Name: Gian Carlo Marotta
Class: CS 5393 002 (MWF 3:00PM)
Collaborators: -

Github Link: https://github.com/gcmarottaSMU/Marotta_CS5393-002_Project3

References:

https://stackoverflow.com/ (Stack Overflow)

https://www.w3schools.com/cpp/ (W3 Schools)

https://en.cppreference.com/w/ (C++ Reference)

https://www.learncpp.com/ (Learn C++)

https://old.reddit.com/r/cpp/ (cpp Reddit)

Project 3: Don't Be Sentimental!

Introduction

This project focuses on building a sentiment analysis classifier using C++. The goal is to classify tweets as either positive or negative based on their content. Sentiment analysis is a common task in machine learning, particularly in the field of natural language processing. The project involves three phases: training, prediction, and evaluation. During training, the classifier learns word associations based on labeled sentiment data. In the prediction phase, it classifies new, unseen tweets using the patterns identified during training. Finally, it evaluates its performance by comparing predictions to ground truth sentiment values, calculating the model's accuracy. Key learning objectives include implementing dynamic memory management via a custom string class, using efficient data structures, and understanding runtime complexity.

Objectives and Design Documentation

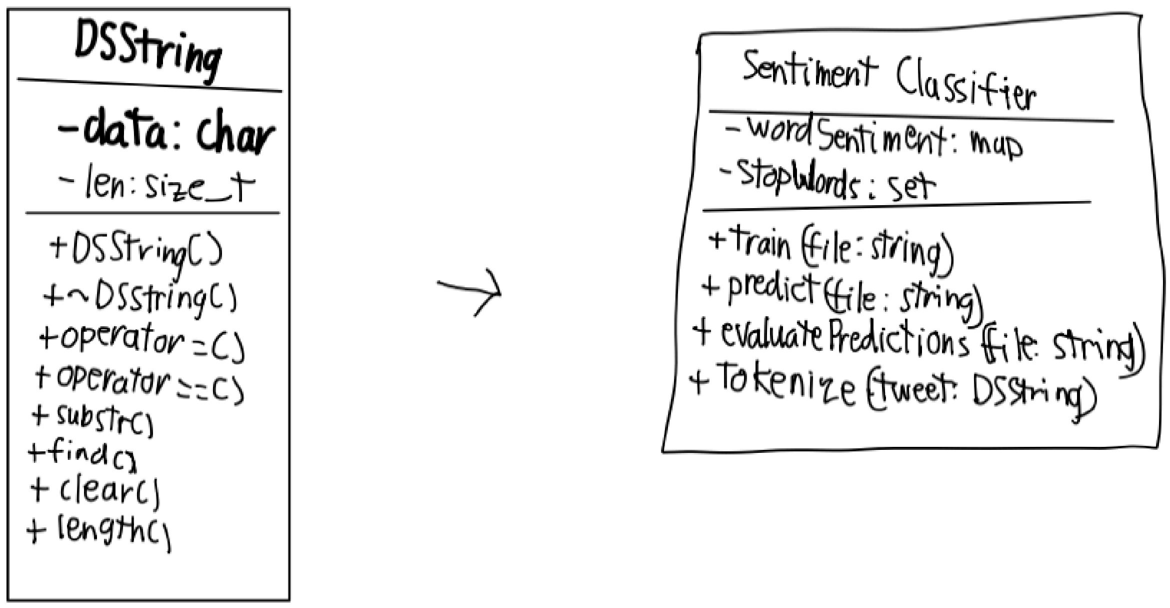
The primary objective of this project is to implement a robust sentiment analysis system that processes and classifies large datasets efficiently. This involves creating a `DSString` class to manage dynamic memory and handle strings without relying on the STL `std::string`. Additionally, the `SentimentClassifier` class performs the core tasks of the project: training on labeled data, predicting sentiments for unseen data, and evaluating predictions against ground truth data. Emphasis is placed on modularity, readability, and maintainability of code, with each function handling a distinct task.

System Architecture

The project consists of two main classes:

1. DSString Class: A custom implementation of a string class that includes dynamic memory allocation, rule-of-three compliance, and various utility functions like `substr`, `find`, and overloaded operators. This class is critical for handling textual data efficiently in the absence of `std::string`.
2. SentimentClassifier Class: Responsible for the following core functionalities:
    - Training: Reads labeled data, tokenizes tweets, and builds a word-sentiment map.
    - Prediction: Classifies unseen tweets based on the trained model.
    - Evaluation: Compares predictions with ground truth, calculates accuracy, and logs misclassified tweets.

