In [1]:
```
import numpy
import urllib
import scipy.optimize
import random
from collections import defaultdict
import nltk
import string
from nltk.stem.porter import *
from sklearn import linear_model
nltk.download("stopwords")
def parseData(fname):
  for l in urllib.urlopen(fname):
    yield eval(l)
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\xinme\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping corpora\stopwords.zip.
```

In [408]:
```
### Just the first 5000 reviews

print "Reading data..."
data = list(parseData("http://jmcauley.ucsd.edu/cse190/data/beer/beer_50000.json"))[:5000]
print "done"
```

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

Q1

In [409]:
```
### Ignore capitalization and remove punctuation
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 [d for d in nltk.bigrams(r.split())]:
        wordCount[w] += 1
# unique bigrams numbers
print('number of unique bigrams amongst all of the reviews')
print(len(wordCount))
```

```
number of unique bigrams amongst all of the reviews
182246
```

In [410]:
```
# take the 5 most popular words
counts = [(wordCount[w], w) for w in wordCount]
counts.sort()
counts.reverse()
print('the 5 most-frequently-occurring bigrams')
print(counts[:5])
```

```
the 5 most-frequently-occurring bigrams
[(4587, ('with', 'a')), (2595, ('in', 'the')), (2245, ('of', 'the')), (2056,
('is', 'a')), (2033, ('on', 'the'))]
```

Q2

In [411]:
```
# Use the most popular 1000 words to save computation time
words = [x[1] for x in counts[:1000]]
```

In [412]:
```
### Sentiment analysis
wordId = dict(zip(words, range(len(words))))
wordSet = set(words)
def feature(datum):
    feat = [0]*len(words)
    r = ''.join([c for c in datum['review/text'].lower() if not c in punctuati
on])
    for w in [d for d in nltk.bigrams(r.split())]:
        if w in words:
            feat[wordId[w]] += 1
    feat.append(1) #offset
    return feat

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

In [413]:
```
#With regularization
clf = linear_model.Ridge(1.0, fit_intercept=False)
clf.fit(X, y)
theta = clf.coef_
predictions = clf.predict(X)
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y,predictions)
```

In [414]:
```
print('mse',mse)
```

```
('mse', 0.34315301406136334)
```

Q3

In [406]:
```
### Ignore capitalization and remove punctuation
uniwordCount ={'foam':0, 'smell':0, 'banana':0, 'lactic':0 ,'tart':0}
punctuation = set(string.punctuation)
five = ['foam', 'smell', 'banana', 'lactic' ,'tart']
for w in five:
    for d in data:
        r = ''.join([c for c in d['review/text'].lower() if not c in punctuati
on])
        if w in r.split():
            uniwordCount[w] += 1
print(uniwordCount)
```

```
{'foam': 364, 'smell': 1449, 'banana': 105, 'lactic': 6, 'tart': 78}
```

In [287]:
```
# Check that all of 5000 reviews have 'review/text'
len([d for d in data if 'review/text' in d])
```

Out[287]:
```
5000
```

In [404]:
```
import math
idf =  {'foam':0, 'smell':0, 'banana':0, 'lactic':0 ,'tart':0}
for w in five:
    idf[w] = math.log10(len(data)/float(uniwordCount[w]))
print('idf scores')
print(idf)
```

```
idf scores
{'foam': 1.1378686206869628, 'smell': 0.5379016188648442, 'banana': 1.6777807
052660807, 'lactic': 2.9208187539523753, 'tart': 1.8068754016455384}
```

In [403]:
```
tf = {'foam':0, 'smell':0, 'banana':0, 'lactic':0 ,'tart':0}
r = ''.join([c for c in data[0]['review/text'].lower() if not c in punctuation
])
for w in r.split():
        if w in five:
            tf[w] += 1
print('tf value')
print(tf)
```

```
tf value
{'foam': 2, 'smell': 1, 'banana': 2, 'lactic': 2, 'tart': 1}
```

In [405]:
```
tfidf = {'foam':0, 'smell':0, 'banana':0, 'lactic':0 ,'tart':0}
for w in five:
    tfidf[w] = idf[w]*tf[w]
print('tfidf scores')
print(tfidf)
```

```
tfidf scores
{'foam': 2.2757372413739256, 'smell': 0.5379016188648442, 'banana': 3.3555614
105321614, 'lactic': 5.841637507904751, 'tart': 1.8068754016455384}
```

Q4,Q5

In [434]:
```
worduni = 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():
        if w not in worduni:
            worduni[w] = 1
        else:
            worduni[w] += 1
print(len(worduni))
# take the 5 most popular words
countuni = [(worduni[w], w) for w in worduni]
countuni.sort()
countuni.reverse()
print(countuni[:5])
```

