess and tokenize the text.

2. Define a list of sentiment words and associated scores.

3. Define contextual modifiers (negation, intensifiers, adversative terms like “but”).

4. For each word in the text:

• If it’s a sentiment word, add its score to a cumulative score.

• If a modifier (e.g., “not”) appears before a sentiment word, adjust the score.

• If “but” appears, reset the score and focus on the sentiment after “but.”

5. Summarize the overall score and classify sentiment based on this score.

Advantages:

• Context Awareness: Handles contextual cues that modify sentiment, like negations and intensifiers.

• Better Accuracy: Captures sentiment shifts within a sentence.

Disadvantages:

• Complex Rule Management: Requires setup for each modifier and specific rules for context handling.

• Cannot Detect All Nuances: Struggles with sarcasm and extremely complex contexts.

Time Complexity: O(n), where n is the number of words.

4. Dynamic Programming (DP) Approach for Segment-Based Sentiment

Methodology:

This approach leverages dynamic programming to find the most optimal segmentation or sentiment across the text. The text is broken down into overlapping segments, and DP is used to compute the best cumulative sentiment by storing and reusing computed scores of each segment. This allows for efficiently capturing sentiment shifts.

Algorithm:

1. Preprocess and tokenize the text.

2. Divide the text into overlapping segments or windows (e.g., each with 3-5 words).

3. Initialize a DP table to store scores for each segment.

4. For each segment:

• Calculate a sentiment score using a lexicon-based approach.

• Store the score in the DP table, adjusting for overlaps.

5. Aggregate segment scores from the DP table to calculate the overall sentiment.

Advantages:

• Efficient for Long Texts: DP reduces redundant calculations by reusing segment scores.

• Localized Sentiment Detection: Captures shifts in sentiment across different segments.

Disadvantages:

• Increased Complexity: Managing a DP table and overlapping segments adds complexity.

• Higher Computational Cost: Requires additional memory and processing time.

Time Complexity: O(n * w), where n is the number of words, and w is the window (segment) size.

Summary Comparison Table

Approach	Advantages	Disadvantages	Time Complexity
Lexicon-Based Keyword Scoring	Simple, fast, easy to implement	Limited vocabulary, lacks context	O(n)
Rule-Based Sentiment with Phrases	Precise with negations and phrases	Complex rules, limited to predefined phrases	O(n + k \cdot m)
Pattern Matching with Context	Context-aware, nuanced scoring	Complex rule setup	O(n)
Dynamic Programming Segment-Based	Efficient for large texts, localized detection	Complex, higher memory cost	O(n * w)

Problem Statement: -

In today's digital world, social media and online platforms generate vast amounts of textual data, making it challenging to understand public sentiment and opinions effectively. For businesses, policymakers, and researchers, accurately analyzing sentiment from this data is essential to make informed decisions, respond to customer feedback, and understand societal trends. Traditional machine learning methods for sentiment analysis can be complex and computationally demanding, requiring large labeled datasets and extensive training. This project aims to design a sentiment analysis tool that uses lexicon-based approaches to identify and quantify sentiment in textual data without relying on machine learning or deep learning techniques.

The goal is to implement an efficient, rule-based sentiment analysis system that leverages predefined sentiment lexicons, context modifiers (like negations and intensifiers), and pattern-matching algorithms. By applying different lexicon-based and context-sensitive approaches, the project will capture sentiment from individual words and phrases while handling contextual variations such as "very good" or "not bad." This tool will provide a user-friendly and resource-efficient alternative for analyzing sentiment, ideal for small businesses, non-profits, or individuals who need quick insights from text data without advanced computational resources.

Proposed Methodology: C++ Pattern matching with context based scoring

- **Define Keywords and Context Scores**: Create a mapping of keywords to their respective scores and another mapping to define context-based adjustments.
- **Pattern Matching**: Search through the text for occurrences of these keywords.
- **Context Scoring**: Adjust the score based on the surrounding words.

Detailed Steps of Implementation:

Steps of Implementation

1. Define the Problem

- **Objective**: Build a sentiment analysis tool that scores text based on keywords and context.
- **Input**: Text data from social media posts.
- **Output**: A sentiment score that reflects the sentiment of the input text based on matched patterns and their context.

2. Set Up the Development Environment

- **Choose a Programming Language**: In this case, we will use C++.
- **Development Tools**: Use an IDE like Visual Studio, Code::Blocks, or any text editor with C++ compiler support.
- **Compiler**: Ensure you have a C++ compiler (like g++) installed on your system.

3. Design the Data Structures

- **Keyword and Sentiment Scores**: Use a hash map (unordered_map) to store keywords and their associated sentiment scores.
- **Context Modifiers**: Create another hash map for context modifiers that can adjust the sentiment score based on surrounding words.

4. Preprocess Input Text

- **Tokenization**: Write a function to break the input text into tokens (words).
- **Normalization**: Implement functionality to remove punctuation and convert all tokens to lowercase.

