Recommender Sys & Web Mining Hw3

Hao-en Sung (wrangle1005@gmail.com)
Department of Computer Science
University of California, San Diego
San Diego, CA 92092

Tasks

Using the code provided on the webpage, read the first 5000 reviews from the corpus, and read the reviews without capitalization or punctuation.

Problem 1

First, I use a dictionary to record the counts for each unigram and bigram. After that, I sort both of them by the number of appearances and put them into another dictionary for following sections.

There are overall 19426 unigrams and 182246 bigrams. The most frequent 1000 bigrams are shown as follows.

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

Table 1: Five most Frequent Bigrams

Problem 2

When using the most frequent 1000 bigrams, I am available achieve 0.343153 in terms of Mean Squared Error (MSE) score.

Problem 3

When using both the most frequent 1000 unigrams and bigrams, I can improve my MSE score from 0.343153 to 0.289548.

Problem 4

When mix-considering the most frequent 1000 terms, the five most positive features and most negative features are shown as Table 2 and Table 3.

Problem 5

To solve this problem, I need to recalculate the term frequencies inverse term frequencies for each word. Later, I just iterate through these five words — 'foam', 'smell', 'banana', 'lactic', and 'tart' — and print out their results, which are recorded in Table 4.

Unigram: sort	0.511067
Bigram: ('a', 'bad')	0.224651
Bigram: ('of', 'these')	0.219666
Bigram: ('not', 'bad')	0.213625
Bigram: ('the', 'best')	0.210810

Table 2: Five Most Frequent Unigram / Bigram

Bigram: ('sort', 'of')	-0.634356
Unigram: water	-0.270650
Unigram: corn	-0.237203
Bigram: ('the', 'background')	-0.217837
Unigram: straw	-0.197361

Table 3: Five Most Infrequent Unigram / Bigram

It is noticeable that I use the definition for both term-frequency and term-frequency-inverse document frequency from slides.

Text	idf	tf-idf
foam	1.137869	2.275737
smell	0.537902	0.537902
banana	1.677781	3.355561
lactic	2.920819	5.841638
tart	1.806875	1.806875

Table 4: idf and tf-idf

Problem 6

In order to solve problem 6 to 8, I recalculate the *tf-idf* of each word for all reviews, instead of merely the 1000 common ones. For problem 6, I just use Sklearn to help calculate the cosine similarity between first and second reviews, which is 0.065882.

Problem 7

The largest cosine similarity between first review and other reviews is 0.296868, whose beerID and profileName are 72146 and spicelab, respectively.

Problem 8

When using *tf-idf* as features, I get 0.278760 as MSE, which is slightly better than simply using unigram and/or bigram as features (0.289548).

