Exploration of Recommendation System Algorithms

Shaoxuan Chen, Jingying Gao, Yun Jiang, Yilin Zhu

1 Abstract

In this project, three different types of recommendation system algorithms were explored on the MovieLens 100K dataset, including hybrid recommendation system using content-based filtering (CBF) & collaborative filtering (CF) recommenders, XGBoost and graph neural network (GNN). We showed that the hybrid recommender and XGBoost outperform the conventional filtering algorithms and basic tree-based methods, respectively. Meanwhile, GNN performed better in prediction than hybrid recommender and XGBoost for the smallest value of RMSE, 0.9359. Also, we found that incorporating more features into the model does not guarantee better performance. GNN could dig out a large amount of information simply from the rating matrix or user-movie network.

2 Introduction

With the rise of E-commerce and online business, from Facebook to Amazon to Netflix, recommendation systems are becoming increasingly important as they assist consumers in making decisions by recommending products and services tailored to their tastes and preferences. A good recommendation helps improve the customer's personal experience as well as the companies' business development. In this lab, we target on establishing a movie recommendation system which provides movie ratings and offers suggestions using the MovieLens 100K dataset. Different methods, from traditional collaborative filtering methods to XGBoost, and then to GNN, were explored and compared w.r.t prediction accuracy and efficiency.

3 Hybrid Recommendation System

The first methods we tried is the hybrid recommendation system. Given a set of users $U = \{u_1, u_2, ..., u_N\}$ and a set of items $I = \{i_1, i_2, ..., i_M\}$, let $u \in U$ be a user, $i \in I$ be an item, and each user u rates a set of items $RI_u = \{u_{i1}, u_{i2}, ..., u_{ip}\}$[3]. The rating of user u on item i is denoted by $r_{u,i}$, and a recommendation

system predicts the rating of user u on the item i is denoted by $\hat{r}_{u,i}$[3]. CBF and CF recommenders are two fundamental methods used to construct a recommendation system. CBF predicts the rating $\hat{r}_{u,i}$ based on the past ratings of the user u on items which are similar to the item i[3]. CF predicts the rating $\hat{r}_{u,i}$ based on the ratings of other users who are similar to the user u rated on the item i[3]. Both of them have its own advantages and disadvantages. Thus, combining them to create a hybrid recommender can complement their weaknesses and finally resulting in a recommender with better prediction ability.

3.1 Content-based Filtering Recommenders

Suppose if a user likes movie 1 and movie 2, then we can assume he/she also likes a similar movie k. Hence a CBF recommender is based on the similarity of movie attributes. The CBF algorithm finds the set of movies $RI_u = \{u_{i1}, u_{i2}, ..., u_{ip}\}$ previously rated by the user u and selects the movies that are similar to the targeted movie i using a similarity measure [3]. Three widely used machine learning algorithms were applied to select the most similar movies to the targeted movie i, which includes Lasso regression, KNN regression and Support Vector regression (SVR).

Lasso regression can be used to reduce model complexity, prevent over-fitting and is helpful for feature selection, as the regularization term $\lambda \sum_{j=0}^{p} |w_j|$ in the cost function $\frac{1}{2n} \sum_{i=1}^{M} (y_i - \sum_{j=0}^{p} w_j \times x_{ij})^2 + \lambda \sum_{j=0}^{p} |w_j|$ can lead to zero coefficients[2]. Here, λ was set to 1.0 to determine a large penalty should be paid if using large coefficients.

KNN regression can be used to calculate the average similarity score of the target movie based on a similarity measure of the K nearest neighbors[1]. Euclidean distance was chosen to measure the similarity, i.e., $d(x_i, y_i) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$.

SVR can be used to find the best fit line which is referred as hyperplane that contains the maximum of data points[9]. The nearest data points on either side of the hyperplane are termed Support Vectors, and they are used to plot the boundary line[9]. SVR aims to identify this fit line within a certain distance from the hyperplane and the boundary line[9]. Radial Basis Function kernel was employed in this algorithm to introduce a non-linearity to the SVR model.

In these CBF recommenders, users' ratings and movie's genres were used to build an individual model for each user. The prediction results were shown in Table 1. Lasso CBF was chosen for a hybrid recommender because of the smallest RMSE.

