# Report: Sentiment Analysis in C

Hardik Gera, McMaster University April 9, 2024

## 1 Introduction

This is a C program for performing sentiment analysis on text data. The program reads a lexicon of words with associated sentiment scores from a file, and then uses this lexicon to calculate the sentiment score of sentences in another file.

The program defines several functions:

1) read dictionary: This function reads the lexicon file, which should contain one word per line, followed by its sentiment score and standard deviation. The function returns an array of WordScore structures, each representing a word and its associated score.

2) is intensifier: This function checks if a given word is an intensifier (e.g., "very", "really", "extremely"). Intensifiers are used to increase the sentiment score of the following word.

3) calculate sentence score: This function calculates the sentiment score of a given sentence. It tokenizes the sentence into words, and for each word, it checks if it is an intensifier or a negation word (e.g., "not"). If the word is in the lexicon, its score is added to the total score of the sentence, possibly modified by an intensifier or negation.

4) print sentiment analysis: This function reads sentences from a file and prints their sentiment scores.

The main function of the program takes two command-line arguments: the name of the lexicon file and the name of the file containing sentences to analyze. It reads the lexicon into memory, performs sentiment analysis on the sentences, and then frees the memory used by the lexicon.

## 2 Explanation of the code

```
1 Header Inclusions:
2 <stdio.h>: Provides standard input/output functions like fopen, fclose, ↩
    fprintf, printf, etc., for file and console operations.
3 <stdlib.h>: Provides general utility functions like malloc, realloc, free ↩
    for memory allocation and deallocation.
4 <string.h>: Provides string manipulation functions like strcmp, strtok, ↩
    strdup for comparing, tokenizing, and copying strings.
5 <ctype.h>: Provides character classification functions like tolower for ↩
    converting characters to lowercase.
6 sentiment_analysis.h : This contains function prototypes or definitions ↩
    related to sentiment analysis that are specific to this project.
```

## 2.1 Function Implementation

```
1  read_dictionary: Takes the filename of a sentiment lexicon file and a pointer to an integer (size) to store the number of words read. Opens the file for reading. If unsuccessful, prints an error message and returns NULL. Initializes lexicon to NULL (empty pointer) and size to 0. Uses a while loop to read each line of the file using fgets. Inside the loop: Reallocates memory for the lexicon array to accommodate a new WordScore struct for each line. Allocates memory for the word field of the new struct using malloc. Uses sscanf to parse the line into the word, score, and standard deviation fields of the struct. Increments size to keep track of the number of words read. Closes the file using fclose. Returns the lexicon pointer.
2
3  is_intensifier: Takes a constant character pointer (word) as input. Defines an array of intensifier words ("very", "really", "extremely"). Iterates through the intensifier array using a for loop. Inside the loop, compares the input word with each intensifier using strcmp. If a match is found, returns 1 (true). If no match is found after iterating through all intensifiers, returns 0 (false).
4
5  calculate_sentence_score function: This function takes three arguments:
6  lexicon: Pointer to an array of WordScore structs containing sentiment lexicon data (word, score, standard deviation).
7  lexicon_size: Integer representing the number of words in the lexicon.
8  sentence: Constant character pointer to the sentence for which sentiment is to be calculated.
9
10 The function calculates the sentiment score of the sentence by iterating through its words and considering the following factors:
11 1. Sentence Preprocessing: Creates a copy of the sentence string using strdup to avoid modifying the original input. Converts all characters in the sentence copy to lowercase using a loop with tolower. This ensures case-insensitive sentiment analysis (e.g., "Good" and "good" are considered the same).
12 2. Word Processing Loop: Uses strtok to tokenize the sentence copy into individual words based on delimiters (spaces, punctuation). Initializes variables: total_score: Float to accumulate the sentiment score of the sentence. word_count: Integer to count the number of words in the sentence. intensifier: Flag (0 or 1) to track the presence of an intensifier before a word.
13 3. Looping Through Words: Iterates through the words in the sentence using a while loop with strtok:
14    If the current word is an intensifier (calling is_intensifier), sets the intensifier flag to 1. This indicates that the sentiment of the following word (if found in the lexicon) should be intensified
```

.

Otherwise (for non-intensifier words):

Iterates through the lexicon array using a nested for loop to find a matching word.

If a match is found (using strcmp), performs the following:

Adds the sentiment score of the word from the lexicon to total_score. The score is multiplied by 2 if an intensifier preceded the word (accounting for intensification). This increases the impact of positive or negative sentiment when an intensifier is present.

Resets the intensifier flag to 0 after processing each word. This ensures intensification applies only to the next immediate word.

Increments word_count to keep track of the total number of words processed.

4. Negation Handling: Introduces negation handling:

Initializes a negation flag (0 or 1) to track negation. This flag is flipped whenever a negation word ("not" or "n't") is encountered.

