In [1]:
```
import gzip
from collections import defaultdict
import numpy as np
import nltk
import string
def readGz(f):
  for l in gzip.open(f):
    yield eval(l)
```

In [2]:
```
# Read file
data = list(readGz('train.json.gz'))
```

Q1

In [3]:
```
train = data[0:len(data)/2]
validation = data[len(data)/2:len(data)]
buy = [(d['reviewerID'],d['itemID']) for d in validation]
#train set
purtrain = [(d['reviewerID'],d['itemID']) for d in train]
print(len(validation),len(buy),len(purtrain))
```

```
(100000, 100000, 100000)
```

In [5]:
```
# Build dictionary for paired user and items
buypair = {}
user = []
item = []
for d in data:
    if d['reviewerID'] in buypair and (d['itemID'] not in buypair[d['reviewerI
D']]):
        buypair[d['reviewerID']].append(d['itemID'])
    elif d['reviewerID'] not in buypair:
        buypair[d['reviewerID']] = [d['itemID']]
    user.append(d['reviewerID'])
    item.append(d['itemID'])
# Delete the repeated users and items
user = list(set(user))
item = list(set(item))
print(len(user),len(item),len(buy))
```

```
(39239, 19914, 100000)
```

In [6]:
```
# Construct not purchased pair of length 100000
import random
notbuypair = []
for i in range(len(data)):
    u = random.choice(user)
    it = random.choice(item)
    if (it in buypair[u]) == False:
        notbuypair.append((u,it))
    if len(notbuypair) == len(data)/2:
        break
```

In [7]:
```
# Construct the validation set
validation = buy+notbuypair
print(len(validation),len(buy),len(notbuypair))
```

```
(200000, 100000, 100000)
```

In [8]:
```
### Would-purchase baseline: just rank which businesses are popular and which
 are not, and return '1' if a business is among the top-ranked
businessCount = defaultdict(int)
totalPurchases = 0
# Use train set i.e. first half of the original dataset
for d in purtrain:
    businessCount[d[1]] += 1
totalPurchases = len(purtrain);
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/1.94:
        print(ic,i)
        break

predictions = open("predictions_Purchase.txt", 'w')
for u,i in validation:
    if i in return1:
        predictions.write(u + '-' + i + ",1\n")
    else:
        predictions.write(u + '-' + i + ",0\n")

predictions.close()
print(len(return1))
```

```
(7, 'I024467446')
3935
```

In [9]:
```
acc = 0
total = 0
check = []
q = 0
for l in open("predictions_Purchase.txt"):
    if l.startswith("reviewerID"):
        continue
    pair,pre = l.strip().split(',')
    u,i = pair.strip().split('-')
    corr = -1
    # If the user never buy item, we assume that he/she will not buy new item
    if u not in buypair:
        corr = 0
        q +=1
    elif (i in buypair[u]) == True:
        corr = 1
    else:
        corr = 0
    total += 1
    if corr == int(pre):
        acc+=1
    check.append((corr,int(pre)))
acc = acc/(total*1.0)
print(acc,total,q)
```

```
(0.62877, 200000, 0)
```

Q2

We observe that when the threshold of total purchses is very low or very high, the accuracy will be around 0.5. For instance, totalPurchases/2, acc = 0.628425; but if totalPurchases/200, acc = 0.50234; totalPurchases/1, acc = 0.502485. Actually, when totalPurchases/1.94, we get the highest acc=0.629055. As 1.94 is very close to 2, we can say that threshod around 50% will give us the best performance. This is caused by the special sturcture of validation set, the accuracy for the first half will be very high if we regard all of the items as popular, which means the classifier will tend to predict positive labels. The second half is randomly selected unpurchased pairs, and the accuracy on the second half will be higher if less items are popular, which means the classifier tend to predict 0 labels. When the threshold is around 50%, the classifier is more balanced and thus gives the best performance.

Our observation of true positive rate and true negative rate could be a reference for the choice of the best threshold. We observe that our most popular model get a very low true positive rate and a high true negative rate when we define a small ratio of purchases to be popular. since the accuracy on the randomly constructed second half is better. e.g. tnr, tpr = 0.99944, 0.01039 if we define the most popular items in 1/100 of purchases to be popular. and a very high true positive rate and a low true negative rate when we define a large ratio of purchases to be popular. since the accuracy on the first half is better. e.g. tnr, tpr = 0.027, 0.97797 if we define all of the items in total purchases to be popular.

