Report

Chris Chen

March 7, 2024

1 Introduction

Sentiment analysis is a analysis technique that can determine the opinion of a sentence or a chunk of text. Sentiment analysis is the automatic identification and classification of sentiments expressed in text. Basic sentiment analysis can categorize words into positive, negative, and neutralIn coding, we are doing the rule based sentiment analysis. In computer languages, because of the limitations of logical computation, we usually assign a value to each word, with 0 representing a neutral word, a positive number representing a positive word, and a negative number representing a negative word. Programmers usually use VADER which stands for Valence Aware Dictionary and sEntiment Reasoner. in this dictionary, programmers can programmatically call the polarity score, standard deviation, and sentiment scores of each word. intensity scores for each word, which are then used to calculate the scores in a sentence for computerized sentiment analysis.

2 Appendix

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

//default lenth
#define MAX_LINE_LENGTH 1000

struct word {
    char *word;
    float score;
    float SD;
    int SIS_array[10];
};

// Function to parse a line and extract word, score, SD, and SIS_array
int parse_line(char *line, struct word *w) {
    char *token = strtok(line, "\t"); // Using ’\t’ as the delimiter
    if (token == NULL) return 0; // Empty line
    //set word to a new copy of token
    w->word = strdup(token);

    token = strtok(NULL, "\t");
    if (token == NULL) return 0; // Invalid format
    //From ASCII to Float
    w->score = atof(token);

    token = strtok(NULL, "\t");
    if (token == NULL) return 0; // Invalid format
    //From ASCII to Float
    w->SD = atof(token);

    token = strtok(NULL, "[]");
    if (token == NULL) return 0; // Invalid format
    char *sis_token = strtok(token, ", ");
    for (int i = 0; i < 10; i++) {
        if (sis_token == NULL) return 0; // Invalid format
        //Convert from ASCII to INT
        w->SIS_array[i] = atoi(sis_token);
        sis_token = strtok(NULL, ", ");
    }

    return 1;
}

int main(int argc, char *argv[]) {
    printf("      string sample                                                   score\n");
    printf("----------------------
    ----------------------------------------
    -----------------------------------------------------\n");
    // Check if the correct number of command-line arguments is provided
    if (argc != 3) {
        printf("Usage: %s <lexicon_file> <validation_file>\n", argv[0]);
        return 1;
    }

    // Open the lexicon file
    FILE *lexicon_file = fopen(argv[1], "r");
    //if it is no such file
    if (lexicon_file == NULL) {
        printf("Error: Unable to open lexicon file %s\n", argv[1]);
        return 1;
    }

    // Read and parse the lexicon file
    struct word *lexicon_words = NULL;
    int num_lexicon_words = 0;
    int lexicon_capacity = 0;
    char lexicon_line[MAX_LINE_LENGTH];
    while (fgets(lexicon_line, sizeof(lexicon_line), lexicon_file)) {
        struct word new_word;
        if (parse_line(lexicon_line, &new_word)) {
            // Check if array needs to be resized
            if (num_lexicon_words >= lexicon_capacity) {
                lexicon_capacity = (lexicon_capacity == 0) ? 1 : lexicon_capacity * 2;
                // memory reallocate
                lexicon_words = realloc(lexicon_words, lexicon_capacity * sizeof(struct word));
                if (lexicon_words == NULL) {
                    printf("Error: Memory allocation failed\n");
                    fclose(lexicon_file);
                    return 1;
                }
            }
            // Add new word to lexicon array
            lexicon_words[num_lexicon_words++] = new_word;
        } else {
            printf("Warning: Unable to parse line in lexicon file: %s", lexicon_line);
        }
    }
    //close the file
    fclose(lexicon_file);

    // Open the validation file
    FILE *validation_file = fopen(argv[2], "r");
    if (validation_file == NULL) {
        printf("Error: Unable to open validation file %s\n", argv[2]);
        return 1;
    }

