CSE 158 hw3

Q1:

```
accuracy:  0.503035
```

Q2:

```
using threhold of the 23rd percentile of populatity has better accuracy
accuracy:  0.605045
```

Q3:

See attached code for revisited baseline implementation

Q4:

```
Name in Kaggle leaderboard: ah
score: 0.66490
email: kyhor@ucsd.edu
```

Q5:

```
accuracy:  0.303265143313
```

Q6:

```
in category  0
the 10 most freq words compare to other category:
[(0.002175286194542249, u'was'), (0.001500068333356113, u'brunch'), (0.001339
9477708666598, u'food'), (0.0011746499194647225, u'breakfast'), (0.0011005589
218836542, u'the'), (0.0009329331391129273, u'menu'), (0.0009089905274235052,
 u'had'), (0.000886761295411112, u'we'), (0.0008312637692096166, u'service'),
 (0.0007225987727119151, u'cheese')]
in category  1
the 10 most freq words compare to other category:
[(0.00701462999249837, u'a'), (0.006821227082725281, u'bar'), (0.004088571612
044616, u'beer'), (0.003733732714768682, u'drinks'), (0.003436296934870655, u
'to'), (0.0023448711410517186, u'music'), (0.0022085462291689412, u'drink'),
```

(0.002184483474731238, u'place'), (0.0019377070246698307, u'great'), (0.00191
94253737134272, u'on')]
in category 2
the 10 most freq words compare to other category:
[(0.0036505782016691274, u'sushi'), (0.0034279808752587784, u'thai'), (0.0028
3088925374733, u'food'), (0.0018819230336621314, u'chinese'), (0.001775886909
6226333, u'indian'), (0.001714629619503988, u'ramen'), (0.0017007372866780436,
u'noodles'), (0.0016597045001939326, u'dishes'), (0.0016414764900811955, u's
picy'), (0.0015997406998564554, u'rice')]
in category 3
the 10 most freq words compare to other category:
[(0.0052544144488884045, u'pizza'), (0.0021033283259192298, u'greek'), (0.002
069022802490735, u'tapas'), (0.001884410832389204, u'beer'), (0.0014396067802
135688, u'german'), (0.0013528984000063908, u'was'), (0.001312396558186601, u
'irish'), (0.0012888843037278871, u'and'), (0.0011370927350144262, u'selectio
n'), (0.001100415535234297, u'great')]
in category 4
the 10 most freq words compare to other category:
[(0.011207066202888968, u'pizza'), (0.007388416328120984, u'italian'), (0.002
710103282768671, u'pasta'), (0.002013858605495693, u'wine'), (0.0018986211818
658707, u'pizzas'), (0.0015970016060480857, u'restaurant'), (0.00140745480431
26194, u'bread'), (0.0013880640358192326, u'very'), (0.0012647406991659036, u
'crust'), (0.0010659039946276283, u'dinner')]
in category 5
the 10 most freq words compare to other category:
[(0.012826947378188602, u'burger'), (0.00855198590504825, u'fries'), (0.00771
0177214924195, u'burgers'), (0.004921990401718415, u'are'), (0.00349823540389
35654, u'fast'), (0.0019950578743474466, u'you'), (0.0018688445656536296, u't
hey'), (0.0017272377734654483, u'their'), (0.0013301900138296943, u'sandwich
'), (0.001313702400723392, u'get')]
in category 6
the 10 most freq words compare to other category:
[(0.008286398759824433, u'mexican'), (0.006626800245453056, u'tacos'), (0.004
181219222944536, u'food'), (0.0033970522649591464, u'taco'), (0.0028444486406
160583, u'margaritas'), (0.0028319757010799693, u'salsa'), (0.002751671713071
9576, u'burrito'), (0.002137910200811109, u'are'), (0.001664989839715766, u'c
hips'), (0.0016416075106612475, u'guacamole')]
in category 7

```
the 10 most freq words compare to other category:
[(0.006272328295358751, u'seafood'), (0.00400446683413696, u'fish'), (0.00347
7326999261822, u'was'), (0.0033507210670156136, u'we'), (0.003257046337286488,
 u'lobster'), (0.002696693871776414, u'oysters'), (0.00258580149905419, u'cra
b'), (0.002007316602284844, u'fresh'), (0.0017662751819412237, u'the'), (0.00
16548325849343008, u'chowder')]
in category  8
the 10 most freq words compare to other category:
[(0.0232878688844021, u'coffee'), (0.0029825656746150305, u'shop'), (0.002445
295192317125, u'starbucks'), (0.0020598776555306487, u'espresso'), (0.0020500
67339727523, u'i'), (0.001974491008991987, u'wifi'), (0.001970572956363684, u
'cup'), (0.0019066246235643792, u'tea'), (0.0018962920951300852, u'to'), (0.0
018394683239147907, u'cafe')]
in category  9
the 10 most freq words compare to other category:
[(0.007877076819206828, u'sandwiches'), (0.0074071285966947325, u'sandwich'),
 (0.0029028993204293584, u'bread'), (0.002897498333143152, u'cheese'), (0.002
8120777134341387, u'their'), (0.002596173878376931, u'deli'), (0.001901234527
16358, u'they'), (0.0016866537630584946, u'lunch'), (0.0016192162856795561, u
'are'), (0.0013669009474340452, u'beef')]
```

