Collaborative Filtering

Charilaos I. Kanatsoulis, Navid NaderiAlizadeh,
Alejandro Parada-Mayorga, Alejandro Ribeiro, and Luana Ruiz *

June 6, 2023

1 Collaborative Filtering

The objective of this lab is to design a recommendation system that predicts the ratings that customers would give to a certain product. Say, the rating that a moviegoer would give to a specific movie, or the rating that an online shopper would give to a particular offering. A possible approach to making these predictions is to leverage the ratings that customers have given to this or similar products in the past. This is called collaborative filtering.

A schematic representation of collaborative filtering is shown in Figure 1. The underlying assumption is that there is a true set of ratings that different customers would give to specific products. These ratings remain unobserved and are denoted by $\bar{\mathbf{X}}$ in Figure 1. What we *do* have available are a subset of these ratings. They are represented by $\mathbf{X}$ in Figure 1 where all of the missing ratings are represented by a blank space. This is a reasonable model of reality. Each of us has seen a small number of movies or bought a small number of offerings. Thus, the ratings matrix $\mathbf{X}_u$ contains only a few entries that correspond to rated products. Our goal is to recover estimates $\mathbf{Y}$ of the unobserved ratings $\bar{\mathbf{X}}$.

As a specific example, we use the MovieLens-100k dataset. The MovieLens-100k dataset consists of ratings given by U users to P movies (products).

*In alphabetical order.

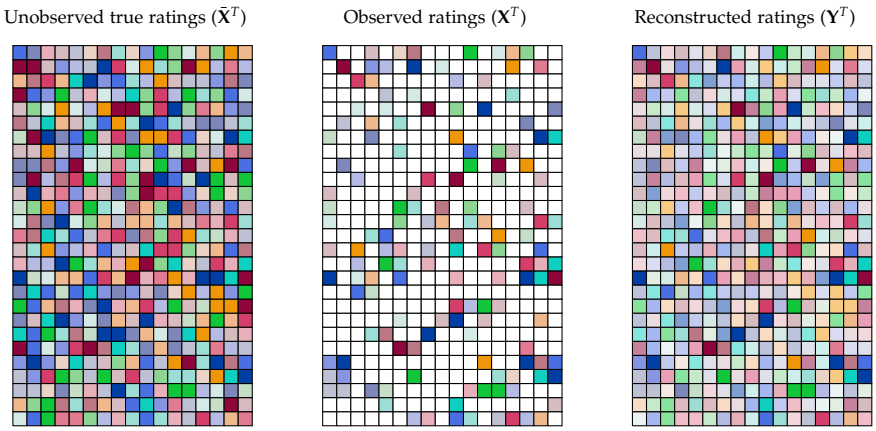

Figure 1. Recommendation with Collaborative Filtering.

The existing movie ratings are integer values between 1 and 5. Therefore, the data are represented by a $U \times P$ matrix X where x_{up} is the rating that user u gives to movie p. If user u has not rated movie p, we adopt the convention that $x_{up} = 0$. We see that each row of this matrix corresponds to a vector of ratings $\mathbf{x}_u$ of a particular user.

1.1 Product Similarity Graph

To build the collaborative filtering system, we use the rating history of all movies and all users to compute a graph of product similarities. This is a graph in which nodes p represent different movies and weighted edges S_{pq} denote similarities between products p and q. The edges of the graph are grouped in the adjacency matrix $\mathbf{S}$.

To compute the entries S_{pq} of the product similarity graph we use the raw $U \times P$ movie rating matrix to evaluate crosscorrelations between movie ratings of products p and q. To make matters simpler we have constructed this graph already and are making it available as part of the dataset.

Task 1 Download the movie rating data to your computer and upload the data "movie_data.p" to this processing environment. Plot the adjacency

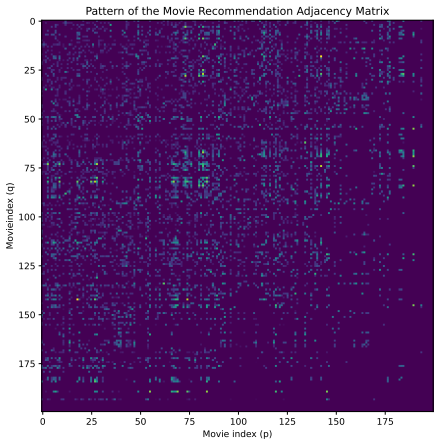

Figure 2. Adjacency Matrix of the Movie Similarity Graph. Brighter dots correspond to pairs of movies that different watchers tend to score with similar ratings.

