# Exploration of various Movie Recommendation Systems


| Divakar sai | Dipen Anjan | Nikhil Nandigama | Dittu Akanksh | Nikhil Injarapu |
|---|---|---|---|---|
| 210050143 | 210050079 | 210050105 | 210050079 | 210050069 |



***Abstract*—In this project we explore diverse recommendation models and conducts an in-depth analysis of movie suggestions generated by these models. The comparison of these models is based on the Root Mean Square Error (RMSE) loss metric, offering a quantitative assessment of their predictive accuracy. The study not only delves into the intricacies of each recommendation model but also seeks to identify patterns in the recommended movies.**


## I. Introduction

In the vast landscape of digital entertainment, where an abundance of movies awaits viewers, the role of recommendation systems emerges as a guiding force, helping users navigate through the sea of options. Movie recommendation systems are integral components of streaming platforms, leveraging advanced algorithms to predict user preferences and provide tailored suggestions. These systems not only enhance user experience but also contribute significantly to content discoverability. As viewers seek personalized and relevant movie recommendations, the intricacies of these recommendation systems become increasingly crucial.

## II. Models

The basic and the improvised models which you will go through this project are the following

(A) Matrix factorization
(B) Matrix factorization with affiliations
(C) Deep Learning model
(D) KNN algorithm
(E) KNN improvised algorithm
(F) Study about GNN models

### A. Matrix factorization

Matrix factorization for movie recommendation is an algorithm that leverages latent features to predict user ratings in a collaborative filtering framework. By optimizing user and item matrices through mini batch gradient descent, the model learns to generate personalized recommendations based on past user-item interactions. Given below is the high level algorithm for finding the optimum matices

**Input:**

- $R$: User-item matrix where $R_{ij}$ is the rating given by user $i$ to item $j$.
- $K$: Number of latent features.

**Output:**

- Matrices $P$ and $Q$ representing the factorized user and item matrices.

**Initialization:**

- Randomly initialize user matrix $P$ and item matrix $Q$.
- Set the learning rate ($\alpha$), regularization parameter ($\beta$), and number of iterations.

**Optimization:**

1) For each $(i, j)$ in ratings:

$$\hat{r}_{ij} = f(P_i, Q_j)$$

$$E = \sum_{(i,j)\in \text{ratings}} (r_{ij} - \hat{r}_{ij})^2 + \beta \left( \|P\|^2 + \|Q\|^2 \right)$$

2) Update user and item matrices using mini batch gradient descent:

$$P_i = P_i + \alpha \left( \frac{\partial E}{\partial P_i} - \beta P_i \right)$$

$$Q_j = Q_j + \alpha \left( \frac{\partial E}{\partial Q_j} - \beta Q_j \right)$$

3) Repeat the optimization process for a predefined number of iterations or until convergence.

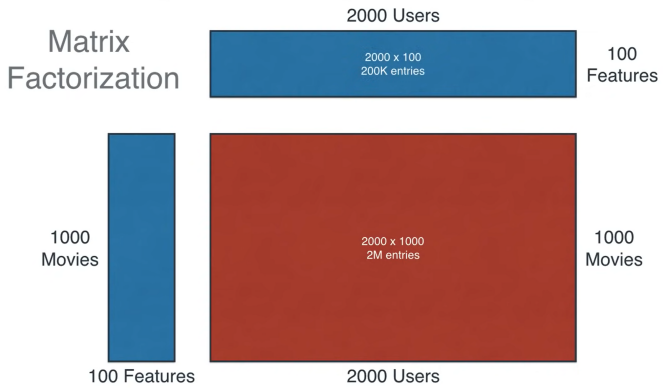
**Prediction:**

$$\hat{r}_{ij} = f(P_i, Q_j)$$

### B. Matrix factorization with affiliations

In the above part we randomly initialized the matrices P and Q but for improvement in performance we can initialize matrix P using the user information such as age, gender, occupation , etc. Similarly for initializing matrix Q we can use movie information such as Genre,

release date, duration ,etc.




***Results using above Matrix factorization:*** We generated predictions for the top 10 recommendations for user Id 4 using the matrix factorization algorithm. The function $f$ based on dot procuct, was utilized and we experimented by varying embedding sizes. The following data represents the recommendations provided by our system using embedding sizes of 10,100,200

**MF old with embed size = 10 and epoch 60**
Safe Passage (1994) 10.030209
Some Mother's Son (1996) 6.359978
Santa with Muscles (1996) 6.3316317
Entertaining Angels (1996) 6.3235936
Great Day in Harlem, A (1994) 6.3084235
Someone Else's America (1995) 6.2757297
The Deadly Cure (1996) 6.264159
Pather Panchali (1955) 6.208237
Saint of Fort Washington, The (1993) 6.072458
Spanish Prisoner, The (1997) 6.043453

**MF old with embed size = 100 with epoch 60**
Safe Passage (1994) 8.35602
Boys, Les (1997) 6.619635
Someone Else's America (1995) 6.6014967
Great Day in Harlem, A (1994) 6.497389
Pather Panchali (1955) 6.4865656
Some Mother's Son (1996) 6.478733
Saint of Fort Washington, The (1993) 6.42613
Anna (1996) 6.353144
Santa with Muscles (1996) 6.100077
The Deadly Cure (1996) 6.0461483

There are certain points that deviate from the recommended criteria, appearing only for one embed size and not across all. These inconsistencies can be eliminated, and we can focus on providing recommendations based solely on the consistent points. The ratings obtained surpass 5 because we did not impose a restriction limiting ratings exclusively to that value.When employing non-linear models with the sigmoid function as the final step, we have constrained this value to a maximum of 5.

**MF old with embed size = 200 with epoch 60**
Faust (1994) 6.2579846
Some Mother's Son (1996) 5.9840093
Santa with Muscles (1996) 5.9502373
Pather Panchali (1955) 5.889647
Fille seule, La (A Single Girl) (1995) 5.8645086
Grosse Fatigue (1994) 5.8046007
Kaspar Hauser (1993) 5.760988
Saint of Fort Washington (1993) 5.6737504
The Deadly Cure (1996) 5.630692
Three Things About her (1966) 5.6281657

As evident from the three recommendations above, larger embeddings have the potential to enable the model to represent users and items more expressively. This allows the model to capture finer details and preferences, as indicated by the observed decrease in the maximum predicted rating or the predicted rating range.

***Results obtained by varying function "$f$":*** The data below illustrates recommendations generated by our system using various functions, including dot product, non-linear models, polynomials, and others.

**1) Dot product :**

**MF with dot product function**
Safe Passage (1994) 8.35602
Boys, Les (1997) 6.619635
Someone Else's America (1995) 6.6014967
Great Day in Harlem, A (1994) 6.497389
Pather Panchali (1955) 6.4865656
Some Mother's Son (1996) 6.478733
Saint of Fort Washington (1993) 6.4261327
Anna (1996) 6.353144
Santa with Muscles (1996) 6.100077
The Deadly Cure (1996) 6.0461483

**2) Non linear architecture :**

**MF with non-linear architecture**
Sliding Doors (1998) 4.99
Brothers in Trouble (1995) 4.99
Some Mother's Son (1996) 4.99
Spanish Prisoner, The (1997) 4.99
Butcher Boy, The (1998) 4.99
Butcher Boy, The (1998) 4.99
Bitter Sugar (Azucar Amargo) (1996) 4.99
American Dream (1990) 4.99
Santa with Muscles (1996) 4.99
Hugo Pool (1997) 4.99