Q7:

My code assumed that accuracy only count for predict = valid = is bar

```
c =  0.01  with accuracy:  0.032286455069
c =  0.1  with accuracy:  0.0327286455069
c =  1  with accuracy:  0.0339280870705
c =  10  with accuracy:  0.032565613197
c =  100  with accuracy:  0.0321465430423
c =  1  has highest accuracy: 0.0339280870705
```

Q8:

```
Accuracy on valid set:  0.5521465430423
```

Name in kaggle leaderboard: alexhor
Score = 0.35870
Email: kyhor@ucsd.edu

HW3

November 14, 2017

In [ ]:
```
import gzip
from collections import defaultdict

def readGz(f):
  for l in gzip.open(f):
    yield eval(l)

### Rating baseline: compute averages for each user, or return the global average if we'

allRatings = []
userRatings = defaultdict(list)
for l in readGz("train.json.gz"):
  user,business = l['userID'],l['businessID']
  allRatings.append(l['rating'])
  userRatings[user].append(l['rating'])

globalAverage = sum(allRatings) / len(allRatings)
userAverage = {}
for u in userRatings:
  userAverage[u] = sum(userRatings[u]) / len(userRatings[u])

predictions = open("predictions_Rating.txt", 'w')
for l in open("pairs_Rating.txt"):
  if l.startswith("userID"):
    #header
    predictions.write(l)
    continue
  u,i = l.strip().split('-')
  if u in userAverage:
    predictions.write(u + '-' + i + ',' + str(userAverage[u]) + '\n')
  else:
    predictions.write(u + '-' + i + ',' + str(globalAverage) + '\n')

predictions.close()

### Would-visit baseline: just rank which businesses are popular and which are not, and

businessCount = defaultdict(int)
```

```
totalPurchases = 0

for l in readGz("train.json.gz"):
  user,business = l['userID'],l['businessID']
  businessCount[business] += 1
  totalPurchases += 1

mostPopular = [(businessCount[x], x) for x in businessCount]
mostPopular.sort()
mostPopular.reverse()

return1 = set()
count = 0
for ic, i in mostPopular:
  count += ic
  return1.add(i)
  if count > totalPurchases/2: break

predictions = open("predictions_Visit.txt", 'w')
for l in open("pairs_Visit.txt"):
  if l.startswith("userID"):
    #header
    predictions.write(l)
    continue
  u,i = l.strip().split('-')
  if i in return1:
    predictions.write(u + '-' + i + ",1\n")
  else:
    predictions.write(u + '-' + i + ",0\n")

predictions.close()

### Category prediction baseline: Just consider some of the most common words from each

catDict = {
  "American Restaurant": 0,
  "Bar": 1,
  "Asian Restaurant": 2,
  "European Restaurant": 3,
  "Italian Restaurant": 4,
  "Fast Food Restaurant": 5,
  "Mexican Restaurant": 6,
  "Seafood Restaurant": 7,
  "Coffee Shop": 8,
  "Sandwich Shop": 9
}

predictions = open("predictions_Category.txt", 'w')
```

```
predictions.write("userID-reviewHash,category\n")
for l in readGz("test_Category.json.gz"):
  cat = catDict['American Restaurant'] # If there's no evidence, just choose the most co
  words = l['reviewText'].lower()
  if 'america' in words:
    cat = catDict['American Restaurant']
  if 'bar' in words or 'beer' in words:
    cat = catDict['Bar']
  if 'asia' in words:
    cat = catDict['Asian Restaurant']
  if 'europe' in words:
    cat = catDict['European Restaurant']
  if 'italian' in words:
    cat = catDict['Italian Restaurant']
  if 'fast' in words:
    cat = catDict['Fast Food Restaurant']
  if 'mexic' in words:
    cat = catDict['Mexican Restaurant']
  if 'coffee' in words:
    cat = catDict['Coffee Shop']
  if 'sandwich' in words:
    cat = catDict['Sandwich Shop']
  predictions.write(l['userID'] + '-' + l['reviewHash'] + "," + str(cat) + "\n")

predictions.close()
```

