In [1]:
```
import gzip
import random
import numpy as np
from collections import defaultdict
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LogisticRegression
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
def readGz(path):
    for l in gzip.open(path, 'rt'):
        yield eval(l)

def readCSV(path):
    f = gzip.open(path, 'rt')
    f.readline()
    for l in f:
        yield l.strip().split(',')
```

In [3]:
```
example = readCSV("train_Interactions.csv.gz")
print(next(example))
del example
```

```
['u79354815', 'b14275065', '4']
```

Question 1

In [68]:
```
### Would-read

train_size = 190000
val_size   = 10000

data = [line for line in readCSV("train_Interactions.csv.gz")]
train = data[:train_size]
val   = data[train_size:]
print(len(data))
print(len(train))
print(len(val))
```

```
200000
190000
10000
```

In [69]:
```
booksReadBy = defaultdict(set)
train_booksReadBy = defaultdict(set)
val_booksReadBy = defaultdict(set)
usersReadBook = defaultdict(set)
val_usersReadBook = defaultdict(set)
train_usersReadBook = defaultdict(set)
all_books = set()
val_all_books = set()
train_all_books = set()

for user, book, _ in data:
    all_books.add(book)
    usersReadBook[book].add(user)
    booksReadBy[user].add(book)

for user, book, _ in train:
    train_all_books.add(book)
    train_usersReadBook[book].add(user)
    train_booksReadBy[user].add(book)

for user, book, _ in val:
    val_all_books.add(book)
    val_usersReadBook[book].add(user)
    val_booksReadBy[user].add(book)

val_unread = []
all_books_count = len(all_books)
for user, book, _ in val:
    unread_book = random.sample(all_books, 1)
    while(unread_book in list(booksReadBy[user])):
        unread_book = random.sample(all_books, 1)
    val_unread.append([user, str(unread_book[0]), '-1'])

val = val + val_unread
print(len(val))
print(val[0:3])
```

```
20000
[['u35176258', 'b30592470', '3'], ['u30851063', 'b81941226', '3'], ['u3136841
4', 'b40097012', '5']]
```

In [70]:
```
bookCount = defaultdict(int)
total_books_read = 0

for user, book, _ in train:
    bookCount[book]  += 1
    total_books_read += 1

mostPopular = [(bookCount[book], book) for book in bookCount]
mostPopular.sort()
mostPopular.reverse()
```

In [71]:
```
def popular_books_set(mostPopular, threshold_ratio):
    return1 = set()
    cur_book_count = 0
    for book_count, book in mostPopular:
        cur_book_count += book_count
        return1.add(book)
        if cur_book_count > total_books_read *\
        threshold_ratio:
            break
    return return1
```

In [72]:
```
return1 = popular_books_set(mostPopular, threshold_ratio = 0.5)
predictions = []
for data_point in val:
    user, book, _ = data_point
    prediction = book in return1
    predictions.append(prediction)

labels = [int(rating) >= 0 for _, _, rating in val]
predictions = np.array(predictions)
labels      = np.array(labels)
acc = sum(predictions == labels) / len(predictions)
print('Accuracy on validation set is:', acc)
```

```
Accuracy on validation set is: 0.64245
```

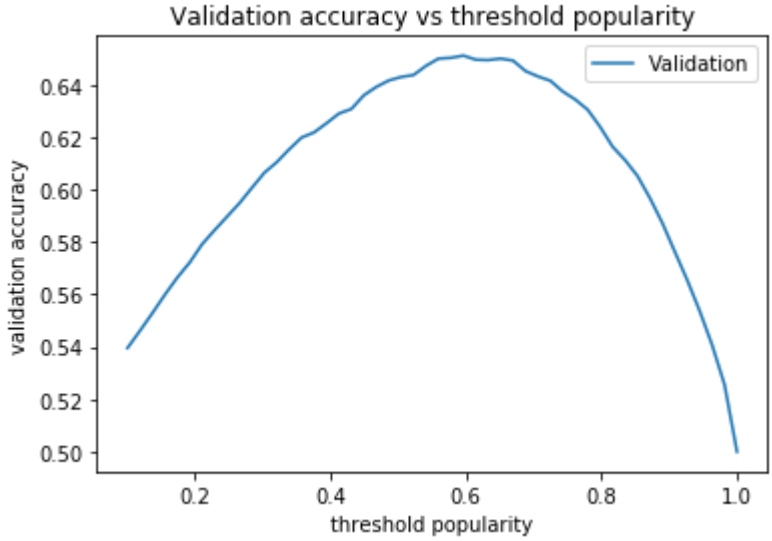
Question 2

In [73]:
```
accuracies = []
ratio_count = 50
threshold_ratios = np.linspace(0.1, 1, ratio_count)

for ratio in threshold_ratios:
    return1 = popular_books_set(mostPopular, ratio)
    predictions = []
    for data_point in val:
        user, book, _ = data_point
        prediction = book in return1
        predictions.append(prediction)

    labels = [int(rating) >= 0 for _, _, rating in val]
    predictions = np.array(predictions)
    labels      = np.array(labels)
    acc = sum(predictions == labels) / len(predictions)
    accuracies.append(acc)
```

In [74]:
```
plt.plot(threshold_ratios, accuracies, label='Validation')
plt.ylabel('validation accuracy')
plt.xlabel('threshold popularity')
plt.title('Validation accuracy vs threshold popularity')
plt.legend()
plt.show()
```



In [75]:
```
indx1 = accuracies.index(max(accuracies))
print('Ratio with best accuracy is:', threshold_ratios[indx1])
print('It has accuracy:', max(accuracies))
```

```
Ratio with best accuracy is: 0.5959183673469388
It has accuracy: 0.65105
```

To get the best ratio, I ran 50 separate test on the validation test. Each test has a different population threshold, and the accuracy of each test is recorded. The results are shown in the above graph. From the curve above, the best threshold value is around 0.6 (also printed under the curve). The get a better understanding I should also use metrics such as Balanced Error Rate, Recall, Precision etc, which I am not asked to do in this question.

Question 3

In [12]:
```
def jaccard(set1, set2):
    """
    Returns the Jaccard similarity between two sets,
    set1 & set2
    """
    set_intersection = len(set1.intersection(set2))
    set_union = len(set1.union(set2))
    if set_union == 0:
        return 0
    else:
        return set_intersection / set_union
```

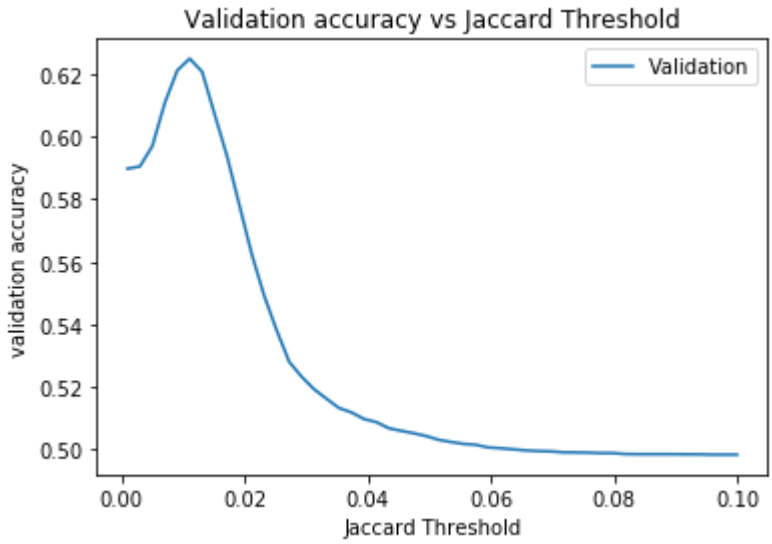
In [13]:
```
def jaccard_prediction(jac_sims):
    """
    Returns the Jaccard similarity between two sets,
    set1 & set2
    """
    prediction = False
    if jac_sims != []:
        prediction = max(jac_sims) >= jaccard_threshold
    return prediction
```

In [31]:
```
thresholds = np.linspace(0.001, 0.1, 50)
accuracies = []
loop_count = 0
for jaccard_threshold in thresholds:
    loop_count += 1
    if loop_count % 10 == 0: print(loop_count, end = ' ')
    predictions = []
    for user, book_predict, _ in val:
        books_user_read = train_booksReadBy[user]
        jac_sims = []
        for users_book in books_user_read:
            users_read_book_predict = train_usersReadBook[book_predict]
            users_read_users_book = train_usersReadBook[users_book]
            jac_sim = jaccard(users_read_book_predict, users_read_users_book)
            jac_sims.append(jac_sim)
        prediction = jaccard_prediction(jac_sims)
        predictions.append(prediction)

    predictions = np.array(predictions)
    acc = sum(predictions == labels) / len(predictions)
    accuracies.append(acc)
```

```
10 20 30 40 50
```

In [32]:
```
plt.plot(thresholds, accuracies, label='Validation')
plt.ylabel('validation accuracy')
plt.xlabel('Jaccard Threshold')
plt.title('Validation accuracy vs Jaccard Threshold')
plt.legend()
plt.show()
```



In [33]:
```
indx2 = accuracies.index(max(accuracies))
print('Jaccard threshold with best accuracy is:',\
      thresholds[indx2])
print('This threshold has validation accuracy:',\
      accuracies[indx2])
```

```
Jaccard threshold with best accuracy is: 0.011102040816326531
This threshold has validation accuracy: 0.62495
```

Question 4

In [34]:
```
sizes = 10
pops = np.linspace(0.2, 0.7, sizes)
jaccards = np.linspace(0.01, 0.1, sizes)
print(len(pops))
print(len(jaccards))
```

```
10
10
```

In [14]:
```
def calc_jac(user, book_predict):
    books_user_read = train_booksReadBy[user]
    jac_sims = []
    for users_book in books_user_read:
        users_read_book_predict = train_usersReadBook[book_predict]
        users_read_users_book = train_usersReadBook[users_book]
        jac_sim = jaccard(users_read_book_predict, users_read_users_book)
        jac_sims.append(jac_sim)
    return jac_sims
```

In [39]:
```
pop_accuracies = []
loop_count = 0
max_acc = 0
size1, size2 = len(pops), len(jaccards)
for pop_threshold in pops:
    accuracies = []
    return1 = popular_books_set(mostPopular, pop_threshold)
    for jaccard_threshold in jaccards:
        loop_count += 1
        if loop_count % 10 == 0: print(loop_count, end = ' ')
        predictions = []
        for user, book_predict, _ in val:
                books_user_read = train_booksReadBy[user]
                jac_sims = calc_jac(user, book_predict)
                prediction_jac = jaccard_prediction(jac_sims)
                prediction_pop = book_predict in return1
                prediction = prediction_jac or prediction_pop
                predictions.append(prediction)
        predictions = np.array(predictions)
        acc = sum(predictions == labels) / len(predictions)
        accuracies.append(acc)
    pop_accuracies.append(accuracies)
```

```
10 20 30 40 50 60 70 80 90 100
```

In [40]:
```
best_accs = [max(acc) for acc in pop_accuracies]
indx1 = best_accs.index(max(best_accs))
print('Population threshold with best accuracy is:',\
      pops[indx1])
best_jaccards = pop_accuracies[indx1]
indx2 = best_jaccards.index(max(best_jaccards))
print('Jaccard threshold with best accuracy:',\
      jaccards[indx2])
print('Best accuracy', pop_accuracies[indx1][indx2])
```

```
Population threshold with best accuracy is: 0.5333333333333333
Jaccard threshold with best accuracy: 0.030000000000000006
Best accuracy 0.65905
```

I find the best results (out of the ones I tried) to be as shown above (for the validation set).

Question 5

My username on Kaggle is: Arda Cankat Bati

In [15]:
```
def calc_jac_train(user, book_predict):
    books_user_read = train_booksReadBy[user]
    jac_sims = []
    for users_book in books_user_read:
        if users_book == book_predict: continue
        users_read_book_predict = train_usersReadBook[book_predict]
        users_read_users_book = train_usersReadBook[users_book]
        jac_sim = jaccard(users_read_book_predict, users_read_users_book)
        jac_sims.append(jac_sim)
    return jac_sims
```

In [16]:
```
def predict_datapoint(user, book_predict):
    books_user_read = booksReadBy[user]
    jac_sims = calc_jac(user, book_predict)
    prediction_jac = max(jac_sims) >= jaccard_threshold
    prediction_pop = book_predict in return1
    prediction = prediction_jac or prediction_pop
    return prediction
```

In [ ]:
```
pop_threshold = 0.53
jaccard_threshold = 0.03
return1 = popular_books_set(mostPopular, 0.6)

with open("predictions_Read.txt", 'w') as predictions:
    for l in open("pairs_Read.txt"):
        if l.startswith("userID"): # it's just the header
            predictions.write(l)
            continue
        user, book = l.strip().split('-') # it is a datapoint
        prediction = predict_datapoint(user, book)
        if prediction:
            predictions.write(user + '-' + book + ",1\n")
        else:
            predictions.write(user + '-' + book + ",0\n")
```

Question 9

In [80]:
```
### Ratings Prediction

train_size = 190000
val_size   = 10000
data       = [line for line in readCSV("train_Interactions.csv.gz")]
train      = data[:train_size]
val        = data[train_size:]

allRatings = []
userBookRatings = defaultdict(lambda: defaultdict(float))
userRatings = defaultdict(list)
userBooks   = defaultdict(set)
bookUsers   = defaultdict(set)

for user, book, rating in train:
    rating = int(rating)
    allRatings.append(rating)
    userRatings[user].append(rating)
    userBookRatings[user][book] = rating
    userBooks[user].add(book)
    bookUsers[book].add(user)

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

In [81]:
```
# Coordinate Descent
def coordinate_descent(lambda_opt = 1, threshold = 4 * 10**(-5)):

    alpha_sum, bu_sum, bb_sum = 0, 0, 0

    train_len = len(train)
    bu = defaultdict(lambda: 1)
    bb = defaultdict(lambda: 1)
    alpha = 0

    conv = 1; prev_MSE = 1
    while(conv > threshold):
        alpha_sum = 0
        for user, book, _ in train:
            alpha_sum += userBookRatings[user][book] - (bu[user] + bb[book])
        alpha = alpha_sum / train_len

        for user in userRatings:
            bu_sum = 0
            for book in userBooks[user]:
                bu_sum += userBookRatings[user][book] - (alpha + bb[book])
            bu[user] = bu_sum / (lambda_opt + len(userBooks[user]))

        for book in bookUsers:
            bb_sum = 0
            for user in bookUsers[book]:
                bb_sum += userBookRatings[user][book] - (alpha + bu[user])
            bb[book] = bb_sum / (lambda_opt + len(bookUsers[book]))

        rating_labels = []
        diff = 0
        for user, book, rating in val:
            user_rating = alpha + bu[user] + bb[book]
            diff += (user_rating - int(rating)) ** 2

        cur_MSE = diff / len(val)
        conv = abs(cur_MSE / prev_MSE - 1)
        prev_MSE = cur_MSE

    return alpha, bu, bb

alpha, bu, bb = coordinate_descent(lambda_opt = 1)
```

In [82]:
```
rating_labels = []
diff = 0
for user, book, rating in val:
    user_rating = alpha + bu[user] + bb[book]
    diff += (user_rating - int(rating)) ** 2

MSE = diff / len(val)
print('MSE on the validation set', MSE)
```

```
MSE on the validation set 1.1186486793316754
```

Question 10

In [83]:
```
bu_sorted = list(bu.items())
bu_sorted = sorted(bu_sorted, key = lambda x: x[1])
bb_sorted = list(bb.items())
bb_sorted = sorted(bb_sorted, key = lambda x: x[1])
print('User with lowest beta_user score, and corresponding score:', bu_sorted[
0])
print('User with lowest beta_user score, and corresponding score:', bu_sorted[
-1])
print('Book with lowest beta_book score, and corresponding score:', bb_sorted[
0])
print('Book with lowest beta_book score, and corresponding score:', bb_sorted[
-1])
print(bu_sorted[-1])
```

```
User with lowest beta_user score, and corresponding score: ('u48313610', -3.0
761521887629506)
User with lowest beta_user score, and corresponding score: ('u06559157', 1.98
24130435487222)
Book with lowest beta_book score, and corresponding score: ('b84091840', -0.8
591322875560226)
Book with lowest beta_book score, and corresponding score: ('b19925500', 2.35
4087719007011)
('u06559157', 1.9824130435487222)
```

TODO

Question 11

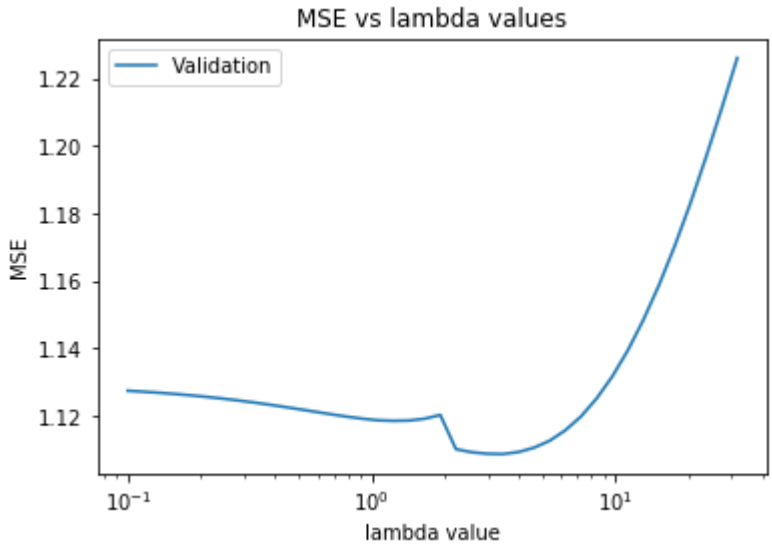
In [84]:
```
lambda_values = np.logspace(-1, 1.5, num = 40)
print(lambda_values)
```

```
[ 0.1         0.1159051   0.13433993  0.15570684  0.18047218  0.20917647
  0.2424462   0.28100752  0.32570207  0.37750532  0.43754794  0.5071404
  0.58780161  0.68129207  0.78965229  0.91524731  1.06081836  1.22954263
  1.42510267  1.65176674  1.91448198  2.21898234  2.57191381  2.9809794
  3.45510729  4.00464573  4.64158883  5.3798384   6.23550734  7.22727132
  8.3767764   9.70911147 11.25335583 13.04321387 15.11775071 17.5222448
 20.30917621 23.53937198 27.28333376 31.6227766 ]
```

In [58]:
```
MSEs = []
loop_count = 0
for lambda_opt in lambda_values:
    loop_count += 1; print(loop_count, end = ', ')
    alpha, bu, bb = coordinate_descent(lambda_opt, 10**(-4))
    rating_labels = []
    diff = 0
    for user, book, rating in val:
        user_rating = alpha + bu[user] + bb[book]
        diff += (user_rating - int(rating)) ** 2
    MSE = diff / len(val)
    MSEs.append(MSE)
```

```
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 2
2, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40,
```

In [64]:
```
plt.plot(lambda_values, MSEs, label='Validation')
plt.ylabel('MSE')
plt.xlabel('lambda value'), plt.xscale('log')
plt.title('MSE vs lambda values')
plt.legend()
plt.show()
```



In [89]:
```
indx = MSEs.index(min(MSEs))
print('Lambda which has lowest MSE is:', lambda_values[indx])
print('This lambda value has MSE:     ', MSEs[indx])
```

```
Lambda which has lowest MSE is: 3.455107294592218
This lambda value has MSE:      1.1086230831726713
```

In [66]:
```
alpha, bu, bb = coordinate_descent(3.45, 10**(-4))
```

In [67]:
```
with open("predictions_Rating.txt", 'w') as predictions:
    for l in open("pairs_Rating.txt"):
        if l.startswith("userID"):
            #header
            predictions.write(l)
            continue
        user, book = l.strip().split('-')
        user_rating = alpha + bu[user] + bb[book]
        predictions.write(user + '-' + book + ',' + str(user_rating) + '\n')
```