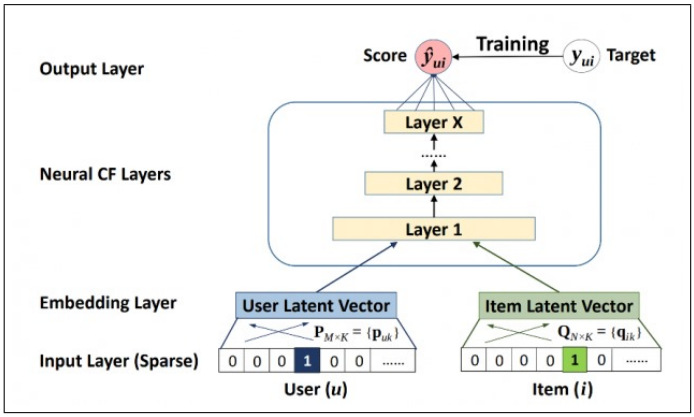
**3) Polynomial**

**MF with polynomial function**
Boy's Life 2 (1997) 5.77244
Pather Panchali (1955) 5.6470733
Braindead (1992) 5.6064134
Paths of Glory (1957) 5.5410275
In the Company of Men (1997) 5.525915
Bad Taste (1987) 5.503521
Jude (1996) 5.4075465
Red , Green Firecracker (1994) 5.3921475
Aparajito (1956) 5.3711925
Persuasion (1995) 5.3162622

The marginal improvement in test error observed when transitioning from a linear to a non-linear function could be attributed to the relatively small size of the dataset or potential randomness within the data that our system may struggle to interpret.*Pather panchali, Some mother's son ,Santa with muscles* are the consistent recommendations for user ID 4.

### *C. Deep learning model*

Neural Networks (NNs) have revolutionized recommendation systems by offering a dynamic and powerful approach to understanding user preferences. In contrast to traditional methods, NNs excel at learning intricate patterns and relationships from vast user-item interaction data. Their ability to capture complex, non-linear interactions enables more accurate predictions, making them a cornerstone in crafting personalized and engaging user experiences across various online platforms.




**Input:**

- $R$: User-item matrix where $R_{ij}$ is the rating given by user $i$ to item $j$.
- $U$: Number of users.
- $I$: Number of items (movies).
- $K$: Number of latent features.
- $E$: Number of epochs for training.
- $\alpha$: Learning rate.

**Neural Network Architecture:**

1) **Embedding Layer:**
   - Embedding layer for users: $U \times K$ matrix.
   - Embedding layer for items: $I \times K$ matrix.
2) **Concatenation Layer:**
   - Concatenate user and item embeddings.
3) **Feedforward Neural Network:**
   - One or more fully connected layers with ReLU activation functions.
4) **Output Layer:**
   - Single output node with a linear activation function (for regression tasks).

**Algorithm:**

1) **Initialization:**
   - Initialize user,item embeddings randomly.
   - Define the neural network architecture.
2) **Training:**
   - For each epoch $e$ in $E$: For each user-item pair $(i, j)$ with a known rating:
     a) Compute the predicted rating $\hat{r}_{ij}$ using the neural network.
     b) Compute the mean squared error loss: $L = (r_{ij} - \hat{r}_{ij})^2$.
     c) Update model parameters using back propagation and mini batch gradient descent:

New parameters = Old parameters - $\alpha \times$ Grad

3) **Prediction:**
   - After training, the model can predict unknown ratings for user-item pairs.

***Results using Neural Networks :***

**Neural Networks with 2 layers**
Safe Passage (1994) 4.878627
Someone Else's America (1995) 4.8498297
Great Day in Harlem, A (1994) 4.847664
Pather Panchali (1955) 4.846305
Santa with Muscles (1996) 4.8421774
Entertaining Angels (1996) 4.8415327
Saint of Fort Washington, The (1993) 4.840782
Some Mother's Son (1996) 4.8392386
The Deadly Cure (1996) 4.837982
Bitter Sugar (Azucar Amargo) (1996) 4.832143

**NeuralNetworks with 4 layers**
Santa with Muscles (1996) 4.8839583
Someone Else's America (1995) 4.882688
Pather Panchali (1955) 4.879611
Safe Passage (1994) 4.879333
Great Day in Harlem, A (1994) 4.8788643
Saint of Fort Washington, The (1993) 4.8787336
The Deadly Cure (1996) 4.876513
Entertaining Angels (1996) 4.8714266
Bitter Sugar (Azucar Amargo) (1996) 4.8678055
Some Mother's Son (1996) 4.8674407

As evident from the preceding two results, increasing the number of layers, and consequently, augmenting the model's complexity, does not yield significant changes in the outcomes. The recommendations remain consistent across both scenarios, indicating that the incremental layering has limited impact on the results.

### *D. KNN algorithm*

This approach is straight forward.To predict the rating between a specific user and a given movie

- Calculate Similarity Scores:
  - For a specific movie, compute the cosine similarity between that movie and all other unseen movies in the dataset.
  - The similarity score reflects how closely related each unseen movie is to the specific movie.
- Identify Top-K Similar Movies:
  - Determine the top-k movies with the highest similarity scores.
  - These are the movies most similar to the specific movie.
- Weighted Sum Calculation:
  - For each of the top-k similar movies, retrieve the user's ratings.
  - Calculate the weighted sum by multiplying the similarity score of each movie by the user's rating for that movie.
- Final Rating Prediction:
  - Sum up the weighted scores obtained for each top-k similar movie to get the overall weighted sum
  - This total represents the predicted rating for the specific movie by the user.
- Handling Unseen Movies:
  - If a movie hasn't been seen by the user (i.e., the user hasn't rated it), consider its weight as 0 in the calculation.
  - This ensures that unseen movies do not contribute to the weighted sum, as there is no user rating available for them.

**Recommended movies by Basic KNN**
B. Monkey (1998) 5.00
Emma (1996) 5.00
Face/Off (1997) 5.00
Good Will Hunting (1997) 5.00
Mat' i syn (1997) 5.00
Mrs. Brown (Her Majesty, Mrs. Brown) (1997) 5.00
My Best Friend's Wedding (1997) 5.00
That Thing You Do! (1996) 5.00
Thousand Acres, A (1997) 5.00
Anna Karenina (1997) 5.00

***Drawbacks:*** If a user rates a single similar movie as 5, this KNN algorithm often predicts the same high rating for all other unseen movies. This results in a substantial number of movies being inaccurately assigned a rating of 5, leading to an inflation of ratings in datasets with a large movie inventory.When a user hasn't watched any movies similar to the target movie, all corresponding weights become zero in the collaborative filtering approach, making it impossible to predict ratings for unseen movies.

### E. KNN improvised algorithm

This algorithm represents an enhancement over the previous one, focusing primarily on individual user preferences rather than user-user interactions or item-item interactions. It predicts ratings based solely on the movies watched by the user, disregarding interactions with all other items. The step-by-step process for implementing this algorithm is as follows:

1) **Define Rating Matrix:**
   - Create a matrix containing ratings for all movies. Unpredicted ratings are represented as NaN (Not a Number).
2) **Normalize Data:**
   - Normalize the ratings data such that the mean rating for every user is 0.
   - Set unpredicted rating values to 0 after normalization.
3) **Feature Vector for Movie:**
   - Define the feature vector for a specific movie as the row inside the matrix corresponding to that movie.
4) **User-Specific Training Data:**
   - For a specific user, gather all movies that the user has seen into a separate matrix ($X_{\text{train}}$).
   - The feature vectors for these seen movies are used as the training features ($X_{\text{train}}$).
   - The corresponding ratings given by the user for these seen movies are collected as the target values ($y_{\text{train}}$).
5) **Train k-NN Classifier:**
   - Use the collected $X_{\text{train}}$ and $y_{\text{train}}$ data to train a k-NN (k-Nearest Neighbors) classifier.
   - The classifier learns the patterns and relationships between the feature vectors of movies and the target values (user ratings).
6) **Prediction Process:**
   - To predict the rating for a specific user and a movie:
   - Retrieve the feature vector for the specific movie from the matrix.
   - Substitute this feature vector into the trained k-NN classifier.
   - The output from the classifier is the predicted rating for that movie by the user.

Through hyperparameter tuning, we systematically optimized the value of k to achieve the lowest possible test loss, ensuring superior performance of the model.