In [ ]:

In [ ]:
```
alluser = list()
allbus = set()
added = dict()
count = 0
valid = open('valid1.txt','w')
valid2 = open('valid2.txt','w')
valid.write('userID-businessID,actual,prediction\n')
valid2.write('userID-businessID,actual,prediction\n')
for l in readGz("train.json.gz"):
    user,business = l['userID'],l['businessID']
    if user in alluser:
        added[user].append(business)
    else:
        added[user]= list()
        added[user].append(business)
        alluser.append(user)
    allbus.add(business)
    if count >= 100000 and count<200000:
        valid.write(user + '-' + business +',1'+"\n")
    count +=1
```

```
        for i in range(100000):
            u = i%len(alluser)
            user = alluser[u]
            diffbus = allbus.difference(added[user])
            if len(diffbus)>0:
                newb = diffbus.pop()
                added[user].append(newb)
                valid2.write(user + '-' + newb +",0" +"\n")
        valid.close()
        valid2.close()
```

```
In [ ]: print 'Question 1:'
        businessCount = defaultdict(int)
        totalPurchases = 0
        for l in readGz("train.json.gz"):
            user,business = l['userID'],l['businessID']
            businessCount[business] += 1
            totalPurchases += 1
            if totalPurchases >= 100000:
                break
        mostPopular = [(businessCount[x], x) for x in businessCount]
        mostPopular.sort()
        mostPopular.reverse()
        return1 = set()
        count = 0
        for ic, i in mostPopular:
            count += ic
            return1.add(i)
            if count > totalPurchases/2:
                break
        samples = float(0)
        correct = float(0)
        for l in open("valid1.txt"):
            samples += 1
            if l.startswith("userID"):
                #header
                samples -= 1
                continue
            u,i = l.strip().split('-')
            i = i.split(',')[0]
            if i in return1:
                correct += 1
        for l in open("valid2.txt"):
            samples += 1
            if l.startswith("userID"):
                #header
                samples -= 1
                continue
```

```
            u,i = l.strip().split('-')
            i = i.split(',')[0]
            if not i in return1:
                correct += 1
        print 'accuracy: ', correct/samples
```

In [ ]:
```
        print 'Question 2:'
        businessCount = defaultdict(int)
        totalPurchases = 0
        for l in readGz("train.json.gz"):
            user,business = l['userID'],l['businessID']
            businessCount[business] += 1
            totalPurchases += 1
            if totalPurchases >= 100000:
                break
        mostPopular = [(businessCount[x], x) for x in businessCount]
        mostPopular.sort()
        mostPopular.reverse()
        return1 = set()
        count = 0
        for ic, i in mostPopular:
            count += ic
            return1.add(i)
            if count > totalPurchases*(0.23):
                break
        samples = float(0)
        correct = float(0)
        for l in open("valid1.txt"):
            samples += 1
            if l.startswith("userID"):
                #header
                samples -= 1
                continue
            u,i = l.strip().split('-')
            i = i.split(',')[0]
            if i in return1:
                correct += 1
        for l in open("valid2.txt"):
            samples += 1
            if l.startswith("userID"):
                #header
                samples -= 1
                continue
            u,i = l.strip().split('-')
            i = i.split(',')[0]
            if not i in return1:
                correct += 1
        print 'accuracy: ', correct/samples
```

```
        print 'using threhold of the 23rd percentile of populatity has better accuracy'
```

In [ ]:
```
print 'Question 3:'
print 'revisit baseline '
def revisit_baseline(userID,busID,allusercats,allbuscats):
    if not userID in allusercats.keys():
        return 0
    if not busID in allbuscats.keys():
        return 0
    usercats = allusercats[userID]
    buscats = allbuscats[busID]
    for bc in buscats:
        if bc in usercats:
            return 1
    return 0
```

