Unsupervised Learning - Matrix Completion

MInDS @ Mines

In this lecture we will introduce matrix completion and its applicability to recommendation systems. Recommendations systems are used to populate the "suggested friends" features on social media networks, suggest movies to watch on Netflix. In this handout we summarize two widely used techniques in matrix completion with recommendation systems as the application. We first discuss a nearest neighbors approach followed by a rank minimization technique.

Recommendation Systems

Recommendation systems are systems with the goal of making recommendations based on some underlying data. There are two main types of recommendation systems, content-based filtering and collaborative filtering[1].

With content-based filtering, the model recommends content based on the relationship with the currently viewed item. Content-based filtering focuses on the meaning of the content you're currently viewing and its content-only relationship to other available content. Content-based filtering therefore requires an understanding of the data itself[2] and extracting representative features that can be used to effectively measure content similarities. We will briefly cover how we can apply this when we discuss feature learning and application specific feature extraction. For the rest of this lecture however, we will focus more heavily on collaborative filtering.

Collaborative filtering focuses on shared behavior rather than content meaning. It looks at what others have liked and will recommend things based on that if you have also liked things they have liked. A good example of this is in social networks: you are likely friends with people who are common friends with your friends. The more mutual friends you share, the more likely you are to actually be friends with them.

[1] There also exist hybrid systems that combine these two approaches

[2] You can think of Spotify's song recommendations as content-based filtering since they recommend songs based on what you're listening to.

Overview of Matrix Completion

Matrix completion is a task designed to **fill in missing values** into a partially observed matrix **X**. An intuitive example of matrix completion can be explained by a movie-ratings matrix[3]. The problem is defined as: Given a ratings matrix **X** in which each entry x_{ij} represents the rating of movie j by customer i if customer i has watched movie j and is otherwise missing, we would like to predict the remaining entries in order to make good recommendations to customers on what to watch next. In this problem we start with a sparsely populated matrix **X**, where each row represents the collection of ratings for a specific customer, and try to learn another more dense matrix **M** that should contain which movies customer's in **X** would be interested in watching but have not provided a rating.

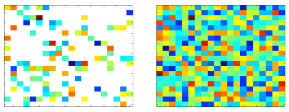

Figure 1: Visualizaiton of matrix completion **Left:** incomplete matrix (X). **Right**: matrix completion result (M) (after the application of some algorithm).

[3] The most famous movie-ratings matrix problem was called the *Netflix Prize*. The Netflix Prize was an open competition designed to automatically predict user ratings for films, based on previous ratings without any other information about the users or films. The winners of this competition in 2009 received a one-million dollar prize.

k-Nearest Neighbors for Matrix Completion

k-Nearest Neighbors (kNN) is a useful algorithm that matches points with their k closest neighbors. This algorithm is particularly useful since it can be effective with an arbitrary number of variables and is trivial to implement.

When trying to determine the missing values in $\mathbf{X}$ the k nearest neighbors algorithm finds the k closest neighbors to a particular observation (row in $\mathbf{X}$) with missing data and then assigns the missing values based on the non-missing values in the neighbors. The *key assumption* behind using kNN to fill in missing values is that a missing value can be approximated by values contained within records that are closest to it.

Two considerations must be taken into account when using the kNN approach. First, we determine the value for k, how many neighbors are used to predict a given missing value. Second, we determine the distance metric that will be used to quantify how close two records are in $\mathbf{X}$; for example, we could use either the Euclidean[4] or the Manhattan[5] distance to determine how closely related two records are to each other.

[4] $D_{Euclidean}(\mathbf{x}_i, \mathbf{y}_i) = ||\mathbf{x}_i - \mathbf{y}_i||_2$
[5] $D_{Manhattan}(\mathbf{x}_i, \mathbf{y}_i) = ||\mathbf{x}_i - \mathbf{y}_i||_1$

Low Rank Matrix Completion

In many real world applications data can be grouped based on a small number of characteristics. For example, let's suppose we can group movies into a small number of genres (e.g. comedies, thrillers, documentaries, etc.), then we can use an individual's preference for a specific genre to fill in any missing values in a given matrix $\mathbf{X}$. Our *key assumption*, when applying an algorithm that incorporates this low rank, is that a user that has highly rated a comedy will also like other movies that are within the same genre.

From the mathematical perspective we model this genre discovery problem as a constrained rank minimization problem. The *rank* of a matrix is defined as the number of linearly independent[6] column vectors in a matrix. Here we provide the general objective to minimize the rank of a matrix $\mathbf{M}$ with regards to the matrix completion problem:

$$\begin{aligned} \min_{\mathbf{M}} \quad & \text{rank}(\mathbf{M}) \\ \text{s.t.} \quad & M_{ij} = X_{ij} \end{aligned} \tag{1}$$

[6] A set of vectors is said to be linearly dependent if one of the vectors in the set can be defined as a linear combination of the others; if no vector in the set can be written in this way, then the vectors are said to be *linearly independent*.

Note that the objective described in Eq. 1 is constrained by $M_{ij} = X_{ij}$. This ensures that our learned low-rank matrix $\mathbf{M}$ is consistent with the partially observed matrix $\mathbf{X}$. Once the matrix $\mathbf{M}$ has been learned, it is possible to inspect its contents of $\mathbf{M}$ to reveal the missing entries in $\mathbf{X}$. We end our introduction of low-rank matrix completion with a simple example:

In Fig. 2 we can see that the data in $\mathbf{X}$ can be reconstructed by a rank-1 matrix $\mathbf{M}$. Applying this assumption to $\mathbf{X}$ and it is easy to see that $\mathbf{M}$ is readily derived. Although we provide a simple example in Fig. 2, it is unclear how we can design an algorithm for a more complicated $\mathbf{X}$. To address this ambiguity

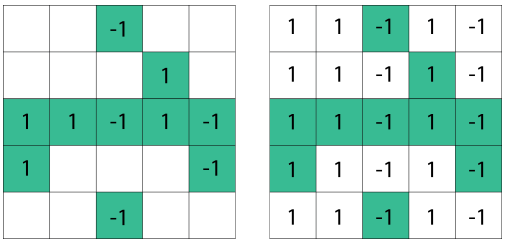

Figure 2: Matrix completion of a partially revealed 5 by 5 matrix with rank-1. **Left**: observed incomplete matrix (X). **Right**: matrix completion result (M).

we take Eq. 1 and reformulate it as a *matrix factorization* problem.

Matrix Factorization Approach

Instead of tackling the problem defined in Eq. 1 we model the matrix **X** as the multiplication of two smaller matrices **P** and **Q** such that **X** $\approx$ **PQ**. We can write this matrix factorization as,

$$\begin{aligned} \min_{\mathbf{P},\mathbf{Q}} \quad & ||\mathbf{X} - \mathbf{PQ}||_F^2 \\ \text{where} \quad & \mathbf{X} \in \mathbb{R}^{n \times d},\ \mathbf{P} \in \mathbb{R}^{n \times r},\ \mathbf{Q} \in \mathbb{R}^{r \times d}, \\ & r << d \end{aligned} \tag{2}$$

Note that this model introduces a new quantity r that serves as a dimension in both **P** and **Q**. This r defines the *rank* of the factorization. Intuitively, this model assumes that every entry in **X** can be represented by r different effects; this is analogous to fitting a user's movie into a particular genre (or collection of genres). Once this optimization problem has been solved, the missing values of **X** can be be determined from the **PQ** factorization. Now we briefly discuss two optimization algorithms that can be used to solve the problem in Eq. 2

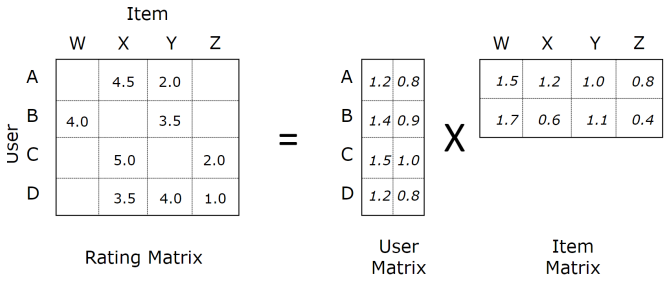

Figure 3: Visualization of the matrix factorization approach for matrix completion. In this example $X \in \mathbb{R}^{4\times4}$, $P \in \mathbb{R}^{4\times2}$, $Q \in \mathbb{R}^{2\times4}$. Here, we represent the original *four* items into *two* separate groupings.

