The log-odds ratio with an informative (and uninformative) Dirichlet prior (described in Monroe et al. 2009, Fighting Words) is a common method for finding distinctive terms in two datasets (see Jurafsky et al. 2014 for an example article that uses it to make an empirical argument). This method for finding distinguishing words combines a number of desirable properties:

- it specifies an intuitive metric (the log-odds) for comparing word probabilities in two corpora
- it incorporates prior information in the form of pseudocounts, which can either act as a smoothing factor (in the uninformative case) or incorporate real information about the expected frequency of words overall.
- it accounts for variability of a frequency estimate by essentially converting the log-odds to a z-score.

In this homework you will implement both of these ratios and compare the results.

In [84]:

```
import sys, operator, math, nltk, itertools
from collections import Counter
```

In [2]:

```
def read_and_tokenize(filename):

    with open(filename, encoding="utf-8") as file:
        tokens=[]
        # lowercase
        for line in file:
            data=line.rstrip().lower()
            # This dataset is already tokenized, so we can split on whitespace
            tokens.extend(data.split(" "))
        return tokens
```

The data we'll use in this case comes from a sample of 1000 positive and 1000 negative movie reviews from the Large Movie Review Dataset. The version of the data used in this homework has already been tokenized for you.

In [3]:

```
negative_tokens=read_and_tokenize("../data/negative.reviews.txt")
positive_tokens=read_and_tokenize("../data/positive.reviews.txt")
```

n^i = number of words in corpus i (likewise for j)

In [4]:

```
len(positive_tokens)
```

Out[4]:

```
282868
```

In [10]:

```
len(negative_tokens)
```

Out[10]:

```
266021
```

y_w^i = count of word w in corpus i (likewise for j)

File failed to load: /extensions/MathZoom.js

```
for both reviews and vocab dictionary
```

```
pos_counts=Counter()
neg_counts=Counter()

vocab={}
```

In [24]:

```
#count distinct word count in each list
for token in negative_tokens:
    neg_counts[token]+=1
    vocab[token]=1

for token in positive_tokens:
    pos_counts[token]+=1
    vocab[token]=1
```

In [20]:

```
len(pos_counts)
```

Out[20]:

```
20760
```

In [21]:

```
len(neg_counts)
```

Out[21]:

```
19198
```

V = size of vocabulary (number of distinct word types)

In [19]:

```
#total of all distinct tokens in both list
len(vocab)
```

Out[19]:

```
29595
```

Q1. Implement the log-odds ratio with an uninformative Dirichlet prior. This value, $\hat{\zeta}_w^{(i-j)}$ for word w reflecting the difference in usage between corpus i and corpus j, is given by the following equation:

$$\hat{\zeta}_w^{(i-j)} = \frac{\hat{d}_w^{(i-j)}}{\sqrt{\sigma^2\left(\hat{d}_w^{(i-j)}\right)}}$$

Where:

$$\hat{d}_w^{(i-j)} = \log\left(\frac{y_w^i + \alpha_w}{n^i + \alpha_0 - y_w^i - \alpha_w)}\right) - \log\left(\frac{y_w^j + \alpha_w}{n^j + \alpha_0 - y_w^j - \alpha_w)}\right)$$

$$\sigma^2\left(\hat{d}_w^{(i-j)}\right) \approx \frac{1}{y_w^i + \alpha_w} + \frac{1}{y_w^j + \alpha_w}$$

And:

- y_w^i = count of word w in corpus i (likewise for j)

- V = size of vocabulary (number of distinct word types)

- $\alpha_0 = V * \alpha_w$
- $n^i =$ number of words in corpus i (likewise for j)

Here the two corpora are the positive movie reviews (e.g., i = positive) and the negative movie reviews (e.g., j = negative). Using this metric, print out the 25 words most strongly aligned with the positive corpus, and 25 words most strongly aligned with the negative corpus.

In [107…

```
#one_tokens -> neg ;  two_tokens -> pos
def logodds_with_uninformative_prior(one_tokens, two_tokens, display=25):

    # complete this section

    #initialize Counters for both reviews and vocab dictionary

    neg_counts=Counter()
    pos_counts=Counter()

    vocab={}

    #count distinct word count in each list
    for token in one_tokens:
        neg_counts[token]+=1
        vocab[token]=1

    for token in two_tokens:
        pos_counts[token]+=1
        vocab[token]=1

    #constant
    a = 0.01
    n_pos = len(positive_tokens)
    n_neg = len(negative_tokens)
    V = len(vocab)

    #numerator
    for k, v in vocab.items():
        v1 = (pos_counts[k] + a) / (n_pos + a * V - pos_counts[k] - a)
        v2 = (neg_counts[k] + a) / (n_neg + a * V - neg_counts[k] - a)

        p_pos = math.log(v1)
        p_neg = math.log(v2)

        n = p_pos - p_neg

        #denomanator

        sig = (1 / (pos_counts[k] + a)) + (1 / (neg_counts[k] + a))
        d = math.sqrt (sig)

        #output
        o = n / d

        #update the dictionary value to the new output
        vocab[k] = o

    #sort dictionary
    sorted_dict = sorted(vocab.items(), key=operator.itemgetter(1), reverse=True)
```

File failed to load: /extensions/MathZoom.js

```python
    positiveTopWords = sorted_dict[:display]
    negativeTopWords = sorted_dict[-display:][::-1]

    return positiveTopWords, negativeTopWords
```

In [108…

```python
logodds_with_uninformative_prior(negative_tokens, positive_tokens)
```

Out[108…

```
([('great', 9.607540224421118),
  ('his', 8.444585050679482),
  ('best', 8.221173530354823),
  ('as', 8.068276896558293),
  ('and', 8.008455816960856),
  ('love', 7.4664889052138665),
  ('war', 7.231860280397061),
  ('excellent', 7.142854492584097),
  ('wonderful', 6.697252740688556),
  ('is', 6.559130822066367),
  ('her', 6.38883111352343),
  ('performance', 6.052211880700309),
  (',', 5.93698228878491),
  ('of', 5.768768034806312),
  ('life', 5.7217812496838585),
  ('highly', 5.706899505669689),
  ('world', 5.664462780541281),
  ('perfect', 5.547139675935874),
  ('in', 5.489693080913076),
  ('always', 5.465521212926001),
  ('performances', 5.380094126983327),
  ('beautiful', 5.3556137393424645),
  ('most', 5.198384926387417),
  ('tony', 5.148053450218753),
  ('loved', 5.0921802161916085)],
 [('bad', -15.874118374895222),
  ('?', -15.035079097668289),
  ("n't", -11.949393783552106),
  ('movie', -10.959996673992299),
  ('worst', -9.92867459130721),
  ('i', -9.448181178355652),
  ('just', -9.122270515824324),
  ('...', -8.675997956464013),
  ('was', -8.617584504048052),
  ('no', -7.999249396143025),
  ('do', -7.521169947814628),
  ('awful', -7.511891984576927),
  ('terrible', -7.446279268276979),
  ('they', -7.372767849274727),
  ('horrible', -7.052577619117426),
  ('why', -7.019505671855516),
  ('this', -6.934937867357723),
  ('poor', -6.930684966472034),
  ('boring', -6.709363182052385),
  ('any', -6.684833313059192),
  ('waste', -6.674077981733288),
  ('script', -6.6612890317425215),
  ('worse', -6.6012235452154595),
  ('have', -6.55152365358799),
  ('stupid', -6.4750566877612865)])
```

File failed to load: /extensions/MathZoom.js onstant α_w in the equations above, what happens to $\hat{\zeta}_w^{(i-j)}$, $\hat{d}_w^{(i-j)}$ and

$\sigma^2\left(\hat{d}_w^{(i-j)}\right)$ (i.e., do they get bigger or smaller)? Answer this by plugging the following values in your implementation of these two quantities, and varying α_w (and, consequently, α_0).

- $y_w^i = 34$
- $y_w^j = 17$
- $n^i = 1000$
- $n^j = 1000$
- $V = 500$

$$\hat{d}_w^{(i-j)} = \log\left(\frac{y_w^i + \alpha_w}{n^i + \alpha_0 - y_w^i - \alpha_w)}\right) - \log\left(\frac{y_w^j + \alpha_w}{n^j + \alpha_0 - y_w^j - \alpha_w)}\right)$$

$$\sigma^2\left(\hat{d}_w^{(i-j)}\right) \approx \frac{1}{y_w^i + \alpha_w} + \frac{1}{y_w^j + \alpha_w}$$

In [96]:

```
def explore_alpha(a):

    #constant
    #a = 0.01
    y_i = 34
    y_j = 17
    n_i = 1000
    n_j = 1000
    V = 500

    #numerator

    v1 = (y_i + a) / (n_i + a * V - y_i - a)
    v2 = (y_j + a) / (n_j + a * V - y_j - a)

    p_pos = math.log(v1)
    p_neg = math.log(v2)

    n = p_pos - p_neg

    #denomanator

    sig = (1 / (y_i + a)) + (1 / (y_j + a))
    d = math.sqrt (sig)

    #output
    o = n / d

    return o, n, sig
```

In [97]:

```
original_a = 0.01
explore_alpha(original_a)
```

Out[97]:

```
(2.391507700522409 7, 0.7102095992733704, 0.08819206440820998)
```

File failed to load: /extensions/MathZoom.js

```
bigger_a = 10
```

```
explore_alpha(bigger_a)
```

Out[98]:

```
(2.0092779913600722, 0.4912029668423381, 0.05976430976430976)
```

As alpha increase, these values decrease.

Now let's make that prior informative by including information about the overall frequency of a given word in a background corpus (i.e., a corpus that represents general word usage, without regard for labeled subcorpora). To do so, there are only two small changes to make:

- We need to gather a background corpus b and calculate $\hat{\pi}_w$, the relative frequency of word w in b (i.e., the number of times w occurs in b divided by the number of words in b).
- In the uninformative prior above, α_w was a constant (0.01) and $\alpha_0 = V * \alpha_w$. Let us now set $\alpha_0 = 1000$ and $\alpha_w = \hat{\pi}_w * \alpha_0$. This reflects a pseudocount capturing the fractional number of times we would expect to see word w in a sample of 1000 words.