In [ ]:
```
print 'Question 4:'
allusercats = dict()
allbuscats = dict()
for l in readGz("train.json.gz"):
    user,bus = l['userID'],l['businessID']
    if not user in allusercats.keys():
        allusercats[user] = set()
    if not bus in allbuscats.keys():
        allbuscats[bus] = set()
    for c in l['categories']:
        allusercats[user].add(c)
        allbuscats[bus].add(c)
predictions = open("predictions_revisit.txt", 'w')
for l in open("pairs_Visit.txt"):
    if l.startswith("userID"):
        #header
        predictions.write(l)
        continue
    u,i = l.strip().split('-')
    if revisit_baseline(u,i,allusercats,allbuscats):
        predictions.write(u + '-' + i + ",1\n")
    else:
        predictions.write(u + '-' + i + ",0\n")
predictions.close()
print 'user name: ah'
print 'score: 0.66490'
print 'email: kyhor@ucsd.edu'
```

In [ ]:
```
print 'prepare data for Question 5:'
def mostpop(cidcounts):
    mostpop = list()
```

```
        for i in range(len(cidcount)):
            mostpop.append((cidcount[i],i))
        mostpop = sorted(mostpop)
        return mostpop

    with_cid = list()
    for l in readGz('train.json.gz'):
        if 'categoryID' in l.keys():
            with_cid.append(l)
    train = with_cid[:len(with_cid)/2]
    valid = with_cid[len(with_cid)/2:]
    cidcount = [0,0,0,0,0,0,0,0,0,0]
    for t in train:
        cidcount[int(t['categoryID'])] += 1

    most_popular = mostpop(cidcount)
    poprank = dict()
    for i in range(1,11):
        poprank[most_popular[-i][1]] = i
```

In [ ]:
```
print 'prepare data for Question 5:'
train_user_cids = dict()
predict_visit = dict()
for t in train:
    u,c = t['userID'],t['categoryID']
    if not u in train_user_cids.keys():
        train_user_cids[u] = [0,0,0,0,0,0,0,0,0,0]
    train_user_cids[u][int(c)] += 1

for u in train_user_cids.keys():
    mp = mostpop(train_user_cids[u])
    if mp[-1][0] != mp[-2][0]:
        predict_visit[u] = mp[-1][1]
    else:
        if poprank[mp[-1][1]] < poprank[mp[-2][1]]:
            predict_visit[u] = mp[-1][1]
        else:
            predict_visit[u] = mp[-2][1]
```

In [ ]:
```
print 'continue Question 5:'
samples = float(0)
correct = float(0)
nosee = 0
for t in valid:
    samples += 1
    u,c  = t['userID'],t['categoryID']
```

```
            if not u in predict_visit.keys():
                predict_cid  = 0
                nosee +=1
            else:
                predict_cid = predict_visit[u]
            if int(c) == int(predict_cid):
                correct += 1
        print 'accuracy: ',correct/samples
```

In [ ]:
```
import string
print 'Question 6:'

def feq_500_word_in_cat(cat):
    wordCount = defaultdict(int)
    punctuation = set(string.punctuation)
    for t in train:
        if int(t['categoryID']) == int(cat):
            r = ''.join([c for c in t['reviewText'].lower() if not c in punctuation])
            for w in r.split():
                wordCount[w] += 1
    counts = [(wordCount[w], w) for w in wordCount]
    counts.sort()
    counts.reverse()
    words500 = [x[1] for x in counts[:500]]
    counts500 = [x[0] for x in counts[:500]]
    return counts[:500],wordCount

def more_feq_in(cat,wordCount,total_count_500):
    print 'in category ',cat
    morefq = list()
    ci,wc = feq_500_word_in_cat(cat)
    wi_500 =  [x[1] for x in ci[:500]]
    ci_500 = [x[0] for x in ci[:500]]
    ci_total500 = sum(ci_500)
    for i in range(500):
        w = wi_500[i]
        c_in_i = wc[w]
        fq_in_i = float(c_in_i)/ci_total500
        c_w_app = wordCount[w]
        fq_w = float(c_w_app)/total_count_500
        morefq.append((fq_in_i - fq_w ,w))
    morefq.sort()
    morefq.reverse()
    print 'the 10 most freq words compare to other category:'
    print morefq[:10]


wordCount = defaultdict(int)
```

```
    punctuation = set(string.punctuation)
    for t in train:
        r = ''.join([c for c in t['reviewText'].lower() if not c in punctuation])
        for w in r.split():
            wordCount[w] += 1
    counts = [(wordCount[w], w) for w in wordCount]
    counts.sort()
    counts.reverse()
    words500 = [x[1] for x in counts[:500]]
    counts500 = [x[0] for x in counts[:500]]
    total_count_500 = sum(counts500)

    for i in range(10):
        more_feq_in(i,wordCount,total_count_500)
```

