1 Matrix Factorization and Recommender Systems

Required Reading:

1. Watch this video `https://www.youtube.com/watch?v=zzTbptEdKhY`
2. Chapter 9 from "Mining of Massive Datasets."

The Utility Matrix

In a recommendation system there are two classes of entities, which we will refer to as users and items. Users have preferences for certain items, and these preferences must be teased out of the data. The data is represented as a utility matrix, giving for each user-item pair, a value that represents what is known about the degree of preference of that user for that item.

Example: In Table 1 we see an example utility matrix, representing users' ratings of movies on a 1-5 scale, with 5 the highest rating. Blanks represent the situation where the user has not rated the movie. The movie names are HP1, HP2, and HP3 for Harry Potter 1, 2, and 3, TW for Twilight, and SW1, SW2, and SW3 for Star Wars episodes 1, 2, and 3. The users are represented by capital letters A through D.

	HP1	HP2	HP3	TW	SW1	SW2	SW3
A	4		5	1			
B	5	5	4				
C				2	4	5	
D			3				3

Table 1: A utility matrix representing ratings of movies on a 1-5 scale

Most user-movie pairs have blanks, meaning the user has not seen/rated the movie. In practice, the matrix would be much sparser, with the typical user rating only a tiny fraction of all available movies. The goal of a recommendation system is to predict the blanks in the utility matrix. For example, would user A like the movie SW2? In most real systems the goal is to discover some entries in each row that are likely to be high, then take the top C of these items to display as recommendations.

1.1 Populating the Utility Matrix

Without a utility matrix, it is almost impossible to recommend items. Acquiring data from which to build a utility matrix is often difficult. There are two general approaches to discovering the value users place on items.

1. Ask users to rate items (aka *explicit rating*). Movie ratings are generally obtained this way, and some online stores try to obtain ratings from their purchasers. This approach is limited in its effectiveness, since generally users are unwilling to provide responses, and the information from those who do may be biased by the very fact that it comes from people willing to provide ratings.

2. Make inferences from users' behavior (aka *implicit rating*). If a user buys a product at Amazon, watches a movie on YouTube, or reads a news article, etc., then the user can be said to "like" this item, regardless of if they actually end up rating it or not.

1.2 Utility Matrix for Implicit Ratings

In the case of implicit ratings, the Utility Matrix has just one non-blank value, 1 means that the user likes the item (watch a movie / video, viewed or purchased a product).

If in Table 1 instead of having ratings (in scale 1-5) we had the information of whether a user watched a particular movie, our initial utility matrix would look like Table 2. In this case, Table 2 would represent an implicit utility matrix.

For the case of implicit ratings, we could also consider the matrix in Table 3, where blanks are replaced by 0's, but this *assumes missingness is truly reflective of negative ratings.* (Does this make sense? In what cases yes, in what cases no?)

To moderate that assumtion, another option is to sample 0s from the missing values while still leaving some blank (as shown in Table 4).

	HP1	HP2	HP3	TW	SW1	SW2	SW3
A	1		1	1			
B	1	1	1				
C				1	1	1	
D			1				1

Table 2: A utility matrix representing movies watched by users

	HP1	HP2	HP3	TW	SW1	SW2	SW3
A	1	0	1	1	0	0	0
B	1	1	1	0	0	0	0
C	0	0	0	1	1	1	0
D	0	0	1	0	0	0	1

Table 3: A utility matrix representing movies watched by users where all missing values are filled as negatives (0s).

	HP1	HP2	HP3	TW	SW1	SW2	SW3
A	1	0		1	1		0
B	1	1	1	0	0		0
C	0		1	1	1		0
D		0	1		0		1

Table 4: A utility matrix representing movies watched by users where a sample of missing values are filled as negatives

Matrix Factorization

Collaborative filtering recommends items by matching users with other users with similar interests. The items recommended to a user are those preferred by similar users. Collaborative filtering systems are usually categorized into two subgroups: memory-based and model-based. Memory-based methods memorize the utility matrix and make recommendations based on the relationship between the queried user and the rest of the utility matrix often using a distance and K-nearest neighbours. Model-based methods fit a parameterized model to the given utility matrix and then make recommendations based on the model. Here we introduce a model based approach Matrix Factorization.

The matrix factorization algorithm approximates the utility matrix as a product of two long, thin matrices. This view makes sense if there are a relatively small set of features of items and users that determine the rating.

