Sentiment Analysis

Tanishq Chahal, McMaster University 05/04/2024

1) Functions header

The header file defines functions and a structure for handling data related to sentiment analysis. It includes functionalities for creating an array of structures, reading data from a file, conducting sentiment analysis, and freeing memory..

Listing 1: functions.h

```
#ifndef FUNCTIONS_H
#define FUNCTIONS_H

// Structure to store the words, their scores, standard deviation and SIS array
struct words
{
    char *word;
    float score;
    float SD;
    int SIS_array[10];
};

// Function to make an array of struct
void make_aos(int *wordlist_size, struct words **wordlist);

// Function to read data from the file and store it in an array of struct
void get_data(char *file_name,int *wordlist_size, struct words **wordlist);

// Funstion to free memory allocated for the array of struct
void free_data(struct words *wordlist);

// Function to perform Sentiment Analysis using VADER
void sentiment_analysis(struct words *wordlist, char *validation_file);

#endif
```

2) functions.c

The functions.c file contains a collection of functions related to sentiment analysis. Firstly, it imports the necessary libraries, including the 'functions.h' header file which contains the function declarations.

Listing 2: function.c

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

3) make_aos() function

This function dynamically allocates memory for an array of structures based on the provided size (wordlist_size). It correctly assigns the allocated memory to the pointer *wordlist.

Listing 3: make_aos() function

```c
// Function to make an array of structures
void make_aos(int *wordlist_size, struct words **wordlist)
{
    // Allocate memory for the array of structures
    *wordlist = malloc(sizeof(struct words) * (*wordlist_size));

    // Check if memory is allocated
    if (*wordlist == NULL)
    {
        printf("Error: Unable to allocate memory\n");
        exit(1);
    }
}
```

4) get_data() function

The function reads data from the specified file and stores it in the array of structures pointed to by *wordlist. It handles dynamically resizing the array when it becomes full. The data is tokenized using strtok() and stored appropriately in the structure fields. It correctly deallocates the memory if an error occurs during memory reallocation.

Listing 4: get_data() function

```
// Function to read data from the file and store it in an array of structures
void get_data(char *file_name,int *wordlist_size, struct words **wordlist)
{
    // Open the file
    FILE *file = fopen(file_name, "r");

    // Check if the file is opened
    if (file == NULL)
    {
        printf("Error: Unable to open file %s\n", file_name);
        exit(1);
    }

    // array to store one line of the file
    char line[200];
    // counter to keep track of the number of words(lines)
    int c = 0;

    // Read the file line by line
    while (fgets(line, sizeof(line), file) != NULL)
    {
        // Check if the array is full
        if (c >= *wordlist_size)
        {
            // Increase the size of the array by 1000
            *wordlist_size += 1000;
            // Reallocate memory for the array
            *wordlist = realloc(*wordlist, sizeof(struct words) * (*wordlist_size));

            // Check if memory is allocated
            if (*wordlist == NULL)
            {
                printf("Error: Unable to allocate memory\n");
                exit(1);
            }

        }

        // Structure to store the data
        struct words new_word;

        // Tokenize the line
        char *token = strtok(line, " \t\n");

```

```
45          // Store the data in the structure
46          new_word.word = strdup(token);
47
48          token = strtok(NULL, " \t\n");
49          new_word.score = atof(token);
50
51          token = strtok(NULL, " \t\n");
52          new_word.SD = atof(token);
53
54          // Tokenize the SIS array
55          char *arr_token = strtok(NULL, ", []\t\n");
56          int i = 0;
57
58          // Store the SIS array in the structure
59          while (arr_token != NULL)
60          {
61              new_word.SIS_array[i] = atoi(arr_token);
62              arr_token = strtok(NULL, ", []\t\n");
63              i++;
64          }
65
66          // Store the structure in the array
67          (*wordlist)[c] = new_word;
68          c++;
69
70      }
71      // Close the file
72      fclose(file);
73  }
```

5) free_data() function

This function correctly frees the memory allocated for each word in the array of structures and then frees the memory for the array itself.