	Baseline	**Lasso**	**KNN-2**	**SVR**
Training RMSE	0.9956	1.009290	1.161614	1.010930

Table 1: Training RMSE of Different Models

3.2 Collaborative Filtering Recommenders

The neighborhood-based CF can recommend movies to a user that similar users also liked. Thus, CF is based on past interaction between users and movies. In this CF algorithm, the rating of user u on movie i is calculated based on the ratings of like-mined users that rate similarly on the movie i[3]. Thus, it depends strongly on the number of neighbors that rate the movie i and share similar ratings as the user u[3]. So neighborhood-based CF by KNN method was used to achieve prediction. This model calculates predictions based on the preferences of the most similar users regarding movie taste such that user data like age and will not have any influence on the recommendations[10]. The main procedures are as follows.

First, creating a rating matrix and replacing missing values with 0, then calculating user similarities by using Pearson correlation similarity score (Pearson correlation is a score between -1 and 1, where -1 indicates total negative correlation and 1 indicates total positive correlation, whereas 0 indicates that the two entities are in no way correlated with each other, $P_{u,v} = similarity(u,v) = \frac{\sum_{i \in RI_{u,v}} (r_{u,i} - \bar{r}_u)(r_{v,i} - \bar{r}_v)}{\sqrt{\sum_{i \in RI_{u,v}} (r_{u,i} - \bar{r}_u)^2} \sqrt{\sum_{i \in RI_{u,v}} (r_{v,i} - \bar{r}_v)^2}}$)[10]. Next, predicting ratings for every user by taking the average rating given by the K nearest neighbors who have rated the movie. KNN-40 CF was chosen for the hybrid recommender because of the smallest RMSE.

	Baseline	**KNN-10**	**KNN-20**	**KNN-30**	**KNN-40**	**KNN-50**
Training RMSE	0.9956	1.002173	0.985629	0.981519	0.980630	0.98245

Table 2: Training RMSE of Different Models

3.3 Hybrid Recommenders

Now, We aim to combine both the best CBF and CF models to create a hybrid recommender. Different weights for Lasso CBF and KNN-40 CF were tried and evaluated by their training RMSEs. As shown in Table 3, CBF model with 0.4 weight plus the CF model with 0.6 weight was chosen as the hybrid recommender for the smallest RMSE. Furthermore, testing RMSE of this hybrid recommender with a single CBF and CF recommender were compared. As shown in Table 4, it turned out that the hybrid recommender still has the smallest testing RMSE. Hence we can conclude that the hybrid recommender has better prediction performance than CBF or CF recommender, individually.

	Baseline	**(0.5,0.5)**	**(0.45,0.55)**	**(0.4,0.6)**
Training RMSE	0.9956	0.9515	0.9505	0.9503
	(0.35,0.65)	**(0.3,0.7)**	**(0.25,0.75)**	**(0.2,0.8)**
Training RMSE	0.9511	0.9527	0.9552	0.9586

Table 3: Training RMSE of Different Models

	CBF	CF	(0.4,0.6)
Testing RMSE	1.0133	0.9836	0.9530

Table 4: Testing RMSE of Different Models

4 XGBoost Trees for Regression

4.1 Methods and Derivation

The loss function used in XGBoost is the Squared-Error Loss, which is defined as:

$$L(y_i, p_i) = \frac{1}{2}(y_i - p_i)^2,$$

where y_i is the ith true value of the response variable, and p_i is the corresponding predicted value. XGBoost uses this loss function combined with other terms to build trees, which can be expressed as:

$$\min_O V = \sum_{i=1}^{n} L(y_i, p_i) + \frac{1}{2}\lambda O_{value}^2 + \gamma T,$$

where V is the function that we need to optimize, γ is the user defined complexity parameter, λ is the regularization parameter, O_value is the output value, and T is the number of terminal nodes in the tree. So the goal is to find an output value for the leaf that minimize equation V. Note that since we optimize the output value from the first tree, so we can replace the prediction p_i in the above function with $p^0 + O_{value}$, which p^0 represents the initial prediction. The optimization function now becomes:

$$\min_O V = \sum_{i=1}^{n} L(y_i, p_i^0 + O_{value}) + \frac{1}{2}\lambda O_{value}^2 + \gamma T.$$

XGBoost uses the Second Order Tayler Approximation for both Regression and Classification. The optimization function V can be expanded as:

$$\min_O V = \sum_{i=1}^{n} L(y_i, p_i) + [\frac{d}{dp_i} L(y_i, p_i)]O_{value} + [\frac{d^2}{dp_i^2} L(y_i, p_i)]O_{value}^2 + \frac{1}{2}\lambda O_{value}^2 + \gamma T.$$