    // Read and process the validation file
    char validation_line[MAX_LINE_LENGTH];
    while (fgets(validation_line, sizeof(validation_line), validation_file)) {
        // Process each sentence in the validation file
        validation_line[strcspn(validation_line, "\n")] = 0;
        //temp to store the line
        char *temp = strdup(validation_line);
        float sentence_score = 0.0;
        int word_count = 0;

        //Separate the line by space
        char *token = strtok(validation_line, " "); // Tokenize the line by space

        while (token != NULL) {
            word_count++;
            int len = strlen(token);
            // Check if the token exists in the lexicon
            int found_in_lexicon = 0;




            for (int i = 0; i < num_lexicon_words; i++) {
                if (strcmp(lexicon_words[i].word, token) == 0 || strcmp(":)", token) == 0) {
                    found_in_lexicon = 1;
                    break; // Exit the loop if the token is found in the lexicon
                }
            }


            // If the token is not found in the lexicon and contains symbols, truncate it
            if (found_in_lexicon == 0) {
                int len = strlen(token);
                for (int j = 0; j < len; j++) {
                    if (!isalpha(token[j])) {
                        token[j] = '\0'; // Replace punctuation with null character to truncate the word
                        break; // Stop processing the current token after truncating
                    }
                }
            }




            //default the words to lower case
            for (int i = 0; i < len; i++) {
                token[i] = tolower(token[i]);
            }


            // Look for the token in the lexicon and add its score to the sentence score
            for (int i = 0; i < num_lexicon_words; i++) {

                if (strcmp(lexicon_words[i].word, token) == 0) {
                    sentence_score += lexicon_words[i].score;
                    break; // Stop searching once the word is found
                }
            }
            token = strtok(NULL, " "); // Move to the next token

        }
        if (word_count > 0 ) {
            float avg_score = sentence_score / word_count;
            printf("%-105s%.2f\n", temp, avg_score); // Align the output using %-100s format specifier
        } else {
            printf("Not valid\n");
        }
        // Compute and print the average score for the sentence

    }

    // Free allocated memory for lexicon words
    for (int i = 0; i < num_lexicon_words; i++) {
        free(lexicon_words[i].word);
    }
    free(lexicon_words);

    fclose(validation_file);
    return 0;
}
```

3 Explaination

First we need to store the word score information and so on in the corresponding text file. I will store the score, standard deviation, etc. of the corresponding word in the word struct. I have used the strtok function to separate the information and store it separately. The program then opens the user-supplied file and performs a sentiment analysis on the sentences in it. the underlying logic of the analysis is still to use the strtok function on a line of sentences to split it into individual words and then retrieve them individually in the stored struct. The program adds up the scores and divides them by the total number of words to arrive at a score. Which is the following:

```
      string sample
                                                         score
--------------------------------------------------------------------

VADER is smart, handsome, and funny.
                                                        0.97
VADER is smart, handsome, and funny!
                                                        0.97
VADER is very smart, handsome, and funny.
                                                        0.83
VADER is VERY SMART, handsome, and FUNNY.
                                                        0.83
VADER is VERY SMART, handsome, and FUNNY!!!
                                                        0.83
VADER is VERY SMART, uber handsome, and FRIGGIN FUNNY!!!
                                                        0.64
VADER is not smart, handsome, nor funny.
                                                        0.83
The book was good.
                                                        0.47
At least it isn't a horrible book.
                                                        -0.36
The book was only kind of good.
                                                        0.61
The plot was good, but the characters are uncompelling and the
    dialog is not great.                                    0.27
Today SUX!
                                                        -0.75
Today only kinda sux! But I'll get by, lol
                                                        0.16
Make sure you :) or :D today!
                                                        0.80
```

```
Not bad at all
                                           -0.62
```

This is the out put of the program with the text file that we provided and the user may change the VADER package and input file, it will also work for same format.

4 Conclusion

In conclusion, the program works with most text files and most VADER packages of the same format to perform a simple sentiment analysis based on the package, which can be used to analyze textual opinions to a certain extent.