```
19426
[(30695, 'a'), (27569, 'the'), (19512, 'and'), (15935, 'of'), (12623, 'is')]
```

In [435]:
```
# Use the most popular 1000 words to save computation time
words1000 = [x[1] for x in countuni[:1000]]
wordId1000 = dict(zip(words1000, range(len(words1000))))
reviewtotal2 = [(''.join([c for c in d['review/text'].lower() if c not in punc
tuation])).split() for d in data]
```

In [333]:
```
# Compute number of documents where the term t appears
idfdict = defaultdict(int)
for w in words1000:
    idfdict[w] = 0
    for r in reviewtotal2:
        if w in set(r):
            idfdict[w] += 1
# Compute idf for all (t,D)
idflog = defaultdict(float)
for w in words1000:
    # To avoid zero division, we adjust the denominator by adding one
    idflog[w] = math.log10(len(data)/float(1+idfdict[w]))
X = numpy.array([[0.0]*len(words1000)]*len(data))
# Compute tf for all (t,d)
for i in range(len(data)):
    for w in reviewtotal2[i]:
        if w in words1000:
            X[i][wordId1000[w]] += 1
# Compute tfidf for all (t,d,D)
for i in range(len(data)):
    for w in words1000:
        if X[i][wordId1000[w]] > 0:
            X[i][wordId1000[w]] = float(X[i][wordId1000[w]])*idflog[w]
```

In [338]:
```
# Q4
cosA = X[0]
cosB = X[1]
cosim = dot(cosA, cosB)/(norm(cosA)*norm(cosB))
print('The cosine similarity between the first and the second review')
print(cosim)
# Q5
cossimilarity = [0]*len(data)
for i in range(1,len(data)):
    # To avoid zero division, we set the similarity to 0 if the norm is 0
    n = float(norm(X[i]))
    if n > 0 :
        cossimilarity[i] = dot(X[0], X[i])/float(norm(X[0])*n)
maxcos = cossimilarity.index(max(cossimilarity))
print('The highest cosine similarity compared to the first review is ')
print(max(cossimilarity))
print(data[maxcos])
```

```
The cosine similarity between the first and the second review
0.10629834153948434
The highest cosine similarity compared to the first review is
0.3171149222428099
{'beer/style': 'Pumpkin Ale', 'beer/ABV': 8.4, 'beer/beerId': '52211', 'revie
w/timeStruct': {'wday': 0, 'isdst': 0, 'mday': 14, 'hour': 0, 'min': 24, 'se
c': 50, 'year': 2011, 'yday': 318, 'mon': 11}, 'review/aroma': 5.0, 'review/a
ppearance': 4.0, 'review/timeUnix': 1321230290, 'review/palate': 3.5, 'revie
w/taste': 4.0, 'beer/name': "Frog's Hollow Double Pumpkin Ale", 'beer/brewerI
d': '14879', 'review/overall': 4.0, 'review/text': 'Poured from a 22oz bottle
to a Dogfish Head Snifter.\t\tColor: Slight hazy orange with an off white hea
d.\t\tSmell: Cinnamon, banana, pumpkin and nutmeg.\t\tTaste: Alcohol, pumpki
n, nutmeg, allspice and a hint of banana.\t\tMouthfeel: Medium carbonation, s
mooth, medium dryness on the palate.\t\tOverall: The smell is GREAT! The bana
na was a huge surprise for me. The taste had too much alcohol presence. Seeme
d to overpower the other flavors. Cheers!', 'user/profileName': 'Heatwave33'}
```

Q6

In [436]:
```
# We use the most popular 1000 words due to computation limit
wordss = words1000
```

In [455]:
```
wordssId = dict(zip(wordss, range(len(wordss))))
idfdict2 = defaultdict(int)
for w in wordss:
    idfdict2[w] = 0
    for r in reviewtotal2:
        if w in set(r):
            idfdict2[w] += 1
idflog2 = defaultdict(float)
# Compute idf
for w in wordss:
    # To avoid zero division, we adjust the denominator by adding one (the met
hod suggested in Wiki)
    idflog2[w] = math.log10(len(data)/float(idfdict2[w]+1))
```

In [458]:
```
# Compute tf
X2 = numpy.array([[0.0]*(len(wordss)+1)]*len(data))
for i in range(len(data)):
    for w in reviewtotal2[i]:
        if w in wordss:
            X2[i][wordssId[w]] += 1
```

In [459]:
```
# Compute tfidf
for i in range(len(data)):
    for w in wordss:
        if X2[i][wordssId[w]]>0:
            X2[i][wordssId[w]] = float(X2[i][wordssId[w]])*idflog2[w]
```

In [460]:
```
# Offset
for i in range(len(data)):
    X2[i][len(wordss)]= 1
```

In [461]:
```
#With regularization
clf2 = linear_model.Ridge(1.0, fit_intercept=False)
clf2.fit(X2, y)
theta2 = clf2.coef_
predictions2 = clf2.predict(X2)
mse2 = mean_squared_error(y,predictions2)
print(mse2)
```

