The Netflix Prize

RECOMMENDATION SYSTEM

Jan Pogłód, Paulina Kulczyk

The Task:

	Id	Year	Movie
0	1	2003.0	Dinosaur Planet
1	2	2004.0	Isle of Man TT 2004 Review
2	3	1997.0	Character
3	4	1994.0	Paula Abdul's Get Up & Dance
4	5	2004.0	The Rise and Fall of ECW

	Cust_Id	Rating	Date	Movie_Id
1	1488844	3.0	2005-09-06	1
2	822109	5.0	2005-05-13	1
3	885013	4.0	2005-10-19	1
4	30878	4.0	2005-12-26	1
5	823519	3.0	2004-05-03	1

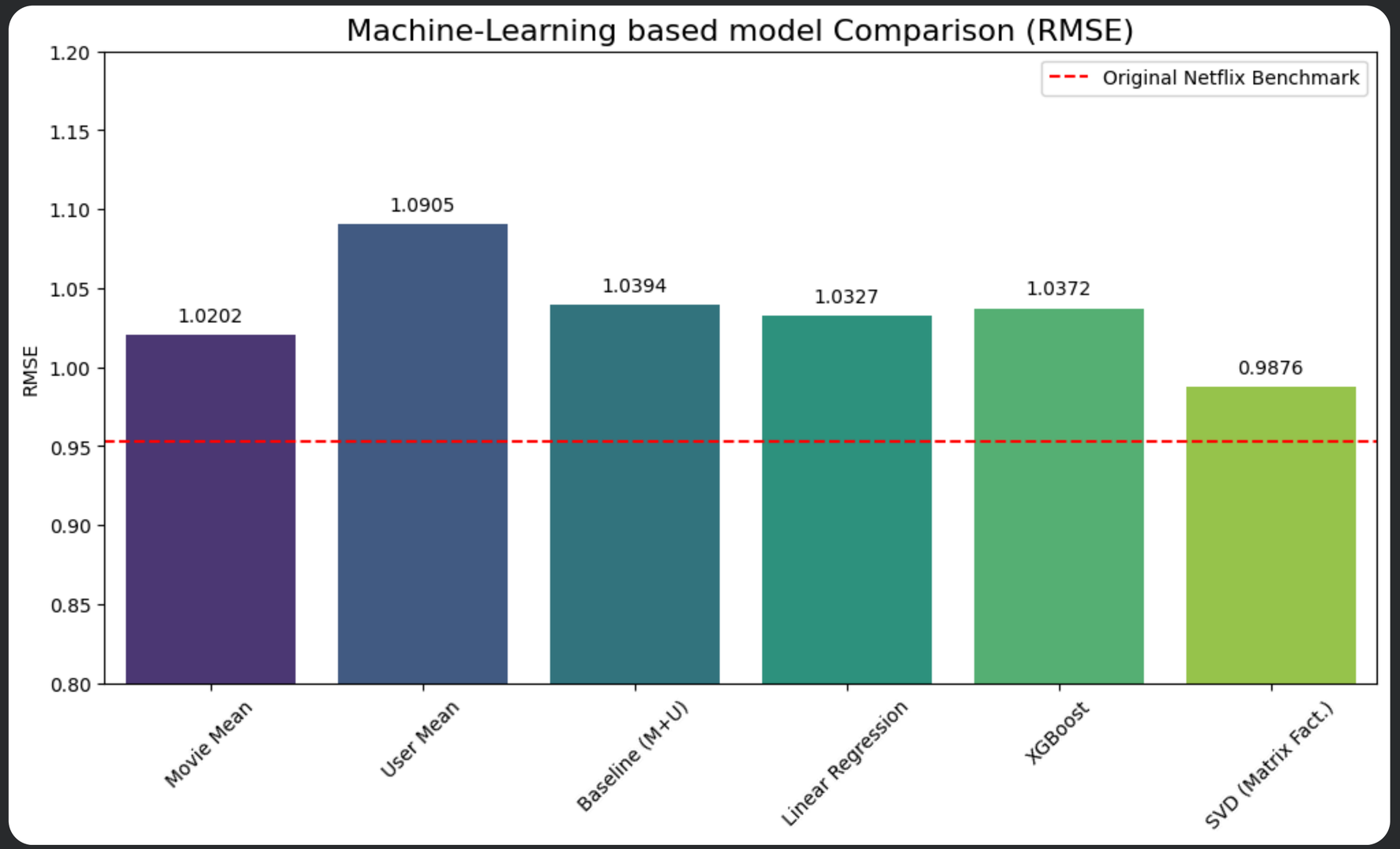
train set: 1 599 711 ratings
test set: 399 928 ratings

Goal: To find the best RMSE value for predicting next user's rating for particular movie

