Assignment 4

Rule-Based Sentiment Analysis

Pedram Pasandide

Due Date: April 10th

1 Introduction

Take a look at the What is Sentiment Analysis?. Sentiment analysis is a technique used to analyze text and determine the sentiment or opinion expressed within it. It's incredibly useful across diverse fields such as marketing, customer service, and finance. By understanding the sentiment behind text data, businesses can make informed decisions, predict trends, and improve their products or services to better meet customer needs.

Rule-based sentiment analysis is one approach to analyzing sentiment that relies on predefined rules and patterns to determine the sentiment of a piece of text. These rules are typically based on linguistic patterns, keywords, or grammatical structures associated with specific sentiments. While rule-based sentiment analysis can be straightforward and interpretable, it may lack the flexibility and nuance of more advanced machine learning approaches. In this assignment, we will develop a simple **Rule-based** Sentiment Analysis.

2 Read the Dictionary (10 points)

VADER (Valence Aware Dictionary and sEntiment Reasoner) is a lexicon and rule-based sentiment analysis tool specifically designed for analyzing the sentiment of text. It is extensively used in natural language processing tasks such as social media sentiment analysis, customer feedback analysis, and opinion mining. The VADER lexicon consists of a list of words with corresponding sentiment scores, indicating the positivity, negativity, and neutrality of each word. We downloaded the VADER lexicon (`vader_lexicon.txt`) from the official GitHub repository from here .

To write a C code to read the downloaded dictionary `vader_lexicon.txt`, we'll need to parse the file and store the words along with their sentiment scores in a data structure for further sentiment analysis computations. For example, the line 4450 in `vader_lexicon.txt`, we have "love 3.2 0.4 [3, 3, 3, 3, 3, 3, 3, 4, 4, 3]", where "love" is the **word**, "3.2" is the polarity **score**, "0.4" is the **standard deviation**, and "[3, 3, 3, 3, 3, 3, 3, 4, 4, 3]" represents the **sentiment intensity scores**. Each word with its corresponding score **must** be saved in a `struct`:

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

Read the files with dynamic memory allocation (`malloc` and `realloc`). Your code must be able to handle all potential errors. Let's say if I add a few more words to the files, your program must be able still read the updated dictionaries. I have to be able to compile your code with `Makefile`, and run it with `./mySA vader_lexicon.txt <other_inputs>`, where `mySA` is the executable object file created by your `Makefile`. At this point we don't have any `<other_inputs>`.

3 Sentiment Analysis (7 points)

This is a rule-based sentiment analysis, and we want to keep it simple. The best way I can explain what we have to compute is giving you an example. Let's say I found a comment from a social media platform saying:

"I loved COMPSCI1XC3 course. The lectuer notes were top notch :D"

All the words have negative score except for "top-notch" and "loved". "lecture" is a neutral vocabulary in our dictionary. However, "lectuer" is a typo and if not listed in the positive and negative dictionary we should give it a neutral score (= 0). The string ":D" is an emoji icon. At the end, I have scores for each word in the sentence:

words	I	loved	COMPSCI1XC3	course	The	lectuer	notes	are	top	notch	:D	**sum**
scores	0	2.9	0	0	0	0	0	0	0.8	0	2.3	+6

We have 11 words and the summation (NOT absolute) of +6. The **average** is equal to $+6/11 = +0.54$. What we want is to compute the **average score** for each given sentence. Take a look at another example:

words	good	assignments	but	the	lectures	were	terribly	boring	**sum**
scores	1.9	0	0	0	0	0	-2.6	-1.3	-2

The average score here is $-2/8 = -0.25$. Technically, we use only `char *word` and `float score` from the data. In file `validation.txt`, we have 15 sample of comments. At this point I have to be able to run you code using `./mySA vader_lexicon.txt validation.txt`. Then, your program must print the final score for each sentence in the terminal like:

```
            string sample                                                             score
-----------------------------------------------------------------------------------------------------
VADER is smart, handsome, and funny.                                                   <s1>
VADER is smart, handsome, and funny!                                                   <s2>
VADER is very smart, handsome, and funny.                                              <s3>
VADER is VERY SMART, handsome, and FUNNY.                                              <s4>
VADER is VERY SMART, handsome, and FUNNY!!!                                            <s5>
VADER is VERY SMART, uber handsome, and FRIGGIN FUNNY!!!                               <s6>
VADER is not smart, handsome, nor funny.                                               <s7>
The book was good.                                                                     <s8>
At least it isn't a horrible book.                                                     <s9>
The book was only kind of good.                                                        <s10>
The plot was good, but the characters are uncompelling and the dialog is not great.    <s11>
Today SUX!                                                                             <s12>
Today only kinda sux! But I'll get by, lol                                             <s13>
Make sure you :) or :D today!                                                          <s14>
Not bad at all                                                                         <s15>
```