Listing 5: free_data() function

```
1  // Function to free memory allocated for the array of structures
2  void free_data(struct words *wordlist)
3  {
4      // Free the memory allocated for each word
5      for (int i = 0; wordlist[i].word != NULL; i++) {
6          free(wordlist[i].word);
7      }
8      // Free the memory allocated for the array
```

```
    free(wordlist);
}
```

6) sentiment_analysis() function

This function performs sentiment analysis on the text lines read from the validation file. It correctly opens the validation file and reads lines from it. Sentiment analysis is done by comparing each word in the line with the words stored in the array of structures (wordlist). It accumulates the total score for each line based on the words' scores.

Listing 6: sentiment_analysis() function

```
// Function to perform Sentiment Analysis using VADER
void sentiment_analysis(struct words *wordlist, char *validation_file)
{
    // Open the validation file
    FILE *val_file = fopen(validation_file, "r");

    // Check if the file is opened
    if (val_file == NULL)
    {
        printf("Error: Unable to open file %s\n", validation_file);
        exit(1);
    }

    // Read the file line by line
    char line[200];

    // Perform the sentiment analysis
    while (fgets(line, sizeof(line), val_file) != NULL)
    {
        // Variables to store the total score and the number of words in ↩
           each line
        float total_score = 0;
        int c = 0;

        // Copy the line to another variable
        char line_copy[200];
        strcpy(line_copy, line);

        // Remove the newline character put by fgets
        line_copy[strlen(line_copy) - 1] = '\0';

        // Convert the line to lowercase
        for (int i = 0; line[i] != '\0'; i++)
```

```
33          {
34              line[i] = tolower(line[i]);
35          }
36
37          // Tokenize the line
38          char *token = strtok(line, " \n\t!,.");
39
40          // Calculate the total score
41          while (token != NULL)
42          {
43              // iterate through the wordlist
44              for (int i = 0; wordlist[i].word != NULL; i++)
45              {
46                  // Compare the token with the word in the wordlist
47                  if (strcmp(token, wordlist[i].word) == 0)
48                  {
49                      // Add the score of the word to the total score
50                      total_score += wordlist[i].score;
51                  }
52              }
53              c++;
54              // Get the next word
55              token = strtok(NULL, " \n\t!,.");
56          }
57
58          // Check if the number of words is greater than 0
59          if (c > 0)
60          {
61              // Calculate the average score
62              total_score = total_score / c;
63              // Print the line and the total score (%-100s is used to print↩
                    the string in 100 characters width to the left and %10.2f↩
                    is used to print the float with 2 decimal places)
64              printf("%-100s %10.2f", line_copy, total_score);
65              printf("\n");
66          }
67
68      }
69
70      // Close the file
71      fclose(val_file);
72  }
```

7) main() function

The main() function serves as the entry point of the program. It orchestrates the flow of the program, from argument handling to memory management and execution of sentiment analysis, ensuring proper functionality and resource utilization.

Listing 7: main() function

```
1  int main(int argc, char *argv[])
2  {
3      // Check if the number of arguments is correct
4      if (argc != 3)
5      {
6          // Print the usage of the program
7          printf("Usage: %s <SA_dictionary> <Validation_file>\n", argv[0]);
8          return 1;
9      }
10
11     // Get the file name and the validation file
12     char *file_name = argv[1];
13     char *validation_file = argv[2];
14
15     // initialize the wordlist
16     int wordslist_size = 5000;
17     struct words *wordlist = NULL;
18
19     // Make the array of structures
20     make_aos(&wordslist_size, &wordlist);
21
22     // Get the data from the file and store it in the wordlist
23     get_data(file_name, &wordslist_size, &wordlist);
24
25     // Print the header
26     printf("              String sample ↩
                                                                                ↩
        Score\n");
27     printf("↩
        ------------------------------------------------------------------------
        n");
28
29     // Perform the sentiment analysis
30     sentiment_analysis(wordlist, validation_file);
31
32     // Free the data
33     free_data(wordlist);
34
35     return 0;
```

```
}
```

8) Makefile

Listing 8: Makefile

```
CC = gcc
CFLAGS = -Wall -Wextra
EXECUTABLE = mySA
SRC = sa.c
OBJSRC = functions.c
OBJ = functions.o
HLP = functions.h

$(EXECUTABLE): $(SRC) $(OBJ)
    $(CC) $(CFLAGS) -o $(EXECUTABLE) $(SRC) $(OBJ)
$(OBJ): $(OBJSRC) $(HLP)
    $(CC) -c $(OBJSRC) $(CFLAGS) -o $(OBJ)
clean:
    rm -f mySA functions.o
```

9) Usage

- Compile the program using 'make' command.
- Run the program using the following command.

