ANALYSIS OF MOVIE RECOMMENDATION NETWORK

Jet Osorio – 1359976
Angela Le – 14180121
Tegh Singh – 13213929
Tejashwini Gadale - 13843661

1. TABLE OF CONTENTS

1. TABLE OF CONTENTS 1

2. INTRODUCTION 2

3. PRIOR WORK 3

4. DATA COLLECTION AND PREPROCESSING 4

5. NETWORK ANALYSIS AND VISUALIZATION 4

6. RECOMMENDATION SYSTEMS 6

6.1. Non-Personalised Recommendation Systems (Statistical Techniques) 6

6.2. Personalised Recommendation Systems 6

6.2.1. Intuition for Collaborative Filtering (Pearson Correlation) 7

6.2.2. Derivation of the Formula for Pearson Correlation 7

7. rmse RESULTS 14

8. rmse EVALUATION 14

8.1. Limitations of various Recommendation Systems Approaches 14

8.2. Issues with Collaborative Filtering 14

9. CONCLUSION 15

10. REFERENCES 16

2. INTRODUCTION

Recommendation systems were initially designed to deal with the problem of information overload. A recommendation system is a basic algorithm that filters user-related data from huge data to allow users to relate and suggest goods. Recommender systems are software tools that provide product or item recommendations based on a study of information about products that many consumers are interested in, as well as the customers and their previous purchase behaviour (Jeong et al., 2013). In the recommendation system, collaborative filtering is a prominent strategy.

Finding valuable information on the internet has become a huge difficulty due to the fast growth of material on the World Wide Web. A movie recommendation system assists users in making judgments in difficult data domains with a huge volume of data. In the recommender system, a variety of strategies have been presented. Recommender systems have grown in popularity in recent years, and they are now used in several settings, including movie recommendations. A movie recommendation system predicts a user's "rating" or "preference" for a particular item.

The MovieLens dataset, which contains information on movies and users as well as the rating a user provides to a movie, will be used in this report to create a basic recommendation system that can predict which movies a user would like and forecast the rating a user will give to a movie.

3. PRIOR WORK

The recommender system has grown in popularity in recent years. It has been employed in a variety of contexts, including books, news, movies, music, and consumer goods. Restaurants retail, financial services, YouTube, Netflix and Twitter profiles all have recommender systems. These recommendation systems anticipate ratings and user satisfaction using different forms of filtering, allowing users to buy/consume things based on their interests or requirements. Information regarding a person's life might provide insight into how the user would respond in certain scenarios because it can help users discover something they might not find otherwise (Souri et al., 2018). The recommender system is a viable alternative to search algorithms.

Content-based filtering places emphasis on the item's features as well as the user's preferences (Aggarwal 2016). This filtering functions with data provided by the user. A user profile is constructed based on the information provided by the user, which is then utilised to provide guidance to users. The filtering engine becomes more accurate as the user offers more input and accepts recommendations. These algorithms attempt to suggest movies that are comparable to ones that the user has previously enjoyed. Many of the movies that were nominated are compared to those that were previously ranked by the user. This strategy lays the groundwork for data retrieval and filtering studies.

Collaborative filtering collects settings or "taste" data from several users to automatically predict or filter information about user preferences. The collaborative filtering technique is based on the assumption that if users "U1" and "U2" have similar tastes, then "U1" will tend to have "U2's" taste on topics that are different from those of random users (Terveen and Hill, 2001). By assessing user similarities and anticipating user ratings of an item based on comparable user ratings on the same list, collaborative recommender systems receive a list of recommended items (Zou et al., 2018). Kumar et al. proposed a movie recommendation system based on a collaborative filtering approach. Collaborative filtering utilises the information provided by users, this data is then analysed, and the user is recommended a movie, that is arranged in order of highest rating first. The system also enables the user to select the criteria they want the movie to be recommended based on (Kumar et al., 2015).

A hybrid recommender system is used to make content-based and collaborative-based filtering predictions independently or in combination. Hybrid approaches can be employed in a variety of contexts, and it can be utilised to improve content-based capabilities. It is also capable of combining all of them into a single format. De Campos et al. have investigated at two types of traditional recommender systems: content-based filtering and collaborative filtering. He proposed a novel approach that combines Bayesian networks with collaborative filtering, as each has its own disadvantages. Their proposed system is tailored to the given problem and includes probability distributions for making conclusions (De Campos et al., 2010).