In [ ]:

In [ ]:
```
from sklearn import svm
from collections import defaultdict
import string

print 'Question 7:'

def featQ7 (t,word500Pos):
    feat = [0]*500
    punctuation = set(string.punctuation)
    r = ''.join([c for c in t['reviewText'].lower() if not c in punctuation])
    for w in r.split():
        feat[word500Pos[w]] = 1
    return feat

wordCount = defaultdict(int)
punctuation = set(string.punctuation)
for t in train:
    r = ''.join([c for c in t['reviewText'].lower() if not c in punctuation])
    for w in r.split():
        wordCount[w] += 1
counts = [(wordCount[w], w) for w in wordCount]
counts.sort()
counts.reverse()
words500 = [x[1] for x in counts[:500]]
word500Pos = defaultdict(int)
for i in range(len(words500)):
    word500Pos[words500[i]] = i
X_train = [featQ7(t,word500Pos) for t in train]
X_valid = [featQ7(t,word500Pos) for t in valid]
y_train = [ 1 == int(t['categoryID'])for t in train]
y_valid = [ int(t['categoryID']) for t in valid]
```

```
        bestc = -1
        bestacc = -1
        print 'data is ready'
        for c in [0.01,0.1,1,10,100]:
            clf = svm.SVC(C=c)
            clf.fit(X_train, y_train)
            valid_pred = clf.predict(X_valid)
            acc = [(pre and val) for (pre,val) in zip(valid_pred,y_valid)]
            correct = sum (acc)
            accuracy = float(correct)/len(acc)
            print 'c = ',c,' with accuracy: ', accuracy
            if accuracy > bestacc:
                bestacc = accuracy
                bestc = c
        print 'c = ',bestc,' has highest accuracy:',bestacc

In [ ]: print 'Question 8:'

        def featQ8(t,word500Pos):
            feat = [0]*500
            punctuation = set(string.punctuation)
            r = ''.join([c for c in t['reviewText'].lower() if not c in punctuation])
            for w in r.split():
                feat[word500Pos[w]] = 1
            return feat

        def train_catsvm(cat,train,c,X_train):
            y_train = [int(t['categoryID']) == int(cat) for t in train]
            clf = svm.SVC(C=c)
            clf.fit(X_train,y_train)
            return clf

        def find_best_c_for_catsvm(tarin,valid,cat,X_train,X_valid):
            y_valid = [int(cat) == int(t['categoryID'])for t in valid]
             #y_valid = [ int(t['categoryID'])for t in valid]
            bestc = -1
            bestacc = -1
            for c in [0.01,0.1,1,10,100]:
                clf = train_catsvm(cat,train,c,X_train)
                valid_pred = clf.predict(X_valid)
                acc = [(pre == val) for (pre,val) in zip(valid_pred,y_valid)]
                correct = sum (acc)
                accuracy = float(correct)/len(acc)
                if accuracy > bestacc:
                    bestacc = accuracy
                    bestc = c

            return bestc
```

```
        wordCount = defaultdict(int)
        punctuation = set(string.punctuation)
        for t in train:
            r = ''.join([c for c in t['reviewText'].lower() if not c in punctuation])
            for w in r.split():
                wordCount[w] += 1
        counts = [(wordCount[w], w) for w in wordCount]
        counts.sort()
        counts.reverse()
        word500 = [x[1] for x in counts[:500]]
        word500Pos = defaultdict(int)
        for i in range(len(words500)):
            word500Pos[words500[i]] = i
        X_train = [featQ8(t,word500Pos) for t in train]
        X_valid = [featQ8(t,word500Pos) for t in valid]
        y_valid_cid = [int(rev['categoryID']) for rev in valid]
        best_c = [find_best_c_for_catsvm(train,valid,cat,X_train,X_valid) for cat in range(10)]
        best_catsvm = [train_catsvm(cat,train,best_c[cat],X_train)for cat in range(10)]
        print 'Data ready'
        correct = flaot(0)
        samples = len(y_valid_cid)
        for i in len(y_valid):
            best_score = -1
            best_guess = -1
            for cat in range(10):
                clf = best_catsvm[cat]
                score = clf.decision_function(X_valid[i])
                if score > best_score:
                    best_score = score
                    best_guess = cat
            if y_valid_cid[i] == best_guess:
                correct += 1
        print 'Accuracy: ',correct/samples
```

In [ ]: