In [85]:

```
import numpy as np
import urllib
import scipy.optimize
from random import shuffle
import string
from collections import defaultdict
from nltk import bigrams
from nltk.stem.porter import *
from sklearn import linear_model
from sklearn.metrics import mean_squared_error
from sklearn.metrics.pairwise import cosine_similarity
from math import log
```

In [86]:

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

In [87]:

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

```
Reading data...
done
```

Problem 1

In [88]:

```
### Ignore capitalization and remove punctuation
bigramCount = defaultdict(int)
punctuation = set(string.punctuation)
#stemmer = PorterStemmer()
for d in data:
    r = ''.join([c for c in d['review/text'].lower() if not c in punctuatio
    words = r.split()
    for i in range(0,len(words)-1):
        bigram = words[i] + " " + words[i+1]
        bigramCount[bigram] += 1
```

In [89]:

```
# Find the top k words
countsBigram = [(bigramCount[d], d) for d in bigramCount.keys()]
countsBigram.sort()
countsBigram.reverse()
```

In [90]:

```
k = 5
kTopBigram = [d for d in countsBigram[:k]]
print("Most used bigrams: ", kTopBigram)
```

```
Most used bigrams:  [(4587, 'with a'), (2595, 'in the'), (2245, 'of th
e'), (2056, 'is a'), (2033, 'on the')]
```

Problem 2

In [91]:

```
bigrams = [c[1] for c in countsBigram[:1000]]
bigramId = dict(zip(bigrams, range(len(bigrams))))
bigramSet = set(bigrams)
```

In [92]:

```
def feature(text):
    feat = [0]*len(bigrams)
    words = text.split()
    for i in range(len(words)-1):
        bigram = words[i] + " " + words[i+1]
        try:
            feat[bigramId[bigram]] += 1
        except KeyError:
            continue
    feat.append(1) #offset
    return feat
```

In [93]:

```
##reviewText = [''.join([c for c in d['review/text'].lower() if not c in pu
```

In [94]:

```
reviewText = [''.join([c for c in datum['review/text'].lower() if not c in
```

In [95]:

```
X_2 = []
for i in range(len(data)):
    X_2.append(feature(reviewText[i]))
y_2 = [d['review/overall'] for d in data]
```

In [96]:

```
# Least squares with regularization
reg = 1.0
clf = linear_model.Ridge(reg, fit_intercept=False)
clf.fit(X_2, y_2)
theta = clf.coef_
predictions = clf.predict(X_2)
```

In [97]:

```
print("MSE:", mean_squared_error(y_2, predictions))
```

```
MSE: 0.34315301406136334
```

In [ ]:

Problem 3

In [98]:

```
# Find idf
df = defaultdict(int)
for d in data:
    r = ''.join([c for c in d['review/text'].lower() if not c in punctuatio
    words = set(r.split())
    for w in words:
        df[w] += 1
```

In [99]:

```
def idf(word):
    f = df[word]
    if f == 0:
        # Return maximum idf
        return log(len(data), 10)
    return log(len(data) / float(f))
```

In [100]:

```
def tf(word, reviewText):
    words = reviewText.split()
    c = 0
    for w in words: # Could use stemming here
        if w == word:
            c += 1
    return c
```

In [101]:

```
def tf_idf(word, reviewText):
    return tf(word, reviewText) * idf(word)
```

In [102]:

```
words = ['foam', 'smell', 'banana', 'lactic', 'tart']
print("IDF / TF-IDF for the words: ")
for w in words:
    print("Word: {:5s} \t IDF: {:2.4f}, \t TF-IDF: {:2.4f}, \t TF: {:2.0f}
    .format(w, idf(w), tf_idf(w, reviewText[0]), tf(w, reviewText[0])))
```

```
IDF / TF-IDF for the words:
Word: foam       IDF: 2.6200,    TF-IDF: 5.2401,         TF:  2
Word: smell      IDF: 1.2386,    TF-IDF: 1.2386,         TF:  1
Word: banana     IDF: 3.8632,    TF-IDF: 7.7265,         TF:  2
Word: lactic     IDF: 6.7254,    TF-IDF: 13.4509,        TF:  2
Word: tart       IDF: 4.1605,    TF-IDF: 4.1605,         TF:  1
```

