In [1]:
```
import gzip
import numpy as np
import random
import string
import nltk
from sklearn import svm
from collections import defaultdict
```

In [2]:
```
def readGz(f):
  for l in gzip.open(f):
    yield eval(l)
```

In [3]:
```
# Visit prediction
# 1
```

In [4]:
```
# 100,000 sets for training and another 100,000 for validation
businessCount = defaultdict(int)
totalPurchases = 0
visitVal = []

for l in readGz("train.json.gz"):
    if (totalPurchases < 100000):
        user,business = l['userID'],l['businessID']
        businessCount[business] += 1
        totalPurchases += 1
    else:
        user,business = l['userID'],l['businessID']
        visitVal.append((user, business, 1))
```

In [5]:
```
# create 100,000 more non-visited data points


visited = set()
uniqueUser = set()
uniqueBuss = set()

for u,b,c in visitVal:
    uniqueUser.add(u)
    uniqueBuss.add(b)
    visited.add((u,b))

for l in range(0,100000):
    user = random.sample(uniqueUser, 1)[0]
    business = random.sample(uniqueBuss, 1)[0]
    while ((user,business) in visited):
        user = random.sample(uniqueUser, 1)[0]
        business = random.sample(uniqueBuss, 1)[0]
    visitVal.append((user,business, 0))
```

In [6]:
```
len(visitVal)
```

Out[6]: 200000

In [7]:
```
mostPopular = [(businessCount[x], x) for x in businessCount]
mostPopular.sort()
mostPopular.reverse()
```

In [8]:
```
return1 = set()
count = 0
for ic, i in mostPopular:
    count += ic
    return1.add(i)
    if count > totalPurchases/2: break

correct = 0
index = 0

for u,b,c in visitVal:
    if (((b in return1) and (c==1)) or ((b not in return1) and (c==0))) :
        correct += 1
    index += 1

accuracy = float(correct)/len(visitVal)
print accuracy
```

```
0.623225
```

In [9]:
```
#2
```

In [10]:
```
# Try 75th percentile
return1 = set()
count = 0
for ic, i in mostPopular:
    count += ic
    return1.add(i)
    if count > totalPurchases/4: break

correct = 0
index = 0

for u,b,c in visitVal:
    if (((b in return1) and (c==1)) or ((b not in return1) and (c==0))) :
        correct += 1
    index += 1

accuracy = float(correct)/len(visitVal)
print accuracy
```

```
0.58495
```

In [11]:
```
# Try 25th percentile

return1 = set()
count = 0
for ic, i in mostPopular:
    count += ic
    return1.add(i)
    if count > 3*totalPurchases/4: break

correct = 0
index = 0

for u,b,c in visitVal:
    if (((b in return1) and (c==1)) or ((b not in return1) and (c==0))) :
        correct += 1
    index += 1

accuracy = float(correct)/len(visitVal)
print accuracy
```

```
0.607935
```

In [12]:
```
# Try 40th percentile

return1 = set()
count = 0
for ic, i in mostPopular:
    count += ic
    return1.add(i)
    if count > 6*totalPurchases/10: break

correct = 0
index = 0

for u,b,c in visitVal:
    if (((b in return1) and (c==1)) or ((b not in return1) and (c==0))) :
        correct += 1
    index += 1

accuracy = float(correct)/len(visitVal)
print accuracy
```

```
0.62487
```

In [13]:
```
# Try 60th percentile

return1 = set()
count = 0
for ic, i in mostPopular:
    count += ic
    return1.add(i)
    if count > 4*totalPurchases/10: break

correct = 0
index = 0

for u,b,c in visitVal:
    if (((b in return1) and (c==1)) or ((b not in return1) and (c==0))) :
        correct += 1
    index += 1

accuracy = float(correct)/len(visitVal)
print accuracy
```

```
0.613105
```

In [14]:
```
# It seems that 50th percentile gives the highest acuracy and although the 40t
h percentile test give slightly better in my case,
# I believe this is due to random chance. I think it is the best because
```

In [15]:
```
# 3
# Users may tend to repeatedly visit business of the same type. Build a baseli
ne that returns ‘True’ if
# a user has visited a business of the same category before (at least one cate
gory in common), or zero
# otherwise
```

In [16]:
```
# 100,000 sets for training and another 100,000 for validation
businessCount = defaultdict(int)
totalPurchases = 0
visitVal = []
userCats = {}

for l in readGz("train.json.gz"):
    if (totalPurchases < 100000):
        user,business,categories = l['userID'],l['businessID'],l['categories']
        userCats[user] = list(set(userCats.get(user,[])).union(categories))

    else:
        user,business,categories = l['userID'],l['businessID'],l['categories']
        visitVal.append((user, business, categories, 1))

    totalPurchases += 1
```

In [17]:
```
userCats['U600574911']
```

Out[17]:
```
[u'Donut Shop',
 u'European Restaurant',
 u'Chinese Restaurant',
 u'Middle Eastern Restaurant',
 u'Dessert Shop',
 u'Asian Restaurant',
 u'Ice Cream Shop']
```

In [18]:
```
# create 100,000 more non-visited data points

visited = set()
uniqueUser = set()
uniqueBuss = set()
bussCat = {}

for u,b,c,d in visitVal:
    uniqueUser.add(u)
    uniqueBuss.add(b)
    visited.add((u,b))
    bussCat[b] = c
```

In [19]:
```
for l in range(0,100000):
    user = random.sample(uniqueUser, 1)[0]
    bussiness = random.sample(uniqueBuss, 1)[0]

    while ((user,business) in visited):
        user = random.sample(uniqueUser, 1)[0]
        business = random.sample(uniqueBuss, 1)[0]

    visitVal.append((user,business, bussCat[business], 0))
```

In [20]:
```
visitVal[100]
```

Out[20]:
```
('U237829706', 'B556394301', [u'Restaurant', u'Soul Food Restaurant'], 1)
```

In [21]:
```
visitVal[100001]
```

Out[21]:
```
('U296072251', 'B589288198', [u'American Restaurant'], 0)
```

In [22]:
```
# Test the accuracy

predicts = []
for user,business,cat,d in visitVal:
    cat1 = userCats.get(user,[])

    intersection = list(set(cat1) & set(cat))

    if intersection:
        predicts.append(1)
    else:
        predicts.append(0)

accuracy = [(a[3]==b) for (a,b) in zip(visitVal,predicts)]
print 1.0*sum(accuracy)/len(accuracy)
```

```
0.62923
```

In [23]:
```
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('-')
    cat1 = userCats.get(u,[])
    cat2 = bussCat.get(i,[])
    intersection = list(set(cat1) & set(cat2))

    if intersection:
        predictions.write(u + '-' + i + ",1\n")
    else:
        predictions.write(u + '-' + i + ",0\n")

predictions.close()
```

In [24]:
```
# 4
# I uploaded my solution to Kaggle and got a score of 0.64930
# My user name is ddinata on kaggle
```

In [25]:
```
# Category prediction
# 5
```

In [26]:
```
hasCat = []

for l in readGz("train.json.gz"):
    if 'categoryID' in l:
        hasCat.append(l)
```

In [27]:
```
trainCat = hasCat[:len(hasCat)/2]
valCat = hasCat[len(hasCat)/2:]
```

In [28]:
```
print len(trainCat)
print len(valCat)
print trainCat[:5]
```

```
35097
35098
[{'rating': 4.0, 'reviewHash': 'R567271252', 'businessID': 'B423621081', 'uni
xReviewTime': 1378865039, 'reviewText': u"I'm a vegetarian, but every so ofte
n I want a hotdog with lots of toppings.  And a tall can of beer.  Frank has
got that covered.  And they have a cool warehouse space with some pinball mac
hines.  Prices are a little high for hotdogs...fancy hotdogs, but hotdogs non
etheless.  Good location and service, but gets crowded and loud.", 'userID':
'U985379327', 'reviewTime': u'Sep 10, 2013', 'categories': [u'American Restau
rant', u'Cafe', u'Hot Dog Restaurant'], 'categoryID': 0}, {'rating': 4.0, 're
viewHash': 'R985248711', 'businessID': 'B734024511', 'unixReviewTime': 137297
7506, 'reviewText': u"Asia Cafe is hands down the best Chinese food in Austi
n. Their menu has about 100 different options and it's really authentic. The
spicy fish and the garlic pork are two of my favorites. It's a bit far from d
owntown Austin, right on the outskirts of Round Rock, but it's worth the driv
e.", 'userID': 'U272385455', 'reviewTime': u'Jul 4, 2013', 'categories': [u'C
hinese Restaurant', u'Asian Restaurant'], 'categoryID': 2}, {'rating': 4.0,
'reviewHash': 'R250266072', 'businessID': 'B276843680', 'unixReviewTime': 126
2822400, 'reviewText': u'the name says it all!', 'userID': 'U258716760', 'rev
iewTime': u'Jan 6, 2010', 'categories': [u'Bar'], 'categoryID': 1}, {'ratin
g': 5.0, 'reviewHash': 'R526444776', 'businessID': 'B929679987', 'unixReviewT
ime': 1393117550, 'reviewText': u'The deep fried burger is a must try. Defini
tely the best burger I have had ever. Tasty tasty tasty.', 'userID': 'U794428
800', 'reviewTime': u'Feb 22, 2014', 'categories': [u'Eastern European Restau
rant', u'European Restaurant', u'Delivery Restaurant'], 'categoryID': 3}, {'r
ating': 4.0, 'reviewHash': 'R355737687', 'businessID': 'B006984008', 'unixRev
iewTime': 1355291729, 'reviewText': u'Great happy hour spot.  Stuffed olives
with Tequila are really great and unique.', 'userID': 'U827182046', 'reviewTi
me': u'Dec 11, 2012', 'categories': [u'Mexican Restaurant', u'Latin American
Restaurant', u'Nuevo Latino Restaurant'], 'categoryID': 6}]
```

In [29]:
```
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
}

userCats = {}
for l in trainCat:
    user,business,categoryID = l['userID'],l['businessID'],l['categoryID']
    if user not in userCats:
        userCats[user] = {}
    if categoryID not in userCats[user]:
        userCats[user][categoryID] = 0
    userCats[user][categoryID] += 1
```

In [30]:
```
userCats[random.sample(userCats, 1)[0]]
```

Out[30]:
```
{0: 1}
```

In [31]:
```
max(userCats['U156843408'], key=userCats['U156843408'].get)
```

Out[31]:
```
2
```

In [32]:
```
predicts = []
for l in valCat:
    if l['userID'] not in userCats:
        predicts.append(0)
        continue
    prediction = max(userCats[l['userID']], key=userCats[l['userID']].get)
    predicts.append(prediction)

accuracy = [(a['categoryID']==b) for (a,b) in zip(valCat,predicts)]
print 1.0*sum(accuracy)/len(accuracy)
```

```
0.292039432446
```

In [33]:
```
#6
```

In [34]:
```
wordCount = defaultdict(int)
punctuation = set(string.punctuation)
stemmer = nltk.stem.porter.PorterStemmer()
for l in trainCat:
    for w in l['reviewText'].split():
        w = ''.join([c for c in w.lower() if not c in punctuation])
        w = stemmer.stem(w)
        wordCount[w] += 1
print len(wordCount)
```

```
26833
```

In [35]:
```
counts = [(wordCount[w], w) for w in wordCount]
counts.sort()
counts.reverse()

# words contain the 500 most popular words in the entire trainCat
words = [x[1] for x in counts [:500]]

sum500 = sum(wordCount[w] for w in words)
print sum500
```

```
1107503
```

In [36]:
```
frequency = {}

for w in words:
    frequency[w] = 1.0*wordCount[w]/sum500
```

In [37]:
```
# Make sets for each category from the train set

americanTrain = []
barTrain = []
asianTrain = []
europeanTrain = []
italianTrain = []
fastTrain = []
mexicanTrain = []
seafoodTrain = []
coffeeTrain = []
sandwichTrain = []

americanfreq = {}
barfreq = {}
asianfreq = {}
europeanfreq = {}
italianfreq = {}
fastfreq = {}
mexicanfreq = {}
seafoodfreq = {}
coffeefreq = {}
sandwichfreq = {}

for l in trainCat:
    if l['categoryID'] == 0:
        americanTrain.append(l)
    if l['categoryID'] == 1:
        barTrain.append(l)
    if l['categoryID'] == 2:
        asianTrain.append(l)
    if l['categoryID'] == 3:
        europeanTrain.append(l)
    if l['categoryID'] == 4:
        italianTrain.append(l)
    if l['categoryID'] == 5:
        fastTrain.append(l)
    if l['categoryID'] == 6:
        mexicanTrain.append(l)
    if l['categoryID'] == 7:
        seafoodTrain.append(l)
    if l['categoryID'] == 8:
        coffeeTrain.append(l)
    if l['categoryID'] == 9:
        sandwichTrain.append(l)
```

In [38]:
```
def freqCat(_train, _freq):
    _wordCount = defaultdict(int)
    punctuation = set(string.punctuation)
    stemmer = nltk.stem.porter.PorterStemmer()
    for l in _train:
        for w in l['reviewText'].split():
            w = ''.join([c for c in w.lower() if not c in punctuation])
            w = stemmer.stem(w)
            _wordCount[w] += 1

    _sum500 = sum(_wordCount[w] for w in words)
    print _sum500

    for w in words:
        _freq[w] = 1.0*_wordCount[w]/_sum500
```

In [39]:
```
freqCat(americanTrain, americanfreq)
```

```
339375
```

In [40]:
```
freqCat(barTrain, barfreq)
```

```
144194
```

In [41]:
```
freqCat(asianTrain, asianfreq)
```

```
234667
```

In [42]:
```
freqCat(europeanTrain, europeanfreq)
```

```
53876
```

In [43]:
```
freqCat(italianTrain, italianfreq)
```

```
32612
```

In [44]:
```
freqCat(fastTrain, fastfreq)
```

```
31500
```

In [45]:
```
freqCat(mexicanTrain, mexicanfreq)
```

```
107397
```

In [46]:
```
freqCat(seafoodTrain, seafoodfreq)
```

```
60175
```

In [47]:
```
freqCat(coffeeTrain, coffeefreq)
```

```
74090
```

In [48]:
```
freqCat(sandwichTrain, sandwichfreq)
```

```
29617
```

In [49]:
```
diffamericanfreq = {}
diffbarfreq = {}
diffasianfreq = {}
diffeuropeanfreq = {}
diffitalianfreq = {}
difffastfreq = {}
diffmexicanfreq = {}
diffseafoodfreq = {}
diffcoffeefreq = {}
diffsandwichfreq = {}

for w in words:
    diffamericanfreq[w] = americanfreq[w] - frequency[w]
    diffbarfreq[w] = barfreq[w] - frequency[w]
    diffasianfreq[w] = asianfreq[w] - frequency[w]
    diffeuropeanfreq[w] = europeanfreq[w]- frequency[w]
    diffitalianfreq[w] = italianfreq[w]- frequency[w]
    difffastfreq[w] = fastfreq[w]- frequency[w]
    diffmexicanfreq[w] = mexicanfreq[w]- frequency[w]
    diffseafoodfreq[w] = seafoodfreq[w]- frequency[w]
    diffcoffeefreq[w] = coffeefreq[w]- frequency[w]
    diffsandwichfreq[w] = sandwichfreq[w]- frequency[w]
```

In [50]:
```
# List the 10 words that appear more frequently in each category
```

In [51]:
```
moreFrequentAmerican = sorted(diffamericanfreq, key=diffamericanfreq.get, reve
rse=True)[:10]
print moreFrequentAmerican
```

```
[u'wa', u'the', u'brunch', u'food', u'breakfast', u'burger', u'menu', u'had',
u'we', u'servic']
```

In [52]:
```
moreFrequentBar = sorted(diffbarfreq, key=diffbarfreq.get, reverse=True)[:10]
print moreFrequentBar
```

```
[u'a', u'bar', u'drink', u'beer', u'to', u'night', u'music', u'place', u'grea
t', u'of']
```

In [53]:
```
moreFrequentAsian = sorted(diffasianfreq, key=diffasianfreq.get, reverse=True)
[:10]
print moreFrequentAsian
```

```
[u'sushi', u'thai', u'food', u'noodl', u'dish', u'chines', u'restaur', u'rol
l', u'indian', u'ramen']
```

In [54]:
```
moreFrequentEuropean = sorted(diffeuropeanfreq, key=diffeuropeanfreq.get, reve
rse=True)[:10]
print moreFrequentEuropean
```

```
[u'pizza', u'beer', u'and', u'wa', u'great', u'the', u'with', u'a', u'wine',
u'chees']
```

In [55]:
```
moreFrequentItalian = sorted(diffitalianfreq, key=diffitalianfreq.get, reverse
=True)[:10]
print moreFrequentItalian
```

```
[u'pizza', u'wine', u'restaur', u'the', u'veri', u'bread', u'wa', u'and', u'r
eserv', u'great']
```

In [56]:
```
moreFrequentFast = sorted(difffastfreq, key=difffastfreq.get, reverse=True)[:1
0]
print moreFrequentFast
```

```
[u'burger', u'fri', u'are', u'fast', u'you', u'they', u'their', u'sandwich',
u'order', u'line']
```

In [57]:
```
moreFrequentMexican = sorted(diffmexicanfreq, key=diffmexicanfreq.get, reverse
=True)[:10]
print moreFrequentMexican
```

```
[u'taco', u'mexican', u'food', u'burrito', u'margarita', u'salsa', u'are',
u'chip', u'i', u'the']
```

In [58]:
```
moreFrequentSeafood = sorted(diffseafoodfreq, key=diffseafoodfreq.get, reverse
=True)[:10]
print moreFrequentSeafood
```

```
[u'seafood', u'fish', u'wa', u'the', u'we', u'crab', u'fresh', u'view', u'ste
ak', u'roll']
```

In [59]:
```
moreFrequentCoffee = sorted(diffcoffeefreq, key=diffcoffeefreq.get, reverse=Tr
ue)[:10]
print moreFrequentCoffee
```

```
[u'coffe', u'shop', u'to', u'i', u'a', u'tea', u'they', u'cafe', u'place',
u'their']
```

In [60]:
```
moreFrequentSandwich = sorted(diffsandwichfreq, key=diffsandwichfreq.get, reve
rse=True)[:10]
print moreFrequentSandwich
```

```
[u'sandwich', u'chees', u'bread', u'their', u'they', u'i', u'are', u'lunch',
u'soup', '']
```

In [61]:
```
# 7
#Train an SVM to distinguish Bars and non-Bars only. That is, discard all othe
r categories from the
#training set in order to train a binary classifier, but keep them in the vali
dation set (your classifier will
#simply be wrong for those instances). Train for C ∈ {0.01, 0.1, 1, 10, 100}
 (the regularization parameter
#for the SVM) and report the best performance you obtain on the validation set
```

In [62]:
```
# make a mapping of word to index
word_to_index = {}
index_to_word = {}
index = 0

for w in words:
    word_to_index[w] = index
    index_to_word[index] = w
    index += 1
```

In [63]:
```
# Make a feature vector out of the 500 most common words and train to categori
ze bar

barX_train = []
barY_train = []

punctuation = set(string.punctuation)
stemmer = nltk.stem.porter.PorterStemmer()

for l in trainCat[:3500]:

    featureVector = [0] * 500
    for w in l['reviewText'].split():
        w = ''.join([c for c in w.lower() if not c in punctuation])
        w = stemmer.stem(w)
        if (w in words):
            featureVector[word_to_index[w]] += 1

    barX_train.append(featureVector)
    if l['categoryID'] == 1:
        barY_train.append(1)
    else:
        barY_train.append(0)
```

In [64]:
```
barX_val = []

for l in valCat[:3500]:

    featureVector = [0] * 500
    for w in l['reviewText'].split():
        w = ''.join([c for c in w.lower() if not c in punctuation])
        w = stemmer.stem(w)
        if (w in words):
            featureVector[word_to_index[w]] += 1

    barX_val.append(featureVector)
```

In [65]:
```
clf = svm.SVC(C=0.01, kernel='linear')
clf.fit(barX_train, barY_train)


val_predictions = clf.predict(barX_val)

accuracy = []

for (a,b) in zip(valCat[:3500], val_predictions):
    if a['categoryID'] == 1 and b==1:
        accuracy.append(1)
    if a['categoryID'] != 1 and b==0:
        accuracy.append(1)
    else:
        accuracy.append(0)

print 1.0*sum(accuracy)/len(accuracy)
```

```
0.877032810271
```

In [66]:
```
clf = svm.SVC(C=0.1, kernel='linear')
clf.fit(barX_train, barY_train)


val_predictions = clf.predict(barX_val)

accuracy = []

for (a,b) in zip(valCat[:3500], val_predictions):
    if a['categoryID'] == 1 and b==1:
        accuracy.append(1)
    if a['categoryID'] != 1 and b==0:
        accuracy.append(1)
    else:
        accuracy.append(0)

print 1.0*sum(accuracy)/len(accuracy)
```

```
0.864383180173
```

In [67]:
```
clf = svm.SVC(C=1, kernel='linear')
clf.fit(barX_train, barY_train)


val_predictions = clf.predict(barX_val)

accuracy = []

for (a,b) in zip(valCat[:3500], val_predictions):
    if a['categoryID'] == 1 and b==1:
        accuracy.append(1)
    if a['categoryID'] != 1 and b==0:
        accuracy.append(1)
    else:
        accuracy.append(0)

print 1.0*sum(accuracy)/len(accuracy)
```

```
0.848725678268
```

In [68]:
```
clf = svm.SVC(C=10, kernel='linear')
clf.fit(barX_train, barY_train)


val_predictions = clf.predict(barX_val)

accuracy = []

for (a,b) in zip(valCat[:3500], val_predictions):
    if a['categoryID'] == 1 and b==1:
        accuracy.append(1)
    if a['categoryID'] != 1 and b==0:
        accuracy.append(1)
    else:
        accuracy.append(0)

print 1.0*sum(accuracy)/len(accuracy)
```

```
0.825635419514
```

In [69]:
```
clf = svm.SVC(C=100, kernel='linear')
clf.fit(barX_train, barY_train)


val_predictions = clf.predict(barX_val)

accuracy = []

for (a,b) in zip(valCat[:3500], val_predictions):
    if a['categoryID'] == 1 and b==1:
        accuracy.append(1)
    if a['categoryID'] != 1 and b==0:
        accuracy.append(1)
    else:
        accuracy.append(0)

print 1.0*sum(accuracy)/len(accuracy)
```

```
0.822519083969
```

In [ ]:
```
# C = 0.01 seems to be give the best accuracy
```

In [ ]:
```
# 8
```

In [73]:
```
def train_val_prep(id):
    barX_train = []
    barY_train = []

    punctuation = set(string.punctuation)
    stemmer = nltk.stem.porter.PorterStemmer()

    for l in trainCat[:3500]:

        featureVector = [0] * 500
        for w in l['reviewText'].split():
            w = ''.join([c for c in w.lower() if not c in punctuation])
            w = stemmer.stem(w)
            if (w in words):
                featureVector[word_to_index[w]] += 1

        barX_train.append(featureVector)
        if l['categoryID'] == id:
            barY_train.append(1)
        else:
            barY_train.append(0)
    barX_val = []

    for l in valCat[:3500]:

        featureVector = [0] * 500
        for w in l['reviewText'].split():
            w = ''.join([c for c in w.lower() if not c in punctuation])
            w = stemmer.stem(w)
            if (w in words):
                featureVector[word_to_index[w]] += 1

        barX_val.append(featureVector)

    return (barX_train, barY_train, barX_val)
```

In [74]:
```
def svm_decision(x_train, y_train, x_val, _c):

    clf = svm.SVC(C=_c, kernel='linear')
    clf.fit(x_train, y_train)


    val_predictions = clf.predict(x_val)
    decision = clf.decision_function(x_val)
    return decision
```

In [76]:
```
cat_prep = []

for c in range(10):
    cat_prep.append(train_val_prep(c))
    print c
```

```
0
1
2
3
4
5
6
7
8
9
```

In [79]:
```
cat_2d = []
cVals = [0.01, 0.1, 1, 10, 100]
counter = 0

for c in cVals:
    cat_predicts = []
    for l in range(10):
        cat_predicts.append(svm_decision(cat_prep[l][0], cat_prep[l][1], cat_p
rep[l][2],c))
        counter += 1
        print counter
    cat_2d.append(cat_predicts)
```

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
```

In [80]:
```
len(cat_2d)
```

Out[80]: 5

In [84]:
```
predictions = []
for c in cat_2d:
    preds = []
    for i in range(3500):
        value = -10000
        index = -1
        for j in range(10):
            if c[j][i] > value:
                value = c[j][i]
                index = j
        preds.append(index)
    predictions.append(preds)
```

In [92]:
```
def accuracy_get(item):
    accuracy = []

    Y_val= []
    for c in valCat[:3500]:
        Y_val.append(c['categoryID'])

    for (a,b) in zip(Y_val, item):
        if a==b:
            accuracy.append(1)
        else:
            accuracy.append(0)

    print 1.0*sum(accuracy)/len(accuracy)
```

In [93]:
```
for c in predictions:
    accuracy_get(c)
```

```
0.511142857143
0.498
0.462857142857
0.423428571429
0.384571428571
```

In [94]:
```
# The highest c value is  0.01
```

In [95]:
```
# create predictions for kaggle
```

In [97]:
```
valid_features = []
valid_labels = []

for l in readGz("test_Category.json.gz"):
        featureVector = [0] * 500
        for w in l['reviewText'].split():
            w = ''.join([c for c in w.lower() if not c in punctuation])
            w = stemmer.stem(w)
            if (w in words):
                featureVector[word_to_index[w]] += 1

        valid_features.append(featureVector)
```

In [104]:
```
cat_prep2 = []
for i in range(10):
    barX_train = []
    barY_train = []

    punctuation = set(string.punctuation)
    stemmer = nltk.stem.porter.PorterStemmer()

    for l in trainCat[:3500]:

        featureVector = [0] * 500
        for w in l['reviewText'].split():
            w = ''.join([c for c in w.lower() if not c in punctuation])
            w = stemmer.stem(w)
            if (w in words):
                featureVector[word_to_index[w]] += 1

        barX_train.append(featureVector)
        if l['categoryID'] == i:
            barY_train.append(1)
        else:
            barY_train.append(0)
    cat_prep2.append((barX_train,barY_train,valid_features))
```

In [105]:
```
cat_3d = []
cat_predicts = []
c = 0.01
for l in range(10):
    cat_predicts.append(svm_decision(cat_prep2[l][0], cat_prep2[l][1], cat_pre
p2[l][2],c))
    counter += 1
    print counter
cat_3d.append(cat_predicts)
```

```
51
52
53
54
55
56
57
58
59
60
```

In [111]:
```
pred_list = []
for c in cat_3d:
    preds = []
    for i in range(20000):
        value = -10000
        index = -1
        for j in range(10):
            if c[j][i] > value:
                value = c[j][i]
                index = j
        preds.append(index)
    pred_list.append(preds)
```

In [114]:
```
pred = pred_list[0]
```

In [117]:
```
predictions = open("predictions_Category.txt", 'w')
predictions.write("userID-reviewHash,category\n")
for (l,p) in zip(readGz("test_Category.json.gz"),pred):

    predictions.write(l['userID'] + '-' + l['reviewHash'] + "," + str(p) + "\n
")
predictions.close()
```

In [ ]:
```
# My user name on kaggle is ddinata
# I got a score of 0.50790
```