DATA 643 Project 1 | Global Baseline Predictors and RMSE

In this first assignment, we'll attempt to predict ratings with very little information. We'll first look at just raw averages across all (training dataset) users. We'll then account for "bias" by normalizing across users and across items.

You'll be working with ratings in a user-item matrix, where each rating may be (1) assigned to a training dataset, (2) assigned to a test dataset, or (3) missing.

Please code as much of your work as possible in R or Python. You may use standard functions (e.g. from base R and the tidyverse or from a standard scientific Python tool stack). Your project should be delivered in an R Markdown or a Jupyter notebook, then the notebook should be saved into a GitHub repository. You should include a link to your GitHub repository in your assignment submission link.

Preparation. Start by watching Parts K through P from this playlist from the Coursera/Stanford Networks Illustrated course (total run time is about 22 minutes): https://www.youtube.com/playlist?list=PLuKhJYywjDe96T2L0-zXFU5Up2jqXlWI9 (https://www.youtube.com/playlist?list=PLuKhJYywjDe96T2L0-zXFU5Up2jqXlWI9)

• Briefly describe the recommender system that you're going to build out from a business perspective, e.g. "This system recommends data science books to readers." • Find a dataset, or build out your own toy dataset. As a minimum requirement for complexity, please include numeric ratings for at least five users, across at least five items, with some missing data. • Load your data into (for example) an R or pandas dataframe, a Python dictionary or list of lists, (or another data structure of your choosing). From there, create a user-item matrix. • If you choose to work with a large dataset, you're encouraged to also create a small, relatively dense "user-item" matrix as a subset so that you can hand-verify your calculations. • Break your ratings into separate training and test datasets. • Using your training data, calculate the raw average (mean) rating for every user-item combination. • Calculate the RMSE for raw average for both your training data and your test data. • Using your training data, calculate the bias for each user and each item. • From the raw average, and the appropriate user and item biases, calculate the baseline predictors for every user-item combination. • Calculate the RMSE for the baseline predictors for both your training data and your test data. • Summarize your results. You may work in a small group (2 or 3 people) on this assignment.

In [1]:
```
# import modules and functions
import pandas as pd
import numpy as np
import csv
import csv
import os
from pandas import *
from numpy import *
from math import *
import matplotlib.pyplot as plt
```

This system recommends books to readers.

1. Load data into pandas dataframes.

In [2]:
```
path_1 = 'https://raw.githubusercontent.com/YunMai-SPS/DA643/master/DA643_Proj
ect_1/BX-CSV-Dump/BX-Book-Ratings.csv'
rating = pd.read_csv(path_1)
```

In [3]:
```
path_2 = 'https://raw.githubusercontent.com/YunMai-SPS/DA643/master/DA643_Proj
ect_1/BX-CSV-Dump/BX-Books.csv'
book = pd.read_csv(path_2)

path_3 = 'https://raw.githubusercontent.com/YunMai-SPS/DA643/master/DA643_Proj
ect_1/BX-CSV-Dump/BX-Users.csv'
user = pd.read_csv(path_3)
```

```
D:\Program Files\Anaconda3\lib\site-packages\IPython\core\interactiveshell.p
y:2717: DtypeWarning: Columns (9,10,11,12,13,14,15,16,17,18,19,20,21) have mi
xed types. Specify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
D:\Program Files\Anaconda3\lib\site-packages\IPython\core\interactiveshell.p
y:2717: DtypeWarning: Columns (7,8) have mixed types. Specify dtype option on
import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

The data were downloaded from BOOK-CORSSING DATASET which is freely available for research use.

Reference:

1.Cai-Nicolas Ziegler, Sean M. McNee, Joseph A. Konstan, Georg Lausen,Proceedings of the 14th International World Wide Web Conference (WWW '05), Improving Recommendation Lists Through Topic Diversification, May 10-14, 2005, Chiba, Japan.

1. Clean and prepare data.

1.1 Ratings and books data.

In [4]:
```
rating.shape
```

Out[4]: (1048575, 1)

In [5]:
```
rating.columns
```

Out[5]: Index(['User-ID;"ISBN";"Book-Rating"'], dtype='object')

In [6]:
```
rating = DataFrame([item.split(';')[0:3] for item in rating['User-ID;"ISBN";"B
ook-Rating"']]) # split one column into several columns
rating.columns = ['User-ID','ISBN','Book-Rating']