4. DATA COLLECTION AND PREPROCESSING

The paper utilizes GroupLens' collection of movies 5-star rating data obtained from the MovieLens website, which were accumulated over the various periods of time (MovieLens, 2013). There are several datasets in relation to the MovieLens, but for the purposes of this paper, the latest MovieLens 100K dataset will be employed. The dataset being used will have 100,000 ratings alongside 3,600 tag applications applied to around 9,000 movies by approximately 600 users between 1996 to 2018 (Harper & Konstan, 2015). Users were selected at random, where each user would have rated at least 20 movies, with their ids being anonymized and no other user information provided.

Slight pre-processing was done to the dataset to ensure optimal model representation and movie recommendation results. For the model, the ratings data was adjusted to be usable as an edge list for the social and information network visualization tool, Gephi, where movie ID is listed as the Source, the user ID is listed as the Target, and rating is listed as Weight. Further alteration of the user ID was applied to allow the software to differentiate user ID and movie IDs to allow differentiation between movie nodes and user nodes.

5. NETWORK ANALYSIS AND VISUALIZATION

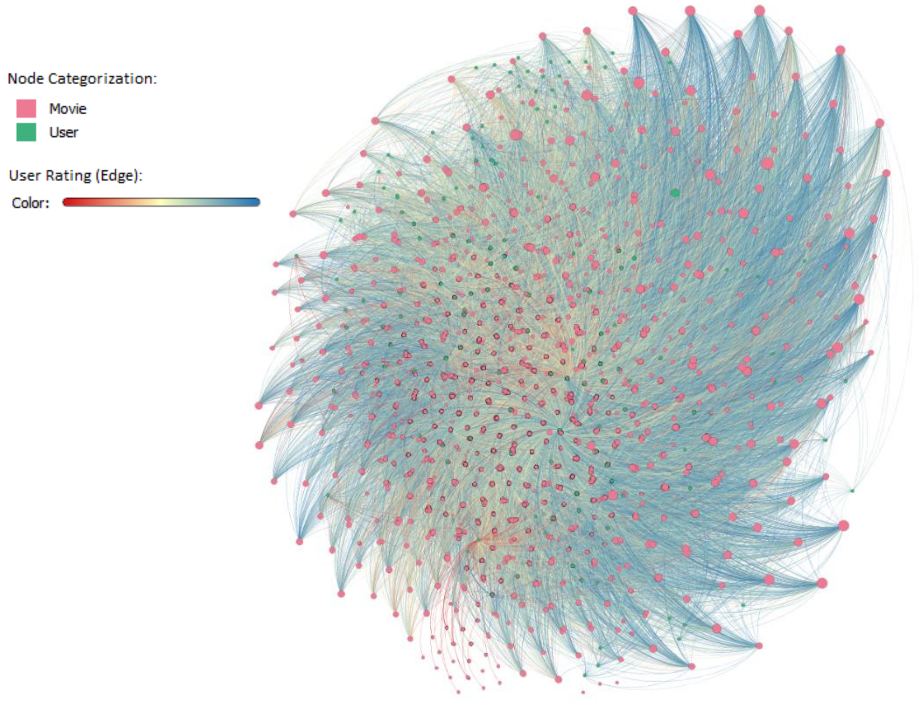
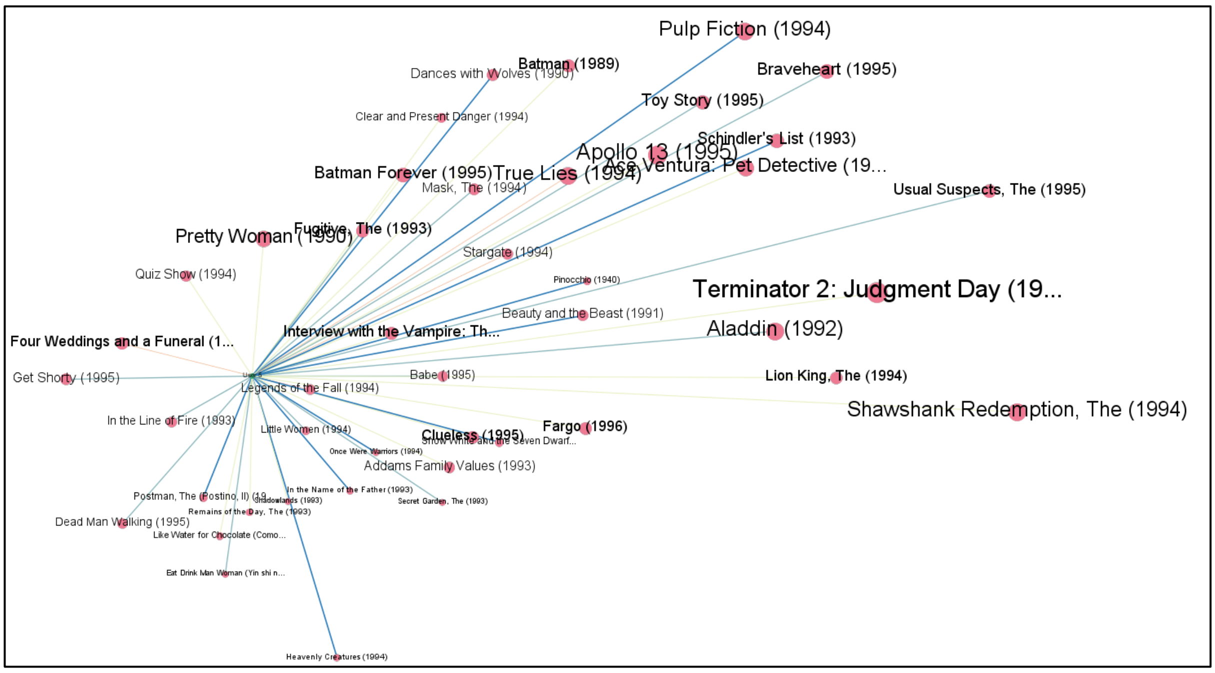


Figure 1: Movies Network Model

Figure 1 represents the overall Movies Network Model, portraying the network of the MovieLens data. This model is a large undirected graph which portrays the interaction between users and movies, demonstrating the user rating of each movie through the colour of the edges, with blue being that of a high rating and red being a low rating. Movies that were lacking any user ratings were filtered out using the 'Giant Component' filter, displaying a large, interconnected graph. Given the amount of data within the dataset, the visualized model can be seen to be quite convoluted, which could also be attributed to the edges of the graph contributing a two-way directed edge between user nodes and movie nodes. These two-way directed edges can be explained as users providing a rating to a movie, where users are able to rate multiple movies and movies can be rated by multiple users.



Pulp Fiction (1994)
Dances with Wolves (1990)
Batman (1989)
Braveheart (1995)
Toy Story (1995)
Clear and Present Danger (1994)
Schindler's List (1993)
Apollo 13 (1995)
Ace Ventura: Pet Detective (19...
Batman Forever (1995)
True Lies (1994)
Mask, The (1994)
Usual Suspects, The (1995)
Pretty Woman (1990)
Fugitive, The (1993)
Stargate (1994)
Quiz Show (1994)
Pinocchio (1940)
Terminator 2: Judgment Day (19...
Beauty and the Beast (1991)
Four Weddings and a Funeral (1...
Interview with the Vampire: Th...
Aladdin (1992)
Get Shorty (1995)
Babe (1995)
Lion King, The (1994)
Legends of the Fall (1994)
Shawshank Redemption, The (1994)
In the Line of Fire (1993)
Little Women (1994)
Clueless (1995)
Snow White and the Seven Dwarf...
Fargo (1996)
Once Were Warriors (1994)
Addams Family Values (1993)
In the Name of the Father (1993)
Postman, The (Postino, Il) (19...
Shadowlands (1993)
Secret Garden, The (1993)
Remains of the Day, The (1993)
Dead Man Walking (1995)
Like Water for Chocolate (Como...
Eat Drink Man Woman (Yin shi n...
Heavenly Creatures (1994)

Figure 2: User-Filtered Network

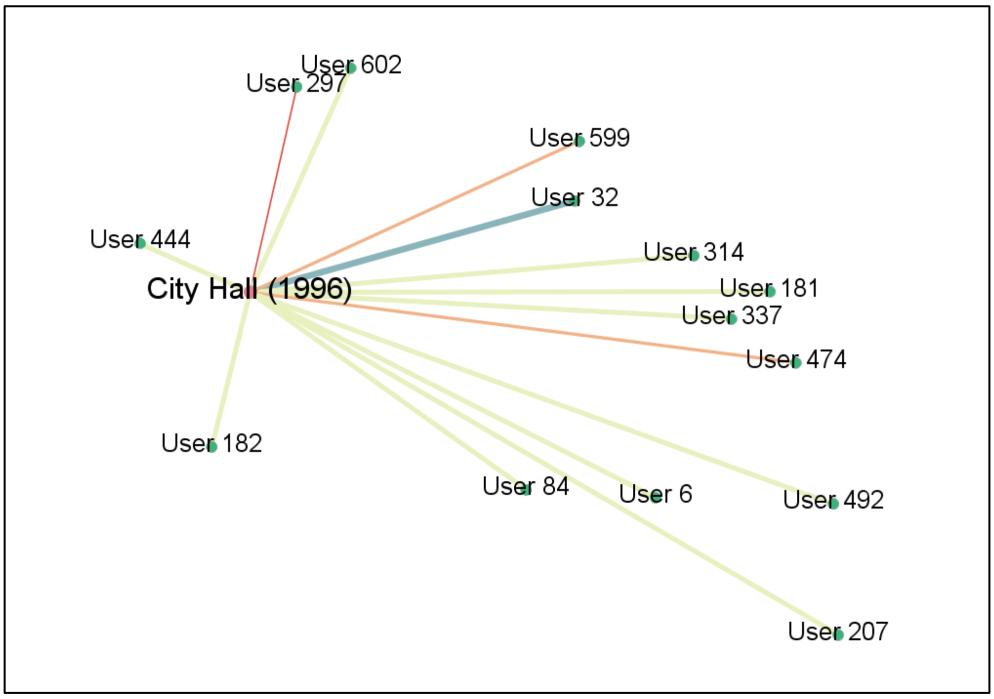


Figure 3: Movie-Filtered Network

However, through Ego Networks as portrayed in Figure 2 and Figure 3, a better demonstration of the specifics of the model is displayed. In Figure 2, it portrays an Ego Network where it shows all the movies rated by the specified user and their rating of the movie as indicated by the edge colour. The opposite can be said for figure 3, in which a movie is the central node in the ego network, displaying all the users who have rated it and their general rating of the movie.

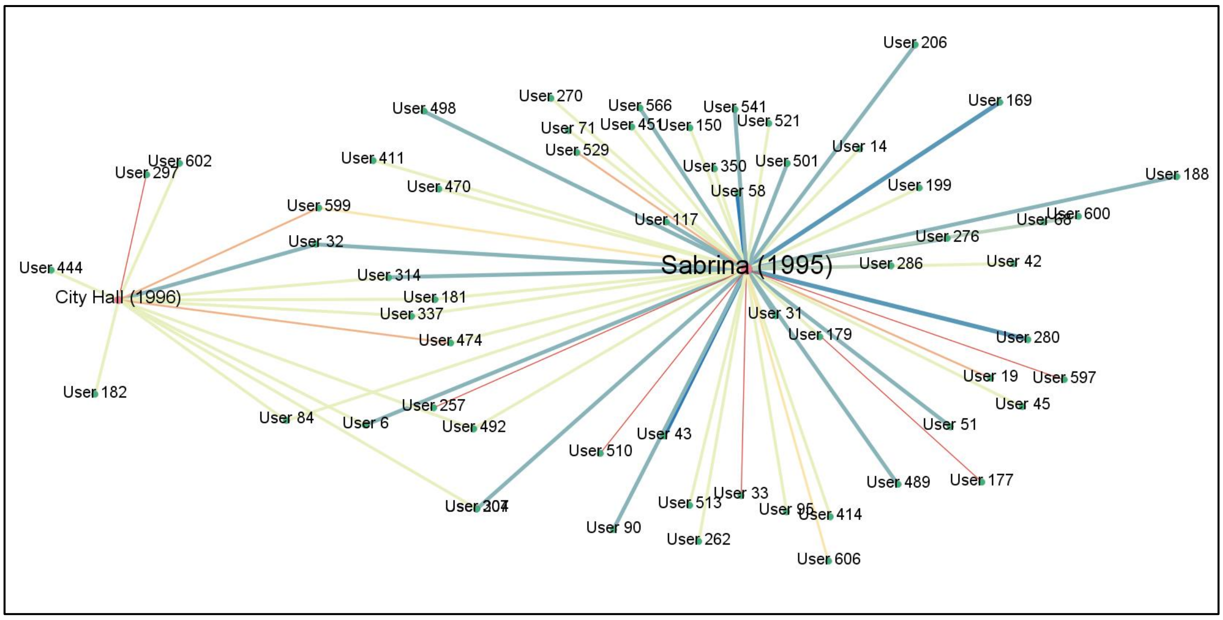


Figure 4: Interconnected Ego-Networks

Figure 4 portrays the ego-networks of two movie nodes. This figure demonstrates the basis of this project's collaborative filtering method of movie recommendation, which shows the users who have rated both movies. Within this figure, it provides a small example of what the following movie recommendation entails, where it shows users rating the movies similarly.

6. RECOMMENDATION SYSTEMS

6.1. Non-Personalised Recommendation Systems (Statistical Techniques)

In developing a recommendation system one potential approach is to use the average. However, what if there is a product that has one, 5-star rating and another with an average rating of 4.99 for 1000 ratings.

As a result of this issue, Bayesian statistics are often used to account for the uncertainty of the rating of a product. A potential approach could be used Wilsonx-Score interval to build a confidence interval of the true mean rating of a product. Then the lowest value in the confidence rating can be taken so that way product with great uncertainty in its rating will not be recommended.

Another statistical approach is "Association Rule Mining". By examining what products are brought together rules can be developed. This can potentially be used to recommend products to a user, while still accounting that some products are bought frequently. So, although let says every time cheese is bought water is bought as well that does not necessarily mean there is a relationship between them.

6.2. Personalised Recommendation Systems

There are personalised recommendations. These provide recommendations based on a person's interest. Collaborative filtering provides a way to provide personalised recommendations based on similarities between users or items.

6.2.1. Intuition for Collaborative Filtering (Pearson Correlation)

Notice that people may have different rating averages. For instance, person A may have an average rating of 3.9/5 and Person B may have an average rating of 4.5/5. So, when we are recommending movies, we should we need to consider the deviation from the average rating has if person A rates a movie 5-stars that is going lot meaningful than person B rating a movie 5-stars.

6.2.2. Derivation of the Formula for Pearson Correlation

The similarity metric we are using is Pearson correlation. We will explain the relationship between Pearson correlation and the formula for sim (x, y) down below.

The Pearson population correlation is defined as:

$$Corr(X, Y) = \frac{Cov(X, Y)}{\sigma_x \cdot \sigma_y}$$

Note there two main types of correlation sample and population.

As we do not, we have all the data we will need to use sample correlation. This done by instead using the sample covariance and sample standard deviation and sample mean.

$$sim(x, y) = \frac{\dfrac{\sum_{s \in S_{xy}} (r_{xs} - \bar{r}_x)(r_{xy} - \bar{r}_y)}{n-1}}{\sqrt{\dfrac{\sum_{s \in S_{xy}} (r_{xs} - \bar{r}_x)^2}{n-1}}\sqrt{\dfrac{\sum_{s \in S_{xy}} (r_{xy} - \bar{r}_y)^2}{n-1}}}$$

$$= \frac{\sum_{s \in S_{xy}} (r_{xs} - \bar{r}_x)(r_{xy} - \bar{r}_y)}{\sqrt{\sum_{s \in S_{xy}} (r_{xs} - \bar{r}_x)^2}\sqrt{\sum_{s \in S_{xy}} (r_{xy} - \bar{r}_y)^2}}$$

Disclaimer: Most of the code used has been taken from the Lazyprogrammer GitHub page. Minor changes were made to refactor the code and bring it all into one file.

We will first begin by importing the necessary Python packages.

```
import numpy as np
import pandas as pd
```

```
from sklearn.utils import shuffle
```

```
from collections import Counter
from sortedcontainers import SortedList
```

We now begin pre-processing the data a bit. We will make the user ids go from 0 to n-1 and do the same for the movie id. As the timestamp column is not needed it will be dropped.

```
df = pd.read_csv('rating.csv')

# make the user ids go from 0...N-1
df.userId = df.userId - 1

# create a mapping for movie ids
unique_movie_ids = set(df.movieId.values)
movie2idx = {}
count = 0
for movie_id in unique_movie_ids:
    movie2idx[movie_id] = count
    count += 1

# add them to the data frame
# takes awhile
df['movie_idx'] = df['movieId'].map(movie2idx)

df = df.drop(columns=['timestamp'])
```

Notice that the user2rating matrix (I.e., utility matrix) size is going to be too large to process on single computer due to RAM issues. As a result, a smaller dataframe needs to be constructed.

As seen below, there would be 3, 703, 856, 792 values in the dataframe if all the data is used.

```
N = df['userId'].nunique() # number of users
M = df['movieId'].nunique() # number of movies
```

```
N*M
```

```
3703856792
```

To create a smaller dataframe we use the following code.

The Counter function returns a dictionary with count of the occurrences of key in the series. We then set n = 1000 and m = 200 to control the user ids number and movies used for the calculation of the weight matrix.

```
# number of users and movies we would like to keep
n = 1000
m = 200

user_ids = [u for u, c in user_ids_count.most_common(n)]
movie_ids = [m for m, c in movie_ids_count.most_common(m)]
```

We then store this result in the variable df_small. As dataframe assignment works based on pointers we need to use the copy method to get a copy and not a view.

```
# make a copy, otherwise ids won't be overwritten
df_small = df[df.userId.isin(user_ids) & df.movie_idx.isin(movie_ids)].copy()
```

We then split the data in training and testing

```
df = df_small


# split into train and test
df = shuffle(df, random_state=1)
cutoff = int(0.8*len(df))
df_train = df.iloc[:cutoff]
df_test = df.iloc[cutoff:]
```

We now need to create user2movie, movie2user and usermovie2rating dictionaries.

```
# a dictionary to tell us which users have rated which movies
user2movie = {}
# a dicationary to tell us which movies have been rated by which users
movie2user = {}
# a dictionary to look up ratings
usermovie2rating = {}
```

To do this we first define some functions:

Note: The comment explains the logic of the code and provides an alternative approach not mentioned in the original code.

```
# Note to avoid using the append function defaultdict from collections can be used instead
# To avoid a key error an if-statement is used for new user-ids and an else for existing ids

count = 0
def update_user2movie_and_movie2user(row):
    global count
    count += 1
    if count % 100000 == 0:
        print("processed: %.3f" % (float(count)/cutoff))

    i = int(row.userId)
    j = int(row.movieId)
    if i not in user2movie:
        user2movie[i] = [j]
    else:
        user2movie[i].append(j)

    if j not in movie2user:
        movie2user[j] = [i]
    else:
        movie2user[j].append(i)

    usermovie2rating[(i,j)] = row.rating
```

We define the function for test data a bit differently as it is not used to construct the weight matrix.

```
def update_usermovie2rating_test(row):
    global count
    count += 1
    if count % 100000 == 0:
        print("processed: %.3f" % (float(count)/len(df_test)))

    i = int(row.userId)
    j = int(row.movie_idx)
    usermovie2rating_test[(i,j)] = row.rating
```

We then apply these functions using the apply method for train and test respectively.

```
df_train.apply(update_user2movie_and_movie2user, axis=1)
```

```
df_test.apply(update_usermovie2rating_test, axis=1)
```

In preparation for the creation of the weights we create the following variables.

```
K = 25 # number of neighbors we'd like to consider
limit = 5 # number of common movies users must have in common in order to consider
neighbors = [] # store neighbors in this list
averages = [] # each user's average rating for later use
deviations = [] # each user's deviation for later use
```

We get the average rating and deviation for each user to make a prediction regarding the predicted rating.

```
for i in range(N):
    # find the 25 closest users to user i
    movies_i = user2movie[i]
    movies_i_set = set(movies_i)

    # calculate avg and deviation
    ratings_i = { movie:usermovie2rating[(i, movie)] for movie in movies_i }
    avg_i = np.mean(list(ratings_i.values()))
    dev_i = { movie:(rating - avg_i) for movie, rating in ratings_i.items() }
    dev_i_values = np.array(list(dev_i.values()))
    sigma_i = np.sqrt(dev_i_values.dot(dev_i_values))

    # save these for later use
    averages.append(avg_i)
    deviations.append(dev_i)
```

Continuing the code from above for each user we calculate the correlation coefficient. This is the Pearson's correlation formula we derived above. The Pearson correlation between two users is the weight between them.

Note that since Corr(X, Y) = Corr(Y, X) the number of weights that need to be calculated can be reduced by using the symmetry although this was not done for this code.

We calculate the weights for each user. If the weight is large enough the user id for the neighbour and the weight are added in a sorted list I.e., variable sl. The K variable controls the number of neighbours in this case a maximum of 25 neighbours is allowed.

```
sl = SortedList()
for j in range(N):
    # don't include yourself
    if j != i:
        movies_j = user2movie[j]
        movies_j_set = set(movies_j)
        common_movies = (movies_i_set & movies_j_set) # intersection
        if len(common_movies) > limit:
            # calculate avg and deviation
            ratings_j = { movie:usermovie2rating[(j, movie)] for movie in movies_j }
            avg_j = np.mean(list(ratings_j.values()))
            dev_j = { movie:(rating - avg_j) for movie, rating in ratings_j.items() }
            dev_j_values = np.array(list(dev_j.values()))
            sigma_j = np.sqrt(dev_j_values.dot(dev_j_values))

            # calculate correlation coefficient
            numerator = sum(dev_i[m]*dev_j[m] for m in common_movies)
            w_ij = numerator / (sigma_i * sigma_j)

            # insert into sorted list and truncate
            # negate weight, because list is sorted ascending
            # maximum value (1) is "closest"
            sl.add((w_ij, j))
            if len(sl) > K:
                del sl[0]
```

The weights and user id are appended together as tuple in the list neighbours

```
    # store the neighbors
    neighbors.append(sl)
```

With the weight matrix in the neighbours variable, we now have everything we need now to make a prediction.

```
def predict(i, m):
    # calculate the weighted sum of deviations
    numerator = 0
    denominator = 0
    for weight, j in neighbors[i]:
        # remember, the weight is stored as its negative
        # so the negative of the negative weight is the positive weight
        try:
            numerator += weight * deviations[j][m]
            denominator += abs(weight)
        except KeyError:
            # neighbor may not have rated the same movie
            # don't want to do dictionary lookup twice
            # so just throw exception
            pass

    if denominator == 0:
        prediction = averages[i]
    else:
        prediction = numerator / denominator + averages[i]
    prediction = min(5, prediction)
    prediction = max(0.5, prediction) # min rating is 0.5
    return prediction
```

The code for prediction is based on the image below where i and i' represent different users and j represent the movies.

The \omega_ j represents the sets of users that have rated movie j that user I has rated. In this scenario a limit of 5 is place on the number of movies required to be in common.

A couple of notes:

1. To avoid the denominator potentially equalling zero we make use of the absolute value sign
2. We do not need add $\bar{r}_i$ back to the rating as it is some unknown constant. The deviance will tell us whether movie will be well like or not. A positive deviance suggesting the movie will be well liked and a negative deviance meaning not well liked.

$$\widehat{rating}(i, j) = \bar{r}_i + \frac{\sum_{i` \in \Omega_j} w_{ii`} \cdot \left(r_{i`j} - \bar{r}_{i`}\right)}{\sum_{i` \in \Omega_j} \left|w_{ii`}\right|}$$

Point 2 means that if we know:

$$\frac{\sum_{i` \in \Omega_j} w_{ii`} \cdot \left(r_{i`j} - \bar{r}_{i`} \right)}{\sum_{i` \in \Omega_j} \left| w_{ii`} \right|}$$

Then we can recommend a movie to a user.

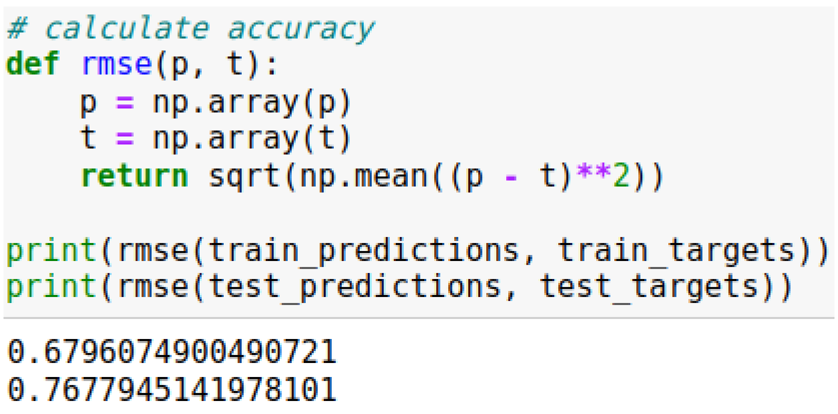
For our implementation for deciding which weights will be used in the calculation we have used weights with the highest positive value. But note that negative weights could also potentially be useful. If for instance Bob likes action movies and Alice dislikes action movies. Knowing what Alice does not like could be useful.

We will now use the predict function. As it is not possible to calculate the mean squared error on the unknown data, we will use the data with known ratings.

```
train_predictions = []
train_targets = []
for (i, m), target in usermovie2rating.items():
    # calculate the prediction for this movie
    prediction = predict(i, m)

    # save the prediction and target
    train_predictions.append(prediction)
    train_targets.append(target)

test_predictions = []
test_targets = []
# same thing for test set
for (i, m), target in usermovie2rating_test.items():
    # calculate the prediction for this movie
    prediction = predict(i, m)

    # save the prediction and target
    test_predictions.append(prediction)
    test_targets.append(target)
```

7. rmse RESULTS

The results for the rmse are:



```python
# calculate accuracy
def rmse(p, t):
    p = np.array(p)
    t = np.array(t)
    return sqrt(np.mean((p - t)**2))

print(rmse(train_predictions, train_targets))
print(rmse(test_predictions, test_targets))
```

```
0.6796074900490721
0.7677945141978101
```

The results for user-user are reasonably accurate. Looking at the benchmark paperswithcode we see most algorithms have around 0.8-0.85 rmse (paperswithcode 2021). However, since we are only using a subset of the full data the actual rmse for our implementation will most be a bit higher given some movies might not have many ratings from users.

