CIS606 – Lecture 3

Woon Wei Lee, Jacob Crandall
Spring 2014, 10:00am-11:15am,
Mondays and Thursdays

For today:

- Matrix Factorization, cont'd

Matrix Factorization – column/row views

$k = 3$

$$\begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & \cdots & r_{1,n} \\ r_{2,1} & \cdots & \cdots & \cdots & r_{2,n} \\ \vdots & \cdots & \ddots & \cdots & \vdots \\ r_{m,1} & \cdots & \cdots & \cdots & r_{m,n} \end{bmatrix} = \begin{bmatrix} p_{1,1} & p_{1,2} & p_{1,3} \\ p_{2,1} & p_{2,2} & p_{2,3} \\ \vdots & \vdots & \vdots \\ p_{m,1} & \cdots & p_{m,3} \end{bmatrix} \times \begin{bmatrix} q_{1,1} & q_{1,2} & q_{1,3} & \cdots & q_{1,n} \\ q_{2,1} & \cdots & \cdots & \cdots & q_{2,n} \\ q_{3,1} & \cdots & \cdots & \cdots & q_{3,n} \end{bmatrix}$$

R P Q

Column view (item space):

$$\begin{bmatrix} r_{1,1} \\ \vdots \\ \vdots \\ r_{m,1} \end{bmatrix} = \begin{bmatrix} p_{1,1} & p_{1,2} & p_{1,3} \\ p_{2,1} & p_{2,2} & p_{2,3} \\ \vdots & \vdots & \vdots \\ p_{m,1} & \cdots & p_{m,3} \end{bmatrix} \times \begin{bmatrix} q_{1,1} \\ q_{2,1} \\ q_{3,1} \end{bmatrix} = \begin{bmatrix} q_{1,1} \cdot p_{1,1} + q_{2,1} \cdot p_{1,2} + q_{3,1} \cdot p_{1,3} \\ q_{1,1} \cdot p_{2,1} + q_{2,1} \cdot p_{2,2} + q_{3,1} \cdot p_{2,3} \\ \vdots \\ q_{1,1} \cdot p_{m,1} + q_{2,1} \cdot p_{m,2} + q_{3,1} \cdot p_{m,3} \end{bmatrix}$$

Row view (user space):

$$\begin{bmatrix} r_{1,1} & r_{1,2} & \cdots & r_{1,n} \end{bmatrix} = \begin{bmatrix} p_{1,1} & p_{1,2} & p_{1,3} \end{bmatrix} \times \begin{bmatrix} q_{1,1} & q_{1,2} & \cdots & q_{1,n} \\ q_{2,1} & \cdots & \cdots & q_{2,n} \\ q_{3,1} & \cdots & \cdots & q_{3,n} \end{bmatrix}$$

“MFCF.m”

```
% R -> matrix of ratings; if no rating provided, should be -1
% k -> number of latent factors

%% R from the lecture
if ~exist("R")
  R=ones(6,5)*-1;
  R(2,1)=4;R(4,1)=5;R(5,1)=4.5;R(6,1)=3;
  R(1,2)=2.5;R(4,2)=2;R(6,2)=5;
  R(3,3)=3;R(5,3)=3;R(6,3)=3;
  R(1,4)=5;R(4,4)=4;R(6,4)=2.5;R(4,5)=1.5;R(5,5)=3;
endif

%% Setting some parameters
rand('state',200);
num_iter=500; % Number of iterations (through the entire set of ratings)
lr=0.1; % Learning Rate
k=3; % Number of latent factors

%% Getting size of matrix
n=size(R,1); % Number of users
m=size(R,2); % Number of items/movies

%% Creating P and Q
P=rand(n,k)*1; Q=rand(k,m)*1;

%% Finding existing ratings
[foo1,foo2]=find(R>-1); ratings=[foo1,foo2];

%% Starting the training
for iteration=1:num_iter

  err=R-P*Q;

  %% Update matrix (will accumulate changes over entire epoch before adding to P and Q)
  p_update=zeros(size(P));   q_update=zeros(size(Q));

  for rating=1:length(ratings)

    i=ratings(rating,1);  j=ratings(rating,2);

    %% Calculating updates to P and Q separately
    p_update(i,:)+=lr*err(i,j)*Q(:,j)';   q_update(:,j)+=lr*err(i,j)*P(i,:)';

  endfor

  %% Updating P and Q matrices
  P+=p_update;  Q+=q_update;

endfor
```