Gradient descent algorithm:

Gradient descent is a first-order optimization algorithm that is widely used in the field of machine learning. Conceptually, it is a simple iterative optimization process that assumes the existence of a cost function and arbitrary

initial values for the optimization variables. In every iteration it re-computes the gradient of the cost function with respect to the optimization variables, and updates them in a step that is proportional to the negative of the gradient of the cost function, targeting at minimizing the cost function, until it converges to a minimum point. However, optimizing a cost function with gradient descent only guarantees convergence to a local minimum.

Gradient descent has been shown to work well optimizing matrix factorization models. However, it is not a popular choice for an optimizer for matrix factorization if the dimensionality of the original rating matrix is high, as there are effectively $(n \times r + r \times d)$ parameters to optimize.

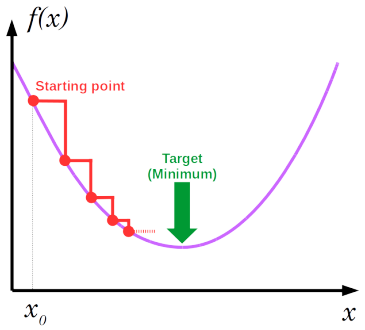

Figure 4: An illustration of gradient descent reaching the optimal value.

Alternating least squares algorithm:

Alternating least squares is a two-step iterative optimization process. In every iteration it first fixes P and solves for Q, and following that it fixes Q and solves for P. The full algorithm for this approach is shown in Algorithm 1.

The alternating least squares approach was one of the main algorithms incorporated in the winning entry of the Netflix Prize.

Algorithm 1: Alternating Minimization for Matrix Completion

Input: Partially observed matrix X

Output: Fully observed prediction PQ

1 Initialize P, Q
2 **while** *not converge* **do**
3 $\quad \mathsf{P}^{t+1} = \mathsf{X}\mathsf{Q}^T(\mathsf{Q}\mathsf{Q}^T)^{-1}$
4 $\quad \mathsf{Q}^{t+1} = (\mathsf{P}^T\mathsf{P})^{-1}\mathsf{P}^T\mathsf{X}$
5 **end**

In update steps for P and Q, the cost function in Eq. 2 either decreases or stays unchanged. Alternating between the two steps guarantees reduction of the cost function, until convergence. Similar to gradient descent optimization, it is guaranteed to converge only to a local minimum, and ultimately depends on the initial values for P or Q.

Here, we show how we found these values. We derive the objective function with respect to P and Q separately and set them equal to 0.

$$\frac{\partial||\mathsf{X}-\mathsf{P}\mathsf{Q}||_F^2}{\partial \mathsf{P}} = -2(\mathsf{X}-\mathsf{P}\mathsf{Q})\mathsf{Q}^T, \qquad \frac{\partial||\mathsf{X}-\mathsf{P}\mathsf{Q}||_F^2}{\partial \mathsf{Q}} = -2\mathsf{P}^T(\mathsf{X}-\mathsf{P}\mathsf{Q}). \tag{3}$$

We set the derivatives to 0 and get, (4)

$$(\mathsf{X}-\mathsf{P}\mathsf{Q})\mathsf{Q}^T = 0, \qquad \mathsf{P}^T(\mathsf{X}-\mathsf{P}\mathsf{Q}) = 0, \tag{5}$$

$$\mathsf{X}\mathsf{Q}^T - \mathsf{P}\mathsf{Q}\mathsf{Q}^T = 0, \qquad \mathsf{P}^T\mathsf{X} - \mathsf{P}^T\mathsf{P}\mathsf{Q} = 0, \tag{6}$$

$$\mathsf{P}\mathsf{Q}\mathsf{Q}^T = \mathsf{X}\mathsf{Q}^T, \qquad \mathsf{P}^T\mathsf{P}\mathsf{Q} = \mathsf{P}^T\mathsf{X}, \tag{7}$$

$$\mathsf{P} = \mathsf{X}\mathsf{Q}^T(\mathsf{Q}\mathsf{Q}^T)^{-1}, \qquad \mathsf{Q} = (\mathsf{P}^T\mathsf{P})^{-1}\mathsf{P}^T\mathsf{X}. \tag{8}$$

Other important methods (that are outside the scope of our class) for matrix completion can be found at `https://statweb.stanford.edu/~candes/papers/SVT.pdf`