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 [194…

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

In [195…

```
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 whites
            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 [196…

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

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)
- α_w = 0.01
- 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 [ ]:

In [197…

```
def logodds_with_uninformative_prior(one_tokens, two_tokens, display=25):
    # complete this section
    all_tokens = one_tokens + two_tokens
    V = set(all_tokens)
    # y_w_i
    one_tokens_count = dict()
    # y_w_j
    two_tokens_count = dict()
    log_term_one = dict()
    log_term_two = dict()
    # difference between both log terms
    log_diff = dict()
    variance = dict()
    stdev = dict()
    # final score
    z_score = dict()
    # calculate y_w_i and y_w_j for all words in the vocabulary
    for token in V:
        one_tokens_count[token] = one_tokens.count(token)
        two_tokens_count[token] = two_tokens.count(token)
    alpha_w = 0.01
    alpha_0 = len(V) * alpha_w
    # n_i
    one_tokens_size = len(one_tokens)
    # n_j
    two_tokens_size = len(two_tokens)
    for word in V:
        log_term_one[word] = math.log((one_tokens_count[word] + alpha_w) /
        log_term_two[word] = math.log((two_tokens_count[word] + alpha_w) /
        log_diff[word] = log_term_one[word] - log_term_two[word]
        # calculate standard deviation
        variance[word] = ((1/(one_tokens_count[word] + alpha_w)) + (1/(two_
        stdev[word] = math.sqrt(variance[word])
        # final score
        z_score[word] = log_diff[word]/stdev[word]
    sorted_zscores = sorted(z_score.items(), key=lambda item: item[1])
    print("words most strongly aligned with negative reviews:")
    for score in sorted_zscores[:display]:
        print(score)
    print("words most strongly aligned with positive reviews:")
    for score in sorted_zscores[-25:]:
        print(score)
```

In [198…

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

```
words most strongly aligned with negative reviews:
('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)
words most strongly aligned with positive reviews:
('loved', 5.0921802161916085)
('tony', 5.148053450218753)
('most', 5.198384926387417)
('beautiful', 5.3556137393424645)
('performances', 5.380094126983327)
('always', 5.465521212926001)
('in', 5.489693080913076)
('perfect', 5.547139675935874)
('world', 5.664462780541281)
('highly', 5.706899505669689)
('life', 5.7217812496838585)
('of', 5.768768034806312)
(',', 5.93698228878491)
('performance', 6.052211880700309)
('her', 6.38883111352343)
('is', 6.559130822066367)
('wonderful', 6.697252740688556)
('excellent', 7.142854492584097)
('war', 7.231860280397061)
('love', 7.4664889052138665)
('and', 8.008455816960856)
('as', 8.068276896558293)
('best', 8.221173530354823)
('his', 8.444585050679482)
('great', 9.607540224421118)
```

Q2: As you increase the constant 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).

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

Answer: The values seem to be decreasing as a_w is varied.

In [199…

```
def logodds_with_uninformative_prior_Q2(one_tokens, two_tokens):
    # complete this section
    y_i_w = 34
    y_j_w = 17
    n_i = 1000
    n_j = 1000
    V = 500
    a_w_list = [0.01, 0.1, 1, 3, 5]
    for a_w in a_w_list:
        a_0 = V * a_w
        log_one = math.log((y_i_w + a_w) / (n_i + a_0 - y_i_w - a_w))
        log_two = math.log((y_j_w + a_w) / (n_j + a_0 - y_j_w - a_w))
        log_diff = log_one - log_two
        stdev = (1/(y_i_w + a_w)) + (1/(y_j_w + a_w))
        z_score = log_diff / math.sqrt(stdev)
        print("Values for a_w = ", a_w,":")
        print("stdev: ",stdev)
        print("d_w: ", log_diff)
        print("z_score: ", z_score,"\n")
logodds_with_uninformative_prior_Q2(positive_tokens, negative_tokens)
```

