Title of my document

John Doe

2013-09-01

1 Motivation

User modeling, adaptation, and personalization techniques have hit the mainstream. The explosion of social network websites, on-line user-generated content platforms, and the tremendous growth in computational power of mobile devices are generating incredibly large amounts of user data, and an increasing desire of users to "personalize" (their desktop, e-mail, news site, phone).

The potential value of personalization has become clear both as a commodity for the benefit or enjoyment of end-users, and as an enabler of new or better services a strategic opportunity to enhance and expand businesses.

An exciting characteristic of recommender systems is that they draw the interest of industry and businesses while posing very interesting research and scientific challenges.

In spite of significant progress in the research community, and industry efforts to bring the benefits of new techniques to end-users, there are still important gaps that make personalization and adaptation difficult for users. Research activities still often focus on narrow problems, such as incremental accuracy improvements of current techniques, sometimes with ideal hypotheses, or tend to overspecialize on a few applicative problems (typically TV or movie recommenders sometimes simply because of the availability of data). This restrains de facto the range of other applications where personalization technologies might be useful as well.

Thus, we may have reached a good point to take a step back to seek perspective in the research done in recommender systems. This workshop contrives for a new uptake on past experiences and lessons learned. We propose an analytic outlook on new research directions, or ones that still require substantial research, with a special focus on their practical adoption in working applications, and the barriers to be met in this path.

2 Latent Matrix Factorization

Recommendations can be generated by a wide range of algorithms. While user-based or item-based collaborative filtering methods are simple and intuitive, matrix factorization techniques are usually more effective because they allow us to discover the latent features underlying the interactions between users and items. Of course, matrix factorization is simply a mathematical tool for playing around with matrices, and is therefore applicable in many scenarios where one would like to find out something hidden under the data.

In this tutorial, we will go through the basic ideas and the mathematics of matrix factorization, and then we will present a simple implementation in Python. We will proceed with the assumption that we are dealing with user ratings (e.g. an integer score from the range of 1 to 5) of items in a recommendation system.

2.1 Basic Idea

Just as its name suggests, matrix factorization is to, obviously, factorize a matrix, i.e. to find out two (or more) matrices such that when you multiply them you will get back the original matrix.

As mentioned above, from an application point of view, matrix factorization can be used to discover latent features underlying the interactions between two different kinds of entities. And one obvious application is to predict ratings in collaborative filtering.

In a recommendation system such as Netflix or MovieLens, there is a group of users and a set of items (movies for the above two systems). Given that each users have rated some items in the system, we would like to predict how the users would rate the items that they have not yet rated, such that we can make recommendations to the users. In this case, all the information we have about the existing ratings can be represented in a matrix. Assume now we have 5 users and 10 items, and ratings are integers ranging from 1 to 5, the matrix may look something like this:

Table 1: User Ratings

	D1	D2	D3	D4
U1	5	3	-	1
U2	4	-	-	1
U3	1	1	-	5
U4	1	-	-	4
U5	-	1	-	4

Hence, the task of predicting the missing ratings can be considered as filling in the blanks such that the values would be consistent with the existing ratings in the matrix.

The intuition behind using matrix factorization to solve this problem is that there should be some latent features that determine how a user rates an item. For example, two users would give high ratings to a certain movie if they both like the actors/actresses of the movie, or if the movie is an action movie, which is a genre preferred by both users. Hence, if we can discover these latent features, we should be able to predict a rating with respect to a certain user and a

certain item, because the features associated with the user should match with the features associated with the item.

2.2 The Mathematics of Matrix Factorization

Having discussed the intuition behind matrix factorization, we can now go on to work on the mathematics. Firstly, we have a set U of users, and a set D of items. Let $\mathbf{R}$ of size $|U| \times |D|$ be the matrix that contains all the ratings that the users have assigned to the items. Also, we assume that we would like to discover K latent features. Our task, then, is to find two matrics matrices $\mathbf{P}$ (a $|U| \times K$ matrix) and $\mathbf{Q}$ (a $|D| \times K$ matrix) such that their product approximates $\mathbf{R}$:

$$R \approx P \times Q^T = \widehat{R}$$

In this way, each row of $\mathbf{P}$ would represent the strength of the associations between a user and the features. Similarly, each row of $\mathbf{Q}$ would represent the strength of the associations between an item and the features. To get the prediction of a rating of an item d_j by u_i, we can calculate the dot product of the two vectors corresponding to u_i and d_j:

$$r_{ij} = p_i^T q_j = \sum_{k=1}^{K} p_{ik} q_{kj}$$

Now, we have to find a way to obtain $\mathbf{P}$ and $\mathbf{Q}$. One way to approach this problem is the first intialize the two matrices with some values, calculate how 'different their product is to $\mathbf{M}$, and then try to minimize this difference iteratively. Such a method is called gradient descent, aiming at finding a local minimum of the difference.