Masdar INSTITUTE

Name\ Item	Star Wars	Jaws	Avatar	Alien	Chicken Little
Richard	?	2.5	?	5	?
Ahmad	4	?	?	?	?
Fauziah	?	?	3	?	?
Foo	5	2	?	4	1.5
Kok Hwa	4.5	?	3	?	3
Latiff	3	5	3	2.5	?

```
octave:2> R
R =

  -1.0000   2.5000  -1.0000   5.0000  -1.0000
   4.0000  -1.0000  -1.0000  -1.0000  -1.0000
  -1.0000  -1.0000   3.0000  -1.0000  -1.0000
   5.0000   2.0000  -1.0000   4.0000   1.5000
   4.5000  -1.0000   3.0000  -1.0000   3.0000
   3.0000   5.0000   3.0000   2.5000  -1.0000
```

"user"

```
octave:3> P*Q
ans =

   0.34272   2.52687  -1.00079   5.03051  -0.71402
   4.02154  -2.29753  -1.48025   1.41438   0.45060
   4.98530   3.20245   3.01199   1.09968   2.92448
   5.05794   2.04201   0.44535   4.04519   1.51778
   4.53568   0.69530   3.01550  -2.25491   3.01363
   3.05068   5.04411   3.02009   2.54337   2.22625
```

Predicted value

Masdar INSTITUTE

Matrix Factorization – problem with overfitting

(Recap) Optimization problem

$$R \approx PQ = \hat{R}$$

$$\hat{r}_{ij} = \sum_{h=1}^{k} p_{ih} q_{hj} = \boldsymbol{p}_{\boldsymbol{i}} \boldsymbol{q}_{\boldsymbol{j}}$$

$$e_{ij} = \frac{1}{2}(r_{ij} - \hat{r}_{ij})^2$$

$$SSE = \sum_{i,\, j=1}^{i=n,\, j=m} e_{ij}$$

$$(P_{opt}, Q_{opt}) = \underset{P,Q}{argmin}\, SSE$$

$$\frac{\partial e_{ij}}{\partial p_{ih}} = (\hat{r}_{ij} - r_{ij}).\, q_{hj}$$

$$\frac{\partial e_{ij}}{\partial q_{hj}} = (\hat{r}_{ij} - r_{ij}).\, p_{ih}$$

Update equations

$$p_{ih}(t+1) = p_{ih}(t) + \eta.(\hat{r}_{ij} - r_{ij}).\, q_{hj}$$

$$q_{hj}(t+1) = q_{hj}(t) + \eta.(\hat{r}_{ij} - r_{ij}).\, q_{ih}$$

- **Main problem is that this only constrains the solution matrix on the *I*'s and *j*'s which correspond to known ratings**
  - In many cases, optimization problem has too many degrees of freedom

    + as always, data is noisy, high dimensional, etc..

    → "overfitting"

(Cont'd)

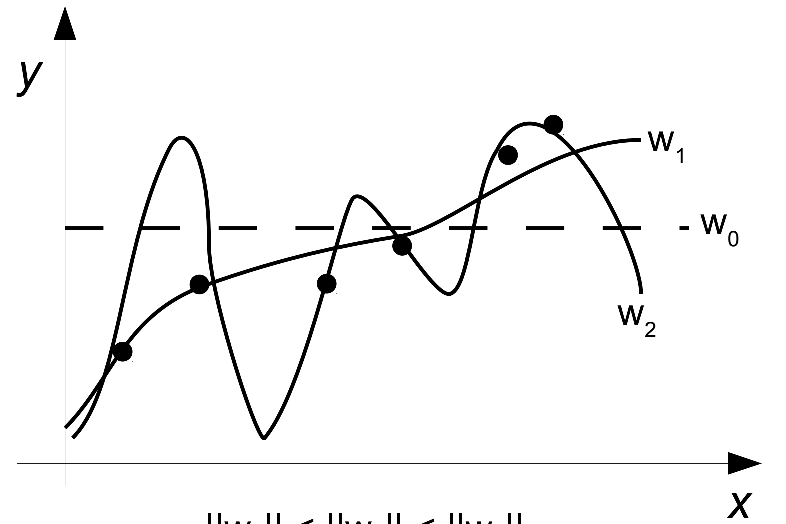
- **General approach → regularization!**
  - (refresher) – we would like to constrain the space of possible solutions
  - In many machine learning applications, a very common constraint is that the function should be "smooth"
- **Can be "encouraged" by setting small values to the weights/parameter vectors**
  - To see, this, imagine a neural network with all weights set to 0