```
0.278759714116527
```

Q7

In [203]:
```
print "Reading data..."
datatotal = list(parseData("http://jmcauley.ucsd.edu/cse190/data/beer/beer_50000.json"))
print "done"
```

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

In [204]:
```
import random
train = list()
validation = list()
test = list()
for i in range(5000):
    train.append(random.choice(datatotal))
    validation.append(random.choice(datatotal))
    test.append(random.choice(datatotal))
print(len(train),len(validation),len(test))
```

```
(5000, 5000, 5000)
```

I built 8x3 feature vectors for 8 models and train,validation,test set.

As I looked up online and some documents says that log e is the general base of tfidf. I used log e instead of 10 in this question

In [346]:
```
# Train feature vector for no punctuation, unigram
reviewtotalnop= [(''.join([c for c in d['review/text'].lower() if c not in pun
ctuation])).split() for d in train]
# Unigram Remove punctuation
wordnop = defaultdict(int)
for d in reviewtotalnop:
    for w in d:
        wordnop[w] += 1
print(len(wordnop))
# take the 5 most popular words
countnop = [(wordnop[w], w) for w in wordnop]
countnop.sort()
countnop.reverse()
print(countnop[:5])
# Use the most popular 1000 words to save computation time
wordsnnop = [x[1] for x in countnop[:1000]]
wordIdnop = dict(zip(wordsnnop, range(len(wordsnnop))))
# Feature vector for word counts
featnop=numpy.array([[0]*(len(wordsnnop)+1)]*len(train))
for i in range(len(train)):
    for w in reviewtotalnop[i]:
        if w in wordsnnop:
            featnop[i][wordIdnop[w]] += 1

for i in range(len(train)):
    featnop[i][len(wordsnnop)]= 1
```

```
19573
[(31026, 'a'), (27994, 'the'), (19536, 'and'), (16377, 'of'), (12952, 'is')]
```

In [462]:
```
# Feature vector for tfidf
idfdictnop = defaultdict(int)
for w in wordsnnop:
    idfdictnop[w] = 0
    for r in reviewtotalnop:
        if w in set(r):
            idfdictnop[w] += 1
idflognop = defaultdict(float)
# Compute idf
for w in wordsnnop:
    # To avoid zero division, we adjust the denominator by adding one
    idflognop[w] = math.log(len(train)/float(1+idfdictnop[w]))
Xnop = numpy.array([[0.0]*(len(wordsnnop)+1)]*len(train))
# Compute tfidf
for i in range(len(train)):
    for w in wordsnnop:
        Xnop[i][wordIdnop[w]] = float(featnop[i][wordIdnop[w]])*idflognop[w]
# Offset
for i in range(len(train)):
    Xnop[i][len(wordsnnop)]= 1
```

In [240]:
```
# Validation feature vector for no punctuation, unigram
reviewtotalnopval= [(''.join([c for c in d['review/text'].lower() if c not in
punctuation])).split() for d in validation]

# Feature vector for word counts
featnopval=numpy.array([[0]*(len(wordsnnop)+1)]*len(validation))
for i in range(len(validation)):
    for w in reviewtotalnopval[i]:
        if w in wordsnnop:
            featnopval[i][wordIdnop[w]] += 1
for i in range(len(validation)):
    featnopval[i][len(wordsnnop)]= 1
```

In [463]:
```
# Feature vector for tfidf
idfdictnop = defaultdict(int)
for w in wordsnnop:
    idfdictnop[w] = 0
    for r in reviewtotalnopval:
        if w in set(r):
            idfdictnop[w] += 1
idflognop = defaultdict(float)
# Compute idf
for w in wordsnnop:
    # To avoid zero division, we adjust the denominator by adding one
    idflognop[w] = math.log(len(validation)/float(1+idfdictnop[w]))
Xnopval = numpy.array([[0.0]*(len(wordsnnop)+1)]*len(validation))
# Compute tfidf
for i in range(len(validation)):
    for w in wordsnnop:
        if w in reviewtotalnopval[i]:
            Xnopval[i][wordIdnop[w]] = float(featnopval[i][wordIdnop[w]])*idfl
ognop[w]
# Offset
for i in range(len(validation)):
    Xnopval[i][len(wordsnnop)]= 1
```

In [241]:
```
# Test feature vector for no punctuation, unigram
reviewtotalnoptest= [(''.join([c for c in d['review/text'].lower() if c not in
 punctuation])).split() for d in test]