Now if we use g to represent the Gradient, i.e., $\frac{d}{dp}L(y,p)$, and h to represent the Hessian, i.e., $\frac{d^2}{dp^2}L(y,p)$. The above function can be further simplified as below:

$$\min_O V = (g_1 + g_2 + ... + g_n)O_{value} + \frac{1}{2}(h_1 + h_2 + ... + h_n + \lambda)O_{value}^2.$$

The above function can be minimized by taking the first order derivative and set it to zero:

$$\frac{d}{dO_{value}}(g_1 + g_2 + ... + g_n)O_{value} + \frac{1}{2}(h_1 + h_2 + ... + h_n + \lambda)O_{value}^2 = 0$$

We can solve for the Output Value for the leaf:

$$O_{value} = \frac{-(g_1 + g_2 + ... + g_n)}{(h_1 + h_2 + ... + h_n + \lambda)}$$

Combined with the Gradient and Hessian we derived from the loss function:

$$gi = \frac{d}{dp_i}\frac{1}{2}(y_i - p_i)^2 = -(y_i - p_i)$$

$$h_i = \frac{d^2}{dp_i^2}\frac{1}{2}(y_i - p_i)^2 = \frac{d}{dp_i} - (y_i - p_i) = 1$$

The Output Value is finally derived as:

$$O_{value} = \frac{Sum\ of\ Residuals}{Number\ of\ Residuals + \lambda}.$$

Moreover, in XGBoost algorithm, Similarity Score, denote as S, is also a very important factor. And it is defined simply by flip the sign of the above optimization function:

$$S = -(g_1 + g_2 + ... + g_n)O_{value} - \frac{1}{2}(h_1 + h_2 + ... + h_n + \lambda)O_{value}^2$$

Combined the solution of g and h we derived above, we can have:

$$S = \frac{(g_1 + g_2 + ... + g_n)^2}{(h_1 + h_2 + ... + h_n + \lambda)^2} = \frac{Square\ of\ Sum\ of\ Residuals}{Number\ of\ Residuals + \lambda}$$

4.2 Algorithm

After the derivation of Output Value and Similarity Score. We can then apply them into the algorithm of XGBoost to make decisions of whether or not to prune the tree and split the nodes. The algorithm for XGBoost is summarized below. Note that as for the Regularization Parameter λ, if $\lambda > 0$, it will result in more pruning by shrinking the Similarity Scores, and it results in smaller Output Values for the leaves.

Algorithm 1: XGBoost Trees for Regression

Required: The initial prediction 0.5, learning rate ϵ (default = 0.3), User-defined tree complexity parameter γ, Regularization parameter λ: $\lambda > 0$ for more pruning, The maximum depth of the tree D (default = 6), Minimum value of residual R

```
while The depth of the tree ≤ D and Residuals ¿ R do
    for Each node in the tree do
        for Each way to split the node into Left and Right do
            Calculate the split value to separate the node into Left and Right
            Residuals = Observed values - Predicted values (Note: The initial prediction for all = 0.5)
            Similarity Score = Square of Sum of Residuals / Number of Residuals+λ
            Gain = Left_Similarity + Right_Similarity - Root_Similarity
            if Gain - γ > 0 then
                Do Not prune the tree;
                Output values for the leaves = Sum of Residuals / Number of Residuals+λ;
                Choose to make the split by the way that gives the largest gain;
                Prediction = Initial prediction + ϵ× Output value;
                Update the residual values by the new prediction values
            else
                Prune the tree;
            end
        end
    end
end
```

The XGBoost algorithm is extremely fast in computing and accurate in prediction. One reason for that is the formulas for computing as shown above are very simple. The other reason for that is XGBoost combines several other optimization methods and database techniques. And it can be summarized as follow. 1.Advanced Regularization: Both L_1 and L_2 regularization are used in XGBoost, which enable it has better generalization abilities. 2.Approximate Greedy Algorithm: XGBoost uses quantiles of candidate thresholds to split the node, which makes it faster than greedy algorithm, brute force or random generator used in the other algorithms. 3. Weighed Quantile Sketches Algorithm: When the data is to large to do computation, XGBoost first separate the data to different computers, and then merge then to make a histogram. Then choose the quantiles as thresholds to make the spilt. One thing which is very important here is that the sum