    → "flatline"
  - To see this, remember that for a single layer NN, gradient of output w.r.t. input is:

$$y = f(w^T x)$$

$$\frac{dy}{dx} = f'(w^T x) w$$

  - For a fixed value of $w^T x$, it can be seen that the gradient of the function is directly proportional to ||w||



$||w_0|| < ||w_1|| < ||w_2||$

Regularized Matrix Factorization

- Aim: Where multiple values of *p* and *q* are "feasible"

  → prefer smaller values!

- Let's revisit Bayes' Theorem:

$$p(\theta|x) = \frac{p(x|\theta)\, p(\theta)}{p(x)}$$

$$\propto p(x|\theta)\, p(\theta)$$

- In this case:

$$\theta = \{p_{ih}, q_{hj} | i = 1..n, h = 1..k, j = 1..m\}$$

- For standard least squares optimization → minimize the -ve log likelihood:

$$p_{ih}^{*} = \underset{p_{ih}}{argmin}\left[-\log p(x|\theta)\right]$$

$$= \underset{p_{ih}}{argmin}\left[-\log \exp\left(\frac{-(r_{ij} - \hat{r}_{ij})^2}{\sigma^2}\right)\right]$$

$$= \underset{p_{ih}}{argmin}(r_{ij} - \hat{r}_{ij})^2$$

- Bayes' rule provides a built in technique for stating our "preferences"
  - By adjusting the prior distribution for the parameters
  - Optimize the posterior distribution instead
  - To reduce the magnitude of the "weights", we impose a zero mean gaussian prior on the weights

$$p_{ih}^{*} = \underset{p_{ih}}{argmin}\left[-\log p(x|\theta)\, p(\theta)\right]$$

$$= \underset{p_{ih}}{argmin}\left[-\log \exp\left(\frac{-(r_{ij} - \hat{r}_{ij})^2}{\sigma^2}\right)\right] \ldots$$

$$\ldots - \log\left[\exp\left(\frac{-p_{ih}^2}{\sigma_w^2}\right)\right]$$

$$= \underset{p_{ih}}{argmin}\left[\zeta_1 (r_{ij} - \hat{r}_{ij})^2 + \zeta_2\, p_{ih}^2\right]$$

Update equations

$$p_{ih}(t+1) = p_{ih}(t) + \eta_1 . (\hat{r}_{ij} - r_{ij}) . q_{hj} - \eta_2\, p_{ih}$$

$$q_{hj}(t+1) = q_{hj}(t) + \eta_1 . (\hat{r}_{ij} - r_{ij}) .\, p_{ih} - \eta_2\, q_{hi}$$

rmfcf_v2.m

```
err=R-P*Q;
%% disp(["Iteration: ",num2str(iteration),". Total error is ",num2str(sum(sum(err.^2)))]);

%% Update matrix (will accumulate changes over entire epoch before adding to P and Q)
p_update=zeros(size(P));
q_update=zeros(size(Q));

for rating=1:length(ratings)

  i=ratings(rating,1);
  j=ratings(rating,2);

  %% Calculating updates to P and Q separately
  p_update(i,:)+=lr*err(i,j)*Q(:,j)'-lambda*P(i,:);
  q_update(:,j)+=lr*err(i,j)*P(i,:)'-lambda*Q(:,j);

endfor

%% Updating P and Q matrices
P+=p_update;
Q+=q_update;

endfor
```