matrix **S** as an image. ■

Success in Task 2 must have produced the plot in Figure 2. In this figure each bright dot corresponds to a large entry $S(p,q)$. This denotes a pair of movies to which watchers tend to give similar scores. For instance, say that when someone scores ”Star Wars IV” highly, they are likely to score ”Star Wars V” highly and that the converse is also true; poor scores in one correlate with poor scores in the other. The entry $S(\text{”Star Wars IV”}, \text{”Star Wars V”})$ is large because the crosscorrelation between the scores of these entries is high.

Fainter entries $S(p,q)$ denote pairs of movies with less socre correlation. Perhaps between ”Star Wars” and ”Star Trek” which have overlapping but not identical fan bases. Dark entries $S(p,q)$ correspond to pairs movies with no correlation between audience scores. Say when p is the index of ”Star Wars” and q is the index of ”Little Miss Sunshine.”

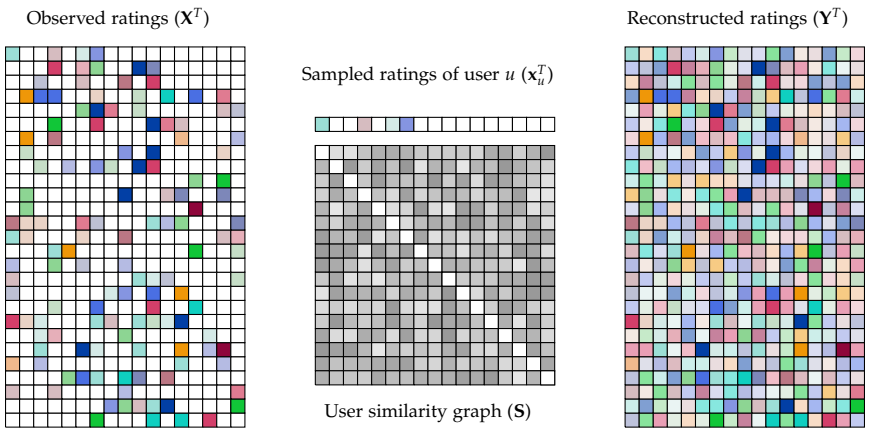

Figure 3. Reconstruction of Movie Ratings with a Movie Similarity Matrix

1.2 Rating Signals

The vector of ratings $\mathbf{x}_u$ of a particular user is interpreted as a signal supported on the graph. That is, a signal in which the pth component x_{up} is associated with node p. In this context, the weights of the product similarity graph become an expectation of similarity between ratings x_{up} and x_{uq}. If S_{pq} is large we expect these ratings to be similar. If S_{pq} is small we have no expectation of proximity or not between them.

We then have a system with the architecture shown in Figure 3. Rating signals $\mathbf{x}_u$ of individual users are extracted from the raw rating matrix and are interpreted as signals supported on the graph $\mathbf{S}$ that we loaded in Task 2. We want to leverage the graph $\mathbf{S}$ to make rating predictions $\mathbf{y}_u$ for this particular user.

We will, more precisely, develop and evaluate a graph filter and a graph neural network (GNN) for making these rating predictions.

1.3 Rating Data Format and Rating Loss

To train the collaborative filtering system we use rating histories to create a dataset with entries $(\mathbf{x}_n, y_n, p_n)$. In these entries $\mathbf{x}_n$ is a vector that contains the ratings of a particular user, y_n is a scalar that contains a rating that we want to predict, and p_n is the index of the movie (product) that corresponds to the rating y_n. To evaluate this collaborative filtering system we use rating histories to create a dataset with entries having the same format. Both of these datasets can be constructed from the raw $U \times P$ movie rating matrix, but to make matters simpler we have constructed them already and are making them available as part of the dataset.

If we have a function $\hat{\mathbf{y}}_n = \Phi(\mathbf{x}_n; \mathcal{H})$ that makes rating predictions out of availbale ratings, we can evaluate the goodness of this function with the squared loss

$$\ell\big(\Phi(\mathbf{x}_n; \mathcal{H}), y_n\big) \;=\; \Big[\big(\hat{\mathbf{y}}_n\big)_{p_n} - y_n\Big]^2 \;=\; \Big[\big(\Phi(\mathbf{x}_n; \mathcal{H})\big)_{p_n} - y_n\Big]^2. \quad (1)$$

Notice that in this expression the function $\hat{\mathbf{y}}_n = \Phi(\mathbf{x}_n; \mathcal{H})$ makes predictions for all movies. However, we isolate entry p_n and compare it against the rating y_n. We do this, because the rating y_n of movie p_n is the one we have available in the training or test sets.

