Course Name - Recommender Systems
Professor Name - Prof. Mamata Jenamani
Department Name - Industrial and Systems Engineering
Institute Name - Indian Institute of Technology Kharagpur
Week - 04
Lecture – 18

Lecture 18: The latent factor model

Hello everyone. We continue our discussion on model based collaborative filtering. And in this context we have in first two lectures we have seen model based collaborative filtering with respect to generic machine learning model, then we started talking about latent factor model. And specifically in the last class we saw the basic idea behind this latent factor models which is UB decomposition. With help of one example we saw that how the matrix completion problem can be solved from ah with random with after randomizing the values of U's and B and trying to minimize the error. Now, this process which we discussed with help of one example now we are going to see it formally.

And specifically in a situation when U and V you may try to optimize at the same time ok. So, let us move ahead. So, this is the content we are going to talk about unconstrained matrix factorization problem. So, this is the most fundamental form of matrix factorization and the problem is unconstrained.

In one of the earlier lecture I told you what is what an unconstrained problem is every optimization problem is going to have one objective function. So, we when we talk about any optimization problem there are two parts one is objective function and a set of constraints. So, this objective function which in this case is the loss function or the error function has to be minimized. And constraints are the set of boundaries that you put. So, beyond you put certain limit on the resources that you have.

$$\text{Minimize } J = \frac{1}{2}||R - UV^T||^2$$
$$\text{subject to:}$$
$$\text{No constraints on } U \text{ and } V$$

So, those make the constraints. So, these constraints are the this objective function is subject to these constraints. So, in this particular setting we do not have these constraints. So, therefore, it is an unconstrained matrix factorization problem ok. So, what we are trying to do basically the element wise element this is called the Frobenius norm Frobenius norm.

So, whenever we compare two matrices two matrices let us say M is one matrix and R is another matrix. So, both the matrix first of all to find on the Frobenius norm both of them

has to have the same dimension then element wise first element first element you take the subtraction take the square. So, if you do that we call it a Frobenius norm. So, in a typical optimization problem we have to find the we have to minimize the distance between these two. However, this is only possible if both R and this new matrix which will be computed using u and v transpose both of them are of same dimension and both of them have values.

So, now in the context of matrix with missing entries only a subset of entries of R are known. Therefore we cannot compute the Frobenius Frobenius norm as such. However, for whatever elements are available or whatever elements are observed we can still create one loss function. So, let S be that set is the set of all R i j all i j values where R i j that original matrix let us call it R and some of there are some entries which are missing. So, for whatever i j values a R i j is observable observed then let S be that set.

$$S = \{(i, j) : r_{ij} \text{ is observed}\}$$

$$\hat{r}_{ij} = \sum_{s=1}^{k} u_{is} \cdot v_{js}$$

$$U = [u_{is}]_{m \times k} \text{ and } V = [v_{js}]_{n \times k}$$

Now because we are taking R to be u and v transpose R i j cap R i j hat which is the estimated value of the element R i j how it has you it has to be computed this is your u matrix this is let us say this is your v matrix and this is your some R hat which value has to be computed. So, this is you are trying to get this value which is R i j. So, i th row you have to take how many entries are there in i th row because this is our latent user latent factor matrix and assuming that it has k dimensions. So, there will be k entries. So, there will be k entries here.

So, similarly this has to be multiplied with corresponding j th column. So, how many entries will be there in the j th column again k entries. So, that is how you are supposed to multiply element by element ok. So, here matrix R is of dimension this is what I have already told you. Now this is the error minimization problem this we saw this was what our R hat i j.

So, this R hat i j has to be subtracted from R i j and you take the square and take the sum of square as usual over what over whatever entries are observable. So, in the matrix R only those entries which are available for them we sum take this sum and this in these are the individual error terms. Now, this half is purposefully multiplied because when we try taking the differentiation we have to be making it 2 into this. So, therefore, this 2 gets cancelled otherwise 2 remains. So, this is immaterial whether you put half or not, but if you put half that 2 gets cancelled that is the reason for which half is here.

The modified objective function, for incomplete matrices

$$\text{Minimize } J = \frac{1}{2}\sum_{(i,j)\in S} e_{ij}^2 = \frac{1}{2}\sum_{(i,j)\in S}\left(r_{ij} - \sum_{s=1}^{k} u_{is}\cdot v_{js}\right)^2$$

subject to:

No constraints on U and V

Note:

$S = \{(i,j) : r_{ij} \text{ is observed}\}$ and $\hat{r}_{ij} = \sum_{s=1}^{k} u_{is}\cdot v_{js}$

Now, with respect to this objective function how do we find out the optimal value? To find out the optimal value every optimization problem is a search problem and in that search problem you are supposed to supposed to find out this is a again a some kind of quadratic function. So, because it is a quadratic function let us say in 2 dimension it will be having a form like this and let us say in 3 dimension it will be something like this. So, therefore, this is our point this is our point for a very small problem you can a 2 dimensional 3 dimensional are a very simple problem you can try taking the derivative in this case because there is one variable and the partial derivatives with respect to both the both the variables. Here there is one variable this is the function and this is the variable x and here you have 2 variables one x direction and y direction this is something the x something y and this is your function f. So, we have to take the partial derivative with respect to both and set them to 0.

So, that is what is and partial derivatives the vector of partial derivative is called the gradient. So, you take the partial derivative with respect to each element. So, now, the question is how many elements how many elements in u what was the dimension of u dimension of u original dimension of r was r was m cross n and because we thought of keeping k latent factors u was of dimension m cross k and v was of dimension v was of dimension n cross k. So, for how many values we are going to determine how many entries are there m into k entries here and n into k entries here. So, one such equation will be there for each of the entry for each of the entry and you take the error function.

So, total this many number of variables for which you will be finding the gradient with respect to this error function set I mean the ordinarily you have to set everything to 0 and solve. So, how many equations m into k plus n into k that many number of equations you have to solve, but so, therefore, and those equations it becomes extremely difficult now. So, therefore, you have to search for some kind of iterative procedure to do it. So, which means in each iteration you will update the values you start with certain random values and in each iteration you keep on updating. So, what is this gradient equations? So, this is

the gradient equation which we obtained from here this is the gradient equation for all the elements of u this is the gradient equation for all the elements of v.

The objective function $J = \frac{1}{2} \sum_{(i,j) \in S} e_{ij}^2 = \frac{1}{2} \sum_{(i,j) \in S} \left(r_{ij} - \sum_{s=1}^{k} u_{is} \cdot v_{js} \right)^2$

The gradients

$$\frac{\partial J}{\partial u_{iq}} = \sum_{j:(i,j) \in S} \left(r_{ij} - \sum_{s=1}^{k} u_{is} \cdot v_{js} \right) (-v_{jq}) \quad \forall i \in \{1 \ldots m\}, q \in \{1 \ldots k\}$$

$$= \sum_{j:(i,j) \in S} (e_{ij})(-v_{jq}) \quad \forall i \in \{1 \ldots m\}, q \in \{1 \ldots k\}$$

$$\frac{\partial J}{\partial v_{jq}} = \sum_{i:(i,j) \in S} \left(r_{ij} - \sum_{s=1}^{k} u_{is} \cdot v_{js} \right) (-u_{iq}) \quad \forall j \in \{1 \ldots n\}, q \in \{1 \ldots k\}$$

$$= \sum_{i:(i,j) \in S} (e_{ij})(-u_{iq}) \quad \forall j \in \{1 \ldots n\}, q \in \{1 \ldots k\}$$

So, gradient equation for so, at each iteration you have to move in the direction of the gradient in the direction of the gradient. So, actually gradient gives you the direction in which the rate of change is highest in the function. So, if you move in the ah opposite direction of the gradient your value of the function decrease. So, therefore, at any point ah I mean the you take the gradient and then try whatever point from from which you are starting you move on the surface of the function in the negative direction of that negative direction of the gradient that gives you the fastest decrease. So, correspondingly you have this.

So, here what is how do you get this error term? This error term is coming from here now look at this this ultimately is the error and when you take the partial derivative with respect to ah any of this variable then 2 1s and 2 is getting cancelled and this term this part is your error term ok. So, with this error e i j. So, this e i j is getting reflected here. So, now so, how do you improve? So, if you have given certain original value q i q you add the gradient and move in that sorry you subtract the gradient gradient and move in that direction and similarly for v this is the case sorry update equations in terms of vector form. What is u? u is the sorry in the matrix form u is the entire matrix of u with all these update terms where this is the original value this is you are adding the update and you keep on iterating.

This value of alpha which is step style sometimes it is called the learning parameter is chosen based on using some kind of standard numerical method which is ah discussed in general in non-linear programming. Now, once we have these update equations what we do? We can take all the values at a time for which the entries are observed and try updating. So, this process is called gradient descent because you are going in the negative

direction of the gradient. So, here you are selecting all the r i values which are observed not the values which are have blank entries. Now, you have to create some kind of convergence criteria like the difference between the original matrix and the new matrix or the difference between the last error function or this error function is some delta or something.