We denote the utility matrix Y with elements $\{y_{ij}\}$, where $i \in 1, \ldots, n_u$ $j \in 1, \ldots, n_m$ and n_u is the number of users while n_m is the number of items. The decomposition has the equation $\hat{Y} = UV^T$ where U is $n_u \times K$ and V is $n_m \times K$. Every rating y_{ij} gets the prediction

$$\hat{y}_{ij} = u_i \cdot v_j$$

where u_i is the K-dimensional user embedding vector and v_j is the K-dimensional item embedding vector. Writing this out fully looks something like:

$$\begin{bmatrix} 5 & 1 & 4 & 5 & 1 \\ 5 & 2 & 1 & 4 & \\ 1 & 4 & 1 & 1 & 2 \\ 4 & 1 & 5 & 5 & 4 \\ 5 & 3 & 3 & & 4 \\ 1 & 5 & 1 & 1 & 1 \\ 5 & 1 & 5 & 5 & 4 \end{bmatrix} \approx \begin{bmatrix} u_{11} & u_{12} & \ldots & u_{1K} \\ u_{21} & u_{22} & \ldots & u_{2K} \\ u_{31} & u_{32} & \ldots & u_{3K} \\ u_{41} & u_{42} & \ldots & u_{4K} \\ u_{51} & u_{52} & \ldots & u_{5K} \\ u_{61} & u_{62} & \ldots & u_{6K} \\ u_{71} & u_{72} & \ldots & u_{7K} \end{bmatrix} \times \begin{bmatrix} v_{11} & v_{21} & v_{31} & v_{41} & v_{51} \\ v_{12} & v_{22} & v_{32} & v_{42} & v_{52} \\ \vdots & & & & \vdots \\ v_{1K} & v_{2K} & v_{3K} & v_{4K} & v_{5K} \end{bmatrix}$$

Here is a toy example in witch we have 7 users and 5 movies, and the choice of the number of parameters is dictated by $K = 2$.

$$\begin{bmatrix} 5 & 1 & 4 & 5 & 1 \\ 5 & 2 & 1 & 4 & \\ 1 & 4 & 1 & 1 & 2 \\ 4 & 1 & 5 & 5 & 4 \\ 5 & 3 & 3 & & 4 \\ 1 & 5 & 1 & 1 & 1 \\ 5 & 1 & 5 & 5 & 4 \end{bmatrix} \approx \begin{bmatrix} 0.2 & 3.4 \\ 3.6 & 1.0 \\ 2.6 & 0.6 \\ 0.9 & 3.7 \\ 2.0 & 3.4 \\ 2.9 & 0.5 \\ 0.8 & 3.9 \end{bmatrix} \times \begin{bmatrix} 0.0 & 1.5 & 0.1 & 0.0 & 0.7 \\ 1.3 & 0.0 & 1.2 & 1.4 & 0.7 \end{bmatrix}$$

We can gain insight into what these matrices represent by looking more closely at the estimated parameters (sometimes called "loadings") in terms of either the users or movies. U is the representation of users in some low dimensional space. For user i, the vector u_i is the representation of user i in terms of types of user preferences. It would have high magnitude loadings for underlying concepts which that user either extremely likes/dislikes (for example, "likes romance", "likes comedy", or more complicated and nuanced preferences). Similarly

or an item j, v_j is the representation of the items in the same low dimensional space. An item will have a high magnitude loading if it is representative of the underlying concept ("has romance themes", "is comedic," etc.).

1.3 Computing similar items and similar users

The row vector $v_j = (v_{j1}, \ldots, v_{jK})$ can be interpreted as a learned feature vector for item j. If we now wanted to find items closely related to j we can look for items that v_ℓ that have small Euclidean distance to v_j. That is, we want

$$\|v_j - v_\ell\|$$

to be small. We can identify similar users in the same fashion by using vectors u_i and u_k.

1.4 Loss Function

Many loss functions can work for this kind of model. The typical choice is the mean squared error (MSE) for observed values of the utility matrix. Let r_{ij} be 1 if the element y_{ij} of the utility matrix is non-empty and 0 otherwise. Then the MSE loss is:

$$E(u, v) = \frac{1}{N} \sum_{(i,j):r_{ij}=1} (y_{ij} - u_i v_j)^2$$

where N is the number of non-blank elements of the utility matrix Y.

1.5 Gradient descent with momentum

We can minimize the MSE using gradient descent.

Recall in gradient descent if we are trying to learn some weights w and we have some error function $E(w)$, at each iteration we update w as a small step into the direction of the negative gradient

$$w_{t+1} = w_t - \eta \nabla E(w_t)$$

where the parameter η is known as the learning rate.

A popular version of gradient descent is gradient descent with momentum. Instead of using the gradient directl, we can use a "moving average" of our gradients by incorporating a momentum term. Here is the modified gradient with momentum:

$$v_t = \beta v_{t-1} + (1 - \beta) \nabla E(w_t)$$

The updating equation for the model parameter becomes:

$$w_{t+1} = w_t - \eta v_t$$

1.6 Gradient descent for Matrix Factorization

We are trying to minimize mean square error:

$$E(U,V) = \frac{1}{N} \sum_{(i,j):r_{ij}=1} (y_{ij} - u_i \cdot v_j)^2$$

where N is the number of non-blank elements of Y and $u_i \cdot v_j = \sum_{s=1}^{K} u_{is} v_{js}$. Consider the derivative of E with respect to $u_{\ell k}$ or v_{jk}. Here i and ℓ are user indices, j is an item index, and k is an index for which dimension of the low-dim. embedding we're referring to.

Then the general from of GD equations look like:

$$\frac{\partial E}{\partial u_{\ell k}} = -\frac{2}{N} \sum_{j:r_{\ell j}=1} (y_{\ell j} - u_\ell \cdot v_j) \frac{\partial E}{\partial u_{\ell k}} \quad (1)$$

$$\frac{\partial E}{\partial v_{jk}} = -\frac{2}{N} \sum_{i:r_{ij}=1} (y_{ij} - u_i \cdot v_j) \frac{\partial E}{\partial v_{jk}} \quad (2)$$

To see why for (1), note that

$$\frac{\partial E}{\partial u_{\ell k}} = \frac{\partial (u_{ij} \cdot v_j)}{\partial u_{\ell k}} = \begin{cases} v_{jk}, & \text{if } \ell = i \\ 0, & \text{otherwise} \end{cases}$$

Therefore the gradient descent updating equation for (1) is:

$$u_{\ell k} \leftarrow u_{\ell k} + \frac{2\eta}{N} \sum_{j:r_{\ell j}=1} (y_{\ell j} - u_\ell \cdot v_j) v_{\ell k}$$

and a similar parallel argument gives for (2):

$$v_{jk} \leftarrow v_{jk} + \frac{2\eta}{N} \sum_{i:r_{ij}=1} (y_{ij} - u_i \cdot v_j) u_{ik}$$

For a full derivation, see the detailed notes called "Details of Gradient Descent for Matrix Factorization."

1.7 Vectorizing gradient descent

For fast implementation of gradient decent, let's vectorize the updating equations.

Let R be the $n_u \times n_m$ matrix with elements $\{r_{ij}\}$. Let $\Delta = (Y - UV^T) \otimes R$, where the operation $\otimes$ represents elementwise matrix multiplication. Let $\dfrac{\partial E}{\partial U}$ and $\dfrac{\partial E}{\partial V}$ denote the matrix with elements $\dfrac{\partial E}{\partial u_{ik}}$ and $\dfrac{\partial E}{\partial v_{jk}}$ respectively. Then we can write the matrix of partial derivatives with the following equations:

$$\frac{\partial E}{\partial U} = -\frac{2}{N} \Delta \cdot V$$

$$\frac{\partial E}{\partial V} = -\frac{2}{N}\Delta^T \cdot U$$

Exercise: Verify that these equations are equivalent to equations 1 & 2 by showing that the element (i, k) from a matrix multiplication is the dot product of the i row on the first matrix and the k column of the second matrix.

1.8 Avoiding overfitting

We can add a regularization term to our error function.

$$\frac{1}{N}\sum_{(i,j):r_{ij}=1}(y_{ij} - u_i v_j)^2 + \lambda\left(\sum_{i=1}^{n_u}\sum_{k=1}^{K} u_{ik}^2 + \sum_{i=1}^{n_m}\sum_{k=1}^{K} v_{ik}^2\right)$$

Where $N = \sum_{ij} r_{ij}$

Exercise. How would the gradient descent equations change? Derive the new updating equations with this regularization implemented.

Interview Questions

1. What types of recommender systems exist and can you describe them?
2. How would you evaluate a recommender system?
3. How would you design a recommender system for scientists on mendeley.com, a website for research articles?
4. What is the "cold start" phenomenon?
5. How would you design a recommendation system for a marketplace like Amazon?
6. How would you build a recommendation system for an entertainment platform similar to Netflix?

References

[1] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of Massive Datasets.* Cambridge University Press, New York, NY, USA, 2nd edition, 2014.

2 Details of Gradient Descent for Matrix Factorization

Set Up:

Start with a utility matrix $Y \in \mathbb{R}^{n_u \times n_m}$.

EX: Ratings of items:

If there are n_u users and n_m items, then Y looks like the inside array of this table:

	Item 1	$\cdots$	Item j	$\cdots$	Item n_m
user 1	y_{11}	$\cdots$		$\cdots$	
$\vdots$		$\ddots$			
user i		$\cdots$	y_{ij}	$\cdots$	
$\vdots$				$\ddots$	
user n_u		$\cdots$		$\cdots$	$y_{n_u n_m}$

In most cases this matrix is really sparse. (Why?)

Goal for Matrix Factorization:

Find "skinny" matrices U & V such that $Y \approx UV^T$.

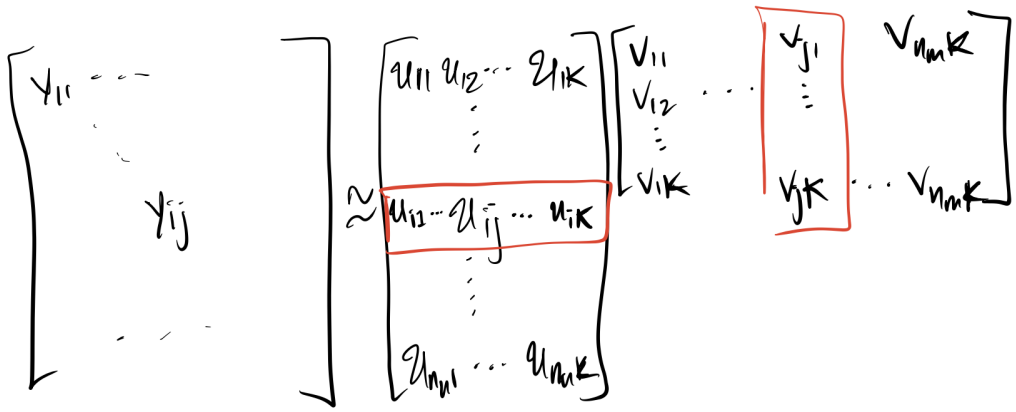
i.e:

$$Y \in \mathbb{R}^{n_u \times n_m}, \quad U \in \mathbb{R}^{n_u \times K}, \quad V^T \in \mathbb{R}^{K \times n_m}.$$

If K is small, then the matrix is "skinny" ... dimension reduction is happening!

Matrix Factorization Model:

$$\hat{y}_{ij} = (UV^T)_{ij} = \langle u_i, v_j \rangle = \sum_s u_{is} v_{js} \quad \leftarrow \text{dot product of } i\text{th row of } U \text{ and } j\text{th col of } V^T.$$



Parameters:

What are the parameters of this model? The elements of U & V.
How many parameters are there in this model? $(n_u + n_m) \times K$
How to find the optimal values of parameters?
Minimize some loss, denoted $E(U, V)$.
Let

$$r_{ij} = \begin{cases} 1 & \text{if } y_{ij} \text{ is not empty} \\ 0 & \text{else} \end{cases}$$

and $N = \sum_{i,j} r_{ij}$, i.e. the number of non-empty elements of Y.
Then we can define a squared error loss for the matrix factorization as:

$$E(U, V) = \frac{1}{N} \sum_{(i,j):r_{ij}=1} (y_{ij} - (UV^T)_{ij})^2.$$

Reminder: Useful Derivative Properties

Additivity:

$$(f + g)'(x) = f'(x) + g'(x).$$

Chain Rule for Composition:

$$(f \circ g)'(x) = f'(g(x)) \cdot g'(x).$$

Ex: $L(x) = (y - f(x))^2$

$$L'(x) = -2(y - f(x)) \cdot f'(x).$$

Back to Matrix Factorization

Our loss function for the MF model is:

$$E(U, V) = \frac{1}{N} \sum_{(i,j):r_{ij}=1} (y_{ij} - \sum_s u_{is} v_{js})^2.$$

Let's find the updating equations for elements of U & V, one at a time.
We will need:

$$\frac{\partial E}{\partial u_{ik}} \quad \text{and} \quad \frac{\partial E}{\partial v_{jk}} \quad \forall i, j.$$

Intermediate Result:

Recall $\hat{y}_{ij} = (UV^T)_{ij} = \sum_s u_{is} v_{js}$.
Then:

$$\frac{\partial \hat{y}_{ij}}{\partial u_{lk}} = \begin{cases} 0 & \text{if } \ell \neq i \\ v_{jk} & \text{if } \ell = i. \end{cases}$$

(i & ℓ are user indices.)

Illustrative Ex: If $K = 2$:

$$\hat{y}_{34} = u_{31}v_{41} + u_{32}v_{42}.$$

If we use $\ell = 5$, we see:

$$\frac{\partial \hat{y}_{34}}{\partial u_{51}} = \frac{\partial}{\partial u_{51}}(u_{31}v_{41} + u_{32}v_{42}) = 0 + 0 = 0.$$

If we use $\ell = 3$, we see:

$$\frac{\partial \hat{y}_{34}}{\partial u_{31}} = \frac{\partial}{\partial u_{31}}(u_{31}v_{41} + u_{32}v_{42}) = v_{41} + 0 = v_{41}.$$

So we see that $\dfrac{\partial \hat{y}_{ij}}{\partial u_{\ell k}} = v_{jk}$ only when $i = \ell$, if else, it's 0.

Back to gradient descent for the main loss function:

We know:

$$\frac{\partial E}{\partial u_{ik}} = \frac{1}{N} \sum_{(i,j):r_{ij}=1} \frac{\partial}{\partial u_{ik}}(y_{ij} - \hat{y}_{ij})^2.$$

$$= \frac{1}{N} \sum_{(i,j):r_{ij}=1} [-2(y_{ij} - \hat{y}_{ij}) \times \frac{\partial \hat{y}_{ij}}{\partial u_{\ell k}}].$$

$$= -\frac{2}{N} \sum_{j:r_{ij}=1} (y_{ij} - \hat{y}_{ij}) \cdot v_{jk}.$$

Therefore, the updating equation for the parameters of U is:

$$u_{\ell k} \leftarrow u_{\ell k} + \frac{2\eta}{N} \sum_{j:r_{\ell j}=1} (y_{\ell j} - \hat{y}_{\ell j}) v_{jk}.$$

Exercise: Find the updating equation for v_{jk}.

A similar derivation for v_{jk} gives:

$$v_{jk} \leftarrow v_{jk} + \frac{2\eta}{N} \sum_{i:r_{ij}=1} (y_{ij} - \hat{y}_{ij}) u_{ik}.$$

Vectorizing Gradient Descent for Matrix Factorization

Recall from above, the GD updating equations are:

$$u_{ik} \leftarrow u_{ik} + \frac{2\eta}{N} \sum_{j:r_{ij}=1} \left(y_{ij} - u_i \cdot v_j\right) v_{jk},$$

$$v_{jk} \leftarrow v_{jk} + \frac{2\eta}{N} \sum_{i:r_{ij}=1} \left(y_{ij} - u_i \cdot v_j\right) u_{ik},$$

& R is the matrix with $r_{ij} = 1$ if y_{ij} not empty, 0 else.

Then define Δ as

$$\Delta = \left(Y - U \cdot V^T\right) \otimes R.$$

where $\otimes$ represents elementwise matrix multiplication. **What is Δ_{ij}?**

$$\Delta_{ij} = \left(y_{ij} - u_i \cdot v_j\right) r_{ij}.$$

Define the gradient matrices:

$$\frac{\partial E}{\partial U} = \text{ the matrix w/ elements } \frac{\partial E}{\partial u_{ik}}, \qquad \frac{\partial E}{\partial V} = \text{ the matrix w/ elements } \frac{\partial E}{\partial v_{jk}}.$$

Claim:

$$\frac{\partial E}{\partial U} = -\frac{2}{N} \Delta V \quad \text{and} \quad \frac{\partial E}{\partial V} = -\frac{2}{N} \Delta^T U.$$

Proof: Let's show equality element-by-element.

$$\left(\frac{\partial E}{\partial U}\right)_{ik} = \frac{\partial E}{\partial u_{ik}} = -\frac{2}{N} \sum_j \Delta_{ij} v_{jk} \quad \text{(def of mat. mult.)}$$

Plug in Δ_{ij}:

$$\left(\frac{\partial E}{\partial U}\right)_{ik} = -\frac{2}{N} \sum_j \left(y_{ij} - u_i \cdot v_j\right) r_{ij} v_{jk} = -\frac{2}{N} \sum_{j:r_{ij}=1} \left(y_{ij} - u_i \cdot v_j\right) v_{jk}.$$

That's all! We have shown how to vectorize the gradient descent updates for matrix factorization. A parallel argument goes for $\dfrac{\partial E}{\partial V}$.

3 Interview Questions

1. What types of recommender systems exist and can you describe them?
2. How would you evaluate a recommender system?
3. How would you design a recommender system for scientists on mendeley.com, a website for research articles?
4. What is the "cold start" phenomenon?
5. How would you design a recommendation system for a marketplace like Amazon?
6. How would you build a recommendation system for an entertainment platform similar to Netflix?

References

[1] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of Massive Datasets*. Cambridge University Press, New York, NY, USA, 2nd edition, 2014.