In [2]:

```
import numpy as np
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
import operator
from math import log
import re
from random import shuffle
from sklearn.metrics import mean_squared_error
```

In [3]:

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

### Just the first 5000 reviews

print ("Reading data...")
fullData = list(parseData("http://jmcauley.ucsd.edu/cse190/data/beer/
beer_50000.json"))
data = fullData[:5000]
shuffle(fullData)
trainData = fullData[:5000]
valData = fullData[5000:10000]
testData = fullData[10000:]
print ("done")
translator = str.maketrans(dict.fromkeys(string.punctuation))
```

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

In [10]:

```
### How many unique words are there?

wordCount = defaultdict(int)
for d in data:
    for w in d['review/text'].split():
        wordCount[w] += 1

print(len(wordCount))

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

### Just take the most popular words...

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 punc
tuation])
    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]]
toBeFound = words.copy()
```

```
36225
```

In [11]:

```
### Bigram counts
bigramCount = defaultdict(int)
reviewBigramContent = []
for d in data:
    text = d['review/text']
    removed = text.translate(translator)
    lowered = removed.lower()
    wordList = lowered.split()
    prev = None
    reviewSet = []
    for w in wordList:
        if (prev is None):
            bigramCount[('/',w)] += 1
            reviewSet.append(('/',w))
        else:
            bigramCount[(prev,w)] += 1
            reviewSet.append((prev,w))
        prev = w
    reviewBigramContent.append(reviewSet)
frequentWords = sorted(list(bigramCount.items()), key=operator.itemgetter(1))
frequentWords.reverse()
bigrams = set(bigramCount.keys())
bigramNoDup = list(bigrams)
print('')
print('Total number of unique bigrams: {}'.format(len(frequentWords)))
print('Top 5 bigrams:')
for i in range(5):
    print('Bigram {} with count {}'.format(frequentWords[i][0],frequentWords[i][1]))

wordId = dict(zip(words, range(len(words))))
wordSet = set(words)

def feature(datum, wordKey):
    feat = [0]*len(words)
    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[wordKey[w]] += 1
    feat.append(1) #offset
    return feat

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

#No regularization
#theta,residuals,rank,s = numpy.linalg.lstsq(X, y)

#With regularization
clf = linear_model.Ridge(1.0, fit_intercept=False)
clf.fit(X, y)
theta = clf.coef_
```

```
predictions = clf.predict(X)
err = mean_squared_error(y,predictions)
print('\nMSE of unigram model: {}'.format(err))

### Bigram model
def featureBi(bigramList,wordKey):
    feat = [0] * len(bigrams)
    for w in bigramList:
        feat[wordKey[w]] += 1
    feat.append(1)
    return feat
wordId = dict(zip(bigramNoDup, range(len(bigramNoDup))))
X = [featureBi(d,wordId) for d in reviewBigramContent]
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)
err = mean_squared_error(y,predictions)
print('\nMSE of bigram model: {}'.format(err))
```

```
Total number of unique bigrams: 182902
Top 5 bigrams:
Bigram ('with', 'a') with count 4587
Bigram ('in', 'the') with count 2595
Bigram ('of', 'the') with count 2245
Bigram ('is', 'a') with count 2056
Bigram ('on', 'the') with count 2033

MSE of unigram model: 0.2787546353022137

MSE of bigram model: 0.0004163648437118055
```

In [12]:

```
reviewsAndWords = []
allWords = set()
for d in data:
    text = d['review/text']
    words = ((text.translate(translator)).lower()).split()
    reviewsAndWords.append(words)
    for w in words:
        allWords.add(w)
```

In [13]:

```
checkWords = ['foam','smell','banana','lactic','tart']
totalDocuments = len(reviewsAndWords)
inverseFrequencies = defaultdict(float)
for word in allWords:
    doc = 0
    for rev in reviewsAndWords:
        if word in rev:
            doc += 1
    inverseFrequencies[word] = log(totalDocuments/doc, 10)
```

In [14]:

```
for word in checkWords:
    print('IDF score for word "{}": {}'.format(word,inverseFrequencies[word]))
```