Iterates through the remaining words in the sentence (after the strtok loop finishes) using another while loop:

If the current word is "not" or "n't" (negation indicators), flips the negation flag (becomes 1 if it was 0, and vice versa).

Otherwise (for non-negation words):

Similar to the previous loop for non-intensifier words, searches for the word in the lexicon.

However, before adding the score to total_score, considers the negation flag:

If negation is active (negation == 1), the sentiment score from the lexicon is negated (multiplied by -1) before adding to the total score. This flips the sentiment for words following a negation indicator.

5. Final Calculations and Return: After processing all words, the total_score reflects the sentiment of the sentence considering individual word sentiment, intensification, and negation. To normalize the score based on the sentence length, its divided by word_count. This provides an average sentiment score per word. The function returns the calculated sentiment score (average sentiment per word).

print_sentiment_analysis: Takes three arguments:

filename: Constant character pointer to the file containing sentences for sentiment analysis.

lexicon: Pointer to an array of WordScore structs containing sentiment lexicon data.

lexicon_size: Integer representing the number of words in the lexicon.

Opens the file for reading. If unsuccessful, prints an error message and returns. Declares a character array line to store each line read from

the file. Uses a while loop to read each line of the file using fgets. Inside the loop: Calls calculate_sentence_score to calculate the sentiment score for the current line. Prints the line and its calculated score using printf. Closes the file using fclose.

37

38 main: This is the programs entry point. Takes two command-line arguments (argc) and an array of character pointers (argv) to access them. Checks if there are exactly two command-line arguments (argc == 3). If not, prints an error message and usage instructions, then exits with an error code (1). Declares an integer lexicon_size to store the number of words in the lexicon. Calls read_dictionary to read the lexicon file specified in the first argument (argv[1]) and store the data in the lexicon pointer. It also sets lexicon_size. If read_dictionary returns NULL (indicating an error reading the lexicon), prints an error message and exits with an error code (1). ... Calls print_sentiment_analysis to analyze the sentences in the file specified by the second argument (argv[2]), using the loaded lexicon (lexicon) and its size (lexicon_size). Frees the memory allocated for individual words in the lexicon array using a loop with free. Frees the memory allocated for the lexicon array itself using free. Returns 0 to indicate successful program execution.

---

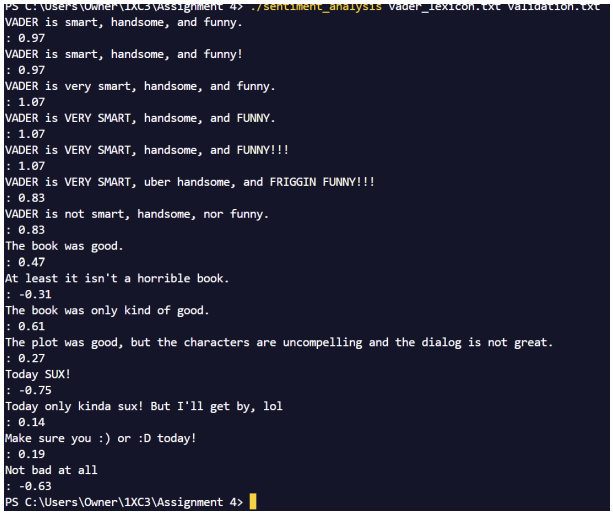


Figure 1: Sentiment Analysis in C

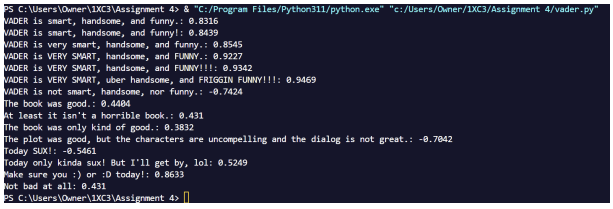


Figure 2: Example of Sentiment Analysis in Python

# 3 Code

The complete C code for the Sentiment Analysis is provided on Avenue to Learn. Here's a text form code of the file.

## 3.1 Main

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "sentiment_analysis.h"

// Function implementations
WordScore *read_dictionary(const char *filename, int *size)
{
    // Implementation...
    FILE *file = fopen(filename, "r");
    if (file == NULL)
    {
        perror("Failed to open the file");
        return NULL;
    }

    WordScore *lexicon = NULL;
    char line[256];
    *size = 0;

    while (fgets(line, sizeof(line), file))
    {
        lexicon = realloc(lexicon, (*size + 1) * sizeof(WordScore));
        lexicon[*size].word = malloc(256 * sizeof(char));

        sscanf(line, "%s %f %f", lexicon[*size].word, &lexicon[*size].score, &lexicon[*size].sd);
        (*size)++;
    }

    fclose(file);
    return lexicon;
}
int is_intensifier(const char *word)
{
    const char *intensifiers[] = {"very", "really", "extremely"};
    int num_intensifiers = sizeof(intensifiers) / sizeof(intensifiers[0]);

```