# Feature vector for word counts
featnoptest=numpy.array([[0]*(len(wordsnnop)+1)]*len(test))
for i in range(len(test)):
    for w in reviewtotalnoptest[i]:
        if w in wordsnnop:
            featnoptest[i][wordIdnop[w]] += 1
for i in range(len(test)):
    featnoptest[i][len(wordsnnop)]= 1
```

In [464]:
```
# Feature vector for tfidf
idfdictnop = defaultdict(int)
for w in wordsnnop:
    idfdictnop[w] = 0
    for r in reviewtotalnoptest:
        if w in set(r):
            idfdictnop[w] += 1
idflognop = defaultdict(float)
# Compute idf
for w in wordsnnop:
    # To avoid zero division, we adjust the denominator by adding one
    idflognop[w] = math.log(len(test)/float(1+idfdictnop[w]))
Xnoptest = numpy.array([[0.0]*(len(wordsnnop)+1)]*len(test))
# Compute tfidf
for i in range(len(test)):
    for w in wordsnnop:
        if w in reviewtotalnoptest[i]:
            Xnoptest[i][wordIdnop[w]] = float(featnoptest[i][wordIdnop[w]])*id
flognop[w]
# Offset
for i in range(len(test)):
    Xnoptest[i][len(wordsnnop)]= 1
```

In [242]:
```
# Train feature vector for punctuation, unigram
reviewtotalkep= [(''.join([c for c in d['review/text'].lower() ])).split() for
 d in train]

# Unigram Keep punctuation
wordkep = defaultdict(int)
for d in reviewtotalkep:
    for w in d:
        wordkep[w] += 1
print(len(wordkep))
# take the 5 most popular words
countkep = [(wordkep[w], w) for w in wordkep]
countkep.sort()
countkep.reverse()
print(countkep[:5])
# Use the most popular 1000 words to save computation time
wordskkep = [x[1] for x in countkep[:1000]]
wordIdkep = dict(zip(wordskkep, range(len(wordskkep))))

# Feature vector for word counts
featkep=numpy.array([[0]*(len(wordskkep)+1)]*len(train))
for i in range(len(train)):
    for w in reviewtotalkep[i]:
        if w in wordskkep:
            featkep[i][wordIdkep[w]] += 1
for i in range(len(train)):
    featkep[i][len(wordskkep)]= 1
```

```
32133
[(30131, 'a'), (27952, 'the'), (19440, 'and'), (16270, 'of'), (12789, 'is')]
```

In [465]:
```
# Feature vector for tfidf
idfdictkep = defaultdict(int)
for w in wordskkep:
    idfdictkep[w] = 0
    for r in reviewtotalkep:
        if w in set(r):
            idfdictkep[w] += 1
idflogkep = defaultdict(float)
# Compute idf
for w in wordskkep:
    # To avoid zero division, we adjust the denominator by adding one
    idflogkep[w] = math.log(len(train)/float(1+idfdictkep[w]))
Xkep = numpy.array([[0.0]*(len(wordskkep)+1)]*len(train))
# Compute tfidf
for i in range(len(train)):
    for w in wordskkep:
        Xkep[i][wordIdkep[w]] = float(featkep[i][wordIdkep[w]])*idflogkep[w]
# Offset
for i in range(len(train)):
    Xkep[i][len(wordskkep)]= 1
```

In [243]:
```
# Validation feature vector for punctuation, unigram
reviewtotalkepval= [(''.join([c for c in d['review/text'].lower() ])).split()
for d in validation]

# Feature vector for word counts
featkepval=numpy.array([[0]*(len(wordskkep)+1)]*len(validation))
for i in range(len(validation)):
    for w in reviewtotalkepval[i]:
        if w in wordskkep:
            featkepval[i][wordIdkep[w]] += 1
for i in range(len(validation)):
    featkepval[i][len(wordskkep)]= 1
```

In [466]:
```
# Feature vector for tfidf
idfdictkep = defaultdict(int)
for w in wordskkep:
    idfdictkep[w] = 0
    for r in reviewtotalkepval:
        if w in set(r):
            idfdictkep[w] += 1
idflogkep = defaultdict(float)
# Compute idf
for w in wordskkep:
    # To avoid zero division, we adjust the denominator by adding one
    idflogkep[w] = math.log(len(validation)/float(1+idfdictkep[w]))
Xkepval = numpy.array([[0.0]*(len(wordskkep)+1)]*len(validation))
# Compute tfidf
for i in range(len(validation)):
    for w in wordskkep:
        Xkepval[i][wordIdkep[w]] = float(featkepval[i][wordIdkep[w]])*idflogke
p[w]
# Offset
for i in range(len(validation)):
    Xkepval[i][len(wordskkep)]= 1
```

In [244]:
```
# Test feature vector for punctuation, unigram
reviewtotalkeptest= [(''.join([c for c in d['review/text'].lower() ])).split()
 for d in test]