In [10]:
```
ch = [0,0,0,0]
for d in check:
    # FN
    if d[0] == 1 and d[1]==0:
        ch[0] +=1
    # FP
    elif d[0] == 0 and d[1] == 1:
        ch[1] +=1
    # TP
    elif d[0] == 1 and d[1] ==1:
        ch[2] +=1
    #TN
    else:
        ch[3] +=1
tnr = ch[3]/((ch[3]+ch[1])*1.0)
tpr = ch[2]/((ch[2]+ch[0])*1.0)
print(ch,tnr,tpr)
```

```
([54267, 19979, 45733, 80021], 0.80021, 0.45733)
```

Q3

In [11]:
```
# Construct dict of the whole train data (200000) of users-categories or items
-categories
cateU={}
for d in data:
    if d['reviewerID'] in cateU and (d['categoryID'] not in cateU[d['reviewerI
D']]):
        cateU[d['reviewerID']].append(d['categoryID'])
    elif d['reviewerID'] not in cateU:
        cateU[d['reviewerID']] = [d['categoryID']]

cateI={}
#As the length of mul is 0, we are sure that each item only corresponds to one
 category
mul = []
for d in data:
    if d['itemID'] in cateI and d['categoryID'] != cateI[d['itemID']]:
        mul.append(d['itemID'])
    elif d['itemID'] not in cateI:
        cateI[d['itemID']] = d['categoryID']
print(len(mul))
```

```
0
```

In [13]:
```
def predictions_Purchase_categories(user,item):
    # If this item has never been purchased or the user is new, we randomly
    # true or false. Assume the buy and not buy probability are equal in this
 case.
    if item not in cateI or user not in cateU:
        return np.random.randint(2)
    elif cateI[item] in cateU[user]:
        return True
    else:
        return False
```

Q4 Kaggle Username: mollyyyyy; Email: xil397@ucsd.edu

Here for improving accuracy we combine the most popular and categories model. As we observed that the most popular model has a high true negative rate at its best accuracy, we tend to trust the most popular classifier when these two classifiers has different results and the most popular model predict 0. Conversely, since the most popular model has a moderate true positive rate i.e.<0.5 at its best accuracy, we tend to trust the categories classifier when these two classifiers has different results and the most popular model predict 1. The result for this classifier is 0.63714.

In [14]:
```
predictions = open("predictions_Purchase.txt", 'w')
for l in open("pairs_Purchase.txt"):
    if l.startswith("reviewerID"):
        predictions.write(l)
        continue
    u,i = l.strip().split('-')
    # predict 1 if both models predict 1
    if predictions_Purchase_categories(u,i) == True and i in return1:
        predictions.write(u + '-' + i +",1\n")
    # predict 0 if both models predict 0
    elif predictions_Purchase_categories(u,i) == False and i not in return1:
        predictions.write(u + '-' + i + ",0\n")
    else:
        predictions.write(u + '-' + i + ",0\n")
predictions.close()
```

Q5

In [310]:
```
catego = [d for d in data if 'categoryID' in d]
traincat = catego[0:len(catego)/2]
validcat = catego[len(catego)/2:len(catego)]
print(len(catego),len(traincat),len(validcat))
```

```
(200000, 100000, 100000)
```

In [311]:
```
# Construct dictionary for the globally most frequently bought categories
catdict = defaultdict(int)
for d in data:
    catdict[d['categoryID']] += 1
print(catdict)
```

```
defaultdict(<type 'int'>, {0: 141398, 1: 51416, 2: 2329, 3: 1881, 4: 2976})
```

In [312]:
```
# Define function to get a list of tuples containing a specific element
def search_tuple(tups, elem):
        return filter(lambda tup: elem in tup, tups)
```

