Project - High Dimensional Data Analysis
-
Recommendation systems via approximate matrix factorization
-

Valentin Leplat and Nicolas Gillis

1 Introduction

The aim of a recommendation system is to predict users' preferences for certain products, using their preferences and those of other users (this is called collaborative filtering, see for example `https://en.wikipedia.org/wiki/Collaborative_filtering`). This problem has enormous economic potential, especially for online retailers and platforms such as Netflix. Indeed, good recommendations will increase the likelihood of a purchase. For example, in 2007, Netflix launched a competition to improve its predictions: the team who provided an algorithm improving Netflix's predictions by 10% won 1\$ million; see `https://en.wikipedia.org/wiki/Netflix_Prize`.

Imagine, for example, that a number of people have been asked to rate a number of films on a scale of 1 to 5, and you get the following preference matrix:

	The Matrix	Once	Spiderman	Peter Pan	X-Men
Nathalie	2	3	2	?	?
John	?	2	?	4	3
Marc	3	?	3	?	4
Oliver	?	3	?	4	3
Matthew	?	?	?	?	2
Bill	1	4	3	4	?

Question marks mean that the film has not been viewed (or rated). You'd like to predict whether or not Nathalie will like the film Peter Pan.

2 Simple mathematical model: matrix factorization

A very effective model often used in practice uses matrix factorization. In particular, the winners of the Netflix \$1 million competition used this technique.; see the paper by Koren, Bell and Volinsky, *Matrix Factorization Techniques for Recommender Systems*[1], IEEE Computer, 2009. This model is based on the following assumption: user behavior can be modeled via a linear combination of a smaller number of users (called 'type' or 'feature' users, linked to gender, age, culture, etc.). This is equivalent to assuming that preferences for a film can be modeled by a linear combination of feature films (related to the different types of film – children's, crime, horror, actors, director, etc.).

[1] `https://www.inf.unibz.it/~ricci/ISR/papers/ieeecomputer.pdf`

Let's denote X the m-by-n dimensional preference matrix (m users, n films) and r the number of 'feature' users. So we'd like to model user behavior from linear combinations of 'feature' user behavior, i.e. we'd like to approximate each line of X with a linear combination of r vectors to be determined. Denoting $V(k,:)$ with $1 \leq k \leq r$ the 'feature' user preference vectors, we then have

$$\underbrace{X(i,:)}_{i^{\text{th}}\text{ user}} \approx \sum_{k=1}^{r} \underbrace{U(i,k)}_{k^{\text{th}}\text{ weight for } i^{\text{th}}\text{ user}} \underbrace{V(k,:)}_{k^{\text{th}}\text{ 'feature' user}}, \quad \forall i.$$

In other words, we'd like to find two matrices U (m rows and r columns) and V (r rows and n columns) such that $X \approx UV$.

These matrices, U and V, are the parameters of the model and must be estimated.

Here's an example for the matrix in the table above:

$$\begin{aligned} X &= \begin{pmatrix} 2 & 3 & 2 & ? & ? \\ ? & 2 & ? & 4 & 3 \\ 3 & ? & 3 & ? & 4 \\ ? & 3 & ? & 4 & 3 \\ ? & ? & ? & ? & 2 \\ 1 & 4 & 3 & 4 & ? \end{pmatrix} \\ &\approx \begin{pmatrix} 1 & 0 & 0 \\ 0.5 & 0.5 & 0 \\ 0 & 0.5 & 0.5 \\ 0.4 & 0.2 & 0.4 \\ 0.5 & 0 & 0.5 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 3 & 2 & 5 & 1 \\ 5 & 1 & 3 & 3 & 5 \\ 1 & 4 & 3 & 4 & 3 \end{pmatrix} = UV \\ &= \begin{pmatrix} 2 & 3 & 2 & 5 & 1 \\ 3.5 & 2 & 2.5 & 4 & 3 \\ 3 & 2.5 & 3 & 3.5 & 4 \\ 2.2 & 3 & 2.6 & 4.2 & 2.6 \\ 1.5 & 3.5 & 2.5 & 4.5 & 2 \\ 1 & 4 & 3 & 4 & 3 \end{pmatrix}. \end{aligned}$$

Based on this decomposition, it looks like Nathalie will enjoy Peter Pan, but not X-Men. The matrices U and V are what we call an approximate factorization of the matrix X.

To quantify the quality of an approximation, we'll use the following objective function:

$$\sum_{(i,j)\in\Omega} (X - UV)_{ij}^2,$$

where Ω denotes the set of known entries (some X entries are unknown). By defining the W matrix as follows

$$W_{ij} = \begin{cases} 1 & \text{if } (i,j) \in \Omega \\ 0 & \text{otherwise} \end{cases},$$

the corresponding optimization problem can be written as follows:

$$\min_{U\in\mathbb{R}^{m\times r}, V\in\mathbb{R}^{r\times n}} \quad \sum_{i=1}^{m}\sum_{j=1}^{n} W_{ij}\,(X-UV)_{ij}^2\,, \tag{1}$$

and the unknown inputs of X can be given any value (e.g. 0).

The aim of this project is at first to develop and analyze one or more methods for solving the problem. We will then try to improve the model to obtain better predictions. All the information can be found on `https://www.kaggle.com/t/b46ea03ede224b9db93b17886565ee41`.