Listing 9: Run

```
    ./SA <dictionary\_name> <validation\_file>
```

- The dictionary file should be of the following format

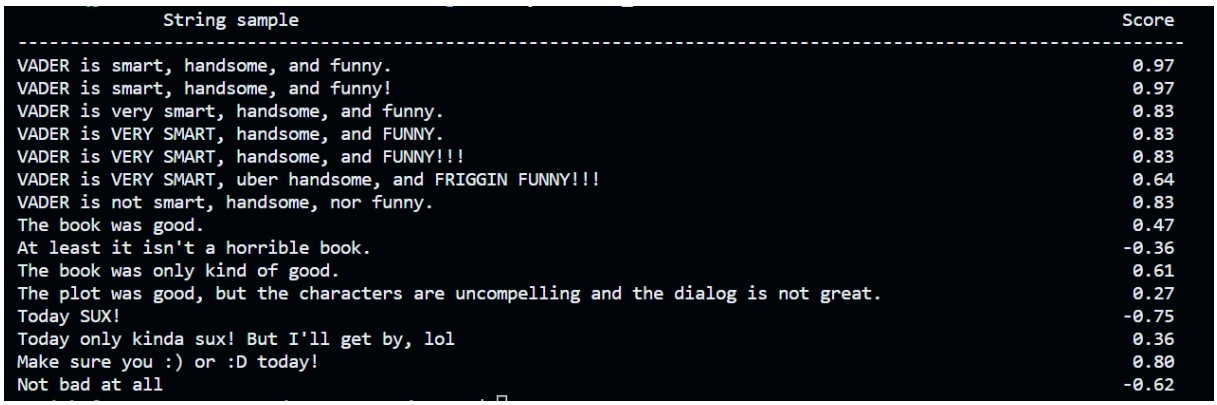
Listing 10: Dictionary format

```
    <word> <score> <standard_deviation> [<SIS_array>]
```

- Output:

Listing 11: Output Example

```
    ./mySA vader_lexicon.txt validation.txt
```



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	0.36
Make sure you :) or :D today!	0.80
Not bad at all	-0.62