# Feature vector for word counts
featkeptest=numpy.array([[0]*(len(wordskkep)+1)]*len(test))
for i in range(len(test)):
    for w in reviewtotalkeptest[i]:
        if w in wordskkep:
            featkeptest[i][wordIdkep[w]] += 1
for i in range(len(test)):
    featkeptest[i][len(wordskkep)]= 1
```

In [467]:
```
# Feature vector for tfidf
idfdictkep = defaultdict(int)
for w in wordskkep:
    idfdictkep[w] = 0
    for r in reviewtotalkeptest:
        if w in set(r):
            idfdictkep[w] += 1
idflogkep = defaultdict(float)
# Compute idf
for w in wordskkep:
    # To avoid zero division, we adjust the denominator by adding one
    idflogkep[w] = math.log(len(test)/float(1+idfdictkep[w]))
Xkeptest = numpy.array([[0.0]*(len(wordskkep)+1)]*len(test))
# Compute tfidf
for i in range(len(test)):
    for w in wordskkep:
        Xkeptest[i][wordIdkep[w]] = float(featkeptest[i][wordIdkep[w]])*idflog
kep[w]
# Offset
for i in range(len(test)):
    Xkeptest[i][len(wordskkep)]= 1
```

In [245]:
```
# Train feature vector for no punctuation, bigram
reviewtotalnopbi= [(''.join([c for c in d['review/text'].lower() if c not in p
unctuation])).split() for d in train]

# Bigram Remove punctuation
wordnopbi = defaultdict(int)
for r in reviewtotalnopbi:
    for w in [d for d in nltk.bigrams(r)]:
        wordnopbi[w] += 1
# unique bigrams numbers
print(len(wordnopbi))
countnopbi = [(wordnopbi[w], w) for w in wordnopbi]
countnopbi.sort()
countnopbi.reverse()
print(countnopbi[:5])
# Use the most popular 1000 words to save computation time
wordsnnopbi = [x[1] for x in countnopbi[:1000]]
wordIdnopbi = dict(zip(wordsnnopbi, range(len(wordsnnopbi))))

# Feature vector for word counts
featnopbi=numpy.array([[0]*(len(wordsnnopbi)+1)]*len(train))
for i in range(len(train)):
    for w in [d for d in nltk.bigrams(reviewtotalnopbi[i])]:
        if w in wordsnnopbi:
            featnopbi[i][wordIdnopbi[w]] += 1
for i in range(len(train)):
    featnopbi[i][len(wordsnnopbi)]= 1
```

```
179327
[(4752, ('with', 'a')), (2658, ('in', 'the')), (2403, ('of', 'the')), (2086,
('is', 'a')), (1989, ('on', 'the'))]
```

In [476]:
```
# Feature vector for tfidf
idfdictnopbi = defaultdict(int)
for w in wordsnnopbi:
    idfdictnopbi[w] = 0
    for r in reviewtotalnopbi:
        if w in set([d for d in nltk.bigrams(r)]):
            idfdictnopbi[w] += 1
idflognopbi = defaultdict(float)
# Compute idf
for w in wordsnnopbi:
    # To avoid zero division, we adjust the denominator by adding one
    idflognopbi[w] = math.log(len(train)/float(1+idfdictnopbi[w]))
Xnopbi = numpy.array([[0.0]*(len(wordsnnopbi)+1)]*len(train))
# Compute tfidf
for i in range(len(train)):
    for w in wordsnnopbi:
        Xnopbi[i][wordIdnopbi[w]] = float(featnopbi[i][wordIdnopbi[w]])*idflog
nopbi[w]
# Offset
for i in range(len(train)):
    Xnopbi[i][len(wordsnnopbi)]= 1
```

In [246]:
```
# Validation feature vector for no punctuation, bigram
reviewtotalnopbival= [(''.join([c for c in d['review/text'].lower() if c not i
n punctuation])).split() for d in validation]