In [ ]:

Problem 4

In [103]:

```
unigramCount = defaultdict(int)
for d in data:
    r = ''.join([c for c in d['review/text'].lower() if not c in punctuati
    for w in r.split():
        unigramCount[w] += 1
```

In [104]:

```
countsUnigram = [(unigramCount[w], w) for w in unigramCount]
countsUnigram.sort()
countsUnigram.reverse()
```

In [105]:

```
unigrams = [x[1] for x in countsUnigram[:1000]]
```

In [106]:

```
unigramId = dict(zip(unigrams, range(len(unigrams))))
```

In [107]:

```
def feature_4(reviewText):
    feat = [0]*len(unigrams)
    words = reviewText.split()
    for w in words:
        try:
            feat[unigramId[w]] = tf_idf(w, reviewText)
        except KeyError:
            continue
    feat.append(1) #offset
    return feat
```

In [108]:

```
X_4 = np.array([feature_4(d) for d in reviewText])
y_4 = np.array([d['review/overall'] for d in data])
```

In [109]:

```
print("Cosine similarity between review 1 and 2:", cosine_similarity(X_4[0
```

```
Cosine similarity between review 1 and 2: 0.10979549082335394
```

In [ ]:

Problem 5 ¶

In [110]:
```
similarities = []
for i in range(1,len(data)):
    d = data[i]
    similarity = cosine_similarity(X_4[0:1], X_4[i:i+1])[0,0]
    similarities.append((similarity, (d['beer/beerId'], d['user/profileNam
similarities.sort()
similarities.reverse()
```

In [111]:
```
top = similarities[0]
print("Top cosine similarity: {:0.8f}, userId: {}, beerId: {}".format(top[
```

```
Top cosine similarity: 0.31944885, userId: Heatwave33, beerId: 52211
```

Problem 6

In [112]:
```
clf = linear_model.Ridge(1.0, fit_intercept=False)
clf.fit(X_4, y_4)
predictions_4 = clf.predict(X_4)
```

In [113]:
```
print("MSE is: ", mean_squared_error(predictions_4, y_4))
```

```
MSE is:  0.2787424900566092
```

Problem 7

In [154]:

```
import numpy as np
import urllib
import scipy.optimize
from random import shuffle
import string
from collections import defaultdict
from nltk import bigrams
from nltk.stem.porter import *
from sklearn import linear_model
from sklearn.metrics import mean_squared_error
from sklearn.metrics.pairwise import cosine_similarity
from math import log
```

In [155]:

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

In [156]:

```
print("Reading data...")
data = list(parseData("file:beer_50000.json"))
print("done")
```

```
Reading data...
done
```

In [157]:

```
shuffle(data)
```

In [158]:

```
data_train = data[:5000]
data_valid = data[5000:10000]
data_test = data[10000:15000]
```

1. unigrams, removing punctuation, tfidf

In [159]:

```
punctuation = set(string.punctuation)
reviewText = [''.join([c for c in datum['review/text'].lower() if not c in
```

In [160]:

```
unigramCount = defaultdict(int)
for d in data:
    r = ''.join([c for c in d['review/text'].lower() if not c in punctuati
    for w in r.split():
        unigramCount[w] += 1
```

In [161]:

```
countsUnigram = [(unigramCount[w], w) for w in unigramCount]
unigrams = [x[1] for x in countsUnigram[:1000]]
unigramId = dict(zip(unigrams, range(len(unigrams))))
unigramSet = set(unigrams)
```

In [162]:
```
countsUnigram.sort()
countsUnigram.reverse()
```

In [163]:
```
def feature_000(datum):
    feat = [0] * len(unigrams)
    unigramCount_rt = defaultdict(int)
    r = ''.join([c for c in datum['review/text'].lower() if not c in punct
    for w in r.split():
        unigramCount_rt[w] += 1
    for unigram in unigramId:
        if unigram in unigramCount_rt:
            feat[unigramId[unigram]] = unigramCount_rt[unigram] *  log(150
    feat.append(1) #offset
    return feat
```