```
IDF score for word "foam": 1.1378686206869628
IDF score for word "smell": 0.5379016188648442
IDF score for word "banana": 1.6777807052660807
IDF score for word "lactic": 2.920818753952375
IDF score for word "tart": 1.8068754016455382
```

In [15]:

```
for word in checkWords:
    c = reviewsAndWords[0].count(word)
    print('TF-IDF score for word "{}" in first document: {}'.format(word,c*inverseFrequencies[word]))
```

```
TF-IDF score for word "foam" in first document: 2.2757372413739256
TF-IDF score for word "smell" in first document: 0.5379016188648442
TF-IDF score for word "banana" in first document: 3.3555614105321614
TF-IDF score for word "lactic" in first document: 5.84163750790475
TF-IDF score for word "tart" in first document: 1.8068754016455382
```

In [16]:

```
v1 = []
v2 = []
for word in allWords:
    c1 = reviewsAndWords[0].count(word)
    c2 = reviewsAndWords[1].count(word)
    v1.append(c1*inverseFrequencies[word])
    v2.append(c2*inverseFrequencies[word])
def cos_sim(a, b):
    """Takes 2 vectors a, b and returns the cosine similarity according
    to the definition of the dot product
    """
    dot_product = np.dot(a, b)
    norm_a = np.linalg.norm(a)
    norm_b = np.linalg.norm(b)
    if norm_a == 0 or norm_b == 0:
        return 0
    return dot_product / (norm_a * norm_b)
print('Cosine similarity of first and second review: {}'.format(cos_sim(v1,v2)))
```

```
Cosine similarity of first and second review: 0.06588193974744382
```

In [17]:

```
vectors = []
for i in range(len(reviewsAndWords)):
    v = []
    for word in allWords:
        c = reviewsAndWords[i].count(word)
        v.append(c*inverseFrequencies[word])
    vectors.append(v)
```

In [18]:

```
v1 = vectors[0]

maxCos = 0
index = 1
maxIndex = -1
for v in vectors[1:]:
    cos = cos_sim(v1,v)
    if cos > maxCos:
        maxCos = cos
        maxIndex = index
    index += 1
print('Review with highest cosine similarity to first review has beer
Id "{}" and profileName "{}"'.format(data[maxIndex]['beer/beerId'], d
ata[maxIndex]['user/profileName']))
```

```
Review with highest cosine similarity to first review has beerId "721
46" and profileName "spicelab"
```

In [19]:

```
vectors = []
for i in range(len(reviewsAndWords)):
    v = []
    v.append(1)
    for word in toBeFound:
        c = reviewsAndWords[i].count(word)
        v.append(c*inverseFrequencies[word])
    vectors.append(v)
X = vectors.copy()
y = [d['review/overall'] for d in data]

clf = linear_model.Ridge(1, fit_intercept=False)
clf.fit(X, y)
theta = clf.coef_
predictions = clf.predict(X)
err = mean_squared_error(y,predictions)
print('MSE of TF-IDF model: {}'.format(err))
```

```
MSE of TF-IDF model: 0.27875956007772285
```

In [4]:

```
### Model testing
def getData(reviews, train, uni, punc, tfidf, trainWords=None):
    revWords = []
    allWords = set()
    for review in reviews:
        text = review['review/text']
        if not punc:
            words = ((text.translate(translator)).lower()).split()
        else:
            words = re.findall(r"[\w']+|[.,!?;]",text.lower())
        revWords.append(words)
    if uni:
        revCounts = []
        wordInverseFreq = defaultdict(int)
        for rev in revWords:
            wordCounts = defaultdict(int)
            for word in rev:
                allWords.add(word)
                wordCounts[word] += 1
            revCounts.append(wordCounts)
        total = len(reviews)
        for word in allWords:
            count = 0
            for rev in revCounts:
                if word in rev:
                    count += 1
            wordInverseFreq[word] = log(total/count,10)
    else:
        revCounts = []
        wordInverseFreq = defaultdict(int)
        for rev in revWords:
            pairCounts = defaultdict(int)
            for i in range(len(rev)-1):
                pair = (rev[i], rev[i+1])
                allWords.add(pair)
                pairCounts[pair] += 1
            revCounts.append(pairCounts)
        total = len(reviews)
        for pair in allWords:
            count = 0
            for rev in revCounts:
                if pair in rev:
                    count += 1
            wordInverseFreq[pair] = log(total/count,10)
    if not tfidf:
        if train:
            X = []
            for rev in revCounts:
                feat = []
                for word in allWords:
                    if word in rev.keys():
                        feat.append(rev[word])
                    else:
                        feat.append(0)
                X.append(feat)
        else:
            X = []
            for rev in revCounts:
                feat = []
                for word in trainWords:
                    if word in rev.keys():
                        feat.append(rev[word])
                    else:
                        feat.append(0)
                X.append(feat)
    else:
        if train:
            X = []
            for rev in revCounts:
                feat = []
                for word in allWords:
                    if word in rev.keys():
                        feat.append(rev[word] * wordInverseFreq[word
])
                    else:
                        feat.append(0)
                X.append(feat)
        else:
            X = []
            for rev in revCounts:
                feat = []
                for word in trainWords:
                    if word in rev.keys():
                        feat.append(rev[word] * wordInverseFreq[word
])
                    else:
                        feat.append(0)
                X.append(feat)
    return X, list(allWords)
```

In [7]:

```
def train_and_test(unigram,punctuation,tfidf):
    X_train, words = getData(trainData, True, unigram,punctuation,tfi
df)
    y_train = [r['review/overall'] for r in trainData]
    X_val, _ = getData(valData,False,unigram,punctuation,tfidf,words)
    y_val = [r['review/overall'] for r in valData]
    regularizers = [0.01,0.1,1,10,100]
    for reg in regularizers:
        clf = linear_model.Ridge(reg, fit_intercept=False)
        clf.fit(X_train, y_train)
        theta = clf.coef_
        predictions = clf.predict(X_val)
        err = mean_squared_error(y_val,predictions)
        print('Words: {}, Punctuation: {}, Model: {}, Regularizer: {}
, MSE: {}'.format('Unigram' if unigram else 'Bigram','Kept' if punctu
ation else 'Removed','TF-IDF' if tfidf else 'Counts',reg,err))
```

In [8]:

```
unigram = [False,True]
punctuation = [True,False]
tfidf = [True,False]

from multiprocessing import Process

processes = []

for j in punctuation:
    for k in tfidf:
        train_and_test(False,j,k)

for j in punctuation:
    for k in tfidf:
        p = Process(target=train_and_test, args=((True,j,k)))
        p.start()
        p.join()
        processes.append(p)
for p in processes:
    p.join()
```