We remark the fact that the function $\hat{\mathbf{y}} = \Phi(\mathbf{x}; \mathcal{H})$ makes predictions for all movies is important during operation. The idea of the recommendation system is to identify the subset of products that the customer would rate highly. They are the ones that we will recommend. This is why we want a system that has a graph signal as an output even though the available dataset has scalar outputs.

Task 2 Write a function to evaluate the training loss in (1). ■

2 Graph Convolutions

Let $\mathbf{S}$ denote a matrix representation of a graph. Supported on the nodes of the graph we are given a graph signal $\mathbf{x}$. We also consider a set of K

coefficients h_k from $k = 0$ to $k = K - 1$. A graph convolutional filter is a linear map acting on $\mathbf{x}$ defined as a polynomial on the matrix representation of the graph with coefficients h_k,

$$\mathbf{z} \;=\; \mathbf{h} *_{\mathbf{S}} \mathbf{x} \;=\; \sum_{k=0}^{K-1} h_k \mathbf{S}^k \mathbf{x}. \tag{2}$$

Graph convolutions generalize convolutions in time to graphs. That this is true can be seen if we represent time with a directed line graph. Considering (2) for the particular case in which $\mathbf{S}$ is the adjacency matrix of this line graph, the product $\mathbf{S}^k\mathbf{x}$ results in a k-shift of the time signal $\mathbf{x}$. For this reason we sometimes refer to $\mathbf{S}$ as a shift operator. We also point out that although we work with an adjacency matrix in this lab any matrix representation of the graph can be used in (2).

One advantage that graph filters share with time convolutions is their locality. To see this, define the diffusion sequence as a collection of graph signals $\mathbf{u}_k = \mathbf{S}^k\mathbf{x}$ and rewrite the filter in (2) as,

$$\mathbf{z} \;=\; \sum_{k=0}^{K-1} h_k \mathbf{S}^k \mathbf{x} \;=\; \sum_{k=0}^{K} h_k \mathbf{u}_k \tag{3}$$

It is ready to see that the diffusion sequence is given by the recursion $\mathbf{z}_k = \mathbf{S}\mathbf{z}_{k-1}$ with $\mathbf{z}_0 = \mathbf{x}$. Further observing that $S_{ij} \neq 0$ only when the pair (i, j) is an edge of the graph, we see that the entries of the diffusion sequence satisfy

$$u_{k,i} \;=\; \sum_{j:(i,j)\in\mathcal{E}} S_{ij} u_{k-1,j}. \tag{4}$$

We can therefore interpret graph filters as operators that propagate information through adjacent nodes. This is analogous to the propagation of information in time with the application of time shifts. The locality of graph convolutions is one of the motivations for their use in the processing of information supported on graphs. The other reason is their *equivariance to permutations.*

Because it aggregates with a weighted sum the information from neighboring nodes, the operation in (4) is sometimes called an aggregation information. Because it aggregates at node i information that is passed from adjacent nodes j, we sometimes say that graph filters are message-passing architectures and the GNNs that are derived from them are called message passing GNNs.

2.1 Graph Convolutions with Multiple Features

To increase the representation power of graph filters we extend them to add multiple features. In these filters the input is a matrix $\mathbf{X}$ and the output is another matrix matrix $\mathbf{Y}$. The filter coefficients are matrices $\mathbf{H}_k$ and the filter itself is a generalization of (2) in which the matrices $\mathbf{H}_k$ replace the scalars h_k,

$$\mathbf{Z} = \sum_{k=0}^{K} \mathbf{S}^k \mathbf{X} \mathbf{H}_k. \tag{5}$$

In (5), the input feature matrix $\mathbf{X}$ has dimension $N \times F$ and the output feature matrix $\mathbf{Y}$ has dimension $N \times G$. This means that each of the F columns of $\mathbf{X}$ represents a separate input feature whereas each of the G columns of $\mathbf{Y}$ represents an output feature. To match dimensions, the filter coefficient matrices $\mathbf{H}_k$ must be of dimension $F \times G$.

Other than the fact that it represents an input-output relationship between matrices instead of vectors, (5) has the same structure of (2).

In particular, we can define a diffusion sequence $\mathbf{U}_k$ through the recursion $\mathbf{U}_k = \mathbf{S}\mathbf{U}_{k-1}$ and rewrite (5) as

$$\mathbf{Z} \;=\; \sum_{k=0}^{K-1} h_k \mathbf{S}^k \mathbf{X} \;=\; \sum_{k=0}^{K} h_k \mathbf{U}_k. \tag{6}$$

This is worth remarking because we can write the diffusion sequence as a message passing aggregation operation. Indeed, if $\mathbf{u}_{k,i}$ is the ith row of $\mathbf{U}_k$ we can write the diffusion sequence recursion as

$$\mathbf{u}_{k,i} \;=\; \sum_{j:(i,j)\in\mathcal{E}} S_{ij} \mathbf{u}_{k-1,j}. \tag{7}$$

In (7) nodes j in the neighborhood of i pass the message $\mathbf{u}_{k-1,j}$. Node i aggregates these messages to create the updated message $\mathbf{u}_{k,i}$ that it passes on to its neighbors.

Task 3 Write a function that implements a graph filter. This function takes as inputs the shift operator $\mathbf{S}$, the filter coefficients $\mathbf{H}_k$ and the input signal $\mathbf{X}$. To further improve practical performance we add a bias term $\mathbf{B}$ to the filter operation. That is, we refine (5) with the opearation,

$$\mathbf{Z} = \sum_{k=0}^{K} \mathbf{S}^k \mathbf{X} \mathbf{H}_k + \mathbf{B}. \tag{8}$$

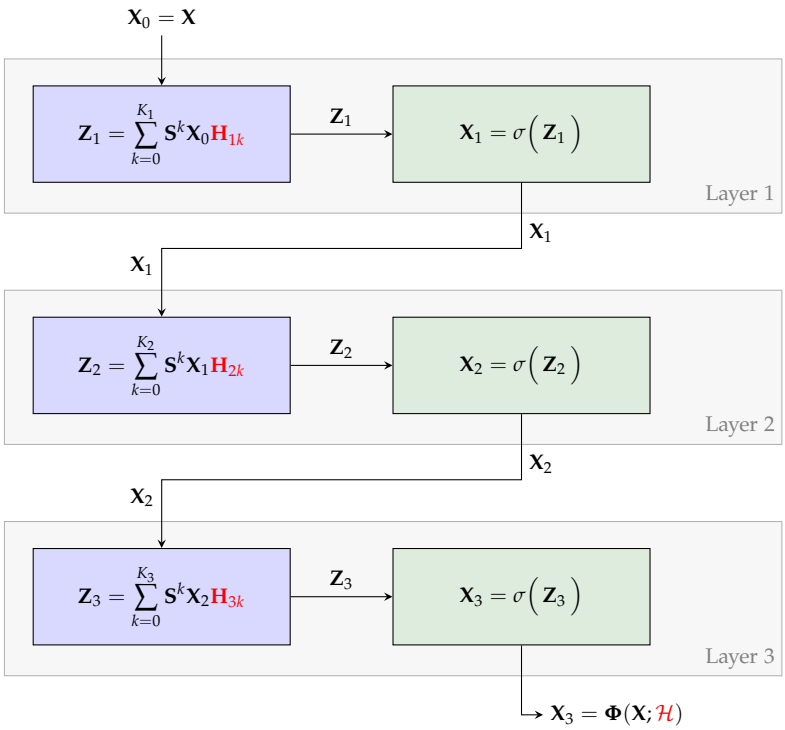

Figure 4. A Graph Neural Network (GNN) with three layers. A GNN is a composition of layers, each of which is itself the composition of a linear graph filter with a pointwise nonlinearity. [cf. (9)].

The bias $\mathbf{B}$ is also passed as a parameter to the filter function. ■

Task 4 Train a graph filter to predict movie ratings. Plot the evolution of the training loss and evaluate the loss in the test dataset. To obtain a good loss we need to experiment with the length of the filter – the number of filter taps K.

A graph filter is sometimes called a linear GNN. It is a GNN that does not use nonlinear operations.

3 Graph Neural Networks

Graph Neural Networks (GNNs) are information processing architectures made up of a composition of layers, each of which is itself the composition of a linear graph filter with a pointwise nonlinearity.

For a network with a given number of layers L we define the input output relationship through the recursion

$$\mathbf{X}_\ell = \sigma\Big(\mathbf{Z}_\ell \Big) = \sigma \left(\sum_{k=0}^{K_\ell} \mathbf{S}^k \mathbf{X}_{l-1} \mathbf{H}_{\ell k} \right), \tag{9}$$

In this recursion the output of Layer $\ell - 1$ is $\mathbf{X}_{l-1}$ and it is recast as an input to Layer ℓ. In this layer, the input $\mathbf{X}_{l-1}$ is processed with a graph filter to produce the intermediate output $\mathbf{Z}_\ell$. The coefficients of this graph filter are the matrices $\mathbf{H}_{\ell k}$. This intermediate output is processed with a pointwise nonlinearity σ to produce the output $\mathbf{X}_\ell$ of Layer ℓ. That the nonlinear operation is pointwise means that it is acting separately on each entry of $\mathbf{Z}_l$.

To complete the recursion we redefine the input $\mathbf{X}$ as the output of Layer 0, $\mathbf{X}_0 = \mathbf{X}$. The output of the neural network is the output of layer L, $\mathbf{X}_L = \mathbf{\Phi}(\mathbf{X}; \mathcal{H})$. In this notation $\mathcal{H}$ is the tensor $\mathcal{H} := [\mathbf{H}_{11}, \ldots, \mathbf{H}_{LK\ell}]$ that groups all of the filters that are used at each of the L layers.

A graph neural network with three layers is depicted in Figure 4.

3.1 Graph Neural Network Specification

To specify a GNN we need to specify the number of layers L and the characteristics of the filters that are used at each layer. The latter are the number of filter taps K_ℓ and the number of features F_ℓ at the output of the layer. The number of features F_0 must match the number of features at the input and the number of features F_L must match the number of features at the output. Observe that the number of features at the output of Layer $(\ell - 1)$ determines the number of features at the input of Layer ℓ. Then, the filter coefficients at Layer ℓ are of dimension $F_{\ell-1} \times F_\ell$.

Task 5 Program a class that implements a GNN with L layers. This class receives as initialization parameters a GNN specification consisting of the number of layers L and vectors $[K_1, \ldots, K_L]$ and $[F_0, F_1, \ldots, F_L]$ containing the number of taps and the number of features of each layer.

Endow the class with a method that takes an input feature $\mathbf{X}$ and produces the corresponding output feature $\mathbf{\Phi}(\mathbf{X}; \mathcal{H})$. ■

Task 6 Train a GNN to predict movie ratings. Plot the evolution of the training loss and evaluate the loss in the test dataset. To obtain a good loss we need to experiment with the number of layers and the number of filter taps per layer.

3.2 Mathematical Properties of Graph Neural Networks

In the following tasks, we will verify empirically three important mathematical properties of graph filters and GNNs: permutation equivariance, stability to graph perturbations, and transferability.

Task 7 To verify that graph filters and GNNs are equivariant to node relabelings, generate $\mathbf{S}' = \mathbf{P}\mathbf{S}\mathbf{P}^T$, where $\mathbf{P}$ is a permutation matrix. To maintain permutation equivariance you should also permute the entries of the input graph signal $\mathbf{x}$ as $\mathbf{x}' = \mathbf{P}\mathbf{x}$. Then run the trained GNN with $\mathbf{S}'$ and $\mathbf{x}'$. You should be able to verify that the output is just a permutation of the original output and that the test performance is exactly the same.

Task 8 To verify that graph filters and GNNs are stable to graph perturbations you should add or subtract a couple of edges to the original graph adjacency. Then run the trained GNN with $\tilde{\mathbf{S}}$, where $\tilde{\mathbf{S}}$ is the perturbed graph shift operator, and verify that the performance of the GNN is approximately the same.

Task 9 To verify that graph filters and GNNs are transferable you should run the trained GNN on a graph of larger size. This graph can be found in the provided dataset. Then you should be able to verify that although the graph filter or the GNN are trained on a small graph, execution on larger graphs does not significantly drop the performance.

4 Solution

A solution to this lab using Pytorch is available in https://gnn.seas.upenn.edu/wp-content/uploads/2023/06/GNN_lab.zip. The zip file includes the Jupyter Notebook of the lab and the data file required for execution.