3 Project organization and report content

To complete the project, I suggest you follow the steps below:

1. Describe and implement a method for solving the problem (1). Use this implementation to make some initial predictions: first for the simple matrix above, and then for the data available on Kaggle.

2. Develop an improved model of (1), and implement a method for estimating the parameters of this new model, i.e. solving the corresponding new optimization problem. Use this implementation to make initial predictions: first on the simple matrix above, and then on Kaggle. Improve your results.

3. Try to refine your model based on the numerical results obtained on Kaggle. Try to understand the influence of different parameters.

4. *To go even further* (Bonus): Explore more advanced models from the recent literature: choose one of them, motivate your choice, briefly describe the model, implement it (do not use the implementation of others or external libraries) and benchmark with previous points.

It's a good idea to write your report as you go along. The report will contain 4 main parts:

1. Introduction,

2. description of the models used,

3. description of the algorithm(s) (method for finding the parameters of the models used), and

4. presentation, interpretation and discussion of results (influence of model on prediction quality, influence of choice of algorithm and initial solution, etc.).

A Block coordinate descent method

A simple and effective method for solving optimization problems of type (1) is the block coordinate descent method. Let the problem be

$$\min_{U,V} f(U,V),$$

where there are two blocks of variables: U and V (unconstrained), and f is the objective function.

The block coordinate descent method will optimize the variables U and V alternately. We choose an initial solution $\left(U^{(0)}, V^{(0)}\right)$, and then iterate: for $k = 0, 1, 2, \ldots,$

1. Fix $U^{(k)}$ and update
$$V^{(k+1)} \leftarrow \min_V f\left(U^{(k)}, V\right),$$
from $V^{(k)}$..
2. Set $V^{(k+1)}$ and update
$$U^{(k+1)} \leftarrow \min_U f\left(U, V^{(k+1)}\right),$$
from $U^{(k)}$.

B Update: gradient descent

To update U and V, we can use the gradient descent method. Given the problem

$$\min_U h(U),$$

where U contains the (unconstrained) variables, and h is the objective, the gradient method chooses an initial solution $U^{(0)}$, and then iterates: for $k = 0, 1, 2, \ldots,$

1. Gradient calculation:
$$g\left(U^{(k)}\right) = \nabla f\left(U^{(k)}\right).$$
The gradient is the direction of greatest slope: it is a vector/matrix of the same dimension as the U variable, and contains the partial derivative with respect to each entry of U.

   Example. *The function $f(u) = u_1^2 + u_2^3 + u_1 u_2 + 3$ where $u = (u_1, u_2)$ has gradient*
$$g(u) = \nabla f(u) = \left(2u_1 + u_2, 3u_2^2 + u_1\right).$$
2. Descent step:
$$U^{(k+1)} \leftarrow U^{(k)} - \alpha^{(k)} g\left(U^{(k)}\right),$$
where $\alpha^{(k)}$ is the step size.

To choose $\alpha^{(k)}$, we can use backtracking, for example:

- we initialize
$$\alpha^{(k)} = \beta \alpha^{(k-1)},$$
for $\beta > 1$,

- while $h\left(U^{(k+1)}\right) > h\left(U^{(k)}\right)$, we decrease the step size

  $$\alpha^{(k)} \leftarrow \gamma \alpha^{(k-1)},$$

  for $\gamma < 1$.

  More advanced methods may be considered

  1. for the gradient descent step: the *Accelerated Gradient Descent* proposed by Yurii Nesterov,
  2. for the step size computation: the use of *Armijo rule* (eq. 4.5 here), or *constant strategy* if the Lipschitz constant $L > 0$ for the gradient exists and can be computed.
  3. Problem (1) can be solved using the *Alternating Direction Method of Multipliers* (ADMM), see the lecture here for instance. Note that Higher-order optimization methods may also be considered if the computational complexity of each step remains tractable.

C Example: rank 1 factorization

Consider the problem (1) for $r = 1$, i.e.

$$\min_{u \in \mathbb{R}^m, v \in \mathbb{R}^n} \sum_{i,j} W_{ij}(X_{ij} - u_i v_j)^2.$$

Since the problem is perfectly symmetrical in u and v, as $X = uv^\top \iff X^\top = vu^\top$, the update of v will be the same as that of u, apart from one transposition. We'll concentrate on updating u using the gradient method. By defining

$$f(u) = \sum_{i,j} W_{ij}(X_{ij} - u_i v_j)^2,$$

the gradient is given by

$$[g(u)]_k = [\nabla f(u)]_k = \frac{\partial f(u)}{\partial u_k} = \frac{\partial}{\partial u_k} \sum_{i,j} W_{ij}(X_{ij} - u_i v_j)^2 = -2 \sum_j W_{kj}(X_{kj} - u_k v_j) v_j.$$

In vector form, this gives

$$g(u) = \nabla f(u) = \left[W \circ (X - uv^\top)\right] v,$$

where $\circ$ is the element-by-element product between two matrices (in Matlab: `a.*b`, in python: `numpy.multiply(a,b)`). We can now easily implement the gradient method to minimize $f(u)$; see `function updateWLRAgrad` in colab file.