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

ML-based Aproach

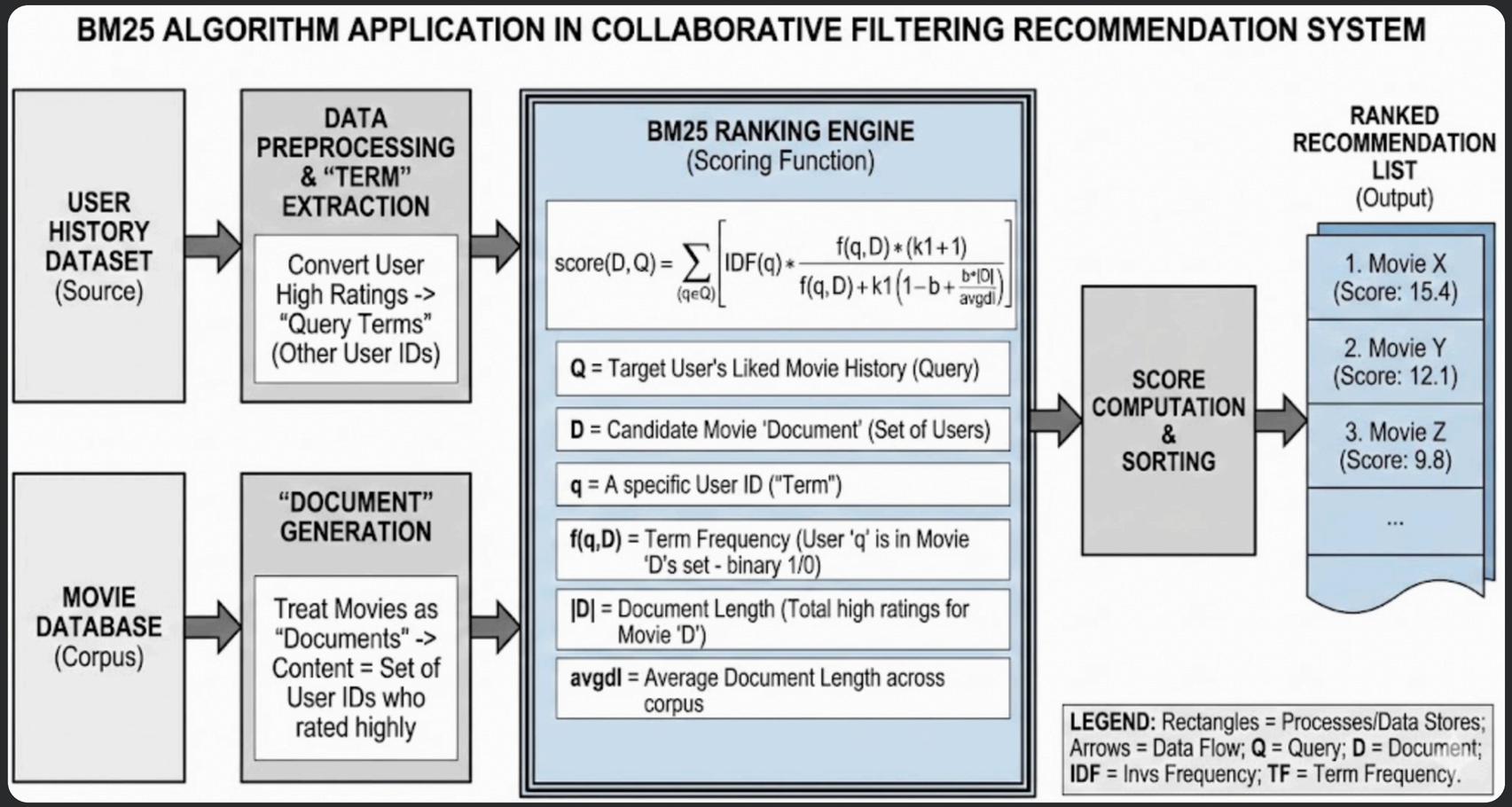
1) Movie avg
2) User avg
3) M + U avg
4) LinearRegression
5) XGBoost
6) SVD



BM25

Methodology:
1) Instead of looking for words in documents, we look for Users in Movies
2) The “content” of the document (movie) is a list of ID numbers of users who rated it highly.
3) Query = history of user: which movies did he rated and how high?