```
Values for a_w =  0.01 :
stdev:  0.08819206440820998
d_w:  0.7102095992733704
z_score:  2.3915077005224097

Values for a_w =  0.1 :
stdev:  0.08780504536022363
d_w:  0.7068143816959451
z_score:  2.3853144648767124

Values for a_w =  1 :
stdev:  0.08412698412698412
d_w:  0.6765135879981137
z_score:  2.3324313166697372

Values for a_w =  3 :
stdev:  0.07702702702702703
d_w:  0.6220640801287232
z_score:  2.241370009622216

Values for a_w =  5 :
stdev:  0.0710955710955711
d_w:  0.5774190440502149
z_score:  2.1655580494073816
```

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 [200…

```
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 [201…

```
def logodds_with_informative_prior(one_tokens, two_tokens, priors, display=
    # complete this section
    all_tokens = one_tokens + two_tokens
    V = set(all_tokens)
    one_tokens_count = dict()
    two_tokens_count = dict()
    log_diff = dict()
    variance = dict()
    stdev = dict()
    log_term_one = dict()
    log_term_two = dict()
    alpha_w = dict()
    z_score = dict()
    for token in V:
        one_tokens_count[token] = one_tokens.count(token)
        two_tokens_count[token] = two_tokens.count(token)
    alpha_0 = 1000.0
    one_tokens_size = len(one_tokens)
    two_tokens_size = len(two_tokens)
    #calculate most strongly aligned words with corpus one
    #calculate log term
    for word in V:
        alpha_w[word] = priors[word] * alpha_0
        log_term_one[word] = math.log((one_tokens_count[word] + alpha_w[wo
        log_term_two[word] = math.log((two_tokens_count[word] + alpha_w[wo
        log_diff[word] = log_term_one[word] - log_term_two[word]
        #calculate standard deviation
        variance[word] = ((1/(one_tokens_count[word] + alpha_w[word])) + (
        stdev[word] = math.sqrt(variance[word])
        z_score[word] = log_diff[word]/stdev[word]
    sorted_zscores = sorted(z_score.items(), key=lambda item: item[1])
    print("negative:")
    for i in sorted_zscores[:display]:
        print(i)
    print("positive:")
    for i in sorted_zscores[-25:]:
        print(i)
```

In [202…

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

```
negative:
('bad', -15.81836305010025)
('?', -14.947622559858601)
("n't", -11.815474486956932)
('movie', -10.806524116178911)
('worst', -9.953089748636955)
('i', -9.196606767018425)
('just', -9.06140313065345)
('...', -8.632836680955426)
('was', -8.475482376722722)
('no', -7.957764060198799)
('awful', -7.510624608399143)
('do', -7.470913986493858)
('terrible', -7.447597213269431)
('they', -7.312015445148056)
('horrible', -7.052757149373359)
('why', -6.998125020986118)
('poor', -6.919688335083014)
('this', -6.774744719676981)
('waste', -6.71790227167431)
('boring', -6.69886046630574)
('any', -6.65932067794369)
('script', -6.646104180026674)
('worse', -6.594852288437093)
('have', -6.486327243629887)
('stupid', -6.465921941287761)
positive:
('loved', 5.08329726141815)
('tony', 5.15070412801554)
('most', 5.1749359389737855)
('in', 5.325752226227236)
('beautiful', 5.343549998652135)
('performances', 5.369257524450838)
(',', 5.448405249802173)
('always', 5.451257273749744)
('of', 5.508254865540279)
('perfect', 5.5378157390733)
('world', 5.648943020246218)
('life', 5.701034146690215)
('highly', 5.70204848795536)
('performance', 6.037821691032667)
('is', 6.336666027183984)
('her', 6.340752279574094)
('wonderful', 6.690019311974002)
('excellent', 7.132917297392506)
('war', 7.222016000500284)
('love', 7.440079484895621)
('and', 7.611445816715601)
('as', 7.94086252986073)
('best', 8.19291714830546)
('his', 8.356245720100485)
('great', 9.567364135154286)
```

In [ ]: