Project#2 for CS5393-002_DSC++

The following are questions for Project 2. Your response expectations are outlined per question. Your research is open book and collaborative.

Collaboration is encouraged, BUT responses and submissions are individual. Ensure the use of your own comments for your responses. Submission file should be named <Lastname_CS5393-002_Project2>.
Please ensure a paragraph outlines the objectives of the code (Design Documentation). 1-2-3 line responses, and shortcuts are penalized with a zero. Your name, Assignment type and number, the class information may be comments before your Design Documentation. Collaborators, references may be comments after your Design Documentation.
NOTE: Code READABILTY, meaningful OBJECT NAMES, good DESIGN DOCUMENTATION and prolific use of COMMENTS will have a POSITIVE BIAS on your grade. A UML (Unified Modeling Language) diagram is mandatory.

Sentiment Analysis

Dataset: CS2341-Assignment-2/data at main · lucastli/CS2341-Assignment-2 · GitHub

```
# Assignment: Don't Be Sentimental!
**Objective:**
* Practice the use of git and GitHub.
* Implement a class (DSString) that uses dynamic memory allocation and
requires the implementation of the rule-of-three.
* Practice debugging and memory leak detection.
* Use of STL container (vectors, maps, etc.)
* Analyze the runtime complexity.
* Write a program with a command line interface.
* Design, implement, and test a small system.
** Demonstration
**The lab TA will expect that you already have most of that week's work
completed. Come with prepared questions and be prepared to answer the TA's
questions about your code. You may leave the lab early if you can show the
TA that you have finished the work stated below.**
** Requirements of documentation are identical to Project 2
## Introduction
Have you ever read a tweet and thought, “Gee, what a positive outlook!” or
“Wow, why so negative, friend?”  Can computers make the same
determination?  They can surely try!
In Machine Learning, the task of assigning a **label** to a data item is
called **classification** (putting things into different classes or
categories).  The more specific name for what we’re going to do is
sentiment analysis because you’re trying to determine the “sentiment” or
attitude based on the words in a tweet.  The goal of the project is to
build a sentiment classifier! Aren’t you excited??  ( ← That would be
positive sentiment!)
```

You'll be given a set of tweets called the **training data set** that are already pre-classified by humans as **positive** or **negative** based on their content. You'll analyze the frequency of occurrences of words in the tweets to develop a classification scheme. Using your classification scheme, you'll then classify another set of tweets called the **testing data set** and **predict** if each tweet has positive sentiment or negative sentiment.

Building a Classifier

The goal in classification is to assign a class label to each element of a data set, positive or negative in our case. Of course, we would want this done with the highest accuracy possible. At a high level, the process to build a classifier (and many other machine learning models) has two major steps:

1. Training Phase
   - Input is a set of tweets with each tweet's associated sentiment value. This is the training data set.
   - Example: Assume you have 10 tweets and each is pre-classified with positive or negative sentiment. How might you go about analyzing the words in these 10 tweets to find words more commonly associated with negative sentiment and words more commonly associated with positive sentiment?
   - The result of the training step can be a list of words that have an associated positive or negative sentiment with them depending on which type of tweet they appeared in more frequently. Or, you might have 2 lists of words: one list is positive, one list is negative.
2. Testing Phase
   - In the testing phase, for a set of tweets called the testing data set, you predict the sentiment by using the list or lists of words extracted during the training phase.
   - Behind the scenes, you already know the sentiment of each of the tweets in the testing data set. We'll call this the actual sentiment or ground truth. You then compare your predicted sentiment to the known sentiment for each of the testing tweets. Some of the predictions will be correct; some will be wrong. The percentage correct is the accuracy, but more on this later.

The Data

The data set we will be using in this project comes from real tweets posted around 11-12 years ago. The original data was retrieved from Kaggle at https://www.kaggle.com/kazanova/sentiment140. It has been pre-processed it into the file format we are using for this project. For more information, please see Go, A., Bhayani, R. and Huang, L., 2009. Twitter sentiment classification using distant supervision. CS224N Project Report, Stanford, 1(2009), p.12.

The preprocessed dataset can is already included in the data folder in the project repo. Note that cmake will copy the data `folder` to the `build` folder.

Input files

There will be 3 different input files:

* Training Data
* Testing Data Tweet (no sentiment column)
* Testing Ground Truth (id and sentiment for testing data for you to compare against).