RMSE: 1.0325



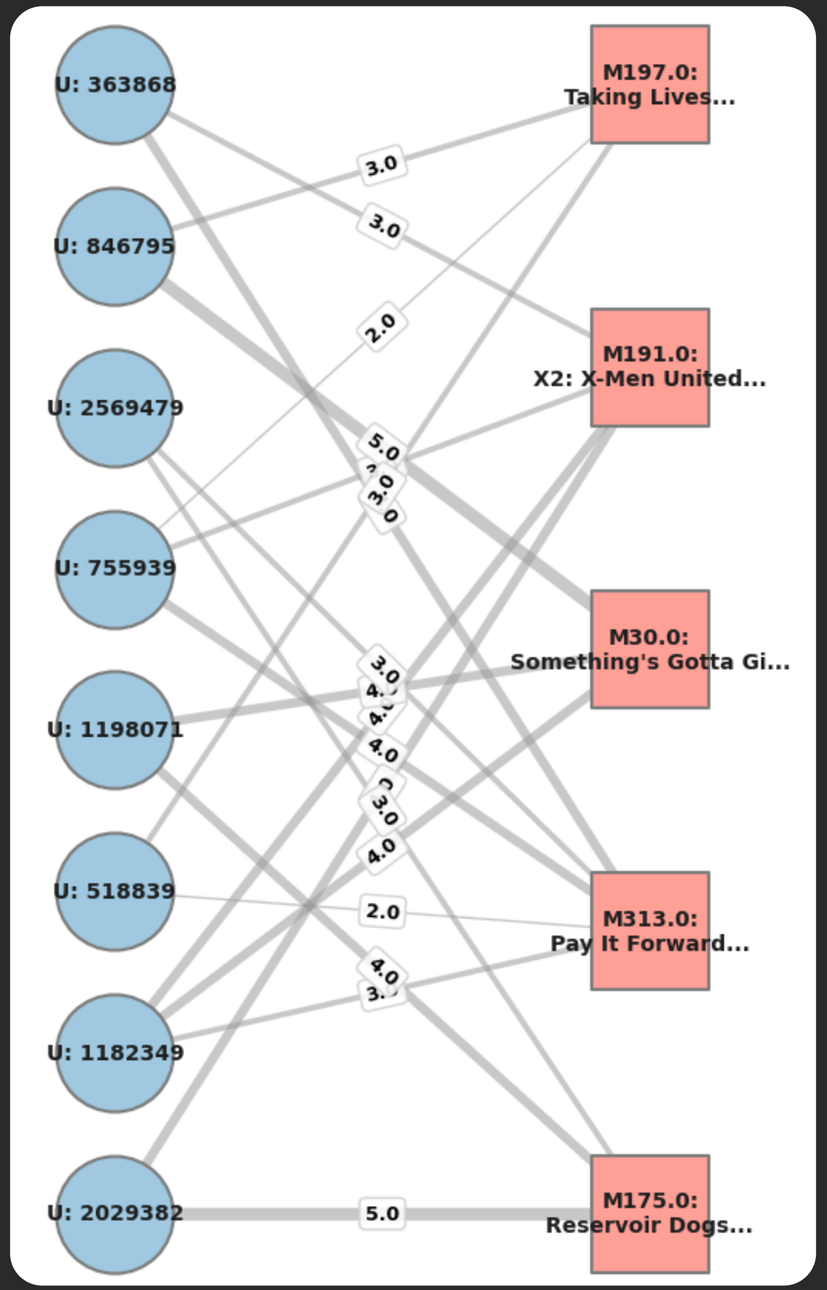
Goal: The algorithm evaluates how well a given movie (Document D) matches the user's taste (Query Q).

- Strong BM25 advantage: BM25 ensures that the system does not only recommend the most popular hits, but promotes films that are actually relevant, applying a penalty for too many random ratings

