Testing sentiment analysis; recursion practice

Announcements
- Homework 10 due Thursday

Outline
- Testing a word sentiment model
- Recursion practice

Testing
- Important to evaluate the accuracy of the model
  - Use the majority of the labeled data for training and the remainder of the labeled data for testing
  - K-fold cross validation --- divide the data into \( k \) equal-sized parts; train the model on the first \( k-1 \) parts and test on the last part; then, train on a different set of \( k-1 \) parts and test on the remaining part; repeat until all \( k \) parts have been used for training once
- Example testing set
  5 This product works very well.
  1 This product is a piece of junk.
  1 Very poorly made product!
- Model is only as good as the data
  - Small dataset may not provide enough examples
    - E.g., train on 9000 Yelp or Amazon reviews results in about 66%-72% accuracy in labeling 1000 Yelp or Amazon reviews as positive, negative, or neutral
  - There may not be many examples with certain labels or features
    - E.g., given a dataset with mostly positive reviews, it is hard to determine which words imply a negative attitude
  - Inherent skew in the data may cause skewed results when used in practice
    - E.g., a disproportionate number of recidivism cases involve people of color, so a machine learning model to predict a person’s likelihood of recidivism may be negatively skewed toward people of color
- Write a function called `test` that takes a review score, a list of words (i.e., a review), and a dictionary of word ratings and returns True if the estimated review score is within the same “bucket” as the provided review score. The buckets are: < 2.5 Positive, > 3.5 Negative, 2.5-3.5 (inclusive) Neutral.
  High-level steps:
  - Call compute_score
  - Determine bucket for provided score
  - Determine bucket for computed score
  - Return True if buckets are equal; otherwise False
Recursion practice

- Write a recursive function called `count_occurrences` that takes an integer and a list of integers and returns a count of the number of times the integer appears in the list. For example:

  ```python
  count_occurrences(1, [2, 1, 3, 1, 1, 4]) should return 3
  count_occurrences(5, []) should return 0
  ```

  ```python
def count_occurrences(find, lst):
    if len(lst) == 0:
      return 0
    else:
      first = lst[0]
      rest = lst[1:]
      count = count_occurrences(find, rest)
      if first == find:
        count += 1
      return count
  ```

- Write a recursive function called `swapcase` that takes a string and returns a new string in which every uppercase letter is converted to lowercase and vice versa. For example:

  ```python
  swap_case("ABcdeF") should return "abCDEF"
  swap_case("") should return "" (empty string)
  ```

  ```python
def swap_case(string):
    if len(string) == 0:
      return ''
    else:
      first = string[0]
      rest = string[1:]
      rest_swap = swap_case(rest)
      if first.islower():
        return first.upper() + rest_swap
      else:
        return first.lower() + rest_swap
  ```

- Consider the following function:

  ```python
def mystery_for(string):
    result = ""
    for i in range(len(string), 0, -1):
      result = (string[i-1] * i) + result
    return result
  ```

  a) What is the result of `mystery_for("abcd")`?

    "abbccdddd"

  b) Write a function called `mystery_while` that behaves the same as `mystery_for` but uses a `while` loop instead of a for loop.

  ```python
def mystery_while(string):
    result = ""
    i = len(string)
    while i > 0:
      result = (string[i-1] * i) + result
      i = i - 1
    return result
  ```
c) Write a recursive function called mystery_rec that behaves the same as mystery_for but uses recursion instead of a loop.

```python
def mystery_rec(string):
    if len(string) <= 1:
        return string
    else:
        last = string[-1]
        rest = string[:-1]
        rest_result = mystery_rec(rest)
        return rest_result + last * len(string)
```

- Write a recursive function called draw_circle that takes an x- and y-coordinates, a radius, and a Turtle and produces drawings like the following:

The radius of each circle is half the radius of its enclosing circle. The minimum radius is 10.

```python
def draw_circle(x, y, r, t):
    # Go to position
    t.up()
    t.setpos(x, y)
    t.down()
    # Draw circle
    t.circle(r)
    # Draw inner circles
    r = r//2
    if r > 10:
        draw_circle(x+r, y+r, r, t)
        draw_circle(x-r, y+r, r, t)
```