In [313]:
```
# Construct purchased categories dictionary for each user in training set
usrcat = defaultdict(list)
for d in traincat:
    # If user not in this dict, add the pair(category,category count)
    if d['reviewerID'] not in usrcat:
        usrcat[d['reviewerID']].append((d['categoryID'],1))
    else:
        fil = search_tuple(usrcat[d['reviewerID']],d['categoryID'])
        # If user in this dict but does not contain this category, add the pai
r(category,category count)
        if len(fil) == 0:
            usrcat[d['reviewerID']].append((d['categoryID'],1))
        else:
            # If user in this dict and contains this category
            # update the pair(category,category count)
            ca,count = fil[0]
            count = count +1
            # Increment count for exsiting categories
            for i in range(len(usrcat[d['reviewerID']])):
                caa,cou = usrcat[d['reviewerID']][i]
                if caa == ca:
                    li = list(usrcat[d['reviewerID']][i])
                    li[1] = count
                    usrcat[d['reviewerID']][i] = tuple(li)
```

In [314]:
```
# Dictionary store most frequently purchased categories
sortdict = defaultdict(str)
for usr, calist in usrcat.iteritems():
    # If the user just bought one category, this is the most popular one
    if len(calist) == 1:
        sortdict[usr] = calist[0][0]
    else:
        # Sort the category by count in descending order
        calist = sorted(calist, key=lambda x: x[1],reverse=True)
        tie = search_tuple(calist, calist[0][1])
        # If there is no tie, choose the category with the most count
        if len(tie) == 1:
            sortdict[usr] = calist[0][0]
        else:
            cat = (-1,-1)
            # Compare global popularity in the tie case and choose the most gl
obally popular
            for tupca,tupco in tie:
                if catdict[tupca] > cat[1]:
                    cat = list(cat)
                    cat[0] = tupca
                    cat[1] = catdict[tupca]
                    cat = tuple(cat)
            sortdict[usr] = cat[0]
```

In [315]:
```
predictions = open("predictions_Categories.txt", 'w')
for d in validcat:
    r = d['reviewerID']
    c = d['categoryID']
    i = d['itemID']
    # If this is a new user, return 0
    if r not in sortdict:
        predictions.write( i+ '-' + str(c) + ",0\n")
    else:
        # Write the predict and correct category ID
        s = ","+str(sortdict[r])+"\n"
        predictions.write( i+ '-' + str(c) + s)
predictions.close()
```

In [316]:
```
# Compute accuracy
acc = 0
total = 0
for l in open("predictions_Categories.txt"):
    pair,pre = l.strip().split(',')
    i,c = pair.strip().split('-')
    total += 1
    if c == pre:
        acc+=1
acc = acc/(total*1.0)
print(acc,total)
```

```
(0.79957, 100000)
```

Q6

In [317]:
```
# Build dataset for each category
text = [d['reviewText'] for d in traincat if 'reviewText' in d]
textwomen = [d['reviewText'] for d in traincat if d['categoryID'] == 0]
textmen = [d['reviewText'] for d in traincat if d['categoryID'] == 1]
textgirl = [d['reviewText'] for d in traincat if d['categoryID'] == 2]
textboy = [d['reviewText'] for d in traincat if d['categoryID'] == 3]
textbaby = [d['reviewText'] for d in traincat if d['categoryID'] == 4]
print(len(text),len(textwomen),len(textmen),len(textgirl),len(textboy),len(tex
tbaby))
```

```
(100000, 70878, 25512, 1178, 925, 1507)
```

In [318]:
```
# Remove punctuation and capitalization
wordCount = defaultdict(int)
punctuation = set(string.punctuation)
for t in text:
    r = ''.join([c for c in t.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()
# Get the most 500 popular words
words = [x[1] for x in counts[:500]]
```

In [319]:
```
# Count number of times the word appears in each category
wordCountwomen = defaultdict(int)
wordCountmen = defaultdict(int)
wordCountgirl = defaultdict(int)
wordCountboy = defaultdict(int)
wordCountbaby = defaultdict(int)
for t in textwomen:
    r = ''.join([c for c in t.lower() if not c in punctuation])
    for w in r.split():
        if w in words:
            wordCountwomen[w] +=1
for t in textmen:
    r = ''.join([c for c in t.lower() if not c in punctuation])
    for w in r.split():
        if w in words:
            wordCountmen[w] +=1
for t in textgirl:
    r = ''.join([c for c in t.lower() if not c in punctuation])
    for w in r.split():
        if w in words:
            wordCountgirl[w] +=1
for t in textboy:
    r = ''.join([c for c in t.lower() if not c in punctuation])
    for w in r.split():
        if w in words:
            wordCountboy[w] +=1
for t in textbaby:
    r = ''.join([c for c in t.lower() if not c in punctuation])
    for w in r.split():
        if w in words:
            wordCountbaby[w] +=1
```

In [320]:
```
# Frequency overall of 500 words in trainset
freq = defaultdict(float)
totalocc = 0
for w in words:
    freq[w] = wordCount[w]
    totalocc += wordCount[w]
for key in freq:
    freq[key] = freq[key]/(1.0*totalocc)
print(len(freq))
```

```
500
```

In [321]:
```
# Frequency of 500 words in trainset per category
freqwomen = defaultdict(float)
totaloccwomen = 0
freqm = defaultdict(float)
totaloccm = 0
freqg = defaultdict(float)
totaloccg = 0
freqb = defaultdict(float)
totaloccb = 0
freqba = defaultdict(float)
totaloccba = 0
for w in words:
    freqwomen[w] = wordCountwomen[w]
    totaloccwomen += wordCountwomen[w]
    freqm[w] = wordCountmen[w]
    totaloccm += wordCountmen[w]
    freqg[w] = wordCountgirl[w]
    totaloccg += wordCountgirl[w]
    freqb[w] = wordCountboy[w]
    totaloccb += wordCountboy[w]
    freqba[w] = wordCountbaby[w]
    totaloccba += wordCountbaby[w]
for key in freqwomen:
    freqwomen[key] = freqwomen[key]/(1.0*totaloccwomen)
for key in freqm:
    freqm[key] = freqm[key]/(1.0*totaloccm)
for key in freqg:
    freqg[key] = freqg[key]/(1.0*totaloccg)
for key in freqb:
    freqb[key] = freqb[key]/(1.0*totaloccb)
for key in freqba:
    freqba[key] = freqba[key]/(1.0*totaloccba)
```

In [322]:
```
# find which words are more popular in one category compared to their overall
 frequency across all categories
popw=[(freqwomen[w]-freq[w],w) for w in words]
popm=[(freqm[w]-freq[w],w) for w in words]
popg=[(freqg[w]-freq[w],w) for w in words]
popb=[(freqb[w]-freq[w],w) for w in words]
popba=[(freqba[w]-freq[w],w) for w in words]
popw.sort()
popm.sort()
popg.sort()
popb.sort()
popba.sort()
popw.reverse()
popm.reverse()
popg.reverse()
popb.reverse()
popba.reverse()
```

In [331]:
```
print('10 words that are more frequent in that category compared to other cate
gories ')
print("Women: ",[d[1] for d in popw[0:10]])
print("Men: ",[d[1] for d in popm[0:10]])
print("Girls: ",[d[1] for d in popg[0:10]])
print("Boys: ",[d[1] for d in popb[0:10]])
print("Baby: ",[d[1] for d in popba[0:10]])
```

```
10 words that are more frequent in that category compared to other categories

('Women: ', ['i', 'it', 'love', 'bra', 'wear', 'but', 'so', 'was', 'cute', 's
ize'])
('Men: ', ['he', 'watch', 'the', 'for', 'of', 'good', 'husband', 'these', 'hi
s', 'you'])
('Girls: ', ['she', 'her', 'daughter', 'for', 'my', 'old', 'it', 'year', 'lov
es', 'we'])
('Boys: ', ['he', 'son', 'for', 'my', 'old', 'him', 'his', 'year', 'we', 'lov
es'])
('Baby: ', ['for', 'they', 'these', 'are', 'her', 'cute', 'old', 'my', 'littl
e', 'she'])
```

Q7

In [325]:
```
# Subsample the trainset due to the limitation of computation device and time
subtrain = list()
l = len(traincat)/4
for i in range(l):
    s = random.choice(traincat)
    subtrain.append(s)
print(len(subtrain))
```

```
25000
```

In [326]:
```
# Build feature matrix X and target vector Y for Women's clothing and non-Wome
n's clothing
X = list()
Y = list()
for d in subtrain:
    text = d['reviewText']
    xx = np.zeros(500)
    for t in text.split():
        if t in words:
            xx[words.index(t)] = 1
    X.append(xx)
    if d['categoryID'] == 0:
        Y.append(1)
    else:
        Y.append(0)
```

In [327]:
```
# Build validation matrix to be predicted by SVM
validd = list()
validY = list()
for d in validcat:
    text = d['reviewText']
    v = np.zeros(500)
    for t in text.split():
        if t in words:
            v[words.index(t)] = 1
    validd.append(v)
    if d['categoryID'] == 0:
        validY.append(1)
    else:
        validY.append(0)
```

In [328]:
```
X = np.array(X)
Y = np.array(Y)
validd = np.array(validd)
validY = np.array(validY)
print(X.shape,Y.shape,validd.shape,validY.shape)
```

```
((25000L, 500L), (25000L,), (100000L, 500L), (100000L,))
```

In [329]:
```
# Define accuracy function by comparing the predict and correct vector
def acc(pred,corr):
    acc = 0
    for i in range(len(pred)):
        if pred[i] == corr[i]:
            acc+=1
    acc = acc/(len(corr)*1.0)
    return acc
```

In [34]:
```
from sklearn.svm import LinearSVC
#subsample 10% - best performance 0.7674 if c = 0.01
for i in [0.01,0.1,1,10,100]:
    clf = LinearSVC(C=i,max_iter=1000000)
    clf.fit(X, Y)
    pred = clf.predict(validd)
    print(acc(pred,validY))
```

```
0.7674
0.76629
0.76532
0.76522
0.76525
```

In [761]:
```
#subsample 20%  - best performance 0.77356 if c = 0.01
for i in [0.01,0.1,1,10,100]:
    clf = LinearSVC(C=i,max_iter=1000000)
    clf.fit(X, Y)
    pred = clf.predict(validd)
    print(acc(pred,validY))
```

```
0.77356
0.77354
0.77333
0.77333
0.77334
```

In [330]:
```
# subsample 25%  - best performance 0.77366, 0.77365 if c = 100 or 0.1
# The high accuracy 0.77366, 0.77364 at c=100, c=10 might due to the computati
on limit, so we say
# both c = 100, 10 and c = 0.1 perform well
for i in [0.01,0.1,1,10,100]:
    clf = LinearSVC(C=i,max_iter=1000000)
    clf.fit(X, Y)
    pred = clf.predict(validd)
    print(acc(pred,validY))
```

```
0.77345
0.77365
0.77352
0.77364
0.77366
```

Q8

In [273]:
```
# Construct target vector Y for all 5 categories
Ym = list()
Yg = list()
Yb = list()
Yba = list()
for d in subtrain:
    if d['categoryID'] == 1:
        Ym.append(1)
        Yg.append(0)
        Yb.append(0)
        Yba.append(0)
    elif d['categoryID'] == 2:
        Ym.append(0)
        Yg.append(1)
        Yb.append(0)
        Yba.append(0)
    elif d['categoryID'] == 3:
        Ym.append(0)
        Yb.append(1)
        Yg.append(0)
        Yba.append(0)
    elif d['categoryID'] == 4:
        Ym.append(0)
        Yba.append(1)
        Yg.append(0)
        Yb.append(0)
    else:
        Ym.append(0)
        Yba.append(0)
        Yg.append(0)
        Yb.append(0)
```

In [274]:
```
print(len(subtrain),len(Ym),len(Yg),len(Yb),len(Yba))
```

```
(25000, 25000, 25000, 25000, 25000)
```

In [275]:
```
Ym = np.array(Ym)
Yg = np.array(Yg)
Yb = np.array(Yb)
Yba = np.array(Yba)
```

In [276]:
```
# Train a large classifier for each c value
# A large classifier has 5 sub classifiers for all of the categories
#subsample 10%, 20%,25%
clfarr = list()
for y in [Y,Ym,Yg,Yb,Yba]:
    clf = LinearSVC(C=0.01,max_iter=1000000)
    clf.fit(X, y)
    clfarr.append(clf)
```

In [277]:
```
#subsample 10%,20%,25%
clfarr2 = list()
for y in [Y,Ym,Yg,Yb,Yba]:
    clf = LinearSVC(C=0.1,max_iter=1000000)
    clf.fit(X, y)
    clfarr2.append(clf)
```

In [278]:
```
#subsample 10%,20%,25%
clfarr3 = list()
for y in [Y,Ym,Yg,Yb,Yba]:
    clf = LinearSVC(C=1,max_iter=1000000)
    clf.fit(X, y)
    clfarr3.append(clf)
```

In [279]:
```
#subsample 10%,20%,25%
clfarr4 = list()
for y in [Y,Ym,Yg,Yb,Yba]:
    clf = LinearSVC(C=10,max_iter=1000000)
    clf.fit(X, y)
    clfarr4.append(clf)
```

In [280]:
```
#subsample 10%,20%,25%
clfarr5 = list()
for y in [Y,Ym,Yg,Yb,Yba]:
    clf = LinearSVC(C=100,max_iter=1000000)
    clf.fit(X, y)
    clfarr5.append(clf)
```

In [281]:
```
# Build confidence array for all of the c values classifiers groups
confarr = list()
for clf in clfarr:
    conf = clf.decision_function(validd)
    confarr.append(conf)
```

In [282]:
```
confarr2 = list()
for clf in clfarr2:
    conf = clf.decision_function(validd)
    confarr2.append(conf)
```

In [283]:
```
confarr3 = list()
for clf in clfarr3:
    conf = clf.decision_function(validd)
    confarr3.append(conf)
```

In [284]:
```
confarr4 = list()
for clf in clfarr4:
    conf = clf.decision_function(validd)
    confarr4.append(conf)
```

In [285]:
```
confarr5 = list()
for clf in clfarr5:
    conf = clf.decision_function(validd)
    confarr5.append(conf)
```

In [286]:
```
# Measure the performance of large classifiers with different c values
predictions = open("predictions_Categories.txt", 'w')
acc = np.zeros(5)
arr = [confarr,confarr2,confarr3,confarr4,confarr5]
for i in range(len(validcat)):
    l = validcat[i]
    for k in range(len(arr)):
        conf = arr[k]
        precat = -1
        maxconf = -100
        for j in range(5):
            if conf[j][i] > maxconf:
                maxconf = confarr[j][i]
                precat = j
        if int(precat) == int(l['categoryID']):
            acc[k]+=1
acc = acc/(1.0*len(validcat))
print("c=0.01 c=0.1 c=1 c=10 c=100")
print(acc)
```

```
c=0.01 c=0.1 c=1 c=10 c=100
[0.71684 0.71681 0.71681 0.71682 0.71682]
```

Subsample 10%: [c=0.01 c=0.1 c=1 c=10 c=100] , [0.71679 0.71681 0.71664 0.71661 0.7166]. The performance at c=0.1 is the best. Subsample 20%: [c=0.01 c=0.1 c=1 c=10 c=100], [0.71682 0.71681 0.71678 0.71679 0.71679]. The performance at c=0.01 is the best Subsample 25%: [c=0.01 c=0.1 c=1 c=10 c=100], [0.71684 0.71681 0.71681 0.71682 0.71682]. The performance at c=0.01 is the best

In [287]:
```
### Category prediction baseline: Just consider some of the most common words
 from each category

catDict = {
  "Women": 0,
  "Men": 1,
  "Girls": 2,
  "Boys": 3,
  "Baby": 4
}
# For each reviewtext in test dataset, predict its category by choosing the hi
ghest confidence score
# among the 5 large classifiers (25 sub classifiers in total)
predictions = open("predictions_Category.txt", 'w')
predictions.write("reviewerID-reviewHash,category\n")
for l in readGz("test_Category.json.gz"):
    text = l['reviewText'].lower()
    xx = np.zeros(500)
    for t in text.split():
        if t in words:
            xx[words.index(t)] = 1
    xx = np.array(xx).reshape(1,500)
    maxper = -100
    catpred = -1
    for clfar in [clfarr,clfarr2,clfarr3,clfarr4,clfarr5]:
        for i in range(len(clfar)):
            m = clfar[i].decision_function(xx)
            if m > maxper:
                maxper = m
                catpred = i

    cat = catpred
    predictions.write(l['reviewerID'] + '-' + l['reviewHash'] + "," + str(cat)
 + "\n")
predictions.close()
```