UML Diagram



Command-Line Terminal

The program accepts five command-line arguments:

1. Training File: Contains labeled tweets (positive or negative) used for training.
2. Testing File: Contains unlabeled tweets whose sentiments need to be predicted.
3. Ground Truth File: Provides actual sentiment values for the test tweets.
4. Results File: Outputs the predicted sentiments in the format `Sentiment, TweetID`.
5. Accuracy File: Logs the classifier's accuracy and any misclassified tweets.

The program was ran on macOS using Terminal.
It uses 2 commands:

- g++ -std=c++11 -o sentiment sentiment.cpp
- ./sentiment data/train_dataset_20k.csv data/test_dataset_10k.csv data/test_dataset_sentiment_10k.csv results.csv accuracy.txt

Training Algorithm

The training algorithm processes the labeled dataset line by line. Each tweet is tokenized into individual words, and each word is associated with a sentiment score in a `std::unordered_map`. Words appearing in positive tweets increment their scores, while those in negative tweets decrement them. This builds a vocabulary with words categorized by their sentiment contributions, enabling sentiment prediction.

Classification Logic

The classifier tokenizes each tweet in the test set and sums the sentiment scores of its words. If the total score is non-negative, the tweet is classified as positive; otherwise, it is classified as negative. This logic leverages the word-sentiment map built during training to infer the sentiment of unseen data.

Tokenizer Implementation

The tokenizer splits tweets into words using `std::stringstream` and applies the following preprocessing:

1. Case Normalization: Converts all characters to lowercase.
2. Punctuation Removal: Strips out punctuation to focus on pure words.
3. Stop Words Filtering: Removes common words (e.g., "the", "and") that do not contribute to sentiment analysis.
4. Stemming: Reduces words to their root forms, e.g., "running" becomes "run".

This ensures that only meaningful words contribute to sentiment classification.

Memory Management

The `DSString` class implements the rule of three (constructor, copy constructor, and destructor) to manage dynamic memory efficiently. This ensures that memory is correctly allocated, copied, and deallocated, preventing memory leaks. The program also avoids unnecessary copies of large datasets by using efficient data structures such as `std::unordered_map` and `std::vector`.

Runtime Complexity

- Training Phase:
  Complexity is `O(N * M)`, where `N` is the number of tweets and `M` is the average number of words per tweet. This is due to the need to tokenize each tweet and update the word-sentiment map.
- Prediction Phase:
  Complexity is `O(K * M)`, where `K` is the number of test tweets. Each tweet is tokenized, and sentiment scores are computed from the word-sentiment map.
- Evaluation Phase:
  Complexity is `O(K)`, where `K` is the number of test tweets. Predictions are compared with ground truth values.

These complexities ensure scalability with reasonably large datasets.

Testing and Debugging

The system was tested with datasets of varying sizes to ensure correctness and performance. Edge cases included empty tweets, tweets with only stop words, and highly imbalanced datasets. The accuracy evaluation was verified by comparing manually classified test data with program output. Debugging focused on memory leaks, incorrect tokenization, and misclassifications.

Known Issues and Limitations

- Simple Stemming: The current implementation uses a basic stemmer that may not handle complex word forms accurately.
- Stop Words: The stop word list is minimal and may miss some common neutral terms.
- Limited Features: The classifier only uses word frequency and does not account for more sophisticated features like n-grams or contextual sentiment.

Potential Enhancements

- Advanced Stemming: Integrating a more sophisticated stemming library to handle complex word forms.
- Stop Words Expansion: Using a comprehensive stop word list tailored to sentiment analysis.
- Naive Bayes Model: Implementing a probabilistic classifier like Naive Bayes to improve accuracy.
- Multilingual Support: Extending the classifier to handle tweets in multiple languages.

Conclusion

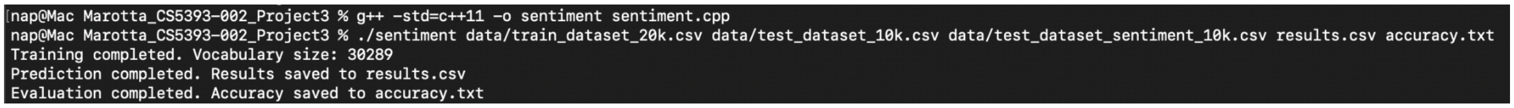
This project successfully implemented a sentiment analysis system in C++. Key learnings included designing custom data structures, implementing efficient algorithms, and managing dynamic memory. The classifier achieved reasonable accuracy on test datasets, demonstrating the effectiveness of the word-sentiment mapping approach.