The gradients

$$\frac{\partial J}{\partial u_{iq}} = \sum_{j:(i,j)\in S} (e_{ij})(-v_{jq}) \quad \forall i \in \{1 \dots m\}, q \in \{1 \dots k\}$$

$$\frac{\partial J}{\partial v_{jq}} = \sum_{i:(i,j)\in S} (e_{ij})(-u_{iq}) \quad \forall j \in \{1 \dots n\}, q \in \{1 \dots k\}$$

The update equations

$$U \Leftarrow U + \alpha EV$$
$$V \Leftarrow V + \alpha E^T U$$

So, decide your convergence criteria and iterate this procedure. During this iteration what do you what did you do? You tried updating this randomly given values and this you did considering all the S all the elements in S at a time ok. And if the number of entries are huge then this task is going to be again very time taking because each iteration itself is going to take considerable amount of time. So, therefore, another approach is you do not have to do everything at a time. May be you can take one row at a time one row of X or one column of sorry one row of ah U or I mean the matrix R is m cross n dimension.

So, either you take this entire row all the U i values how do you get in the modified all the taking all the U i values in that or taking a column at a time. So, when I do this only the portions contributed by i j I consider for all the k elements and this is the portion contributed by i j for all the k elements. And with this for individual entries I write this. So, when I write these individual entries how is the different from the original one? Look here I was taking for all i and all j, but here we are doing for individual i and individual j of course, all the k elements which are the latent factors we are supposed to consider. So, accordingly because you are now using your gradient over here and trying to update this is your new function ok.

Algorithm *GD*(Ratings Matrix: R, Learning Rate: α)
begin
  Randomly initialize matrices U and V;
  $S = \{(i, j) : r_{ij}$ is observed$\}$;
  while not(convergence) **do**
  begin
    Compute each error $e_{ij} \in S$ as the observed entries of $R - UV^T$;
    for each user-component pair (i, q) **do** $u_{iq}^+ \Leftarrow u_{iq} + \alpha \cdot \sum_{j:(i,j)\in S} e_{ij} \cdot v_{jq}$;
    for each item-component pair (j, q) **do** $v_{jq}^+ \Leftarrow v_{jq} + \alpha \cdot \sum_{i:(i,j)\in S} e_{ij} \cdot u_{iq}$;
    for each user-component pair (i, q) **do** $u_{iq} \Leftarrow u_{iq}^+$;
    for each item-component pair (j, q) **do** $v_{jq} \Leftarrow v_{jq}^+$;
    Check convergence condition;
  end
end

So, with this new function now you do something called stochastic gradient descent. This is your entire set where R i j all R i j's are observed now this is only a subset ok you only a subset. So, for each i j you do this and you iterate till your convergence criteria is met. Now, we have already talked about the concept of regularization the ridge regression while talking about the ridge regression case. So, and why do we do it? To avoid overfitting.

$$u_{iq} \Leftarrow u_{iq} - \alpha \cdot \left[\frac{\partial J}{\partial u_{iq}}\right]_{\text{Portion contributed by } (i,j)} \qquad \forall q \in \{1 \ldots k\}$$

$$v_{jq} \Leftarrow v_{jq} - \alpha \cdot \left[\frac{\partial J}{\partial v_{jq}}\right]_{\text{Portion contributed by } (i,j)} \qquad \forall q \in \{1 \ldots k\}$$

The $2 \cdot k$ updates *specific to the observed entry* $(i, j) \in S$, are

$$u_{iq} \Leftarrow u_{iq} + \alpha \cdot e_{ij} \cdot v_{jq} \quad \forall q \in \{1 \ldots k\}$$
$$v_{jq} \Leftarrow v_{jq} + \alpha \cdot e_{ij} \cdot u_{iq} \quad \forall q \in \{1 \ldots k\}$$

In vectorized form

$$\overline{u_i} \Leftarrow \overline{u_i} + \alpha\, e_{ij}\, \overline{v_j}$$
$$\overline{v_j} \Leftarrow \overline{v_j} + \alpha\, e_{ij}\, \overline{u_i}$$

So, here also the overfitting may occur. So, therefore, it is also possible that you can add regularization terms. So, they this regularization terms are considering individual u i s and v j s. So, correspondingly you find out the gradient after you find I mean while you find out the gradient now because of these values look this of course, when you take with respect to u i q this part is assumed to be constant and this part will this part will not have any effect the last part, but here the this part will have some effect. So, therefore, you get this term and similarly moving ahead you with v s you get this term this is fairly simple you can take the partial derivative and find out and from these gradients you have your

update equation only thing is that whatever was your error now this regularization term is also getting subtracted ok.

