# **COMP9417 - Machine Learning & Data Mining**

Project Report  
(Movie Recommender System)

|  |  |
| --- | --- |
| **Name** | **zId** |
| Jagadish Ragavan | z5226835 |
| Phillip Vasilescu | z5208739 |
| Raghav Singh | z5190887 |
| Srihari Vasudevan | z5200336 |

# 

# 

Link to resources: <https://drive.google.com/open?id=1NGjfhGpko0xB5_O4y_cszby-ysJDBKuW>

# 

# 

# 

# 

# 

# **Introduction**

The global outbreak of accessible internet and smart devices has opened up a plethora of movie streaming platforms, allowing more people than ever access to an ever increasing array of films. With over 4 million IMDb titles, users often are overwhelmed by the number of options available for them to watch and unable to make an informed decision on the matter of just what to watch. We propose a Movie Recommender System which tailors to the interests of the user and suggests movies the user might like as a solution to this problem. This system should save the user time of choosing the recommendations and give suggestions they themselves may not have even been considering or aware of.

Large companies such as Netflix, Amazon and YouTube, to name a few, make use of recommender systems to recommend content and products to their customers, based on the users previous recorded feedback. These systems are often immensely valuable to companies (Lohr, 2009), with their business model relying on giving the consumers what they want, even if the consumers aren’t yet aware that they want it. These systems are tailored to the purposes of these large platforms as they are fueled by the vast libraries of information gathered on users and given by users that would be difficult to meaningfully interpreted by other methods in such a way that tailors individually. However, certain methods do have their downsides, which must always be kept in mind when implementing these systems so that any disadvantages can be mitigated, and the advantages of the approach leveraged.

The focus of this project is to explore the Collaborative Filtering approach for recommender systems and observe the advantages and disadvantages of the method, specifically in the context of movie recommendations. We will compare the 2 approaches of constructing similarity matrix between user-to-user and movie-to-movie models. We will also look at how this model is affected if we remove the bias by deducting the mean of ratings. Finally, we will compare all of this to an alternating least square algorithm using PySpark libraries. We completed a course on DataCamp ([“Building Recommendation Engines with PySpark”](https://www.datacamp.com/courses/recommendation-engines-in-pyspark)) to learn this model.

# **Related Work**

Recommender systems have been on the rise in all domains, and researchers are looking into these topics to broaden their horizons and improve the efficiency of this system which is being implemented in all fields. Researchers state one of the most successful techniques for recommender systems as Collaborative filtering (R. Torres, 2004). Recommender systems have crept their way into all domains such as e-commerce (Panniello U., 2014), music (N. Hariri, 2012), movie recommendation (Colombo-Mendoza L. O., 2015), audio CD (Maes, 1995) and even research papers (S. M. McNee, 2002). The world is moving towards automation and recommender systems seem like a step towards an automated sales person for all domains of commerce.

# **Implementation**

### Dataset

The dataset used for training and testing purpose is the MovieLens Dataset. We are going forward with the latest dataset with 100,836 user ratings. The dataset consists of the information of 100,000 ratings and 3,600 tag applications applied to 9,000 movies by 600 users, where each user has rated at least 20 movies. The dataset consists of 4 csv files ‘links.csv’, ‘movies.csv’, ‘ratings.csv’ and ‘tags.csv’, from these we compiled the relevant information for use by our algorithm.

The ‘ratings.csv’ specifically contains information about the ratings given by all the users for the movies that they have rated. The users are represented by their user ids and give ratings on a scale from 0 to 5 stars in increments of 0.5. These ratings form the basis of our recommender system, they are the means by which we inform our recommendation, using these to draw comparisons between users and establish similarities between users with similar taste ie. those who have given similar recommendations to the same or similar movies. As each user in the dataset has given a minimum of 20 reviews, this is sufficient for our algorithm to profile the users in order to establish similarities between them.

### Data preparation:

The data obtained from the dataset is cleaned before the training and testing of the algorithm. The ratings of a movie are normalised with respect to the users who have rated them. This is done so because a person can be either too critical or too lenient in their ratings. This tendency of the user is observed for all their ratings and the movies rated by them are given a normalized rating with respect to other users.

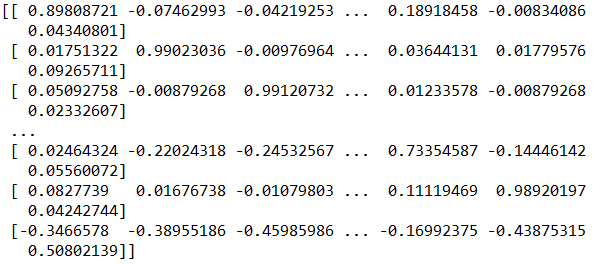
Collaborative Filtering:

A recommender system using collaborative filtering predicts the likelihood of a user liking a movie based on the user's past behaviour and the similarity to other users. We tried implementing two methods using collaborative filtering: user-based and item-based.

User-based Collaborative Filtering:

User based collaborative filtering creates a *user vs user* matrix where each row and column represent the user. The matrix is filled with the similarity between each user. With this method we can deduce how similar a user is with respect to other users.

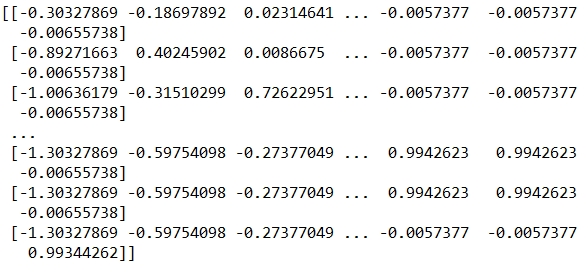
However, a user based system has a few notable disadvantages. Scalability issues arise and will be difficult to handle as the number of users will naturally be much more than the number of movies. Taking into consideration the fact that habits of people may change over time, a user based system does not provide an accurate recommendation if the user's tastes were to change. It relies on the categorisation of users as similar under the assumption their tastes are static.

****

**(Fig 1.1: This is the user similarity matrix)**

Item-based Collaborative Filtering:

Item based collaborative filtering create a *movie vs movie* matrix where each row and each column represents the movies. In this method, a cosine based similarity is calculated among the items, rather than the users. It calculates the item similarities and gives a prediction of the rating for a particular item. (item = movies)

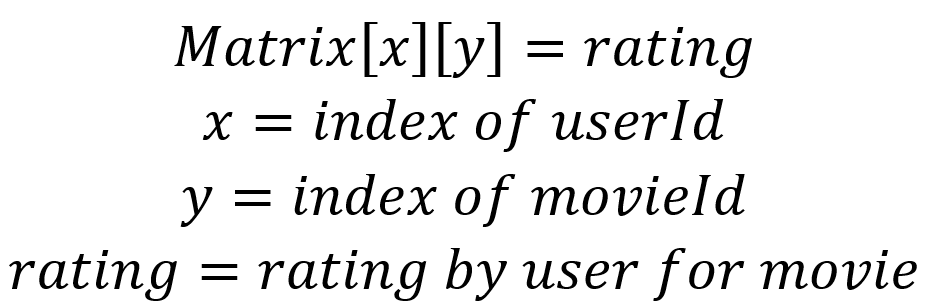
****

**(Fig 1.2: This is the movie similarity matrix)**

**A brief description of the implementation:**

We use the ratings.csv in which there are 4 columns (‘user id’, ‘movie id’, ‘ratings’ and ‘timestamp’). \*‘timestamp’ is not used. Not every user rated every movie and the ‘userId’ was not consistent so we created a mapping by creating a list of set and using its userId’s index to represent them. We replaced this for similarity matrix. The same process was done for similarity for ‘movieId’ as well.