This allows us to specify that a common word like "the" (which has a relative frequency of ≈ 0.04) would have $\alpha_w = 40$, while an infrequent word like "beneficiaries" (relative frequency ≈ 0.00002) would have $\alpha_w = 0.02$.

Q3. Implement a log-odds ratio with informative prior, using a larger background corpus of 5M tokens drawn from the same dataset (given to you as `priors` below, which contains the relative frequencies of words calculated from that corpus) and set $\alpha_0 = 1000$. Using this metric, print out again the 25 words most strongly aligned with the positive corpus, and 25 words most strongly aligned with the negative corpus. Is there a meaningful difference?

In [99]:

```
def read_priors(filename):
    counts=Counter()
    freqs={}
    tokens=read_and_tokenize(filename)
    total=len(tokens)

    for token in tokens:
        counts[token]+=1

    for word in counts:
        freqs[word]=counts[word]/total

    return freqs

priors=read_priors("../data/sentiment.background.txt")
```

In [109…

```
def logodds_with_informative_prior(one_tokens, two_tokens, priors, display=25):

    # complete this section
     # complete this section

    #initialize Counters for both reviews and vocab dictionary

    neg_counts=Counter()
    pos_counts=Counter()

    vocab={}
```

File failed to load: /extensions/MathZoom.js

```
    #count distinct word count in each list
    for token in one_tokens:
        neg_counts[token]+=1
        vocab[token]=1

    for token in two_tokens:
        pos_counts[token]+=1
        vocab[token]=1

    #constant
    a_0 = 1000
    n_pos = len(positive_tokens)
    n_neg = len(negative_tokens)
    V = len(vocab)

    #numerator
    for k, v in vocab.items():
        v1 = (pos_counts[k] + a_0 * priors[k]) / (n_pos + a_0 - pos_counts[k] - a_0 * p
        v2 = (neg_counts[k] + a_0 * priors[k]) / (n_neg + a_0 - neg_counts[k] - a_0 * p

        p_pos = math.log(v1)
        p_neg = math.log(v2)

        n = p_pos - p_neg

        #denomanator

        sig = (1 / (pos_counts[k] + a_0 * priors[k])) + (1 / (neg_counts[k] + a_0 * pri
        d = math.sqrt (sig)

        #output
        o = n / d

        #update the dictionary value to the new output
        vocab[k] = o

    #sort dictionary
    sorted_dict = sorted(vocab.items(), key=operator.itemgetter(1), reverse=True)
    #get the ranks
    positiveTopWords = sorted_dict[:display]
    negativeTopWords = sorted_dict[-display:][::-1]

    return positiveTopWords, negativeTopWords
```

In [110…

```
logodds_with_informative_prior(negative_tokens, positive_tokens, priors)
```

Out[110…

```
([('great', 9.591039308150844),
  ('his', 8.427876598766474),
  ('best', 8.207390895133855),
  ('as', 8.051850426312377),
  ('and', 7.990231407305706),
  ('love', 7.4541507138623375),
  ('war', 7.2257038478692515),
  ('excellent', 7.136307614907904),
  ('wonderful', 6.692625533714691),
  ('is', 6.543717647282005),
  ('her', 6.3771[illegible]4200775),
  ('performance', [illegible]042737335969907),
```

```
 (',', 5.9215701353486665),
 ('of', 5.754969742770618),
 ('life', 5.712104514477569),
 ('highly', 5.703588456479672),
 ('world', 5.6554566648511475),
 ('perfect', 5.540329074257345),
 ('in', 5.477091362874526),
 ('always', 5.456154882740344),
 ('performances', 5.371497571484572),
 ('beautiful', 5.346392148729829),
 ('most', 5.188839322996821),
 ('tony', 5.151557465549794),
 ('loved', 5.08520207589855)],
[('bad', -15.858436155657074),
 ('?', -15.012171561204545),
 ("n't", -11.92973245486743),
 ('movie', -10.942156841842946),
 ('worst', -9.958385464468778),
 ('i', -9.43394122124469),
 ('just', -9.10719606286112),
 ('...', -8.661491103140472),
 ('was', -8.60428995816332),
 ('no', -7.98623150105582),
 ('awful', -7.513361158186305),
 ('do', -7.509005013285732),
 ('terrible', -7.450145065620428),
 ('they', -7.360873277110952),
 ('horrible', -7.054947486972777),
 ('why', -7.008333826955033),
 ('this', -6.924931742171541),
 ('poor', -6.923265303686221),
 ('waste', -6.719492793687231),
 ('boring', -6.70218207705369),
 ('any', -6.674061366310463),
 ('script', -6.651941912393764),
 ('worse', -6.597130696108337),
 ('have', -6.541331663788079),
 ('stupid', -6.468495330825354)])
```

There's no differences in the words, but the value has changed. The words are the same, most likely because they have high frequencies.

In [ ]:

File failed to load: /extensions/MathZoom.js