In [1]:

```
import gzip
from collections import defaultdict
import numpy as np
import random
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
def readGz(f):
  for l in gzip.open(f):
    yield eval(l)

print "Reading data..."
data = list(readGz('train.json.gz'))
print "Done"
```

```
Reading data...
Done
```

In [63]:

```
# 1. Although we have built a validation set, it only
# consists of positive samples. For this task we also
# need examples of user/item pairs that weren't purchased.
# Build such a set by randomly sampling users and items until
# you have 100,000 non-purchased user/item pairs. This random
# sample combined with your 100,000 validation reviews now
# corresponds to the complete validation set for the purchase
# prediction task. Evaluate the performance (accuracy) of the
# baseline model on the validation set you have built (1 mark).

train = data[:100000]
valid = data[100000:]

user_item = defaultdict(set)
user = set()
item = set()
for d in train:
    user.add(d['reviewerID'])
    item.add(d['itemID'])
    user_item[d['reviewerID']].add(d['itemID'])
```

In [28]:

```
non_user_item = defaultdict(set)
count = 0
while count < 100000:
    u = random.choice(tuple(user))
    i = random.choice(tuple(item))
    if i not in user_item[u]:
        non_user_item[u].add(i)
        count += 1
        if count % 10000 == 0:
            print str(count)
```

```
10000
20000
30000
40000
50000
60000
70000
80000
90000
100000
```

In [34]:

```
businessCount = defaultdict(int)
totalPurchases = 0

for l in readGz("train.json.gz"):
  user,business = l['reviewerID'],l['itemID']
  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
```

In [35]:

```
new_valid = []
for u in non_user_item:
    for i in non_user_item[u]:
        new_valid.append((u,i))
for d in valid:
    new_valid.append((d['reviewerID'],d['itemID']))

y_valid = []
for i in range(100000):
    y_valid.append(0)
for i in range(100000):
    y_valid.append(1)

correct = 0;
for i in range(200000):
    d = new_valid[i]
    y = y_valid[i]
    if d[1] in return1 and y == 1:
        correct += 1
    elif d[1] not in return1 and y == 0:
        correct += 1

print "acc : " + str(correct * 1.0 / 200000)
```

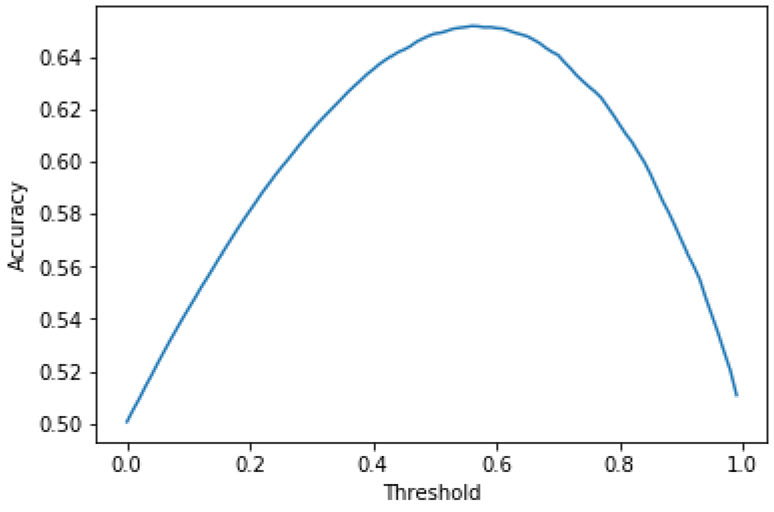
```
acc : 0.648895
```

In [36]:

```
# 2. The existing ‘purchase prediction’ baseline just returns True
#if the item in question is ‘popular,’ using a threshold of the 50th
#percentile of popularity (totalPurchases/2). Assuming that the ‘non-purchased’
#test examples are a random sample of user-purchase pairs, is this particular
#threshold value the best? If not, see if you can find a better one (and report
#its performance), or if so, explain why it is the best (1 mark).
def predict(threshold):
    return1 = set()
    count = 0
    for ic, i in mostPopular:
        count += ic
        return1.add(i)
        if count > totalPurchases * threshold: break
    correct = 0;
    for i in range(200000):
        d = new_valid[i]
        y = y_valid[i]
        if d[1] in return1 and y == 1:
            correct += 1
        elif d[1] not in return1 and y == 0:
            correct += 1
    return correct * 1.0 / 200000
```

In [37]:

```
acc = []
for i in range(100):
    acc.append(predict(i / 100.0))
plt.xlabel("Threshold")
plt.ylabel("Accuracy")
plt.plot(np.array(range(len(acc))) / 100., acc)
plt.show()
```



In [41]:

```
print "Maximum accuracy:" , max(acc)
print "Threshold for max:", acc.index(max(acc))
```

```
Maximum accuracy: 0.65182
Threshold for max: 56
```

In [44]:

```
#3. Users may tend to repeatedly purchase items of the same type.
#Build a baseline that returns ‘True’ if a user has purchased an
#item of the same category before (at least one category in common),
#or zero otherwise (1 mark).
user_category = defaultdict(list)
item_category = defaultdict(list)
for d in train:
    u = d['reviewerID']
    i = d['itemID']
    for c in d['categories']:
        user_category[u].append(c)
        item_category[i].append(c)

def pre_by_cate(u,i):
    for c in item_category[i]:
        if c in user_category[u]:
            return 1
    return 0
```

In [46]:

```
#4. To run our model on the test set, we’ll have to use the
#files ‘pairs Purchase.txt’ to find the review- erID/itemID pairs
#about which we have to make predictions. Using that data, run the
#above model and upload your solution to Kaggle. Tell us your Kaggle
#user name (1 mark). If you’ve already uploaded a better solution to Kaggle, tha
t’s fine too!

### Submitted under name NanShao

predictions = open("/Users/nan/Desktop/pairs_Purchase.txt", 'w')
for l in open("pairs_Purchase.txt"):
    if l.startswith("reviewerID"):
        #header
        predictions.write(l)
        continue
    u,i = l.strip().split('-')
    predictions.write(u + '-' + i + ',' + str(pre_by_cate(u,i)) + '\n')
predictions.close()
```

In [47]:

```
#5. What is the performance of a trivial predictor
#rating(user, item) = α
#on the validation set, and what is the value of α (1 mark)?

from sklearn.metrics import mean_squared_error
train5 = data[:100000]
valid5 = data[100000:]

# alpha
y_train5 = [d['rating'] for d in train5]
alpha = np.mean(y_train5)
print "alpha : " + str(alpha)

# accuracy on validation set
predictions = [alpha for d in valid5]
real = [d['rating'] for d in valid5]
print "MSE for #5 : ", mean_squared_error(real,predictions)
```

```
alpha : 4.232
MSE for #5 :  1.22248112
```

In [51]:

```
#6. Fit a predictor of the form
#rating(user, item) ≃ α + βuser + βitem,
#by fitting the mean and the two bias terms as described in the lecture notes. U
se a regularization
#parameter of λ = 1. Report the MSE on the validation set (1 mark).

beta_user = defaultdict(float)
beta_item = defaultdict(float)
alp = 0
lamb = 1
userrating = defaultdict(lambda: defaultdict(int))
businessrating = defaultdict(lambda: defaultdict(int))
for d in train:
    uu,bb = d['reviewerID'],d['itemID']
    userrating[uu][bb] = d['rating']
    businessrating[bb][uu] = d['rating']
    beta_user[uu] = 0
    beta_item[bb] = 0
```

In [55]:

```
import math
times = 50
lamb = 1
for i in range(times):
    alp = 0
    for uu in userrating:
        for bb in userrating[uu]:
            alp += userrating[uu][bb] - (beta_user[uu] + beta_item[bb])
    alp /= len(train)
    for uu in userrating:
        beta_user[uu] = 0
        for bb in userrating[uu]:
            beta_user[uu] += userrating[uu][bb] - (alp + beta_item[bb])
        beta_user[uu] /= (lamb + len(userrating[uu]))
    for bb in businessrating:
        beta_item[bb] = 0
        for uu in businessrating[bb]:
            beta_item[bb] += businessrating[bb][uu] - (alp + beta_user[uu])
        beta_item[bb] /= (lamb + len(businessrating[bb]))
    mse = 0
    for uu in userrating:
        for bb in userrating[uu]:
            mse += (alp + beta_user[uu] + beta_item[bb] - userrating[uu][bb]) **
2
    mse /= len(train)

validation_user = defaultdict(lambda: defaultdict(int))
validation_business = defaultdict(lambda: defaultdict(int))
for d in valid:
    uu, bb = d['reviewerID'], d['itemID']
    validation_user[uu][bb] = d['rating']
    validation_business[bb][uu] = d['rating']
mse = 0
for uu in validation_user:
    for bb in validation_user[uu]:
        mse += ((alp + (beta_user[uu] if uu in beta_user else 0) +
               (beta_item[bb] if bb in beta_item else 0) -
                validation_user[uu][bb]) **2)

mse /= len(valid)
print (repr(mse))
```

```
1.2811187854104154
```

In [68]:

```
# 7. Report the user and item IDs that have the largest and smallest
# values of β (1 mark).

max_u = ''
min_u = ''
maxbu = -100000000
minbu = 100000000
max_i = ''
min_i = ''
maxbi = -10000000
minbi = 10000000

for u in beta_user:
    if beta_user[u] > maxbu:
        maxbu = beta_user[u]
        max_u = u
    if beta_user[u] < minbu:
        minbu = beta_user[u]
        min_u = u
for i in beta_item:
    if beta_item[i] > maxbi:
        maxbi = beta_item[i]
        max_i = i
    if beta_item[i] < minbi:
        minbi = beta_item[i]
        min_i = i

print "max userid (max beta) :" + str(max_u) +'\t'+ str(maxbu)
print "max itemid (max beta) :" + str(max_i) +'\t'+ str(maxbi)
print "min userid (min beta) :" + str(min_u) +'\t'+ str(minbu)
print "min itemid (min beta) :" + str(min_i) +'\t'+ str(minbi)
```

```
max userid (max beta) :U495776285       1.48674942808
max itemid (max beta) :I809804570       1.27001482359
min userid (min beta) :U204516481       -2.5517035367
min itemid (min beta) :I511389419       -2.57571195579
```

In [71]:

```
# 8. Find a better value of λ using your validation set.
# Report the value you chose, its MSE, and upload your solution
# to Kaggle by running it on the test data (1 mark).


def fun8(lamb):
    times = 50
    for i in range(times):
        alp = 0
        for uu in userrating:
            for bb in userrating[uu]:
                alp += userrating[uu][bb] - (beta_user[uu] + beta_item[bb])
        alp /= len(train)
        for uu in userrating:
            beta_user[uu] = 0
            for bb in userrating[uu]:
                beta_user[uu] += userrating[uu][bb] - (alp + beta_item[bb])
            beta_user[uu] /= (lamb + len(userrating[uu]))
        for bb in businessrating:
            beta_item[bb] = 0
            for uu in businessrating[bb]:
                beta_item[bb] += businessrating[bb][uu] - (alp + beta_user[uu])
            beta_item[bb] /= (lamb + len(businessrating[bb]))
        mse = 0
        for uu in userrating:
            for bb in userrating[uu]:
                mse += (alp + beta_user[uu] + beta_item[bb] - userrating[uu][bb
]) **2
        mse /= len(train)

    validation_user = defaultdict(lambda: defaultdict(int))
    validation_business = defaultdict(lambda: defaultdict(int))
    for d in valid:
        uu, bb = d['reviewerID'], d['itemID']
        validation_user[uu][bb] = d['rating']
        validation_business[bb][uu] = d['rating']
    mse = 0
    for uu in validation_user:
        for bb in validation_user[uu]:
            mse += ((alp + (beta_user[uu] if uu in beta_user else 0) +
                  (beta_item[bb] if bb in beta_item else 0) -
                   validation_user[uu][bb]) **2)

    mse /= len(valid)
    return mse
```

In [72]:

```
lambdas = [0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0, 10000.0]
mse = []
for l in lambdas:
    mse.append(fun8(l))
for i in range(len(mse)):
    m = mse[i]
    l = lambdas[i]
    print str(m) +'\t'+ str(l)
```

```
1.98669483077   0.0001
1.98376825326   0.001
1.95376537543   0.01
1.74886745347   0.1
1.28111877898   1.0
1.14160309795   10.0
1.19982480867   100.0
1.21962246223   1000.0
1.22218649372   10000.0
```

In [73]:

```
fun8(10.0)
```

Out[73]:

```
1.1416030979547096
```

In [75]:

```
predictions = open("/Users/nan/Desktop/pairs_Rating.txt", 'w')
for l in open("pairs_Rating.txt"):
    if l.startswith("reviewerID"):
        #header
        predictions.write(l)
        continue
    u,i = l.strip().split('-')
    x = alp
    if u in beta_user:
        x += beta_user[u]
    if i in beta_item:
        x += beta_item[i]
    predictions.write(u + '-' + i + ',' + str(x) + '\n')

predictions.close()
```

In [ ]:

```

```