Wordplay

Extended from code at http://programminghistorian.org/lessons/counting-frequencies (http://programminghistorian.org/lessons/counting-frequencies)

Suppose we have two books -- maybe we'd like to see if they were written by the same author, or are otherwise similar. One approach to this is to evaluate the word use frequency in both texts, and then compute a "similarity" or "distance" measure between those two word frequencies. A related approach is to evaluate the frequency of one word being followed by another word (a "word pair"), and see the similarity in use of word pairs by the two texts. Or maybe we're interested in seeing the set of all words that come after a given word in that text.

Here we'll get some practice using **dictionaries** and **sets**, with such wordplay as our motivating example.

Some data to play with...

```
In [ ]: word_string1 = 'it was the best of times it was the worst of times '
        word_string2 = 'it was the age of wisdom it was the age of foolishness'
        word_string = word_string1 + word_string2

        words1 = word_string1.split()  #converts string with spaces to list of words
        words2 = word_string2.split()
        words = words1 + words2
        print("words:", words)
```

Practice 1

Now let's implement two procedures that will be useful for our task: `get_freq` and `get_pair_freq` .

Implement a function that computes the frequencies of the words in the given list `words` as a dictionary where keys are words and values are the number of occurrences of each word.

```
In [ ]: def get_freq(words):
            pass

        print(get_freq(words))
```

Practice 2

Implement a function that computes the frequencies of pairs of consecutive words in the given list `words` . The output should be dictionary where keys are pairs of words (pick the right data structure to store these pairs - remember that it has to be hashable!)and values are the number of occurrences of each pair in `words` .

```
In [ ]: def get_pair_freq(words):
            pass

        print(get_pair_freq(words))
```

Common strategy: build dictionaries that help answer other kinds of questions, like what (set of) words follow each word in our data?

In [ ]:
```
word_after = {}
prev = words[0]
for w in words[1:]:
    if prev not in word_after:
        word_after[prev] = {w}
    else:
        word_after[prev].add(w)
    prev = w
print("Words followed by\n" + str(word_after) + "\n")
```

A more pythonic approach is shown below, using zip() and setdefault().

- zip is very handy for jointly processing the i_th element from multiple lists. Note that in python3 zip is a generator, so is very efficient.

In [ ]:
```
# More pythonic:
word_after = {}
for w1, w2 in zip(words, words[1:]):
    word_after.setdefault(w1, set()).add(w2)
print("Words followed by\n" + str(word_after) + "\n")
```

Practice 3

Suppose we want to identify the **high frequency words** (i.e., sort word frequency from high to low). Our dictionary is "backwards" so doesn't directly answer this. An approach is to build a list (a data structure which we can sort) from information about word frequencies and then correlate the highest frequencies with the words they belong to. Implement `sort_freq_dict` below.

In [ ]:
```
def sort_freq_dict(freq):
    """
    freq: frequency dictionary
    Returns a list of tuples of the form ( frequency, word ) in
    decreasing order of the first element, the frequency.
    """
    pass

print(sort_freq_dict(get_freq(words)))
```

A more pythonic way is using the sorted() built-in function, with the `reverse` keyword argument set to True.

In [ ]:
```
#More pythonic:
def sort_freq_dict(freq):
    return sorted([(val, key) for key, val in freq.items()], reverse=True)

word_freq = get_freq(words)
words_by_frequency = sort_freq_dict(word_freq)
print(words_by_frequency)
```

Word frequency "similarity"

Next, we can build a **similarity measure** between two word frequencies. We'll use a typical "geometric" notion of distance or similarity referred to as "cosine similarity (https://en.wikipedia.org/wiki/Cosine_similarity)." We build this from vector measures of word frequency including the "norm" and the "dot product", and then calculate a (normalized) cosine distance.

The mathematical details are not crucial here -- but we do want to note how we use dictionary and set operations.

In [ ]:
```python
def freq_norm(freq):
    """ Norm of frequencies, as root-mean-sum-of-square of freqs """
    sum_square = 0
    for w, num in freq.items():    #iterates over key, val in dictionary
        sum_square += num**2
    return sum_square**0.5

def freq_dot(freq1, freq2):
    """ Dot product of two word freqs, as sum of products of freqs for words"""
    words_in_both = set(freq1) & set(freq2)
    total = 0
    for w in words_in_both:
        total += freq1[w] * freq2[w]
    return total

import math
#Returns similarity between two word (or pair) frequencies dictionaries.
# 1 indicates identical cosine word (or pair) frequencies;
# 0 indicates completely different word (pairs) frequencies
def freq_similarity(freq1, freq2):
    d = freq_dot(freq1, freq2)/(freq_norm(freq1)*freq_norm(freq2))
    ang = math.acos(min(1.0, d))
    return 1 - ang/(math.pi/2)
```

Practice 4

Implement `freq_norm` in a more pythonic way by using comprehension.

In [ ]:
```python
# REPEATING FROM ABOVE FOR CONVENIENCE
def freq_norm(freq):
    """ Norm of frequencies, as root-mean-sum-of-square of freqs """
    sum_square = 0
    for w, num in freq.items():    #iterates over key, val in dictionary
        sum_square += num**2
    return sum_square**0.5

print(freq_norm(get_freq(words)))
```

In [ ]:
```python
def freq_norm(freq):
    pass

print(freq_norm(get_freq(words)))
```

Practice 5

Implement `freq_dot` above in a more pythonic way by using comprehension.

In [ ]:
```python
# REPEATING FROM ABOVE FOR CONVENIENCE
def freq_dot(freq1, freq2):
    """ Dot product of two word freqs, as sum of products of freqs for words"""
    words_in_both = set(freq1) & set(freq2)
    total = 0
    for w in words_in_both:
        total += freq1[w] * freq2[w]
    return total
```

In [ ]:
```python
def freq_dot(freq1, freq2):
    pass
```

Test on some examples

In [ ]:
```
# Some quick tests/examples
x = {'arm':40, 'brick':2}
y = {'cat':3, 'arm':30}
print("Combined words:", set(x) | set(y))
print("freq_norm of", x, ":", freq_norm(x))
print("freq_norm of", y, ":", freq_norm(y))
print("freq_dot of", x, "and", y, ":", freq_dot(x,y))
print("freq_similarity:", freq_similarity(x,y))
```

In [ ]:
```
# Try it out with our short phrases
words3 = "this is a random sentence good enough for any random day".split()
print("words1:", words1, "\nwords2:", words2, "\nwords3:", words3, "\n")

# build word and pair frequency dictionaries, and calculate some similarities
freq1 = get_freq(words1)
freq2 = get_freq(words2)
freq3 = get_freq(words3)
print("words1 vs. words2 -- word use similarity: ", freq_similarity(freq1, freq2))
print("words1 vs. words3 -- word use similarity: ", freq_similarity(freq1, freq3))
print("words3 vs. words3 -- word use similarity: ", freq_similarity(freq3, freq3))
```

In [ ]:
```
# try that for similarity of WORD PAIR use...
pair1 = get_pair_freq(words1)
pair2 = get_pair_freq(words2)
pair3 = get_pair_freq(words3)
print("words1 vs. words2 -- pair use similarity: ", freq_similarity(pair1, pair2))
print("words1 vs. words3 -- pair use similarity: ", freq_similarity(pair1, pair3))
```

Now let's do it with some actual books!

In [ ]:
```
with open('resources/hamlet.txt', 'r') as f:
    hamlet = f.read().replace('\n', '').lower()
hamlet_words = hamlet.split()

with open('resources/macbeth.txt', 'r') as f:
    macbeth = f.read().replace('\n', '').lower()
macbeth_words = macbeth.split()

with open('resources/alice_in_wonderland.txt', 'r') as f:
    alice = f.read().replace('\n', '').lower()
alice_words = alice.split()

print(len(hamlet_words), len(macbeth_words), len(alice_words))
```

With the text from those books in hand, let's look at similarities...

In [ ]:
```
hamlet_freq = get_freq(hamlet_words)
macbeth_freq = get_freq(macbeth_words)
alice_freq = get_freq(alice_words)
print("similarity of word freq between hamlet & macbeth:", freq_similarity(hamlet_freq, macbeth_freq)
print("similarity of word freq between alice & macbeth:", freq_similarity(alice_freq, macbeth_freq))

hamlet_pair = get_pair_freq(hamlet_words)
macbeth_pair = get_pair_freq(macbeth_words)
alice_pair = get_pair_freq(alice_words)
print("\nsimilarity of word pairs between hamlet & macbeth:", \
      freq_similarity(hamlet_pair, macbeth_pair))
print("similarity of word pairs between alice & macbeth:", \
      freq_similarity(alice_pair, macbeth_pair))
```

So we've confirmed that **Macbeth** is more similar to **Hamlet** than to **Alice in Wonderland**, both in word use and in word pair usage. Good to know!