Lab 8: Sentiment analysis

COMP130
Instructor: John MacCormick

# Automated Sentiment Analysis

*Sentiment analysis* is the process of reading some text and determining the author’s feelings (positive or negative) toward the subject on which they are writing. For example, consider the following two film reviews:

*It almost feels as if the movie is more interested in entertaining itself than in amusing us.*

*A positively thrilling combination of ethnography and all the intrigue, betrayal, deceit and murder of a Shakespearean tragedy or a juicy soap opera*

The first review expresses a negative sentiment, whereas the second one expresses a positive sentiment. The task for automated sentiment analysisis to write a computer program that analyzes a piece of text and determines if it is expressing a positive or negative sentiment.

One way to do this is to use some existing labeled data to learn associations between words and sentiments. Internet review sites can be a good source of labeled data for this. Consider the following example reviews:

* "5 stars: Loved it!"
* "0 stars: Absolutely pathetic!"

The basic idea is that these reviews can be used to associate a positive sentiment ("5 stars") with the words "Loved" and "it" and a negative sentiment ("0 stars") with the words "Absolutely" and "pathetic".

# Our Rotten Tomatoes dataset

The dataset for this lab is based on movie reviews and is from real movie reviews posted at [Rotten Tomatoes](https://www.rottentomatoes.com/).

The dataset consists of three files:

* training.txt: We will use this as *training data* to assign sentiments to words in our sentiment analysis program. It contains over 8000 reviews.
* toy.txt: This contains only five reviews, selected from the training data. Data scientists often use a *toy* data set like this for debugging and testing.
* validation.txt: This contains about 100 reviews that are not in the training data. The validation data can be used for checking the accuracy of our programs, because it has not already been used for training. In this lab, the validation data is used only in the optional extension, not in the required portion of the lab.

These files are available to be downloaded in a zip file provided as resources for this lab. Extract these files and place them in your lab8 folder.

All three files store reviews in the same format. Open the training.txt file to observe the formatting. (You can use any *text editor* app for viewing the file. On Windows, Notepad is a suitable program for this. On a Mac, TextEdit is suitable. Most likely, you can simply double-click on the file and it will open in a suitable editor.)

The second line of the training.txt file provides a useful example:

4 This quiet, introspective and entertaining independent is worth seeking.

The first character of line is an integer between zero and four inclusive, indicating the rating of the review. The value 4 is the most positive rating and 0 is the most negative rating. After the integer rating, the line contains a space character followed by the text of the review itself.

# Reading the Dataset

Create a new Python script called lab08.py, save it in your lab8 folder and add code to it as you work through the lab. You should also create a responses document for written answers, as usual.

**Qu 1.** Write a function print\_first\_line(). This function should open the training.txt file, read the first line, print the first line, then close the file.

Here is some code that counts the lines in the training.txt file.

file = open('training.txt')

count = 0

line = file.readline()

while line != '':

 count = count + 1

 line = file.readline()

file.close()

print('training.txt has ' + str(count) + ' lines.')

**Qu 2.** Encapsulate and generalize this code by writing a function with the signature count\_lines(filename). This function should print the number of lines in the file named by the string parameter filename.

**Qu 3.** (Answer in responses document.) Use the function you wrote in the previous question to determine the number of lines in training.txt, toy.txt, and validation.txt. Record the results as your answer to this question.

**Qu 4.** Write a function with the signature get\_rating(line). This fruitful function should accept a single string parameter line, representing one line of text from the dataset. It returns an integer, which is the numerical rating given at the start of the line. For example, when line is the second line of training.txt as shown above, the function returns the integer 4. Hint: use string indexing.

The following test function can be used to test the correctness of your function.

def test\_get\_rating():

 assert get\_rating('4 I laughed, I cried, it was better than cats.') == 4

 assert get\_rating('0 Two thumbs down.') == 0

 print('get\_rating passed tests')

**Qu 5.** Write a function with the signature get\_review(line). This fruitful function should accept a single string parameter line, representing one line of text from the dataset. It returns a string, which is text of the review, excluding the initial numerical rating and space character. For example, when line is

4 This quiet, introspective and entertaining independent is worth seeking.

the function returns 'This quiet, introspective and entertaining independent is worth seeking.'. Hint: use string slicing.

**Qu 6.** Use assertions to write a test function for the function in the previous question. Use a format and style similar to the test\_get\_rating function provided above.

# Cleaning the reviews

Data scientists often *clean* their data before analyzing it. We clean data by removing unnecessary or irrelevant features, so that the analysis can be performed more easily and accurately. To clean the reviews in our dataset, we would like to remove punctuation characters, remove space characters and other unwanted characters at the beginning or end, and convert all alphabetic characters to lowercase. Below is a function for removing punctuation from a given string. It uses techniques that we will cover later in the semester, so there is no need to understand the details.

def remove\_punctuation(text):

 """Return a copy of the given text with all characters other than alphabetic and

 space characters removed."""

 valid\_chars = []

 for c in text:

 if c.isalpha() or c == ' ':

 valid\_chars.append(c)

 new\_text = ''.join(valid\_chars)

 return new\_text

Paste the above function into your Python script, and use it as part of your solution to the next question.

Cleaning text often involves the removal of *whitespace*. Whitespace consists of space characters, newline characters, and a number of other characters that produce no visible output. The strip() method of the string class removes whitespace characters from the beginning and end of a string.

