CSE 258 - Homework 4

```
from collections import defaultdict
import numpy as np
import string
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
```

```
def parseData(fname):
  for l in open(fname):
    yield eval(l)

### Just the first 5000 reviews

data = list(parseData("beer.json"))[:5000]
```

Task 1

```
### 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 punctuation])
  words = r.split()
  for i in range(0,len(words)-1):
    bigram = words[i] + " " + words[i+1]
    bigramCount[bigram] += 1
```

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

Answer

```
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 the'),
(2056, 'is a'),
(2033, 'on the')]
```

Task 2

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

```
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
```

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

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

```
# 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)
```

Answer

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

```
MSE: 0.343153014061
```

Task 3

```
unigramCount = 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():
    #w = stemmer.stem(w) # with stemming
    unigramCount[w] += 1
```

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

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

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

```
countsCombined = countsUnigram + countsBigram
countsCombined.sort()
countsCombined.reverse()
```

```
combineds = [x[1] for x in countsCombined[:1000]]
combinedId = dict(zip(combineds, range(len(combineds))))
```

```
def feature_3(text):
  feat = [0]*len(combineds)
  words = text.split()
  for i in range(len(words)-1):
    bigram = words[i] + " " + words[i+1]
    try:
      feat[combinedId[bigram]] += 1
    except KeyError:
      continue
  for w in words:
    try:
      feat[combinedId[w]] += 1
    except KeyError:
      continue
  feat.append(1) #offset
  return feat
```

```
X_3 = []
for i in range(len(data)):
  X_3.append(feature_3(reviewText[i]))
y_3 = [d['review/overall'] for d in data]
```

```
# Least squares with regularization
reg = 1.0
clf = linear_model.Ridge(reg, fit_intercept=False)
clf.fit(X_3, y_3)
theta = clf.coef_
predictions_3 = clf.predict(X_3)
```

Answer

```
print "MSE:", mean_squared_error(y_3, predictions_3)
```

```
MSE: 0.289047333034
```

Task 4

Most negative associated

```
minIdx = np.argpartition(theta, 5)[0:5]
smallest = [(combineds[i], theta[i] ) for i in minIdx]
print "Most negative associated weight:"
for x in smallest:
  print "Unigram/Bigram: {:10s} \t with value: {:0.5f}".format(x[0], x[1])
```

```
Most negative associated weight:
Unigram/Bigram: sort of      with value: -0.63976
Unigram/Bigram: water        with value: -0.27049
Unigram/Bigram: corn         with value: -0.23703
Unigram/Bigram: the background   with value: -0.21625
Unigram/Bigram: straw        with value: -0.19594
```

Most positive associated

```
maxIdx = np.argpartition(theta[0:1000], -5)[-5:]
print maxIdx
largest = [(combineds[i], theta[i] ) for i in maxIdx]
print "Most negative associated weight:"
for x in largest:
  print "Unigram/Bigram: {:10s} \t with value: {:0.5f}".format(x[0], x[1])
```

```
[417 841 916 818 852]
Most positive associated weight:
Unigram/Bigram: the best     with value: 0.20639
Unigram/Bigram: not bad      with value: 0.21688
Unigram/Bigram: sort         with value: 0.51983
Unigram/Bigram: a bad        with value: 0.22882
Unigram/Bigram: of these     with value: 0.22283
```

Task 5

```
# Find idf
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 w in words:
    df[w] += 1
```

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

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

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

Answer

```
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
```

Task 6

```
def feature_6(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
```

```
X_6 = np.array([feature_6(d) for d in reviewText])
y_6 = np.array([d['review/overall'] for d in data])
```

Answer

```
print "Cosine similarity between review 1 and 2:", cosine_similarity(X_6[0:1], X_6[1
```

```
Cosine similarity between review 1 and 2: 0.109795490823
```

Task 7

```
similarities = []
for i in range(1,len(data)):
  d = data[i]
  similarity = cosine_similarity(X_6[0:1], X_6[i:i+1])[0,0]
  similarities.append((similarity, (d['beer/beerId'], d['user/profileName'])))
similarities.sort()
similarities.reverse()
```

Answer

```
top = similarities[0]
print "Top cosine similarity: {:0.5f}, userId: {}, beerId: {}".format(top[0], top[1]
```

```
Top cosine similarity: 0.31945, userId: Heatwave33, beerId: 52211
```

Task 8

```
# Least squares with regularization
reg = 1.0
clf = linear_model.Ridge(reg, fit_intercept=False)
clf.fit(X_6, y_6)
theta = clf.coef_
predictions_6 = clf.predict(X_6)
```

Answer

```
print "MSE:", mean_squared_error(predictions_6, y_6)
```

```
MSE: 0.278742490057
```