Group #233: Anime Movie Recommender System

First Name	Last Name	Email
Charmi	Kalani	ckalani@hawk.iit.edu
Anwesha	Kakoty	akakoty@hawk.iit.edu

Table of Contents

1. Introduction

Recommendation system are used to predict the user ratings or preference a customer or user would give to a certain item. Recommender systems are used in a variety of areas including movies, music, news, books, research articles, search queries, social tags, and products in general. It helps the user view or purchase similar products according to his interests and by doing so it increases the revenue of the website.

We have developed our project on an Anime Recommendation System which suggests the user similar anime according to previously watched and rated anime, which match their interests.

We have used three different algorithms to find out the top-N recommendations and computed its errors MAE and RMSE and Precision Recall for the models. We have created SVD, Item-Based KNN Baseline and Item-Based KNN models for Top-10 recommendation.

We have used Surprise, Pandas, Numpy, Matplotlib libraries for the entire model building and evaluation process.

2. Data

This data set is extracted from myanimelist.net and contains information on user preference data. Each user can add anime to their completed list and give it a rating and this data set is a compilation of those ratings. There are two tables present in the dataset. This dataset was found on Kaggle.

Total number of ratings in the dataset are: 7813737.
Total number of users in the dataset are: 73515.
Total number of anime in the dataset are: 11200.

Anime.csv

- anime_id - myanimelist.net's unique id identifying an anime.
- name - full name of anime.
- genre - comma separated list of genres for this anime.
- type - movie, TV, OVA, etc.
- episodes - how many episodes in this show. (1 if movie).
- rating - average rating out of 10 for this anime.
- members - number of community members that are in this anime's "group".

Rating.csv

- user_id – non-identifiable randomly generated user id.
- anime_id - the anime that this user has rated.
  rating - rating out of 10 this user has assigned (-1 if the user watched it but didn't assign a rating).

```
In [151]: import pandas as pd
          import numpy as np
          import math
          import re
          from scipy.sparse import csr_matrix
          import matplotlib.pyplot as plt
          import seaborn as sns
          import surprise
          from surprise import Reader, Dataset, SVD, evaluate
          sns.set_style("ticks")
```

```
In [167]: # checking for any null values
          print("No of Nan values in our dataframe : ", sum(ratings.isnull().any()))

          No of Nan values in our dataframe :  0
```

```
In [170]: print("Total data ")
          print("-"*50)
          print("\nTotal no of ratings :",ratings.shape[0])
          print("Total No of Users   :", len(np.unique(ratings.user_id)))
          print("Total No of Anime  :", len(np.unique(ratings.anime_id)))

          Total data
          --------------------------------------------------

          Total no of ratings : 7813737
          Total No of Users   : 73515
          Total No of Anime  : 11200
```

3. Problems to be Solved

Our main task is to build a recommendation system based on Anime.

- We must predict the ratings a user would give to the anime he has not yet watched or rated.
- We must minimize the difference between the predicted rating and actual rating (RMSE and MAE) for better accuracy of the recommender system.
- We must provide a list of top-N number of anime, given a user, which the user would most likely watch and like.
- We have to calculate the Precision and Recall for the Top-10 recommendations.

4. KDD

4.1. Data Processing

We have a dataset as large as approximately 7 million rows of data. The models took a long processing time for such huge dataset and thus we preprocessed the data. The data went through slicing, condition being all those users that have given ratings to 634 or lesser movies are removed and all those movies which have 164 or lesser ratings given to itself are removed.

Also, we have divided the ratings by 2, because Surprise Library only supported 1-5 ratings.

In [153]:
```
ratings['rating'] = (ratings['rating'])/2
ratings.head()
```

Out[153]:

	user_id	anime_id	rating
0	1	20	-0.5
1	1	24	-0.5
2	1	79	-0.5
3	1	226	-0.5
4	1	241	-0.5

In [173]:
```
ratings['rating'].describe()
```

Out[173]:
```
count     7.813737e+06
mean      3.072015e+00
std       1.863900e+00
min      -5.000000e-01
25%       3.000000e+00
50%       3.500000e+00
75%       4.500000e+00
max       5.000000e+00
Name: rating, dtype: float64
```

In [154]:
```
f = ['count','mean']

anime_summary = ratings.groupby('anime_id')['rating'].agg(f)
anime_summary.index = anime_summary.index.map(int)
anime_min = round(anime_summary['count'].quantile(0.8),0)
drop_anime_list = anime_summary[anime_summary['count'] < anime_min].index

print('Minimum times of reviews for anime: {}'.format(anime_min))

df_cust_summary = ratings.groupby('user_id')['rating'].agg(f)
df_cust_summary.index = df_cust_summary.index.map(int)
cust_min = round(df_cust_summary['count'].quantile(0.8),0)
drop_cust_list = df_cust_summary[df_cust_summary['count'] < cust_min].index


print('Minimum times of reviews for a customer: {}'.format(cust_min))
```

```
Minimum times of reviews for anime: 634.0
Minimum times of reviews for a customer: 164.0
```

In [155]:
```
print('Original Shape: {}'.format(ratings.shape))
df = ratings[~ratings['anime_id'].isin(drop_anime_list)]
df = ratings[~ratings['user_id'].isin(drop_cust_list)]
print('After Trim Shape: {}'.format(df.shape))
print('-Data Examples-')
print(df.head())
```

```
Original Shape: (7813737, 3)
After Trim Shape: (4821593, 3)
-Data Examples-
     user_id  anime_id  rating
302        5         6     4.0
303        5        15     3.0
304        5        17     3.0
305        5        18     3.0
306        5        20     3.0
```

4.2. Data Mining Methods and Processes

We have implemented three kinds of algorithms for Top-N recommendation. The algorithms used are:

- SVD Algorithm.

  SVD decreases the dimension of the utility matrix by extracting its latent factors. Essentially, we want to turn the recommendation problem into an optimization problem. We can view it as how good we are in predicting the rating for items given a user.

  We have initially built the model and calculated it's RMSE and MAE values.

```
In [174]: from surprise import SVD
          from surprise import Dataset
          from surprise import accuracy
          from surprise.model_selection import train_test_split


          # sample random trainset and testset
          # test set is made of 25% of the ratings.
          trainset, testset = train_test_split(data, test_size=.25)

          # We'll use the famous SVD algorithm.

          # Train the algorithm on the trainset, and predict ratings for the tes
          svd.fit(trainset)
          predictions = svd.test(testset)

In [176]: print(accuracy.rmse(predictions))
          print(accuracy.mae(predictions))

          RMSE: 1.0878
          1.0877919712104502
          MAE:  0.7794
          0.7794174683652741
```

  We then found all those movies that the user has already watched and has rated them more than 4.

```
In [162]: n = 226
          df_n = df[(df['user_id'] == n) & (df['rating'] >4)]
          df_n = df_n.set_index('anime_id')
          nm = anime_data['name']
          df_n = df_n.join(nm)
          print(df_n)

                   user_id  rating                                   name
          anime_id
          180          226     4.5                               Vandread
          181          226     4.5              Vandread: The Second Stage
          245          226     5.0                   Great Teacher Onizuka
          269          226     5.0                                 Bleach
          419          226     4.5                    Samurai Deeper Kyou
          857          226     4.5                               Air Gear
          889          226     4.5                           Black Lagoon
          1067         226     4.5             Kishin Houkou Demonbane (TV)
          1127         226     4.5                   UFO Princess Valkyrie
          1195         226     4.5                        Zero no Tsukaima
          1250         226     5.0                        Erementar Gerad
          1482         226     4.5                             D.Gray-man
          1519         226     4.5          Black Lagoon: The Second Barrage
          1575         226     5.0               Code Geass: Hangyaku no Lelouch
          1604         226     4.5                   Katekyo Hitman Reborn!
          1726         226     4.5                            Devil May Cry
          1840         226     4.5         Zero no Tsukaima: Futatsuki no Kishi
          2002         226     5.0                             Heroic Age
          2904         226     5.0            Code Geass: Hangyaku no Lelouch R2
          2993         226     4.5                      Rosario to Vampire
          3299         226     5.0                  H2O: Footprints in the Sand
          3455         226     4.5                             To LOVE-Ru
          3503         226     4.5                                Kanokon
          3594         226     4.5                          Tears to Tiara
          3712         226     4.5       Zero no Tsukaima: Princesses no Rondo
          4186         226     5.0                   Chrome Shelled Regios
          4214         226     4.5                 Rosario to Vampire Capu2
          4901         226     4.5   Black Lagoon: Roberta&#039;s Blood Trail
          5041         226     5.0                              Guin Saga
          5042         226     4.5                             Kiss x Sis
```

We predicted the estimated ratings that a user would give to the unseen movies.

```
reader = Reader()
user_n = anime_data.copy()
user_n = user_n.reset_index()
#user_n = user_n[~user_n['anime_id'].isin(drop_anime_list)]
svd = SVD()
# getting full dataset
data = Dataset.load_from_df(df[['user_id', 'anime_id', 'rating']], reader)

trainset = data.build_full_trainset()
model = svd.train(trainset)
```

```
user_n['Estimate_Score'] = df['anime_id'].apply(lambda x: svd.predict(n, x).est)

#user_n = user_n.drop('anime_id', axis = 1)

user_n = user_n.sort_values('Estimate_Score', ascending=False)
print(user_n.head(10))
```

```
       anime_id                                               name  \
2164       2847  Pokemon Diamond &amp; Pearl: Dialga vs. Palkia...
3617       1760                                           Golgo 13
4214      14093                       Pokemon Best Wishes! Season 2
4688      31144                                          Mottainai
11185      9311  Kateikyoushi no Oneesan 2 The Animation: H no ...
1103      32365  Boruto: Naruto the Movie - Naruto ga Hokage ni...
9418      16131            Machine Robo: Butchigiri Battle Hackers
3808      31980                       Okusama ga Seitokaichou! OVA
4208      32962                                       Occultic;Nine
749       22125           Kuroko no Basket: Mou Ikkai Yarimasen ka

                                                  genre     type episodes  \
2164      Action, Adventure, Comedy, Drama, Fantasy, Kids    Movie        1
3617           Action, Adventure, Drama, Military, Seinen    Movie        1
4214             Action, Adventure, Comedy, Fantasy, Kids       TV       24
4688                                         Slice of Life  Special        1
11185                                               Hentai      OVA        2
1103    Action, Comedy, Martial Arts, Shounen, Super P...  Special        1
9418                                 Action, Mecha, Sci-Fi       TV       31
3808                     Comedy, Ecchi, Romance, Shounen      OVA        1
4208                                        Mystery, Sci-Fi       TV       12
749                      Comedy, School, Shounen, Sports  Special        1

       rating  members  Estimate_Score
2164     7.34    38238        4.623925
3617     6.93     6344        4.623925
4214     6.78    19603        4.623925
4688     6.66     2234        4.623925
11185    7.15     5091        4.623925
1103     7.68    16868        4.623925
9418     6.22      239        4.623925
3808     6.88    13602        4.623925
4208     6.78    82532        4.587527
749      7.86    20397        4.534112
```

Then, we listed out all the top-10 movies for all users.

```
In [24]: from collections import defaultdict

         from surprise import SVD
         from surprise import Dataset

         def get_top_n(predictions, n=10):

             # First map the predictions to each user.
             top_n = defaultdict(list)
             for uid, iid, true_r, est, _ in predictions:
                 top_n[uid].append((iid, est))

             # Then sort the predictions for each user and retrieve the k highest ones.
             for uid, user_ratings in top_n.items():
                 user_ratings.sort(key=lambda x: x[1], reverse=True)
                 top_n[uid] = user_ratings[:n]

             return top_n

         top_n = get_top_n(predictions, n=10)

         # Print the recommended items for each user
         for uid, user_ratings in top_n.items():
             print(uid, [iid for (iid, _) in user_ratings])
```

```
58378 [401, 395, 43, 538, 834, 543, 883, 468, 1303, 868]
29950 [6, 2985, 245, 16009, 10087, 232, 267, 477, 11759, 5177]
5343 [513, 448, 3342, 849, 1535, 2236, 522, 45, 149, 5902]
28298 [457, 14719, 15417, 28977, 572, 32182, 513, 2001, 7711, 28735]
1797 [28025, 14719, 29803, 3002, 6213, 16706, 20709, 4224, 9253, 11597]
58024 [4181, 11577, 2236, 10087, 30276, 199, 22297, 2759, 6336, 16498]
49652 [10030, 5781, 372, 123, 1210, 6347, 59, 30276, 20, 356]
29797 [10379, 11771, 15335, 7311, 7472, 32182, 8129, 22535, 22145, 5681]
6152 [5356, 10087, 21939, 4282, 10716, 10798, 16067, 17895, 9989, 658]
72115 [4901, 25835, 32, 9617, 2001, 18617, 17074, 9756, 18679, 28025]
46333 [1535, 10408, 10049, 11741, 853, 11843, 512, 5300, 2236, 2890]
```

- Item-Based KNN Algorithm.

We initially, computed similarity matrix using **Pearson Correlation** and got found the RMSE and MAE errors.

```
In [177]: import io

          from surprise import KNNBasic
          from surprise import Dataset
          from surprise import get_dataset_dir


          sim_options = {'name': 'pearson', 'user_based': False}
          knn = KNNBasic(sim_options=sim_options)
          knn.fit(trainset)
          predictions = knn.test(testset)
```

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

```
In [178]: print(accuracy.rmse(predictions))
          print(accuracy.mae(predictions))
```

```
RMSE: 1.1845
1.1844714497331137
MAE:  0.8833
0.8832844551653122
```

Next, we built a model and listed the Top-10 recommendations like “Kimi no Na Wa”:

```
In [185]: from surprise import Dataset
          from surprise import get_dataset_dir

          neighbors = knn.get_neighbors(0, k=10)
          print()
          print('The 10 nearest neighbors of Kimi no Na Wa are:')


          for i in neighbors:
              for j in anime_data['anime_id']:
                  if j== i:
                      print(anime_data['name'][j])
```

```
The 10 nearest neighbors of Kimi no Na Wa are:
Code Geass: Hangyaku no Lelouch
Shokugeki no Souma
Ookami to Koushinryou II
Major S3
Natsume Yuujinchou: Itsuka Yuki no Hi ni
NHK ni Youkoso!
Shelter
InuYasha: Kanketsu-hen
One Piece Film: Gold
```

- Item-Based KNN(Baseline) Algorithm

  This algorithm uses Stochastic Gradient Descent to estimate the biases for optimization. Then it uses Pearson baseline correlation to compute the similarity matrix.

  First, we created the model and computed it’s RMSE and MAE values.

```
In [187]: import io

          from surprise import KNNBaseline
          from surprise import Dataset
          from surprise import get_dataset_dir


          sim_options = {'name': 'pearson_baseline', 'user_based': False}
          bsl_options = {'method': 'sgd',
                         'learning_rate': .005,
                         }
          knn_b = KNNBaseline(bsl_options=bsl_options, sim_options=sim_options)
          knn_b.fit(trainset)
          predictions_b = knn_b.test(testset)
```

```
Estimating biases using sgd...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
```

```
In [188]: print(accuracy.rmse(predictions_b))
          print(accuracy.mae(predictions_b))
```

```
RMSE: 1.0803
1.0802914817857145
MAE:  0.7690
0.7690469968304519
```

The top-10 recommendations predicted are:

```
In [190]: neighbors_b = knn_b.get_neighbors(0, k=10)
          print()
          print('The 10 nearest neighbors of Kimi no Na Wa are:')


          for i in neighbors_b:
              for j in anime_data['anime_id']:
                  if j== i:
                      print(anime_data['name'][j])
```

```
The 10 nearest neighbors of Kimi no Na Wa are:
JoJo no Kimyou na Bouken: Stardust Crusaders
Aoki Hagane no Arpeggio: Ars Nova DC
Kara no Kyoukai 5: Mujun Rasen
Bleach Movie 1: Memories of Nobody
Detective Conan OVA 01: Conan vs. Kid vs. Yaiba
Durarara!!x2 Shou: Watashi no Kokoro wa Nabe Moyou
Sword Art Online: Extra Edition
Huyao Xiao Hongniang: Yue Hong
Slam Dunk
```

And finally we found Precision and Recall for each model.

5. Evaluations and Results

5.1. Evaluation Methods

We have used Hold-Out Evaluation Method for Predicting the Top-10 recommendation results for all the three algorithms. And we have evaluated Precision and Recall for all three models.

Precision being all those movies that are relevant which are recommended.

Recall being all those movies that are recommended from all the relevant movies.

- SVD:
  Precision and Recall: We have taken a threshold as the relevancy factor. Initially we have set the default parameters when defining the function, but we can change the parameters according to our convenience while calling the function.

```
In [32]: from collections import defaultdict
         from surprise import Dataset
         from surprise import SVD
         from surprise.model_selection import KFold

         kf = KFold(n_splits=5)

         def precision_recall_at_k(predictions, k=10, threshold=3.5):
             #Return precision and recall at k metrics for each user.
             # First map the predictions to each user.
             user_est_true = defaultdict(list)
             for uid, _, true_r, est, _ in predictions:
                 user_est_true[uid].append((est, true_r))

             precisions = dict()
             recalls = dict()
             for uid, user_ratings in user_est_true.items():

                 # Sort user ratings by estimated value
                 user_ratings.sort(key=lambda x: x[0], reverse=True)

                 # Number of relevant items
                 n_rel = sum((true_r >= threshold) for (_, true_r) in user_ratings)

                 # Number of recommended items in top k
                 n_rec_k = sum((est >= threshold) for (est, _) in user_ratings[:k])

                 # Number of relevant and recommended items in top k
                 n_rel_and_rec_k = sum(((true_r >= threshold) and (est >= threshold))
                                       for (est, true_r) in user_ratings[:k])

                 # Precision@K: Proportion of recommended items that are relevant
                 precisions[uid] = n_rel_and_rec_k / n_rec_k if n_rec_k != 0 else 1

                 # Recall@K: Proportion of relevant items that are recommended
                 recalls[uid] = n_rel_and_rec_k / n_rel if n_rel != 0 else 1
             return precisions, recalls
```

```
for trainset, testset in kf.split(data):
    precisions, recalls = precision_recall_at_k(predictions, k=5, threshold=4)

    # Precision and recall can then be averaged over all users
print("Precision:", sum(prec for prec in precisions.values()) / len(precisions))
print("Recall:", sum(rec for rec in recalls.values()) / len(recalls))

Precision: 0.8991305230394231
Recall: 0.16636605881617858
```

RMSE and MAE:

```
In [176]: print(accuracy.rmse(predictions))
          print(accuracy.mae(predictions))

RMSE: 1.0878
1.0877919712104502
MAE:  0.7794
0.7794174683652741
```

- Item-Based KNN:

Precision and Recall

```
In [150]: for trainset, testset in kf.split(data):
              precisions, recalls = precision_recall_at_k(predictions, k=5, threshold=4)

              # Precision and recall can then be averaged over all users
          print("Precision:", sum(prec for prec in precisions.values()) / len(precisions))
          print("Recall:", sum(rec for rec in recalls.values()) / len(recalls))

Precision: 0.9556386313913712
Recall: 0.115793686729687 04
```

RMSE and MAE:

```
In [178]: print(accuracy.rmse(predictions))
          print(accuracy.mae(predictions))

RMSE: 1.1845
1.1844714497331137
MAE:  0.8833
0.8832844551653122
```

- Item-Based KNN Baseline.

Precision and Recall

```
In [195]: for trainset, testset in kf.split(data):
              precisions, recalls = precision_recall_at_k(predictions_b, k=5, threshold=4)

              # Precision and recall can then be averaged over all users
          print("Precision:", sum(prec for prec in precisions.values()) / len(precisions))
          print("Recall:", sum(rec for rec in recalls.values()) / len(recalls))

Precision: 0.932154859372506
Recall: 0.1539874206096337
```

RMSE and MAE

```
In [188]: print(accuracy.rmse(predictions_b))
          print(accuracy.mae(predictions_b))

RMSE: 1.0803
1.0802914817857145
MAE:  0.7690
0.7690469968304519
```

5.2. Results and Findings

- After creating three different models, we compared all of them.
- Our project being an Offline Analytics System, we consider RMSE as the key factor and compare models according to the RMSE.
- According to Christopher Aberger's paper on "An Analysis on Collaborative Filtering", SGD stands to be the most promising algorithm for Recommender System amongst all the Matrix Factorization Algorithms.
- We found that KNN Baseline has the least RMSE and MAE, while Item-Based KNN had the maximum Precision.
- Considering RMSE values, KNN Baseline turns out to be the best model.

6. Conclusions and Future Work

6.1. Conclusions

The project being an Offline Analytics Project yet, we consider RMSE as a value of comparison. Thus, KNN Baseline having the least RMSE and MAE values turns out to be the best model amongst all three models.

6.2. Limitations

The dataset being very huge, we did not have enough memory space and processing power for computing the entire dataset.

6.3. Potential Improvements or Future Work

- Use more collaborative filtering techniques like SVD++.
- Develop a Content-Based recommendation system.
- Implement User-Based KNN.