The training data file is formatted as a comma-separated-values (CSV) file containing a list of tweets, each one on a separate line. Each line of the data files include the following fields:
* Sentiment value (negative = 0, positive = 4, numbers in between are not used),
* the tweet id,
* the date the tweet was posted
* Query status (you can ignore this column)
* the twitter username that posted the tweet
* the text of the tweet itself
The testing data set is broken into two files:
* A CSV file containing formatted just like the training data EXCEPT no Sentiment column
* A CSV file containing tweet ID and sentiment for the testing dataset (so you can compare your predictions of sentiment to the actually sentiment ground truth)
Below are two example tweets from the training dataset:
```
4,1467811594,Mon Apr 06 22:20:03 PDT 2009,NO_QUERY,peruna_pony,"Beat TCU"
0,1467811595,Mon Apr 06 22:22:03 PDT 2009,NO_QUERY,the_frog,"Beat SMU"
```
Here is a tweet from the testing dataset:
```
1467811596,Mon Apr 06 22:20:03 PDT 2009,NO_QUERY,peruna_pony,"SMU > TCU"
```
The sentiment file for that testing tweet would be:
```
4, 1467811596
```
Examples of how to do reading and writing can be found [here.](https://github.com/mhahsler/CS2341/tree/main/Chapter1_Programming/io)
Convert strings right after reading to `DSStrings` (see [cstring/STL string examples](https://github.com/mhahsler/CS2341/blob/main/Chapter1_Programming/vector_string/main_cstring_vs_std_string.cpp)).
Running your Program: Training and Testing
Your program should take 5 command line arguments in this order:
1. training data set filename - the file with the training tweets
2. testing data set filename - tweets that your program will classify
3. testing data set sentiment filename - the file with the classifications for the testing tweet data
4. classifier results file name - see Output Files section below
5. classifier accuracy and errors file name - see Output Files section below
Example: In the build folder executing this command in the terminal should work.
```
./sentiment data/train_dataset_20k.csv data/test_dataset_10k.csv data/test_dataset_sentiment_10k.csv results.csv accuracy.txt
```

Here is a code example of how to use the [command line interface in C++.](https://github.com/mhahsler/CS2341/tree/main/Chapter1_Programming/cli)
Note that `CMake` installs a copy of the data directory in the `build` directory and that is why you can directly
use `data` in the path for the files above.
Output Files
Your program should produce two output files. The first file should contain your classifier's
results for all tweets in the testing dataset in the same format as the sentiment file:
```
4, 1467811596
...
```
The second file should be organized as follows:
* The first line of the output file will contain the accuracy, a single floating point number with exactly 3 decimal places of precision. See the section "How good is your classifier" below to understand Accuracy.
* The remaining lines of the file will contain for each tweet that your algorithm incorrectly classified your prediction,
the ground truth and the Tweet ID. This could be useful information as you tweak your algorithm to increase efficiency.
Example of the testing data tweet classifications file:
```
0.500
4, 1, 2323232323
1, 3, 1132553423
...
```
Implementation Requirements
Important: Read this before you start!!
Using GitHub
To use GitHub appropriately, you need to submit regularly new additions to your code and push them to the repository. This is important so you always have a safe backup and we can follow your development process. **Do not just upload your completed project to GitHub at the end or perform a single commit.**
Program Structure
Create a UML diagram. Remember that when you read the problem description then nouns are class candidates and verbs are candidates for member functions. You probably need a SentimentClassifier class. Clearly separate training from testing in your code. You should implement your classifier model in a class and your main function could look like this pseudo code:
```cpp
int main(some args)
{
   SentimentClassifier classifier;

   classifier.train(some args);
   classifier.predict(some args);
   classifier.evaluatePredictions(some args);
```

```
   return 0;
}
```
DSString
You need strings to represent tweets and individual words.
For this project, you must implement your own custom string class (`DSString`). **You may not use the STL string class or any other available string class from the Internet for this exercise. These can only be used to interface to other code (e.g., to read from a file).**
There is a header file called [`DSString.h`](DSString.h) provided in the repo. **You must implement all functions
specified in the header file and you can add more if you like.**
Note that your string implementation contains a c-string with a `\0` terminator and a `len` attribute for getting the string length fast.
You are not allowed to use c-string functions (in headers `<string.h>` or `<cstring>`) in your string class since they use C memory management with lots of pointers.
You need to implement all string operations yourself by manipulating the character array.
Your program should exclusively use your DSString class for strings.
You may use a `char[]` array/buffer or a `std::string` only temporarily when you are reading from or writing to a file or to interface with other libraries (e.g., a stemmer).
There is a simple test file for DSString available in the repo. Your DSString implementation should be able to work with it.
Note: the implementation of DSString is missing in the repo, so the code does not compile!
Tokenizer For Breaking Tweets into Words
You may use the existing c-string tokenizer (`strtok()`) or STL solutions (`std::stringstream`) in this project.
 ==> Document with 1/2 a page how the C-string tokenizer STL works.
Other Data Structures
You may use any STL data structure to store your DSStrings.
An option is a `std::vector` to store words and counters, but you can also use more efficient data structures as long as you can describe what they do and why you use them. We will study these data structures throughout this class.
You can use STL algorithms like `std::sort` and [`std::find`](https://en.cppreference.com/w/cpp/algorithm/find) or you can implement your own functions.
Make sure you have the necessary operators with the appropriate `const` versions overloaded in your class representing words and counters so STL algorithms work.
Classifier
There are implementations of advanced text classifiers available in C++. You could create a list of words that are more frequent in positive tweets and words that are more frequent in negative tweets. A tweet can then be classified using the majority of its words.
==> Write comments in your code that explain how this function works.
You may be inclined to skip the learning and find lists of positive words and/or negative words on the internet and simply count the occurrences of

those words. However, training the classifier is an important part and so this is not allowed in this assignment.
Run Time
Your code might take a while to run on the complete training/testing set. It is always a good idea to start with a smaller dataset (maybe 100 tweets) and use the whole dataset only once you are happy with your code. You will be later asked about the time complexity of your implementation. An important question is:
What does the time complexity depend on? The number of tweets? The number of words? The number of entries in one of your data structures?
Coding Style and Use of GitHub
We will be looking for simple, elegant, well-designed solutions to this problem.
* Organize your code in functions and classes with clear purposes
  to make it easier to read and debug. Remember:
    - UML class diagrams are an important design tool!
    - Each function should do **only** one thing!
* Minimal amount of code in the driver method (main, in the case of C++).
* Proper use of command-line arguments.
* File input and output: It is so important to be able to read from and write to data files.
* Use GitHub to commit each feature separately.
* The right amount of comments in just the right places.
* Proper memory management, including following the rule of 3 (big or 5).
* Your code should compile without warnings!
Note: You will probably find yourself rewriting parts of your code several times to make it
more flexible. This is called **refactoring** and it is a good thing!
How Good is Your Classifier?
Training and testing of a classification algorithm is an iterative process. You'll develop a training algorithm, test it, evaluate its performance, tweak the algo, retrain, retest, etc. How do you know how good your classifier is after each development iteration, though? We will use accuracy as the metric for evaluation.
$$\text{Accuracy} = \frac{(\text{total number of correctly classified tweets from test dataset})}{(\text{total number of tweets in the test data set})} $$
Why should you be interested in this? The TAs will take your final solutions and classify a set of tweets that your algorithm has never seen. They will calculate the accuracies (as defined above) for each submission.
==> * If your classifier produces an accuracy close to 50% then it is is not performing better than random guessing. This means that your code is most likely broken (e.g., it is messing up with indices or counting) and you will lose points. Your classifier needs to reach at least 60% to demonstrate that it is working.
==> * If your submission achieves 72.000% accuracy or higher, you will receive 5 bonus points on the project.
There are several ways to improve accuracy and run time:
* Stop words: You may look into using [stop words](https://en.wikipedia.org/wiki/Stop_word).
* Stemming:

Happy, happier, and happiest all come from the same root word of happy. Finding a common word stem is called [stemming.](https://en.wikipedia.org/wiki/Stemming). You might explore the use of a stemming library to help get to the root word.

Words that appear frequently in both positive and negative tweets are useless for determining sentiment. You can look for a C++ stemming library. If you have to convert `DSString` into `std::string` to use the library then you can do that.

* Character Encoding:

Some tweets might have interesting characters in them that aren't part of the ASCII character set. You might look into [character encoding](https://www.joelonsoftware.com/2003/10/08/the-absolute-minimum-every-software-developer-absolutely-positively-must-know-about-unicode-and-character-sets-no-excuses/) and the `wchar_t` data type (instead of `char`) if this becomes a problem.

* Converting Counts into Probabilities:

[Naive Bayes classifiers](https://en.wikipedia.org/wiki/Naive_Bayes_classifier) are a family of simple "probabilistic classifiers" based on applying Bayes' theorem. You can try to implement this strategy.

Grading Rubric

	Points Possible	Points Awarded
Proper use of GitHub (multiple commits)	5	
DSString Class	20	
UML Class Diagram	5	
Training Algo (+ counters)	25	
Classification Algo (Acc > 60%)	15	
Proper Memory Management	10	
Formatting, Comments, Warnings, etc.	10	
Answers to questions	10	
Accuracy/Improvements Bonus	+5	
48hr early submission Bonus	+5	