Algorithm *SGD*(Ratings Matrix: R, Learning Rate: α)
begin
  Randomly initialize matrices U and V;
  $S = \{(i,j) : r_{ij}$ is observed$\}$;
  while not(convergence) **do**
  begin
    Randomly shuffle observed entries in S;
    for each $(i,j) \in S$ in shuffled order **do**
    begin
      $e_{ij} \Leftarrow r_{ij} - \sum_{s=1}^{k} u_{is} v_{js}$;
      for each $q \in \{1 \ldots k\}$ **do** $u_{iq}^{+} \Leftarrow u_{iq} + \alpha \cdot e_{ij} \cdot v_{jq}$;
      for each $q \in \{1 \ldots k\}$ **do** $v_{jq}^{+} \Leftarrow v_{jq} + \alpha \cdot e_{ij} \cdot u_{iq}$;
      for each $q \in \{1 \ldots k\}$ **do** $u_{iq} = u_{iq}^{+}$ and $v_{jq} = v_{jq}^{+}$;
    end
    Check convergence condition;
  end
end

$$u_{iq} \Leftarrow u_{iq} + \alpha \cdot e_{ij} \cdot v_{jq} \quad \forall q \in \{1 \ldots k\}$$
$$v_{jq} \Leftarrow v_{jq} + \alpha \cdot e_{ij} \cdot u_{iq} \quad \forall q \in \{1 \ldots k\}$$

So, that was for gradient descent and this is for stochastic gradient descent. In stochastic gradient descent individual element at a time you try considering. So, here it was the sum and here individual elements. So, this is now if we do this we can now even think of making something called component wise stochastic gradient descent. So, which means each individual entries now you try updating take for each i j belongs to s do it do it.

Regularization

The objective function

$$\text{Minimize } J = \frac{1}{2} \sum_{(i,j) \in S} e_{ij}^2 + \frac{\lambda}{2} \sum_{i=1}^{m} \sum_{s=1}^{k} u_{is}^2 + \frac{\lambda}{2} \sum_{j=1}^{n} \sum_{s=1}^{k} v_{js}^2$$
$$= \frac{1}{2} \sum_{(i,j) \in S} \left(r_{ij} - \sum_{s=1}^{k} u_{is} \cdot v_{js} \right)^2 + \frac{\lambda}{2} \sum_{i=1}^{m} \sum_{s=1}^{k} u_{is}^2 + \frac{\lambda}{2} \sum_{j=1}^{n} \sum_{s=1}^{k} v_{js}^2$$

The gradients

$$\frac{\partial J}{\partial u_{iq}} = \sum_{j:(i,j) \in S} \left(r_{ij} - \sum_{s=1}^{k} u_{is} \cdot v_{js} \right) (-v_{jq}) + \lambda u_{iq} \quad \forall i \in \{1 \ldots m\}, q \in \{1 \ldots k\}$$
$$= \sum_{j:(i,j) \in S} (e_{ij})(-v_{jq}) + \lambda u_{iq} \quad \forall i \in \{1 \ldots m\}, q \in \{1 \ldots k\}$$
$$\frac{\partial J}{\partial v_{jq}} = \sum_{i:(i,j) \in S} \left(r_{ij} - \sum_{s=1}^{k} u_{is} \cdot v_{js} \right) (-u_{iq}) + \lambda v_{jq} \quad \forall j \in \{1 \ldots n\}, q \in \{1 \ldots k\}$$
$$= \sum_{i:(i,j) \in S} (e_{ij})(-u_{iq}) + \lambda v_{jq} \quad \forall j \in \{1 \ldots n\}, q \in \{1 \ldots k\}$$

The parameter λ is always non-negative and it controls the weight of the regularization term.

So, till the convergence criteria is met. So, through this iterative procedure which is a variation of your normal gradient descent you will be saving a lot of resources computing resources and process will not stop because of lack of resource. So, you continue up to this and you stop ok. So, for this particular lecture I have taken the contents from the first book and of course, the other books are also equally important and in future while talking about this model based collaborative filtering we will be referring them and these are my

conclusions. An unconstrained matrix factorization is the fundamental of latent factor models and in today's lecture we tried looking at how to solve this unconstrained matrix factorization problem while the while some of the entries in the matrix are missing.

