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

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

### Just the first 5000 reviews

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

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

In [2]:

```
### Ignore capitalization and remove punctuation

wordCount = 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 punctuation])
  for w in r.split():
    #w = stemmer.stem(w) # with stemming
    wordCount[w] += 1
```

In [4]:

```
# 1. How many unique bigrams are there amongst all of the reviews?
# List the 5 most-frequently-occurring bigrams along with their number
# of occurrences in the corpus (1 mark).
bigramCount = defaultdict(int)
for d in data:
    r = ''.join([c for c in d['review/text'].lower() if not c in punctuation])
    words = r.split()
    for i in range(0,len(words)-1):
        bigram = words[i] + " " + words[i+1]
        bigramCount[bigram] += 1
```

In [9]:

```
biCountList = [(bigramCount[b],b) for b in bigramCount]
#sorted(biCountList, key=lambda x: x[0], reverse=True)
biCountList.sort()
biCountList.reverse()
for i in range(5):
    print biCountList[i]
```

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

In [11]:

```
#2. The code provided performs least squares using the 1000 most
#common unigrams. Adapt it to use the 1000 most common bigrams and
#report the MSE obtained using the new predictor (use bigrams only,
#i.e., not unigrams+bigrams) (1 mark). Note that the code performs
#regularized regression with a regularization parameter of 1.0.
biwords = [x[1] for x in biCountList[:1000]]
biwordId = dict(zip(biwords, range(len(biwords))))
biwordSet = set(biwords)

def feature(datum):
    feat = [0]*len(biwords)
    r = ''.join([c for c in datum['review/text'].lower() if not c in punctuation
])
    words = r.split()
    for i in range(0,len(words)-1):
        bigram = words[i] + " " + words[i+1]
        if bigram in biwords:
            feat[biwordId[bigram]] += 1
    feat.append(1) #offset
    return feat

X2 = [feature(d) for d in data]
y2 = [d['review/overall'] for d in data]

clf = linear_model.Ridge(1.0, fit_intercept=False)
clf.fit(X2, y2)
theta = clf.coef_
predictions = clf.predict(X2)
```

In [14]:

```
from sklearn.metrics import mean_squared_error
print "MSE:", mean_squared_error(y2, predictions)
```

```
MSE: 0.34315301406136334
```

In [21]:

```
#3. What is the inverse document frequency of the words ‘foam’,
#‘smell’, ‘banana’, ‘lactic’, and ‘tart’? What are their tf-idf
#scores in the first review (using log base 10) (1 mark)?
from math import log
df = defaultdict(int)
for d in data:
    r = ''.join([c for c in d['review/text'].lower() if not c in punctuation])
    words = set(r.split())
    for word in words:
        df[word] += 1
```

In [27]:

```
def idf(word):
    return log(len(data) / df[word]*1.0, 10)
def tf(word, d):
    res = 0
    r = ''.join([c for c in d['review/text'].lower() if not c in punctuation])
    for w in r.split():
        if w == word:
            res += 1
    return res
def tfidf(word, d):
    return tf(word,d) * idf(word)
```

In [26]:

```
words = ['foam','smell','banana','lactic','tart']
for w in words:
    print w + " idf: " + str(idf(w))
for w in words:
    print w + " tfidf in 1st review: " + str(tf(w,data[0])*idf(w))
```

```
foam idf: 1.11394335231
smell idf: 0.47712125472
banana idf: 1.67209785794
lactic idf: 2.92064500141
tart idf: 1.80617997398
foam tfidf in 1st review: 2.22788670461
smell tfidf in 1st review: 0.47712125472
banana tfidf in 1st review: 3.34419571587
lactic tfidf in 1st review: 5.84129000281
tart tfidf in 1st review: 1.80617997398
```

In [29]:

```
#4. What is the cosine similarity between the first and the second
#review in terms of their tf-idf representations (considering unigrams
#only) (1 mark)?
words = [x for x in wordCount]
wordId = dict(zip(words, range(len(words))))
def feature6(d):
    feat = [0]*len(wordCount)
    r = ''.join([c for c in d['review/text'].lower() if not c in punctuation])
    for w in r.split():
        feat[wordId[w]] = tfidf(w,d)
    feat.append(1)
    return feat
```

In [31]:

```
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np
X6 = np.array([feature6(d) for d in data])
```

In [35]:

```
print "Cosine similarity between review 1 and 2:", cosine_similarity(X6[0:1], X6
[1:2])
```

```
Cosine similarity between review 1 and 2: [[0.0673433]]
```

In [38]:

```
#5. Which other review has the highest cosine similarity compared to
#the first review (provide the beerId and profileName, or the text of
#the review) (1 mark)?
similarities = []
for i in range(1,len(data)):
    d = data[i]
    similarity = cosine_similarity(X6[0:1], X6[i:i+1])
    similarities.append((similarity, (d['beer/beerId'])))
similarities.sort()
similarities.reverse()
```

In [41]:

```
print "Top cosine similarity: " + str(similarities[0][0])
print "Top beerId: " + str(similarities[0][1])
```

```
Top cosine similarity: [[0.30083382]]
Top beerId: 72146
```

In [42]:

```
#6. Adapt the original model that uses the 1000 most common unigrams,
#but replace the features with their 1000-dimensional tf-idf
#representations, and report the MSE obtained with the new model.
y6 = np.array([d['review/overall'] for d in data])
clf = linear_model.Ridge(1.0, fit_intercept=False)
clf.fit(X6, y6)
theta = clf.coef_
predictions = clf.predict(X6)
print "MSE:", mean_squared_error(y6, predictions)
```

```
MSE: 0.0007545265679600412
```

In [46]:

```
len(X6[0])
```

Out[46]:

```
19427
```

In [47]:

```
#7. Implement a validation pipeline for this same data, by randomly shuffling th
e data,
#using 5000 reviews for training, another 5000 for validation, and another 5000
 for testing.
#Consider regularization parameters in the range {0.01, 0.1, 1, 10, 100}, and re
port MSEs
#on the test set for the model that performs best on the validation set. Using t
his pipeline,
#compare the following alternatives in terms of their performance:
# • Unigrams vs. bigrams
# • Removing punctuation vs. preserving it. The model that preserves punctuation
 should treat punc-
# tuation characters as separate words, e.g. “Amazing!” would become [‘amazing’,
 ‘!’]
# • tfidf vs. word counts
# In total you should compare 2 × 2 × 2 = 8 models, and produce a table comparin
g their performance (2 marks)
print "Reading data..."
data7 = list(parseData("http://jmcauley.ucsd.edu/cse190/data/beer/beer_50000.jso
n"))[:15000]
print "Done"
```

```
Reading data...
Done
```

In [48]:

```
numpy.random.shuffle(data7)
train = data7[:5000]
valid = data7[5000:10000]
test = data7[10000:]
```

In [144]:

```
# tfidf
def getTfidf(word, d, inputWords, punc, dataset):
    if word in inputWords:
        inpu = inputWords[word]
        myidf = log(len(dataset)*1.0 / (inpu), 10)
    else:
        myidf = log(len(dataset) / 1.0, 10)

    mytf = 0
    if punc:
        r = ''.join([c for c in d['review/text'].lower() if not c in punctuation
])
    else:
        l = []
        for c in d['review/text'].lower():
            if c in punctuation:
                l.append(' '+c)
            else:
                l.append(c)
        r = ''.join(l)
    for w in r.split():
        if w == word:
            mytf += 1
    res = mytf * myidf
    return res
```

In [145]:

```
# get inputWords and inputId
def getInput(data, punc, bi, tfidf):
    gramCount = defaultdict(int)
    for d in data:
        if punc:
            r = ''.join([c for c in d['review/text'].lower() if not c in punctua
tion])
        else:
            l = []
            for c in d['review/text'].lower():
                if c in punctuation:
                    l.append(' '+c)
                else:
                    l.append(c)
            r = ''.join(l)
        words = r.split()
        if bi:
            for i in range(0,len(words)-1):
                bigram = words[i] + " " + words[i+1]
                gramCount[bigram] += 1
        else:
            for w in words:
                gramCount[w] += 1
    counts = [(gramCount[w], w) for w in gramCount]
    counts.sort()
    counts.reverse()
    uni = [x[1] for x in counts[:1000]]
    wordId = dict(zip(uni, range(len(uni))))
    if tfidf:
#         uni = [x for x in gramCount]
#         wordId = dict(zip(uni, range(len(uni))))
#         return (gramCount, wordId)
        wordId = dict(zip(uni, range(len(uni))))
        uni = dict(zip(uni, [x[0] for x in counts[:1000]]))
    return (uni, wordId)
```