5. Implement the Pattern Matching Logic

- **Keyword Matching**: Write a function to iterate through the tokens and check if they match any keywords in the sentiment scores map.
- **Context Scoring**: For each matched keyword, check the surrounding tokens (previous and next) to see if they are context modifiers. Adjust the score accordingly.

6. Combine Functions into Main Logic

- Integrate all functions to read input, process it, and output the final sentiment score.

7. Testing and Validation

- Create test cases with known outputs to validate the functionality of the tool.
- Test with various input scenarios to ensure robustness.

Tokenization Process

1. **Splitting Text**: The primary step in tokenization involves splitting the text based on certain delimiters, such as spaces, punctuation marks, or special characters. For example, the sentence "I love programming!" would be tokenized into ["I", "love", "programming"].

2. **Normalization**: Tokens can be converted to a uniform format. Common normalization techniques include:
    a. **Lowercasing**: Converting all tokens to lowercase (e.g., "Programming" becomes "programming").
    b. **Removing Punctuation**: Stripping out punctuation marks (e.g., "hello!" becomes "hello").

3. **Handling Special Cases**: Depending on the application, additional rules might be applied to handle contractions (e.g., "don't" as "do" and "not"), stopwords (commonly used words like "and", "the", which may be removed), or numerical data.

C++ Implementation

```
#include <iostream>

#include <unordered_map>

#include <vector>

#include <sstream>

#include <algorithm>

#include <cctype>

using namespace std;

// Lexicon with words and their sentiment scores

unordered_map<string, int> lexicon = {

{"happy", 2}, {"excellent", 3}, {"sad", -2}, {"bad", -3},

{"great", 3}, {"terrible", -3}, {"good", 2}, {"poor", -2},
```

```
{"love", 3}, {"hate", -3}, {"enjoy", 2}, {"dislike", -2},
{"amazing", 3}, {"awful", -3}, {"fantastic", 3}, {"disappointing", -2},
{"awesome", 3}, {"worst", -3}, {"superb", 3}, {"boring", -2},
{"wonderful", 3}, {"dreadful", -3}, {"positive", 2}, {"negative", -2},
{"pleased", 2}, {"angry", -2}, {"satisfied", 2}, {"upset", -2},
{"joyful", 3}, {"miserable", -3}, {"delightful", 3}, {"annoyed", -2},
{"respect", 2}, {"disrespect", -2}, {"happy", 3}, {"unhappy", -2},
{"thrilled", 3}, {"depressed", -3}
};
// Function to check if a character is punctuation
bool isPunct(char c) {
return ispunct(static_cast<unsigned char>(c));
}
// Function to tokenize and preprocess text
vector<string> tokenize(const string& text) {
stringstream ss(text);
string word;
vector<string> words;
while (ss >> word) {
// Convert word to lowercase
transform(word.begin(), word.end(), word.begin(), ::tolower);
// Remove punctuation (simple approach)
word.erase(remove_if(word.begin(), word.end(), isPunct), word.end());
```

```
words.push_back(word);
}
return words;
}
// Function to calculate sentiment score based on lexicon and context
int calculateSentimentWithContext(const vector<string>& words) {
int score = 0;
bool negation = false;
for (size_t i = 0; i < words.size(); ++i) {
string word = words[i];

// Handle negation ("not" inverts the next sentiment word)
if (word == "not") {
negation = true;
continue;
}
// Handle intensifiers (e.g., "very" doubles the score of the next word)
bool intensify = (i > 0 && words[i - 1] == "very");
// Handle adversative conjunctions (e.g., "but" resets the score and increases weight of following words)
if (word == "but") {
score = 0; // Reset score before "but"
continue;
}
```

```
// Check if word is in lexicon
if (lexicon.find(word) != lexicon.end()) {
int wordScore = lexicon[word];

// Apply negation if present
if (negation) {
wordScore = -wordScore;
negation = false; // Reset negation after applying
}

// Apply intensification if present
if (intensify) {
wordScore *= 2;
}
// Add word's score to total score
score += wordScore;
}
}
return score;
}
// Function to classify sentiment based on score
string classifySentiment(int score) {
if (score > 0) return "Positive";
```

```
else if (score < 0) return "Negative";
else return "Neutral";
}
// Main function
int main() {
// Take input from the user
cout << "Enter a social media post: ";
string post;
getline(cin, post);
// Tokenize and preprocess the post
vector<string> words = tokenize(post);
// Calculate sentiment score with context-based rules
int score = calculateSentimentWithContext(words);
// Classify the final sentiment
string sentiment = classifySentiment(score);
// Display the results
cout << "Post: " << post << endl;
cout << "Sentiment Score: " << score << endl;
cout << "Sentiment Classification: " << sentiment << endl;
return 0;
}
```

OUTPUT:

```
Enter a social media post: I am very happy with the excellent service,
    but not satisfied with the delay.
Post: I am very happy with the excellent service, but not satisfied
    with the delay.
Sentiment Score: 3
Sentiment Classification: Positive
```

```
Enter a social media post: I love the food, but the service was
    terrible and slow.
Post: I love the food, but the service was terrible and slow.
Sentiment Score: -2
Sentiment Classification: Negative
```

```
Enter a social media post: The movie was okay, not amazing but not
    terrible either.
Post: The movie was okay, not amazing but not terrible either.
Sentiment Score: 0
Sentiment Classification: Neutral
```

Challenges faced and how we overcame them:

Challenge: Limited Vocabulary for Sentiment Analysis

- **Explanation**: At first, we had a limited set of words to identify positive and negative sentiments, which wasn't enough to capture the full meaning of different posts.
- **Solution**: We expanded our lexicon by adding more words with different sentiment strengths. This made the tool more accurate in identifying a wider range of emotions.