of weight within each quantile should be the same. And the weight is just the Hessian. 4. Sparsity-Aware Spilt Finding: Whenever there is any missing data. XGBoost separate the whole data into missing and non-missing datasets. And use both of the residuals in each dataset to calculate left gain and right gain. Further missing data will be categorized into the direction chosen by the the large gain value. 5. Blocks for Out-of-Core computation: As for speed of accessing the data with in CPU, from fast to slow is cache memory, main memory and then hard drive. XGBoost firstly assigns the Gradient and Hessian, which are very important data in making decisions, to the cache memory. Then the extra data will be assigned to main memory and finally to hard drive. This database technique also make XGBoost very fast in computation. Note that as for the Regularization Parameter λ, if $\lambda > 0$, it will result in more pruning by shrinking the Similarity Scores, and it results in smaller Output Values for the leaves.

4.3 Prediction Results

The data was applied to three tree-based methods: Random Forest, Gradient Boosting and XGBoost. The results are summarized as below.

Table 5: Prediction Results of Tree-Based methods

	Random Forest	**Gradient Boosting**	**XGBoost**
Test RMSE	0.98	1.0	0.94
Time/(second)	17.145	4.491	0.21

From the above table, we can see that XGBoost performs the best in prediction because of the smallest RMSE. Also, as for computational efficiency, we verify that XGBoost is extremely fast since it only takes 0.21 second, which is much shorter than the time needed in the other two methods.

5 Graph Neural Networks

In the following section, we introduce our investigation on graph neural networks and how could it be applied on the movie ratings prediction problem. This part will be organized in the following way: First, we introduce the definition and representation of a graph, as well as the node embeddings. Second, we revisit some fundamental concepts and tools for deep learning. Finally, we present two types of graph neural networks that we used for the prediction problem.

5.1 Introduction to Graph and Node Embeddings

Definition 1.1 : A heterogeneous graph is defined as $\mathbf{G} = (\mathbf{V}, \mathbf{E}, \mathbf{R}, \mathbf{T})$, where $v_i \in \mathbf{V}$ are nodes with different node types $\mathbf{T}(v_i)$ and $e_{ij} := (v_i, r, v_j) \in \mathbf{E}$ are edges with different relation types $r \in \mathbf{R}$.

Because there is only one relation type in our user-movie network, we will just denote the graph $\mathbf{G}$ by $(\mathbf{V}, \mathbf{E}, \mathbf{T})$ or just $(\mathbf{V}, \mathbf{E})$ if there is only one type of nodes. Generally, the problem could be treated as a supervised link attribute inference problem. Our goal is to build a valid network, conduct graph splitting to intentionally hide some ratings, and then predict these ratings using some supervised learning techniques (*e.g.* Deep learning methods).

To accomplish the link prediction task, we need to further introduce the concept of node embeddings.

Definition 1.2 : Assume we have a graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$. The procedure that encode nodes so that similarity in the embedding space approximates similarity in the graph is called node embeddings. *i.e.* Find some encoder function ENC that maps each node v to a d-dimensional embedding space: $\text{ENC}(v) = Z_v$ such that

$$\text{Similarity}(u, v) \approx Z_u^T \cdot Z_v,$$

where the inner product is one way to measure the similarity in the embedding space.

Several methods were developed to realize the node embeddings. In this project, we utilize a prevailing and efficient way-graph neural networks, which encode the nodes by multiple layers of non-linear transformations based on the graph structure.

5.2 Basic Concepts in Deep Learning

Some concepts and techniques in deep learning that we used to build our model are revisited in this section. First of all, the prediction task is formulated as an optimization problem:

$$\theta = \arg\min_{\theta \in \Theta} \mathcal{L}(y, f(x, \theta)),$$

where the loss function $\mathcal{L} = ||y - f(x, \theta)||_2^2$. The objective function will be optimized with the techniques of back propogation and a variant of mini-batch stochastic gradient descent-Adam algorithm.

Definition 1.3 : Mini-batch stochastic gradient descent(SGD) is designed to increase the computational efficiency of gradient descent operation. At each step, a minibatch $\mathcal{B}$ containing a subset of the entire data is selected to compute the gradient $g_t = \partial_\theta \frac{1}{\mathcal{B}} \sum_{i \in \mathcal{B}} f(x_i, \theta)$, and then proceed to update parameters with this gradient $\theta \leftarrow \theta - \eta_t g_t$, where η_t is the learning rate to be tuned.

