CSE 158 HW 4

Q1:

```
(4587, 'with a'),
(2595, 'in the'),
(2245, 'of the'),
(2056, 'is a'),
(2033, 'on the')

numbre of unique bigrams:  182246
```

Q2:

```
MSE: 0.343153014061
```

Q3:

```
MSE 0.289047333034
```

Q4:

```
Negative:  ['sort of', 'water', 'corn', 'the background', 'straw']
Positive:  ['sort', 'a bad', 'of these', 'not bad', 'the best']
```

Q5:

IDF:

idf foam : -1.13786862069

idf smell : -0.537901618865

idf banana : -1.67778070527

idf lactic : -2.92081875395

idf tart : -1.80687540165

TF-IDF in first review

'foam': -2.2757372413739256,

'smell': -0.5379016188648442,

'banana': -3.355561410532161,

'lactic': -5.84163750790475,

'tart': -1.8068754016455382

Q6:

cosine similariy: 0.106130241679

Q7:

beer review most similar to first :

beerId: 52211

profileName: Heatwave33

Q8:

MSE: 0.278759560078

HW4

March 25, 2018

In [122]:
```
import numpy
import urllib
import scipy.optimize
import random
from collections import defaultdict
from sklearn import linear_model
from sklearn.metrics import mean_squared_error
import string


def parseData(fname):
  for l in urllib.urlopen(fname):
    yield eval(l)

print "Reading data..."
data = list(parseData("beer_50000.json"))[:5000]
print "done"
punctuation = set(string.punctuation)
```

```
Reading data...
```

```
---------------------------------------------------------------------------

KeyboardInterrupt                         Traceback (most recent call last)

<ipython-input-122-6b1e48a13984> in <module>()
 14
 15 print "Reading data..."
---> 16 data = list(parseData("beer_50000.json"))[:5000]
 17 print "done"
 18 punctuation = set(string.punctuation)


<ipython-input-122-6b1e48a13984> in parseData(fname)
 11 def parseData(fname):
 12   for l in urllib.urlopen(fname):
```

```
---> 13     yield eval(l)
     14
     15 print "Reading data..."


KeyboardInterrupt:
```

In [ ]:
```
print 'Question 1:'
bigramCount = defaultdict(int)
for d in data:
    r = ''.join([c for c in d['review/text'].lower() if not c in punctuation])
    wl = r.split()
    if len(wl) > 1:
        for i in range(len(wl)-1):
            bg = wl[i] + ' ' + wl [i+1]
            bigramCount[bg] += 1

mostPop =  [(bigramCount[bg],bg)for bg in bigramCount.keys()]
mostPop.sort()
mostPop.reverse()
print 'Answer 1:'
print mostPop[:5]
print 'numbre of unique bigrams: ', len(bigramCount.keys())
```

In [ ]:
```
print 'Question 2:'
bigrs = [mp[1] for mp in mostPop[:1000]]
bigrsId = dict(zip(bigrs, range(len(bigrs))))
bigrsSet = set(bigrs)

def getbgs(wordlist):
  bgs = []
  for i in range(len(wordlist)-1):
    s = wordlist[i] + ' ' + wordlist[i+1]
    bgs.append(s)
  return bgs

def featureQ2(datum):
  feat = [0]*1000
  r = ''.join([c for c in datum['review/text'].lower() if not c in punctuation])
  bgs = getbgs(r.split())
  for b in bgs:
    if b in bigrsSet:
      feat[bigrsId[b]] += 1
  feat.append(1) #offset
  return feat

X = [featureQ2(d) for d in data]
```

```
y = [d['review/overall'] for d in data]

clf = linear_model.Ridge(1.0, fit_intercept=False)
clf.fit(X, y)
predictions = clf.predict(X)
print "MSE: " + str(mean_squared_error(predictions,y))
```

In [ ]:
```
print 'Question 3:'

def getbgs(wordlist):
  bgs = []
  for i in range(len(wordlist)-1):
    s = wordlist[i] + ' ' + wordlist[i+1]
    bgs.append(s)
  return bgs

b_n_u = defaultdict(int)
punctuation = set(string.punctuation)

for d in data:
  r = ''.join([c for c in d['review/text'].lower() if not c in punctuation])
  wordlist = r.split()
  for w in wordlist:
    b_n_u[w] += 1
  blist = getbgs(wordlist)
  for s in blist:
    b_n_u[s] += 1

bcounts = [(b_n_u[w], w) for w in b_n_u.keys()]
bcounts.sort()
bcounts.reverse()

mix1000 = [x[1] for x in bcounts[:1000]]
mixgramId = dict(zip(mix1000, range(len(mix1000))))
mix1000Set = set(mix1000)

def featureQ3(datum):
  feat = [0]*1000
  r = ''.join([c for c in datum['review/text'].lower() if not c in punctuation])
  wordlist = r.split()
  for w in wordlist:
    if w in mix1000Set:
      feat[mixgramId[w]] += 1
  blist = getbgs(wordlist)
  for b in blist:
    if b in mix1000Set:
      feat[mixgramId[b]] += 1
  feat.append(1) #offset
```

```
    return feat

X = [featureQ3(d) for d in data]
y = [d['review/overall'] for d in data]

clf = linear_model.Ridge(1.0, fit_intercept=False)
clf.fit(X, y)
predictions = clf.predict(X)
theta = clf.coef_
print "MSE " + str(mean_squared_error(predictions,y))
```