**Qu 7.** Write a function with the signature clean\_review(review). This fruitful function should accept a single string parameter review, representing the text of a review from the dataset. Example: review could be 'This quiet, introspective and entertaining independent is worth seeking.'. The function returns a copy of the review text that has been converted to lowercase, with leading and trailing whitespace removed, and punctuation eliminated. Hint: use the function provided above and methods from the str class. The test function below will help check for correctness.

def test\_clean\_review():

 assert clean\_review('I lauGHed I CrieD') == 'i laughed i cried', 'review text is not all lowercase'

 assert clean\_review(' Two ThumBS DowN ') == 'two thumbs down', 'spaces at start and end not removed'

 assert clean\_review(' This "1" wasn`t BAD!!!! ') == 'this wasnt bad', 'nonalpha chars not removed'

 print('clean\_review passed tests')

# Estimating the sentiment of a word

When writing the next few functions, we employ the practice of *incremental development*. The ultimate goal is to write a function that will compute the average rating associated with any given word. But we will first tackle simpler versions of that task.

**Qu 8.** Write a function with the signature average\_rating\_v1(filename). This fruitful function should accept a single string parameter filename, which is the name of the filename storing one of our datasets (such as training.txt or toy.txt). It returns a float which is the average value of the ratings in the dataset. Test your function using the following test function:

def test\_average\_rating\_v1():

 eps = 0.001

 assert abs(average\_rating\_v1('toy.txt') - 2.8) < eps, 'failed on toy.txt'

 assert abs(average\_rating\_v1('training.txt') - 2.0626) < eps, 'failed on training.txt'

 print('test\_average\_rating\_v1 passed tests')

**Qu 9.** Write a function with the signature count\_occurrences(filename, word). The filename parameter is the same as in the previous question. The word parameter is a string. This fruitful function returns an integer which is the number of reviews in the dataset containing the given word. For example, in the toy.txt dataset, the number of occurrences of 'and' is 3. (Note: the word 'and' is actually used four times in this file, but it occurs twice in one of the reviews. Therefore, there are only three reviews containing the word 'and'.) Test your function using the following test function:

def test\_count\_occurrences():

 assert count\_occurrences('toy.txt', 'wonderful') == 3

 assert count\_occurrences('toy.txt', 'to') == 2

 assert count\_occurrences('toy.txt', 'asdfghjk') == 0

 assert count\_occurrences('toy.txt', 'imax') == 1

 assert count\_occurrences('training.txt', 'awesome') == 2

 print('count\_occurrences passed tests')

**Qu 10.** Write a function with the signature average\_rating\_v2(filename). The filename parameter is the same as in the previous questions. This fruitful function returns a float which is the average value of the ratings in the dataset, **considering only reviews that contain the word 'wonderful'**. Test your function using the following test function:

def test\_average\_rating\_v2():

 eps = 0.001

 assert abs(average\_rating\_v2('toy.txt') - 3.6666) < eps, 'failed on toy.txt'

 assert abs(average\_rating\_v2('training.txt') - 3.4166) < eps, 'failed on training.txt'

 print('test\_average\_rating\_v2 passed tests')

**Qu 11.** Now we are ready for the final goal of our incremental development. Using the code, knowledge, and experience from the earlier questions, write a function with the signature average\_rating(filename, word). The filename parameter is the same as in the previous questions. This fruitful function returns a float which is the average value of the ratings in the dataset, **considering only reviews that contain the given** word. Test your function using the following test function:

def test\_average\_rating():

 eps = 0.001

 assert abs(average\_rating('toy.txt', 'wonderful') - 3.6666) < eps

 assert abs(average\_rating('training.txt', 'wonderful') - 3.4166) < eps

 assert abs(average\_rating('training.txt', 'awesome') - 4.0000) < eps

 assert abs(average\_rating('training.txt', 'boring') - 1.1428) < eps

 assert abs(average\_rating('training.txt', 'really') - 1.7557) < eps

 print('test\_average\_rating passed tests')

**Qu 12.** (Answer in responses document.) Find the average rating of at least five different words that are not used in the above tests. Choose words that reveal interesting characteristics of the program you have written. Give the results, and write a paragraph of about 100 words discussing these results. For example, you can consider questions such as: Which results seem to be reasonable? Are there any that are unreasonable? What do the results reveal about the strengths and weaknesses of this approach to sentiment analysis?

# Optional: estimating the sentiment of a review

Define the *score* of a word to be the average rating that it receives in training.txt, as computed by the average\_rating function. Now suppose we are given a new review, which is not in training.txt, and does not have any associated numerical rating. For example, we may be given the review 'Competently directed but terminally cute drama.'. We define the *estimated sentiment* of a review to be the average score of the words in that review. Suggested activities for this optional extension of the lab are:

* Write a function that computes the estimated sentiment of a review. Note that you can iterate over the words in a string text by using the code fragment for word in text.split():…
* Write a function that computes the average error of the estimated sentiment compared to the actual review scores in the validation.txt file.
* Discuss whether the average error is better than could be obtained by guessing the middle score (2.0) for every review.

**Acknowledgment.** The dataset of Rotten Tomatoes reviews was originally produced for a [Kaggle machine learning competition](https://www.kaggle.com/c/sentiment-analysis-on-movie-reviews/overview) and has since been adapted for a similar assignment by Eric D. Manley and Timothy M. Urness at Drake University. This lab employs their version of the dataset. This lab is a substantial rewrite by John MacCormick of an earlier version by Grant Braught, with some contributions by Lev Fruchter.