Definition 1.4 : Adaptive moment estimation (Adam) algorithm [7] combines the idea of momentum and the second moment of the gradient. At each stage, Adam compute two state variables

$$M_t = \beta_1 M_{t-1} + (1 - \beta_1) g_t, \quad G_t = \beta_2 G_{t-1} + (1 - \beta_2) g_t \odot g_t.$$

Here β_1 and β_2 are nonnegative weighting parameters. M_t and G_t could be viewed as the momentum and the second moment of the gradient, respectively. The parameters will then be updated by

$$\theta \leftarrow \theta - \frac{\eta_t}{\sqrt{\hat{G}_t + \epsilon}} \hat{M}_t,$$

where $\hat{G}_t, \hat{M}_t$ are corrected state variables and ϵ is a predefined sufficiently small positive number.

Definition 1.5 : Each layer of multi-layer perceptron (MLP) combines linear transformation and a non-linearity function:

$$x^{l+1} = \sigma(W_l x^l + b^l),$$

where W_l is the weight matrix that transforms hidden representation at layer l to layer $l+1$, b^l is the bias for layer l, and $\sigma(\cdot)$ is a function adding non-linearity to the neural network, *e.g.* $\text{ReLu}(x) = \max(x, 0)$.

5.3 Graph Convolutional Neural Network (GCNN) and Its Realization

5.3.1 Introduction

In recommendation system under collaborative filtering framework, a natural solution is to use graphs as generic descriptors of signal structure with signal values associated to nodes and edges expressing expected similarity between signal components. As shown in Figure 1, we can take movies or users or (movie, user) pairs as different nodes.[4] The signal value of each node is the associated movie rating. Different nodes could be connected by edges and each edge represent the similarity of past ratings between two nodes. The underlying assumption is that there exist an underlying set of true ratings or scores, but that we only observe a subset of those scores. The set of unseen scores can be estimated from the set of observed scores.

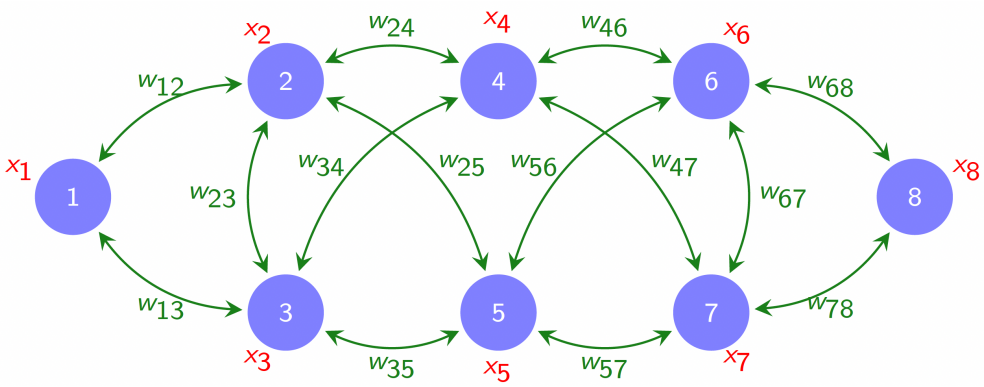


Figure 1: Schematic example of a graph structure

To make reasonable prediction for missing ratings, it is practical to use Graph Convolutional Neural Network

(GCNN). In Figure 2, we show the general structure and representation of GCNN. We could see that a GCNN with L layers follows L recursions of the form

$$x_l = \sigma\Big[z_l\Big] = \sigma\Big[\sum_{k=0}^{K-1} h_{lk} S^k x_{l-1}\Big] \tag{1}$$

where S is the graph shift operator which represents the connection relationship between nodes, such as adjacency matrix. h_{lk} are the graph convolutional filters, the values of which will be determined from training process. σ is the graph nonlinearity function. Finally, the output of GCNN is just the output of the last layer L:

$$x_L = \Phi(x; \mathrm{S}, H) \tag{2}$$

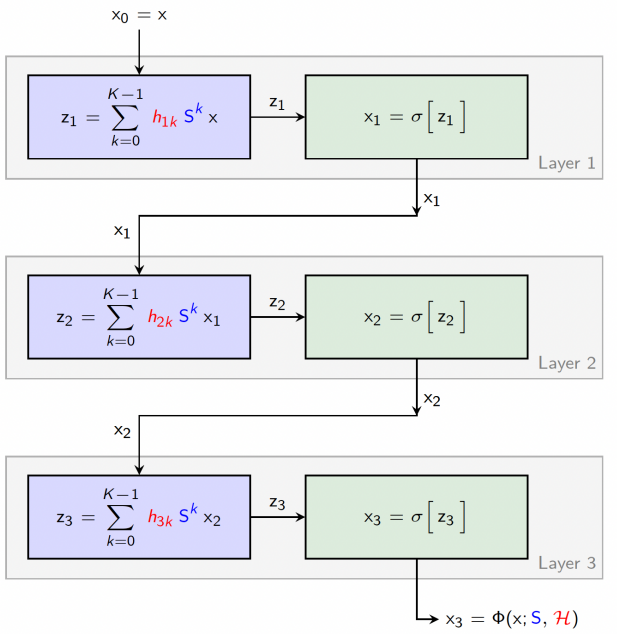


Figure 2: Workflow of Graph Convolutional Neural Network

5.3.2 Implementation details

In this section, we use the same training and testing data set as used in the previous sections. To construct the graph shift operator S, we first build the adjacency matrix $\mathbf{A}$ which measures the similarity between

different nodes

$$\mathbf{A} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ w_{n1} & w_{n2} & \cdots & w_{nn} \end{bmatrix} \tag{3}$$

The value w_{ij} is the similarity between node i and j, and it can be measured in a similar way as in previous section, like cosine similarity. The final graph shift operator is the normalized adjacency matrix through

$$\begin{aligned} \mathbf{S} &= \mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2} \\ L_{ij} &= -A_{ij}, L_{ii} = \sum_{j(i)} w_{ij} \\ D_{ij} &= 0, D_{ii} = L_{ii} \end{aligned} \tag{4}$$

The GCNN model was implemented based on a GNN architecture, Alelab GNN library [5]. The training process was conducted to obtain an optimal set of graph convolutional filters hyperparameters. We use Adam optimizer and ReLU implemented by PyTorch in our GNN model. During the training, we set the number of epochs as 50, batch size as 5 and the learning rate as 0.005.

5.3.3 Results

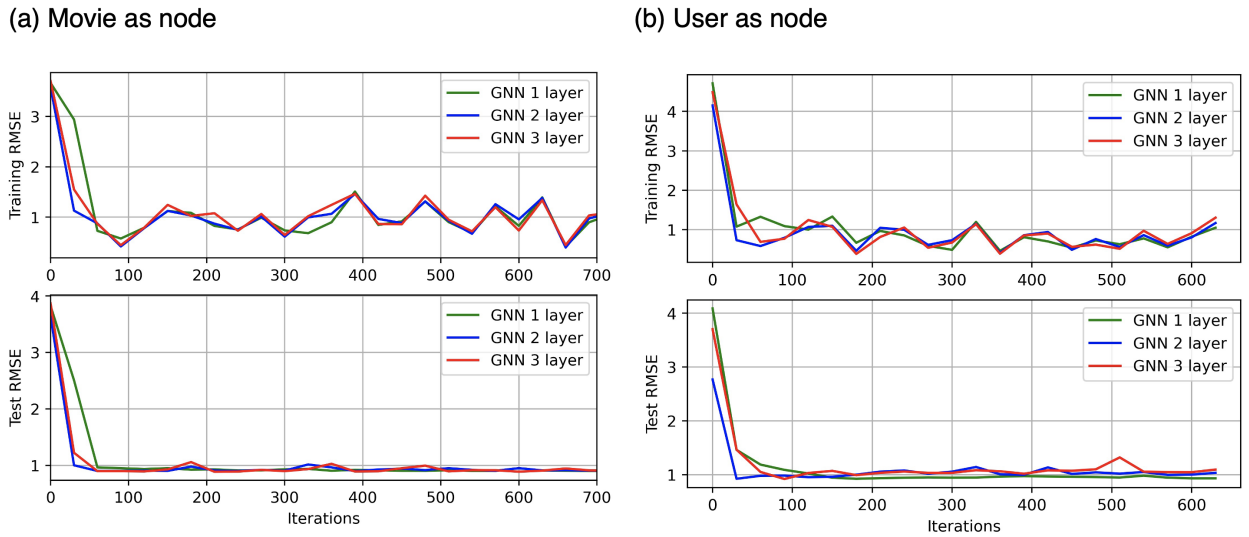


Figure 3: Training and test RMSE during training iterations for (a) movies as graph nodes, (b) users as graph nodes

In this section, we give two independent examples of GCNN simulation. The first one is taking movies as graph nodes and we demonstrate the result with Movie number 258 for prediction purpose. As shown

in Figure 3(a), as the number of training iteration increases, both training RMSE and test RMSE quickly decrease and converge to certain values around 1. Interestingly, adding more layers to the GCNN model does not significantly improve the behavior of outputs, where 1-layer GNN, 2-layer GNN and 3-layer GNN show similar RMSE for training and testing data. Similar conclusion can be drew from the Figure 3(b) which is the result from GCNN taking users as nodes instead.

Table 6: Root mean square error (RMSE) for test predictions using different models

GCNN Model	Movies as nodes	Users as nodes
GNN 1 layer	0.91	0.97
GNN 2 layer	0.89	0.99
GNN 3 layer	0.88	0.92

The values of RMSE on testing data set with differing GCNN model are shown in Table 6. Adding more layers (less than 4) to the GCNN model may decrease the RMSE value to a minor extent but there still exists an exception.

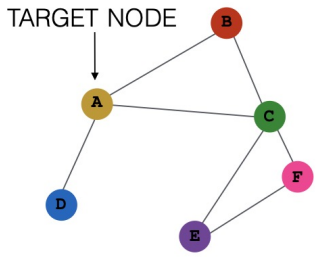
Finally, we need to mention that in the current implementation of GCNN models, we take either movies or users as nodes. Such implementation is robust and straightforward but has certain limitations. For example, such model cannot efficiently provide rating predictions for all (movie,user) pairs. To derive more practical way for movie ratings prediction, more sophisticated model should be considered and we will introduce GraphSAGE model in the next section.

5.4 GraphSAGE and Its Realization

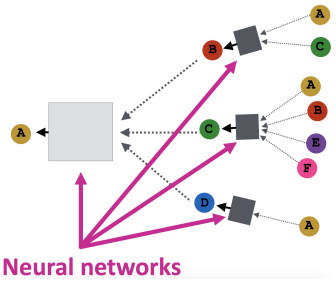
In this section we present another variant of graph neural networks-GraphSAGE. To formulate our report more succinctly, we would just identify the major differences from the our last model-GCNN, including the model setting and framework of the graph neural network.

The user-movie network is constructed with two types of nodes, users and movies. The resulting graph is bipartite since its vertices could be divided into two independent groups, in which there are no intra-group edges. The framework of GraphSAGE could be illustrated with the following four figures:

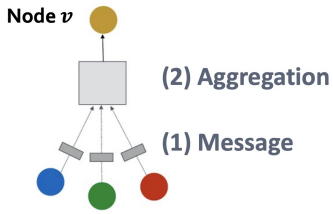
Suppose we are given a graph and a target node **A**. Our primary goal is to generate the node embedding for **A** based on its local graph neighborhoods. GraphSAGE employs a message passing path as shown in figure 4(b) to encode the node **A**, which is called a (2-layers) computational graph for the specific node. Each neural network in the computational graph is responsible for transmitting and aggregating the information from the node itself and its neighbors, as illustrated by the figure 4(c). In sum, we could formulate each neural network by



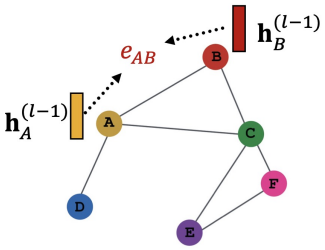
(a) Input Graph



(b) Message Passing path



(c) Inner Structure of a single neural network



(d) Link prediction given two node embeddings

Figure 4: GraphSAGE Framework [6]

$$h_v^{k+1} = \sigma\left(W_k \sum_{u \in N(v)} \frac{h_u^k}{|N(v)|} + B_k h_v^k\right), \ \forall k \in \{0, 1, ..., L-1\},$$

where h_v^k is the embedding of node v at layer k (note that h_v^0 is the initialized embedding for node v. Common initialization methods are constant initialization, one-hot initialization and self attributes initialization, etc.), L is the total number of layers, $N(v)$ denotes the set of neighbors of node v, W_k and B_k are trainable weighted matrix, and σ is a non-linearity function.