rating['ISBN'] = rating['ISBN'].str.replace('"', "")
rating['Book-Rating'] = rating['Book-Rating'].str.replace('"', "")
```

In [7]:
```
rating.head(5)
```

Out[7]:

	User-ID	ISBN	Book-Rating
0	276725	034545104X	0
1	276726	0155061224	5
2	276727	0446520802	0
3	276729	052165615X	3
4	276729	0521795028	6

In [8]:
```
rating.dtypes
```

Out[8]:
```
User-ID        object
ISBN           object
Book-Rating    object
dtype: object
```

In [9]:
```
rating.shape
```

Out[9]:
```
(1048575, 3)
```

In [10]:
```
rating['Book-Rating']=to_numeric(rating['Book-Rating'],errors='coerce')
```

In [11]:
```
rating['Book-Rating'].dtype
```

Out[11]:
```
dtype('int64')
```

In [12]:
```
book_im=book.iloc[:,0]
book_im = book_im.to_frame().reset_index()
book_im.columns = ['index','ISBN;"Book-Title";"Book-Author";"Year-Of-Publicati
on";"Publisher";"Image-URL-S";"Image-URL-M";"Image-URL-L"']
book =  DataFrame([item.split(';')[0:8] for item in book_im['ISBN;"Book-Titl
e";"Book-Author";"Year-Of-Publication";"Publisher";"Image-URL-S";"Image-URL-
M";"Image-URL-L"']])
book.columns = ['ISBN','Book-Title','Book-Author','Year-Of-Publication','Publi
sher','Image-URL-S','Image-URL-M','Image-URL-L']
```

In [13]:
```
book.head(5)
```

Out[13]:

	ISBN	Book-Title	Book-Author	Year-Of-Publication	Publisher	
0	0195153448	"Classical Mythology"	"Mark P. O. Morford"	"2002"	"Oxford University Press"	"http://images.amazon.
1	0002005018	"Clara Callan"	"Richard Bruce Wright"	"2001"	"HarperFlamingo Canada"	"http://images.amazon.
2	0060973129	"Decision in Normandy"	"Carlo D'Este"	"1991"	"HarperPerennial"	"http://images.amazon.
3	0374157065	"Flu: The Story of the Great Influenza Pandemi...	"Gina Bari Kolata"	"1999"	"Farrar Straus Giroux"	"http://images.amazon.
4	0393045218	"The Mummies of Urumchi"	"E. J. W. Barber"	"1999"	"W. W. Norton &amp	Company"

In [14]:
```
rating['Book-Title'] = rating['ISBN']
rating['Book-Title'] = rating['Book-Title'].map(book.set_index('ISBN')['Book-T
itle'])
rating['Book-Title'] = rating['Book-Title'].str.replace('"', "")
rating.head(6)
```

Out[14]:

	User-ID	ISBN	Book-Rating	Book-Title
0	276725	034545104X	0	Flesh Tones: A Novel
1	276726	0155061224	5	Rites of Passage
2	276727	0446520802	0	The Notebook
3	276729	052165615X	3	Help!: Level 1
4	276729	0521795028	6	The Amsterdam Connection : Level 4 (Cambridge ...
5	276733	2080674722	0	Les Particules Elementaires

In [15]:
```
#drop rows whose Book-Title is missing
rating = rating.dropna(subset=['Book-Title'])
```

In [16]:
```
#drop rows whose Book-Rating is 0
rating.columns = ['User-ID','ISBN','Rating','Book-Title']
rating = rating[rating.Rating != 0]
```

1.2 Users data

In [17]:
```
user.head(2)
```

Out[17]:

	User-ID;"Location";"Age"	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unname
0	1;"nyc	new york	usa";NULL	NaN	NaN	NaN	NaN
1	2;"stockton	california	usa";"18"	NaN	NaN	NaN	NaN

In [18]:
```
user.columns
```

Out[18]:
```
Index(['User-ID;"Location";"Age"', 'Unnamed: 1', 'Unnamed: 2', 'Unnamed: 3',
       'Unnamed: 4', 'Unnamed: 5', 'Unnamed: 6', 'Unnamed: 7', 'Unnamed: 8'],
      dtype='object')
```

In [19]:
```
user_im=user.iloc[:,0:3]
user_im.columns = ['User-ID;"Location";"Age"','State','Age']
from string import *
user_im['User-ID;"Location";"Age"']=user_im['User-ID;"Location";"Age"'].str.re
place('\d+\;\"', '')
user_im['Age']=user_im['Age'].str.replace('\";', '')
user_im['Age']=user_im['Age'].str.replace('\"', '')
user_im['Age2'] = user_im['Age']
user_im['Age']=user_im['Age'].str.replace('\d+', '')
user_im['Age']=user_im['Age'].str.replace('NULL', '')
user_im['Age2'] = user_im['Age2'].str.replace('[a-zA-Z]+', '')
user_im['Age2']=user_im['Age2'].str.replace('\";|\";"|"', '')
user_im.columns = ['Location','State','Country','Age']
user_im['User-ID'] = user_im.index
user= user_im
user.head(10)
```

```
D:\Program Files\Anaconda3\lib\site-packages\ipykernel_launcher.py:4: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/st
able/indexing.html#indexing-view-versus-copy
  after removing the cwd from sys.path.
D:\Program Files\Anaconda3\lib\site-packages\ipykernel_launcher.py:5: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/st
able/indexing.html#indexing-view-versus-copy
  """
D:\Program Files\Anaconda3\lib\site-packages\ipykernel_launcher.py:6: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/st
able/indexing.html#indexing-view-versus-copy

D:\Program Files\Anaconda3\lib\site-packages\ipykernel_launcher.py:7: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/st
able/indexing.html#indexing-view-versus-copy
  import sys
D:\Program Files\Anaconda3\lib\site-packages\ipykernel_launcher.py:8: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/st
able/indexing.html#indexing-view-versus-copy

D:\Program Files\Anaconda3\lib\site-packages\ipykernel_launcher.py:9: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/st
able/indexing.html#indexing-view-versus-copy
  if __name__ == '__main__':
D:\Program Files\Anaconda3\lib\site-packages\ipykernel_launcher.py:10: Settin
gWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/st
able/indexing.html#indexing-view-versus-copy
  # Remove the CWD from sys.path while we load stuff.
D:\Program Files\Anaconda3\lib\site-packages\ipykernel_launcher.py:11: Settin
```

```
gWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/st
able/indexing.html#indexing-view-versus-copy
  # This is added back by InteractiveShellApp.init_path()
D:\Program Files\Anaconda3\lib\site-packages\ipykernel_launcher.py:13: Settin
gWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/st
able/indexing.html#indexing-view-versus-copy
  del sys.path[0]
```

Out[19]:

	Location	State	Country	Age	User-ID
0	nyc	new york	usa		0
1	stockton	california	usa	18	1
2	moscow	yukon territory	russia		2
3	porto	v.n.gaia	portugal	17	3
4	farnborough	hants	united kingdom		4
5	santa monica	california	usa	61	5
6	washington	dc	usa		6
7	timmins	ontario	canada		7
8	germantown	tennessee	usa		8
9	albacete	wisconsin	spain	26	9

1.3 Remove sparse data to increase density in the Rating table

In [20]:
```
rating.shape
```

Out[20]: (351646, 4)

In [21]:
```
# Keep a copy of the original dataframe
ratingC =rating.copy()
ratingC.head(2)
```

Out[21]:

	User-ID	ISBN	Rating	Book-Title
1	276726	0155061224	5	Rites of Passage
3	276729	052165615X	3	Help!: Level 1

In [22]:
```
rating = ratingC
```

In [23]:
```
# filter users rated more than 1000 books
rating_1 = rating.groupby('User-ID').filter(lambda x: len(x) > 1000)
rating_1.shape
```

Out[23]: (19590, 4)

In [24]:
```
filter_book1 = rating_1['User-ID'].unique()
filter_book1
```

Out[24]:
```
array(['11676', '23902', '76499', '98391', '153662', '189835', '235105'], dty
pe=object)
```

In [25]:
```
# filter books rated by more than 200 users
rating_2 = rating.groupby('ISBN').filter(lambda x: len(x) > 200)
rating_2.shape
```

Out[25]: (5820, 4)

In [26]:
```
filter_book2 = rating_2['ISBN'].unique()
filter_book2
```

Out[26]:
```
array(['0385504209', '0971880107', '0671021001', '044023722X',
       '0452282152', '0312195516', '0316666343', '0671027360',
       '0316601950', '0316769487', '0142001740', '0446672211',
       '067976402X', '0375727345', '0345337662', '059035342X',
       '0060928336', '0060930535', '0743418174', '0786868716'], dtype=object)
```

In [27]:
```
rating_3 = pd.merge(rating_1,rating_2, on='ISBN', how='inner')
rating_3.columns
```

Out[27]:
```
Index(['User-ID_x', 'ISBN', 'Rating_x', 'Book-Title_x', 'User-ID_y',
       'Rating_y', 'Book-Title_y'],
      dtype='object')
```

In [28]:
```
rating_4 =  rating_3[['User-ID_x', 'ISBN', 'Rating_x', 'Book-Title_x']]
rating_4.columns = ['User-ID', 'ISBN', 'Rating', 'Title']
rating_5 =  rating_3[['User-ID_y','ISBN','Rating_y', 'Book-Title_y']]
rating_5.columns = ['User-ID', 'ISBN', 'Rating', 'Title']
rating_6 = pd.concat([rating_4, rating_5])
rating_6.shape
```

Out[28]: (15602, 4)

In [29]:
```
filter_book = rating_6.ISBN.unique()
filter_book
```

Out[29]:
```
array(['0312195516', '0316666343', '0345337662', '0375727345',
       '0385504209', '044023722X', '0446672211', '059035342X',
       '067976402X', '0743418174', '0786868716', '0971880107', '0671021001'],
dtype=object)
```

In [30]:
```
myfilter_book = filter_book[0:4]
rating_7 = rating_6[rating_6.ISBN.isin(myfilter_book)]
rating_7.shape
```

Out[30]: (6486, 4)

In [31]:
```
rating=rating_7
```

In [32]:
```
# sparsity = 1 - (ratings/users*items)
non_zero= (rating['Rating']!=0).sum()
sparsity = 1- non_zero/(rating.shape[0] * rating.shape[1])
a = "{0:.0f}%".format(sparsity*100)
print('sparsity of Book-Rating dataset:',a)
```

```
sparsity of Book-Rating dataset: 75%
```

2. Create a user-item matrix

2.1 Break book-ratings into separate training and test datasets.

In [33]:
```
# use Cross_validation.train_test_split in scikit-learn library to shuffle and
 split the data into two datasets:80% for training and 20% for testing.
from sklearn import cross_validation as cv
train_data, test_data = cv.train_test_split(rating, test_size=0.2)
```

```
D:\Program Files\Anaconda3\lib\site-packages\sklearn\cross_validation.py:44:
 DeprecationWarning: This module was deprecated in version 0.18 in favor of t
he model_selection module into which all the refactored classes and functions
are moved. Also note that the interface of the new CV iterators are different
from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [34]:
```
train_data.head(5)
```

Out[34]:

	User-ID	ISBN	Rating	Title
2250	113089	0316666343	9	The Lovely Bones: A Novel
1751	161114	0316666343	7	The Lovely Bones: A Novel
1630	98391	0316666343	9	The Lovely Bones: A Novel
2510	235105	0316666343	8	The Lovely Bones: A Novel
1059	11676	0316666343	5	The Lovely Bones: A Novel

In [35]:
```
train_data.shape
```

Out[35]: (5188, 4)

In [36]:
```
test_data.shape
```

Out[36]: (1298, 4)

In [37]:
```
test_user1 = list(test_data['User-ID'].unique())
len(test_user1)
```

Out[37]: 549

In [38]:
```
# remove the users that are not in training dataset because we can only use tr
aining dataset to calculate user bias

# uniq user-IDs of training dataset
desired_test_user = list(train_data['User-ID'].unique())
len(desired_test_user)
```

Out[38]: 1288

In [39]:
```
# remove the users that are not in the list of uniq user-IDs of training datas
et
test_data =test_data[test_data['User-ID'].isin(desired_test_user)]
test_data.shape
```

Out[39]: (1220, 4)

In [40]:
```
# Some users's data were dropped
test_user = list(test_data['User-ID'].unique())
len(test_user)
```

Out[40]: 494

2.2 calculate the raw average of training dataset

In [41]:
```
non_zero = (train_data['Rating']!=0).sum()
non_zero
```

Out[41]: 5188

In [42]:
```
# sparsity = 1 - (ratings/users*items)
non_zero= (train_data['Rating']!=0).sum()
sparsity_train = 1- non_zero/(train_data.shape[0] * train_data.shape[1])
a = "{0:.0f}%".format(sparsity_train*100)
print('sparsity of training dataset:',a)
```

```
sparsity of training dataset: 75%
```

In [43]:
```
#pivot train_data and test_data to generate two user-item matrices, one for tr
aining and another for testing
train_matrix = train_data.pivot_table(index='User-ID', columns='ISBN',
values='Rating')
test_matrix = test_data.pivot_table(index='User-ID', columns='ISBN', values='R
ating')
```

In [44]:
```
train_matrix.head(5)
```

Out[44]:

ISBN	0312195516	0316666343	0345337662	0375727345
User-ID				
100043	NaN	NaN	NaN	8.0
100251	NaN	NaN	5.0	NaN
100343	NaN	3.0	NaN	NaN
100459	NaN	NaN	NaN	8.0
100471	NaN	NaN	NaN	8.0

In [45]:
```
train_matrix.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1288 entries, 100043 to 99997
Data columns (total 4 columns):
0312195516    340 non-null float64
0316666343    633 non-null float64
0345337662    171 non-null float64
0375727345    201 non-null float64
dtypes: float64(4)
memory usage: 50.3+ KB
```

In [46]:
```
train_matrix = train_matrix.fillna(0)
train_matrix.head(5)
```

Out[46]:

ISBN	0312195516	0316666343	0345337662	0375727345
User-ID				
100043	0.0	0.0	0.0	8.0
100251	0.0	0.0	5.0	0.0
100343	0.0	3.0	0.0	0.0
100459	0.0	0.0	0.0	8.0
100471	0.0	0.0	0.0	8.0

In [73]:
```
count_non_zero = (train_matrix != 0).sum()
count_non_zero = count_non_zero.sum()
count_non_zero
```

Out[73]: 1345

In [78]:
```
sum_non_zero = train_matrix.sum(axis=1)
sum_non_zero = sum_non_zero.sum()
sum_non_zero
```

Out[78]: 10743.0

In [79]:
```
raw_avg = sum_non_zero/count_non_zero
raw_avg
```

Out[79]: 7.987360594795539

2.3 Calculate the RMSE for raw average for both your training data and your test data.

In [80]:
```
#RMSE for taining dataset

train_matrix = train_matrix.fillna(0)
se = 0
for r in range(train_matrix.shape[0]):
    for c in range(train_matrix.shape[1]):
        if train_matrix.iloc[r,c] != 0:
            se = se + (train_matrix.iloc[r,c]- raw_avg)**2
mse = se/count_non_zero
rmse_1_train = sqrt(mse)
print('RMSE for taining dataset based on raw_average is:', rmse_1_train)
```

```
RMSE for taining dataset based on raw_average is: 1.6457188190300436
```

In [81]:
```
#RMSE for testing dataset

test_matrix = test_matrix.fillna(0)
se = 0
for r in range(test_matrix.shape[0]):
    for c in range(test_matrix.shape[1]):
        if test_matrix.iloc[r,c] != 0:
            se = se + (test_matrix.iloc[r,c]- raw_avg)**2
mse = se/count_non_zero
rmse_1_test = sqrt(mse)
print('RMSE for testing dataset based on raw_average is:', rmse_1_test)
```

```
RMSE for testing dataset based on raw_average is: 1.0257074482614486
```

2.4 Using the training data to calculate the bias for each user and each item.

In [110]:
```
train_matrix = train_matrix.replace(0, np.NaN)
train_matrix = train_matrix.dropna(how='all')
train_matrix = train_matrix.dropna(how='all',axis=1)
train_matrix.head(5)
```

Out[110]:

ISBN	0312195516	0316666343	0345337662	0375727345
User-ID				
100043	NaN	NaN	NaN	8.0
100251	NaN	NaN	5.0	NaN
100343	NaN	3.0	NaN	NaN
100459	NaN	NaN	NaN	8.0
100471	NaN	NaN	NaN	8.0

In [111]:
```
train_matrix.shape
```

Out[111]:
```
(1288, 4)
```

In [116]:
```
user_bias = train_matrix.mean(axis=1)
for i in range(user_bias.shape[0]):
    user_bias[i] = user_bias[i] - raw_avg
user_bias.head(5)
```

Out[116]:
```
User-ID
100043    0.212221
100251   -2.787779
100343   -4.787779
100459    0.212221
100471    0.212221
dtype: float64
```

In [117]:
```
book_bias = train_matrix.mean(axis=0)
for i in range(book_bias.shape[0]):
    book_bias[i] = book_bias[i] - raw_avg
book_bias.head()
```

Out[117]:
```
ISBN
0312195516    0.379868
0316666343    0.384416
0345337662   -0.039241
0375727345   -0.484297
dtype: float64
```

In [118]:
```
train_matrix.index
```

Out[118]:
```
Index(['100043', '100251', '100343', '100459', '100471', '100781', '1008',
       '10112', '101154', '101305',
       ...
       '97874', '98356', '98391', '98551', '98783', '98904', '99172', '9929
8',
       '99349', '99997'],
      dtype='object', name='User-ID', length=1288)
```

2.5 From the raw average, and the appropriate user and item biases, calculate the baseline predictors for every user-item combination.

In [120]:
```
# first create an empty DataFrame:
predictor_matrix = pd.DataFrame(index=train_matrix.index, columns=book_bias.in
dex)
predictor_matrix = predictor_matrix.fillna(0) # with 0s rather than NaNs
predictor_matrix = DataFrame(predictor_matrix)

# baseline predictor
for r in range(train_matrix.shape[0]):
    for c in range(book_bias.shape[0]):
        predictor_matrix.iloc[r,c] = raw_avg + user_bias[r] + book_bias[c]
predictor_matrix.head(20)
```

Out[120]:

ISBN	0312195516	0316666343	0345337662	0375727345
User-ID				
100043	8.379868	8.384416	7.960759	7.515703
100251	5.379868	5.384416	4.960759	4.515703
100343	3.379868	3.384416	2.960759	2.515703
100459	8.379868	8.384416	7.960759	7.515703
100471	8.379868	8.384416	7.960759	7.515703
100781	10.379868	10.384416	9.960759	9.515703
1008	9.379868	9.384416	8.960759	8.515703
10112	7.379868	7.384416	6.960759	6.515703
101154	8.379868	8.384416	7.960759	7.515703
101305	5.379868	5.384416	4.960759	4.515703
101338	9.379868	9.384416	8.960759	8.515703
101606	8.379868	8.384416	7.960759	7.515703
101703	9.379868	9.384416	8.960759	8.515703
101857	9.379868	9.384416	8.960759	8.515703
101935	6.379868	6.384416	5.960759	5.515703
10199	7.379868	7.384416	6.960759	6.515703
10314	9.379868	9.384416	8.960759	8.515703
103304	8.379868	8.384416	7.960759	7.515703
103336	10.379868	10.384416	9.960759	9.515703
103734	7.379868	7.384416	6.960759	6.515703

In [122]:
```
predictor_matrix[predictor_matrix >10] = 10
predictor_matrix[predictor_matrix <1] = 1
predictor_matrix.head(20)
```

Out[122]:

ISBN	0312195516	0316666343	0345337662	0375727345
User-ID				
100043	8.379868	8.384416	7.960759	7.515703
100251	5.379868	5.384416	4.960759	4.515703
100343	3.379868	3.384416	2.960759	2.515703
100459	8.379868	8.384416	7.960759	7.515703
100471	8.379868	8.384416	7.960759	7.515703
100781	10.000000	10.000000	9.960759	9.515703
1008	9.379868	9.384416	8.960759	8.515703
10112	7.379868	7.384416	6.960759	6.515703
101154	8.379868	8.384416	7.960759	7.515703
101305	5.379868	5.384416	4.960759	4.515703
101338	9.379868	9.384416	8.960759	8.515703
101606	8.379868	8.384416	7.960759	7.515703
101703	9.379868	9.384416	8.960759	8.515703
101857	9.379868	9.384416	8.960759	8.515703
101935	6.379868	6.384416	5.960759	5.515703
10199	7.379868	7.384416	6.960759	6.515703
10314	9.379868	9.384416	8.960759	8.515703
103304	8.379868	8.384416	7.960759	7.515703
103336	10.000000	10.000000	9.960759	9.515703
103734	7.379868	7.384416	6.960759	6.515703

2.6 Calculate the RMSE for the baseline predictors for both the training data and your test data.

2.6.1 RMSE for the baseline predictors for the training data

In [125]:
```
#RMSE
train_matrix = train_matrix.fillna(0)
se = 0
for r in range(train_matrix.shape[0]):
    for c in range(train_matrix.shape[1]):
        if train_matrix.iloc[r,c] != 0:
            se = se + (train_matrix.iloc[r,c]- predictor_matrix.iloc[r,c])**2

mse = se/count_non_zero
rmse_2_train = sqrt(mse)
print('RMSE of baseline predictor for training dataset:', rmse_2_train)
```

```
RMSE of baseline predictor for training dataset: 0.448324533462584
```

In [126]:
```
improve_train = 1- (rmse_2_train/rmse_1_train)
improve_train = "{0:.0f}%".format(abs(improve_train)*100)
print('baseline predictor improved RMSE by', improve_train, 'for training data
set:')
```

```
baseline predictor improved RMSE by 73% for training dataset:
```

2.6.2 RMSE for the baseline predictors for the testing data

In [127]:
```
# evaluate the prediction accuracy in the test dataset
test_matrix.head(10)
```

Out[127]:

ISBN	0312195516	0316666343	0345337662	0375727345
User-ID				
100471	0.0	0.0	0.0	8.0
100781	0.0	10.0	0.0	0.0
101154	8.0	0.0	0.0	0.0
103734	0.0	7.0	0.0	0.0
103835	0.0	6.0	0.0	0.0
103876	7.0	0.0	0.0	0.0
104880	0.0	8.0	0.0	0.0
106225	0.0	8.0	0.0	0.0
106412	0.0	9.0	0.0	0.0
106846	0.0	6.0	0.0	0.0

In [128]:
```
test_matrix.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 494 entries, 100471 to 99997
Data columns (total 4 columns):
0312195516    494 non-null float64
0316666343    494 non-null float64
0345337662    494 non-null float64
0375727345    494 non-null float64
dtypes: float64(4)
memory usage: 39.3+ KB
```

In [132]:
```
test_matrix = test_matrix.replace(0, np.NaN)
test_matrix = test_matrix.dropna(how='all')
test_matrix = test_matrix.dropna(how='all',axis=1)
test_matrix.head(5)
```

Out[132]:

ISBN	0312195516	0316666343	0345337662	0375727345
User-ID				
100471	NaN	NaN	NaN	8.0
100781	NaN	10.0	NaN	NaN
101154	8.0	NaN	NaN	NaN
103734	NaN	7.0	NaN	NaN
103835	NaN	6.0	NaN	NaN

In [133]:
```
user_bias_te = test_matrix.mean(axis=1)-raw_avg
user_bias_te
```

Out[133]:

User-ID	
100471	0.212221
100781	2.212221
101154	0.212221
103734	-0.787779
103835	-1.787779
103876	-0.787779
104880	0.212221
106225	0.212221
106412	1.212221
106846	-1.787779
106874	2.212221
1075	-0.787779
108158	1.212221
108405	1.212221
109122	2.212221
110165	1.212221
110695	1.212221
111136	1.212221
111199	-0.787779
112345	-0.787779
112977	-0.787779
113270	1.212221
113519	1.212221
113680	-2.787779
113989	0.212221
113992	-2.787779
114368	-2.787779
114389	-0.787779
114813	1.212221
115003	0.212221
...	...
78448	1.212221
80084	1.212221
80497	0.212221
80538	0.212221
80945	1.212221
81215	-2.787779
81263	1.212221
81318	-5.787779
81977	-0.787779
8253	2.212221
82563	2.212221
82797	2.212221
82888	2.212221
85526	0.212221
88229	2.212221
88937	-0.787779
90763	2.212221
91103	1.212221
92566	0.212221
92964	1.212221
93046	-1.787779
93800	-1.787779
94347	1.212221
94639	1.212221
9486	2.212221

```
95359    -0.121113
96667    -2.787779
97874     1.212221
98391     1.212221
99997     1.212221
Length: 494, dtype: float64
```

In [134]:
```
book_bias_te = test_matrix.mean(axis=0)-raw_avg
book_bias_te
```

Out[134]:
```
ISBN
0312195516    0.221070
0316666343    0.472409
0345337662   -0.537779
0375727345   -0.376821
dtype: float64
```

In [135]:
```
# first create an empty DataFrame for test dataset predictor:
predictor_matrix_te = pd.DataFrame(index=test_matrix.index, columns=book_bias_
te.index)
predictor_matrix_te = predictor_matrix_te.fillna(0) # with 0s rather than NaNs
```

In [136]:
```
# baseline predictor for test dataset
for r in range(test_matrix.shape[0]):
    for c in range(test_matrix.shape[1]):
        predictor_matrix_te.iloc[r,c] = raw_avg + user_bias_te[r] + book_bias_
te[c]
predictor_matrix_te.head(5)
```

Out[136]:

ISBN	0312195516	0316666343	0345337662	0375727345
User-ID				
100471	8.22107	8.472409	7.462221	7.623179
100781	10.22107	10.472409	9.462221	9.623179
101154	8.22107	8.472409	7.462221	7.623179
103734	7.22107	7.472409	6.462221	6.623179
103835	6.22107	6.472409	5.462221	5.623179

In [137]:
```
predictor_matrix_te[predictor_matrix_te >10] = 10
predictor_matrix_te[predictor_matrix_te <1] = 1
predictor_matrix_te.head(5)
```

Out[137]:

ISBN	0312195516	0316666343	0345337662	0375727345
User-ID				
100471	8.22107	8.472409	7.462221	7.623179
100781	10.00000	10.000000	9.462221	9.623179
101154	8.22107	8.472409	7.462221	7.623179
103734	7.22107	7.472409	6.462221	6.623179
103835	6.22107	6.472409	5.462221	5.623179

In [ ]:
```
improve_test = 1- (rmse_2_test/rmse_1_test)
improve_test = "{0:.0f}%".format(abs(improve_test)*100)
print('baseline predictor improved RMSE by', improve_test, 'for testing datase
t:')
```

In [140]:
```
count_non_zero_te = test_matrix.sum()
count_non_zero_te = count_non_zero_te.sum()
count_non_zero_te
```

Out[140]: 4110.0

In [141]:
```
#RMSE
test_matrix = test_matrix.fillna(0)
se_te = 0
for r in range(test_matrix.shape[0]):
    for c in range(test_matrix.shape[1]):
        if test_matrix.iloc[r,c] != 0:
            se_te= se_te + (test_matrix.iloc[r,c]-
predictor_matrix_te.iloc[r,c])**2

mse_te = se_te/count_non_zero_te
rmse_2_test= sqrt(mse_te)
print('RMSE of baseline predictor for testing dataset :', rmse_2_test)
```

```
RMSE of baseline predictor for testing dataset : 0.15681359996270988
```

In [142]:
```
improve_test = 1- (rmse_2_test/rmse_1_test)
improve_test = "{0:.0f}%".format(abs(improve_test)*100)
print('baseline predictor improved RMSE by', improve_test, 'for testing datase
t:')
```

```
baseline predictor improved RMSE by 85% for testing dataset:
```

2.7 Calculate the RMSE for the neighborhood predictors for both the training data and your test data.

**2.7.1 Calculate the baseline error based on train dataset.

In [145]:
```
# baseline errors which are based on train set
train_matrix = train_matrix.replace(0, np.NaN)
train_error = train_matrix - predictor_matrix
train_error.head(10)
```

Out[145]:

ISBN	0312195516	0316666343	0345337662	0375727345
User-ID				
100043	NaN	NaN	NaN	0.484297
100251	NaN	NaN	0.039241	NaN
100343	NaN	-0.384416	NaN	NaN
100459	NaN	NaN	NaN	0.484297
100471	NaN	NaN	NaN	0.484297
100781	NaN	0.000000	NaN	NaN
1008	-0.379868	NaN	NaN	NaN
10112	NaN	NaN	0.039241	NaN
101154	-0.379868	NaN	NaN	NaN
101305	-0.379868	NaN	NaN	NaN

In [146]:
```
train_error.shape
```

Out[146]: (1288, 4)

2.7.2 Calculate the consine similarity between books based on baseline error derived from train dataset.

In [147]:
```
# consine sililarity of training dataset
# first create an empty DataFrame:
sim_1 = pd.DataFrame(index=book_bias.index, columns=book_bias.index)
sim_1 = sim_1.fillna(0) # with 0s rather than NaNs
```

In [148]:
```
train_error = pd.DataFrame(train_error)
sim_1 = pd.DataFrame(sim_1)
```

In [149]:
```
train_error = train_error.fillna(0)
```

In [150]:
```
from scipy.spatial.distance import cosine
for r in range(sim_1.shape[1]):
    for c in range(sim_1.shape[1]):
        sim_1.iloc[r,c] = 1- cosine(train_error.iloc[:,r],
train_error.iloc[:,c])
sim_1.columns = book_bias.index
sim_1
```

Out[150]:

ISBN	0312195516	0316666343	0345337662	0375727345
ISBN				
0312195516	1.000000	-0.282398	-0.164486	-0.095207
0316666343	-0.282398	1.000000	-0.109992	-0.152710
0345337662	-0.164486	-0.109992	1.000000	-0.017150
0375727345	-0.095207	-0.152710	-0.017150	1.000000

In [151]:
```
# try a different mathod to calculate cosine similarity
from sklearn.metrics import pairwise_distances
```

In [152]:
```
sim_2 = 1- pairwise_distances(train_error.T,  metric="cosine")
sim_2 = pd.DataFrame(sim_2)
sim_2.columns = book_bias.index
sim_2.index = book_bias.index
sim_2
```

Out[152]:

ISBN	0312195516	0316666343	0345337662	0375727345
ISBN				
0312195516	1.000000	-0.282398	-0.164486	-0.095207
0316666343	-0.282398	1.000000	-0.109992	-0.152710
0345337662	-0.164486	-0.109992	1.000000	-0.017150
0375727345	-0.095207	-0.152710	-0.017150	1.000000

the results of the baseline error from two differnt methods are the same.

In [153]:
```
# neighbourhood baseline error
neighbour = Series(index=book_bias.index)
neighbour = neighbour.fillna('A')
for c in range(sim_1.shape[1]):
    neighbour[c]= sim_1.iloc[:,c].argmin()
neighbour
```

Out[153]:
```
ISBN
0312195516    0316666343
0316666343    0312195516
0345337662    0312195516
0375727345    0316666343
dtype: object
```

In [154]:
```
for c in range(sim_1.shape[1]):
    print('The neighbour of',sim_1.columns[c],'is',neighbour[c])
```

```
The neighbour of 0312195516 is 0316666343
The neighbour of 0316666343 is 0312195516
The neighbour of 0345337662 is 0312195516
The neighbour of 0375727345 is 0316666343
```

In [155]:
```
# create empty Serie
neighbour_baseline_error = Series(index=book_bias.index)
neighbour_baseline_error = neighbour_baseline_error.fillna(0)

# find the closest neighbour with the maximum absolute error value
for i in range(book_bias.shape[0]):
    neighbour_baseline_error[i] = sim_1.iloc[:,i].min()

neighbour_baseline_error
```

Out[155]:
```
ISBN
0312195516   -0.282398
0316666343   -0.282398
0345337662   -0.164486
0375727345   -0.152710
dtype: float64
```

In [156]:
```
# neighbourhood predictor for training dataset

# create an empty DataFrame:
neighbour_predictor = pd.DataFrame(index=train_matrix.index,
columns=book_bias.index)
neighbour_predictor = neighbour_predictor.fillna(0) # with 0s rather than NaNs


# fill with predicted ratings
for r in range(predictor_matrix.shape[0]):
    for c in range(predictor_matrix.shape[1]):
        neighbour_predictor.iloc[r,c]  = predictor_matrix.iloc[r,c] + neighbou
r_baseline_error[c]
neighbour_predictor.head(5)
```

Out[156]:

ISBN	0312195516	0316666343	0345337662	0375727345
User-ID				
100043	8.097469	8.102018	7.796272	7.362993
100251	5.097469	5.102018	4.796272	4.362993
100343	3.097469	3.102018	2.796272	2.362993
100459	8.097469	8.102018	7.796272	7.362993
100471	8.097469	8.102018	7.796272	7.362993

2.7.3 Calculate the RMSE for the neighborhood predictors for the training data and your test data.

RMSE for the training data

In [157]:
```
# RMSE
train_matrix = train_matrix.fillna(0)
se = 0
for r in range(train_matrix.shape[0]):
    for c in range(train_matrix.shape[1]):
        if train_matrix.iloc[r,c] != 0:
            se= se + (train_matrix.iloc[r,c]-
neighbour_predictor.iloc[r,c])**2

mse = se/count_non_zero
rmse_nghb_train = sqrt(mse)
print('RMSE of neighborhood predictor for training dataset :',
rmse_nghb_train)
```

```
RMSE of neighborhood predictor for training dataset : 0.407586030788146
```

In [160]:
```
improve_train = 1- (rmse_nghb_train/rmse_2_train)
improve_train_nb = "{0:.0f}%".format(abs(improve_train)*100)
print('Neighborhood similirity improve RMSE by', improve_train_nb, 'for traini
ng dataset:')
```

```
Neighborhood similirity improve RMSE by 9% for training dataset:
```

RMSE for the testing data

In [166]:
```
# neighbourhood predictor for testing dataset

# create an empty DataFrame:
neighbour_predictor_te = pd.DataFrame(index=test_matrix.index, columns=book_bi
as.index)
neighbour_predictor_te = neighbour_predictor_te.fillna(0) # with 0s rather tha
n NaNs

# fill with predicted ratings
for r in range(test_matrix.shape[0]):
    for c in range(book_bias_te.shape[0]):
        neighbour_predictor_te.iloc[r,c]  = predictor_matrix_te.iloc[r,c] + ne
ighbour_baseline_error[c]

neighbour_predictor_te.head()
```

Out[166]:

ISBN	0312195516	0316666343	0345337662	0375727345
User-ID				
100471	7.938672	8.190010	7.297734	7.470469
100781	9.717602	9.717602	9.297734	9.470469
101154	7.938672	8.190010	7.297734	7.470469
103734	6.938672	7.190010	6.297734	6.470469
103835	5.938672	6.190010	5.297734	5.470469

In [167]:
```
# RMSE
test_matrix = test_matrix.fillna(0)
se = 0
for r in range(test_matrix.shape[0]):
    for c in range(test_matrix.shape[1]):
        if test_matrix.iloc[r,c] != 0:
            se = se + (test_matrix.iloc[r,c]-
neighbour_predictor_te.iloc[r,c])**2

mse = se/count_non_zero
rmse_nghb_test = sqrt(mse)
print('RMSE of neighborhood predictor for testing dataset :', rmse_nghb_test)
```

```
RMSE of neighborhood predictor for testing dataset : 0.22794753164699097
```

In [169]:
```
improve_test = 1- (rmse_nghb_test/rmse_2_test)
improve_test_nb = "{0:.0f}%".format(improve_test*100)
print('neighborhood similirity did not improve but reduce RMSE for baseline pr
edictor by', improve_test_nb, 'for testing dataset:')
```

```
neighborhood similirity did not improve but reduce RMSE for baseline predicto
r by -45% for testing dataset:
```

3. Summary

In [171]:
```
print('This system recommends books to readers.')
print('  1. Raw average rating for training dataset is',raw_avg)

print('  2.RMSE for raw average for the training data is',rmse_1_train)
print('  3.RMSE for raw average for the test data is',rmse_1_test)

print('  4.RMSE for the baseline predictors for the training data is',rmse_2_t
rain)
print('  5.RMSE for the baseline predictors for the testing data is',rmse_2_te
st)
print('  6.Baseline predictor improved RMSE by', improve_train, 'for training
 dataset:')
print('  7.Baseline predictor improved RMSE by', improve_test, 'for testing da
taset:')

print('  8.RMSE for the neighborhood predictor for the training data is',rmse_
nghb_train)
print('  9.RMSE for the neighborhood predictor for the training data is',rmse_
nghb_test)
print('  10.Neighborhood predictors did not improve but reduced RMSE by', impr
ove_train_nb, 'for training dataset:')
print('  11.Neighborhood similirity did not improve but reduce RMSE for baseli
ne predictor by', improve_test_nb, 'for testing dataset:')
```

```
This system recommends books to readers.
  1. Raw average rating for training dataset is 7.78777949113
  2.RMSE for raw average for the training data is 1.6457188190300436
  3.RMSE for raw average for the test data is 1.0257074482614486
  4.RMSE for the baseline predictors for the training data is 0.4483245334625
84
  5.RMSE for the baseline predictors for the testing data is 0.15681359996270
988
  6.Baseline predictor improved RMSE by 0.09086833227661839 for training data
set:
  7.Baseline predictor improved RMSE by -0.4536209340337616 for testing datas
et:
  8.RMSE for the neighborhood predictor for the training data is 0.4075860307
88146
  9.RMSE for the neighborhood predictor for the training data is 0.2279475316
4699097
  10.Neighborhood predictors did not improve but reduced RMSE by 9% for train
ing dataset:
  11.Neighborhood similirity did not improve but reduce RMSE for baseline pre
dictor by -45% for testing dataset:
```