Here we consider the squared error because the estimated rating can be either higher or lower than the real rating.

To minimize the error, we have to know in which direction we have to modify the values of p_{ik} and q_{kj}. In other words, we need to know the gradient at the current values, and therefore we differentiate the above equation with respect to these two variables separately:

$$\frac{\partial}{\partial p_{ik}} e_{ij}^2 = -2(r_{ij} - \widehat{r_{ij}})(q_{kj}) = -2e_{ij}q_{kj}$$
$$\frac{\partial}{\partial q_{ik}} e_{ij}^2 = -2(r_{ij} - \widehat{r_{ij}})(p_{ik}) = -2e_{ij}p_{ik}$$

Having obtained the gradient, we can now formulate the update rules for both p_{ik} and q_{kj}:

$$p_{ik}^{'} = p_{ik} + \alpha \frac{\partial}{\partial p_{ik}} e_{ij}^2 = p_{ik} + 2\alpha e_{ij} q_{kj}$$
$$q_{kj}^{'} = q_{kj} + \alpha \frac{\partial}{\partial q_{kj}} e_{ij}^2 = q_{kj} + 2\alpha e_{ij} p_{ik}$$

Here, α is a constant whose value determines the rate of approaching the minimum. Usually we will choose a small value for α, say 0.0002. This is because if we make too large a step towards the minimum we may run into the risk of missing the minimum and end up oscillating around the minimum.

A question might have come to your mind by now: if we find two matrices $\mathbf{P}$ and $\mathbf{Q}$ such that $\mathbf{P} \times \mathbf{Q}$ approximates $\mathbf{R}$, isnt that our predictions of all the unseen ratings will all be zeros? In fact, we are not really trying to come up with $\mathbf{P}$ and $\mathbf{Q}$ such that we can reproduce $\mathbf{R}$ exactly. Instead, we will only try to minimise the errors of the observed user-item pairs. In other words, if we let T be a set of tuples, each of which is in the form of (u_i, d_j, r_{ij}), such that T contains all the observed user-item pairs together with the associated ratings, we are only trying to minimise every e_{ij} for $(u_i, d_j, r_{ij}) \in T$. (In other words, T is our set of training data.) As for the rest of the unknowns, we will be able to determine their values once the associations between the users, items and features have been learnt.

Using the above update rules, we can then iteratively perform the operation until the error converges to its minimum. We can check the overall error as calculated using the following equation and determine when we should stop the process.

$$E = p_i^T q_j = \sum_{(u_i, d_j, r_{ij}) \in T} e_{ij} = \sum_{(u_i, d_j, r_{ij}) \in T} (r_{ij} - \sum_{k=1}^{K} p_{ik} q_{kj})^2$$

2.3 Regularization

The above algorithm is a very basic algorithm for factorizing a matrix. There are a lot of methods to make things look more complicated. A common extension to this basic algorithm is to introduce regularization to avoid overfitting. This is done by adding a parameter β and modify the squared error as follows:

$$e_{ij}^2 = (r_{ij} - \sum_{k=1}^{K} p_{ik} q_{kj})^2 + \frac{\beta}{2} \sum_{k=1}^{K} (||P||^2 + ||Q||^2)^2$$

In other words, the new parameter β is used to control the magnitudes of the user-feature and item-feature vectors such that P and Q would give a good approximation of R without having to contain large numbers. In practice, β is set to some values in the range of 0.02. The new update rules for this squared error can be obtained by a procedure similar to the one described above. The new update rules are as follows.

$$p_{ik}^{'} = p_{ik} + \alpha \frac{\partial}{\partial p_{ik}} e_{ij}^2 = p_{ik} + \alpha(2e_{ij}q_{kj} - \beta p_{ik})$$
$$q_{kj}^{'} = q_{kj} + \alpha \frac{\partial}{\partial q_{kj}} e_{ij}^2 = q_{kj} + \alpha(2e_{ij}p_{ik} - \beta q_{kj})$$

