Latant factor model

The latant factor model comes from matrix singular decomposition:

, in which A represents the user-movie data matrix, represents the hidden user-concepts relationship and represents the hidden concepts-movie relationship.

SVD algorithm helps us find the hidden common elements and concepts, which can be used to predict missing values in matrix A. The dimensionality of denotes the number of hidden concepts and is predefined by human.

When predicting missing values, we do the following:

1. Use SVD to decompose user-movie matrix A to
2. We don’t need the exact value of , so rewrite this equation as , in which is true for non-zero valus in A and .
3. Predict

But the problem is matrix A itself has a lot of missing values remain to be predicted. If we fill these missing values as 0, then SVD will predict all these missing values as 0, which is not intended and unreasonable.

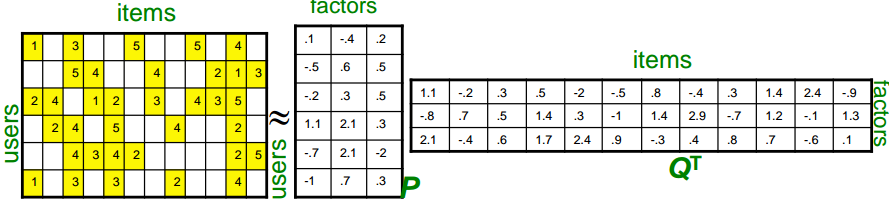
So we can’t adopt the traditional SVD approach. And the evolved algorithm is Latant Factor Model. The key point of this model is:

1. Goal: find martix P and Q such that:

, is satisfied **except for the missing values.**

1. Approach: Stochastic Gradient Decent on P and Q.

That is to find, by SDG, P and Q such that:



In order to avoid overfitting, we add some regularization term to the loss function:

Notice: means **all missing values are ignored when computing Loss.**

Solve this using Gradient descent :

1. Initialize P and Q (randomly with normal distribution)
2. Compute Loss
3. Calculate gradient for every elemtent in P and Q: and

, in which

1. Do gradient descent:

, in which is the predefined learning rate.

1. Until a fixed number of loops

About our code:

The python code for our implementation of Latant Factor Model is in **LFM.ipynb**

Here are some details:

1. Use the same train data and validation data as used in Collaborative Filterring.
2. We chose 100 hidden concepts, which is resulted from repeated experiments.
3. Learning rate: 0.005
4. Regularization parameter: 0.001

Experiments tell us:

1. Training error continiuosly decreases
2. Test error decreases at first and then increases

So we choose the number of iterations to be 24, giving us the minimum test error.

The final average error of this model (only LFM, no CF) when predicting user rating on test data is:

0.6654601311607836, when the full rating scale is 0 to 5.

Combining LFM with CF, Content-based should give us even better result.