Modified update rule

testregularize.m

```
rmat={};rrmat={};

for count=1:10
  mfcf_v3;
  rmat{count}=P*Q.*(R==-1);
end

for count=1:10
  rmfcf_v2;
  rrmat{count}=P*Q.*(R==-1);
end

%% Scoring

score=0;rscore=0;

for c1=1:10
  for c2=c1:10
    score+=sum(sum((rmat{c1}-rmat{c2}).^2));
    rscore+=sum(sum((rrmat{c1}-rrmat{c2}).^2));
  end
end

disp(['Error on "missing" ratings, without regularization:',num2str(score),', and with regularzation:',num2str(rscore)])
```

Masdar INSTITUTE

Cont'd

- **Note**
  - k=3
  - Errors on the *known* ratings have increased a little
  - However, inconsistencies with the unknown (*predicted*) ratings have dropped tremendously

```
octave-3.2.3:8> testregularize
Final error on trained ratings is (unregularized): 119.11
Final error on trained ratings is (unregularized): 257.65
Final error on trained ratings is (unregularized): 129.04
Final error on trained ratings is (unregularized): 79.519
Final error on trained ratings is (unregularized): 123.05
Final error on trained ratings is (unregularized): 136.65
Final error on trained ratings is (unregularized): 193.67
Final error on trained ratings is (unregularized): 196.04
Final error on trained ratings is (unregularized): 116.91
Final error on trained ratings is (unregularized): 171.53
Final error on trained ratings is (regularized): 174.68
Final error on trained ratings is (regularized): 211.85
Final error on trained ratings is (regularized): 211.48
Final error on trained ratings is (regularized): 149.51
Final error on trained ratings is (regularized): 148.49
Final error on trained ratings is (regularized): 154.15
Final error on trained ratings is (regularized): 148.9
Final error on trained ratings is (regularized): 148.22
Final error on trained ratings is (regularized): 211.67
Final error on trained ratings is (regularized): 148.32
Error on "missing" ratings, without regularization:7145.4, and with regularzation:210.8
octave-3.2.3:9>
```

(!)

Cont'd

- **Note**
  - k=2
  - Errors on the known ratings have actually decreased when using regularization
  - Inconsistencies with the unknown (*predicted*) ratings still far superior

```
octave-3.2.3:10> testregularize
Final error on trained ratings is (unregularized): 209.3
Final error on trained ratings is (unregularized): 221.08
Final error on trained ratings is (unregularized): 200.84
Final error on trained ratings is (unregularized): 220.12
Final error on trained ratings is (unregularized): 224.74
Final error on trained ratings is (unregularized): 310.74
Final error on trained ratings is (unregularized): 313.11
Final error on trained ratings is (unregularized): 223.32
Final error on trained ratings is (unregularized): 199.36
Final error on trained ratings is (unregularized): 268.01
Final error on trained ratings is (regularized): 180.52
Final error on trained ratings is (regularized): 180.58
Final error on trained ratings is (regularized): 181.11
Final error on trained ratings is (regularized): 180.53
Final error on trained ratings is (regularized): 181.11
Final error on trained ratings is (regularized): 180.5
Final error on trained ratings is (regularized): 181.11
Final error on trained ratings is (regularized): 180.53
Final error on trained ratings is (regularized): 180.51
Final error on trained ratings is (regularized): 181.11
Error on "missing" ratings, without regularization:7304.2, and with regularzation:593.78
octave-3.2.3:11>
```

Incremental updates

- **Incremental updates of database important**
  - How to deal with addition of new users and items into the dataset
- **With this matrix factorization approach, possible to do efficiently**
  - Addition of new users only require updating of rows of P
  - Columns of Q are theoretically static w.r.t. new users

    Q: why?
  - Addition of new items

    → update specific columns of Q
  - Again, rows in P should theoretically be unaffected
  - In practice → non-essential updates batched.