**Results using KNN improvised:**

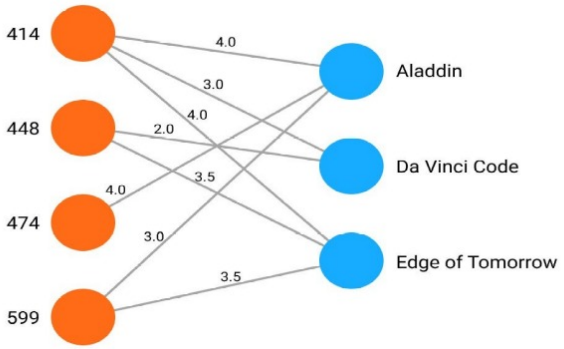
**Recommended movies by KNN improvised**
Mighty, The (1998) 5.0
Grosse Fatigue (1994) 5.0
Last Klezmer (1995) 5.0
Big Blue, The (Grand bleu, Le) (1988) 5.0
Something to Talk About (1995) 5.0
Somebody to Love (1994) 5.0
Star Maker, The (Uomo delle stelle, L') (1995) 5.0
Two Deaths (1995) 5.0
Stefano Quantestorie (1993) 5.0
Crude Oasis, The (1995) 5.0

In KNN, each user is assigned a distinct model, enhancing performance and resulting in unique predictions. This contrasts with other models, where a single model is applied to the entire dataset, yielding different prediction outcomes.

### F. GNN models

Graph Neural Networks operate by exchanging messages among nodes through their edges and aggregating these messages to update each node. This iterative process enables nodes to gather information from distant neighbors and deduce structural features in their vicinity.

The core concept of GNNs involves utilizing a node's local surroundings to form a computational graph for generating node embeddings. Consequently, the model must learn to propagate and transform information within this graph to create the node's feature representation. These GNNs are constructed by stacking layers of message and aggregation functions, with each layer enabling nodes to acquire information from neighbors located an additional hop away.




In a movie recommender system, users and movies are

treated as graph nodes, where edge weights represent user ratings. This configuration forms a bipartite graph.

User node embeddings capture the ratings they would assign to movies, while item embeddings encapsulate the ratings all users would assign to a movie.

GNNs identify user nodes sharing interest in similar items, allowing them to assimilate information from other user nodes to refine their own embeddings. Similarly, item nodes examine users' interests and explore other items these users favor to enhance their item embeddings. optimizing the recommendation scoring function.

We attempted to implement this method based on an online resource. However, we couldnt complete this due to incomplete code and implementation details provided in the source material.

## III. Conclusion

- Linear matrix factorization outperformed non-linear and polynomial transformations on movie embeddings, this maybe due to the linear nature of user-movie interactions and the small dataset size,but in real world, the Non linear performs better than the linear and polynomial.
- Despite generating embeddings for users' personal data, no test loss improvement was observed, attributed to limited user similarities.
- Utilizing distinct X_train and y_train datasets for each user in KNN, resulting in varying models per user, led to improved performance compared to matrix factorization and neural networks, which employ a single model for the entire dataset
- Neural networks outperform matrix factorization by capturing non-linear user-movie interactions through activation functions like ReLU.
- For all models except KNN, we employed the RMSE loss as a performance metric, attaining a test loss close to 0.9 across all the models

## References


[1] Mykhaylo Schwarz, Mykhaylo Lobur, Yuriy Stekh, Analysis of the Effectiveness of Similarity Measures for Recommender Systems, 978-1-5090-5045-1/17/ ©2017 IEEE M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.

[2] Jochen Nessel, Barbara Cimpa, The MovieOracle - Content Based Movie Recommendations, 978-0-7695- 4513-4/11 © 2011 IEEE.

[3] H. Wang, N. Wang, and D.-Y. Yeung. Collaborative deep learning for recommender systems. In Proc. KDD, pages 1235–1244, 2015.

[4] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research, 12:2121–2159, July 2011.

[5] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. Wide & deep learning for recommender systems. CoRR, abs/1606.07792, 2016.

[6] Hao Wang, Naiyan Wang, and Dit-Yan Yeung. Collaborative deep learning for recommender systems. In ACM SIGKDD, pages 1235–1244, 2015.