

# Report

Gia Trong Nguyen, B20-AI. g.nguyen@innopolis.university

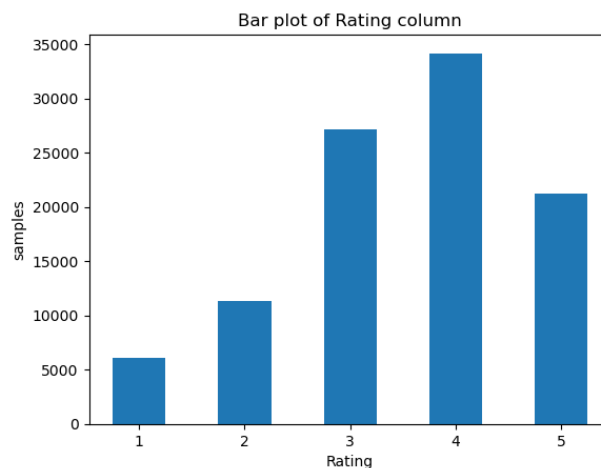
## Introduction

Recommendation systems are algorithms designed to predict and suggest items (such as products, movies, music, etc.) that a user might be interested in, based on their preferences or behavior. These systems are widely used in various applications like e-commerce platforms, streaming services, social media, and more. In this work I solve a recommendation task for MovieLens 100K dataset. To get more information about the task, please check the *Task Description.md* file provided by organizers. To solve the problem I tried two approaches: Deep Neural Networks and classical ML. For advanced MLI take the idea provided by Google inc. from the article *Wide & Deep Learning for Recommender Systems*, where users and items metadata are used. And as a classical ML I took SVD.

## Data Analysis

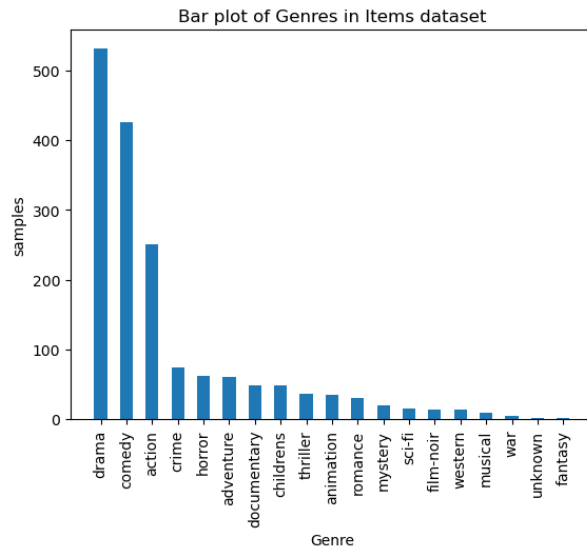
Data analysis part is provided in *1.0-initial-data-exploration.ipynb* file. From here I got 1682 unique movies and 943 unique users.

Firstly, I looked at the distribution of ratings for movies given by users. So most of the ratings is 3, 4, and 5, let them be positive ratings.

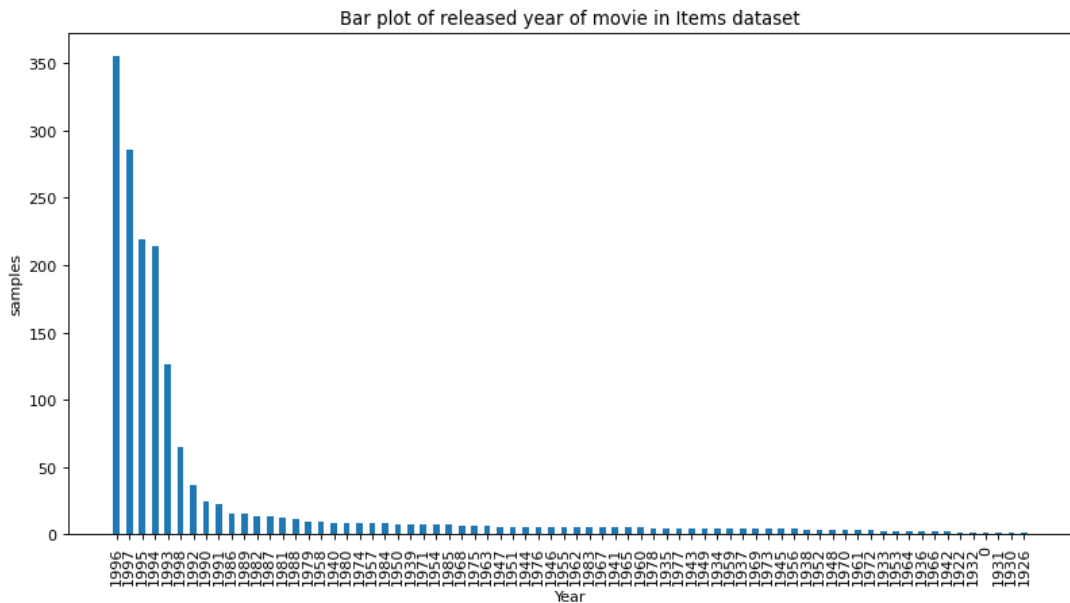


Then I check items metadata:

- Dataset provides movies with 19 different genres. Most of the movies are dramas, comedies and action genres.

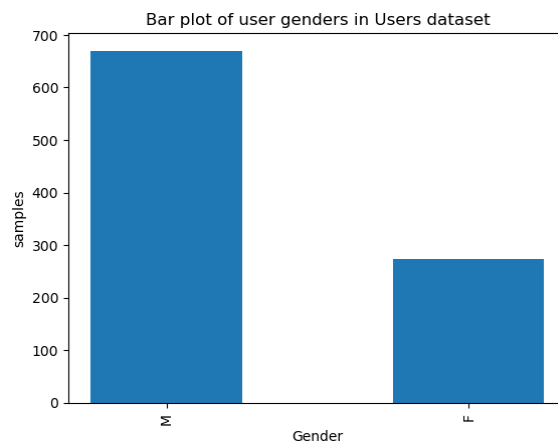


- Most of the movies were created in about 1990-1998. Also some of movies haven't information about date (marked as 0 in the figure)

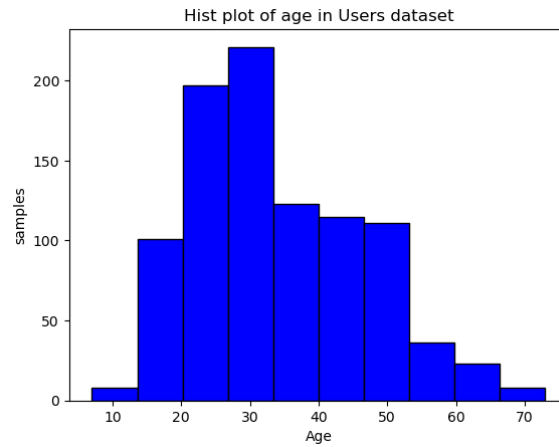


After I checked users metadata:

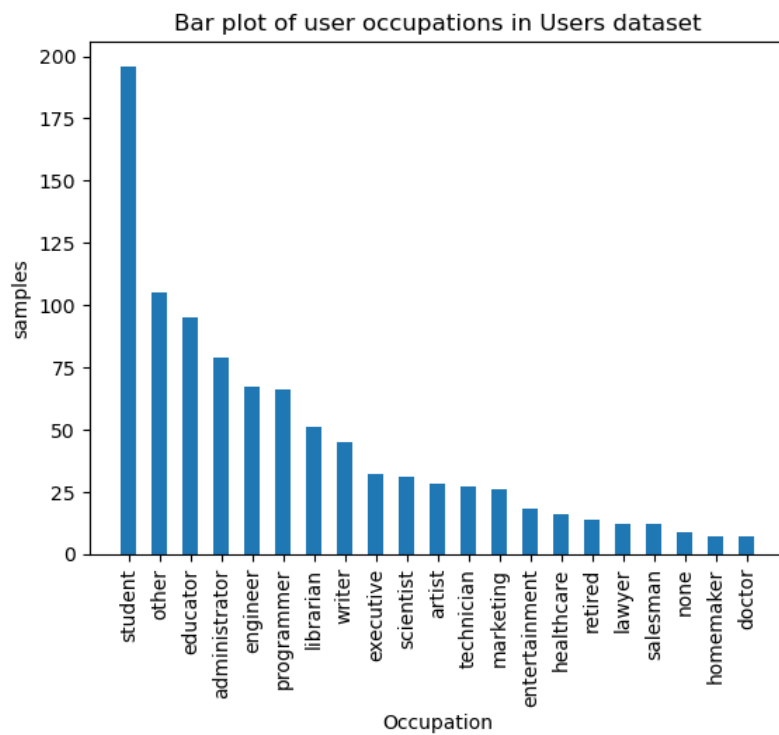
- There are 2 times more men than women.



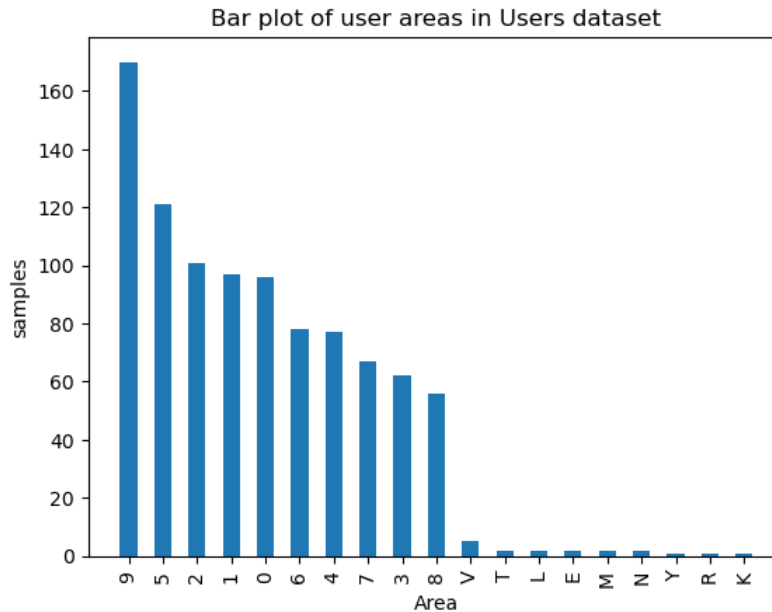
- Most users are in the range from 20 to 50.



- Most users are students.



- distribution of areas, where they live from *zip\_code* column.

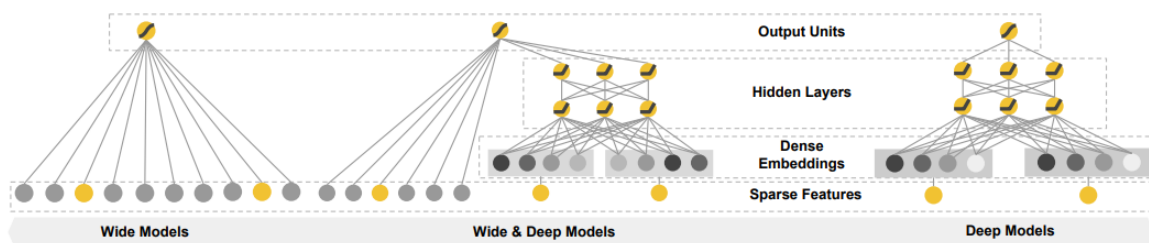


## Model implementation

### Wide and Deep Neural Network

I tried the advanced approach using neural networks proposed by Google inc. from the article *Wide & Deep Learning for Recommender Systems*. It is a supervised learning neural network architecture that combines a wide model and a deep model into one single architecture. The broad model takes in cross-product categorical features as inputs, and it is capable of memorizing the relationship between feature inputs and the dependent variable. On the other hand, the deep model takes in numerical and categorical features as inputs, passes it through multiple layers of neurons, and it is great at generalizing the relationship between feature inputs and the dependent variable.

The wide and deep network are then combined at the end before passing it through an activation function to obtain the prediction. By combining these two models, we are able to enable both memorization and generalization within a single network.



Model was implemented using the PyTorch library. Model architecture that I created looks like this.

```

WideAndDeepModel(
  (emb_1): Embedding(50, 64)
  (emb_2): Embedding(50, 64)
  (emb_3): Embedding(50, 64)
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (dropout): Dropout(p=0.2, inplace=False)
  (deep_layers): Sequential(
    (0): Linear(in_features=193, out_features=256, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.2, inplace=False)
    (3): Linear(in_features=256, out_features=128, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.2, inplace=False)
    (6): Linear(in_features=128, out_features=64, bias=True)
    (7): ReLU()
    (8): Dropout(p=0.2, inplace=False)
    (9): Linear(in_features=64, out_features=64, bias=True)
    (10): ReLU()
  )
  (wide_inputs): Linear(in_features=41, out_features=41, bias=True)
  (output): Sequential(
    (0): Linear(in_features=105, out_features=1, bias=True)
    (1): ReLU()
  )
)

```

## SVD

For the Singular value decomposition I used a *surprise* package. SVD is a matrix factorization technique used in recommendation systems. It's a classical method employed for collaborative filtering in recommender systems, particularly in the context of matrix factorization. In the context of recommendation systems, the basic idea behind SVD is to decompose the user-item interaction matrix into three matrices:

User matrix (U): Represents users and their latent factors or preferences.

Item matrix ( $\Sigma$ ): Represents items and their latent factors or attributes.

Transpose of item matrix ( $V^T$ ): Another item matrix, used to capture complementary information from items.

The decomposition aims to approximate the original user-item matrix by the product of these matrices:

$A \approx U \times \Sigma \times V^T$ , where:

A is the original user-item matrix (usually sparse, with users as rows and items as columns).

U represents users and their preferences in a latent space.

$\Sigma$  is a diagonal matrix that contains the singular values, indicating the importance of each latent factor.

$V^T$  captures item characteristics or attributes in a latent space.

Once the matrices U,  $\Sigma$  and  $V^T$  are derived through SVD, the reconstruction of the user-item matrix can be used for making recommendations. By predicting unknown entries (missing values) in the user-item matrix, recommendations for users can be made based on the predicted ratings for items they haven't interacted with yet. SVD helps in reducing the dimensionality of the user-item matrix by capturing the most important latent factors that contribute to the user-item interactions.

# Model Advantages and Disadvantages

## Wide and Deep Neural Network

Advantages:

- Combining Memorization and Generalization: Wide & Deep models (introduced by Google) blend the strengths of memorization (capturing frequent co-occurrences) and generalization (learning from less frequent but potentially valuable patterns), enhancing the recommendation quality.
- Handling Sparse Data: They handle sparse and categorical data effectively by incorporating both wide linear models (for memorization of specific user-item interactions) and deep neural networks (for learning latent representations).
- Capturing Complex Patterns: The deep component can capture intricate patterns and non-linear relationships between user-item features, allowing for better representation learning and more accurate recommendations.
- Interpretability and Performance: The wide component provides interpretability by explicitly modeling feature interactions, while the deep component improves performance by learning intricate feature interactions automatically.

Disadvantages:

- Complexity and Training Time: Developing and training Wide & Deep models can be complex, especially when dealing with large-scale datasets, requiring substantial computational resources and longer training times.
- Data Preprocessing: Handling categorical features and encoding them appropriately for the wide component might require significant preprocessing efforts to manage categorical data effectively.

## SVD

Advantages:

- Dimensionality Reduction: SVD helps in reducing the dimensionality of the user-item interaction matrix by capturing the most significant latent factors, which can lead to more efficient computations and better scalability.
- Improved Recommendations: By capturing latent factors, SVD can provide more accurate recommendations by identifying underlying patterns in user-item interactions that might not be easily discernible otherwise.

- Interpretability: SVD can offer some level of interpretability as it provides factors that represent latent user preferences and item attributes, making it possible to understand why certain recommendations are made.

Disadvantages:

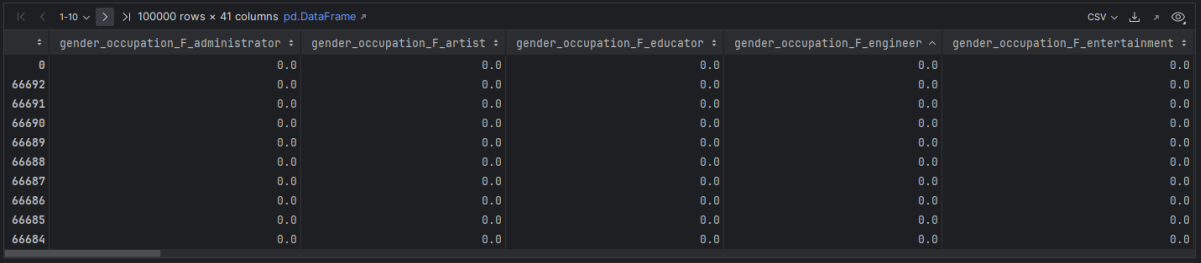
- Sparsity Handling: SVD struggles with handling large-scale sparse matrices efficiently. In real-world scenarios, user-item interaction matrices tend to be very sparse, which can limit its effectiveness.
- Scalability: As the size of the dataset grows, the computational cost of SVD increases significantly, making it less practical for very large datasets or in real-time applications.
- Cold Start for Users/Items: SVD might not perform well for new users or items with very few interactions, as it relies on historical interaction data to derive recommendations.
- Lack of Adaptability: SVD models are static and don't adapt to changes in user preferences or item characteristics over time without retraining, which might lead to less accurate recommendations in dynamic environments.

## Training process

For both methods I divide data into train/test sets by 80% and 20% respectively. Wide and Deep Neural Network show not really good results on test set versus SVD, so I took SVD as the final model.

## Wide and Deep Neural Network

For the wide component, I performed cross-product feature transformation by combining gender and occupation . One-hot encoding was then applied on the cross-product features.



|       | gender_occupation_F_administrator | gender_occupation_F_artist | gender_occupation_F_educator | gender_occupation_F_engineer | gender_occupation_F_entertainment |
|-------|-----------------------------------|----------------------------|------------------------------|------------------------------|-----------------------------------|
| 0     | 0.0                               | 0.0                        | 0.0                          | 0.0                          | 0.0                               |
| 66692 | 0.0                               | 0.0                        | 0.0                          | 0.0                          | 0.0                               |
| 66691 | 0.0                               | 0.0                        | 0.0                          | 0.0                          | 0.0                               |
| 66690 | 0.0                               | 0.0                        | 0.0                          | 0.0                          | 0.0                               |
| 66689 | 0.0                               | 0.0                        | 0.0                          | 0.0                          | 0.0                               |
| 66688 | 0.0                               | 0.0                        | 0.0                          | 0.0                          | 0.0                               |
| 66687 | 0.0                               | 0.0                        | 0.0                          | 0.0                          | 0.0                               |
| 66686 | 0.0                               | 0.0                        | 0.0                          | 0.0                          | 0.0                               |
| 66685 | 0.0                               | 0.0                        | 0.0                          | 0.0                          | 0.0                               |
| 66684 | 0.0                               | 0.0                        | 0.0                          | 0.0                          | 0.0                               |

For the deep component, I used age, gender and occupation features. I combined the sparse genre categorical features into one single genre categorical feature. Label encoding is applied on categorical features while min-max scaling is applied on numerical features.

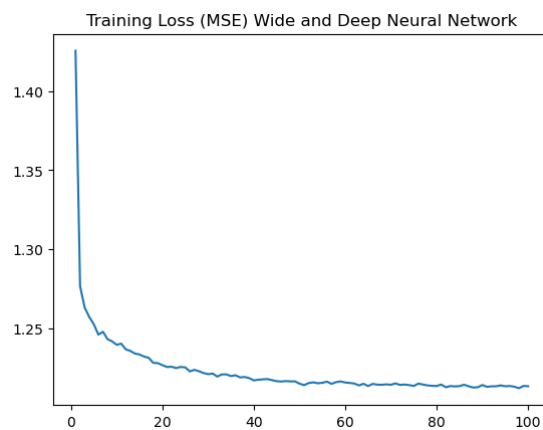
|   | age     | gender | occupation | genre |
|---|---------|--------|------------|-------|
| 0 | 0.80303 | 1      | 15         | 2     |
| 1 | 0.80303 | 1      | 15         | 0     |
| 2 | 0.80303 | 1      | 15         | 5     |
| 3 | 0.80303 | 1      | 15         | 7     |
| 4 | 0.80303 | 1      | 15         | 3     |
| 5 | 0.80303 | 1      | 15         | 7     |
| 6 | 0.80303 | 1      | 15         | 5     |
| 7 | 0.80303 | 1      | 15         | 5     |
| 8 | 0.80303 | 1      | 15         | 7     |
| 9 | 0.80303 | 1      | 15         | 0     |

Some parameters:

```
num_epochs = 100
batch_size = 32
optimizer = optim.Adam(wide_and_deep_model.parameters())
criterion = nn.MSELoss()
```

Hyperparameters for optimizer I took as default from PyTorch.

Training Graphic:



## SVD

Data for training is a dataframe with 'user\_id', 'item\_id', 'rating'.

|   | user_id | movie_id | rating |
|---|---------|----------|--------|
| 0 | 196     | 242      | 3      |
| 1 | 186     | 302      | 3      |
| 2 | 22      | 377      | 1      |
| 3 | 244     | 51       | 2      |
| 4 | 166     | 346      | 1      |

Then I load to surprise.Dataset:



```
reader = Reader(line_format='user item rating', rating_scale=(1, 5))
data = Dataset.load_from_df(df, reader)
```

For training SVD I took default hyperparameters from *surprise* implementation of SVD:

- `n_factors` – The number of factors. Default is 100.
- `n_epochs` – The number of iteration of the SGD procedure. Default is 20.
- `lr_all` – The learning rate for all parameters. Default is 0.005.
- `reg_all` – The regularization term for all parameters. Default is 0.02.

## Evaluation

Since models were trained on 80% of the full dataset, then I need to evaluate them on the remaining 20%. Also I made an additional evaluation for the final model, which is SVD.

### Wide and Deep Neural Network

RMSE on the test set is 1.10 and the MSE on the test set is 1.21. This falls short of the benchmark performance for MovieLens 100k which typically has RMSE in the range of 0.92 to 0.96. I decided to abandon the idea of continuing to work with this model. But on the bright side, it still performs better than a random prediction based on the distribution of the training set, which has an RMSE of 1.52.

### SVD (Final model)

RMSE on the test set shows 0.9352 and MSE shows 0.8745, which is better than the Wide and Deep Neural Network model. Also I calculated Precision@K and Recall@K, where K = 10. Results are 0.9059 and 0.6365 respectively.

### Addition Evaluation

For addition Evaluation process I used `u1.base`, `u1.test`, `u2.base`, `u2.test`, `u3.base`, `u3.test`, `u4.base`, `u4.test`, `u5.base`, `u5.test`, `ua.base`, `ua.test`, `ub.base`, `ub.test` sets. Here I took SVD, train them on `u*.base` and then evaluate them on `u*.test` using MSE, RMSE, Precision@10 and Recall@10.

Evaluation ('u1.base', 'u1.test')

MSE: 0.9142

RMSE: 0.9561

Precision at K: 0.9213

Recall at K: 0.4817

Evaluation ('u2.base', 'u2.test')

MSE: 0.8794

RMSE: 0.9378

Precision at K: 0.9067

Recall at K: 0.5532

Evaluation ('u3.base', 'u3.test')

MSE: 0.8689  
RMSE: 0.9322  
Precision at K: 0.9003  
Recall at K: 0.6242

Evaluation ('u4.base', 'u4.test')  
MSE: 0.8651  
RMSE: 0.9301  
Precision at K: 0.8868  
Recall at K: 0.6411

Evaluation ('u5.base', 'u5.test')  
MSE: 0.8699  
RMSE: 0.9327  
Precision at K: 0.8807  
Recall at K: 0.6607

Evaluation ('ua.base', 'ua.test')  
MSE: 0.9105  
RMSE: 0.9542  
Precision at K: 0.8786  
Recall at K: 0.8685

Evaluation ('ub.base', 'ub.test')  
MSE: 0.9406  
RMSE: 0.9699  
Precision at K: 0.8743  
Recall at K: 0.8725

## Results

In my experiments, SVD performed better for RMSE than Wide and Deep Neural Networks, hence it was chosen as the final model.