Module 3 Homework

The Joy of json

Makes python dictionaries easy to write to/load from files.

```
import json

someData = {'aString':'a string', 'aNumber':132235.345, 'aList':[1,2,3],
            'anotherNestedObject:'{'name':"here's my name"} }
#save to file
with open("jsonfile.json", 'w') as F:
    F.write(json.dumps(someData))

someData = {}
with open("jsonfile.json", 'r') as F:
    someData = json.loads(F.read())

#a jsonlist file has a json object on each line
fiveHundredObjects = [{'my_id':i} for i in range(500)]
with open("fiveHundredObjects.jsonlist", 'w') as F:
    for o in fiveHundredObjects:
        F.write(json.dumps(o) + '\n') #don't forget to end the lines!

#load it back in
fiveHundredObjects = {}
with open("fiveHundredObjects.jsonlist", 'r') as F:
    fiveHundredObjects = [json.loads(line) for line in F.readlines()]
```

Homework 2

Must work in Python 3.8+ without any special libraries (besides those that are in all default Python installations, e.g. sys, os, re, etc.), unless listed below:

- You may import nltk, sklearn, and json
- You may use the lemmatizers and stemmers from nltk
- You may use CountVectorizer and the Naïve Bayes algorithms from sklearn

Homework 2

- Submit a file **hw2.py** with *no outputting code (print or any similar statements) at any level*. This file should contain the top-level functions:
  - `calcNGrams_train(trainFile)`, where:
    - `trainFile` is the name of a text file, where each line is arbitrary real-world (human-generated) text from somewhere.
    - This function should load `trainFile`, and use it to create **n-grams** (n=2, n=3, or some combination). This function must run in under 120 seconds on the TA's laptop.
  - `calcNGrams_test(sentences)`, where:
    - `sentences` is a list of strings, where each string is a sequence of words.
      - Exactly one of these strings consists of words picked entirely at random (using a distribution where all words have equal probability of being selected).
      - The other strings were generated using an n-gram-based algorithm trained on `trainFile`.
    - This function must use the n-grams trained in the previous function to determine which of the sentences in the list is entirely random.
    - **Return**: an integer i, which is the (zero-indexed) index of the sentence in `sentences` which is entirely random.

Homework 2

- Submit a file **hw2.py** with *no outputting code (print or any similar statements) at any level*. This file should ALSO contain the top-level functions:
  - `calcSentiment_train(trainFile)`, where:
    - `trainFile`: The name of a jsonlist file, where each line is a json object. Each object contains:
      - "review": A string which is the review of a movie
      - "sentiment": A Boolean value; `True` if this was a positive review, `False` if it was a negative review.
    - You **must use** a Naïve Bayes algorithm to perform this! Any further optimizations are your choice, as long as it runs under 120 seconds. You are allowed to use sklearn's naïve bayes packages.
  - `calcSentiment_test(review)`, where:
    - `review`: A string which is a review of a movie.
    - **Return**: A boolean which is the predicted sentiment of the review, as determined through Naïve Bayes.

Homework 2

- Comment your code clearly
- ANY issues harming automation may result in a grade of ‘0’. RIGOROUSLY TEST YOUR CODE. Use the example grader provided on Canvas.
- All code written must be your own and not copied from elsewhere.
- If you look at the grader file, you’ll see two values: **baselineCorrect** and **maxCorrect**. This problem is graded based on what fraction of the test set you get correct. If it is exactly **baselineCorrect**, then you get 0 points. If it’s exactly **maxCorrect**, you get 50 points (which is full credit). You can earn up to 60 (out of 50) points for this problem.

End