Figure 1: Enter Caption

9) Appendix

Listing 12: Appendix

```
// Import the necessary libraries
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "functions.h"


int main(int argc, char *argv[])
{
    // Check if the number of arguments is correct
    if (argc != 3)
    {
        // Print the usage of the program
        printf("Usage: %s <SA_dictionary> <Validation_file>\n", argv[0]);
        return 1;
    }

    // Get the file name and the validation file
    char *file_name = argv[1];
    char *validation_file = argv[2];

    // initialize the wordlist
    int wordslist_size = 5000;
    struct words *wordlist = NULL;

    // Make the array of structures
    make_aos(&wordslist_size, &wordlist);

    // Get the data from the file and store it in the wordlist
```

```
31      get_data(file_name, &wordslist_size, &wordlist);
32
33      // Print the header
34      printf("              String sample ↩
                                                                                ↩
           Score\n");
35      printf("↩
           ------------------------------------------------------------------------------
           n");
36
37      // Perform the sentiment analysis
38      sentiment_analysis(wordlist, validation_file);
39
40      // Free the data
41      free_data(wordlist);
42
43      return 0;
44  }
45
46  #include <stdio.h>
47  #include <stdlib.h>
48  #include <string.h>
49  #include <ctype.h>
50  #include "functions.h"
51
52  // Function to make an array of structures
53  void make_aos(int *wordlist_size, struct words **wordlist)
54  {
55      // Allocate memory for the array of structures
56      *wordlist = malloc(sizeof(struct words) * (*wordlist_size));
57
58      // Check if memory is allocated
59      if (*wordlist == NULL)
60      {
61          printf("Error: Unable to allocate memory\n");
62          exit(1);
63      }
64  }
65
66  // Function to read data from the file and store it in an array of ↩
       structures
67  void get_data(char *file_name,int *wordlist_size, struct words **wordlist)
68  {
69      // Open the file
70      FILE *file = fopen(file_name, "r");
71
72      // Check if the file is opened
```

```
73      if (file == NULL)
74      {
75          printf("Error: Unable to open file %s\n", file_name);
76          exit(1);
77      }
78
79      // array to store one line of the file
80      char line[200];
81      // counter to keep track of the number of words(lines)
82      int c = 0;
83
84      // Read the file line by line
85      while (fgets(line, sizeof(line), file) != NULL)
86      {
87          // Check if the array is full
88          if (c >= *wordlist_size)
89          {
90              // Increase the size of the array by 1000
91              *wordlist_size += 1000;
92              // Reallocate memory for the array
93              *wordlist = realloc(*wordlist, sizeof(struct words) * (*↩
                    wordlist_size));
94
95              // Check if memory is allocated
96              if (*wordlist == NULL)
97              {
98                  printf("Error: Unable to allocate memory\n");
99                  exit(1);
100             }
101
102         }
103
104         // Structure to store the data
105         struct words new_word;
106
107         // Tokenize the line
108         char *token = strtok(line, " \t\n");
109
110         // Store the data in the structure
111         new_word.word = strdup(token);
112
113         token = strtok(NULL, " \t\n");
114         new_word.score = atof(token);
115
116         token = strtok(NULL, " \t\n");
117         new_word.SD = atof(token);
118
```

```
        // Tokenize the SIS array
        char *arr_token = strtok(NULL, ", []\t\n");
        int i = 0;

        // Store the SIS array in the structure
        while (arr_token != NULL)
        {
            new_word.SIS_array[i] = atoi(arr_token);
            arr_token = strtok(NULL, ", []\t\n");
            i++;
        }

        // Store the structure in the array
        (*wordlist)[c] = new_word;
        c++;

    }
    // Close the file
    fclose(file);
}

// Function to free memory allocated for the array of structures
void free_data(struct words *wordlist)
{
    // Free the memory allocated for each word
    for (int i = 0; wordlist[i].word != NULL; i++) {
        free(wordlist[i].word);
    }
    // Free the memory allocated for the array
    free(wordlist);
}

// Function to perform Sentiment Analysis using VADER
void sentiment_analysis(struct words *wordlist, char *validation_file)
{
    // Open the validation file
    FILE *val_file = fopen(validation_file, "r");

    // Check if the file is opened
    if (val_file == NULL)
    {
        printf("Error: Unable to open file %s\n", validation_file);
        exit(1);
    }

    // Read the file line by line
    char line[200];
```

```
166
167     // Perform the sentiment analysis
168     while (fgets(line, sizeof(line), val_file) != NULL)
169     {
170         // Variables to store the total score and the number of words in ↩
                each line
171         float total_score = 0;
172         int c = 0;
173 
174         // Copy the line to another variable
175         char line_copy[200];
176         strcpy(line_copy, line);
177 
178         // Remove the newline character put by fgets
179         line_copy[strlen(line_copy) - 1] = '\0';
180 
181         // Convert the line to lowercase
182         for (int i = 0; line[i] != '\0'; i++)
183         {
184             line[i] = tolower(line[i]);
185         }
186 
187         // Tokenize the line
188         char *token = strtok(line, " \n\t!,.");
189 
190         // Calculate the total score
191         while (token != NULL)
192         {
193             // iterate through the wordlist
194             for (int i = 0; wordlist[i].word != NULL; i++)
195             {
196                 // Compare the token with the word in the wordlist
197                 if (strcmp(token, wordlist[i].word) == 0)
198                 {
199                     // Add the score of the word to the total score
200                     total_score += wordlist[i].score;
201                 }
202             }
203             c++;
204             // Get the next word
205             token = strtok(NULL, " \n\t!,.");
206         }
207 
208         // Check if the number of words is greater than 0
209         if (c > 0)
210         {
211             // Calculate the average score
```

```
            total_score = total_score / c;
            // Print the line and the total score (%-100s is used to print the string in 100 characters width to the left and %10.2f is used to print the float with 2 decimal places)
            printf("%-100s %10.2f", line_copy, total_score);
            printf("\n");
        }

    }

    // Close the file
    fclose(val_file);
}
```

10) References

- Large-language models