Appendix

Code Listing 1: Code for Hw4

```
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
from sklearn.metrics import mean_squared_error
from sklearn.metrics.pairwise import cosine_similarity
import operator
import math


## Preprocessing
# Read data
def parseData(fname):
  for l in urllib.urlopen(fname):
    yield eval(l)

print "\nReading data..."
#data = list(parseData("../dat/beer_50000.json"))[:5000]
data = list(parseData("../dat/beer_5000.json"))
print "done"

# Process data; Count grams
uniCount = defaultdict(int)
biCount = defaultdict(int)
mixCount = defaultdict(int)
punctuation = set(string.punctuation)
stemmer = PorterStemmer()
ndata = []
y = []
for d in data:
    r = ''.join([c for c in d['review/text'].lower() if not c in
                                            punctuation])
    r = r.split()
    ndata.append(r)
    y.append(d['review/overall'])

    for i in range(len(r)):
        uniCount[r[i]] += 1
        mixCount[r[i]] += 1
        if i+1 < len(r):
            biCount[r[i], r[i+1]] += 1
            mixCount[r[i], r[i+1]] += 1

print 'Number of unigrams: %d'%len(uniCount)
print 'Number of bigrams: %d'%len(biCount)
print 'Number of mixgrams: %d'%len(mixCount)

# Sort grams by number of appearance
uniCount = sorted(uniCount.items(), key=operator.itemgetter(1,0),
                                   reverse=True)
biCount = sorted(biCount.items(), key=operator.itemgetter(1,0),
                                   reverse=True)
mixCount = sorted(mixCount.items(), key=operator.itemgetter(1,0),
                                   reverse=True)

# create mapping for grams
```

```
uniMap = dict(zip(map(lambda x: x[0], uniCount), range(len(uniCount)))
                                   )
biMap = dict(zip(map(lambda x: x[0], biCount), range(len(biCount))))
mixMap = dict(zip(map(lambda x: x[0], mixCount), range(len(mixCount)))
                                   )


## Problem 1
print '\nProblem 1:'

for i in xrange(5):
    print biCount[i][0], biCount[i][1]


## Problem 2
print '\nProblem 2:'

# Create features
topK = 1000
def feature(r):
    feat = [0 for i in xrange(topK)]
    for i in range(len(r)):
        if i+1 < len(r):
            bg = biMap[(r[i], r[i+1])]
            if bg < topK:
                feat[bg] += 1
    feat.append(1)
    return feat

X = [feature(r) for r in ndata]

# Learn a model with regularization
clf = linear_model.Ridge(1.0, fit_intercept=False)
clf.fit(X, y)
p = clf.predict(X)

# Calculate MSE
mse = mean_squared_error(y, p)
print 'MSE with 1000 top bigrams: %f'%mse


## Problem 3
print '\nProblem 3:'

# Create features
def feature(r):
    feat = [0 for i in xrange(topK)]
    for i in range(len(r)):
        mg = mixMap[r[i]]
        if mg < topK:
            feat[mg] += 1
        if i+1 < len(r):
            mg = mixMap[(r[i], r[i+1])]
            if mg < topK:
                feat[mg] += 1
    feat.append(1)
    return feat

X = [feature(r) for r in ndata]

# Learn a model with regularization
clf = linear_model.Ridge(1.0, fit_intercept=False)
clf.fit(X, y)
theta = clf.coef_
p = clf.predict(X)
```

```
# Calculate MSE
mse = mean_squared_error(y, p)
print 'MSE with 1000 top unigrams and bigrams: %f'%mse


## Problem 4
print '\nProblem 4:'

pv = zip(theta[:topK], range(topK)) # ignore constant feature
pv = sorted(pv, key=operator.itemgetter(0), reverse=True)

print 'Most positive features ---'
for i in range(5):
    val, idx = pv[i]
    tar = mixCount[idx][0]
    if type(tar) == str:
        print 'Unigram %s: %f'%(mixCount[idx][0], val)
    else:
        print 'Bigram %s: %f'%(mixCount[idx][0], val)

print '\nMost negative features ---'
for i in range(5):
    val, idx = pv[len(pv)-1-i]
    tar = mixCount[idx][0]
    if type(tar) == str:
        print 'Unigram %s: %f'%(mixCount[idx][0], val)
    else:
        print 'Bigram %s: %f'%(mixCount[idx][0], val)


## Problem 5
print '\nProblem 5:'

# Create termF
print "\nRe-reading data..."
termF = []
uniDList = [list() for j in xrange(len(uniCount))]
for i in range(len(ndata)):
    r = ndata[i]
    cnt = [0 for j in xrange(len(uniCount))]
    for j in range(len(r)):
        ug = uniMap[r[j]]
        cnt[ug] += 1
        if len(uniDList[ug]) == 0 or uniDList[ug][-1] != i:
            uniDList[ug].append(i)
    termF.append(cnt)
print 'done\n'

# Calculate uniDList and termF-uniDList for 5 candidates
textLst = ['foam', 'smell', 'banana', 'lactic', 'tart']
for text in textLst:
    ug = uniMap[text]
    idf = math.log10(1.0*len(ndata)/len(uniDList[ug]))
    tf = termF[0][ug]
    print '%s: idf(%f), tf-idf(%f)'%(text,idf,tf*idf)


## Problem 6
print '\nProblem 6:'

valMat = []
for i in range(len(ndata)):
    tf_idf = [0 for j in xrange(len(uniCount))]
    r = ndata[i]
    for j in range(len(r)):
        ug = uniMap[r[j]]
        tf = termF[i][ug]
        idf = math.log10(1.0*len(ndata)/len(uniDList[ug]))
        tf_idf[ug] = tf * idf
    valMat.append(tf_idf)

valMat = [np.array(valLst).reshape(1,-1) for valLst in valMat]
cos = cosine_similarity(valMat[0], valMat[1])
print 'Cosine Similiarity between first and second reviews: %f'%cos


## Problem 7
print '\nProblem 7:'

best_cos = -1
for i in range(1, len(ndata)):
    cos = cosine_similarity(valMat[0], valMat[i])
    if cos > best_cos:
        best_cos = cos
        best_tag = i
print 'Largest Cosine Similarity: %f'%best_cos
print 'Content: %s'%data[best_tag]


## Problem 8
print '\nProblem 8:'

# Create features
def feature(idx, r):
    feat = [0 for i in xrange(topK)]
    for j in range(len(r)):
        ug = uniMap[r[j]]
        if ug < topK:
            tf = termF[idx][ug]
            idf = math.log10(1.0*len(ndata)/len(uniDList[ug]))
            feat[ug] = tf * idf
    feat.append(1)
    return feat

X = [feature(idx, r) for idx, r in zip(range(len(ndata)), ndata)]

# Learn a model with regularization
clf = linear_model.Ridge(1.0, fit_intercept=False)
clf.fit(X, y)
theta = clf.coef_
p = clf.predict(X)

# Calculate MSE
mse = mean_squared_error(y, p)
print 'MSE with tf-idf from 1000 top unigrams: %f'%mse
```