Unsupervised Learning - Matrix Completion

MInDS @ Mines

In this lecture we will introduce matrix completion. This technique is frequently used in industry to make recommendation systems. For example, matrix completion algorithms are used to populate the "suggested friends" features on social media networks, suggest movies to watch on Netflix, and many other real-world applications. In this handout we summarize two widely used techniques in matrix completion. We first discuss a nearest neighbors approach followed by a rank minimization technique. Both of these methods come with their own set of assumptions and provide corresponding solutions to the general matrix completion problem.

A High Level Overview of Matrix Completion

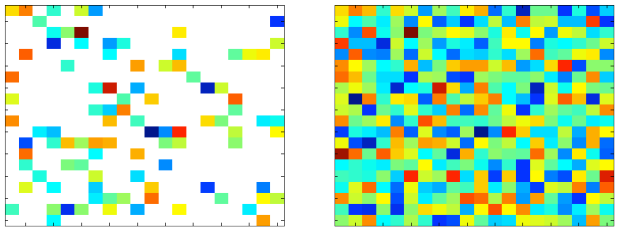


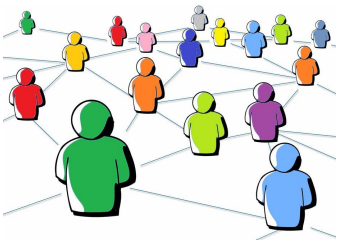
Figure 1: Visualizaiton of matrix completion **Left:** incomplete matrix (X). **Right:** matrix completion result (M) (after the application of some algorithm).

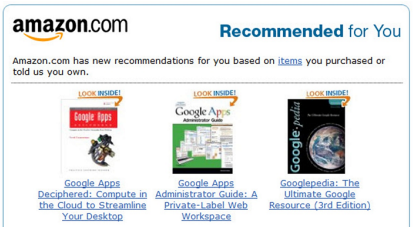
Matrix completion is a task designed to **fill in missing values** into a partially observed matrix X. An intuitive example of matrix completion can be explained by a movie-ratings matrix[1]. The problem is defined as such: Given a ratings matrix X in which each entry x_{ij} represents the rating of 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.

[1] 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 (2009) received a one-million dollar prize.

Why Do We Care?

Matrix completion can be applied across many different problem areas. Whether it's suggesting possible friends on social media websites, what movie to watch, determining what a customer might like to buy, matrix completion is used in a variety of different industries:





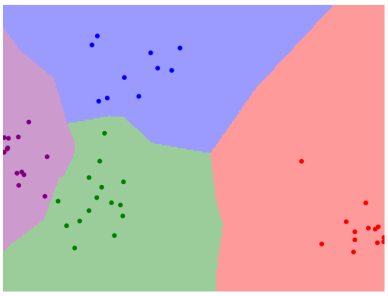
The wide variety of problem that matrix completion algorithms can be used to solve is the main motivating factor for why we are interested in studying this problem. Many algorithms have been developed for matrix completion due to the large number of real-world problems that matrix completion can be applied to solve.

A Simple Approach: k-Nearest Neighbors

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

The Algorithm: When trying to determine the missing values in X the k nearest neighbors algorithm finds the k closest neighbors to a particular observation (row in 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.

Parameter Selection: Two considerations must be taken into account when using the kNN approach. First, the programmer first has to determine how many neighbors are used to predict a given missing value. Second, the programmer also has to determine the distance metric that will be used to quantify how "close" two records are in X; for example, we could use either the Euclidean[2] or the Manhattan[3] distance to determine how closely two records are related.



An image from the Stanford Vision Lab demo illustrating the output of a kNN used for clustering data. In the matrix completion problem a similar clustering activity is performed where unknown features are "filled-in" based on entries that are similar to one another.

[2] $D_{euclidian} = \left(\sum_{i=i}^{n} |x_i - y_i|^2\right)^{1/2}$
[3] $D_{manhattan} = \sum_{i=i}^{n} |x_i - y_i|$

Low Rank Matrix Completion

In many real world applications it can often make sense to assume that data is grouped based on 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 individuals preference for a specific genre to fill in any missing values in a given matrix 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[4] column vectors in a matrix. Here we provide the general objective to minimize the rank of a matrix M with regards to the matrix completion problem:

$$\begin{aligned} \min_{M} \quad & \text{rank}(M) \\ \text{subject to} \quad & M_{ij} = X_{ij} \end{aligned} \tag{1}$$

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

		-1		
			1	
1	1	-1	1	-1
1				-1
		-1		

1	1	-1	1	-1
1	1	-1	1	-1
1	1	-1	1	-1
1	1	-1	1	-1
1	1	-1	1	-1

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).

In Fig. 2 we assume that the data in X can be reconstructed by a rank-1 matrix M. Applying this assumption to X and it is easy to see that 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 X. To address this ambiguity we take Eq. 1 and reformulate it as a *matrix factorization* problem.

The Matrix Factorization Approach

Instead of tackling the problem defined in Eq. 1 we try to 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 follows:

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

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 a fitting a user's movie into a particular genre (or collection of genres). Once this optimization problem has been solved the

[4] 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*.

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

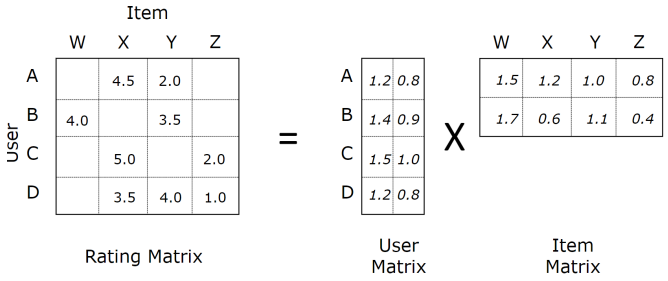
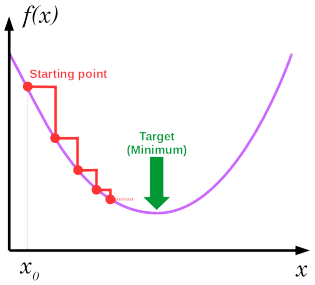


Figure 3: Visualization of the matrix factorization approach for matrix completion. In this example $X \in \mathbb{R}^{4\times4}$, $P \in \mathbb{R}^{4\times2}$, $Q \in \mathbb{R}^{2\times4}$. Intuitively, we are trying to represent the original *four* items into *two* separate groupings.

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 the latter converges to a minimum point. However, optimizing a cost function with gradient descent only guarantees convergence to a local minima.

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.



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

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:

$$\begin{aligned} P^{t+1} &= XQ^T(QQ^T)^{-1} \\ Q^{t+1} &= (P^TP)^{-1}P^TX \end{aligned} \tag{3}$$

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