# Feature vector for word counts
featnopbival=numpy.array([[0]*(len(wordsnnopbi)+1)]*len(validation))
for i in range(len(validation)):
    for w in [d for d in nltk.bigrams(reviewtotalnopbival[i])]:
        if w in wordsnnopbi:
            featnopbival[i][wordIdnopbi[w]] += 1
for i in range(len(validation)):
    featnopbival[i][len(wordsnnopbi)]= 1
```

In [ ]:
```
# Feature vector for tfidf
idfdictnopbi = defaultdict(int)
for w in wordsnnopbi:
    idfdictnopbi[w] = 0
    for r in reviewtotalnopbival:
        if w in set([d for d in nltk.bigrams(r)]):
            idfdictnopbi[w] += 1
idflognopbi = defaultdict(float)
# Compute idf
for w in wordsnnopbi:
    # To avoid zero division, we adjust the denominator by adding one
    idflognopbi[w] = math.log(len(validation)/float(1+idfdictnopbi[w]))
Xnopbival = numpy.array([[0.0]*(len(wordsnnopbi)+1)]*len(validation))
# Compute tfidf
for i in range(len(validation)):
    for w in wordsnnopbi:
        Xnopbival[i][wordIdnopbi[w]] = float(featnopbival[i][wordIdnopbi[w]])*
idflognopbi[w]
# Offset
for i in range(len(validation)):
    Xnopbival[i][len(wordsnnopbi)]= 1
```

In [247]:
```
# Test feature vector for no punctuation, bigram
reviewtotalnopbitest= [(''.join([c for c in d['review/text'].lower() if c not
in punctuation])).split() for d in test]

# Feature vector for word counts
featnopbitest=numpy.array([[0]*(len(wordsnnopbi)+1)]*len(test))
for i in range(len(test)):
    for w in [d for d in nltk.bigrams(reviewtotalnopbitest[i])]:
        if w in wordsnnopbi:
            featnopbitest[i][wordIdnopbi[w]] += 1
for i in range(len(test)):
    featnopbitest[i][len(wordsnnopbi)]= 1
```

In [ ]:
```
# Feature vector for tfidf
idfdictnopbi = defaultdict(int)
for w in wordsnnopbi:
    idfdictnopbi[w] = 0
    for r in reviewtotalnopbitest:
        if w in set([d for d in nltk.bigrams(r)]):
            idfdictnopbi[w] += 1
idflognopbi = defaultdict(float)
# Compute idf
for w in wordsnnopbi:
    # To avoid zero division, we adjust the denominator by adding one
    idflognopbi[w] = math.log(len(test)/float(1+idfdictnopbi[w]))
Xnopbitest = numpy.array([[0.0]*(len(wordsnnopbi)+1)]*len(test))
# Compute tfidf
for i in range(len(test)):
    for w in wordsnnopbi:
        Xnopbitest[i][wordIdnopbi[w]] = float(featnopbitest[i][wordIdnopbi[w
]])*idflognopbi[w]
# Offset
for i in range(len(test)):
    Xnopbitest[i][len(wordsnnopbi)]= 1
```

In [248]:
```
# Train feature vector for punctuation, bigram
reviewtotalkepbi= [(''.join([c for c in d['review/text'].lower() ])).split() f
or d in train]

# Bigram Keep punctuation
wordkepbi = defaultdict(int)
for r in reviewtotalkepbi:
    for w in [d for d in nltk.bigrams(r)]:
        wordkepbi[w] += 1
# unique bigrams numbers
print(len(wordkepbi))
countkepbi = [(wordkepbi[w], w) for w in wordkepbi]
countkepbi.sort()
countkepbi.reverse()
print(countkepbi[:5])
# Use the most popular 1000 words to save computation time
wordskkepbi = [x[1] for x in countkepbi[:1000]]
wordIdkepbi = dict(zip(wordskkepbi, range(len(wordskkepbi))))

# Feature vector for word counts
featkepbi=numpy.array([[0]*(len(wordskkepbi)+1)]*len(train))
for i in range(len(train)):
    for w in [d for d in nltk.bigrams(reviewtotalkepbi[i])]:
        if w in wordskkepbi:
            featkepbi[i][wordIdkepbi[w]] += 1
for i in range(len(train)):
    featkepbi[i][len(wordskkepbi)]= 1
```

```
218702
[(4749, ('with', 'a')), (2642, ('in', 'the')), (2395, ('of', 'the')), (2076,
('is', 'a')), (1974, ('on', 'the'))]
```

In [ ]:
```
# Feature vector for tfidf
idfdictkepbi = defaultdict(int)
for w in wordskkepbi:
    idfdictkepbi[w] = 0
    for r in reviewtotalkepbi:
        if w in set([d for d in nltk.bigrams(r)]):
            idfdictkepbi[w] += 1
idflogkepbi = defaultdict(float)
# Compute idf
for w in wordskkepbi:
    # To avoid zero division, we adjust the denominator by adding one
    idflogkepbi[w] = math.log(len(train)/float(1+idfdictkepbi[w]))
Xkepbi = numpy.array([[0.0]*(len(wordskkepbi)+1)]*len(train))
# Compute tfidf
for i in range(len(train)):
    for w in wordskkepbi:
        Xkepbi[i][wordIdkepbi[w]] = float(featkepbi[i][wordIdkepbi[w]])*idflog
kepbi[w]
# Offset
for i in range(len(train)):
    Xkepbi[i][len(wordskkepbi)]= 1
```

In [249]:
```
# Validation feature vector for punctuation, bigram
reviewtotalkepbival= [(''.join([c for c in d['review/text'].lower() ])).split
() for d in validation]