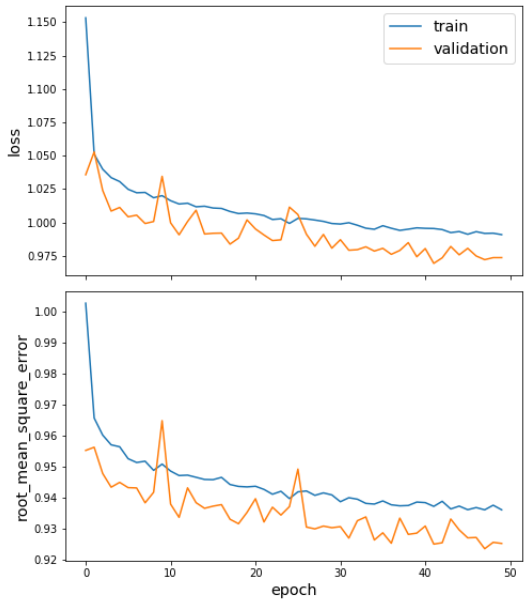
In this formulation, the information transmitted through the graph is the embedding for each node at each layer. And $\sum_{u \in N(v)} \frac{h_u^k}{|N(v)|}$ serves as a permutation equivariant function to aggregate the information collected from $h_u^k, u \in N(v)$ and h_v^k.

Predicting the edge attribute-rating becomes straightforward once we obtained the embeddings for nodes connecting to the edge. Suppose we will be predicting the edge attribute e_{AB} given two node embeddings h_A^{l-1}, h_B^{l-1}. We just need to concatenate the two embeddings to derive the so-called link embedding and feed it into a link regression layer to obtain predicted edge attribute, *i.e.*

$$e_{AB} = f\left(\text{CONCAT}(h_A^{l-1}, h_B^{l-1})\right),$$

where f maps any vector in the embedding space to a real number from 1 to 5. One can refer to this article [8] for one type of sophisticated definition of this $f(\cdot)$.

The training history for the optimal set of hyperparameters is briefly displayed as follows:



The optimal RMSE for validation data is 0.9254 and the resulting RMSE for testing data set is 0.9359.

6 Summary

In this report, we have proposed several models to predict the rating for each movie from each user. First, we developed a hybrid recommender that combines the strength of the CF method and the CBF method. The results indicate that the model averaging is indeed a practical and efficient way to improve the prediction performance. Second, we implemented several tree-based methods, including Random Forest, Gradient Boosting and XGBoost, and compared their prediction performance on the testing data. The comparison results displayed in section 3 unveil the superiority of XGBoost method over this prediction task. Finally, we presented the GNN methods, which, compared to the previous two models, requires fewer input features to tackle the prediction task. Moreover, among all methods that we have implemented, the GraphSAGE algorithm achieves the lowest RMSE 0.9359 on the testing data set. Based upon our investigation, GNNs outperform the other two types of algorithms in terms of both accuracy and data preprocessing complexity.

References

[1] Sangeet Aggarwal. "K-Nearest Neighbors". In: *https://towardsdatascience.com/k-nearest-neighbors-94395f445221* (June 8, 2020).

[2] Saptashwa Bhattacharyya. "Ridge and Lasso Regression: L1 and L2 Regularization". In: *https://towardsdatascience.com/ridge-and-lasso-regression-a-complete-guide-with-python-scikit-learn-e20e34bcbf0b* (Sep 26, 2018).

[3] Belkacem Chikhaoui, Mauricio Chiazzaro, and Shengrui Wang. "An improved hybrid recommender system by combining predictions". In: *2011 IEEE Workshops of International Conference on Advanced Information Networking and Applications*. IEEE. 2011, pp. 644–649.

[4] UPenn Engineering. "Graph Neural Networks". In: *http://gnn.seas.upenn.edu* (Fall 2020).

[5] Fernando Gama et al. "Convolutional Neural Network Architectures for Signals Supported on Graphs". In: *Trans. Sig. Proc.* 67.4 (Feb. 2019), pp. 1034–1049.

[6] Stanford CS224W Jure Leskovec. "Machine Learning with Graphs". In: *http://cs224w.stanford.edu* (Fall 2021).

[7] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *arXiv:1412.6980* 67.4 (2017).

[8] T.N. Kipf R. van den Berg and M. Welling. "Graph convolutional matrix completion". In: *arXiv:1706.02263* (2017).

[9] Ashwin Raj. "Unlocking the True Power of Support Vector Regression". In: *https://towardsdatascience.com/unlocking-the-true-power-of-support-vector-regression-847fd123a4a0* (Oct 3, 2020).

[10] SebastianRokholt. "Hybrid Recommender System". In: *https://github.com/SebastianRokholt/Hybrid-Recommender-System* (Sep, 2021).