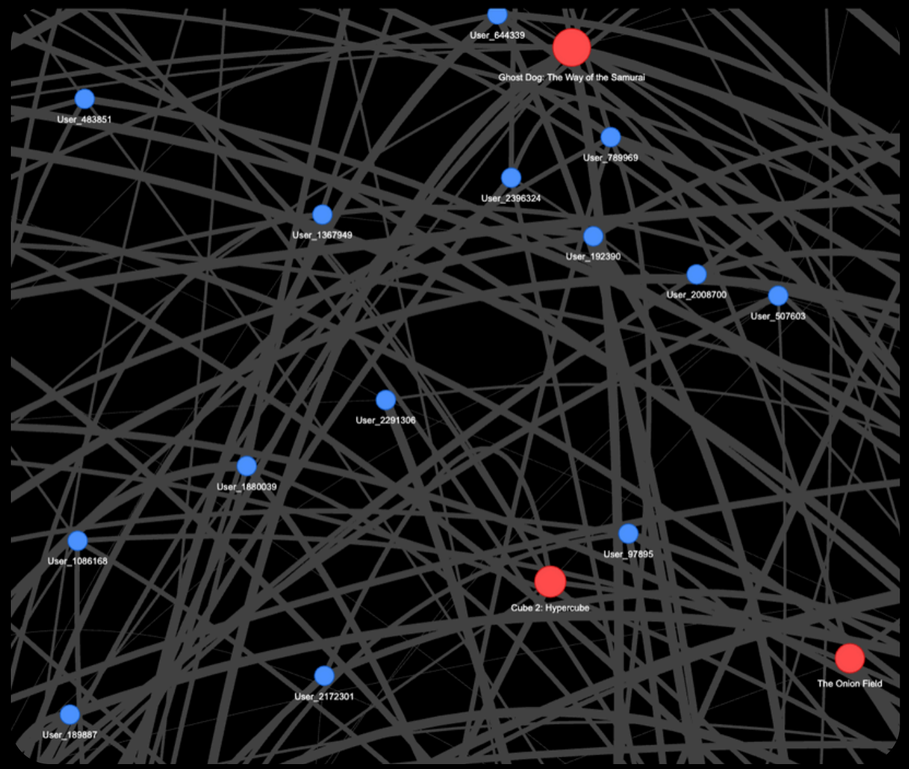
Graph-Based Aproach

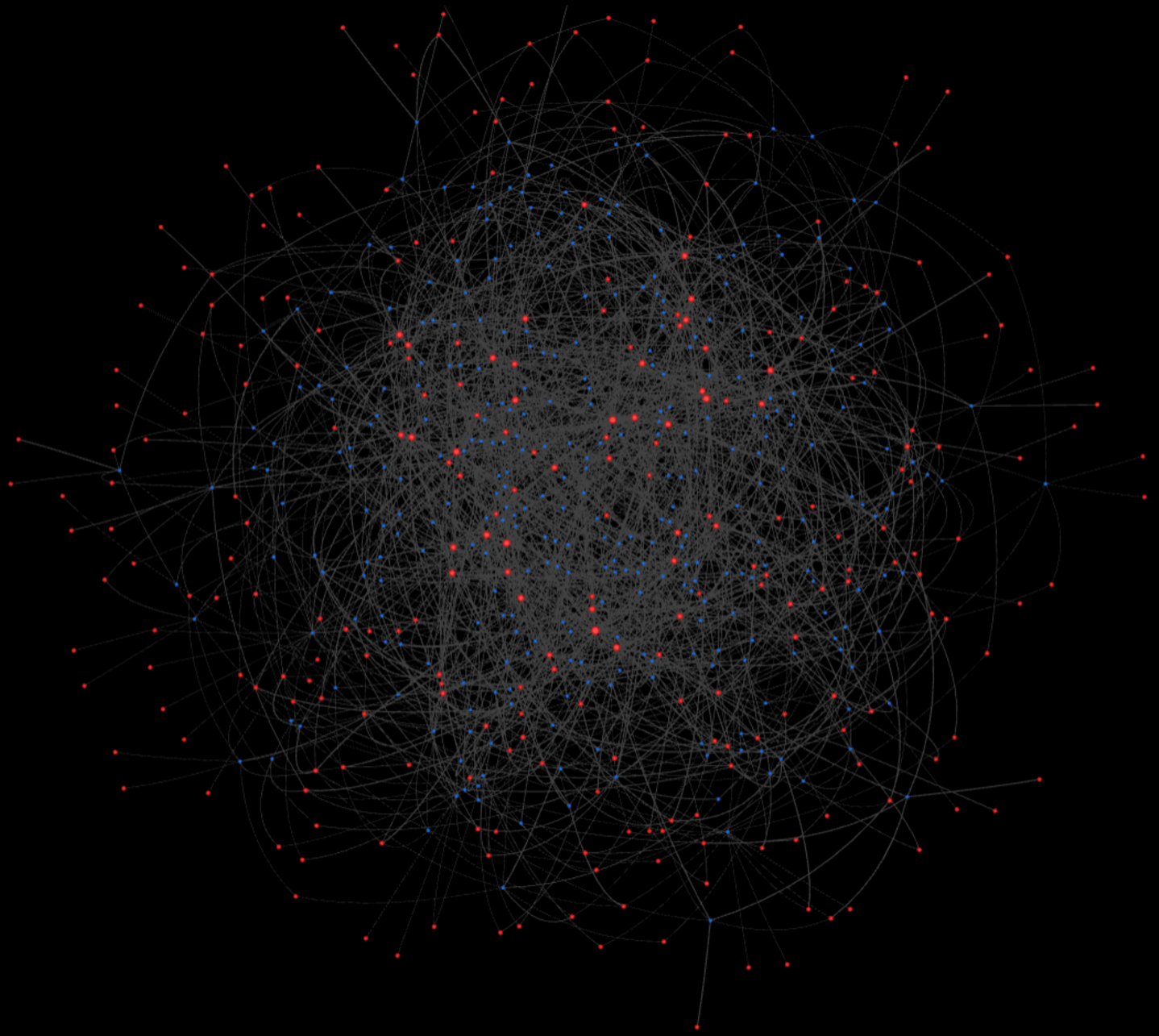
Methodology:
- 323 473 users,
- 361 movies,
- 3 199 422 edges.

1) We represent ratings as edges (1-5).
2) Based on Users and Movies we create the Bipartite Graph

