Item-Based Collaborative Filtering Recommendation System

Harsh Bijlani and **Rongon Chatterjee**
University of Texas at Arlington
701 S Nedderman Dr
Arlington, Texas 76019

Abstract

Recommendation systems play a pivotal role in today's digital space. In this project, we have worked on the MovieLens data set by GroupLens Research to give top ten recommendations with regard to a target movie. We have implemented an item based collaborative filtering system, that implies that our recommendations are dependent on the movie ratings by the users in our data set. In our approach, we have implemented from scratch a nearest neighbors algorithm using different contemporary distance metrics and we have also worked on a correlation based approach. We analyzed the performance of our movie recommendation system by comparing the results of applying different correlation measures and distance metrics in the nearest neighbors function.

1 Introduction

In this digital era, online video streaming platforms are growing rapidly and recommendation systems are a key component of such websites. By giving meaningful recommendations they help in increasing user engagement and makes the website a robust successful product. Rapid growth in the amount of information available online and increasing number of visitors on websites present some key challenges for recommendation systems, such as - producing high quality recommendations, performing many recommendations per second for millions of users and items and producing high coverage in the face of data sparsity (Sarwar et al. 2001)

2 Dataset Description

The data set that we used for this project is the "ml-latest small" data set from the GroupLens' MovieLens website (Harper and Konstan 2015). It contains 100,000 ratings of 9000 movies by 600 users.

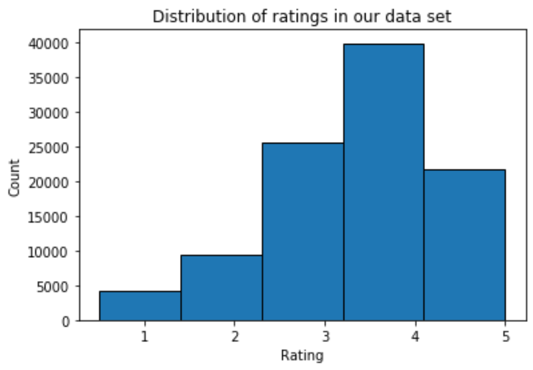


Figure 1: Rating Distribution

On analysing the *movies.csv* file we discovered the following information about its attributes:

1. MovieID
   - Nominal Attribute
   - Type: Positive Integer
   - Is Unique
   - Movie Identifier
2. Title
   - Nominal Attribute
   - Type: Alphanumeric String
   - Is Unique
   - Movie Name and Release Year
3. Genres
   - Nominal Attribute
   - Type: String
   - Not Unique
   - Combinations of different genres separated by "|"

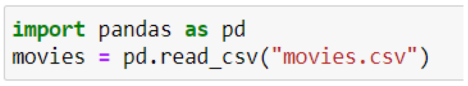
On analysing the *ratings.csv* file we discovered the following information about its attributes:

1. UserId
   - Nominal Attribute
   - Type: Positive Integer
   - Unique with respect to users but not unique with respect to data

- User Identifier

2. MovieID
   - Nominal Attribute
   - Type: Positive Integer
   - Unique with respect to movies but not unique with respect to data
   - Movie Identifier
3. Rating
   - Ordinal Attribute
   - Floating point numbers range(0-5)
   - Not Unique
   - Rating given by user (UserId) to a movie (MovieID)
4. Timestamp
   - Ordinal Attribute
   - Type: Long Integers
   - Not Unique
   - Time at which user (UserId) rated a movie (MovieID)

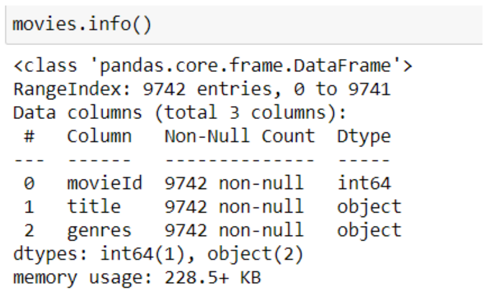
The key challenge in data preprocessing is dimensionality reduction. We did this by removing irrelevant attributes from our data set. By irrelevant we mean to say the attributes that are not important to us in this approach. This helped us to save computation time and to produce more meaningful results. We have used the popular Pandas library in Python for data manipulation and preprocessing. Firstly, we loaded the "movies.csv" file by using the command:



```
import pandas as pd
movies = pd.read_csv("movies.csv")
```

Figure 2: Importing Pandas

Summary of the movies DataFrame:

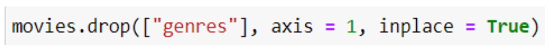


```
movies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9742 entries, 0 to 9741
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   movieId  9742 non-null   int64 
 1   title    9742 non-null   object
 2   genres   9742 non-null   object
dtypes: int64(1), object(2)
memory usage: 228.5+ KB
```

Figure 3: Details of the movie DataFrame

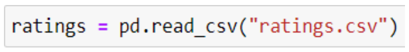
Thereafter, we removed the "genres" column from the movies DataFrame. For that we used the following piece of code:



```
movies.drop(["genres"], axis = 1, inplace = True)
```

Figure 4: Removing irrelevant attributes

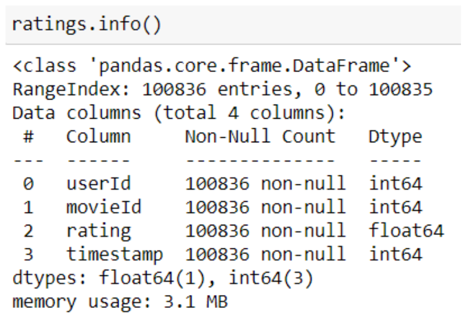
Then we loaded "ratings.csv" file using the command:



```
ratings = pd.read_csv("ratings.csv")
```

Figure 5: Loading the ratings data

Summary of the ratings DataFrame:

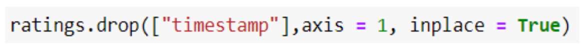


```
ratings.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100836 entries, 0 to 100835
Data columns (total 4 columns):
 #   Column     Non-Null Count   Dtype  
---  ------     --------------   -----  
 0   userId     100836 non-null  int64  
 1   movieId    100836 non-null  int64  
 2   rating     100836 non-null  float64
 3   timestamp  100836 non-null  int64  
dtypes: float64(1), int64(3)
memory usage: 3.1 MB
```

Figure 6: Details of the ratings DataFrame

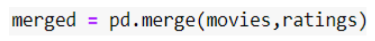
After that, we removed the "timestamp" column from the ratings DataFrame. For that we used the following piece of code:



```
ratings.drop(["timestamp"],axis = 1, inplace = True)
```

Figure 7: Removing irrelevant attributes

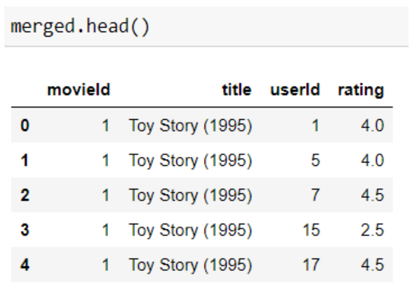
The next step is to join the two DataFrames movies and ratings. That we did using the code:



```
merged = pd.merge(movies,ratings)
```

Figure 8: Merging the DataFrames

The merged DataFrame looks like:

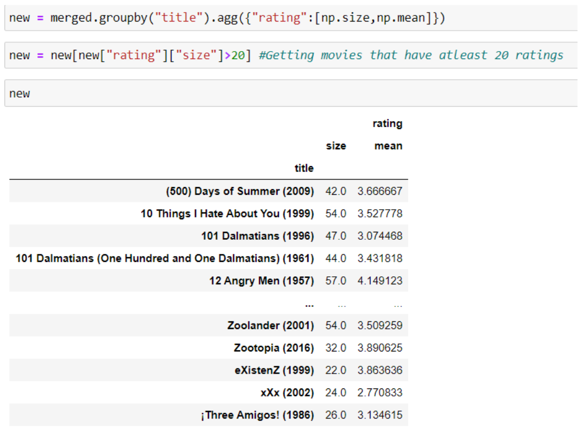


```
merged.head()
```

	movieId	title	userId	rating
0	1	Toy Story (1995)	1	4.0
1	1	Toy Story (1995)	5	4.0
2	1	Toy Story (1995)	7	4.5
3	1	Toy Story (1995)	15	2.5
4	1	Toy Story (1995)	17	4.5

Figure 9: Merged DataFrame

Now, comes the most important step of the data preprocessing part. We want to filter out those movies that have been rated by less than 20 users. This is done so that our model can return better recommendations as more the rating data for each movie, better will be the results. This is possible using some aggregate functions of the pandas and numpy library, which is demonstrated in the following piece of code:



```
new = merged.groupby("title").agg({"rating":[np.size,np.mean]})
```

```
new = new[new["rating"]["size"]>20] #Getting movies that have atleast 20 ratings
```

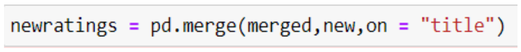
```
new
```

	rating	
	size	mean
title		
(500) Days of Summer (2009)	42.0	3.666667
10 Things I Hate About You (1999)	54.0	3.527778
101 Dalmatians (1996)	47.0	3.074468
101 Dalmatians (One Hundred and One Dalmatians) (1961)	44.0	3.431818
12 Angry Men (1957)	57.0	4.149123
...	...	...
Zoolander (2001)	54.0	3.509259
Zootopia (2016)	32.0	3.890625
eXistenZ (1999)	22.0	3.863636
xXx (2002)	24.0	2.770833
¡Three Amigos! (1986)	26.0	3.134615

Figure 10: Handling the data sparsity problem

In the next step, we created a subset of the merged DataFrame called "new" which is a filtered collection of the movies that have been rated by more than 20 users. The "size" column shows the number of ratings each movie has received.

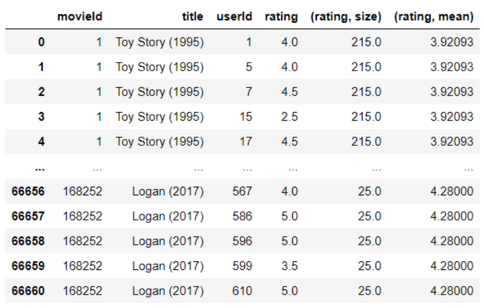
Next, we merged the filtered movies DataFrame, "new" with our already merged DataFrame using the command:



```
newratings = pd.merge(merged,new,on = "title")
```

Figure 11: Merging the filtered DataFrame

Here the parameter "on = "title"" means that the DataFrames are being joined with respect to the title column. Now, the "newratings" DataFrame looks like:



	movieId	title	userId	rating	(rating, size)	(rating, mean)
0	1	Toy Story (1995)	1	4.0	215.0	3.92093
1	1	Toy Story (1995)	5	4.0	215.0	3.92093
2	1	Toy Story (1995)	7	4.5	215.0	3.92093
3	1	Toy Story (1995)	15	2.5	215.0	3.92093
4	1	Toy Story (1995)	17	4.5	215.0	3.92093
...	...	...	...	...	...	...
66656	168252	Logan (2017)	567	4.0	25.0	4.28000
66657	168252	Logan (2017)	586	5.0	25.0	4.28000
66658	168252	Logan (2017)	596	5.0	25.0	4.28000
66659	168252	Logan (2017)	599	3.5	25.0	4.28000
66660	168252	Logan (2017)	610	5.0	25.0	4.28000

Figure 12: DataFrame containing movies that have been rated by more than 20 users

The number 20, was experimentally chosen. We tried many different thresholds such as 5, 10, 20, 50 and 100, and we saw that if we chose a threshold of more than 20, then a lot of popular movies were filtered out, whereas if we selected a number less than 20, then our data set still consisted of a lot of unpopular movies. So, we decided to retain movies that have been rated by at least 20 users.

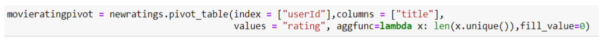
3 Project Description

3.1 Description

Our implementation consists of two approaches, broadly speaking, they are:

1. Correlation based
2. Nearest neighbors approach

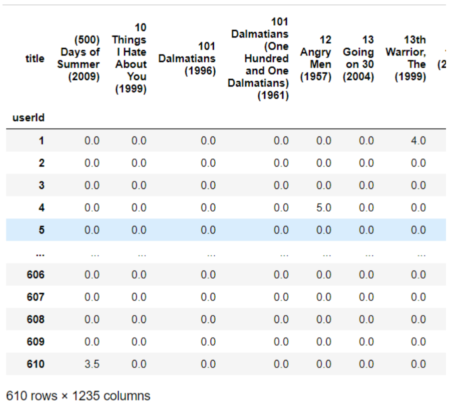
Correlation Approach: As the first step of our implementation, we created a pivot table with "UserID" as the rows and each movie title as the columns. On the values of this pivot table, we have applied a lambda function, so now each value is either a 0 if that movie has not been rated by that user or a 1 if it has been rated by that user. This conversion is necessary for the implementation of cosine similarity. This is achieved using the following piece of code:



```
movieratingpivot = newratings.pivot_table(index = ["userId"],columns = ["title"],
                                          values = "rating", aggfunc=lambda x: len(x.unique()),fill_value=0)
```

Figure 13: Creating a pivot table

The movieratingpivot DataFrame looks like:

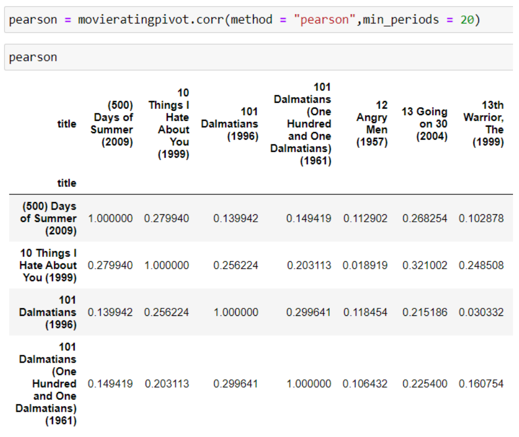


title	(500) Days of Summer (2009)	10 Things I Hate About You (1999)	101 Dalmatians (1996)	101 Dalmatians (One Hundred and One Dalmatians) (1961)	12 Angry Men (1957)	13 Going on 30 (2004)	13th Warrior, The (1999)	(2
userId								
1	0.0	0.0	0.0	0.0	0.0	0.0	4.0	
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	5.0	0.0	0.0	
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	
606	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
607	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
608	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
609	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
610	3.5	0.0	0.0	0.0	0.0	0.0	0.0	

610 rows × 1235 columns

Figure 14: Pivot Table Content

We then apply pearson correlation on this DataFrame using the .corr function of pandas and store the results in a DataFrame called pearson as shown below:



```
pearson = movieratingpivot.corr(method = "pearson",min_periods = 20)
```

```
pearson
```

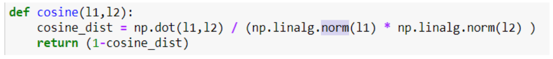
title	(500) Days of Summer (2009)	10 Things I Hate About You (1999)	101 Dalmatians (1996)	101 Dalmatians (One Hundred and One Dalmatians) (1961)	12 Angry Men (1957)	13 Going on 30 (2004)	13th Warrior, The (1999)
title							
(500) Days of Summer (2009)	1.000000	0.279940	0.139942	0.149419	0.112902	0.268254	0.102878
10 Things I Hate About You (1999)	0.279940	1.000000	0.256224	0.203113	0.018919	0.321002	0.248508
101 Dalmatians (1996)	0.139942	0.256224	1.000000	0.299641	0.118454	0.215186	0.030332
101 Dalmatians (One Hundred and One Dalmatians) (1961)	0.149419	0.203113	0.299641	1.000000	0.106432	0.225400	0.160754

Figure 15: Pearson Correlation

Each value tells us the similarity based on the pearson correlation coefficient. Similarly, we applied the spearman correlation coefficient and kendall correlation coefficient and analyzed the results.

Nearest Neighbors Approach: In this approach, we implemented a nearest neighbors function from scratch which can be quite useful for applications like a movie recommendation system. Firstly, we implemented some distance functions like:

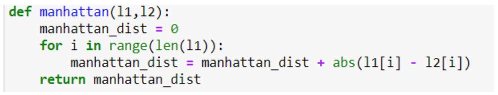
- Cosine Similarity -



```
def cosine(l1,l2):
    cosine_dist = np.dot(l1,l2) / (np.linalg.norm(l1) * np.linalg.norm(l2) )
    return (1-cosine_dist)
```

Figure 16: Implementing Cosine Distance Function

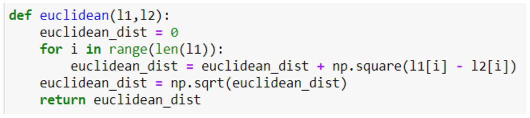
- Manhattan Distance -



```
def manhattan(l1,l2):
    manhattan_dist = 0
    for i in range(len(l1)):
        manhattan_dist = manhattan_dist + abs(l1[i] - l2[i])
    return manhattan_dist
```

Figure 17: Implementing Manhattan Distance Function

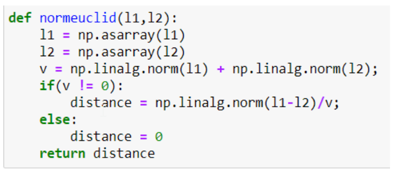
- Euclidean Distance -



```
def euclidean(l1,l2):
    euclidean_dist = 0
    for i in range(len(l1)):
        euclidean_dist = euclidean_dist + np.square(l1[i] - l2[i])
    euclidean_dist = np.sqrt(euclidean_dist)
    return euclidean_dist
```

Figure 18: Implementing Euclidean Distance Function

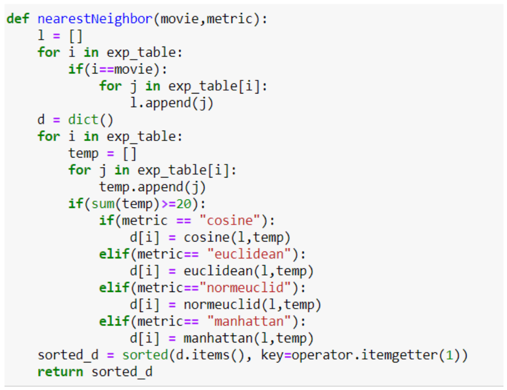
- Normalized Euclidean Distance -



```
def normeuclid(l1,l2):
    l1 = np.asarray(l1)
    l2 = np.asarray(l2)
    v = np.linalg.norm(l1) + np.linalg.norm(l2);
    if(v != 0):
        distance = np.linalg.norm(l1-l2)/v;
    else:
        distance = 0
    return distance
```

Figure 19: Implementing Normalized Euclidean Distance Function

In the next step, we again create a pivot table and fill in the "na" values with 0. The next step is the main part of our implementation wherein we define the nearest neighbor function:



```
def nearestNeighbor(movie,metric):
    l = []
    for i in exp_table:
        if(i==movie):
            for j in exp_table[i]:
                l.append(j)
    d = dict()
    for i in exp_table:
        temp = []
        for j in exp_table[i]:
            temp.append(j)
        if(sum(temp)>=20):
            if(metric == "cosine"):
                d[i] = cosine(l,temp)
            elif(metric== "euclidean"):
                d[i] = euclidean(l,temp)
            elif(metric=="normeuclid"):
                d[i] = normeuclid(l,temp)
            elif(metric== "manhattan"):
                d[i] = manhattan(l,temp)
    sorted_d = sorted(d.items(), key=operator.itemgetter(1))
    return sorted_d
```

Figure 20: Implementing Nearest Neighbors Function from scratch

The nearestNeighbor function takes two parameters, namely, "movie" and "metric". Movie can be any movie title having received a rating by more than 20 users, as our user-ratings pivot table is a filtered collection of all the movies present in the data set that have been rated by more than 20 users. The "metric" parameter can take values "cosine", "euclidean", "normeuclid" and "manhattan". The function first creates a list "l" and appends the ratings given by each user to our target movie. Next, we iterate through each column (movie) in the pivot table and calculate their distance based on the "metric" passed in the argument with our target movie and append it in a dictionary "d". Lastly, we sort our dictionary and return it. The dictionary contains the movie title as key and the distance from our target movie as the corresponding value. Lesser the distance between the movies, more the similarity between those movies.

3.2 Main references used

The implementation has not been adopted from any reference papers but for the literature review and to get an idea of the objective we referred the following papers - (Sarwar et al. 2001), (Gong 2010) and (Feng et al. 2018).

3.3 Difference in APPROACH/METHOD between your project and the main projects of your references

In our project we have implemented a movie recommendation system using two different approaches. One is a correlation based approach and we have written a nearest neighbors function from scratch as well. For the analysis of the results, we have drawn parallels using a lot of parameters such as same director, same genre, popular actor-director etc. We also compared the results of using different distance metrics to generate the top ten movie recommendations for any target movie.

3.4 Difference in ACCURACY/PERFORMANCE between your project and the main projects of your references

Defining accuracy is a bit abstract when it comes to movie recommendations since it is not a classification task but we saw that the quality of recommendations generated by our nearest neighbors algorithm was quite good. Using item based collaborative filtering our model generated movies that had similar actors, directors, actor-director combinations and similar genres. It is important to note here that the similar movies are actually calculated based on the opinions or movie ratings of the other users in our data set, who have watched the target movie. The system is efficient and does not have any significant latency. The only slightly time consuming step is while generating the pivot table, that roughly takes around two to three seconds. Generating recommendations is almost instantaneous.

3.5 List of your contributions in the project (your work)

1. To deal with the challenge of data sparsity we selected a threshold of 20. This allows us to reduce the size of our user-movie rating matrix and yet retain many movies and produce meaningful recommendations.
2. In our approach, we have used multiple correlation measures such as Pearson correlation, Spearman correlation and Kendall correlation and then analyzed the recommendations provided by each method.
3. Implemented a nearest neighbors function, in which we implemented the distance measures such as cosine similarity, euclidean, normalized euclidean and manhattan from scratch.
4. The nearest neighbors algorithm also defined from scratch uses all the above mentioned distance metrics.
5. Compared the results of these different distance metrics applied on the nearest neighbors algorithm.

4 Analysis

4.1 What did I do well?

1. Implemented a nearest neighbors algorithm and multiple distance metrics from scratch.
2. Incorporated multiple correlation measures in the project.
3. Analyzed and compared the results of the two approaches using different correlation measures and distance metrics (for the nearest neighbors function) on several movies.
4. Filtered out unpopular movies using an optimal threshold for this particular data set.

4.2 What could I have done better?

1. Could have implemented a function that would quantify the quality of the recommendations produced.

4.3 What is left for future work?

- We can scale up this model so that it can be applied on a larger dataset such as the MovieLens 25M dataset. We had tried to use that data set but our computers do not have enough processing power to handle that much data.
- We can explore more advanced distance measures and include them in our model
- We can also add a web based GUI to this application to make it more user friendly.

5 Conclusion

1. Satisfactory results when using correlation based approach.
2. Using different correlation coefficients did not have a drastic or even any noticeable change in our recommendation results.
3. Nearest neighbor algorithm gave great results.
4. Cosine similarity metric gave the most optimal results followed by normalized euclidean, then euclidean and lastly manhattan that gave the least optimal results.

References

Feng, J.; Fengs, X.; Zhang, N.; and Peng, J. 2018. An improved collaborative filtering method based on similarity. *PloS one* 13(9).

Gong, S. 2010. A collaborative filtering recommendation algorithm based on user clustering and item clustering. *JSW* 5(7):745–752.

Harper, F. M., and Konstan, J. A. 2015. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.* 5(4).

Sarwar, B.; Karypis, G.; Konstan, J.; and Riedl, J. 2001. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, 285–295.