In [ ]:
```
print 'Question 4:'
weights = zip (theta[:1000],range(len(theta[:1000])))
weights.sort()
neg = [mix1000[weights[i][1]] for i in range(5)]
print 'Negative: ', neg
weights.reverse()
pos = [mix1000[weights[i][1]] for i in range(5)]
print 'Positive: ', pos
```

In [ ]:
```
import math
print 'Question 5:'
idf = defaultdict(float)
punctuation = set(string.punctuation)
for d in data:
  r = ''.join([c for c in d['review/text'].lower() if not c in punctuation])
  for w in set(r.split()):
    idf[w] += 1
docs = len(data)
for k in idf:
  idf[k] = math.log(idf[k]*1.0/docs)/math.log(10)
t = ['foam', 'smell', 'banana', 'lactic', 'tart']
tidf = defaultdict(float)
for w in t:
  print "idf " + w + " : " + str(idf[w])
d = data[0]
r = ''.join([c for c in d['review/text'].lower() if not c in punctuation])
for w in r.split():
  if w in t:
    tidf[w] += idf[w]
tidf = dict(tidf)
print tidf
```

In [ ]:
```
print 'Question 6:'

def idf(t):
    freq = 0
    for d in data:
```

```
            r = ''.join([c for c in d['review/text'].lower() if not c in punctuation])
            if t in r.split():
                freq += 1
        return -numpy.log10(freq * 1.0 / len(data))

    def tf(t, i):
        freq = 0
        r = ''.join([c for c in data[i]['review/text'].lower() if not c in punctuation])
        for w in r.split():
            if w == t:
                freq += 1
        return freq

    uniCount = defaultdict(int)
    for d in data:
        r = ''.join([c for c in d['review/text'].lower() if not c in punctuation])
        for w in r.split():
            uniCount[w] += 1
    counts = [(uniCount[w], w) for w in uniCount]
    counts.sort()
    counts.reverse()
    uni1000 = [x[1] for x in counts[:1000]]
    uni1000_idf = defaultdict(float)
    for w in uni1000:
        uni1000_idf[w] = idf(w)
    tidrv1 = [tf(w, 0) * uni1000_idf[w] for w in uni1000]
    tidrv2 = [tf(w, 1) * uni1000_idf[w] for w in uni1000]
    dot_prod = sum(tidrv1[i] * tidrv2[i] for i in range(1000))
    two_norm_prod = numpy.sqrt(sum(tidrv1[i]**2 for i in range(1000)))*numpy.sqrt(sum(tidrv2
    cs = float(dot_prod) / two_norm_prod
    print "cosine similariy: " + str(cs)
```

In [ ]:
```
print 'Question 7:'
maxi = 0
maxcs = -1000
tidrv1 = [tf(w, 0) * uni1000_idf[w] for w in uni1000]
for i in range(1, len(data)):
    tidrv2 = [tf(w, i) * uni1000_idf[w] for w in uni1000]
    dot_prod = sum(tidrv1[i] * tidrv2[i] for i in range(1000))
    two_norm_prod = numpy.sqrt(sum(tidrv1[i]**2 for i in range(1000))) * numpy.sqrt(sum(
    cs = float(dot_prod) /two_norm_prod
    if cs > maxcs:
        maxcs = cs
        maxi = i
print "beer most similar to first : "
print 'beerId: ', data[maxi]['beer/beerId']
print 'profileName: ', data[maxi]['user/profileName']
```

In [ ]:
```
print 'Question 8:'
```

```
wordCount = defaultdict(int)
punctuation = set(string.punctuation)
for d in data:
  r = ''.join([c for c in d['review/text'].lower() if not c in punctuation])
  for w in r.split():
    wordCount[w] += 1
counts = [(wordCount[w], w) for w in wordCount]
counts.sort()
counts.reverse()
words = [x[1] for x in counts[:1000]]
wordId = dict(zip(words, range(len(words))))
idf = defaultdict(float)
for d in data:
  r = ''.join([c for c in d['review/text'].lower() if not c in punctuation])
  for w in set(r.split()):
    if w in words:
      idf[w] += 1
docs = len(data)
for k in idf:
  idf[k] = math.log(idf[k]*1.0/docs)/math.log(10)


def featureQ8(datum):
  feat = [0]*1000
  r = ''.join([c for c in datum['review/text'].lower() if not c in punctuation])
  for w in r.split():
    if w in words:
      feat[wordId[w]] += idf[w]
  feat.append(1) #offset
  return feat

X = [featureQ8(d) for d in data]
y = [d['review/overall'] for d in data]
clf = linear_model.Ridge(1.0, fit_intercept=False)
clf.fit(X, y)
theta = clf.coef_
predictions = clf.predict(X)
print "MSE: " + str(mean_squared_error(predictions,y))
```