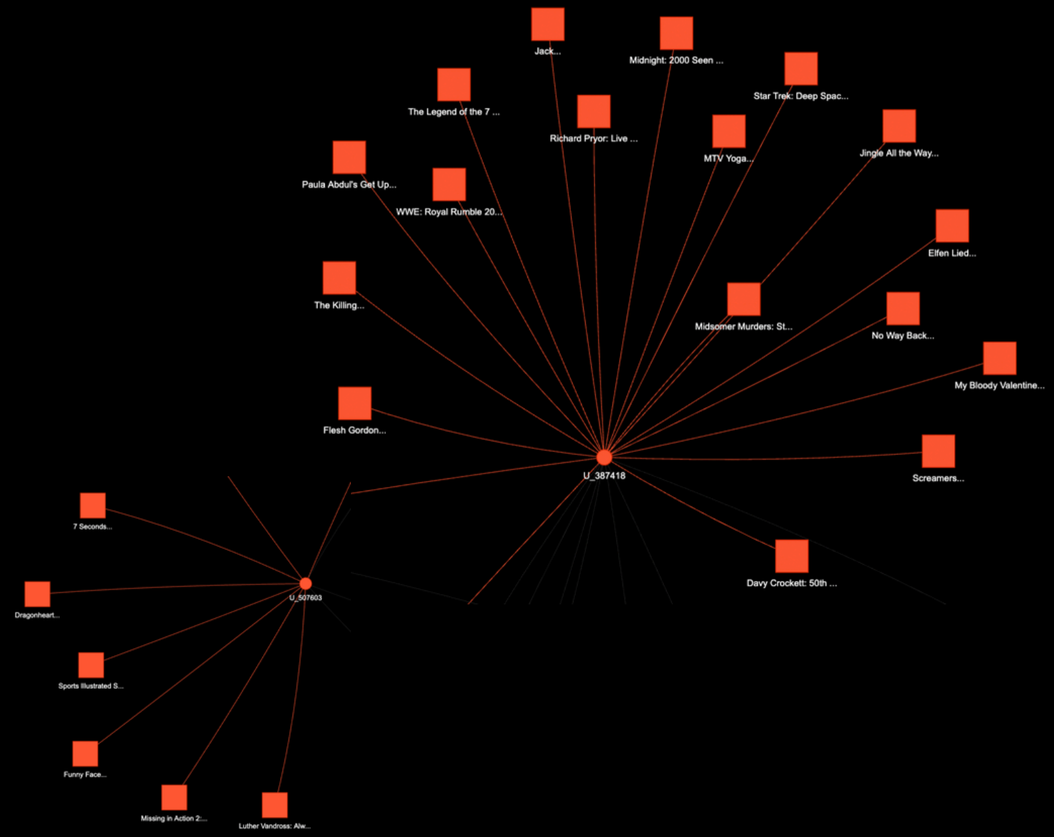


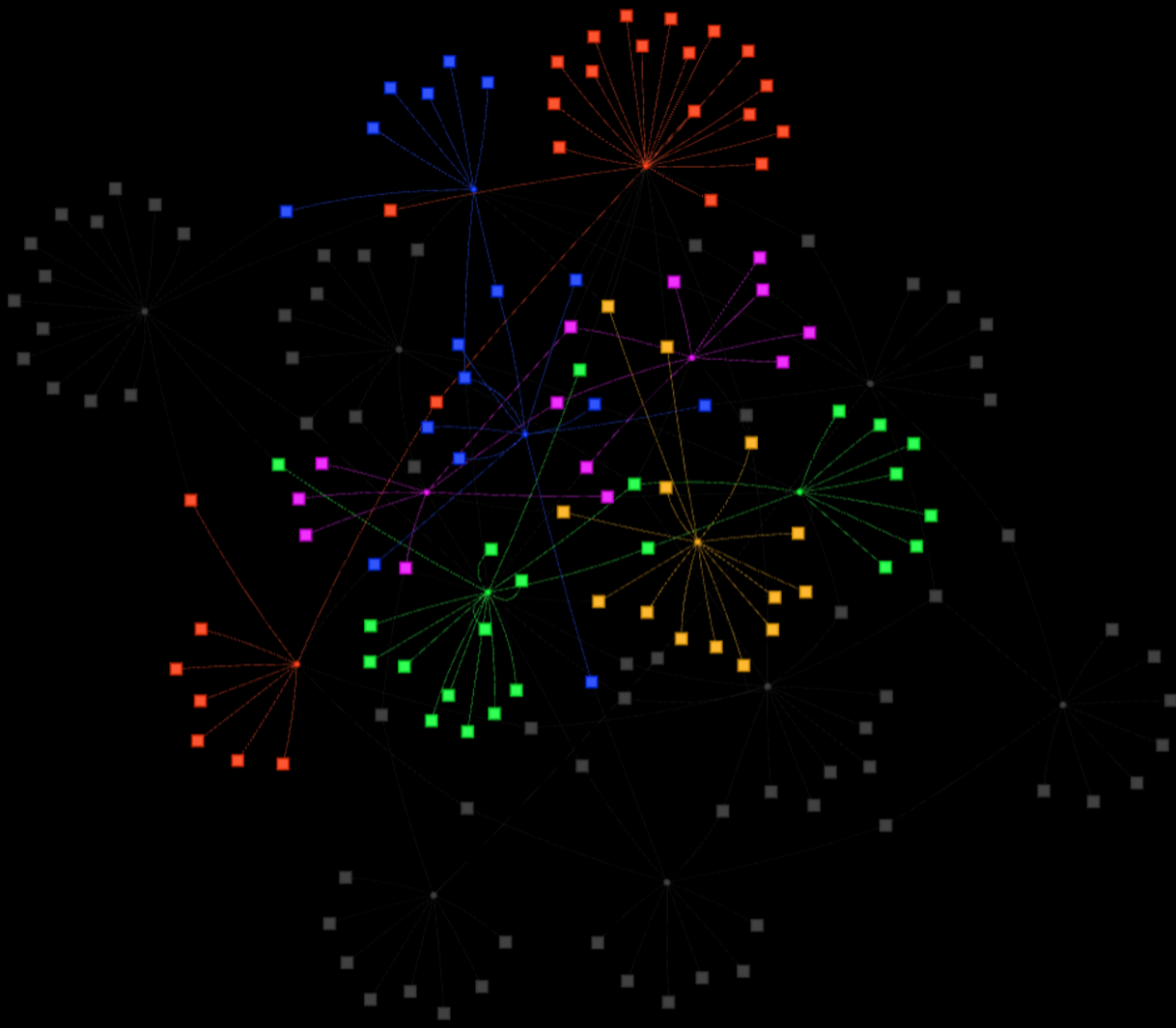
Graph-Based Aproach



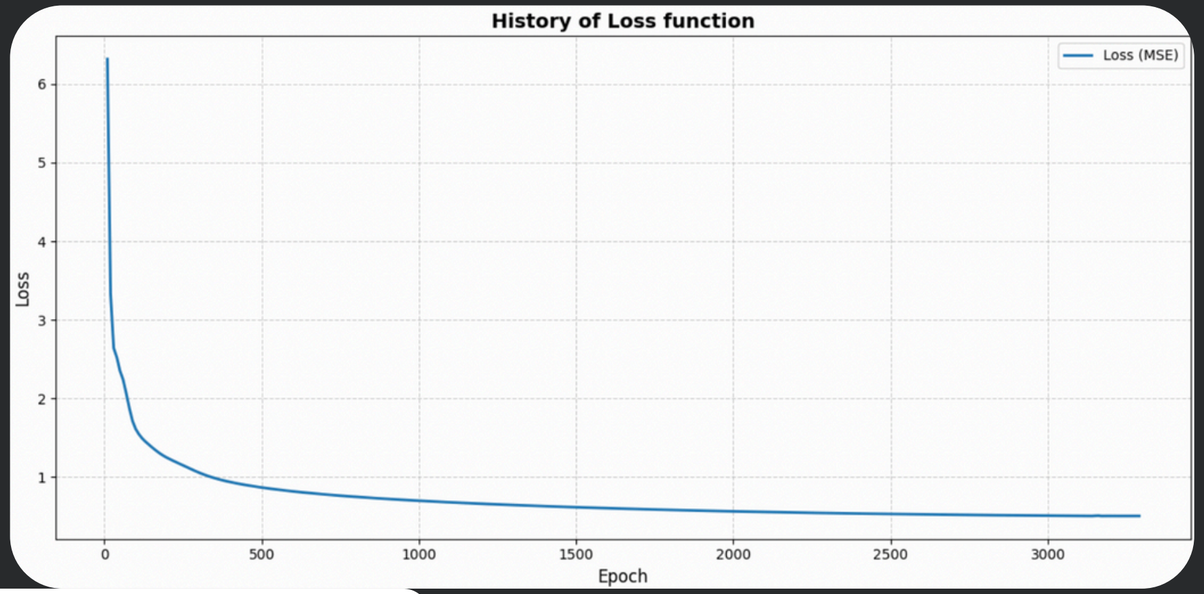


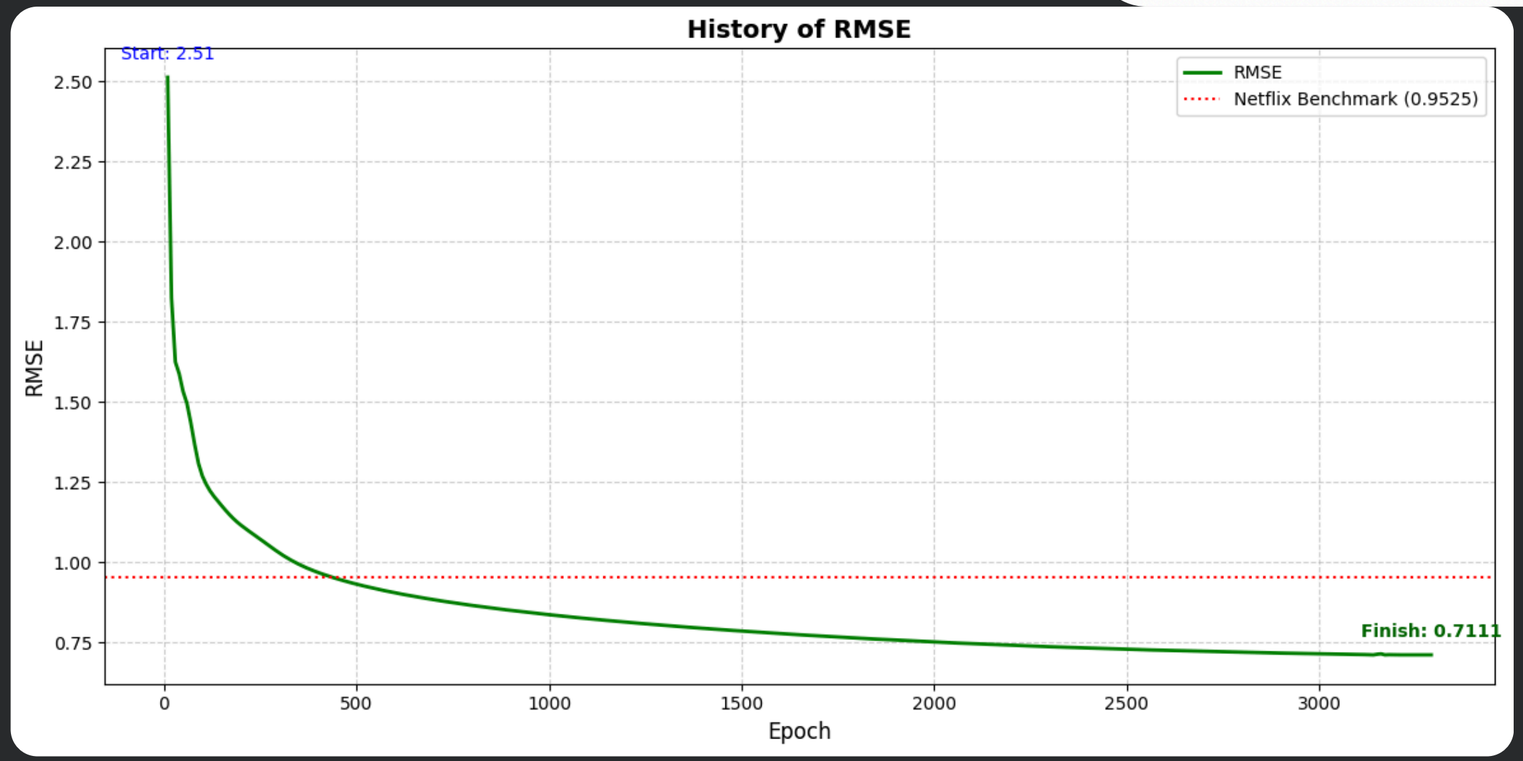
Louvain - community detection





Graph-Based Aproach: LightGCN






```
Epoch 3080, Loss: 0.5078, RMSE: 0.7126, LR: 0.01
Epoch 3090, Loss: 0.5076, RMSE: 0.7124, LR: 0.01
Epoch 3100, Loss: 0.5073, RMSE: 0.7122, LR: 0.01
Epoch 3110, Loss: 0.5070, RMSE: 0.7120, LR: 0.01
Epoch 3120, Loss: 0.5067, RMSE: 0.7118, LR: 0.01
Epoch 3130, Loss: 0.5062, RMSE: 0.7115, LR: 0.01
Epoch 3140, Loss: 0.5055, RMSE: 0.7110, LR: 0.01
Epoch 3150, Loss: 0.5084, RMSE: 0.7130, LR: 0.005
Epoch 3160, Loss: 0.5101, RMSE: 0.7142, LR: 0.005
Epoch 3170, Loss: 0.5055, RMSE: 0.7110, LR: 0.0025
Epoch 3180, Loss: 0.5064, RMSE: 0.7116, LR: 0.00125
Epoch 3190, Loss: 0.5060, RMSE: 0.7113, LR: 0.000625
Epoch 3200, Loss: 0.5056, RMSE: 0.7110, LR: 0.0003125
Epoch 3210, Loss: 0.5057, RMSE: 0.7111, LR: 0.00015625
Epoch 3220, Loss: 0.5056, RMSE: 0.7110, LR: 7.8125e-05
Epoch 3230, Loss: 0.5056, RMSE: 0.7111, LR: 3.90625e-05
Epoch 3240, Loss: 0.5056, RMSE: 0.7111, LR: 1.953125e-05
Epoch 3250, Loss: 0.5056, RMSE: 0.7111, LR: 9.765625e-06
Epoch 3260, Loss: 0.5056, RMSE: 0.7111, LR: 9.765625e-06
Epoch 3270, Loss: 0.5056, RMSE: 0.7111, LR: 4.8828125e-06
Epoch 3280, Loss: 0.5056, RMSE: 0.7111, LR: 2.44140625e-06
Epoch 3290, Loss: 0.5056, RMSE: 0.7111, LR: 1.220703125e-06
```

Graph-Based Aproach: LightGCN

1. Initialization:
Each user and item is assigned a learnable vector - embedding (e.g., self.users_emb and self.items_emb).

2. Neighborhood Propagation:
Nodes "pass" their features to their neighbors (called message). To keep values stable, the influence is scaled by a normalization factor

3. Multi-Layer Processing:
This propagation repeats for N layers. Each layer captures a deeper level of connection (e.g., "friends of friends" or "similar users who liked similar items").

4. Layer Combination:
The model calculates the weighted average (usually a simple mean) of embeddings from all layers, including the initial one. This prevents "over-smoothing" and keeps the original data relevant.

5. Final Prediction:
The predicted rating is the dot product between the final user embedding and the final item embedding.

```python
class LightGCNLayer(MessagePassing):
    def __init__(self):
        super(LightGCNLayer, self).__init__(aggr='add')

    def forward(self, x, edge_index):
        # Compute normalization (1/sqrt(deg(i)*deg(j)))
        row, col = edge_index
        deg = degree(col, x.size(0), dtype=x.dtype)
        deg_inv_sqrt = deg.pow(-0.5)
        deg_inv_sqrt[deg_inv_sqrt == float('inf')] = 0
        norm = deg_inv_sqrt[row] * deg_inv_sqrt[col]

        # Propagation of forward move
        return self.propagate(edge_index, x=x, norm=norm)

    def message(self, x_j, norm):
        return norm.view(-1, 1) * x_j

class LightGCN(nn.Module):
    def __init__(self, num_users, num_items, embedding_dim, n_layers):
        super(LightGCN, self).__init__()
        self.users_emb = nn.Embedding(num_users, embedding_dim)
        self.items_emb = nn.Embedding(num_items, embedding_dim)
        self.layers = nn.ModuleList([LightGCNLayer() for _ in range(n_layers)])

        # Wages initialization
        nn.init.normal_(self.users_emb.weight, std=0.1)
        nn.init.normal_(self.items_emb.weight, std=0.1)

    def forward(self, edge_index):
        x = torch.cat([self.users_emb.weight, self.items_emb.weight], dim=0)
        all_embs = [x]

        for layer in self.layers:
            x = layer(x, edge_index)
            all_embs.append(x)

        # Average of embedings from all of the layers
        final_embs = torch.mean(torch.stack(all_embs, dim=0), dim=0)
        return torch.split(final_embs, [self.users_emb.num_embeddings, self.items_emb.num_embeddings])

# Prediciton: Dot product of final embedings
def predict_rating(user_embs, item_embs, user_indices, item_indices):
    return (user_embs[user_indices] * item_embs[item_indices]).sum(dim=-1)
```