To add new user:

1. Create new row in P, (p_{n+1})
2. Update all elements in p_{n+1}, as follows:

$$p_{(n+1)h}(t+1) = p_{(n+1)h}(t) + \eta . (\hat{r}_{(n+1)j} - r_{(n+1)j}) . q_{hj}$$

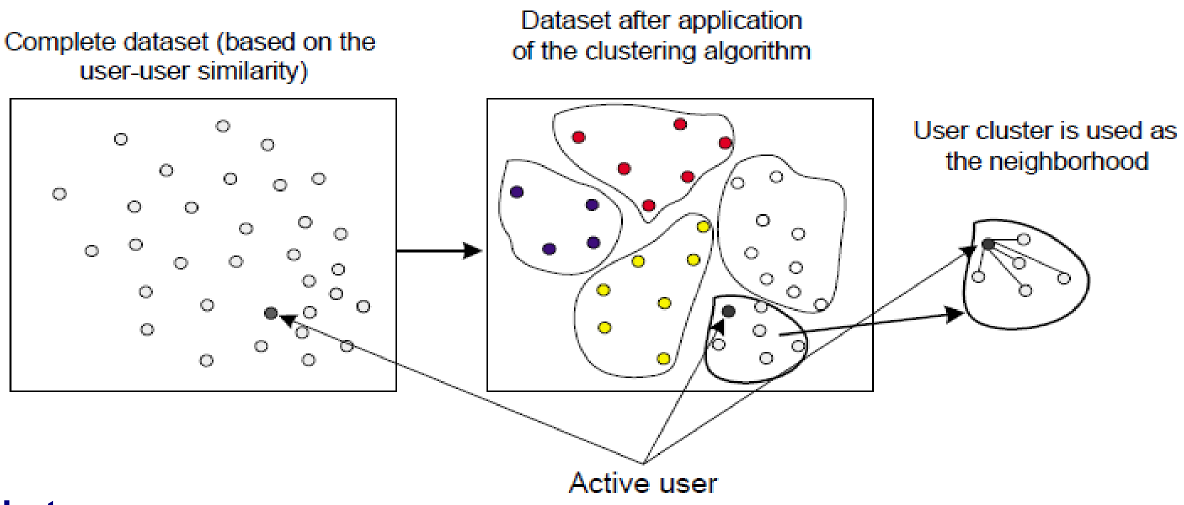
To add new item:

1. Create new column in R (r_{n+1})
2. Update all elements in q_{n+1}, as follows:

$$q_{h(m+1)}(t+1) = p_{h(m+1)}(t) + \eta . (\hat{r}_{i(m+1)} - r_{i(m+1)}) . p_{ih}$$

Masdar INSTITUTE

Hybrid model+memory based: "ClustKNN"



- **Recall that:**
  - Memory based algorithms are conceptually simple, easy to implement and can take in new preference information quite quickly

    → However, nearest neighbor type techniques scale poorly with the size of the dataset

    → poor online performance in the case of very large datasets
  - Model based algorithms can be more elegant, potentially more powerful with greater scope for development

    → However, more complex, might require models to be recalibrated if new information is received frequently
- **Many solutions proposed, an interesting one is "ClustKNN"**
  - Presents itself as a hybrid approach – combines the benefits of model-building, with the simplicity/intuitiveness of a memory based approach

(Cont'd)

- **The ClustKNN Algorithm is roughly composed of the following two stages:**

1. **Model building**
   - Select number of clusters *k*
   - Perform k-means clustering on the user data
   - For each cluster, create a *surrogate user* → { c_1, c_2 … c_k }

     (each c_i is an *m*-dimensional vector representing a "virtual user" , obtained by calculating the centroid (mean vector) for each cluster.)