```
Words: Bigram, Punctuation: Kept, Model: TF-IDF, Regularizer: 0.01, M
SE: 1.8862181310759654
Words: Bigram, Punctuation: Kept, Model: TF-IDF, Regularizer: 0.1, MS
E: 1.8861658649250537
Words: Bigram, Punctuation: Kept, Model: TF-IDF, Regularizer: 1, MSE:
1.885647097566243
Words: Bigram, Punctuation: Kept, Model: TF-IDF, Regularizer: 10, MS
E: 1.8808223634720744
Words: Bigram, Punctuation: Kept, Model: TF-IDF, Regularizer: 100, MS
E: 1.8548149170135138
Words: Bigram, Punctuation: Kept, Model: Counts, Regularizer: 0.01, M
SE: 2.0245113627375444
Words: Bigram, Punctuation: Kept, Model: Counts, Regularizer: 0.1, MS
E: 2.023304775836114
Words: Bigram, Punctuation: Kept, Model: Counts, Regularizer: 1, MSE:
2.0118209329540218
Words: Bigram, Punctuation: Kept, Model: Counts, Regularizer: 10, MS
E: 1.933035297458138
Words: Bigram, Punctuation: Kept, Model: Counts, Regularizer: 100, MS
E: 1.794753905512156
Words: Bigram, Punctuation: Removed, Model: TF-IDF, Regularizer: 0.0
1, MSE: 2.167417092541125
Words: Bigram, Punctuation: Removed, Model: TF-IDF, Regularizer: 0.1,
MSE: 2.167395081397154
Words: Bigram, Punctuation: Removed, Model: TF-IDF, Regularizer: 1, M
SE: 2.167177458626135
Words: Bigram, Punctuation: Removed, Model: TF-IDF, Regularizer: 10,
MSE: 2.165233997268316
Words: Bigram, Punctuation: Removed, Model: TF-IDF, Regularizer: 100,
MSE: 2.1604211159781204
Words: Bigram, Punctuation: Removed, Model: Counts, Regularizer: 0.0
1, MSE: 2.3031320540739304
Words: Bigram, Punctuation: Removed, Model: Counts, Regularizer: 0.1,
MSE: 2.302177299615284
Words: Bigram, Punctuation: Removed, Model: Counts, Regularizer: 1, M
SE: 2.2931072880386636
Words: Bigram, Punctuation: Removed, Model: Counts, Regularizer: 10,
MSE: 2.2325865651085564
Words: Bigram, Punctuation: Removed, Model: Counts, Regularizer: 100,
MSE: 2.1809703271148106
Words: Unigram, Punctuation: Kept, Model: TF-IDF, Regularizer: 0.01,
MSE: 4.01055826797125
Words: Unigram, Punctuation: Kept, Model: TF-IDF, Regularizer: 0.1, M
SE: 3.948424080348474
Words: Unigram, Punctuation: Kept, Model: TF-IDF, Regularizer: 1, MS
E: 3.5364562233438903
Words: Unigram, Punctuation: Kept, Model: TF-IDF, Regularizer: 10, MS
E: 2.568415049160528
Words: Unigram, Punctuation: Kept, Model: TF-IDF, Regularizer: 100, M
SE: 1.8861795490128672
Words: Unigram, Punctuation: Kept, Model: Counts, Regularizer: 0.01,
MSE: 4.322541025784355
Words: Unigram, Punctuation: Kept, Model: Counts, Regularizer: 0.1, M
SE: 3.891389329026427
Words: Unigram, Punctuation: Kept, Model: Counts, Regularizer: 1, MS
E: 2.7900857443768685
```

```
Words: Unigram, Punctuation: Kept, Model: Counts, Regularizer: 10, MS
E: 1.9151691017847603
Words: Unigram, Punctuation: Kept, Model: Counts, Regularizer: 100, M
SE: 1.607167223096318
Words: Unigram, Punctuation: Removed, Model: TF-IDF, Regularizer: 0.0
1, MSE: 3.4711037409717
Words: Unigram, Punctuation: Removed, Model: TF-IDF, Regularizer: 0.
1, MSE: 3.4379850742336084
Words: Unigram, Punctuation: Removed, Model: TF-IDF, Regularizer: 1,
MSE: 3.196840379015003
Words: Unigram, Punctuation: Removed, Model: TF-IDF, Regularizer: 10,
MSE: 2.495611933195827
Words: Unigram, Punctuation: Removed, Model: TF-IDF, Regularizer: 10
0, MSE: 1.9102693980827168
Words: Unigram, Punctuation: Removed, Model: Counts, Regularizer: 0.0
1, MSE: 3.786931886877045
Words: Unigram, Punctuation: Removed, Model: Counts, Regularizer: 0.
1, MSE: 3.534393088459552
Words: Unigram, Punctuation: Removed, Model: Counts, Regularizer: 1,
MSE: 2.7259494347253814
Words: Unigram, Punctuation: Removed, Model: Counts, Regularizer: 10,
MSE: 1.94142584543845
Words: Unigram, Punctuation: Removed, Model: Counts, Regularizer: 10
0, MSE: 1.6493934955017735
```

In [9]:

```
### Best model's performance on the test set
X_train, words = getData(trainData, True, True,True,False)
y_train = [r['review/overall'] for r in trainData]
X_val, _ = getData(testData,False,True,True,False,words)
y_val = [r['review/overall'] for r in testData]
clf = linear_model.Ridge(100, fit_intercept=False)
clf.fit(X_train, y_train)
theta = clf.coef_
predictions = clf.predict(X_val)
err = mean_squared_error(y_val,predictions)
print('Words: {}, Punctuation: {}, Model: {}, Regularizer: {}, MSE:
{}'.format('Unigram','Kept','Counts',100,err))
```

```
Words: Unigram, Punctuation: Kept, Model: Counts, Regularizer: 100, M
SE: 1.6688250630614938
```