```
39      for (int i = 0; i < num_intensifiers; i++)
40      {
41          if (strcmp(word, intensifiers[i]) == 0)
42          {
43              return 1;
44          }
45      }
46
47      return 0;
48  }
49
50  float calculate_sentence_score(WordScore *lexicon, int lexicon_size, const↩
        char *sentence)
51  {
52      char *sentence_copy = strdup(sentence);
53      for (int i = 0; sentence_copy[i]; i++)
54      {
55          sentence_copy[i] = tolower(sentence_copy[i]);
56      }
57      char *word = strtok(sentence_copy, " .,;:!?\"\'\n");
58      float total_score = 0;
59      int word_count = 0;
60      int intensifier = 0;
61
62      while (word != NULL)
63      {
64          if (is_intensifier(word))
65          {
66              intensifier = 1;
67          }
68          else
69          {
70              for (int i = 0; i < lexicon_size; i++)
71              {
72                  if (strcmp(word, lexicon[i].word) == 0)
73                  {
74                      total_score += (intensifier ? 2 : 1) * lexicon[i].↩
                            score;
75                      break;
76                  }
77              }
78              intensifier = 0;
79          }
80          word = strtok(NULL, " .,;:!?\"\'\n");
81          word_count++;
82      }
83      int negation = 0; // Flag to track negation
```

```
84
85      while (word != NULL)
86      {
87          if (strcmp(word, "not") == 0 || strcmp(word, "n't") == 0)
88          {
89              negation = !negation; // Toggle negation flag
90          }
91          else
92          {
93              for (int i = 0; i < lexicon_size; i++)
94              {
95                  if (strcmp(word, lexicon[i].word) == 0)
96                  {
97                      total_score += (intensifier ? 2 : 1) * (negation ? -
                            lexicon[i].score : lexicon[i].score);
98                      break;
99                  }
100             }
101             negation = 0; // Reset negation after processing a word
102         }
103         word = strtok(NULL, " .,;:!?\"\'\n");
104         word_count++;
105     }
106
107     free(sentence_copy);
108     return total_score / word_count;
109 }
110
111 void print_sentiment_analysis(const char *filename, WordScore *lexicon,
        int lexicon_size)
112 {
113     // Implementation...
114     FILE *file = fopen(filename, "r");
115     if (file == NULL)
116     {
117         perror("Failed to open the file");
118         return;
119     }
120
121     char line[256];
122     while (fgets(line, sizeof(line), file))
123     {
124         float score = calculate_sentence_score(lexicon, lexicon_size, line
                );
125         printf("%s: %.2f\n", line, score);
126     }
127
```

```
128     fclose(file);
129 }
130 
131 int main(int argc, char *argv[])
132 {
133     if (argc != 3)
134     {
135         fprintf(stderr, "Usage: %s <lexicon_file> <validation_file>\n", ↩
               argv[0]);
136         return 1;
137     }
138 
139     int lexicon_size;
140     WordScore *lexicon = read_dictionary(argv[1], &lexicon_size);
141     if (lexicon == NULL)
142     {
143         fprintf(stderr, "Failed to read the lexicon\n");
144         return 1;
145     }
146 
147     print_sentiment_analysis(argv[2], lexicon, lexicon_size);
148 
149     for (int i = 0; i < lexicon_size; i++)
150     {
151         free(lexicon[i].word);
152     }
153     free(lexicon);
154 
155     return 0;
156 }
```

### 3.2 Sentiment analysis.h

```
1  #ifndef SENTIMENT_ANALYSIS_H
2  #define SENTIMENT_ANALYSIS_H
3  
4  typedef struct {
5      char *word;
6      float score;
7      float sd;
8      int intensity[10];
9  } WordScore;
10 
11 // Function prototypes
```

```
12 WordScore *read_dictionary(const char *filename, int *size);
13 float calculate_sentence_score(WordScore *lexicon, int lexicon_size, const↩
       char *sentence);
14 void print_sentiment_analysis(const char *filename, WordScore *lexicon, ↩
      int lexicon_size);
15
16 #endif // SENTIMENT_ANALYSIS_H
```

### 3.3 Makefile

```
1
2 CC=gcc
3 CFLAGS=-Wall -Wextra -Wpedantic
4 SRCS=sentiment_analysis.c
5 EXEC=mySA
6
7
8 $(EXEC): $(SRCS)
9     $(CC) $(CFLAGS) -o $(EXEC) $(SRCS)
10
11 clean:
12     rm -f $(EXEC)
```

## 4 Sources


1. https://github.com/Taylor-McNeil/Sentiment-Analysis-in-C
2. https://www.geeksforgeeks.org/what-is-sentiment-analysis/
3. https://chat.openai.com/(Used for taking reference of functions and report.)