Names:	HEMASREE
	SREEJA REDDY NALLA
	NEHA LOKRE
Email:	hodcp@mst.edu
	sng4f@mst.edu
	nlnk3@mst.edu
Course:	CS 5402 SS2022
Assignment:	Group Project
Date:	07-28-2022
GitHublink:	https://git-classes.mst.edu/sng4f/semester_project

https://git-classes.mst.edu/sng4f/semester_project

```
!pip install surprise
import surprise
```

```
    Requirement already satisfied: surprise in c:\users\nehal\anaconda3\lib\site-packages (0.1)
    Requirement already satisfied: scikit-surprise in c:\users\nehal\anaconda3\lib\site-packages (from surprise) (1.1.1)
    Requirement already satisfied: numpy>=1.11.2 in c:\users\nehal\anaconda3\lib\site-packages (from scikit-surprise->surprise) (1.21.5)
    Requirement already satisfied: scipy>=1.0.0 in c:\users\nehal\anaconda3\lib\site-packages (from scikit-surprise->surprise) (1.7.3)
    Requirement already satisfied: six>=1.10.0 in c:\users\nehal\anaconda3\lib\site-packages (from scikit-surprise->surprise) (1.16.0)
    Requirement already satisfied: joblib>=0.11 in c:\users\nehal\anaconda3\lib\site-packages (from scikit-surprise->surprise) (1.1.0)
```

```
# Libraries for data preparation & visualization
import numpy as np
import pandas as pd
import plotly.offline as py
import plotly.graph_objs as go
import plotly.io as pio
pio.renderers.default = "png"

# Ignore printing warnings for general readability
import warnings
warnings.filterwarnings("ignore")

# pip install scikit-surprise
# Importing libraries for model building & evaluation
from sklearn.model_selection import train_test_split
from surprise import Reader, Dataset
from surprise.model_selection import train_test_split, cross_validate, GridSearchCV
from surprise import KNNBasic, KNNWithMeans, KNNWithZScore, KNNBaseline, SVD
from surprise import accuracy
```

Concept Description

In this assignment,our client have acquired Goodreads data and would like to build a simple four book recommender system based off the data present in this three files:

- goodreads_books.csv
- book_tags.csv
- tags.csv

  our client would like to use three different data mining modeling techniques to verify the recommendations are "the best recommendations possible", and assess which technique is the best. They would like to use at least one modeling technique that we have researched on our own

Data Collection

Our client i.e our instructor perrykoob has provied a three data set. The detailed data of these files is explained below:

1. goodreads_books.csv contains metadata about each book.
2. book_tags.csv contains a list of each book, a tag id for a keyword the book has been tagged with, and the number of people who have tagged the book with that keyword.
3. tags.csv contains a list of tag ids and their corresponding keyword tags. goodreads_books.csv and book_tags.csv are tied together through goodreads_book_id . book_tags.csv and tags.csv are tied together through tag_id .

Example Description

Level of Measurement

TAGS.CSV FILE EXAMPLES

- Tagid : A tag is a unique string, but can be more than one word long. Every tag has a separate id—so "history," "History" and "HISTORY" have three separate IDs. This is a nominal attribute.
- Tagname : tag name represent the genre of the book.This is a nominal attribute.

Book_tags.CSV FILE EXAMPLES

- Goodreads_book id : In order to recognize the book we need a bookid. This will help in counting the number of books available.This is a nominal attribute.

- Tag_id : A tag is a unique string, but can be more than one word long.every book has a unique tag id.This is a nominal attribute.
- count : count attribute gives a total count of books.This is a nominal attribute.

Goodreads_books.CSV FILE EXAMPLES

- Goodreads_book id : In order to recognize the book we need a bookid. This will help in counting the number of books available.This is a nominal attribute.
- title : This attribute has different titles of books.This is a nominal attribute.
- LINK : This attribute has links of each book.This is a nominal attribute.
- series : A novel sequence is a set or series of novels which share common themes, characters, or settings, but where each novel has its own title and free-standing storyline, and can thus be read independently or out of sequence.This is a nominal attribute.
- cover_link : This attribute has coverlinks of each book. This is a nominal attribute
- author : This attribute has name of the author of each book. This is a nominal attribute
- author link: This attribute has name of the link of each book.This is a nominal attribute.
- rating_count: This attribute has rating count of each book.This is a nominal attribute.
- review_count: This attribute has a count of review of each book.This is a nominal attribute.
- average_rating: This attribute has average rating of each book.This is a nominal attribute.
- five_star_ratings:This attribute has 5 star rating data of every book.This is a nominal attribute.
- four_star_ratings: This attribute has 4 star rating data of every book.This is a nominal attribute.
- three_star_ratings: This attribute has 3 star rating data of every book.This is a nominal attribute.
- two_star_ratings: This attribute has 2 star rating data of every book.This is a nominal attribute.
- one_star_ratings: This attribute has 1 star rating data of every book.This is a nominal attribute.
- number_of_pages: This attribute has count of number of pages of every book.This is a nominal attribute.
- date_published: It consists of date published of each book.This is a nominal attribute.
- publisher: This attribute comprises of the publisher's information.This is a nominal attribute.
- original_title: It has the orginal title data.This is a nominal attribute.
- isbn and isbn13: The International Standard Book Number (ISBN) is a numeric commercial book identifier that is intended to be unique.This is a nominal attribute.
- settings: Location where the books relesed.This is a nominal attribute.
- characters: characters in the book.This is a nominal attribute.
- awards: This attribute has the award information any book received.This is a nominal attribute.
- amazon_redirect_link:This attribute has amazon direct link of every book.This is a nominal attribute.
- worldcat_redirect_link:This attribute has worldcat direct link of every book.This is a nominal attribute.
- description: This attribute has description of every book.This is a nominal attribute.

Data Importing and Wrangling

The data that is been created using (.csv) is imported from source and loaded into the program files. The results of each search is read from the respective comma separated value file (csv) into separate dataframes. A study is done to make sure the data is read into the models.

```
# Loading the dataset
def loaddata(filename):
    df = pd.read_csv(f'{filename}.csv',error_bad_lines=False,warn_bad_lines=False,encoding='latin-1')
    return df

book   = loaddata("goodreads_books")
booktag   = loaddata("book_tags")
tag   = loaddata("tags")
```

```
book
```

0	630104	Inner Circle	https://www.goodreads.com//book/show/630104.In...	(Private #5)	https://i.gr-assets.com/images/S/compressed.ph...	Brian, Julian Peploe	https://www.goodre
1	9487	A Time to Embrace	https://www.goodreads.com//book/show/9487.A_Ti...	(Timeless Love #2)	https://i.gr-assets.com/images/S/compressed.ph...	Karen Kingsbury	https://www.goodre
2	6050894	Take Two	https://www.goodreads.com//book/show/6050894-t...	(Above the Line #2)	https://i.gr-assets.com/images/S/compressed.ph...	Karen Kingsbury	https://www.goodre
3	39030	Reliquary	https://www.goodreads.com//book/show/39030.Rel...	(Pendergast #2)	https://i.gr-assets.com/images/S/compressed.ph...	Douglas Preston, Lincoln Child	https://www.goodre
4	998	The Millionaire Next Door: The Surprising Secr...	https://www.goodreads.com//book/show/998.The_M...	NaN	https://i.gr-assets.com/images/S/compressed.ph...	Thomas J. Stanley, William D. Danko	https://www.goodre

```
booktag
```

	goodreads_book_id	tag_id	count
0	1	30574	167697
1	1	11305	37174
2	1	11557	34173
3	1	8717	12986
4	1	33114	12716
...	...	...	...
9967	31538647	30574	23052
9968	31845516	30574	34617
9969	32075671	30574	5858
9970	32075671	33114	2010
9971	33288638	30574	14116

9972 rows × 3 columns

Life Dewey

```
tag
```

	tag_id	tag_name
0	1691	adventure
1	2516	angels
2	4605	biography
3	4949	book-club
4	5207	books-i-own
...	...	...
78	31745	vampires
79	32865	writing
80	32989	ya
81	33114	young-adult
82	33268	zombies

83 rows × 2 columns

Exploratory Data Analysis

Exploring the tag data

```
tag.shape
```

```
(83, 2)
```

There are 83 rows and 2 columns in tag.csv file

```
tag.head(10)
```

	tag_id	tag_name
0	1691	adventure
1	2516	angels
2	4605	biography
3	4949	book-club
4	5207	books-i-own
5	5951	business

```
# Check datatypes & missing values
tag.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 83 entries, 0 to 82
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   tag_id    83 non-null     int64
 1   tag_name  83 non-null     object
dtypes: int64(1), object(1)
memory usage: 1.4+ KB
```

There are no missing values

```
# Check for duplicate values
print(f'Duplicate entries: {tag.duplicated().sum()}')
```

```
Duplicate entries: 0
```

```
tag['tag_name'].unique()
```

```
array(['adventure', 'angels', 'biography', 'book-club', 'books-i-own',
       'business', 'children', 'childrens', 'classic', 'classics',
       'comics', 'contemporary', 'crime', 'currently-reading',
       'dan-brown', 'dragons', 'drama', 'dystopia', 'dystopian',
       'fantasy', 'favorites', 'favourites', 'feminism', 'fiction',
       'food', 'graphic-novel', 'graphic-novels', 'harry-potter',
       'historical', 'historical-fiction', 'historical-romance',
       'history', 'horror', 'humor', 'hunger-games', 'literature',
       'magic', 'manga', 'memoir', 'mystery', 'mythology', 'new-adult',
       'non-fiction', 'nonfiction', 'owned', 'owned-books', 'paranormal',
       'paranormal-romance', 'philosophy', 'picture-books', 'plays',
       'poetry', 'psychology', 're-read', 'read-in-2016',
       'realistic-fiction', 'religion', 'romance', 'school', 'sci-fi',
       'science', 'science-fiction', 'scifi', 'self-help', 'series',
       'shakespeare', 'short-stories', 'star-wars', 'steampunk',
       'stephen-king', 'thriller', 'time-travel', 'to-read', 'travel',
       'true-crime', 'twilight', 'urban-fantasy', 'vampire', 'vampires',
       'writing', 'ya', 'young-adult', 'zombies'], dtype=object)
```

From above we can see that there are erros in the spellings.

```
tag['tag_name']=tag['tag_name'].replace(['childrens'],'children')
tag['tag_name']=tag['tag_name'].replace(['classics'],'classic')
tag['tag_name']=tag['tag_name'].replace(['dystopian'],'dystopia')
tag['tag_name']=tag['tag_name'].replace(['favourites'],'favorites')
tag['tag_name']=tag['tag_name'].replace(['graphic-novels'],'graphic-novel')
tag['tag_name']=tag['tag_name'].replace(['historical-fiction'],'historical')
tag['tag_name']=tag['tag_name'].replace(['nonfiction'],'non-fiction')
tag['tag_name']=tag['tag_name'].replace(['owned-books'],'owned')
tag['tag_name']=tag['tag_name'].replace(['science-fiction','scifi'],'sci-fi')
tag['tag_name']=tag['tag_name'].replace(['vampires'],'vampire')
```

```
tag['tag_name'].unique()
```

```
array(['adventure', 'angels', 'biography', 'book-club', 'books-i-own',
       'business', 'children', 'classic', 'comics', 'contemporary',
       'crime', 'currently-reading', 'dan-brown', 'dragons', 'drama',
       'dystopia', 'fantasy', 'favorites', 'feminism', 'fiction', 'food',
       'graphic-novel', 'harry-potter', 'historical',
       'historical-romance', 'history', 'horror', 'humor', 'hunger-games',
       'literature', 'magic', 'manga', 'memoir', 'mystery', 'mythology',
       'new-adult', 'non-fiction', 'owned', 'paranormal',
       'paranormal-romance', 'philosophy', 'picture-books', 'plays',
       'poetry', 'psychology', 're-read', 'read-in-2016',
       'realistic-fiction', 'religion', 'romance', 'school', 'sci-fi',
       'science', 'self-help', 'series', 'shakespeare', 'short-stories',
       'star-wars', 'steampunk', 'stephen-king', 'thriller',
       'time-travel', 'to-read', 'travel', 'true-crime', 'twilight',
       'urban-fantasy', 'vampire', 'writing', 'ya', 'young-adult',
       'zombies'], dtype=object)
```

Now we can see that the mistakes in the dataset has beeen rectified.

```
tag = pd.DataFrame(tag.tag_name.value_counts(normalize=True)).reset_index()
tag.columns = ['tag_name','value_counts']
```

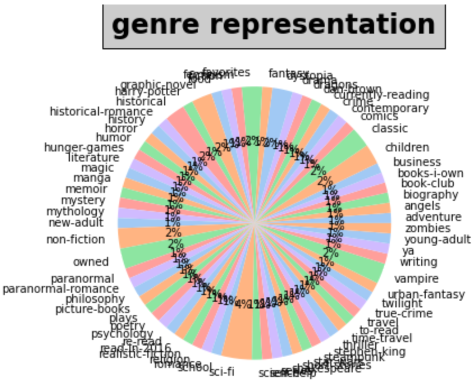
```
tag['countries'] = tag.apply(lambda x: 'other' if (x['value_counts'] < 0.01 or x['tag_name'] == '') else x['tag_name'],axis=1)
```

```
tag = tag.groupby('tag_name')['value_counts'].sum().reset_index()
tag.tag_name.value_counts()
```

```
adventure        1
angels           1
science          1
sci-fi           1
school           1
                ..
historical       1
harry-potter     1
graphic-novel    1
food             1
zombies          1
Name: tag_name, Length: 72, dtype: int64
```

```
import seaborn as sns
import matplotlib.pyplot as plt
#define Seaborn color palette to use
colors = sns.color_palette('pastel')[0:5]

#create pie chart
plt.title("genre representation",bbox={'facecolor':'0.8', 'pad':8},fontsize=24, fontdict={"weight": "bold"},y=1.25,x=0.578)
f = plt.pie(tag['value_counts'], labels = tag['tag_name'], colors = colors, autopct='%.0f%%',radius=1.4)
```



Exploring book data

```
book.shape
```

```
(52201, 28)
```

```
book.head(10)
```

0	630104	Inner Circle	https://www.goodreads.com//book/show/630104.In...	(Private #5)	https://i.gr-assets.com/images/S/compressed.ph...	Kate Brian, Julian Peploe	https://www.goodreads.co
1	9487	A Time to Embrace	https://www.goodreads.com//book/show/9487.A_Ti...	(Timeless Love #2)	https://i.gr-assets.com/images/S/compressed.ph...	Karen Kingsbury	https://www.goodreads.co
2	6050894	Take Two	https://www.goodreads.com//book/show/6050894-t...	(Above the Line #2)	https://i.gr-assets.com/images/S/compressed.ph...	Karen Kingsbury	https://www.goodreads.co

```
#droppping columns which are not used in the recommmendation
book = book.drop(['series','link','number_of_pages','date_published','publisher','original_title','isbn','isbn13','asin','settings','characters','awards','amazon_re
```

Child

```
book.head()
```

	goodreads_book_id	title	author	author_link	rating_count	review_count	average_rating	five_star_ratings	four_sta
0	630104	Inner Circle	Kate Brian, Julian Peploe	https://www.goodreads.com/author/show/94091.Ka...	7597.0	196.0	4.03	3045.0	
1	9487	A Time to Embrace	Karen Kingsbury	https://www.goodreads.com/author/show/3159984....	4179.0	177.0	4.35	2255.0	
2	6050894	Take Two	Karen Kingsbury	https://www.goodreads.com/author/show/3159984....	6288.0	218.0	4.23	3000.0	
3	39030	Reliquary	Douglas Preston, Lincoln Child	https://www.goodreads.com/author/show/12577.Do...	38382.0	1424.0	4.01	12711.0	
4	998	The Millionaire Next Door: The Surprising Secr...	Thomas J. Stanley, William D. Danko	https://www.goodreads.com/author/show/659.Thom...	72168.0	3217.0	4.04	27594.0	

Anthea

```
book.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 52201 entries, 0 to 52200
Data columns (total 12 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   goodreads_book_id   52201 non-null  object 
 1   title               52199 non-null  object 
 2   author              52199 non-null  object 
 3   author_link         52199 non-null  object 
 4   rating_count        52199 non-null  float64
 5   review_count        52199 non-null  float64
 6   average_rating      52199 non-null  float64
 7   five_star_ratings   52199 non-null  float64
 8   four_star_ratings   52199 non-null  float64
 9   three_star_ratings  52199 non-null  float64
 10  two_star_ratings    52199 non-null  float64
 11  one_star_ratings    52199 non-null  float64
dtypes: float64(8), object(4)
memory usage: 4.8+ MB
```

```
print(f'Duplicate entries: {book.duplicated().sum()}')
```

```
Duplicate entries: 0
```

```
#Book ratings count using group by
rating=book.groupby(['title'])['five_star_ratings'].count().sort_values(ascending=False).reset_index()
```

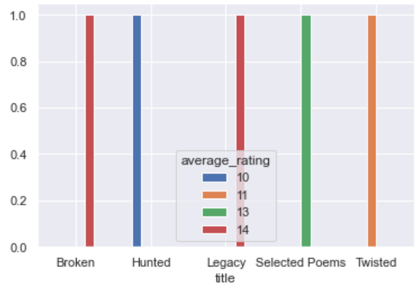
```
plt.figure(figsize=(15,8))
sns.set_theme(style="darkgrid")
ax=sns.barplot(rating['five_star_ratings'][:20],rating['title'][:20],palette='rocket')
ax.set_title('Book five star rating count', fontsize=30,fontweight='bold')
ax.set_xlabel('Rating Count',fontsize=20)
ax.set_ylabel('Book-Titles',fontsize=20)
plt.yticks(fontsize=15)
plt.xticks(fontsize=15)
```

```
(array([ 0.,  2.,  4.,  6.,  8., 10., 12., 14., 16.]),
 [Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, '')])
```


```
#Book ratings count using group by
rate=book.groupby(['title'])['average_rating'].count().sort_values(ascending=False).reset_index()
```



```
lables = pd.crosstab(rate['title'][:5],rate['average_rating'][:5])
barplot = lables.plot.bar(rot=0)
```



book tags data analysis

```
booktag.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9972 entries, 0 to 9971
Data columns (total 3 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   goodreads_book_id  9972 non-null   int64
 1   tag_id             9972 non-null   int64
 2   count              9972 non-null   int64
dtypes: int64(3)
memory usage: 233.8 KB
```

```
booktag.shape
```

```
(9972, 3)
```

```
booktag.head()
```

	goodreads_book_id	tag_id	count
0	1	30574	167697
1	1	11305	37174
2	1	11557	34173
3	1	8717	12986
4	1	33114	12716

```
print(f'Duplicate entries: {booktag.duplicated().sum()}')
```

```
Duplicate entries: 0
```

```
merged_df = pd.merge(booktag,book, left_on="goodreads_book_id", right_on="goodreads_book_id")
```

```
merged_df
```

				Blood Prince	Rowling				
1	1	11305	37174	Harry Potter and the Half-Blood Prince	J.K. Rowling	https://www.goodreads.com/author/show/1077326....	2352925.0	37813.0	4.57
2	1	11557	34173	Harry Potter and the Half-Blood Prince	J.K. Rowling	https://www.goodreads.com/author/show/1077326....	2352925.0	37813.0	4.57
3	1	8717	12986	Harry Potter and the Half-Blood Prince	J.K. Rowling	https://www.goodreads.com/author/show/1077326....	2352925.0	37813.0	4.57
4	1	33114	12716	Harry Potter and the Half-Blood Prince	J.K. Rowling	https://www.goodreads.com/author/show/1077326....	2352925.0	37813.0	4.57
...	...	...	...	...	...	...	...	...	...
7149	31538614	30574	24465	Short Stories from Hogwarts of Power, Politics...	J.K. Rowling, MinaLima	https://www.goodreads.com/author/show/1077326....	25471.0	1687.0	4.19
7150	31538635	30574	17851	Short Stories from Hogwarts of Heroism, Hardsh...	J.K. Rowling, MinaLima	https://www.goodreads.com/author/show/1077326....	36532.0	2434.0	4.23
7151	31538647	30574	23052	Hogwarts: An Incomplete and Unreliable Guide	J.K. Rowling	https://www.goodreads.com/author/show/1077326....	25216.0	1827.0	4.21
7152	32075671	30574	5858	The Hate U Give	Angie Thomas	https://www.goodreads.com/author/show/15049422...	439853.0	53750.0	4.51
7153	32075671	33114	2010	The Hate U Give	Angie Thomas	https://www.goodreads.com/author/show/15049422...	439853.0	53750.0	4.51

7154 rows × 14 columns

```
print(f'Duplicate entries: {merged_df.duplicated().sum()}')
```

```
Duplicate entries: 0
```

```
merged_df.shape
```

```
(7154, 14)
```

Minning and Analytics

- splitting the data into training and testing dataset

```
# creating a surprise object

reader = Reader(rating_scale=(0, 10))
data   = Dataset.load_from_df(merged_df[['goodreads_book_id','title','five_star_ratings']], reader)


# Split the data into training & testing sets. Python's surprise documentation has the steps detailed out
# https://surprise.readthedocs.io/en/stable/FAQ.html

raw_ratings = data.raw_ratings
import random
random.shuffle(raw_ratings)                 # shuffle dataset

threshold   = int(len(raw_ratings)*0.8)

train_raw_ratings = raw_ratings[:threshold] # 80% of data is trainset
test_raw_ratings  = raw_ratings[threshold:] # 20% of data is testset

data.raw_ratings = train_raw_ratings        # data is now the trainset
trainset         = data.build_full_trainset()
testset          = data.construct_testset(test_raw_ratings)
```

Python's surprise library has several built-in algorithms for building rating based recommendation systems

KNN (K Nearest Neighbours), memory based approach

This algorithm takes into consideration up-to 'K' nearest users (in user based collaborative filtering) or 'K' nearest items (in item based collaborative filtering) for making recommendations. By default, the algorithm is 'user-based', and k is 40 (kmin is 1). This means ratings of 40 nearest users are considered while recommending an an item to a user. Some variants of this algorithm include WithMeans, WithZscore & Baseline wherein the average rating of users, or the normalized ZScores of ratings or the baseline rating are also considered as the system generates recommendations

SVD (Singular Value Decomposition), model based approach

This algorithm takes a matrix factorization approach. The user-item rating matrix is factorized into smaller dimension user & item matrices consisting of latent factors (hidden characteristics). By default, number of latent factors is 100. These latent factors are able to capture the known user-item rating preference & in the process are able to predict an estimated rating for all user-item pair where user has not yet rated an item

```
# Trying KNN (K-Nearest Neighbors) & SVD (Singluar Value decomposition) algorithms using default model parameters

models=[KNNBasic(),KNNWithMeans(),KNNWithZScore(),KNNBaseline(),SVD()]
results = {}

for model in models:
    # perform 5 fold cross validation
    # evaluation metrics: mean absolute error & root mean square error
    CV_scores = cross_validate(model, data, measures=["MAE","RMSE"], cv=5, n_jobs=-1)

    # storing the average score across the 5 fold cross validation for each model
    result = pd.DataFrame.from_dict(CV_scores).mean(axis=0).\
             rename({'test_mae':'MAE', 'test_rmse': 'RMSE'})
    results[str(model).split("algorithms.")[1].split("object ")[0]] = result


performance_df = pd.DataFrame.from_dict(results)
print("Model Performance: \n")
performance_df.T.sort_values(by='RMSE')
```

```
Model Performance:
```

	MAE	RMSE	fit_time	test_time
knns.KNNBasic	150618.441073	414735.157878	0.299012	0.013578
matrix_factorization.SVD	150618.188926	416123.580760	0.425262	0.009971
knns.KNNBaseline	150611.989052	416365.876242	0.340963	0.015963
knns.KNNWithZScore	150611.231892	416506.416626	0.479759	0.020169
knns.KNNWithMeans	150610.695144	416729.410735	0.322311	0.010196

KNNWithMeans has the least RMSE (root mean square error) among KNN algorithms

The model fit_time is the maximum for SVD but the model test_time is the least

Machine Learning – Hyperparameter tuning with GridSearchCV KNNWithMeans

user_based: By default, this model parameter is 'True'. The other option 'False', corresponds to an item based approach

min_support: This refers to number of items to consider (in user based approach) or number of users to consider (in item based approach) to calculate similarity before setting it to 0

name: This refers to the distance measure that KNNWithMeans utilizes for calculating similarity. By default, the value is set as 'MSD' i.e., Mean Squared Distance. One other popular distance measure for rating based data is 'cosine' or angular distance. The cosine distance enables calculation of similarity among items & users accounting for inherent rating bias amongst users. E.g., users who like item2 twice as much as item1 may rate items as '8' & '4' if they are generous with their ratings but rate it only '4' & '2' if they are more stringent raters. MSD measures similarity based on the absolute ratings and will not be able to capture this inherent rating bias described above, however, cosine distance measure will be able to capture the same

```
# Hyperparameter tuning - KNNWithMeans

param_grid = { 'sim_options' : {'name': ['msd','cosine'], \
                                'min_support': [3,5], \
                                'user_based': [False, True]}
             }

gridsearchKNNWithMeans = GridSearchCV(KNNWithMeans, param_grid, measures=['mae', 'rmse'], \
                                      cv=5, n_jobs=-1)

gridsearchKNNWithMeans.fit(data)

print(f'MAE Best Parameters:  {gridsearchKNNWithMeans.best_params["mae"]}')
print(f'MAE Best Score:       {gridsearchKNNWithMeans.best_score["mae"]}\n')

print(f'RMSE Best Parameters: {gridsearchKNNWithMeans.best_params["rmse"]}')
print(f'RMSE Best Score:      {gridsearchKNNWithMeans.best_score["rmse"]}\n')
```

```
MAE Best Parameters:  {'sim_options': {'name': 'msd', 'min_support': 3, 'user_based': False}}
MAE Best Score:       150608.52091336614

RMSE Best Parameters: {'sim_options': {'name': 'msd', 'min_support': 3, 'user_based': False}}
RMSE Best Score:      416834.8948419714
```

```
# Model fit & prediction - KNNWithMeans
```

```
sim_options = {'name':'cosine','min_support':3,'user_based':False}
final_model = KNNWithMeans(sim_options=sim_options)

# Fitting the model on trainset & predicting on testset, printing test accuracy
pred = final_model.fit(trainset).test(testset)

print(f'\nUnbiased Testing Performance:')
print(f'MAE: {accuracy.mae(pred)}, RMSE: {accuracy.rmse(pred)}')
```

```
Computing the cosine similarity matrix...
Done computing similarity matrix.

Unbiased Testing Performance:
MAE:  151242.9462
RMSE: 435644.4398
MAE: 151242.9461914745, RMSE: 435644.4397895185
```

- Post Hyperparameter Tuning with GridSearchCV, the best parameters are found to be different for MAE & RMSE metrics.
- 'Cosine' distance measure, min_support of 3 & user_based : False i.e., item based approach have been chosen for building recommendations.
- The logic/code below can be modified to make recommendations using 'MSD' distance & user based method if needed.
- The MAE & RMSE metrics for testset are comparable with what was obtained using cross validation & hyperparameter tuning stages with trainset. Chosen model hence, generalizes well

SVD

n_factors: This refers to number of latent factors (hidden characteristics) for matrix factorization/ dimensionality reduction. By default, the value is 100

n_epochs: This refers to number of iterations of stochiastic gradient descent procedure, utilized by SVD for learning the parameters and minimizing error

lr_all & reg_all: i.e., learning rate and regularization rate. Learning rate is the step size of the said (above) SGD algorithm whereas regularization rate prevents overlearning, so the model may generalize well on data it has not yet seen.

By default these values are set as 0.005 & 0.02

```
# Hyperparameter tuning - SVD

param_grid = {"n_factors": range(10,100,20),
              "n_epochs" : [5, 10, 20],
              "lr_all"   : [0.002, 0.005],
              "reg_all"  : [0.2, 0.5]}

gridsearchSVD = GridSearchCV(SVD, param_grid, measures=['mae', 'rmse'], cv=5, n_jobs=-1)

gridsearchSVD.fit(data)

print(f'MAE Best Parameters:  {gridsearchSVD.best_params["mae"]}')
print(f'MAE Best Score:       {gridsearchSVD.best_score["mae"]}\n')

print(f'RMSE Best Parameters: {gridsearchSVD.best_params["rmse"]}')
print(f'RMSE Best Score:      {gridsearchSVD.best_score["rmse"]}\n')
```

```
MAE Best Parameters:  {'n_factors': 10, 'n_epochs': 10, 'lr_all': 0.002, 'reg_all': 0.5}
MAE Best Score:       150614.19489357804

RMSE Best Parameters: {'n_factors': 10, 'n_epochs': 10, 'lr_all': 0.002, 'reg_all': 0.5}
RMSE Best Score:      417409.7976127246
```

Post Hyperparameter Tuning with GridSearchCV, the best parameters are found to be different for MAE & RMSE metrics 'n_factors':50, 'n_epochs':10, 'lr_all':0.005 & 'reg_all': 0.2 have been chosen for building recommendations

```
# Model fit & prediction - SVD

final_model = SVD(n_factors=50, n_epochs=10, lr_all=0.005, reg_all= 0.2)

# Fitting the model on trainset & predicting on testset, printing test accuracy
pred = final_model.fit(trainset).test(testset)

print(f'\nUnbiased Testing Performance')
print(f'MAE: {accuracy.mae(pred)}, RMSE: {accuracy.rmse(pred)}')
```

```
Unbiased Testing Performance
MAE:  151242.9462
RMSE: 435644.4398
MAE: 151242.9461914745, RMSE: 435644.4397895185
```

The MAE & RMSE metrics for testset are comparable with what was obtained using cross validation & hyperparameter tuning stages with trainset. Chosen model hence again, generalizes well

Evaluation

```
book[book['title']=='Stardust']
```

	goodreads_book_id	title	author	author_link	rating_count	review_count	average_rating	five_star_ratings	four_st
41860	16793	Stardust	Neil Gaiman	https://www.goodreads.com/author/show/1221698....	346051.0	17981.0	4.09	130605.0	

```
# Entire dataset will be used for building recommendations

reader = Reader(rating_scale=(0, 10))
data = Dataset.load_from_df(merged_df[['goodreads_book_id','title','five_star_ratings']], reader)
trainset = data.build_full_trainset()

# A list of useful trainset methods are explained here:
# https://surprise.readthedocs.io/en/stable/trainset.html
```

Two different functions are written to generate recommendations with the final chosen models KNNWithMeans & SVD

```
def generate_recommendationsKNN(userID=16793, like_recommend=5, get_recommend =10):

    ''' This function generates "get_recommend" number of book recommendations using
        KNNWithMeans & item based filtering. The function needs as input three
        different parameters:
        (1) userID i.e., userID for which recommendations need to be generated
        (2) like_recommend i.e., number of top recommendations for the userID to be
        considered for making recommendations
        (3) get_recommend i.e., number of recommendations to generate for the userID
        Default values are: userID=13552, like_recommend=5, get_recommend=10
    '''

    # Compute item based similarity matrix
    sim_options       = {'name':'cosine','min_support':3,'user_based':False}
    similarity_matrix = KNNWithMeans(sim_options=sim_options).fit(trainset).\
                        compute_similarities()

    userID      = trainset.to_inner_uid(userID)    # converts the raw userID to innerID
    userRatings = trainset.ur[userID]              # method .ur takes user innerID &
                                                   # returns back user ratings


    # userRatings is a list of tuples [(,),(,),(,)..]. Each tuple contains item & rating
    # given by the user for that item. Next, the tuples will be sorted within the list
    # in decreasing order of rating. Then top 'like_recommend' items & ratings are extracted

    temp_df = pd.DataFrame(userRatings).sort_values(by=1, ascending=False).\
              head(like_recommend)
    userRatings = temp_df.to_records(index=False)

    # for each (item,rating) in top like_recommend user items, multiply the user rating for
    # the item with the similarity score (later is obtained from item similarity_matrix) for
    # all items. This helps calculate the weighted rating for all items. The weighted ratings
    # are added & divided by sum of weights to estimate rating the user would give an item

    recommendations   = {}

    for user_top_item, user_top_item_rating  in userRatings:

        all_item_indices          =   list(pd.DataFrame(similarity_matrix)[user_top_item].index)
        all_item_weighted_rating  =   list(pd.DataFrame(similarity_matrix)[user_top_item].values*\
                                          user_top_item_rating)

        all_item_weights          =   list(pd.DataFrame(similarity_matrix)[user_top_item].values)


        # All items & final estimated ratings are added to a dictionary called recommendations

        for index in range(len(all_item_indices)):
            if index in recommendations:
                # sum of weighted ratings
                recommendations[index] += all_item_weighted_rating[index]
            else:
                recommendations[index]  = all_item_weighted_rating[index]


    for index in range(len(all_item_indices)):
            if all_item_weights[index]  !=0:
                # final ratings (sum of weighted ratings/sum of weights)
                recommendations[index]   =recommendations[index]/\
                                          (all_item_weights[index]*like_recommend)


    # convert dictionary recommendations to a be a list of tuples [(,),(,),(,)]
    # with each tuple being an item & estimated rating user would give that item
    # sort the tuples within the list to be in decreasing order of estimated ratings

    temp_df = pd.Series(recommendations).reset_index().sort_values(by=0, ascending=False)
    recommendations = list(temp_df.to_records(index=False))

    # return get_recommend number of recommedations (only return items the user
    # has not previously rated)

    final_recommendations = []
    count = 0

    for item, score in recommendations:
```

```
        flag = True
        for userItem, userRating in trainset.ur[userID]:
            if item == userItem:
                flag = False       # If item in recommendations has not been rated by user,
                break              # add to final_recommendations
        if flag == True:
            final_recommendations.append(trainset.to_raw_iid(item))
            count +=1              # trainset has the items stored as inner id,
                                   # convert to raw id & append

        if count > get_recommend:  # Only get 'get_recommend' number of recommendations
            break
    return(final_recommendations)
```

```
recommendationsKNN = generate_recommendationsKNN(userID=16793, like_recommend=5, get_recommend=10)
recommendationsKNN
```

```
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
['Harry Potter and the Half-Blood Prince',
 'Bone Crossed',
 'Hostage to Pleasure',
 "Lament: The Faerie Queen's Deception",
 'Night World, No. 1',
 'The Initiation / The Captive Part I',
 'Suicide Notes',
 'The Hour I First Believed',
 'The Lucky One',
 'Wicked',
 'Killer']
```

```
# SVD

def generate_recommendationsSVD(userID=16793, get_recommend =10):

    ''' This function generates "get_recommend" number of book recommendations
        using Singular value decomposition. The function needs as input two
        different parameters:
        (1) userID i.e., userID for which recommendations need to be generated
        (2) get_recommend i.e., number of recommendations to generate for the userID
        Default values are: userID=13552, get_recommend=10
    '''

    model = SVD(n_factors=50, n_epochs=10, lr_all=0.005, reg_all= 0.2)
    model.fit(trainset)

    # predict rating for all pairs of users & items that are not in the trainset

    testset = trainset.build_anti_testset()
    predictions = model.test(testset)
    predictions_df = pd.DataFrame(predictions)

    # get the top get_recommend predictions for userID

    predictions_userID = predictions_df[predictions_df['uid'] == userID].\
                         sort_values(by="est", ascending = False).head(get_recommend)

    recommendations = []
    recommendations.append(list(predictions_userID['iid']))
    recommendations = recommendations[0]

    return(recommendations)
```

```
recommendationsSVD = generate_recommendationsSVD(userID=16793, get_recommend =10)
recommendationsSVD
```

```
['Harry Potter and the Half-Blood Prince',
 'The Lucky One',
 'Hostage to Pleasure',
 "Lament: The Faerie Queen's Deception",
 'Night World, No. 1',
 'The Initiation / The Captive Part I',
 'Suicide Notes',
 'The Hour I First Believed',
 'Wicked',
 'Bone Crossed']
```

Result

- While the list of recommendations generated using KNNWithMeans & SVD are different (expected as they are different algorithms), there are some similarities in the generated lists too.
- Both algorithms recommended instances of Stardust novels for user 16793. Additionally, the recommended books seem to be a similar genre lending confidence in interpretability of recommendations.
- We have successfully implemented a memory based as well as method based collaborative filtering approach to make recommendations in this project
- In instances with a new user or new item where little is known of the rating preference, collaborative filtering may not be the method of choice for generating recommendations.
- Content based filtering methods may be more appropriate. Often, a hybrid approach is taken for building real time recommendations using multiple different approaches in industry! The project can be extended to build hybrid recommendation systems in the future.

Reference

Surprise python package http://surpriselib.com

Anaconda Cloud. Wordcloud. Retrieved (2022, January 27) from https://anaconda.org/conda-forge/wordcloud

Python pool. (2021). Retrieved (2022, February 17) from https://www.pythonpool.com/matplotlib-figsize

Justify the text in jupyter.Retrieved (2022, February 17) from https://stackoverflow.com/questions/35077507/how-to-right-align-and-justify-align-in-markdown

NumPy.Retrieved (2022, February 17) from https://numpy.org/doc/stable/index.html

Pandas Package. Retrieved (2022, February 17) from https://pandas.pydata.org/

Matplotlib.Retrieved (2022, February 17) from https://matplotlib.org/

Using pandas crosstab to create a bar plot.Retrieved(2022, February 28) from https://www.geeksforgeeks.org/using-pandas-crosstab-to-create-a-bar-plot/#

KNN https://towardsdatascience.com/knn-k-nearest-neighbors-1-a4707b24bd1d#:~:text=KNN%20(K%20%E2%80%94%20Nearest%20Neighbors),(a%20vector)%20from%20other%20.