2. **Prediction Generation – to predict rating prediction for user a and item *h*:**
   - Compute similarity of the target user with each of the surrogate model users who have rated *h*
   - Any measure can be used though Pearson measure is recommended in [Rashid '06]

$$w_{ij} = \frac{\sum_{k \in I} (r_{i,k} - \overline{r_i})(r_{j,k} - \overline{r_j})}{\sqrt{\sum_{k \in I} (r_{i,k} - \overline{r_i})^2} \sqrt{\sum_{k \in I} (r_{j,k} - \overline{r_j})^2}}$$

   - Find the *l* most similar surrogate users
   - Compute prediction using adjusted weight average:

$$p_{a,h} = \overline{r_a} + \kappa \sum_{u=1}^{l} w(a, c_u) * (r_{c_u, h} - \overline{r_{c_u}})$$

Probabilistic interpretation

- **Aim: can we express the matrix factorization based CF in a probabilistic framework?**
  - Consider simple case of “implicit ratings” → i.e. ratings are 1 or 0
  - What we would often like to find is the probability that user *a* would like to view/read/consume item *h*, given matrix R
- **There is a very rich family of models which we can call upon.**
  - For this case, the corresponding model is known as *probabilistic Latent Semantic Analysis* (pLSA)
  - Model is based on:
    - Latent variable *z*
    - conditional multinomial event models

      (Seems familiar?!)
    - Inputs are <a,h> tuples
  - Probability of data is given by:

$$p(a,h,z)=p(h|z)\,p(z|a)\,p(a)$$

Probabilistic interpretation (Cont'd)

- **The training process, in this case, is to approximate all the conditional probabilities,** *p(h|z), p(z|a)*.
  - p(a) typically assumed to be uniform. Hence:

$$p(h,z)=p(h|z)\,p(z|a)\,p(a)$$

$$p(h|a)=\sum_{i=1}^{kp}(h|z_i)\,p(z_i|a)$$

  - Convenient way to express all the products in matrix format:

$$\begin{bmatrix} p(h_1|a_1) & \cdots & p(h_m|a_1) \\ \vdots & \ddots & \vdots \\ p(h_1|a_n) & \cdots & p(h_m|a_n) \end{bmatrix} = \begin{bmatrix} p(z_1|a_1) & \cdots & p(z_k|a_1) \\ \vdots & \ddots & \vdots \\ p(z_1|a_n) & \cdots & p(z_k|a_n) \end{bmatrix} \cdot \begin{bmatrix} p(h_1|z_1) & \cdots & p(h_m|z_1) \\ \vdots & \ddots & \vdots \\ p(h_1|z_k) & \cdots & p(h_m|z_k) \end{bmatrix}$$

$$\underset{n\times m}{R} = \underset{n\times k}{P}\ \underset{k\times m}{Q}$$

Similarity to text/document mining

- **Problems encountered in CF are very similar to problems encountered in document mining**
  - Sparsity
  - Fast/incremental updates important
  - High noise levels
- **Algorithms used also closely related**
  - User-user similarity matrix → document similarity metrics using cosine-similarity
    - Information retrieval
  - Clustering based CF approaches → document clustering
    - Visualization of search results
  - Decomposition of ratings matrix into user and item feature matrices – long and important thread in text mining research:
    - SVD applied to ratings matrix
      - LSA (detection of "concepts")
    - Factorization into "P" and "Q" → similar to pLSA
    - Provides avenue for future enhancements, e.g. LDA
- **As always, overfitting, noise, model determination, etc → universal problems**
  - Regularization approach described earlier is standard solution
  - In neural networks, known as "weight decay"

References

- "Scalable Collaborative Filtering Approaches for Large Recommender Systems", Takacs et al, *Journal of Machine Learning Research*, 2009
- "A Survey of Collaborative Filtering Techniques", Su and Khoshgoftaar, *Advances in Artificial Intelligence,* 2009.
- "Slope One Predictors for Online Rating Based Collaborative Filtering", Lemire and Maclachlan, *SDM'05*, Newport Beach, 2005.
- "Amazon.com Recommendations – Item-to-item Collaborative Filtering", Linden et al, IEEE *Industry Report*, 2003.
- "Item-based Collaborative Filtering Recommendation Algorithms" – Sarwar et al, *WWW10*, Hong Kong, 2001.
- "Empirical Analysis of Predictive Algorithms for Collaborative Filtering", Breese et al, Technical Report MSR-TR-9-12, Microsoft Research, 1998.

+ Gratuitious use of Wikipedia, Google, *et al!*