8. rmse EVALUATION

8.1. Limitations of various Recommendation Systems Approaches

Collaborative filtering can exploit users’ particular preferences. For example, a user might get recommended anime if they are interested in anime. But this could create a problem of a user only getting recommended anime and not something else they might like. This is the explore-exploit dilemma. Should the person’s interest in anime be exploited or should another type of movie they might like be explored.

8.2. Issues with Collaborative Filtering

One big issue is what if there is a new user or new movie. In this scenario, we will not be able to make a prediction for the MovieLens data. This problem is referred to as the cold start. Another issue is the immense computational power required for the collaborative filtering algorithm. The time complexity for calculating the weight matrix for user-user collaborative filtering is approximately $O(N^2 * M - N)$. Where N is the number of users and M is the number of items. This time complexity will be substantially lower for item-item collaborative filtering I.e. $O(M*N - M)$. This is presuming that the number of items is significantly less than the number of users like at Netflix or Amazon.

Note there would be much more data to compare items with each other than comparing with users. As a result, item by item collaborative filtering is done more often than user by using collaborative filtering in practice. In addition, due to the fact, there is more data available for items, often a better model is obtained.

9. CONCLUSION

A recommendation system is a software tool using a basic algorithm that filters user-related data from huge data to allow users to relate and suggest goods that provide product or item recommendations based on consumer behaviour and current trends. The exponential growth of data and material on the World Wide Web has resulted in an increased difficulty in finding information, a movie recommendation system such as would assist users in decision-making over large data domains. Various recommendation strategies which have risen in popularity were explored within the report. Collaborative filtering is a prominent strategy for recommendation systems however there are several limitations with the system such as time complexity which are explored further in the report. Statistical approaches were found to be much more computation feasible and simpler compared to collaborative filtering or approaches involving the neural network.

10. REFERENCES

1. Adomavicius, G. and Tuzhilin, A., 2005. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE transactions on knowledge and data engineering*, *17*(6), pp.734-749.

2. Aggarwal, C.C., 2016. *Recommender systems* (Vol. 1). Cham: Springer International Publishing.

3. De Campos, L.M., Fernández-Luna, J.M., Huete, J.F. and Rueda-Morales, M.A., 2010. Combining content-based and collaborative recommendations: A hybrid approach based on Bayesian networks. *International journal of approximate reasoning*, *51*(7), pp.785-799.

4. Grouplens 2013. MovieLens. Available online: [https://grouplens.org/datasets/movielens/]

5. Harper, F.M. and Konstan, J.A., 2015. The MovieLens datasets: History and context. *ACM transactions on interactive intelligent systems (tiis)*, *5*(4), pp.1-19.

6. Jeong, W.H., Kim, S.J., Park, D.S. and Kwak, J., 2013. Performance improvement of a movie recommendation system based on personal propensity and secure collaborative filtering. *Journal of Information Processing Systems*, *9*(1), pp.157-172.

7. Kumar, M., Yadav, D.K., Singh, A. and Gupta, V.K., 2015. A movie recommender system: Movrec. *International Journal of Computer Applications*, *124*(3).

8. Lazyprogrammer 2022. Recommenders. *Github.* Available online: [https://github.com/lazyprogrammer/machine_learning_examples/tree/master/recommenders]

9. Paperswithcode 2021. Recommendation Systems on MovieLens 1M. Available online: [https://paperswithcode.com/sota/collaborative-filtering-on-movielens-1m]

10. Souri, A., Hosseinpour, S. and Rahmani, A.M., 2018. Personality classification based on profiles of social networks users and the five-factor model of personality. *Human-centric Computing and Information Sciences*, *8*(1), pp.1-15.

11. Terveen, L. and Hill, W., 2001. Beyond recommender systems: Helping people help each other. *HCI in the New Millennium*, *1*(2001), pp.487-509.

12. Wang, Y. and Zhu, L., 2016, December. Research on collaborative filtering recommendation algorithm based on Mahout. In *2016 4th Intl Conf on Applied Computing and Information Technology/3rd Intl Conf on Computational Science/Intelligence and Applied Informatics/1st Intl Conf on Big Data, Cloud Computing, Data Science & Engineering (ACIT-CSII-BCD)* (pp. 400-405). IEEE.

13. Zou, H., He, Y., Zheng, S., Yu, H. and Hu, C., 2018. Online group recommendation with local optimization. *Computer Modeling in Engineering & Sciences*, *115*(2), pp.217-231.