Challenge: Text Preprocessing and Tokenization

- **Explanation**: Real social media posts often have slang, misspellings, punctuation, and emojis, which made it hard to break down the text into understandable words.
- **Solution**: We built a preprocessing step to remove unnecessary punctuation and convert everything to lowercase. This made our tool more reliable in recognizing keywords despite variations.

Challenge: Testing the Tool with Different Inputs

- **Explanation**: We needed to ensure the tool worked accurately on a variety of sentences but had limited examples to test with.
- **Solution**: We created different test cases, including positive, negative, neutral and mixed-sentiment sentences, to improve and validate the tool's accuracy.

Result Analysis: -

Context-based keyword matching is a sophisticated approach in sentiment analysis that significantly enhances the accuracy and relevance of sentiment scoring. Here are the key advantages of this method:

Aspect	Keyword Scoring	Phrase Scoring	Context Scoring	DP Segment Scoring
Negation Handling	Low	Medium	High	Medium
Intensifiers (e.g., "very")	Low	High	High	Medium
Contrast Handling (e.g., "but")	Low	Medium	High	High

1. Improved Accuracy

• Context Sensitivity: By considering the context in which a word appears, this approach reduces the chances of misclassification. For example, in the phrase "not happy," the negation alters the sentiment, allowing for a more accurate score.

• Handling Nuanced Language: The use of context allows the analysis to recognize subtleties in language, such as sarcasm, humor, and irony, which simple keyword matching might overlook.

2. Incorporation of Linguistic Features

• Negation Handling: The ability to identify negation words (e.g., "not," "never") enables the system to reverse sentiment for adjacent keywords, thus reflecting the true sentiment expressed in the text.

• Intensity Modification: Recognizing intensifiers (e.g., "very," "extremely") allows the system to adjust scores accordingly, enhancing the granularity of sentiment analysis.

3. Dynamic Sentiment Scoring

• Real-Time Adjustments: Context-based keyword matching provides a dynamic method to adjust sentiment scores based on surrounding words and phrases, making the analysis more reflective of actual sentiment rather than relying solely on predefined scores.

• Handling Adversative Conjunctions: By resetting the score upon encountering conjunctions like “but,” the method allows for more nuanced interpretations of contrasting sentiments within the same statement.

4. Greater Applicability Across Domains

• Versatile Use Cases: This approach can be effectively applied across various domains such as customer feedback, social media monitoring, and product reviews, making it a versatile tool for businesses and researchers.

• Cultural and Contextual Sensitivity: By considering contextual factors, this method can be adapted for different cultural settings and linguistic nuances, improving its applicability in diverse environments.

5. Enhanced Interpretability

• Clearer Sentiment Attribution: The contextual adjustments provide clearer interpretations of sentiment scores, making it easier for users to understand the reasoning behind the assigned sentiment classification.

• User Trust and Engagement: When users see accurate sentiment analysis, it builds trust in the tool, encouraging more engagement and reliance on the system for decision-making.

Metric	Lexicon-Based Keyword Scoring	Rule-Based with Phrases	Pattern Matching with Context	DP-Based Segment Scoring
Accuracy	Low	Medium	High	High
Processing Speed	High	Medium	Medium	Low
Context Sensitivity	Low	Medium	High	Medium
Memory Usage	Low	Medium	Medium	High

Conclusion:

In this project, we successfully developed a sentiment analysis tool that analyzes social media posts to determine their emotional tone. Through a series of challenges, we learned valuable lessons about natural language processing and text analysis.

We expanded our vocabulary to improve the accuracy of our sentiment assessments and implemented context-based scoring to handle complex sentence structures. By preprocessing text effectively, we ensured that our tool could interpret various formats and styles commonly found in social media.

Our testing showed that the tool could accurately identify positive, negative, and neutral sentiments across different types of posts. This capability can be beneficial for businesses and researchers looking to understand public opinion and sentiment trends on social media platforms

Overall, this project not only enhanced our programming and analytical skills but also deepened our understanding of how sentiment analysis can be applied in real-world scenarios. We are excited about the potential improvements and future enhancements we can make, such as incorporating machine learning techniques for even better accuracy in sentiment detection.

References:

Sentiment Analysis: A complete guide

https://getthematic.com/sentiment-analysis

Speech and Language Processing" by Daniel Jurafsky and James H. Martin.