An Introduction to Matrix factorization and Factorization Machines in Recommendation System, and Beyond

Yuefeng Zhang
yuefeng.zhang@pku.edu.cn

Abstract
This paper aims at a better understanding of matrix factorization (MF), factorization machines (FM), and their combination with deep algorithms’ application in recommendation systems. Specifically, this paper will focus on Singular Value Decomposition (SVD) and its derivations, e.g Funk-SVD, SVD++, etc. Step-by-step formula calculation and explainable pictures are displayed. What’s more, we explain the DeepFM model in which FM is assisted by deep learning. Through numerical examples, we attempt to tie the theory to real-world problems.

Contents

1 Introduction and Overview 2
   1.1 Problem Definition .................................................. 2
   1.2 Matrix factorization (MF) ............................................. 2
      1.2.1 MF for different data sets ..................................... 2
      1.2.2 Comparison between MF and collaborative filtering algorithms ... 3

2 Matrix factorization (MF) 3
   2.1 Traditional SVD ......................................................... 3
      2.1.1 Example - Complete Matrix ................................... 6
      2.1.2 Example - Incomplete Matrix ................................... 6
      2.1.3 Example - Numerical Decomposition of Matrices .......... 7
   2.2 Funk-SVD ................................................................. 8
   2.3 SVD++ ................................................................. 10

3 Factorization Machine (FM) 11
   3.0.1 Field-Aware Factorization Machine (FFM) ................. 13
   3.1 With Deep Learning .................................................. 13
      3.1.1 DeepFM .......................................................... 13
      3.1.2 Model Ensemble ................................................. 14
      3.1.3 Optimization Algorithms ..................................... 13

We would like to thank Gaurav Khanna for the valuable comments and help.
1 Introduction and Overview

General overview of MF and SVD algorithms' usage in the field of recommendation system.

1.1 Problem Definition

To start understanding matrix factorization for recommender systems, we define the following matrices: a rating matrix $R$ which is in general an $R \in \mathbb{R}^{m \times n}$ matrix. Each row of $R$ represents a user and each column represents an item. The values $R_{ui}$ represent the rating (or score) that user $u$ has given to item $i$.

The problem we are trying to solve is as follows: For any collection of users and items, we have the situation where not all users have given ratings to all the items. In fact, if examined the values of $R$, we would find that several values are incomplete. However, we wish to recommend certain items to users. How do we do that? One way is to simply recommend any item the user has not already rated to them; this could mean sending a message to the user daily with a recommendation for an item. However, we quickly see the issue with this; if we keep recommending items to users that they don’t like, they will very quickly start ignoring our messages. We would lose credibility with the user and they may eventually unsubscribe from our messages.

So then we have to ask, “What are the items a user may like?” This is equivalent to asking, “Of the items the user has not yet rated, what is our best guess for their ratings by a user?” If we could come up with a system to guess this well, then we could send a daily email to the user recommending items which our system has predicted the user would rate highly.

Because the user will not rate all items, we expect missing values in this matrix. Our goal is to fill up those values. We use $\hat{R}$ to represent the predicted matrix according to our algorithm.

There are two kinds of rating forms: implicit rating and explicit rating. Take movie recommendation as an example:

1. Implicit: We only know the historical data whether the user watch the movie or not. The data can be represented as 0 and 1 format that 0 means un-watch and 1 means watch.

2. Explicit: Users explicitly rate all the movie they watch in the past. The data can be represented in discrete range, e.g 1, 2, ···, 5.

1.2 Matrix factorization (MF)

One technique we can try is to use historical data to do recommendation. The most famous method for this problem is collaboration filter algorithm: neighborhood methods and Latent Factor Models (LFM). Matrix factorization is a successful implementation of the latter method. In fact, LFM and MF are talking about the same thing: how to auto-complete the score matrix by using dimension reduction methods.

1.2.1 MF for different data sets

There are two types of recommendation data set: explicit representing feedback data and implicit representing feedback data. The former one has score data while the later one only contains users’ preference instead of numeric representation. So the implicit representing feedback data set only knows what users like but doesn’t know what they hate.

MF or LFM can perform great on the former data set because they are like machine learning’s regression problem. For the later data set, we normally treat it as the top-$N$ recommendation problem and need reconstruct input data which can fit the algorithm. So we need to create negative data. There are two rules to construct negative samples for users:

1. Make the number of positive and negative samples balanced.
2. Prior to choose those popular items which aren’t liked by the user. Because in this situation it’s more likely for this user to unlike this particular popular item. The ratio of negative sample number to positive sample number usually set as 3 in real problems. Experiments show that this ratio has effects in exploring recommendation system’s long tail problem.

1.2.2 Comparison between MF and collaborative filtering algorithms

Factorization machine and collaborative filtering algorithms are two major kinds of recommendation algorithms. There are two kinds of collaborative filtering algorithms: user-based and item-based. User-based collaborative filtering algorithm is to recommend the items which similar users have liked while item-based one is to recommend similar items according to users’ part behaviour. Because internet products usually have much larger user number than item number, they prefer to use the item-based collaboration to the user-bases one in their recommendation system considering the computation complexity.

The biggest difference between MF and collaborative filtering (CF) algorithm is that the former one has the learning process and the latter one is based on statistics method. We are going to compare MF and collaborative filtering algorithms from the following aspects. Suppose we have $M$ users and $N$ items and the MF’s latent factor dimension is $F$.

- **Offline Computation Space Complexity**: For user-based CF, space complexity $O(M \cdot M)$; for item-based CF, it is $O(N \cdot N)$. For MF, the space complexity is $O(F \cdot (M + N))$. Since $F$ is far less then $M$ or $N$, so MF method saves more memory,

- **Offline Computation Time Complexity**: Suppose we have $K$ records of users’ rating on items. For user-based CF, time complexity is $O(N \cdot (N/K)^2)$; for item-based CF, it’s $O(M \cdot (M/K)^2)$. Suppose we have $S$ iterations in MF, then its time complexity should be $O(S \cdot F \cdot K)$.

- **Online recommendation Time Complexity**: For CF algorithms, it can be done in real-time. In contrast, MF’s recommendation process is time-consuming that its time complexity for generating a new recommendation result list is $O(F \cdot M \cdot N)$. So it’s hard for MF method to do real-time recommendations. Nowadays, MF recommendation methods is usually divided into two steps: recall and rank. Recall is to select hundreds of potential items from the whole data set and rank is to finely rank those potential items into ordered recommendation list.

2 Matrix factorization (MF)

2.1 Traditional SVD

SVD[1] was first used by Billsus and Pazzani in 1998 in recommendation service. They explored a few machine learning algorithms for collaborative filtering tasks and identified SVD as the best-performing one.

Eigendecomposition is used for square matrix and for non-square matrix we have SVD [2]. Assume we have $m$ users and $n$ items then score matrix will be $R \in \mathbb{R}^{m \times n}$. First, we complete missing values in score matrix as $R'$, e.g. using global average value or user/item average value. Then, we use SVD to decompose $R'$ as following[1]:

$$R' = USV^T$$ (1)

$U \in \mathbb{R}^{m \times m}$ and $V \in \mathbb{R}^{n \times n}$ are unitary matrices, $S \in \mathbb{R}^{k \times k}$ is diagonal matrix which $k \leq \min(m, n)$ is the rank of $R'$ and values on its diagonal are the singular values of matrix $R$ in descending order.

We can try to understand SVD in the aspect of geometric transformation. Let’s think matrix $R'$ is transformation operation. The SVD factors one transformation matrix into three: (orthogonal) x (diagonal) x (orthogonal) which can be viewed from geometric aspect: (rotation) x (stretching) x (rotation). Figure shows this process. $USV^T x$ starts with the rotation to $V^T x$. Then $S$ stretches that vector to $SV^T x$, and $U$ rotates to $Ax = USV^T x$. 

3
In order to do decomposition, we can choose the \( f \) singular values to compose diagonal matrix \( S_f \) and find corresponding rows and columns of those \( f \) singular values in \( U, V \) as \( U_f, V_f \). Suppose the rank of \( S \) is \( r \). Only \( r \) values in matrix \( S \) are not empty. Values on \( S \)'s diagonal are in descending order which mean \( s_{1,1} \leq s_{2,2} \leq \cdots \leq s_{r,r} \). We choose the \( f \) by calculating the power of matrix \( S_f \) to satisfy the equation 2.

\[
\sum_{k} S_{i,k}^2 \leq \sum_{i} S_{i,i}^2 \times 100\% \geq \text{threshold}
\]

Besides this heuristic method of choosing the rank \( k \), there is a rule of thumb: choose \( k \) such that the sum of the top \( k \) singular values is at least \( c \) times as big as the sum of the other singular values, where \( c \) is a domain-dependent constant (like 10, say).

\[
\sum_{k} S_{k} \approx 10 \times \sum_{f+1} S_{k}
\]

Then we can represent the score matrix after dimension decomposition as equation 4.

\[
R^*_f = U_f S_f V_f^T
\]

\( R^*_f(u, i) \) is the forecast rating score of user \( u \) to item \( i \) using decomposition representation. Equation 4 is the final representation of SVD usage in recommendation system. Figure 2 explains how SVD works. By using \( U_f, S_f, V_f \) we can learn latent user’s preference in an efficient way.

We can use SVD to decompose input matrix and utilize item-base collaboration method to recommend. The reason why we prefer item-base method to user-base method here is that for real applications user number is always larger than item number.
The $R_i$ column vector represents all of the ratings from users for item $i$. Similarly, $R_j$ represents all of the users rating for item $j$. We want to get a sense of whether items $i$ and $j$ are “similar”. One way to measure this is to see how “close” these vectors are to one another. For example, if all the users gave both items $i$ and $j$ high ratings (i.e., “4” or “5”), then we would say that items $i$ and $j$ are similar. From a vector point of view, these two vectors would likely be close to one another in $R_1 \times n$. However, if all the users gave item $i$ a very high ratings (all “5”s) but gave item $j$ very bad ratings (all “1”), then we might say that items $i$ and $j$ are not similar. From a vector point of view, these two vectors would be far away from one another in $R_1 \times n$.

To express the similarity in mathematical terms, we can utilize the dot product between two vectors. We can use Euclidean distance, Manhattan distance etc to measure the similarity. Here we choose the cosine similarity as the similarity metric. Concretely, we can define the cosine similarity between two items as

$$\cos(R_i, R_j) = \frac{R_i \cdot R_j}{\|R_i\|\|R_j\|}$$

(5)

When calculating the similarity score between $R_i^*$ and $R_j^*$, for the item-based collaboration, we only consider those values which aren’t empty in input rating matrix. So we define a mask matrix, $mask_{ij}$ which has the following values.

$$mask_{ij} = \begin{cases} 
1, & \text{if } R_{ij} \text{ isn’t empty} \\
0, & \text{if } R_{ij} \text{ is empty} 
\end{cases}$$

(6)

Essentially, this matrix places a value of 1 in positions where the original rating matrix, $R$, had a value; similarly, it places a value of zero where $R$ had a missing value.

For each empty value in rating matrix $R$, we calculate $\hat{R}_{u,j}$ based on user $u$’s rating of item $j$ (the rest of items) and the similarity score between item $i$ and item $j$. The process can be showed in equation 7.

$$\text{similarity}_{total} = \sum_{j \in n, j \neq i} \cos(R_i^* \circ mask_i, R_j^* \circ mask_j)$$

$$\hat{R}_{u,i} = \sum_{j \in n, j \neq i} \frac{\cos(R_i^* \circ mask_i, R_j^* \circ mask_j)}{\text{similarity}_{total}} \cdot R_j^*$$

(7)

where $R_i$ is matrix $R$’s $i$th column vector, $\circ$ represents for the element-wise product (Hadamard product).

Recall that the Hadamard Product between two matrices is defined as

$$[A \circ B]_{ij} = [A]_{ij} [B]_{ij}$$

(8)

for all $1 \leq i \leq m$ and $1 \leq j \leq n$. In order for this product to be defined, the matrices $A$ and $B$ have to be the same dimension but not necessarily square. We should also emphasize that the end result of the Hadamard product is a matrix of the same size. Therefore, the Hadamard product between two $\mathbb{R}^{1 \times n}$ vectors is also a $\mathbb{R}^{1 \times n}$ vector, and not a scalar as we would expect from a dot product.

Now we get the predicted matrix $\hat{R}$ by using SVD decomposition and item-base collaboration filter. The reason why we need the traditional SVD algorithm in recommendation is we want to do decomposition. Solving real world recommendation problems, we will face thousands and millions of users and items to be recommended. It’s a huge challenge for us to solve this problem in limited storage resources and computation resources. Matrix decomposition methods, e.g traditional SVD, help us to achieve the goal within those restrictions.

We may ask what do the $U$, $S$, and $V$ represent physically? Do they have any physical significance? For example each element in $R_{ui}$ represents the rating that user $u$ gave to item $i$. Can we say the same for the elements in $U$ and $V$? The meaning of $U$, $S$, $V$ - they do not have a physical analogy like the matrix $R$. 


The values in the matrix can be negative, e.g. the numeric example in section 2.1.2 so that alone tells us that they are not representing any “rating”. But they can still be explained in an intuitive way.

Every row of the matrix $U$ represents user’s relationship with the particular category that every value means the category’s importance (or relativity), the bigger the value is the more related they are. The matrix $U$’s column number is the category number we mention here and $U$’s row number is the number of users. Every column of the matrix $V$ represents the item’s relationship with the particular category. The matrix $V$’s row number is the category number we mention here and $V$’s column number is the number of users. Matrix $S$’s values on the diagonal represent the relationship between users’ category and items’ category, the bigger the value is the more related they are. To better explain this part, especially what the category means, we will revisit this through the numerical example after equation 9.

2.1.1 Example - Complete Matrix

Show example of how reducing dimensationality (only selecting $f = 2$ can recreate an approximation of the original $R$).

2.1.2 Example - Incomplete Matrix

We’ll first show an example of the power of SVD.

Figure 3: SVD example.

Two different ways to complete the empty values:

1. **Use global average**: use the average value of all the existing values in rating matrix $R$. In our example, global average is calculated by

$$\frac{r_{1,1} + r_{1,2} + r_{1,4} + r_{2,1} + \cdots r_{4,4}}{\text{number non-empty}} = \frac{1 + 3 + 4 + 5 + \cdots + 5}{11} = \frac{37}{11} = 3.36$$

2. **Use user average**: use the average value of the existing ratings for the specific user by averaging across the rows for that user in matrix $R$. For user 1, the empty rating value can be calculated by

$$r_{1,3}^* = \frac{r_{1,1}r_{1,2} + r_{1,4}}{3} = \frac{1 + 3}{3} = 2.67$$

Suppose we choose to use user average value to fill in empty values. We can set $f$ as 2. We can calculate the singular power percentage as equation 2 which should be

$$(14.59^2 + 3.22^2)/(14.59^2 + 3.22^2 + 1.11^2 + 0.23^2) \times 100\% = 99.42\%$$
Then the reconstructed matrix from rank 2 the \( R^* \) will be:

\[
R^* = U_2 S_2 V_2^T
\]

\[
= \begin{pmatrix}
-0.37 & 0.67 \\
-0.64 & -0.15 \\
-0.28 & -0.72 \\
-0.62 & 0.08
\end{pmatrix}
\begin{pmatrix}
14.59 & 0 \\
0 & 3.22
\end{pmatrix}
\begin{pmatrix}
-0.51 & -0.51 & -0.47 & -0.51
\end{pmatrix}
\]

\[
= \begin{pmatrix}
0.98 & 2.87 & 2.96 & 3.88 \\
5.14 & 4.7 & 4.32 & 4.46 \\
3.94 & 1.88 & 1.45 & 0.76 \\
4.39 & 4.60 & 4.33 & 4.71
\end{pmatrix}
\]

(9)

Let’s recall the SVD’s physical explanation. First we want to be clear with a new concept category which means the latent classes we want to project users’ information or items’ information into. In the SVD equation, the category for user is the column number of \( U \) and the category for item is the column number of \( V \). In \( R^* \)’s reconstruction, we set the \( f \) to 2 so the category we use to project the users’ and items’ latent information is 2. For example, the 1st row and 1st column value \(-0.37\) of \( U_2 \) represents for user 1’s relationship with user category 1. The the 1st row and 1th column value 0.98 of \( V_2^T \) represents for user 1’s relationship with item category 1. And first value 14.59 on \( S_2 \)’s diagonal means the importance between user category 1 and item category 2.

Then we can combine the SVD decomposition with item-based collaboration to do predictions. Here we will explain how to predict one empty value detailedly. We can get the mask matrix according to equation (10).

\[
mask = \begin{pmatrix}
1 & 1 & 0 & 1 \\
1 & 0 & 1 & 1 \\
1 & 0 & 1 & 1 \\
0 & 0 & 1 & 1
\end{pmatrix}
\]

(10)

For the empty value \( R_{3,2} \), we first calculate the total similarity score:

\[
similarity_{total} = \cos (R^*_2 \circ mask_2, R^*_1 \circ mask_1) + \cos (R^*_2 \circ mask_2, R^*_3 \circ mask_3) + \cos (R^*_2 \circ mask_2, R^*_4 \circ mask_4)
\]

\[
= \begin{pmatrix}
2.87 \\
0 \\
0 \\
0
\end{pmatrix}
\begin{pmatrix}
0.98 \\
5.14 \\
3.94 \\
0
\end{pmatrix}
+ \begin{pmatrix}
2.87 \\
0 \\
0 \\
0
\end{pmatrix}
\begin{pmatrix}
4.32 \\
0 \\
1.45 \\
4.33
\end{pmatrix}
+ \begin{pmatrix}
2.87 \\
0 \\
0 \\
0
\end{pmatrix}
\begin{pmatrix}
3.88 \\
0 \\
0 \\
0
\end{pmatrix}
\]

\[
= 2.8086 + 0 + 11.1356
\]

\[= 13.94
\]

(11)

We calculate the predicted value \( \hat{R}_{3,2} \) as following:

\[
\hat{R}_{3,2} = \frac{\cos (R^*_2 \circ mask_2, R^*_1 \circ mask_1)R^*_{3,1} + \cos (R^*_2 \circ mask_2, R^*_3 \circ mask_3)R^*_{3,3} + \cos (R^*_2 \circ mask_2, R^*_4 \circ mask_4)R^*_{3,4}}{similarity_{total}}
\]

\[
= \frac{3.94 \times 2.81 + 0 + 0.76 \times 11.14}{13.94}
\]

\[= 19.5289/13.94
\]

\[= 1.4
\]

(12)

So we gain the predicted value \( \hat{R}_{3,2} \). If we want to use discrete ratings 1, 2, ⋯ , 5 we can take the value 1.4 in round to 1. The rest missing values can be predicted in the same manner.

**2.1.3 Example - Numerical Decomposition of Matrices**

\[
\chi^2 = (1 - l_{11}u_{11} + l_{12}u_{21})^2 + (3 - l_{11}u_{12} + l_{12}u_{22})^2 + ... \]

(13)
2.2 Funk-SVD

Traditional SVD needs large storage for completed matrix and its computation complexity is high. In order to solve those two shortcomings, Simon Funk proposed Funk-SVD\cite{4} in 2006 during the ongoing of Netflix Prize competition. Funk-SVD isn’t based on heuristic similarity but applies explicit learning process. Funk-SVD can also be called by latent factor model (LFM).

As we saw in our SVD example, we want to take advantage of an important fact: that the matrices that result in the decomposition can have significantly lower dimension than the original matrix. Here, $f$ is chosen arbitrarily and is a smaller number compared with $m$ and $n$. However, unlike SVD, we do not make any assumptions about the character of the $P$ and $Q$ matrices in $\hat{R} = P^TQ$. That is, in SVD we knew that $U$ and $V$ were unitary matrices and $S$ was a diagonal matrix. In general, for $A = LU$ decomposition, we know that $L$ and $U$ are upper-triangular and lower-triangular matrices, respectively. In the Funk SVD process, we do not place any expectations on $P$ and $Q$; they are simply two component matrices whose product will approximate the original rating matrix.

Let’s define score matrix $R \in \mathbb{R}^{m \times n}$, where $m$ is the number of users and $n$ is the number of items. Each value in this matrix represents user’s rating on item, e.g $\hat{R}(u, i)$ is user $u$’s rating on item $i$. Because we can tens of thousands (if not more) of different items in a real-world scenario, so the dimension of $\hat{R}$ can have problems in computation efficiency. So based on the idea of SVD, Funk proposed this algorithm to compute users’ preference more efficiently.

As we saw with SVD, we can decompose $R$ into three matrices. In the Funk-SVD process, we can decompose score matrix $R$ into the multiplication of two low-dimensional matrices, $P \in \mathbb{R}^{f \times m}$ and $Q \in \mathbb{R}^{f \times n}$\cite{4}.

As we saw in our SVD example, we want to take advantage of an important fact: that the matrices that result in the decomposition can have significantly lower dimension than the original matrix. Here, $f$ is chosen arbitrarily and is a smaller number compared with $m$ and $n$. However, unlike SVD, we do not make any assumptions about the character of the $P$ and $Q$ matrices in $\hat{R} = P^TQ$. That is, in SVD we knew that $U$ and $V$ were unitary matrices and $S$ was a diagonal matrix. In general, for $A = LU$ decomposition, we know that $L$ and $U$ are upper-triangular and lower-triangular matrices, respectively. In the Funk SVD process, we do not expect the same for $P$ and $Q$.

The mathematics of Funk-SVD can be written as follows:

$$\hat{R} = P^TQ$$
$$\hat{R}(u, i) = \hat{r}_{ui}$$
$$\hat{r}_{ui} = \sum_f p_{uf} q_{if}$$

(14)

where $\hat{r}_{ui}$ is the predicted value which can be calculated by $P$ and $Q$ matrices. So the input rating matrix $\hat{R}$ can have many empty value in it and we use the multiplication of $P$ and $Q$ to represent it. Then we can predict those empty values which are our interested parts though the multiplication result of $P$ and $Q$. By using a simple example as figure\cite{4} we can figure out how this decomposition and prediction process works.
In this example, our input rating matrix $R \in \mathbb{R}^{4 \times 4}$ has four users and four items and we use the multiplication result of user matrix $P^T \in \mathbb{R}^{4 \times 2}$ and item matrix $Q \in \mathbb{R}^{2 \times 4}$ to represent it. For the already existing value $R_{AY} = 2.0$ is nearly represented by $P^T_A \cdot Q_Y = 1.2 \cdot 1.0 + 0.8 \cdot 1.1 = 2.08$. After obtaining those two decomposition matrices $P$, $Q$ we can predict the missing values in the rating matrix from $P$, $Q$’s product. In our example, the missing value $R_{AZ}$ can be gained by $P^T_A \cdot Q_Z = 1.2 \cdot 0.8 + 0.8 \cdot 0.4 = 1.28$.

After mastering the whole process of Funk-SVD, we can focus on how to get $P$ and $Q$. Not like the traditional SVD algorithm, we can apply implicit learning process to get those two matrices, i.e we can set the aim to minimum cost on training data to learn $P$, $Q$ matrices. The loss function $C(p, q)$ contains two parts: squared loss and regularization.

$$C(p, q) = \sum_{(u, i) \in \text{Train}} \left( r_{ui} - \sum_{f=1}^{F} p_{uf} q_{if} \right)^2 + \lambda \left( \|p_u\|^2 + \|q_i\|^2 \right)$$ (15)

Here $\lambda$ is the regularization parameter. Stochastic gradient descent (SGD) and other optimization algorithms can be applied to learn the minimum loss and its corresponding $P$, $Q$. We have two target parameters here $p_{uf}$ and $q_{if}$, so we first calculate $C(p, q)$’s partial derivation over them:

$$\frac{\partial C}{\partial p_{uk}} = \sum_i 2(r_{ui} - \sum_f p_{uf} q_{if})(-q_{ik}) + 2\lambda p_{uk}$$
$$\frac{\partial C}{\partial q_{ik}} = \sum_u 2(r_{ui} - \sum_f p_{uf} q_{if})(-p_{uk}) + 2\lambda q_{ik}$$ (16)

Here we use $k$ to represent the $f$ variable. For each $u$ or $i$, define $err_{ui} = r_{ui} - \sum_f p_{uf} q_{if}$. We will talk about the details of SGD and other optimization functions in section 3.1.3. And according to SGD, we update parameters as equation 17 for each $i$ or $u$ in the loop.

$$p_{uk} = p_{uk} + \alpha (q_{ik} \cdot err_{ui} - \lambda p_{uk})$$
$$q_{ik} = q_{ik} + \alpha (p_{uk} \cdot err_{ui} - \lambda q_{ik})$$ (17)
where \( k \in [0, f] \), \( \alpha \) is the learning rate. The pseudo-code of Funk-SVD is as algorithm 1.

Algorithm 1: Funk-SVD Algorithm

```plaintext
Initialize \( P, Q \):
\[ p_{ik}, q_{i,k} \leftarrow \text{random}() / \sqrt{f} \]
for \( step = 1 : n \) do
    for \( u, i, r_{ui} \) in train.items() do
        err_{ui} \leftarrow r_{ui} - <p[u,: f + 1], q[i,: f + 1]>
        Update \( p, q \):
        for \( k = 1 : f \) do
            \[ p_{uk} += \alpha \cdot (q_{ik} \cdot err_{ui} - \lambda p_{uk}) \]
            \[ q_{ik} += \alpha \cdot (p_{uk} \cdot err_{ui} - \lambda q_{ik}) \]
        end for
    end for
end for
return \( P, Q \)
```

There are many different implements of this algorithm and we follow what Funk did in his original paper. The logic behind why this implement update each \( k \) dimension one by one, quote from Funk:

“Anyway, this will train one feature (aspect), and in particular will find the most prominent feature remaining (the one that will most reduce the error that’s left over after previously trained features have done their best). When it’s as good as it’s going to get, shift it onto the pile of done features, and start a new one. For efficiency’s sake, cache the residuals (all 100 million of them) so when you’re training feature 72 you don’t have to wait for predictRating() to re-compute the contributions of the previous 71 features. You will need 2 Gig of ram, a C compiler, and good programming habits to do this.”

2.3 SVD++

SVD++ [5] is raised by Yehuda Koren etc. in Netflix Prize competition. It includes users’ historical scored items into LFM models which takes neighbour effects into consideration. SVD++, like Funk-SVD, learns neighbour relationship by minimize a global loss.

SVD++ [5] is the explicit learning improvement of Item-to-Item Collaboration Filtering (ItemCF) algorithm [6]. Let’s first review how ItemCF works. ItemCF first calculates the similarity between items which is represented as \( w_{ij} \) and then uses item similarity score and users’ historical behaviour to fill up score matrix. Assume \( p_{uj} \) is user \( u \)’s score on item \( j \). ItemCF calculated \( p_{uj} \) as equation 18.

\[
p_{uj} = \sum_{i \in N(u) \cap S(j,K)} w_{ji} r_{ui} \tag{18}
\]

where \( N(u) \) is user \( u \)’s rated item set, \( S(j, K) \) is set of top-\( K \) most related items to item \( j \), \( r_{ui} \) is user \( u \)’s score on item \( i \). In ItemCF \( w_{i,j} \) is defined by \( w_{ij} = \frac{|N(i) \cap N(j)|}{|N(i)|} \) where \(|N(i)|, |N(j)|\) are the number of users who are interested in item \( i, j \), but in SVD++ \( w_{ij} \) is a learnable parameter. Based on ItemCF’s idea and assume \( w_{ij} \) is a learnable parameter, we can write the SVD++ equation as 19. SVD++ can be seen as an extension of Funk-SVD to implicit rating value.

\[
\hat{r}_{ui} = \frac{1}{\sqrt{|N(u)|}} \sum_{j \in N(u)} w_{ij} \tag{19}
\]

Where \( \hat{r}_{ui} \) is the predicted score of user \( u \) on item \( i \). Like Funk-SVD algorithm, loss object of SVD++ can
be defined as following equation:

\[
C(w) = \sum_{(u,i) \in \text{Train}} \left( r_{ui} - \sum_{j \in N(u)} w_{ij} r_{uj} \right)^2 + \lambda w_{ij}^2
\]  

(20)

where \( \lambda \) is the regularization parameter. Because \( w_{ij} \)'s number of parameters can be very large, so Koren proposed to do dimensionality reduction on \( w \). We can decompose \( w \) into two low-dimension vectors' product. Then equation 19 can be rewritten as following equation 21.

\[
\hat{r}_{ui} = \frac{1}{\sqrt{|N(u)|}} \sum_{j \in N(u)} x_i^T y_j
\]

\[
= \frac{1}{\sqrt{|N(u)|}} x_i^T \sum_{j \in N(u)} y_j
\]

(21)

where \( x_i, y_j \) are two \( F \)-dimension vectors, so \( w \)'s number of parameters is eliminated from \( O(i \cdot j) \) to \( O(2 \cdot n \cdot F) \). Moreover, we can add bias terms into equation 21 and combine it with Funk-SVD.

\[
\hat{r}_{ui} = \mu + b_u + b_i + p_u^T q_i + \frac{1}{\sqrt{|N(u)|}} x_i^T \sum_{j \in N(u)} y_j
\]

(22)

where \( \mu \) is global average score of the rating matrix, \( b_u \) is user bias, \( b_i \) is item bias. Those two bias vectors \( b_u \) and \( b_i \) are learnable parameters. Also, in order to not have too many parameters to cause over-fitting, we can arbitrarily set \( x = q \) to reduce the number of parameters. So the final SVD++ can be written as the following equation 23.

\[
\hat{r}_{ui} = \mu + b_u + b_i + q_i^T \left( p_u + \frac{1}{\sqrt{|N(u)|}} \sum_{j \in N(u)} y_j \right)
\]

(23)

The learning process of parameters \( b_u, b_i, p, q \) is similar to Funk-SVD. We can apply SGD according to their partial derivations.

3 Factorization Machine (FM)

Factorization Machine (FM) is designed to combine Support Vector Machine (SVM) and MF. It can handle large sparse data and subsumes MF and SVD++. FM does not only use first-order feature linear components but also second-order (cross-product) of the features to capture more potential relationship inside the features.

Suppose we have \( x \) as input matrix (rating and other features) and \( \hat{y} \) as output. 2-way FM can be written as equation 24 which only concerns first-order and second-order feature interactions. An example of the input and output data format as figure.
Figure 5: Example of FM’s input and output data format.

\[
\hat{y}(x) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle v_i, v_j \rangle x_i x_j \tag{24}
\]

where we have three parameters: \( w_0 \in \mathbb{R}, w \in \mathbb{R}^n, V \in \mathbb{R}^{n \times k} \) that \( w_0 \) is the global bias, \( w_i \) models the strength of the \( i \)-th variable, \( \hat{w}_{i,j} := \langle v_i, v_j \rangle \) models the interaction between the \( i \)-th and \( j \)-th variable which idea is similar with FM that can do dimensionality reduction on \( w \). We can prove the computation complexity of equation (24) is linear.

\[
\sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle v_i, v_j \rangle x_i x_j \\
= \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \langle v_i, v_j \rangle x_i x_j - \frac{1}{2} \sum_{i=1}^{n} \langle v_i, v_i \rangle x_i x_i \\
= \frac{1}{2} \left( \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{f=1}^{k} v_{i,f} v_{j,f} x_i x_j - \sum_{i=1}^{n} \sum_{f=1}^{k} v_{i,f} v_{i,f} x_i x_i \right) \\
= \frac{1}{2} \left( \sum_{f=1}^{k} \left( \sum_{i=1}^{n} v_{i,f} x_i \right)^2 - \sum_{i=1}^{n} v_{i,f}^2 x_i^2 \right) \\
= \frac{1}{2} \sum_{f=1}^{k} \left( \sum_{i=1}^{n} v_{i,f} x_i \right)^2 \tag{25}
\]

So equation (24)’s computation complexity is \( O(k \cdot n) \) where \( k \) is latent factor’s dimension and \( n \) is input \( x \)’s dimension. We can compute those model parameters with gradient decent methods for a variety of losses. The partial gradient of parameters in equation (24) can be computed as following:

\[
\frac{\partial}{\partial \theta} \hat{y}(x) = \begin{cases} 
1, & \text{if } \theta = w_0 \\
x_i, & \text{if } \theta = w_i \\
x_i \sum_{j=1}^{n} v_{j,f} x_j - v_{i,f} x_i^2, & \text{if } \theta = v_{i,f} 
\end{cases} \tag{26}
\]

“In practice, usually a two-way FM model is used, i.e., only the second-order feature interactions are considered to favor computational efficiency.” [8] The equation for the multi-way feature interactions can be found in the original paper [7].
3.0.1 Field-Aware Factorization Machine (FFM)

Field-aware factorization machine (FFM) \cite{10} is an extension to FM. For traditional recommendation problems, e.g. movie recommendation, there are only three available fields User, Item, and Tag. But when it comes to click-through-rate prediction problem, we can have multiple fields, e.g. Publisher (P), Advertiser (A), Gender (G), Click Result. We recognize those feature categories as fields. FFM is the upgrade of factorization machine (FM) by adding the field concept: it considers similar features as a same field. The advantage of FFM over FM is that, it uses different factorized latent factors for different fields of features. “Putting features into fields resolves the issue that the latent factors shared by features that intuitively represent different categories of information may not well generalize the correlation.”\cite{11}

Suppose $X \in \mathbb{R}^{n \times k}$, $k$ is latent factor’s dimension and $f$ is the number of fields. FFM equation based on 2-way FM equation can be written as following:

$$\theta_{\text{FFM}}(wx) = \sum_{j_1=1}^{n} \sum_{j_2=j_1+1}^{n} <\mathbf{v}_{j_1,f_1}, \mathbf{v}_{j_2,f_2}> x_{j_1} x_{j_2}$$  \hspace{1cm} (27)

where $f_1$ and $f_2$ are respectively the fields of $j_1$ and $j_2$. FFM’s number of variables is $O(f \cdot n \cdot k)$, equation 27’s computation complexity is $O(n^2 k)$.

3.1 With Deep Learning

3.1.1 DeepFM

Factorization-Machine based neural network(DeepFM) is a evolutional model of Wide & Deep model from Google. DeepFM has a share input of its ‘wide’ and ‘deep’ part which behind idea is quit similar with the traditional CTR model, Factorization Machine(FM). The model aims at predicting CTR by exphasizing both low- and high- order feature interactions.

DeepFM has two components: FM component and deep component. The input of these two components is the same embedding of input features. The output of the model is the sigmoid result of the combination of this two components.

The architecture of DeepFM model is as figure 6.

Figure 6: Architecture of DeepFM model.

FM part of DeepFM don’t need the feature engineering for inputs but it only realizes first-order and second-order feature interactions because of the parameter number. Suppose we have $m$ fields and embedding
dimension is \( k \), second-order feature interaction needs \( O(k \times m^2) \) parameters while third-order needs \( O(k \times m^3) \). Deep & Cross Net [10] is proposed to solve the parameters’ rapid growth problem.

### 3.1.2 Model Ensemble

The concept of model ensemble is like the concept of wisdom of the crowd. We can take the advantage of a bunch independent models. To be more clear, adopt each one’s good points and avoid its shortcomings. A list of common ensemble methods [11]:

1. Vote
2. Linear blending / weighted blending
3. Bootstrap aggregation (Bagging)
4. Stacking

### 3.1.3 Optimization Algorithms

We can use one frame to explain all optimization algorithms since all those algorithms can be simplified into those four steps [12].

**Algorithm 2:** Outline of general optimization algorithms

| Model parameters: \( \theta \in \mathbb{R}^d \)  |
|-------------------------------|----------------|
| Target function: \( J(\theta) \)  |
| Learning rate: \( \alpha \)  |

for Each epoch \( t \) do

1. Calculate target function’s gradients over model current parameters:
   \[ g_t = \nabla J(\theta_t) \]
2. Calculate 1st and 2nd order momentum based on past gradients:
   \[ m_t = \phi(g_1,g_2,\cdots,g_t); V_t = \psi(g_1,g_2,\cdots,g_t) \]
3. Calculate the gradient should be eliminated:
   \[ \eta_t = \frac{\alpha \cdot m_t}{\sqrt{V_t + \epsilon}} \]
   \( \epsilon \) is a minimal value to protect denominator from zero, usually set \( \epsilon = 1e^{-8} \)
4. Update parameters:
   \[ \theta_{t+1} = \theta_t - \eta_t \]

end for
References

[1] D. Billsus, M. J. Pazzani et al., “Learning collaborative information filters.” in ICML, vol. 98, 1998, pp. 46–54.

[2] [Online]. Available: https://www.cnblogs.com/pinard/p/6251584.html

[3] [Online]. Available: https://web.stanford.edu/class/cs168/l/l9.pdf

[4] S. Funk, 2006. [Online]. Available: http://sifter.org/~simon/journal/20061211.html

[5] Y. Koren, “Factor in the neighbors: Scalable and accurate collaborative filtering,” ACM Transactions on Knowledge Discovery from Data (TKDD), vol. 4, no. 1, pp. 1–24, 2010.

[6] G. Linden, B. Smith, and J. York, “Amazon.com recommendations: Item-to-item collaborative filtering,” IEEE Internet computing, vol. 7, no. 1, pp. 76–80, 2003.

[7] S. Rendle, “Factorization machines,” in 2010 IEEE International Conference on Data Mining, 2010, pp. 995–1000.

[8] [Online]. Available: https://github.com/microsoft/recommenders/blob/master/notebooks/02_model/fm_deep_dive.ipynb

[9] Y. Juan, D. Lefortier, and O. Chapelle, “Field-aware factorization machines in a real-world online advertising system,” in Proceedings of the 26th International Conference on World Wide Web Companion, 2017, pp. 680–688.

[10] R. Wang, B. Fu, G. Fu, and M. Wang, “Deep & cross network for ad click predictions,” in Proceedings of the ADKDD’17, 2017, pp. 1–7.

[11] [Online]. Available: https://zhuanlan.zhihu.com/p/25836678

[12] [Online]. Available: https://zhuanlan.zhihu.com/p/32230623