In [146]:

```
# args:
# inputWords:
# inputId:
# punc: remove punc or not
# bi: use bi or uni
# tfidf: use tfidf or count
def feature7(datum, inputWords, inputId, punc, bi, tfidf, dataset):
    feat = [0]*len(inputWords)
    if punc:
        r = ''.join([c for c in datum['review/text'].lower() if not c in punctua
tion])
    else:
        l = []
        for c in datum['review/text'].lower():
            if c in punctuation:
                l.append(' '+c)
            else:
                l.append(c)
        r = ''.join(l)
    words = r.split()
    if tfidf:
        for w in words:
            if w in inputId:
                feat[inputId[w]] = getTfidf(w, datum, inputWords, punc, dataset)
    else:
        if bi:
            for i in range(0,len(words)-1):
                bigram = words[i] + " " + words[i+1]
                if bigram in inputId:
                    feat[inputId[bigram]] += 1
        else:
            for w in words:
                if w in inputId:
                    feat[inputId[w]] += 1
    feat.append(1) #offset
    return feat
```

In [147]:

```
# args:
# args[0] : uni or bi
# args[1] : remove punc or not
# args[2] : tfidf or not
# args[3] : lambda
# args[4] : training dataset
# args[5] : validation dataset
def fun7(args):
    inputWords, inputId = getInput(args[4], args[1], args[0], args[2])
    X = [feature7(d, inputWords, inputId, args[1], args[0], args[2], args[4]) fo
r d in args[4]]
    y = np.array([d['review/overall'] for d in args[4]])
    clf = linear_model.Ridge(args[3], fit_intercept=False)
    clf.fit(X, y)
    theta = clf.coef_

    inputWords2, inputId2 = getInput(args[5], args[1], args[0], args[2])
    X2 = [feature7(d, inputWords2, inputId2, args[1], args[0], args[2], args[5])
 for d in args[5]]
    y2 = np.array([d['review/overall'] for d in args[5]])

    predictions = clf.predict(X2)
    return mean_squared_error(y2, predictions)
```

In [154]:

```
res1 = defaultdict(list)
# lamda = 0.01
args = [0, 0, 0, 0.01, train, valid]
res1[fun7(args)] = args[:4]
print 1
args = [1, 0, 0, 0.01, train, valid]
res1[fun7(args)] = args[:4]
print 1
args = [0, 1, 0, 0.01, train, valid]
res1[fun7(args)] = args[:4]
print 1
# args = [0, 0, 1, 0.01, train, valid]
# res[fun7(args)] = args[:4]
print 1
args = [1, 1, 0, 0.01, train, valid]
res1[fun7(args)] = args[:4]
print 1
# args = [0, 1, 1, 0.01, train, valid]
# res[fun7(args)] = args[:4]
print 1
args = [1, 0, 1, 0.01, train, valid]
res1[fun7(args)] = args[:4]
print 1
args = [1, 1, 1, 0.01, train, valid]
res1[fun7(args)] = args[:4]
print 1
```

```
1
1
1
1
1
1
1
1
```

In [158]:

```
res2 = defaultdict(list)
# lamda = 0.1
args = [0, 0, 0, 0.1, train, valid]
res2[fun7(args)] = args[:4]
print 1
args = [1, 0, 0, 0.1, train, valid]
res2[fun7(args)] = args[:4]
print 1
args = [0, 1, 0, 0.1, train, valid]
res2[fun7(args)] = args[:4]
print 1
# args = [0, 0, 1, 0.1, train, valid]
# res2[fun7(args)] = args[:4]
print 1
args = [1, 1, 0, 0.1, train, valid]
res2[fun7(args)] = args[:4]
print 1
# args = [0, 1, 1, 0.1, train, valid]
# res2[fun7(args)] = args[:4]
print 1
args = [1, 0, 1, 0.1, train, valid]
res2[fun7(args)] = args[:4]
print 1
args = [1, 1, 1, 0.1, train, valid]
res2[fun7(args)] = args[:4]
print 1
```

```
1
1
1
1
1
1
1
1
```

In [159]:

```
res3 = defaultdict(list)
# lamda = 1.0
args = [0, 0, 0, 1.0, train, valid]
res3[fun7(args)] = args[:4]
print 1
args = [1, 0, 0, 1.0, train, valid]
res3[fun7(args)] = args[:4]
print 1
args = [0, 1, 0, 1.0, train, valid]
res3[fun7(args)] = args[:4]
print 1
# args = [0, 0, 1, 1.0, train, valid]
# res3[fun7(args)] = args[:4]
print 1
args = [1, 1, 0, 1.0, train, valid]
res3[fun7(args)] = args[:4]
print 1
# args = [0, 1, 1, 1.0, train, valid]
# res3[fun7(args)] = args[:4]
print 1
args = [1, 0, 1, 1.0, train, valid]
res3[fun7(args)] = args[:4]
print 1
args = [1, 1, 1, 1.0, train, valid]
res3[fun7(args)] = args[:4]
print 1
```

```
1
1
1
1
1
1
1
1
```