2.4 Implementation in Python

```
import numpy

def matrix_factorization(R, P, Q, K, steps=5000, alpha=0.0002,
    beta=0.02):
    Q = Q.T
    for step in xrange(steps):
        for i in xrange(len(R)):
            for j in xrange(len(R[i])):
                if R[i][j] > 0:
                    eij = R[i][j] - numpy.dot(P[i,:],Q[:,j])
                    for k in xrange(K):
                        P[i][k] = P[i][k] + alpha * (2 * eij * Q[k][j] -
                            beta * P[i][k])
                        Q[k][j] = Q[k][j] + alpha * (2 * eij * P[i][k] -
                            beta * Q[k][j])
        eR = numpy.dot(P,Q)
        e = 0
        for i in xrange(len(R)):
            for j in xrange(len(R[i])):
                if R[i][j] > 0:
                    e = e + pow(R[i][j] - numpy.dot(P[i,:],Q[:,j]), 2)
                    for k in xrange(K):
                        e = e + (beta/2) * (pow(P[i][k],2) +
                            pow(Q[k][j],2))
        if e < 0.001:
            break
    return P, Q.T
```

We can try to apply it to our example mentioned above and see what we would get. Below is a code snippet in Python for running the example.

```
R = [
    [5,3,0,1],
```

```
     [4,0,0,1],
     [1,1,0,5],
     [1,0,0,4],
     [0,1,5,4],
    ]

R = numpy.array(R)

N = len(R)
M = len(R[0])
K = 2

P = numpy.random.rand(N,K)
Q = numpy.random.rand(M,K)

nP, nQ = matrix_factorization(R, P, Q, K)
nR = numpy.dot(nP, nQ.T)
```

And the matrix obtained from the above process would look something like this:

Table 2: Output of Matrix Factorization

	D1	D2	D3	D4
U1	4.97	2.98	2.18	0.98
U2	3.97	2.40	1.97	0.99
U3	1.02	0.93	5.32	4.93
U4	1.00	0.85	4.59	3.93
U5	1.36	1.07	4.89	4.12

We can see that for existing ratings we have the approximations very close to the true values, and we also get some 'predictions' of the unknown values. In this simple example, we can easily see that U1 and U2 have similar taste and they both rated D1 and D2 high, while the rest of the users preferred D3, D4 and D5. When the number of features (K in the Python code) is 2, the algorithm is able to associate the users and items to two different features, and the predictions also follow these associations. For example, we can see that the predicted rating of U4 on D3 is 4.59, because U4 and U5 both rated D4 high.

3 Content Based Recommender Systerms

Content-based recommender systems recommend an item to a user based upon a description of the item and the profile of the users interests.Systems

implementing a content-based recommendation approach analyze a set of documents and/or descriptions of items previously rated by a user, and build a model or profile of user interests based on the features of the objects rated by that user. The profile is a structured representation of user interests, adopted to recommend new interesting items. The recommendation process basically consists in matching up the attributes of the user profile against the attributes of a content object. The result is a relevance judgment that represents the users level of interest in that object. If a profile accurately reflects user preferences, it is of tremendous advantage for the effectiveness of an information access process. For instance, it could be used to filter search results by deciding whether a user is interested in a specific Web page or not and, in the negative case, preventing it from being displayed.

3.1 Advantages and Drawbacks of Content-based Filtering

The adoption of the content-based recommendation paradigm has several advantages when compared to the collaborative one:

- **User Independence** - Content-based recommenders exploit solely ratings provided by the active user to build her own profile.
- **Transparency** - Explanations on how the recommender system works can be provided by explicitly listing content features or descriptions that caused an item to occur in the list of recommendations.
- **New Item** - Content-based recommenders are capable of recommending items not yet rated by any user i.e., they do not suffer from the first-rater problem.

Nonetheless, content-based systems have several shortcomings:

- **Limited Content Analysis** - Content-based techniques have a natural limit in the number and type of features that are associated with the objects they recommend. Domain knowledge is often needed, e.g., for movie recommendations the system needs to know the actors and directors, and sometimes, domain ontologies are also needed. No content-based recommendation system can provide suitable suggestions if the analyzed content does not contain enough information to discriminate items the user likes from items the user does not like.
- **Over-Specialization** - Content-based recommenders have no inherent method for finding something unexpected. The system suggests items whose scores are high when matched against the user profile, hence the user is going to be recommended items similar to those already rated. This drawback is also called serendipity problem to highlight the tendency of the content-based systems to produce recommendations with a limited degree of novelty.

- **New User** - Enough ratings have to be collected before a content-based recommender system can really understand user preferences and provide accurate recommendations. Therefore, when few ratings are available, as for a new user, the system will not be able to provide reliable recommendations.