Don't be Sentimental!

Due: Monday Sept 13, 2021 @ 6 a.m. - pushed to Github in the GH Classroom Assignment repo.

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. Project 1 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 will be one 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 **known sentiment.** You then compare the **predicted sentiment** from the testing phase 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 Real 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. I've 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 pre-protps://www.dropbox.com/s/wqsv7bbjeva2bu3/10k_20k%20tweetcessed dataset for this project can be downloaded from ht%20testing-selected.zip?dl=0

Note that the data files are already in the template repo in the **data**/ folder.

Input files

There will be 3 different input files:

1. Training Data
2. Testing Data (no sentiment column)
3. Testing (id and sentiment for testing data for you to compare against).

The **training data** file is formatted as follows:

- 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**),
    - the tweet id,
    - the date the tweet was posted
    - Query status (you will 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 are two tweets 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
```

Output Files

There will be one output file 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 the Tweet IDs of the tweets from the testing data set that your algorithm **incorrectly** classified. This could be useful information as you tweak your algorithm to increase efficiency

Example of the Testing Data tweet classifications file (These tweet IDs are fake):

```
0.500
2323232323
```

```
1132553423
```

Running your Program

Your program will have two modes

- Running Catch TDD tests
    - this will be indicated by passing no command-line arguments to main (this is explained more below)
- Training & Testing the Classifier
    - this mode will have 4 command line arguments:
        - training data set filename - the file with the training tweets
        - testing data set filename - tweets that your program will classify
        - testing data set sentiment filename - the file with the classes for the testing tweet data
        - output file name - see **Output Files** section above
        - Example:
          `./classifier.out <train data> <test data> <test sentiment> <output>`

Implementation Requirements

- You must implement your own custom string class (DSString) as part of this project. **You may not use the STL string class or any other available string class from the Internet**. Additionally, you may NOT use c-strings (null-terminated character arrays) except to implement your custom string class and as a temporary buffer when reading from a file. There is a DSString.h file in the template repo that you may use as a guidepost for your string class.
    - The only exception here is when you're reading from or writing to a file. You may temporarily convert your DSString to a c-string for this.
- You will also need to implement a suite of tests for the string class using the **CATCH2** library. CATCH will be covered more in lab. You'll provide a testing mode (the mode with no command line args) which will execute your tests suite.
- While there are more advanced algorithms for text classification such as Naive Bayes, you may not use them in this project.
- You may be inclined to find lists of positive words and/or negative words on the internet and simply count the occurrences of those words. However, this is not allowed for this project.

Implementation Options

Below are some things you can explore to increase the accuracy or efficiency of your implementation.

- *Happy*, *happier*, and *happiest* all come from the same root word of *happy*. 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.
- Some tweets might have interesting characters in them that aren't necessarily part of the ascii character set. You might look into the wchar data type if this becomes a problem.

Comments about this Project

This is your opportunity to make a great first impression on the Prof/TAs in CSE 2341. We will be looking for simple, elegant, well-designed solutions to this problem. Please do not try to "wow" us with your knowledge of random, esoteric programming practices. Here is a list of things that you might want to consider while tackling this project:

- Procedural vs Object Oriented Design

    - A seemingly infinite amount of software has been designed, implemented, deployed, maintained, updated, and redeployed using both of these paradigms. One could argue for days, or week even, about which is the "better" paradigm for modern software development. Regardless of which paradigm you choose to use, the most important thing is that you produce an elegant solution following solid software development practices.
- File input and output
    - It is so important to be able to read from and write to data files. Think about some software program that doesn't use files...
- Just the right amount of comments in just the right places
- Minimal amount of code in the driver method (main, in the case of C++)
    - The code in main should be minimal and only used to "get the ball rolling."
- Proper use of command-line arguments
- Proper memory management

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. **The 5 submissions with the highest accuracies will get 10 bonus points!** Yes, this can even be combined with early submission EC.

Grading Rubric

	Points Possible	**Points Awarded**
DSString Class	20	
CATCH Tests	5	
Training Algo Implementation	25	
Classification Algo Implementation	25	
Source Code Quality (formatting, comments, etc.	15	
Proper use of Git and Github	10	