Output

The program outputs two files after execution:

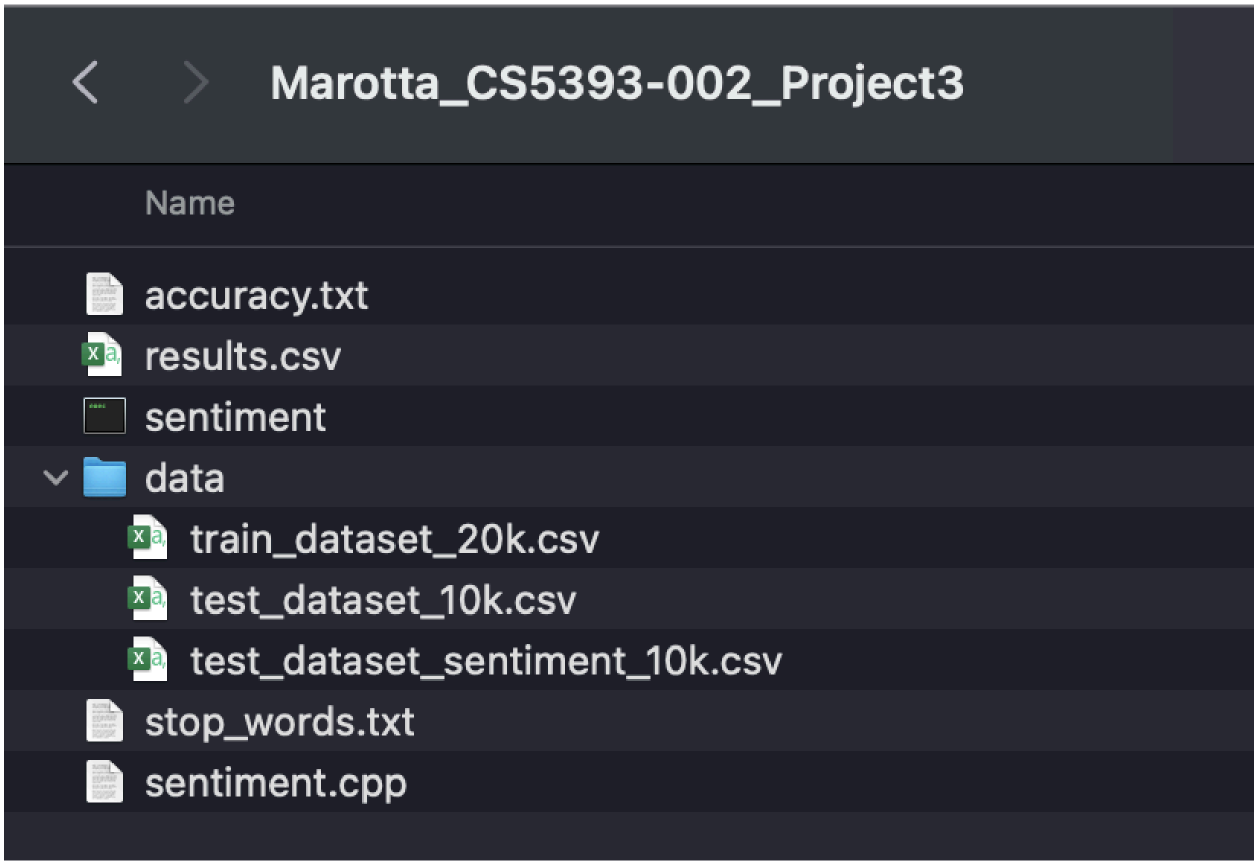
1. Results File: Contains predicted sentiments for each test tweet.
2. Accuracy File: Logs the classifier’s accuracy and lists misclassified tweets.

Input / Output



```
nap@Mac Marotta_CS5393-002_Project3 % g++ -std=c++11 -o sentiment sentiment.cpp
nap@Mac Marotta_CS5393-002_Project3 % ./sentiment data/train_dataset_20k.csv data/test_dataset_10k.csv data/test_dataset_sentiment_10k.csv results.csv accuracy.txt
Training completed. Vocabulary size: 30289
Prediction completed. Results saved to results.csv
Evaluation completed. Accuracy saved to accuracy.txt
```

Folder After Input / Output



Marotta_CS5393-002_Project3

Name

```
accuracy.txt
results.csv
sentiment
data
    train_dataset_20k.csv
    test_dataset_10k.csv
    test_dataset_sentiment_10k.csv
stop_words.txt
sentiment.cpp
```

Name: Gian Carlo Marotta
Class: CS 5393 002 (MWF 3:00PM)
Collaborators: -

Github Link: https://github.com/gcmarottaSMU/Marotta_CS5393-002_Project3

References:

https://stackoverflow.com/ (Stack Overflow)

https://www.w3schools.com/cpp/ (W3 Schools)

https://en.cppreference.com/w/ (C++ Reference)

https://www.learncpp.com/ (Learn C++)

https://old.reddit.com/r/cpp/ (cpp Reddit)