# Feature vector for word counts
featkepbival=numpy.array([[0]*(len(wordskkepbi)+1)]*len(validation))
for i in range(len(validation)):
    for w in [d for d in nltk.bigrams(reviewtotalkepbival[i])]:
        if w in wordskkepbi:
            featkepbival[i][wordIdkepbi[w]] += 1
for i in range(len(validation)):
    featkepbival[i][len(wordskkepbi)]= 1
```

In [ ]:
```
# Feature vector for tfidf
idfdictkepbi = defaultdict(int)
for w in wordskkepbi:
    idfdictkepbi[w] = 0
    for r in reviewtotalkepbival:
        if w in set([d for d in nltk.bigrams(r)]):
            idfdictkepbi[w] += 1
idflogkepbi = defaultdict(float)
# Compute idf
for w in wordskkepbi:
    # To avoid zero division, we adjust the denominator by adding one
    idflogkepbi[w] = math.log(len(validation)/float(1+idfdictkepbi[w]))
Xkepbival = numpy.array([[0.0]*(len(wordskkepbi)+1)]*len(validation))
# Compute tfidf
for i in range(len(validation)):
    for w in wordskkepbi:
        Xkepbival[i][wordIdkepbi[w]] = float(featkepbival[i][wordIdkepbi[w]])*
idflogkepbi[w]
# Offset
for i in range(len(validation)):
    Xkepbival[i][len(wordskkepbi)]= 1
```

In [250]:
```
# Test feature vector for punctuation, bigram
reviewtotalkepbitest= [(''.join([c for c in d['review/text'].lower() ])).split
() for d in test]

# Feature vector for word counts
featkepbitest=numpy.array([[0]*(len(wordskkepbi)+1)]*len(test))
for i in range(len(test)):
    for w in [d for d in nltk.bigrams(reviewtotalkepbitest[i])]:
        if w in wordskkepbi:
            featkepbitest[i][wordIdkepbi[w]] += 1
for i in range(len(test)):
    featkepbitest[i][len(wordskkepbi)]= 1
```

In [ ]:
```
# Feature vector for tfidf
idfdictkepbi = defaultdict(int)
for w in wordskkepbi:
    idfdictkepbi[w] = 0
    for r in reviewtotalkepbitest:
        if w in set([d for d in nltk.bigrams(r)]):
            idfdictkepbi[w] += 1
idflogkepbi = defaultdict(float)
# Compute idf
for w in wordskkepbi:
    # To avoid zero division, we adjust the denominator by adding one
    idflogkepbi[w] = math.log(len(test)/float(1+idfdictkepbi[w]))
Xkepbitest = numpy.array([[0.0]*(len(wordskkepbi)+1)]*len(test))
# Compute tfidf
for i in range(len(test)):
    for w in wordskkepbi:
        Xkepbitest[i][wordIdkepbi[w]] = float(featkepbitest[i][wordIdkepbi[w
]])*idflogkepbi[w]
# Offset
for i in range(len(test)):
    Xkepbitest[i][len(wordskkepbi)]= 1
```

In [251]:
```
ytrain = [d['review/overall'] for d in train]
yval = [d['review/overall'] for d in validation]
ytest = [d['review/overall'] for d in test]
```

In [470]:
```
# Unigram model With regularization [0.01, 0.1, 1, 10, 100]
regarr = [0.01, 0.1, 1, 10, 100]
# validation mse
msearr = list()
# train mse
msetra = list()
for x,v in [(featnop,featnopval),(Xnop,Xnopval),(featkep,featkepval),(Xkep,Xke
pval)]:
    mstr = list()
    mss = list()
    for r in regarr:
        clf = linear_model.Ridge(r, fit_intercept=False)
        clf.fit(x, ytrain)
        predictions = clf.predict(v)
        mse = mean_squared_error(yval,predictions)
        mss.append(mse)
        mstr.append(mean_squared_error(ytrain,clf.predict(x)))
    msearr.append(mss)
    msetra.append(mstr)
```

In [471]:
```
print(msearr)
```

```
[[0.373951497896976, 0.37374591265274953, 0.37185157125309326, 0.360185218665
3092, 0.3857725129032057], [0.37449970918898207, 0.3744844731103517, 0.374343
15966745535, 0.373937209397774, 0.42465924710094693], [0.403242995465363, 0.4
030601898028944, 0.40130345804694273, 0.3890404578453863, 0.404001576994848
4], [0.40318503546060486, 0.4031649385550325, 0.4030021780753027, 0.403069029
31010885, 0.45908474630822177]]
```

In [472]:
```
print(msetra)
```

