1) Generating Random User-Item non Purchased Pair for Validation:

```
uniqueUsers = set()
uniqueItems = set()
uniqueItemCats = {}
itemPurchased = defaultdict(set)
userPurchased = defaultdict(set)
totalPurchases = 0
user_count = 0
train = []
valid = []
for l in readGz("train.json.gz"):
  user,item,catId = l['reviewerID'],l['itemID'],l['categoryID']
  #itemCount[item] += 1
  itemPurchased[user].add(item)
  uniqueUsers.add(user)
  userPurchased[item].add(user)
  uniqueItems.add(item)
  uniqueItemCats[item] = str(catId)
  if user_count<100000:
    train.append([user,item,str(catId)])
  else:
    valid.append((user,item))
  user_count +=1
  totalPurchases += 1
```

```
with open('purchaseValidation.txt', 'w') as fp:
    print("Writing Validation file!!")
    fp.write('reviewerID-itemID,ispurchased\n')
    fp.write('\n'.join('%s-%s,1' % x for x in valid))
with open('purchaseTrain.txt', 'w') as fp:
    print("Writing Training file!!")
    fp.write('reviewerID-itemID-categoryID,ispurchased\n')
    fp.write('\n'.join(x[0]+'-'+x[1]+'-'+x[2]+',1' % x for x in train))
```

```
Writing Validation file!!
Writing Training file!!
```

```
def generateRandomUserItemValidation():
    count = 0
    nonPurchased = []
    while(count<100000):
        randUser = random.choice(uniqueUsers)
        randItem = random.choice(uniqueItems)
        if randItem in itemPurchased[randUser]:
            continue;
        else:
            if (randUser,randItem) not in nonPurchased:
                nonPurchased.append((randUser,randItem));
                count = count+1
                print(count)
            else:
                continue;
    with open('randNonPurchaseValidation.txt', 'w') as fp:
        fp.write('\n'.join('%s-%s,0' % x for x in nonPurchased))
```

```
itemCount = defaultdict(int)
userItemCat = defaultdict(set)
#uniqueItemCats ={}
for l in open("purchaseTrain.txt"):
  u,i,r = l.strip().split('-')
  c,p = r.strip().split(',')
  itemCount[i] += 1
  #uniqueItemCats[i] = c
  userItemCat[u].add(c)
```

2) Baseline Performance on Validation:

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

return1 = set()
count = 0
for ic, i in mostPopular:
  count += ic
  return1.add(i)
  if count > totalPurchases/2: break
y_true =[]
y_pred =[]
predictions = open("predictions_Purchase_Validation.txt", 'w')
for l in open("Validation.txt"):
  if l.startswith("reviewerID"):
    #header
    predictions.write(l)
    continue
  u,r = l.strip().split('-')
  i,ac =  r.strip().split(',')
  y_true.append(ac)
  if i in return1:
    predictions.write(u + '-' + i + ",1\n")
    y_pred.append('1');
  else:
    predictions.write(u + '-' + i + ",0\n")
    y_pred.append('0');
predictions.close()
accuracy_score(y_true,y_pred)
```

```
0.502635
```

When tested on a range of threshold values: 0.5 was not the optimum threshold value

```
0.1 0.575195
0.2 0.61912
0.30000000000000004 0.62766
0.4 0.58986
0.5 0.502635
0.6 0.502635
0.7000000000000001 0.502635
0.8 0.502635
0.9 0.502635
```

```
thresh = list(np.arange(0.2, 0.4,0.01))
for t in thresh:
    return1 = set()
    count = 0
    for ic, i in mostPopular:
      count += ic
      return1.add(i)
      if count > totalPurchases*t: break
    y_true =[]
    y_pred =[]
    predictions = open("predictions_Purchase_Validation.txt", 'w')
    for l in open("Validation.txt"):
      if l.startswith("reviewerID"):
        predictions.write(l)
        continue
      u,r = l.strip().split('-')
      i,ac =  r.strip().split(',')
      y_true.append(ac)
      if i in return1:
        predictions.write(u + '-' + i + ",1\n")
        y_pred.append('1');
      else:
        predictions.write(u + '-' + i + ",0\n")
        y_pred.append('0');
    predictions.close()
    print(t,accuracy_score(y_true,y_pred))
```

```
0.2 0.61912
0.21000000000000002 0.62153
0.22000000000000003 0.62363
0.23000000000000004 0.62606
0.24000000000000005 0.627945
0.25000000000000006 0.628755
0.26000000000000006 0.62928
0.2700000000000001 0.62877
0.2800000000000001 0.628655
0.2900000000000001 0.628695
0.3000000000000001 0.62766
0.3100000000000001 0.62536
0.3200000000000001 0.62278
0.3300000000000001 0.620505
0.34000000000000014 0.618075
0.35000000000000014 0.613185
0.36000000000000015 0.608735
0.37000000000000016 0.604575
0.38000000000000017 0.60026
0.3900000000000002 0.596035
```

Best threshold for the Validation set:

```
return1 = set()
count = 0
for ic, i in mostPopular:
  count += ic
  return1.add(i)
  if count > totalPurchases*0.26: break
y_true =[]
y_pred =[]
predictions = open("predictions_Purchase_Validation.txt", 'w')
for l in open("Validation.txt"):
  if l.startswith("reviewerID"):
    predictions.write(l)
    continue
  u,r = l.strip().split('-')
  i,ac =  r.strip().split(',')
  y_true.append(ac)
  if i in return1:
    predictions.write(u + '-' + i + ",1\n")
    y_pred.append('1');
  else:
    predictions.write(u + '-' + i + ",0\n")
    y_pred.append('0');
predictions.close()
print(0.26,accuracy_score(y_true,y_pred))
```

```
0.26 0.62928
```

3) Using Item Categories to Predict purchases:

```
import itertools
```

```
userCatPurchased = defaultdict(set)
itemCat = defaultdict(set)
itemCount = defaultdict(int)
#newset = set()
totalPurchases = 0
userCount = 0
for l in readGz("train.json.gz"):
    if userCount < 100000:
        user,item = l['reviewerID'],l['itemID']
        #print(l['categories'][0])
        #newset.add(len(l['categories']))
        itemCount[item] += 1
        totalPurchases+=1
        catlist = list(itertools.chain.from_iterable(l['categories']))
        #catset = l['categories'][0]+l['categories'][1]
        userCatPurchased[user].update(catlist)
        itemCat[item].update(catlist)
    userCount +=1
#print(newset)
```

```
predictions = open("predictions_purchase_categories.txt", 'w')
y_true = []
y_pred = []
for l in open("Validation.txt"):
  if l.startswith("reviewerID"):
    #header
    predictions.write(l)
    continue
  u,r = l.strip().split('-')
  i,ac = r.strip().split(',')
  y_true.append(ac)
  if u in userCatPurchased and i in itemCat:
    if len(userCatPurchased[u].intersection(itemCat[i]))>=1:
        predictions.write(u + '-' + i + ',' + '1' + '\n')
        y_pred.append('1')
    else:
        predictions.write(u + '-' + i + ',' + '0' + '\n')
        y_pred.append('0')
  else:
    predictions.write(u + '-' + i + ',' + '0' + '\n')
    y_pred.append('0')
predictions.close()
print(accuracy_score(y_true,y_pred))
```

```
0.62697
```

4) Kaggle username and Team name: Akshi238

5) Rating Prediction using Constant:

```
allRatings = []
regressor = []
userRatings = defaultdict(list)

data = list(readGz("train.json.gz"))
train = data [:100000]
validation = data[100000:]

for l in train:
  user,item = l['reviewerID'],l['itemID']
  allRatings.append(l['rating'])
  regressor.append([user,item,l['rating']])
  #userRatings[user].append(l['rating'])

globalTrainAverage = sum(allRatings) / len(allRatings)

validationRegressor = []
y_valid = []
for l in validation:
    user,item = l['reviewerID'],l['itemID']
    y_valid.append(l['rating'])
    validationRegressor.append([user,item,l['rating']])

with open('ratingTrain.txt', 'w') as fp:
    print("Writing rating Train file!!")
    fp.write('reviewerID-itemID,Rating\n')
    fp.write('\n'.join(x[0]+'-'+x[1]+','+str(x[2]) % x for x in regressor))
```

```
with open('ratingValidation.txt', 'w') as fp:
    print("Writing rating Validation file!!")
    fp.write('reviewerID-itemID,Rating\n')
    fp.write('\n'.join(x[0]+'-'+x[1]+','+str(x[2]) % x for x in validationRegressor))

#userAverage = {}
#for u in userRatings:
#  userAverage[u] = sum(userRatings[u]) / len(userRatings[u])
y_pred =[]
predictions = open("predictions_Rating_Validation.txt", 'w')
for l in open("ratingValidation.txt"):
  if l.startswith("reviewerID"):
    #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(globalTrainAverage) + '\n')
  y_pred.append(globalTrainAverage)

predictions.close()
print('alpha: ')
print(globalTrainAverage)
```

```
Writing rating Train file!!
Writing rating Validation file!!
alpha:
4.232
```

Validation MSE:

```
print(" Validation MSE : ", mean_squared_error(y_valid,y_pred))
```

```
 Validation MSE :  1.22248112
```

Validation RMSE:

```
print(" Validation RMSE : ", np.sqrt(mean_squared_error(y_valid,y_pred)))
```

```
 Validation RMSE :  1.1056586815107092
```

6) Linear Predictor Model:

```python
def get_alpha():
    sum_r = 0
    count_r = 0
    for u in users_items:
        for i in users_items[u]:
            sum_r += train_set[i[1]]['rating']
            count_r += 1
    alpha = float(sum_r)/count_r
    #print(alpha)
    return alpha
```

```python
def getBetaU():
    beta_u = []
    for u in users_items:
        count_hr_u = 0
        count_r_u = 0
        for i in users_items[u]:
            count_r_u += 1
            if train_set[i[1]]['rating'] > alpha:
                count_hr_u += 1
            else:
                count_hr_u -= 1
        beta_u.append(float(count_hr_u)/count_r_u)
    return beta_u
```

```python
def getBetaI():
    beta_i = []
    for i in items_users.keys():
        count_hr_i = 0
        count_r_i = 0
        for u in items_users[i]:
            count_rate += 1
            if train_set[u[1]]['rating'] > alpha:
                count_hr_i += 1
            else:
                count_hr_i -= 1
        beta_i.append(float(count_hr_i)/count_r_i)
    return beta_i
```

```
def update(A,B_user,B_item,lam):
    # Update alpha
    sum_A = 0
    for u in users_items:
        user_index = getIndex(u,users)
        for i in users_items[u]:
            item_index = getIndex(i[0],items)
            sum_A += train_set[i[1]]['rating'] - B_user[user_index] - B_item[item_index]
    A = float(sum_A) / len(train_set)
    # Update beta_user
    for u in users_items:
        sum_Bu = 0
        count_item = 0
        user_index = getIndex(u,users)
        for i in users_items[u]:
            item_index = getIndex(i[0],items)
            count_item += 1
            sum_Bu += train_set[i[1]]['rating'] - A - B_item[item_index]
        B_user[user_index] = float(sum_Bu) / (lam + count_item)
    # Update beta_item
    for i in items_users:
        sum_Bi = 0
        count_user = 0
        item_index = getIndex(i,items)
        for u in items_users[i]:
            user_index = getIndex(u[0],users)
            count_user += 1
            sum_Bi += train_set[u[1]]['rating'] - A - B_user[user_index]
        B_item[item_index] = float(sum_Bi) / (lam + count_user)
    return Alpha,Betauser,Betaitem
```

```
a = get_alpha()
betau = getBetaU()
betai = getBetaI()
print ('MSE init' + str(MSE(a,betau,betai)))
for iter_time in range(10):
    count = iter_time + 1
    a,betau,betai = update(a,betau,betai,1.0)
    print (str(count)+ ' : ' + str(MSE(a,betau,betai)))
```

```
MSE init: 1.695678932406808
1 : 1.4034180864153343
2 : 1.3595702911477703
3 : 1.3488135418813958
4 : 1.3453658040191743
5 : 1.3441740771150283
6 : 1.343761049899788
7 : 1.3436237087545733
8 : 1.3435836150828455
9 : 1.3435770355776104
10 : 1.3435813506067462
```

7) Min and Max Beta Users and Items:

```
print ('Users Min: ' + users[betau.index(min(betau))])
print ('Users Max: ' + users[betau.index(max(betau))])
print ('Items Min: ' + items[betai.index(min(betai))])
print ('Items Max: ' + items[betai.index(max(betai))])
```

```
Users Min: U204516481
Users Max: U495776285
Items Min: I511389419
Items Max: I809804570
```

8) Best lambda was found to be 7 and Validation MSE was: 1.13777

```
a = get_alpha()
betau = getBetaU()
betai = getBetaI()
print ('MSE init' + str(MSE(a,betau,betai)))
for iter_time in range(10):
    count = iter_time + 1
    a,betau,betai = update(a,betau,betai,7.0)
    print (str(count)+ ' : ' + str(MSE(a,betau,betai)))
```

```
MSE init: 1.695678932406808
1 : 1.1740030039961906
2 : 1.1486719520406414
3 : 1.1413876389240192
4 : 1.1389116834263286
5 : 1.1381227614683205
6 : 1.1378798285371197
7 : 1.137805082362027
8 : 1.1377813416667593
9 : 1.1377733002004913
10 : 1.1377703190831638
```