Regularization

The gradients

$$\frac{\partial J}{\partial u_{iq}} = \sum_{j:(i,j)\in S} \left(r_{ij} - \sum_{s=1}^{k} u_{is} \cdot v_{js} \right) (-v_{jq}) + \lambda u_{iq} \quad \forall i \in \{1 \dots m\}, q \in \{1 \dots k\}$$

$$= \sum_{j:(i,j)\in S} (e_{ij})(-v_{jq}) + \lambda u_{iq} \quad \forall i \in \{1 \dots m\}, q \in \{1 \dots k\}$$

$$\frac{\partial J}{\partial v_{jq}} = \sum_{i:(i,j)\in S} \left(r_{ij} - \sum_{s=1}^{k} u_{is} \cdot v_{js} \right) (-u_{iq}) + \lambda v_{jq} \quad \forall j \in \{1 \dots n\}, q \in \{1 \dots k\}$$

$$= \sum_{i:(i,j)\in S} (e_{ij})(-u_{iq}) + \lambda v_{jq} \quad \forall j \in \{1 \dots n\}, q \in \{1 \dots k\}$$

The update equations for gradient descent

$$u_{iq} \Leftarrow u_{iq} + \alpha \left(\sum_{j:(i,j)\in S} e_{ij} \cdot v_{jq} - \lambda \cdot u_{iq} \right) \quad \forall q \in \{1 \dots k\}$$

$$v_{jq} \Leftarrow v_{jq} + \alpha \left(\sum_{i:(i,j)\in S} e_{ij} \cdot u_{iq} - \lambda \cdot v_{jq} \right) \quad \forall q \in \{1 \dots k\}$$

Regularization

The update equations for gradient descent

$$u_{iq} \Leftarrow u_{iq} + \alpha \left(\sum_{j:(i,j)\in S} e_{ij} \cdot v_{jq} - \lambda \cdot u_{iq} \right) \quad \forall q \in \{1 \dots k\}$$

$$v_{jq} \Leftarrow v_{jq} + \alpha \left(\sum_{i:(i,j)\in S} e_{ij} \cdot u_{iq} - \lambda \cdot v_{jq} \right) \quad \forall q \in \{1 \dots k\}$$

The update equations for stochastic gradient descent

$$u_{iq} \Leftarrow u_{iq} + \alpha(e_{ij} \cdot v_{jq} - \lambda \cdot u_{iq}) \quad \forall q \in \{1 \dots k\}$$

$$v_{jq} \Leftarrow v_{jq} + \alpha(e_{ij} \cdot u_{iq} - \lambda \cdot v_{jq}) \quad \forall q \in \{1 \dots k\}$$

So, here we created one error minimization problem where from the available entries in the rating matrix we created the objective function and this error objective function had many variables and precisely same if the number of variables in that function was same as that of number of elements considering u and v together. Now, gradient of gradient with respect to each of these variable is taken and error update equations are developed for gradient descent as well as stochastic gradient descent. And finally, we also considered to make this formulation robust against overfitting by adding regularization term. After adding regularization term we again tried optimizing and we got a new error update equation which was incorporated with regularization terms and in we also looked

at element wise we can run stochastic gradient descent in case the problem becomes very large. Thank you.

Algorithm *ComponentWise-SGD*(Ratings Matrix: R, Learning Rate: α)
begin
  Randomly initialize matrices U and V;
  $S = \{(i, j) : r_{ij}$ is observed$\}$;
  for $q = 1$ to k **do**
  begin
    while not(convergence) **do**
    begin
      Randomly shuffle observed entries in S;
      for each $(i, j) \in S$ in shuffled order **do**
      begin
        $e_{ij} \Leftarrow r_{ij} - u_{iq}v_{jq}$;
        $u_{iq}^{+} \Leftarrow u_{iq} + \alpha \cdot (e_{ij} \cdot v_{jq} - \lambda \cdot u_{iq})$;
        $v_{jq}^{+} \Leftarrow v_{jq} + \alpha \cdot (e_{ij} \cdot u_{iq} - \lambda \cdot v_{jq})$;
        $u_{iq} = u_{iq}^{+}$; $v_{jq} = v_{jq}^{+}$;
      end
      Check convergence condition;
    end
    { Element-wise implementation of $R \Leftarrow R - \overline{U_q}\,\overline{V_q}^{T}$ }
    for each $(i, j) \in S$ **do** $r_{ij} \Leftarrow r_{ij} - u_{iq}v_{jq}$;
  end
end