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 [1]:
```
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)
```

```
words: ['it', 'was', 'the', 'best', 'of', 'times', 'it', 'was', 'the', 'worst', 'of', 'times',
'it', 'was', 'the', 'age', 'of', 'wisdom', 'it', 'was', 'the', 'age', 'of', 'foolishness']
```

Create some interesting dictionaries from these...

In [2]:
```
# Word frequencies
word_freq = {}
for w in words:
    word_freq[w] = word_freq.get(w,0) + 1
print("Word frequencies:", word_freq)
```

```
Word frequencies: {'it': 4, 'was': 4, 'the': 4, 'best': 1, 'of': 4, 'times': 2, 'worst': 1, 'ag
e': 2, 'wisdom': 1, 'foolishness': 1}
```

In [3]:
```
# Word pair frequencies
word_pairs = {}
prev = words[0]
for w in words[1:]:
    pair = (prev, w)
    word_pairs[pair] = word_pairs.get(pair,0) + 1
    prev = w
print("Pair frequencies:", word_pairs)
```

```
Pair frequencies: {('it', 'was'): 4, ('was', 'the'): 4, ('the', 'best'): 1, ('best', 'of'): 1,
('of', 'times'): 2, ('times', 'it'): 2, ('the', 'worst'): 1, ('worst', 'of'): 1, ('the', 'age'):
2, ('age', 'of'): 2, ('of', 'wisdom'): 1, ('wisdom', 'it'): 1, ('of', 'foolishness'): 1}
```

In [4]:
```
# What words follow each word?
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")
```

```
Words followed by
{'it': {'was'}, 'was': {'the'}, 'the': {'best', 'age', 'worst'}, 'best': {'of'}, 'of': {'wisdo
m', 'foolishness', 'times'}, 'times': {'it'}, 'worst': {'of'}, 'age': {'of'}, 'wisdom': {'it'}}
```

In [5]:
```
# More Pythonic:
# 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.
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")
```

```
Words followed by
{'it': {'was'}, 'was': {'the'}, 'the': {'best', 'age', 'worst'}, 'best': {'of'}, 'of': {'wisdo
m', 'foolishness', 'times'}, 'times': {'it'}, 'worst': {'of'}, 'age': {'of'}, 'wisdom': {'it'}}
```

In [6]:
```
"""NOTE: In the above, `setdefault` is another useful method
   for dictionaries that saves a lot of initialization code:

d.setdefault(key, val)

   is equivalent to

if key not in d:
    d[key] = val
d[key]
"""
```

Out[6]:
```
'NOTE: In the above, `setdefault` is another useful method \n   for dictionaries that saves a lo
t of initialization code:\n\nd.setdefault(key, val)\n\n   is equivalent to\n\nif key not in d:\n
    d[key] = val\nd[key]\n'
```

Rewriting as procedures so we can use them later:

In [7]:
```
def get_freq(words):
    word_freq = {}
    for w in words:
        word_freq[w] = word_freq.get(w,0) + 1
    return word_freq

def get_pair_freq(words):
    word_pairs = {}
    prev = words[0]
    for w in words[1:]:
        pair = (prev, w)
        word_pairs[pair] = word_pairs.get(pair,0) + 1
        prev = w
    return word_pairs
```

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 an "inverse" data structure to make answering this question easier:

In [8]:
```
def sort_freq_dict(freq):
    aux = [(freq[key], key) for key in freq] #list comprehension
    aux.sort()    #sort in place
    aux.reverse() #reverse in place
    return aux    #return an (ordered) list, not a dictionary

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

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

```
[(4, 'was'), (4, 'the'), (4, 'of'), (4, 'it'), (2, 'times'), (2, 'age'), (1, 'worst'), (1, 'wisd
om'), (1, 'foolishness'), (1, 'best')]
```

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 [9]:
```
def freq_norm(freq):
    """ Norm of frequencies in freq, as root-mean-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

#More pythonic version:
def freq_norm(freq):
    return sum(num*num for num in freq.values())**0.5    #comprehension on just dict values

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

#More pythonic version (actually easier to understand!)
def freq_dot(freq1, freq2):
    return sum(freq1[w] * freq2[w] for w in set(freq1) & set(freq2))

import math
#Returns similarity between two word (or pair) frequencies dictionaries.
# 1 indicates identical word (or pair) frequencies;
# 0 indicates completely different words (pairs)
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)
```

In [10]:
```
# 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))
```

```
Combined words: {'brick', 'cat', 'arm'}
freq_norm of {'arm': 40, 'brick': 2} : 40.049968789001575
freq_norm of {'cat': 3, 'arm': 30} : 30.14962686336267
freq_dot of {'arm': 40, 'brick': 2} and {'cat': 3, 'arm': 30} : 1200
freq_similarity: 0.9290478472408689
```

In [11]:
```
# 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))
```

```
words1: ['it', 'was', 'the', 'best', 'of', 'times', 'it', 'was', 'the', 'worst', 'of', 'times']
words2: ['it', 'was', 'the', 'age', 'of', 'wisdom', 'it', 'was', 'the', 'age', 'of', 'foolishnes
s']
words3: ['this', 'is', 'a', 'random', 'sentence', 'good', 'enough', 'for', 'any', 'random', 'da
y']

words1 vs. words2 -- word use similarity:  0.5184249085864179
words1 vs. words3 -- word use similarity:  0.0
words3 vs. words3 -- word use similarity:  1.0
```

In [12]:
```
# 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))
```

```
words1 vs. words2 -- pair use similarity:  0.29368642735616235
words1 vs. words3 -- pair use similarity:  0.0
```

Now let's do it with some actual books!

In [13]:
```
with open('hamlet.txt', 'r') as f:
    hamlet = f.read().replace('\n', '').lower()
hamlet_words = hamlet.split()

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

with open('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))
```

```
28218 18154 15189
```

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

In [14]:
```
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_f
req))
print("similarity of word freq between alice & macbeth:", freq_similarity(alice_freq, macbeth_fre
q))

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))
```

```
similarity of word freq between hamlet & macbeth: 0.8234901643970373
similarity of word freq between alice & macbeth: 0.7242123439984074

similarity of word pairs between hamlet & macbeth: 0.3195599532068242
similarity of word pairs between alice & macbeth: 0.23412902997911367
```

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!