In [161]:

```
res4 = defaultdict(list)
# lamda = 10.0
args = [0, 0, 0, 10.0, train, valid]
res4[fun7(args)] = args[:4]
print 1
args = [1, 0, 0, 10.0, train, valid]
res4[fun7(args)] = args[:4]
print 1
args = [0, 1, 0, 10.0, train, valid]
res4[fun7(args)] = args[:4]
print 1
# args = [0, 0, 1, 10.0, train, valid]
# res4[fun7(args)] = args[:4]
print 1
args = [1, 1, 0, 10.0, train, valid]
res4[fun7(args)] = args[:4]
print 1
# args = [0, 1, 1, 10.0, train, valid]
# res4[fun7(args)] = args[:4]
print 1
args = [1, 0, 1, 10.0, train, valid]
res4[fun7(args)] = args[:4]
print 1
args = [1, 1, 1, 10.0, train, valid]
res4[fun7(args)] = args[:4]
print 1
```

```
1
1
1
1
1
1
1
1
```

In [164]:

```
res5 = defaultdict(list)
# lamda = 100.0
args = [0, 0, 0, 100.0, train, valid]
res5[fun7(args)] = args[:4]
print 1
args = [1, 0, 0, 100.0, train, valid]
res5[fun7(args)] = args[:4]
print 1
args = [0, 1, 0, 100.0, train, valid]
res5[fun7(args)] = args[:4]
print 1
# args = [0, 0, 1, 100.0, train, valid]
# res5[fun7(args)] = args[:4]
print 1
args = [1, 1, 0, 100.0, train, valid]
res5[fun7(args)] = args[:4]
print 1
# arg5 = [0, 1, 1, 100.0, train, valid]
# res4[fun7(args)] = args[:4]
print 1
args = [1, 0, 1, 100.0, train, valid]
res5[fun7(args)] = args[:4]
print 1
args = [1, 1, 1, 100.0, train, valid]
res5[fun7(args)] = args[:4]
print 1
```

```
1
1
1
1
1
1
1
1
```

In [ ]:

```
print ("uni/bi\tpunc?\ttfidf?\tlambda\tmse")
print (str(p)+'\t\t'+str(n)+'\t\t'+str(tp)+'\t\t'+str(tn)+'\t\t'+str(fp)+'\t\t'+
str(fn))
```

In [151]:

```
args = [1, 1, 1, 0.01, train, valid]
res[fun7(args)] = args[:4]
```

In [163]:

```
(min(res1),res1[min(res1)])
```

Out[163]:

```
(0.5970640025038999, [1, 1, 1, 0.01])
```

In [167]:

```
mse = [(min(res1),res1[min(res1)]), (min(res2),res2[min(res2)]), (min(res3),res3
[min(res3)]), (min(res4),res4[min(res4)]), (min(res5),res5[min(res5)])]
min(mse)
```

Out[167]:

```
(0.5967228452077881, [1, 1, 1, 10.0])
```

In [166]:

```
```

Out[166]:

```
(0.5967228452077881, [1, 1, 1, 10.0])
```

In [169]:

```
# best lambda = 10.0
res7 = defaultdict(list)
# lamda = 10.0
args = [0, 0, 0, 10.0, train, test]
res7[fun7(args)] = args[:4]
print 1
args = [1, 0, 0, 10.0, train, test]
res7[fun7(args)] = args[:4]
print 1
args = [0, 1, 0, 10.0, train, test]
res7[fun7(args)] = args[:4]
print 1
args = [0, 0, 1, 10.0, train, test]
res4[fun7(args)] = args[:4]
print 1
args = [1, 1, 0, 10.0, train, test]
res7[fun7(args)] = args[:4]
print 1
args = [0, 1, 1, 10.0, train, test]
res4[fun7(args)] = args[:4]
print 1
args = [1, 0, 1, 10.0, train, test]
res7[fun7(args)] = args[:4]
print 1
args = [1, 1, 1, 10.0, train, test]
res7[fun7(args)] = args[:4]
print 1
```

```
1
1
1
1
1
1
1
1
```

In [179]:

```
res7
```

Out[179]:

```
defaultdict(list,
            {0.581299851195812: [1, 1, 1, 10.0],
             0.5967228452077881: [1, 0, 1, 10.0],
             0.6864335242537983: [1, 1, 0, 10.0],
             0.729615915571554: [0, 1, 1, 10.0],
             0.7341914781601425: [0, 1, 0, 10.0],
             0.7654271995308849: [0, 0, 1, 10.0],
             0.7884975249979896: [1, 0, 0, 10.0]})
```

In [ ]:

```

```