Instead of `<s1>`, `<s2>`, ..., `<s15>`, print the scores computed. You can compare your results with the results given by VADER in Python (it is optional), and mention your conclusion in your report. If you make it work similar the VADER package on Python, it can be considered a good project to be added into your resume.

Beside how I should be able to compile, run, and what results must be printed out in the terminal, I don't want to restrict you on the structure of this assignment. To get the full mark for this assignment, you must implement whatever you have learned in this course. **Only some** of common mistakes I find in projects are:

- Not enough comments
- Memory leak, stack overflow
- `Makefile` doesn't work
- Not using memory allocation where it is needed and **potential** stack overflow
- The codes for reading the files cannot handle potential changes within the files
- The structure of codes are not properly designed (`.h` and `.c` files)
- Wrong results
- Warning or Errors during compiling the code

4 Report and Makefile (3 points)

Your report must be in a LaTeX format (`report.tex`).Tell us about your results. For instance, if you're thinking about sharing your project on GitHub, make sure it's easy for users to understand how your program works when they follow your report.

Include all your codes inside the report. You can create a section called appendix, then copy and past all your codes in it. Produce the PDF file (`report.pdf`) to make sure it is working, and submit both of them. Create a `Makefile` and make sure it will work in any OS.

5 Submission On Avenue to Learn

You can use any resources to write the code. Don't forget to mention the source, and please follow the submission guidelines. No zip files on Avenue. Please avoid copying from each other. If the copied percentage exceeds the class average, you may be required to present your code.

- `Makefile`, and all the `.c` and `.h` files needed to compile your code
- `report.tex` **AND** `report.pdf`

Future directions! If you are interested to work in this field in the future, you can read this section. Here I have provided some useful information to start with. Nothing from this section is necessary in this assignment. It is optional, optional to study!
Sentiment analysis is super helpful across various fields, like predicting stock market trends and enhancing customer services! By using sentiment analysis, companies can pinpoint exactly what products to create, instead of just taking wild guesses. But that's not that much simple. Take the sentence "Axe the tax" as an example. Just think about the words individually. Do you think "Axe" is a positive word or a negative? Do you think "tax" implies a "positive" or a "negative" feeling? Do you think in total it has a positive score or negative? Keep your answers in your mind. According to the:

1. File `AFINN-96.txt` on GitHub rep the word "axe" is negative as well as what we have in `vader_lexicon.txt`. Also, the word "taxing" on here is negative. In general, if we say "the" in neutral, a model trained on these data would give us a negative score.

2. ChatGPT says: Link.

But what truly is the answer? Bing Liu explains why the appearance of a specific word in a sentence does not necessarily mean that it implies a positive or negative opinion.
The answer might depend on what the majority of people think about it at **this** moment. And finding that might be another complex problem. Or maybe it needs a giant social media platform having many users, like Twitter (new X). To download the data from X API you can use this code to extract data. Make sure you know about the X policy. You can also go to the Kaggle link to download some already gathered data around a specific topic. For example, tweets only about Bitcoin. Please be careful using that. This field is pretty advanced and needs a lot of study and validations to make sure it works. It needs knowledge from diverse fields. Also people's opinion on social media might affect the real world in just few seconds, and daily data cannot necessarily be everything you need to make a prediction. Second, simple models might not be accurate due to the changes in language itself and how people speak. For instance, how people talk on social media these days might be different from 15 years ago. In this case, we need an "updatable" dictionary.
Also, these days, we have many bots, spams, fake accounts on social media, misleading information, and making it impossible to get something out pure data downloaded from social media platforms.
These days we have a lot more complex NLP models, some listed here, to deal with some of mentioned issues.