Possible improvements:

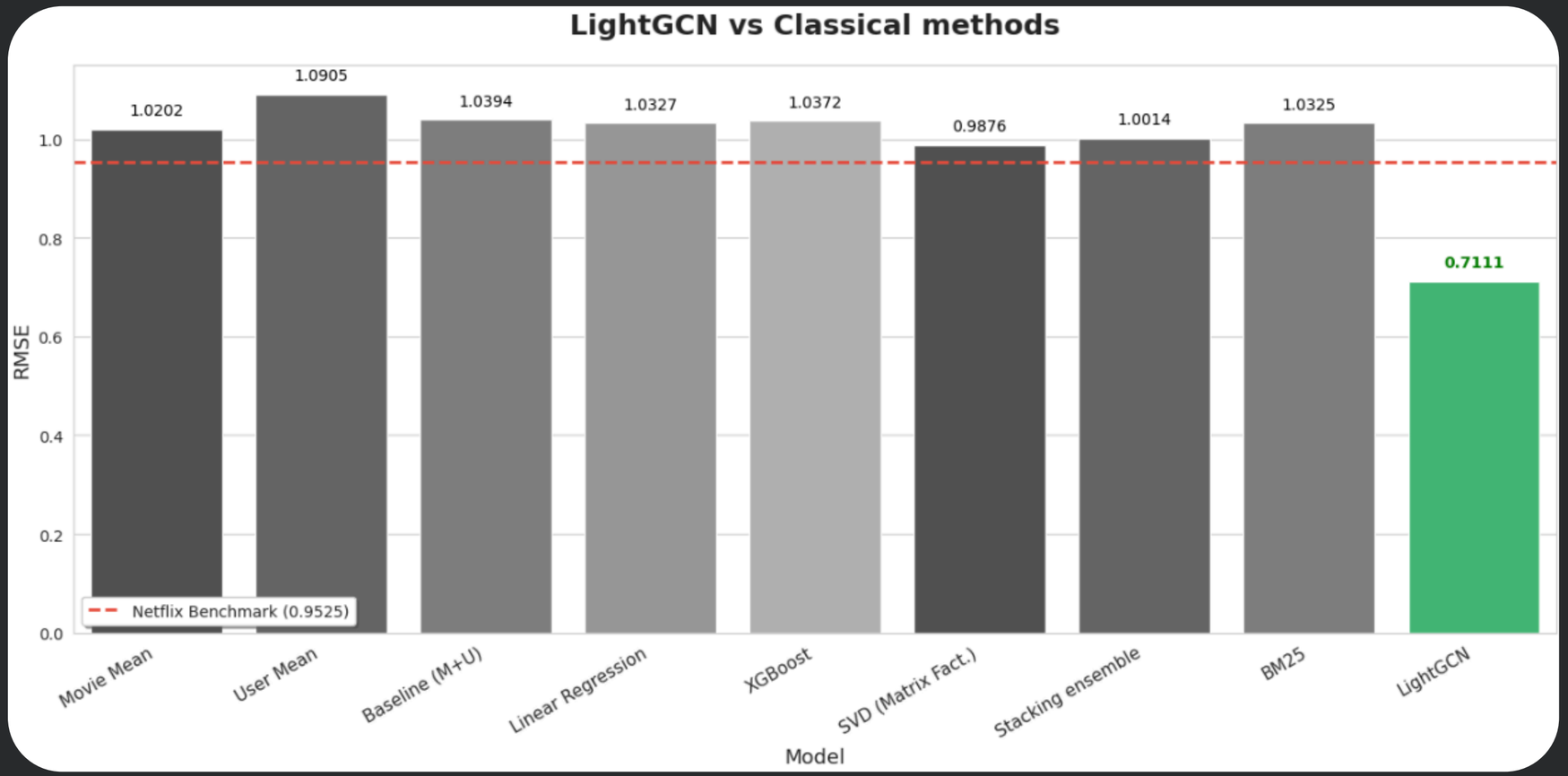
The standard graph construction for GCN does not take into account two important factors that may affect the user:

a. Intent
b. Session-based evaluation

A possible improvement would be to use a better neural network architecture, such as KGIN, which also takes into account user intent (Knowledge Graph Intent Network)

Although, we achieved **RMSE = 0.7111** what is much greater than Netflix's base line 0.9525 which shows strong GCN's capability for reccomandation systems

Results


LightGCN vs Classical methods
RMSE
Model
1.0202
1.0905
1.0394
1.0327
1.0372
0.9876
1.0014
1.0325
0.7111
Netflix Benchmark (0.9525)
Movie Mean
User Mean
Baseline (M+U)
Linear Regression
XGBoost
SVD (Matrix Fact.)
Stacking ensemble
BM25
LightGCN