Now we initialise a matrix with the length of the user list and the length of the movie list and populate it with zeros (np.zeros). Then iterated over the dataframe and initialised and put the ratings into the matrix we initialised in the following format:

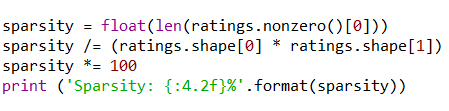


We then split the data into two parts:

* Testing
* Training

These are split by roughly 80% and 20% as each user rates at least 20 movies

We do this by randomly removing 5 ratings for for every user. This is removed from training and will be used for testing. To make the split between testing and training, we do a sparsity check on the generated matrix.



This gives sparsity 1.70%

Now we create the similarity matrix for movies and users by cross multiplying the same matrix’s transpose. Additionally, we have to assign all 0 values 1e-8 to prevent the division error. After this we normalize the similarity matrix.

The algorithm attempts 2 methods:

|  |  |
| --- | --- |
| With Bias | Without bias |
| We continue with the generated similarity matrix. | Calculate mean using every rating given by a particular user and subtract it from all the ratings for that user. |

To increase the accuracy we find a set of similar users/movies to predict their rating. Conventionally it is believed that the more similar users/movies you have the prediction will be more accurate so we start with the similarity of 25 users/movies, then 50, then 100, and finally 200.

These similar users/movies are used to generate a prediction matrix. This matrix has all the missing ratings across every user/movie. For a given number of similar users, the generated prediction matrix is passed along with the training or testing data to find the mean square error. Thus we end up with 4 mean errors for each of the users/movies testing & users/movies training data. There are 4 lists of 4 errors each for biased and unbiased plotted on the graph. Refer to the results section for images.

Algorithm: Collaborative Filtering using similarity matrix for user-user & movie-movie.  
Refer to the appendix for the pseudocode of the program.

**ALS Algorithm Analysis**

After this we did a little research on Alternating Least Squares (ALS) algorithm implementation using Pyspark. For this we did a course on DataCamp. Here is how it worked and why we think it’s better:

For ALS we look at the user-movie rating. We know it works really well with sparse matrices as it fills rating with prediction based on similar users and takes the highest values and make them recommendations that can be offered to users.  
  
As far as the ALS algorithm is concerned, it is done so:  
We can have a matrix R which we can factor into two smaller ones U and P is such a way that when you multiply these gives R. We now look at the error function given below and compare run the values through that. The algorithm feeds ratings and Iteratively changes values to minimise error formula. Finally, multiply U and P that gives R.

  
Here we multiply the cost function into the regularisation function.

RegressionEvaluator works to measure performance. We also import for cross-validate and fine tune hyper parameters of the model.

This model heavily reduces the error as compared to the one we implemented ourselves before.’

# **Experimentation (Results)**

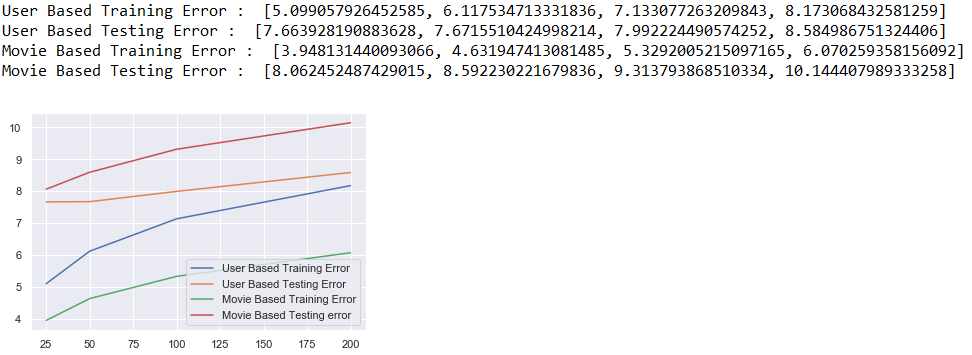
The above graph represent the top most rated 20 movies. We will use this to get a new user to rate them as they might be the popular ones among others.

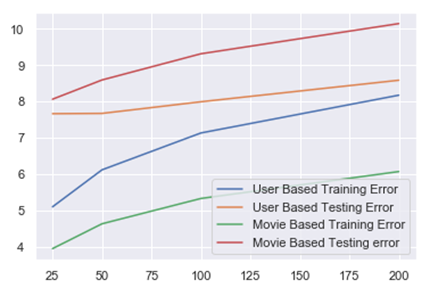
# 

How is the data Tested:

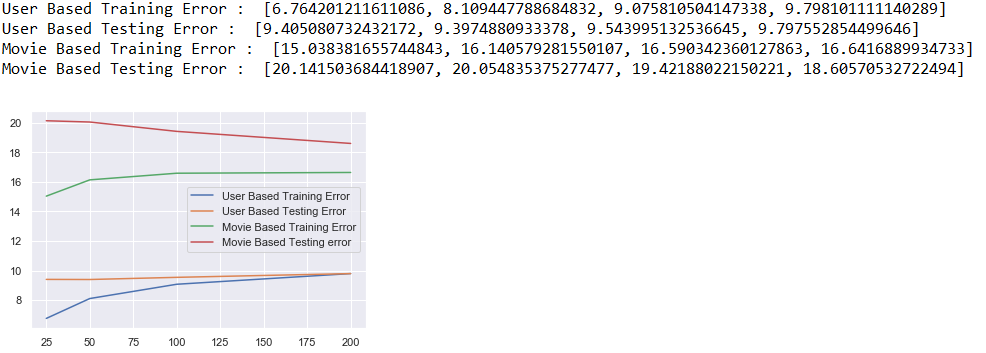
The dataset entire is split into training and testing data. 20% is used for testing and 80% is used for training purposes.

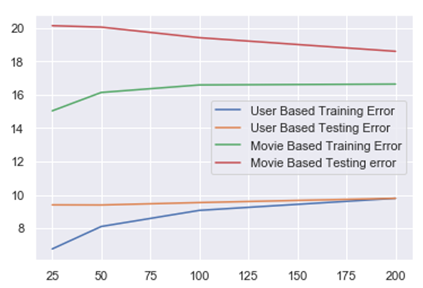
According to the data below we find that unbiased movie similarity matrix yields better result. This is because the bias is based on the user’s rating, which when removed makes the data smooth and consistent among all users. As stated above, the item based collaborative filter has its pros. If we look closely below the error rate keeps decreasing as we increase the number of similar users.





Above is the biased result of the user and movie training and testing data. We can see that the error keeps increasing but is coming to converge. This shows that with enough data there is a chance that the algorithm will yield a more accurate result with a smaller MSE.





Above is the unbiased result of the user and movie training and testing data. We can see that the error keeps decreasing on the test data but increasing on the training data and yet converging. This shows that with enough data there is a chance that the algorithm will yield a more accurate result with a smaller MSE.

# **Future Work**

Training the data in for larger dataset would make the algorithm more robust. But due to computational limitations of the system this project was limited to a smaller dataset. Even though the dataset provided tags for each movie provided by users, the inconsistency and unreliability of the data might cause overfitting. A further cleaning of the tags using a reliability table would provide more information which can be used to better the recommender system. A machine learning algorithm to clean the user tags would be a valuable addition to the algorithm in the future.We tried to implement a GUI based system for taking in a new user. This can be implemented for testing later.

# **Conclusion**

A movie recommender system was implemented using collaborative systems. After analysing the 2 ways to implement collaborative filtering we decided that ALS is the ideal form of recommending movies.

# **References**

<https://www.datacamp.com/courses/recommendation-engines-in-pyspark>  
Jamen Long, Data Scientist (DataCamp): Building Recommendation Engines with PySpark

<https://link.springer.com/content/pdf/10.1007%2FBF02296972.pdf>  
Forrest W. Young, Jan De Leeuw & Yoshio Takane: “Regression with qualitative and quantitative variables: An alternating least squares method with optimal scaling features” - Psychometrika Vol.41, No. 4; December, 1976.

<https://towardsdatascience.com/intro-to-recommender-system-collaborative-filtering-64a238194a26>

Luo, S. (2018). *Introduction to Recommender System*. [online] Medium.

<https://www.nytimes.com/2009/09/22/technology/internet/22netflix.html>

Lohr, S. (2009). *A $1 Million Research Bargain for Netflix, and Maybe a Model for Others*. [online] Nytimes.com

Colombo-Mendoza L. O., Valencia-Garcıa R., Rodrıguez-Gonzalez A., Alor-Hernandez G., and Samper-Zapater J. J., 2015. "RecomMetz: A context-aware knowledge-based mobile recommender system for movie showtimes". *Expert Systems with Applications,* Volume 42, p. 1202–1222.

U. Shardanand and P. Maes., 1995. "Social information filtering: algorithms for automating “word of mouth". *Proceedings of the SIGCHI conference on Human factors in computing systems,* p. 210–217.

N. Hariri, B. Mobasher, and R. Burke., 2012. "Context-aware music recommendation based on latent topic sequential patterns". *Proceedings of the sixth ACM conference on Recommender systems,* p. 131–138.

Panniello U., Tuzhilin A., and Gorgoglione M., 2014. "Comparing context-aware recommender systems in terms of accuracy and diversity". *User Modeling and User-Adapted Interaction,* Volume 24, p. 35–65.

R. Torres, S. M. McNee, M. Abel, J. A. Konstan, and J. Riedl., 2004. "Enhancing digital libraries with TechLens," in Digital Libraries. *Proceedings of the 2004 Joint ACM/IEEE Conference,* p. pp. 228–236. .

S. M. McNee, I. Albert, D. Cosley, P. Gopalkrishnan, S. K. Lam, A. M. Rashid, et al., 2002. "On the recommending of citations for research papers". *Proceedings of the 2002 ACM conference on Computer supported cooperative work,* p. 116–125.

# Appendix

Below is the pseudocode for the program:

|  |
| --- |
| ratings is DataFrame(open\_csv\_file "ratings.csv")  n\_users is Unique(ratings.userId)'s Count  n\_movies is Unique(ratings.movieId)'s Count  utag is Sorted\_list\_of\_Unique(ratings.userId)  mtag is Sorted\_list\_of\_Unique(ratings.movieId)  rate\_matrix is Zeros\_of\_size(n\_users, n\_movies)  loop through each entry in ratings:  set rate\_matrix[userId, movieId] to rating  end loop  set test\_data to 20% of ratings  set train\_data to 80% of ratings  u\_sim is constructed\_similarity\_matrix\_over\_users  m\_sim is constructed\_similarity\_matrix\_over\_movies  u\_sim is u\_sim - (mean over each user)  m\_sim is m\_sim - (mean over each movie)  function get\_top\_n\_users\_pred  INPUT: training\_data, Similarity, Number of Users(n)  OUTPUT: Prediction over Users    loop over size of training\_data across row  set u\_pred to zeros\_of\_size(training\_data.shape)  get top\_n\_users by slicing  loop over size of training\_data across column  set u\_pred to similarity \* training\_data / sum\_of\_similarity\_over\_top\_n\_users  end loop  end loop    function get\_top\_n\_movies\_pred  INPUT: training\_data, Similarity, Number of Movies(n)  OUTPUT: Prediction over Movies    set u\_pred to zeros\_of\_size(training\_data.shape)  loop over size of training\_data across column  get top\_n\_movies by slicing  loop over size of training\_data across column  set u\_pred to similarity \* training\_data / sum\_of\_similarity\_over\_top\_n\_movies  end loop  end loop    set possible\_n\_values to [25, 50, 100, 200]  loop over possible\_n\_values  get u\_pred by calling get\_top\_n\_users\_pred  get m\_pred by calling get\_top\_n\_movies\_pred    set u\_train to Minimum\_Square\_Error(u\_pred, training\_data)  set m\_train to Minimum\_Square\_Error(m\_pred, training\_data)    set u\_test to Minimum\_Square\_Error(u\_pred, testing\_data)  set m\_test to Minimum\_Square\_Error(m\_pred, testing\_data)    Append all values to their respective lists  end loop  print MSE of all values and their graphs |