Can Computers Think? (CSC 106)
Winter 2020
Programming Project 4 **Due: Friday, 3/13/2020 (10pm)**

Sentiment analysis seeks to determine the general attitude of a writer given some text they have written. For example, given the movie review "The film was a breath of fresh air" a sentiment analysis program should realize that this is a positive statement about the movie while the review "It made me want to poke out my eye balls" expresses a negative opinion. In this project you will implement a very simple sentiment analysis system for movie reviews.

Learning Objectives:

1. Practice reading input from files.
2. Practice using dictionaries.
3. Explore sentiment analysis.

> Reminder: Programming projects are *individual* projects. You should not look at anybody else's code, show your code to anybody else, write code for anybody else or let someone else write code for you. Please see the syllabus for what to do when you need help.

1 Movie review dataset

You can download three files with movie reviews from Nexus. The movie reviews come from the Rotten Tomatoes website and their sentiment has been manually rated on a scale form 0 to 4:

0 negative

1 somewhat negative

2 neutral

3 somewhat positive

4 positive

Each file is formatted as follows:

```
0 Devoid of any of the qualities that made the first film so special .
4 The Bai brothers have taken an small slice of history and opened it up for all ...
3 Ramsay and Morton fill this character study with poetic force and buoyant ...
1 You emerge dazed , confused as to whether you 've seen pornography or documentary .
```

There is one review per line. Each line starts with the review's sentiment score followed by the words of the review. The words have already been pre-processed so that all words and punctuation symbols are separated by a blank character.

You are given the following three files with movie reviews:

- `movie_reviews_training.txt` contains 6129 movie reviews. Use this to build up a dictionary of sentiment scores for each word. (See further explanation below.)
- `movie_reviews_mini.txt` contains only the first 15 movie reviews from `movie_reviews_training.txt`. Use this to test out your function for building a word sentiment dictionary on a more manageable file.
- `movie_reviews_dev.txt` contains 800 reviews. Use this to evaluate how well your sentiment analysis system works and to fine-tune it.

2 General approach

We can estimate the sentiment of a word by averaging the sentiment scores of the reviews that the word appears in. For example, if the word *terribly* appears in two reviews with a score of 0 and one review with a score of 3, then the estimated sentiment score for the word *terribly* would be 1 (the average of 0, 0, and 3).

Once we have sentiment scores for a bunch words, we can use those to predict the sentiment of a new movie review by averaging the sentiment scores of all the words in the movie review.

3 Word sentiment dictionary

Download the starter file `sentiment_analysis.py` as well as the three files with movie reviews described above from Nexus.

Write a function `make_word_sentiment_dictionary`. The function should take the name of a file with movie reviews (in the format discussed above) as a parameter. And it should return a word sentiment dictionary. The keys in this dictionary should be individual words, and the values should be (little) dictionaries that assemble the following information about the word:

- the sum of the sentiment scores of the movie reviews in which the word appears (key: `"total score"`)
- the number of movie reviews in which the word appears (key: `"count"`)
- the average sentiment score of the movie reviews in which the word appears (key: `"average score"`)

For example, here is what an excerpt of the word sentiment dictionary might look like:

```
{'nice': {'total score': 38, 'count': 14, 'average score': 2.7142857142857144},
 'little': {'total score': 312, 'count': 154, 'average score': 2.0259740259740258},
 'story': {'total score': 538, 'count': 251, 'average score': 2.143426294820717},
 'process': {'total score': 41, 'count': 17, 'average score': 2.411764705882353},
 ...}
```

Note: Lower case all words before adding them to the dictionary. You can use the built-in string method `.lower` for this purpose.

4 Predicting sentiment scores for reviews

Now add a function `predict_sentiment_score`. Given a movie review as a string and a word sentiment dictionary, this function should return an estimated sentiment score for the review. It should estimate the score by averaging the scores of all the words in the review.

For example, given the review `"This movie is awesome !"` and the word sentiment dictionary created from the data in `movie_reviews_training.txt`, this function should return 2.41.

A movie review may contain some words that are not in word sentiment dictionary. For those words just assume that they are neutral, i.e. assume that they have a score of 2.

5 Classifying movie reviews according to their sentiment

If you test a few of the reviews from the `movie_reviews_dev.txt` file, you will see that we are not able to totally accurately predict the sentiment scores. Most predictions your function makes are closer to the average than the actual scores of the reviews. However, if all we are interested in is whether or not a movie is rated positively, that may not matter. We may still be able to use the predicted scores to decide whether we should go see a movie or not.

Add a function `is_positive` that takes a movie review and a word sentiment dictionary and classifies the review as either positive (i.e. humans would give it a score of 3 or 4) or not. The function should return a boolean value.

Try out your classification function on a few movie reviews from the `movie_reviews_dev.txt` file. The function won't get it right for all reviews, but it should get it right more often than not.

6 Fine tune your system

Now, uncomment the definition of the function `evaluate` as well as the line in the testing area that calls it. This will run your sentiment analysis system all 800 movie reviews from `movie_reviews_dev.txt` and print out some information on how many of them were classified correctly.

It should be relatively easy to classify more than 60% correctly. With further fine-tuning of `is_positive`, you may manage to get an accuracy as high as 75%.

7 Submit

You need to submit the file `sentiment_analysis.py`.

Before you submit, check that your code satisfies the following requirements:

1. Are all of your Python files properly commented? (Header comments consisting of your name and a brief description of the program in that file as well as comments within your code to clarify the logical structure of your program.)
2. Are your programs formatted neatly and consistently? Do you use whitespace (but not too much) to help a human reader understand the logical organization of your code?

3. Did you clean up your code once you got it to work? It should not contain any code snippets that don't contribute to the purpose of the program, commented out code from earlier attempts, or unnecessary comments.

Finally, submit your files to Gradescope (*Project 4: Movie Review Sentiment Analysis*).

8 Grading guidelines

- Correctness: Your program does what the problem specifications require.
- Program logic: You use programming constructs (variables, function definitions, loops etc.) in appropriate ways. Your program does not contain lines that don't contribute. Your logic is not unnecessarily complicated.
- Commenting: Your code is commented as discussed in class. (There is a header comment that states the author and briefly describes the program's purpose. There is a comment for each function. And where helpful, there are internal comments to help clarify the program's logical structure.)
- Organization: You use whitespace to indicate the logical structure of your code. Lines of code that logically belong together are close together in the file. All import statements are at the very top, followed by all function definitions, followed by all other code.