In [164]:
```
X_train = [feature_000(d) for d in data_train]
y_train = [d['review/overall'] for d in data_train]
X_valid = [feature_000(d) for d in data_valid]
y_valid = [d['review/overall'] for d in data_valid]
X_test = [feature_000(d) for d in data_test]
y_test = [d['review/overall'] for d in data_test]
```

In [165]:
```
def compare_unigrams(reg):
    clf = linear_model.Ridge(reg, fit_intercept=False)
    clf.fit(X_train, y_train)
    theta = clf.coef_
    predictions = clf.predict(X_valid)
    predictions = np.matrix(predictions)
    y_validation = np.matrix(y_valid)
    diff = y_validation - predictions
    MSE = diff * diff.T / 5000
    print("reg for unigrams= ", r, "MSE = ", MSE)
```

In [166]:
```
reg = [0.01, 0.1, 1.0, 10, 100]
for r in reg:
    compare_unigrams(r)
```

```
reg for unigrams=  0.01 MSE =  [[0.40196911]]
reg for unigrams=  0.1 MSE =  [[0.39984351]]
reg for unigrams=  1.0 MSE =  [[0.39184914]]
reg for unigrams=  10 MSE =  [[0.37368687]]
reg for unigrams=  100 MSE =  [[0.38978366]]
```

In [167]:
```
clf = linear_model.Ridge(10, fit_intercept=False)
clf.fit(X_train, y_train)
theta = clf.coef_
predictions = clf.predict(X_test)
predictions = np.matrix(predictions)
y_testing = np.matrix(y_test)
diff = y_testing - predictions
MSE = diff * diff.T / 5000
print("optimal lambda = 10, testing MSE = " + str(MSE))
```

```
optimal lambda = 10, testing MSE = [[0.38394586]]
```

2. unigrams, removing punctuation, word counts

In [168]:

```
punctuation = set(string.punctuation)
reviewText = [''.join([c for c in datum['review/text'].lower() if not c in
```

In [169]:

```
unigramCount = defaultdict(int)
for d in data:
    r = ''.join([c for c in d['review/text'].lower() if not c in punctuati
    for w in r.split():
        unigramCount[w] += 1
```

In [170]:

```
countsUnigram = [(unigramCount[w], w) for w in unigramCount]
unigrams = [x[1] for x in countsUnigram[:1000]]
unigramId = dict(zip(unigrams, range(len(unigrams))))
unigramSet = set(unigrams)
```

In [171]:

```
countsUnigram.sort()
countsUnigram.reverse()
```

In [172]:

```
def feature_001(datum):
    feat = [0] * len(unigrams)
    unigramCount_rt = defaultdict(int)
    r = ''.join([c for c in datum['review/text'].lower() if not c in punct
    for w in r.split():
        if w in unigrams:
            feat[unigramId[w]] += 1
    feat.append(1) #offset
    return feat
```

In [173]:

```
X_train = [feature_001(d) for d in data_train]
y_train = [d['review/overall'] for d in data_train]
X_valid = [feature_001(d) for d in data_valid]
y_valid = [d['review/overall'] for d in data_valid]
X_test = [feature_001(d) for d in data_test]
y_test = [d['review/overall'] for d in data_test]
```

In [174]:

```
def compare_unigrams(reg):
    clf = linear_model.Ridge(reg, fit_intercept=False)
    clf.fit(X_train, y_train)
    theta = clf.coef_
    predictions = clf.predict(X_valid)
    predictions = np.matrix(predictions)
    y_validation = np.matrix(y_valid)
    diff = y_validation - predictions
    MSE = diff * diff.T / 5000
    print("reg for unigrams= ", r, "MSE = ", MSE)
```

In [175]:
```
reg = [0.01, 0.1, 1.0, 10, 100]
for r in reg:
    compare_unigrams(r)
```

```
reg for unigrams=  0.01 MSE =  [[0.40064272]]
reg for unigrams=  0.1 MSE =  [[0.3948727]]
reg for unigrams=  1.0 MSE =  [[0.38137263]]
reg for unigrams=  10 MSE =  [[0.36343046]]
reg for unigrams=  100 MSE =  [[0.39376672]]
```

In [176]:
```
clf = linear_model.Ridge(10, fit_intercept=False)
clf.fit(X_train, y_train)
theta = clf.coef_
predictions = clf.predict(X_test)
predictions = np.matrix(predictions)
y_testing = np.matrix(y_test)
diff = y_testing - predictions
MSE = diff * diff.T / 5000
print("optimal lambda = 10, testing MSE = " + str(MSE))
```

```
optimal lambda = 10, testing MSE = [[0.37596804]]
```

3. unigrams, preserving punctuation, tfidf

In [177]:
```
punctuation = set(string.punctuation)
reviewText = [''.join([c for c in datum['review/text'].lower() if not c in
```

In [178]:
```
unigramCount = defaultdict(int)
for d in data:
    unigramList = []
    for c in d['review/text'].lower():
        if c not in punctuation:
            unigramList.append(c)
        else:
            unigramList.append(" ")
            unigramList.append(c)
    r = ''.join(unigramList)
    for w in r.split():
        unigramCount[w] += 1
```

In [179]:
```
countsUnigram = [(unigramCount[w], w) for w in unigramCount]
unigrams = [x[1] for x in countsUnigram[:1000]]
unigramId = dict(zip(unigrams, range(len(unigrams))))
unigramSet = set(unigrams)
```

In [180]:
```
countsUnigram.sort()
countsUnigram.reverse()
```

In [181]:

```
def feature_010(datum):
    feat = [0] * len(unigrams)
    unigramCount_datum = defaultdict(int)

    unigramList = []
    for c in datum['review/text'].lower():
        if c not in punctuation:
            unigramList.append(c)
        else:
            unigramList.append(" ")
            unigramList.append(c)
    r = ''.join(unigramList)

    for w in r.split():
        unigramCount_datum[w] += 1

    for unigram in unigramId:
        if unigram in unigramCount_datum:
            feat[unigramId[unigram]] = unigramCount_datum[unigram] * log(1

    feat.append(1) #offset
    return feat
```

In [182]:

```
X_train = [feature_010(d) for d in data_train]
y_train = [d['review/overall'] for d in data_train]
X_valid = [feature_010(d) for d in data_valid]
y_valid = [d['review/overall'] for d in data_valid]
X_test = [feature_010(d) for d in data_test]
y_test = [d['review/overall'] for d in data_test]
```

In [183]:

```
def compare_unigrams(reg):
    clf = linear_model.Ridge(reg, fit_intercept=False)
    clf.fit(X_train, y_train)
    theta = clf.coef_
    predictions = clf.predict(X_valid)
    predictions = np.matrix(predictions)
    y_validation = np.matrix(y_valid)
    diff = y_validation - predictions
    MSE = diff * diff.T / 5000
    print("reg for unigrams= ", r, "MSE = ", MSE)
```

In [184]:

```
reg = [0.01, 0.1, 1.0, 10, 100]
for r in reg:
    compare_unigrams(r)
```

```
reg for unigrams=  0.01 MSE =  [[0.40508638]]
reg for unigrams=  0.1 MSE =  [[0.40344859]]
reg for unigrams=  1.0 MSE =  [[0.39609762]]
reg for unigrams=  10 MSE =  [[0.37728858]]
reg for unigrams=  100 MSE =  [[0.39379484]]
```

In [185]:

```
clf = linear_model.Ridge(10, fit_intercept=False)
clf.fit(X_train, y_train)
theta = clf.coef_
predictions = clf.predict(X_test)
predictions = np.matrix(predictions)
y_testing = np.matrix(y_test)
diff = y_testing - predictions
MSE = diff * diff.T / 5000
print("optimal lambda = 10, testing MSE = " + str(MSE))
```

```
optimal lambda = 10, testing MSE = [[0.38785107]]
```

4. unigrams, preserving punctuation, word counts

In [186]:

```
punctuation = set(string.punctuation)
reviewText = [''.join([c for c in datum['review/text'].lower() if not c in
```

In [187]:

```
unigramCount = defaultdict(int)
for d in data:
    unigramList = []
    for c in d['review/text'].lower():
        if c not in punctuation:
            unigramList.append(c)
        else:
            unigramList.append(" ")
            unigramList.append(c)
    r = ''.join(unigramList)
    for w in r.split():
        unigramCount[w] += 1
```

In [188]:

```
countsUnigram = [(unigramCount[w], w) for w in unigramCount]
unigrams = [x[1] for x in countsUnigram[:1000]]
unigramId = dict(zip(unigrams, range(len(unigrams))))
unigramSet = set(unigrams)
```

In [189]:

```
countsUnigram.sort()
countsUnigram.reverse()
```

In [190]:

```
def feature_011(datum):
    feat = [0] * len(unigrams)
    #unigramCount_rt = defaultdict(int)
    r = ''.join([c for c in datum['review/text'].lower() if not c in punct
    for w in r.split():
        if w in unigrams:
            feat[unigramId[w]] += 1
    feat.append(1) #offset
    return feat
```

In [191]:

```
X_train = [feature_010(d) for d in data_train]
y_train = [d['review/overall'] for d in data_train]
X_valid = [feature_010(d) for d in data_valid]
y_valid = [d['review/overall'] for d in data_valid]
X_test = [feature_010(d) for d in data_test]
y_test = [d['review/overall'] for d in data_test]
```

In [192]:

```
def compare_unigrams(reg):
    clf = linear_model.Ridge(reg, fit_intercept=False)
    clf.fit(X_train, y_train)
    theta = clf.coef_
    predictions = clf.predict(X_valid)
    predictions = np.matrix(predictions)
    y_validation = np.matrix(y_valid)
    diff = y_validation - predictions
    MSE = diff * diff.T / 5000
    print("reg for unigrams= ", r, "MSE = ", MSE)
```

In [193]:

```
reg = [0.01, 0.1, 1.0, 10, 100]
for r in reg:
    compare_unigrams(r)
```

```
reg for unigrams=  0.01 MSE =  [[0.40508638]]
reg for unigrams=  0.1 MSE =  [[0.40344859]]
reg for unigrams=  1.0 MSE =  [[0.39609762]]
reg for unigrams=  10 MSE =  [[0.37728858]]
reg for unigrams=  100 MSE =  [[0.39379484]]
```

In [194]:

```
clf = linear_model.Ridge(10, fit_intercept=False)
clf.fit(X_train, y_train)
theta = clf.coef_
predictions = clf.predict(X_test)
predictions = np.matrix(predictions)
y_testing = np.matrix(y_test)
diff = y_testing - predictions
MSE = diff * diff.T / 5000
print("optimal lambda = 10, testing MSE = " + str(MSE))
```

```
optimal lambda = 10, testing MSE = [[0.38785107]]
```

5. bigrams, removing punctuation, tfidf

In [195]:

```
bigramCount = defaultdict(int)
punctuation = set(string.punctuation)
#stemmer = PorterStemmer()
for d in data:
    r = ''.join([c for c in d['review/text'].lower() if not c in punctuati
    words = r.split()
    for i in range(0,len(words)-1):
        bigram = words[i] + " " + words[i+1]
        bigramCount[bigram] += 1
```

In [196]:

```
countsBigram = [(bigramCount[d], d) for d in bigramCount.keys()]
countsBigram.sort()
countsBigram.reverse()
```

In [197]:

```
bigrams = [c[1] for c in countsBigram[:1000]]
bigramId = dict(zip(bigrams, range(len(bigrams))))
bigramSet = set(bigrams)
```

In [198]:

```
def feature_100(datum):
    feat = [0] * len(bigrams)
    bigramCount_datum = defaultdict(int)
    r = ''.join([c for c in datum['review/text'].lower() if not c in punct
    words = r.split()
    for i in range(len(words) - 1):
        bigram = words[i] + " " + words[i + 1]
        bigramCount_datum[bigram] += 1

    for bigram in bigramId:
        if bigram in bigramCount_datum:
            feat[bigramId[bigram]] = bigramCount_datum[bigram] * log(15000

    feat.append(1) #offset
    return feat
```

In [199]:

```
X_train = [feature_100(d) for d in data_train]
y_train = [d['review/overall'] for d in data_train]
X_valid = [feature_100(d) for d in data_valid]
y_valid = [d['review/overall'] for d in data_valid]
X_test = [feature_100(d) for d in data_test]
y_test = [d['review/overall'] for d in data_test]
```

In [200]:

```
def compare_bigrams(reg):
    clf = linear_model.Ridge(reg, fit_intercept=False)
    clf.fit(X_train, y_train)
    theta = clf.coef_
    predictions = clf.predict(X_valid)
    predictions = np.matrix(predictions)
    y_validation = np.matrix(y_valid)
    diff = y_validation - predictions
    MSE = diff * diff.T / 5000
    print("reg for bigrams= ", r, "MSE = ", MSE)
```

In [201]:

```
reg = [0.01, 0.1, 1.0, 10, 100]
for r in reg:
    compare_bigrams(r)
```

```
reg for bigrams=  0.01 MSE =  [[0.44638918]]
reg for bigrams=  0.1 MSE =  [[0.44612483]]
reg for bigrams=  1.0 MSE =  [[0.44349019]]
reg for bigrams=  10 MSE =  [[0.42654711]]
reg for bigrams=  100 MSE =  [[0.4132646]]
```

In [202]:

```
clf = linear_model.Ridge(100, fit_intercept=False)
clf.fit(X_train, y_train)
theta = clf.coef_
predictions = clf.predict(X_test)
predictions = np.matrix(predictions)
y_testing = np.matrix(y_test)
diff = y_testing - predictions
MSE = diff * diff.T / 5000
print("optimal lambda = 100, testing MSE = " + str(MSE))
```

```
optimal lambda = 100, testing MSE = [[0.42533635]]
```

6. bigrams, removing punctuation, word counts

In [203]:

```
bigramCount = defaultdict(int)
punctuation = set(string.punctuation)
#stemmer = PorterStemmer()
for d in data:
    r = ''.join([c for c in d['review/text'].lower() if not c in punctuati
    words = r.split()
    for i in range(0,len(words)-1):
        bigram = words[i] + " " + words[i+1]
        bigramCount[bigram] += 1
```

In [204]:

```
countsBigram = [(bigramCount[d], d) for d in bigramCount.keys()]
countsBigram.sort()
countsBigram.reverse()
```

In [205]:

```
bigrams = [c[1] for c in countsBigram[:1000]]
bigramId = dict(zip(bigrams, range(len(bigrams))))
bigramSet = set(bigrams)
```

In [206]:

```
def feature_101(datum):
    feat = [0]*len(bigrams)
    r = ''.join([c for c in datum['review/text'].lower() if not c in punct
    words = r.split()
    for i in range(len(words)-1):
        bigram = words[i] + " " + words[i + 1]
        if bigram in bigrams:
            feat[bigramId[bigram]] += 1
    feat.append(1) #offset
    return feat
```

In [207]:

```
X_train = [feature_101(d) for d in data_train]
y_train = [d['review/overall'] for d in data_train]
X_valid = [feature_101(d) for d in data_valid]
y_valid = [d['review/overall'] for d in data_valid]
X_test = [feature_101(d) for d in data_test]
y_test = [d['review/overall'] for d in data_test]
```

In [208]:

```
def compare_bigrams(reg):
    clf = linear_model.Ridge(reg, fit_intercept=False)
    clf.fit(X_train, y_train)
    theta = clf.coef_
    predictions = clf.predict(X_valid)
    predictions = np.matrix(predictions)
    y_validation = np.matrix(y_valid)
    diff = y_validation - predictions
    MSE = diff * diff.T / 5000
    print("reg for bigrams= ", r, "MSE = ", MSE)
```

In [209]:

```
reg = [0.01, 0.1, 1.0, 10, 100]
for r in reg:
    compare_bigrams(r)
```

```
reg for bigrams=  0.01 MSE =  [[0.44597822]]
reg for bigrams=  0.1 MSE =  [[0.44569662]]
reg for bigrams=  1.0 MSE =  [[0.44299749]]
reg for bigrams=  10 MSE =  [[0.42425457]]
reg for bigrams=  100 MSE =  [[0.41755767]]
```

In [210]:

```
clf = linear_model.Ridge(100, fit_intercept=False)
clf.fit(X_train, y_train)
theta = clf.coef_
predictions = clf.predict(X_test)
predictions = np.matrix(predictions)
y_testing = np.matrix(y_test)
diff = y_testing - predictions
MSE = diff * diff.T / 5000
print("optimal lambda = 100, testing MSE = " + str(MSE))
```

```
optimal lambda = 100, testing MSE = [[0.43058102]]
```

7. bigrams, preserving punctuation, tfidf

In [211]:

```
punctuation = set(string.punctuation)
reviewText = [''.join([c for c in datum['review/text'].lower() if not c in
```

In [212]:

```
bigramCount = defaultdict(int)
for d in data:
    bigramList = []
    for c in d['review/text'].lower():
        if c not in punctuation:
            bigramList.append(c)
        else:
            bigramList.append(" ")
            bigramList.append(c)
    r = ''.join(bigramList)
    words = r.split()
    for i in range(len(words) - 1):
        bigram = words[i] + " " + words[i + 1]
        bigramCount[bigram] += 1
```

In [213]:

```
countsBigram = [(bigramCount[d], d) for d in bigramCount.keys()]
countsBigram.sort()
countsBigram.reverse()
```

In [214]:

```
bigrams = [c[1] for c in countsBigram[:1000]]
bigramId = dict(zip(bigrams, range(len(bigrams))))
bigramSet = set(bigrams)
```

In [215]:

```
def feature_110(datum):
    feat = [0] * len(bigrams)
    bigramCount_datum = defaultdict(int)

    bigramList = []
    for c in datum['review/text'].lower():
        if c not in punctuation:
            bigramList.append(c)
        else:
            bigramList.append(" ")
            bigramList.append(c)
    r = ''.join(bigramList)
    words = r.split()
    for i in range(len(words) - 1):
        bigram = words[i] + " " + words[i + 1]
        bigramCount_datum[bigram] += 1

    for bigram in bigramId:
        if bigram in bigramCount_datum:
            feat[bigramId[bigram]] = bigramCount_datum[bigram] * log(15000

    feat.append(1) #offset
    return feat
```

In [216]:

```
X_train = [feature_110(d) for d in data_train]
y_train = [d['review/overall'] for d in data_train]
X_valid = [feature_110(d) for d in data_valid]
y_valid = [d['review/overall'] for d in data_valid]
X_test = [feature_110(d) for d in data_test]
y_test = [d['review/overall'] for d in data_test]
```

In [217]:

```
def compare_bigrams(reg):
    clf = linear_model.Ridge(reg, fit_intercept=False)
    clf.fit(X_train, y_train)
    theta = clf.coef_
    predictions = clf.predict(X_valid)
    predictions = np.matrix(predictions)
    y_validation = np.matrix(y_valid)
    diff = y_validation - predictions
    MSE = diff * diff.T / 5000
    print("reg for bigrams= ", r, "MSE = ", MSE)
```

In [218]:

```
reg = [0.01, 0.1, 1.0, 10, 100]
for r in reg:
    compare_bigrams(r)
```

```
reg for bigrams=  0.01 MSE =  [[0.44922545]]
reg for bigrams=  0.1 MSE =  [[0.44880617]]
reg for bigrams=  1.0 MSE =  [[0.44514748]]
reg for bigrams=  10 MSE =  [[0.42573408]]
reg for bigrams=  100 MSE =  [[0.41411603]]
```

In [219]:

```
clf = linear_model.Ridge(100, fit_intercept=False)
clf.fit(X_train, y_train)
theta = clf.coef_
predictions = clf.predict(X_test)
predictions = np.matrix(predictions)
y_testing = np.matrix(y_test)
diff = y_testing - predictions
MSE = diff * diff.T / 5000
print("optimal lambda = 100, testing MSE = " + str(MSE))
```

```
optimal lambda = 100, testing MSE = [[0.4299674]]
```

8. bigrams, preserving punctuation, word counts

In [220]:

```
punctuation = set(string.punctuation)
reviewText = [''.join([c for c in datum['review/text'].lower() if not c in
```

In [221]:

```
bigramCount = defaultdict(int)
for d in data:
    bigramList = []
    for c in d['review/text'].lower():
        if c not in punctuation:
            bigramList.append(c)
        else:
            bigramList.append(" ")
            bigramList.append(c)
    r = ''.join(bigramList)
    words = r.split()
    for i in range(len(words) - 1):
        bigram = words[i] + " " + words[i + 1]
        bigramCount[bigram] += 1
```

In [222]:

```
countsBigram = [(bigramCount[d], d) for d in bigramCount.keys()]
countsBigram.sort()
countsBigram.reverse()
```

In [223]:

```
bigrams = [c[1] for c in countsBigram[:1000]]
bigramId = dict(zip(bigrams, range(len(bigrams))))
bigramSet = set(bigrams)
```

In [224]:
```
def feature_111(datum):
    feat = [0] * len(bigrams)
    # bigramCount_datum = defaultdict(int)

    bigramList = []
    for c in datum['review/text'].lower():
        if c not in punctuation:
            bigramList.append(c)
        else:
            bigramList.append(" ")
            bigramList.append(c)
    r = ''.join(bigramList)
    words = r.split()
    for i in range(len(words) - 1):
        bigram = words[i] + " " + words[i + 1]
        if bigram in bigrams:
            feat[bigramId[bigram]] += 1

    feat.append(1) #offset
    return feat
```

In [225]:
```
X_train = [feature_111(d) for d in data_train]
y_train = [d['review/overall'] for d in data_train]
X_valid = [feature_111(d) for d in data_valid]
y_valid = [d['review/overall'] for d in data_valid]
X_test = [feature_111(d) for d in data_test]
y_test = [d['review/overall'] for d in data_test]
```

In [226]:
```
def compare_bigrams(reg):
    clf = linear_model.Ridge(reg, fit_intercept=False)
    clf.fit(X_train, y_train)
    theta = clf.coef_
    predictions = clf.predict(X_valid)
    predictions = np.matrix(predictions)
    y_validation = np.matrix(y_valid)
    diff = y_validation - predictions
    MSE = diff * diff.T / 5000
    print("reg for bigrams= ", r, "MSE = ", MSE)
```

In [227]:
```
reg = [0.01, 0.1, 1.0, 10, 100]
for r in reg:
    compare_bigrams(r)
```

```
reg for bigrams=  0.01 MSE =  [[0.44906436]]
reg for bigrams=  0.1 MSE =  [[0.4487985]]
reg for bigrams=  1.0 MSE =  [[0.44627841]]
reg for bigrams=  10 MSE =  [[0.42936252]]
reg for bigrams=  100 MSE =  [[0.42557995]]
```

In [228]:

```
clf = linear_model.Ridge(100, fit_intercept=False)
clf.fit(X_train, y_train)
theta = clf.coef_
predictions = clf.predict(X_test)
predictions = np.matrix(predictions)
y_testing = np.matrix(y_test)
diff = y_testing - predictions
MSE = diff * diff.T / 5000
print("optimal lambda = 100, testing MSE = " + str(MSE))
```

```
optimal lambda = 100, testing MSE = [[0.44222335]]
```

MSE

Unigrams/Bigrams	**Removing punctuation/Preserve punctuation**	**tfidf/word counts**	**Optimal lambda for regression**	**MSE for test set**
Unigrams	remove	tfidf	10	0.38394586
Unigrams	remove	Word counts	10	0.37596804
Unigrams	preserve	tfidf	10	0.38785107
Unigrams	preserve	Word counts	10	0.38785107
Bigrams	remove	tfidf	100	0.42533635
Bigrams	remove	Word counts	100	0.43058102
Bigrams	Preserve	tfidf	100	0.4299674
Bigrams	Preserve	Word counts	100	0.44222335