```
[[0.24814710994213152, 0.248147507454014, 0.24818006557777275, 0.249901071092
2311, 0.3046489000048014], [0.24814710658573239, 0.24814718173122882, 0.24815
462406481884, 0.24884156327174792, 0.29032968795018477], [0.2609431108791546,
0.2609433274002733, 0.2609641284321213, 0.26251888981961546, 0.31467955906117
81], [0.26094310993029085, 0.26094323140628456, 0.2609539023017212, 0.2617144
362221991, 0.304066330267617]]
```

Choose best regularization value: featnop-10,Xnop-10,featkep-10,Xkep-1

In [473]:
```
# Bigram model With regularization [0.01, 0.1, 1, 10, 100]
regarr = [0.01, 0.1, 1, 10, 100]
# validation mse
msearr = list()
# train mse
msetra = list()
for x,v in [(featnopbi,featnopbival),(Xnopbi,Xnopbival),(featkepbi,featkepbiva
l),(Xkepbi,Xkepbival)]:
    mstr = list()
    mss = list()
    for r in regarr:
        clf = linear_model.Ridge(r, fit_intercept=False)
        clf.fit(x, ytrain)
        predictions = clf.predict(v)
        mse = mean_squared_error(yval,predictions)
        mss.append(mse)
        mstr.append(mean_squared_error(ytrain,clf.predict(x)))
    msearr.append(mss)
    msetra.append(mstr)
```

In [474]:
```
print(msearr)
```

```
[[0.4201960465823673, 0.4199365924962966, 0.4175098630049625, 0.4017353974380
469, 0.40410808622269373], [0.4206941921969392, 0.42068119672094717, 0.420559
11282958425, 0.42007775201335446, 0.46011653026687], [0.43280913403276583, 0.
432516150891534, 0.4297842617183416, 0.4124637083598569, 0.4123664598529638
3], [0.43337441756641293, 0.4333615556482553, 0.43324022574045884, 0.43271023
19642279, 0.4693858241917744]]
```

In [475]:
```
print(msetra)
```

```
[[0.29099173287393876, 0.2909922145557162, 0.29103146063499397, 0.29304546094
38749, 0.3428847675356635], [0.2910145596622187, 0.29101461378290555, 0.29101
999886811425, 0.29153320477531125, 0.32619073090421363], [0.2974685171469052,
0.2974689880114566, 0.29750919499293743, 0.299673420958376, 0.350571938586780
65], [0.2974705279980012, 0.2974705778293868, 0.2974755362491641, 0.297948550
7985134, 0.33056557844051887]]
```

Choose best regularization value: featnopbi-10,Xnopbi-10,featkepbi-100,Xkepbi-10

In [468]:
```
testmse = list()
for (x,t) in [(featkep,featkeptest),(featnop,featnoptest),(featnopbi,featnopbi
test),(Xnop,Xnoptest),(Xnopbi,Xnopbitest),(Xkepbi,Xkepbitest)]:
    clf = linear_model.Ridge(10, fit_intercept=False)
    clf.fit(x, ytrain)
    predictions = clf.predict(t)
    mse = mean_squared_error(ytest,predictions)
    testmse.append(mse)
print(testmse)
```

```
[0.38987983042922086, 0.3714138489886139, 0.43972036463886016, 0.384921071448
5829, 0.46117770003241293, 0.471691133618177]
```

In [398]:
```
testmse = list()
for (x,t) in [(featkepbi,featkepbitest)]:
    clf = linear_model.Ridge(100, fit_intercept=False)
    clf.fit(x, ytrain)
    predictions = clf.predict(t)
    mse = mean_squared_error(ytest,predictions)
    testmse.append(mse)
print(testmse)
```

```
[0.443989261079279]
```

In [469]:
```
testmse = list()
for (x,t) in [(Xkep,Xkeptest)]:
    clf = linear_model.Ridge(1, fit_intercept=False)
    clf.fit(x, ytrain)
    predictions = clf.predict(t)
    mse = mean_squared_error(ytest,predictions)
    testmse.append(mse)
print(testmse)
```

```
[0.4029103560972454]
```

unigram, remove punctuation, word count-0.3714138489886139

unigram, keep punctuation, word count-0.38987983042922086

unigram, remove punctuation, tfidf-0.3849210714485829

unigram, keep punctuation, tfidf-0.4029103560972454

bigram, remove punctuation, word count-0.43972036463886016

bigram, keep punctuation, word count-0.443989261079279

bigram, remove punctuation, tfidf-0.46117770003241293

bigram, keep punctuation, tfidf-0.471691133618177

In [ ]: