Recommendation Systems II

Krzysztof Dembczyński

Intelligent Decision Support Systems Laboratory (IDSS)
Poznań University of Technology, Poland

Software Development Technologies
Master studies, second semester
Academic year 2016/17 (winter course)

Review of the previous lectures

- Mining of massive datasets.
- Evolution of database systems.
- MapReduce.
- Classification and regression.
- Nearest neighbor search.
- Recommendation systems:
  - Content-based systems,
  - Collaborative filtering: nearest-neighbor algorithms, matrix factorization.

Outline

1. Matrix Factorization
2. Summary

Outline

Utility matrix

- A utility matrix $\boldsymbol{Y}$ offers known information about the degree to which a user likes an item.

Utility matrix

- A utility matrix $\boldsymbol{Y}$ offers known information about the degree to which a user likes an item.
- Most entries are unknown, and the essential problem of recommending items to users is predicting the values of the unknown entries based on the values of the known entries.

Utility matrix

- A utility matrix $\boldsymbol{Y}$ offers known information about the degree to which a user likes an item.
- Most entries are unknown, and the essential problem of recommending items to users is predicting the values of the unknown entries based on the values of the known entries.
- **Example**:

	HP1	HP2	HP3	TW	SW1	SW2	SW3
A	4			5	1		
B	5	5	4				
C				2	4	5	
D		3					3

Utility matrix

- A utility matrix $\boldsymbol{Y}$ offers known information about the degree to which a user likes an item.
- Most entries are unknown, and the essential problem of recommending items to users is predicting the values of the unknown entries based on the values of the known entries.
- **Example**:

	HP1	HP2	HP3	TW	SW1	SW2	SW3
A	4			5	1		
B	5	5	4				
C				2	4	5	
D		3					3

- It is not necessary to predict every blank entry in a utility matrix: it is enough to discover some entries in each row that are likely to be high.

Matrix factorization

- One way of predicting the blank values in a utility matrix is to find two long, thin matrices $\mathbf{U}$ and $\mathbf{M}$, whose product is an approximation to the given utility matrix.
- Since the matrix product $\mathbf{U}\mathbf{M}^\top$ gives values for all user-item pairs, that value can be used to predict the value of a blank in the utility matrix.
- The intuitive reason this method makes sense is that often there are a relatively small number of issues (that number is the "thin" dimension of $\mathbf{U}$ and $\mathbf{M}$) that determine whether or not a user likes an item.

Matrix factorization

- Given matrix $\mathbf{Y}$ containing observed values with possible gaps (denoted by $y_{ij} = ?$) build a model based on matrix factorization:

$$\mathbf{Y} \approx \mathbf{Y}' = \mathbf{U}\mathbf{M}^\top$$

  where $\mathbf{U}$ is an $I \times K$ and $\mathbf{M}^\top$ is a $K \times J$ matrix.

- For example, I is the number of users, J is the number of movies in the movie recommender system, and K is number of features describing users and movies.

Matrix factorization

- When $\mathbf{U}$ is fixed, each row is a linear problem in which rows of $\mathbf{U}$ are features vectors and columns of $\mathbf{M}$ are linear classifiers.

$$\hat{\mathbf{Y}} = \begin{bmatrix} 4 & 7 & 5 \\ 5 & 8 & 7 \\ 7 & 12 & 9 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 2 & 1 \\ 2 & 3 \end{bmatrix} \times \begin{bmatrix} 2 & 3 & 3 \\ 1 & 2 & 1 \end{bmatrix}$$

Matrix factorization

- When $\mathbf{U}$ is fixed, each row is a linear problem in which rows of $\mathbf{U}$ are features vectors and columns of $\mathbf{M}$ are linear classifiers.

$$\hat{\mathbf{Y}} = \begin{bmatrix} 4 & 7 & 5 \\ 5 & 8 & 7 \\ 7 & 12 & 9 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 2 & 1 \\ 2 & 3 \end{bmatrix} \times \begin{bmatrix} 2 & 3 & 3 \\ 1 & 2 & 1 \end{bmatrix}$$

- Matrix factorization is learning features that work well across all classification problems.

Matrix factorization

- When $\mathbf{U}$ is fixed, each row is a linear problem in which rows of $\mathbf{U}$ are features vectors and columns of $\mathbf{M}$ are linear classifiers.

$$\hat{\mathbf{Y}} = \begin{bmatrix} 4 & 7 & 5 \\ 5 & 8 & 7 \\ 7 & 12 & 9 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 2 & 1 \\ 2 & 3 \end{bmatrix} \times \begin{bmatrix} 2 & 3 & 3 \\ 1 & 2 & 1 \end{bmatrix}$$

- Matrix factorization is learning features that work well across all classification problems.
- The question is how to learn this features?

Rank-1 matrix factorization

- Consider the simplest case in which $\mathbf{U} = \boldsymbol{u}$ and $\mathbf{M} = \boldsymbol{m}$.

Rank-1 matrix factorization

- Consider the simplest case in which $\mathbf{U} = \boldsymbol{u}$ and $\mathbf{M} = \boldsymbol{m}$.
- The problem can be formulated from the learning perspective as:

$$(\boldsymbol{u}, \boldsymbol{m})^* = \arg\min \sum_{y_{ij} \neq ?} \ell(y_{ij}, u_i m_j)$$

Rank-1 matrix factorization

- Consider the simplest case in which $\mathbf{U} = \boldsymbol{u}$ and $\mathbf{M} = \boldsymbol{m}$.
- The problem can be formulated from the learning perspective as:

$$(\boldsymbol{u}, \boldsymbol{m})^* = \arg\min \sum_{y_{ij} \neq ?} \ell(y_{ij}, u_i m_j)$$

- Consider the squared-error loss:

$$L = \sum_{y_{ij} \neq ?} \ell_{se}(y_{ij}, \hat{y}_{ij}) = \sum_{y_{ij} \neq ?} (y_{ij} - \hat{y}_{ij})^2 \,.$$

Rank-1 matrix factorization

- Consider the simplest case in which $\mathbf{U} = \boldsymbol{u}$ and $\mathbf{M} = \boldsymbol{m}$.
- The problem can be formulated from the learning perspective as:

$$(\boldsymbol{u}, \boldsymbol{m})^* = \arg\min \sum_{y_{ij} \neq ?} \ell(y_{ij}, u_i m_j)$$

- Consider the squared-error loss:

$$L = \sum_{y_{ij} \neq ?} \ell_{se}(y_{ij}, \hat{y}_{ij}) = \sum_{y_{ij} \neq ?} (y_{ij} - \hat{y}_{ij})^2 \,.$$

- Unfortunately, the problem is not convex :(

Rank-1 matrix factorization

- Consider the simplest case in which $\mathbf{U} = \boldsymbol{u}$ and $\mathbf{M} = \boldsymbol{m}$.
- The problem can be formulated from the learning perspective as:

$$(\boldsymbol{u}, \boldsymbol{m})^* = \arg\min \sum_{y_{ij} \neq ?} \ell(y_{ij}, u_i m_j)$$

- Consider the squared-error loss:

$$L = \sum_{y_{ij} \neq ?} \ell_{se}(y_{ij}, \hat{y}_{ij}) = \sum_{y_{ij} \neq ?} (y_{ij} - \hat{y}_{ij})^2 \,.$$

- Unfortunately, the problem is not convex :(
- To solve the optimization problem one usually uses alternating least squares.

Matrix factorization

- Approximation of $\mathbf{Y}$ by a rank-1 matrix is not sufficient.

Matrix factorization

- Approximation of $\mathbf{Y}$ by a rank-1 matrix is not sufficient.
- Possible extensions of the model:

Matrix factorization

- Approximation of $\mathbf{Y}$ by a rank-1 matrix is not sufficient.
- Possible extensions of the model:
  - Use of a larger number of features,

Matrix factorization

- Approximation of $\mathbf{Y}$ by a rank-1 matrix is not sufficient.
- Possible extensions of the model:
  - Use of a larger number of features,
  - Regularization,

Matrix factorization

- Approximation of $\mathbf{Y}$ by a rank-1 matrix is not sufficient.
- Possible extensions of the model:
  - Use of a larger number of features,
  - Regularization,
  - Large-scale learning algorithms.

Larger number of latent features

- Consider the case in which $\mathbf{U}$ and $\mathbf{M}$ contains up to K features:

$$\mathbf{Y} = \hat{\mathbf{Y}} = \mathbf{U}\mathbf{M}^\top$$

Larger number of latent features

- Consider the case in which $\mathbf{U}$ and $\mathbf{M}$ contains up to K features:

$$\mathbf{Y} = \hat{\mathbf{Y}} = \mathbf{U}\mathbf{M}^\top$$

- For example:

$$\hat{\mathbf{Y}} = \begin{bmatrix} 4 & 7 & 5 \\ 5 & 8 & 7 \\ 7 & 12 & 9 \end{bmatrix} = \mathbf{U}\mathbf{M}^\top = \begin{bmatrix} 1 & 2 \\ 2 & 1 \\ 2 & 3 \end{bmatrix} \times \begin{bmatrix} 2 & 3 & 3 \\ 1 & 2 & 1 \end{bmatrix}$$

Larger number of latent features

- Consider the case in which $\mathbf{U}$ and $\mathbf{M}$ contains up to K features:

$$\mathbf{Y} = \hat{\mathbf{Y}} = \mathbf{U}\mathbf{M}^\top$$

- For example:

$$\hat{\mathbf{Y}} = \begin{bmatrix} 4 & 7 & 5 \\ 5 & 8 & 7 \\ 7 & 12 & 9 \end{bmatrix} = \mathbf{U}\mathbf{M}^\top = \begin{bmatrix} 1 & 2 \\ 2 & 1 \\ 2 & 3 \end{bmatrix} \times \begin{bmatrix} 2 & 3 & 3 \\ 1 & 2 & 1 \end{bmatrix}$$

- The prediction is now $\hat{y}_{ij} = \sum_{k=1}^{K} u_{ik} m_{jk}$.

Larger number of latent features

- Consider the case in which $\mathbf{U}$ and $\mathbf{M}$ contains up to K features:

$$\mathbf{Y} = \hat{\mathbf{Y}} = \mathbf{U}\mathbf{M}^\top$$

- For example:

$$\hat{\mathbf{Y}} = \begin{bmatrix} 4 & 7 & 5 \\ 5 & 8 & 7 \\ 7 & 12 & 9 \end{bmatrix} = \mathbf{U}\mathbf{M}^\top = \begin{bmatrix} 1 & 2 \\ 2 & 1 \\ 2 & 3 \end{bmatrix} \times \begin{bmatrix} 2 & 3 & 3 \\ 1 & 2 & 1 \end{bmatrix}$$

- The prediction is now $\hat{y}_{ij} = \sum_{k=1}^{K} u_{ik} m_{jk}$.
- The problem can be formulated as:

$$(\mathbf{U}, \mathbf{M})^* = \underset{(\mathbf{U},\mathbf{M})}{\arg\min} \sum_{y_{ij} \neq ?} \ell(y_{ij}, \sum_{k=1}^{K} u_{ik} m_{jk}) \,.$$

Larger number of latent features

- Optimization approaches:

Larger number of latent features

- Optimization approaches:
  - A use of a coordinate descent approach, which is a straight-forward extension of the algorithm for rank-1 matrix factorization,

Larger number of latent features

- Optimization approaches:
  - A use of a coordinate descent approach, which is a straight-forward extension of the algorithm for rank-1 matrix factorization,
  - Treat the problem as a regular linear regression task and use standard algorithms,

Larger number of latent features

- Optimization approaches:
  - A use of a coordinate descent approach, which is a straight-forward extension of the algorithm for rank-1 matrix factorization,
  - Treat the problem as a regular linear regression task and use standard algorithms,
  - Stochastic gradient descent in a large-scale setting.

Coordinate descent

- The coordinate descent algorithm updates a single element of matrix $\mathbf{U}$ or matrix $\mathbf{M}$ in one iteration.

Coordinate descent

- The coordinate descent algorithm updates a single element of matrix $\mathbf{U}$ or matrix $\mathbf{M}$ in one iteration.
- All the other values of $\mathbf{U}$ and $\mathbf{M}$ are fixed.

Coordinate descent

- The coordinate descent algorithm updates a single element of matrix $\mathbf{U}$ or matrix $\mathbf{M}$ in one iteration.
- All the other values of $\mathbf{U}$ and $\mathbf{M}$ are fixed.
- The corresponding updates are the following:

$$u_{ik}^{*} = \frac{\sum_{j: y_{ij} \neq ?} m_{jk} \left(y_{ij} - \sum_{k' \neq k} u_{ik'} m_{jk'} \right)}{\sum_{j: y_{ij} \neq ?} m_{jk}^{2}} .$$

$$m_{jk}^{*} = \frac{\sum_{i: y_{ij} \neq ?} u_{ik} \left(y_{ij} - \sum_{k' \neq k} u_{ik'} m_{jk'} \right)}{\sum_{i: y_{ij} \neq ?} u_{ik}^{2}} .$$

Optimization algorithm

- Ordering the optimization of the elements of $\mathbf{U}$ and $\mathbf{M}$:

Optimization algorithm

- Ordering the optimization of the elements of $\mathbf{U}$ and $\mathbf{M}$:
  - The simplest thing to do is pick an order, e.g., row-by-row, for the elements of $\mathbf{U}$ and $\mathbf{M}$, and visit them in round-robin fashion.

Optimization algorithm

- Ordering the optimization of the elements of $\mathbf{U}$ and $\mathbf{M}$:
  - The simplest thing to do is pick an order, e.g., row-by-row, for the elements of $\mathbf{U}$ and $\mathbf{M}$, and visit them in round-robin fashion.
  - Randomly pick the element to optimize.

Optimization algorithm

- Ordering the optimization of the elements of $\mathbf{U}$ and $\mathbf{M}$:
  - The simplest thing to do is pick an order, e.g., row-by-row, for the elements of $\mathbf{U}$ and $\mathbf{M}$, and visit them in round-robin fashion.
  - Randomly pick the element to optimize.
  - The problem can also be solved by boosting-like approach:

Optimization algorithm

- Ordering the optimization of the elements of $\mathbf{U}$ and $\mathbf{M}$:
  - The simplest thing to do is pick an order, e.g., row-by-row, for the elements of $\mathbf{U}$ and $\mathbf{M}$, and visit them in round-robin fashion.
  - Randomly pick the element to optimize.
  - The problem can also be solved by boosting-like approach:
    - Compute a solution for $k = 1$,

Optimization algorithm

- Ordering the optimization of the elements of $\mathbf{U}$ and $\mathbf{M}$:
  - The simplest thing to do is pick an order, e.g., row-by-row, for the elements of $\mathbf{U}$ and $\mathbf{M}$, and visit them in round-robin fashion.
  - Randomly pick the element to optimize.
  - The problem can also be solved by boosting-like approach:
    - Compute a solution for k $= 1$,
    - In each next iteration compute a solution for a consecutive k (up to K) using the intermediate predictions of the form

$$\hat{y}_{ij}^{(k)} = \sum_{k'=1}^{k-1} u_{ik'} m_{jk'} .$$

Regularization

- To avoid overfitting the problem should be formulated with a kind of regularization.

Regularization

- To avoid overfitting the problem should be formulated with a kind of regularization.
- For example, we can penalize high values of u_{ik} and m_{jk}.

Regularization

- To avoid overfitting the problem should be formulated with a kind of regularization.
- For example, we can penalize high values of u_{ik} and m_{jk}.
- In other words, we shrink the values towards zero.

Regularization

- To avoid overfitting the problem should be formulated with a kind of regularization.
- For example, we can penalize high values of u_{ik} and m_{jk}.
- In other words, we shrink the values towards zero.
- This makes only sense if we first normalize the values of $\mathbf{Y}$ in a way that the average value is around 0.

Regularization

- To avoid overfitting the problem should be formulated with a kind of regularization.
- For example, we can penalize high values of u_{ik} and m_{jk}.
- In other words, we shrink the values towards zero.
- This makes only sense if we first normalize the values of $\mathbf{Y}$ in a way that the average value is around 0.
- The regularized problem can be formulated as:

$$(\mathbf{U}, \mathbf{M})^* = \arg\min_{(\mathbf{U},\mathbf{M})} \sum_{y_{ij} \neq ?} \left(y_{ij} - \sum_{k=1}^{K} u_{ik} m_{jk} \right)^2 + \lambda \left(\sum_{ik} u_{ik}^2 + m_{jk}^2 \right) .$$

Regularization

- To avoid overfitting the problem should be formulated with a kind of regularization.
- For example, we can penalize high values of u_{ik} and m_{jk}.
- In other words, we shrink the values towards zero.
- This makes only sense if we first normalize the values of $\mathbf{Y}$ in a way that the average value is around 0.
- The regularized problem can be formulated as:

$$(\mathbf{U}, \mathbf{M})^* = \arg\min_{(\mathbf{U},\mathbf{M})} \sum_{y_{ij} \neq ?} \left(y_{ij} - \sum_{k=1}^{K} u_{ik} m_{jk} \right)^2 + \lambda \left(\sum_{ik} u_{ik}^2 + m_{jk}^2 \right) .$$

- Parameter λ should be tuned empirically.

Regularization

- The updates in the coordinate descent algorithm become now:

Regularization

- The updates in the coordinate descent algorithm become now:

$$u_{ik}^* = \frac{\sum_{j:y_{ij}\neq ?} m_{jk} \left(y_{ij} - \sum_{k'\neq k} u_{ik'} m_{jk'} \right)}{\sum_{j:y_{ij}\neq ?} m_{jk}^2 + \lambda} .$$

$$m_{jk}^* = \frac{\sum_{i:y_{ij}\neq ?} u_{ik} \left(y_{ij} - \sum_{k'\neq k} u_{ik'} m_{jk'} \right)}{\sum_{i:y_{ij}\neq ?} u_{ik}^2 + \lambda} .$$

Large-scale learning algorithms

- The traditional least squares algorithms can be inefficient in the large-scale problems.

Large-scale learning algorithms

- The traditional least squares algorithms can be inefficient in the large-scale problems.
- A good solution in this case is the stochastic gradient descent.

Large-scale learning algorithms

- The traditional least squares algorithms can be inefficient in the large-scale problems.
- A good solution in this case is the stochastic gradient descent.
- The stochastic gradient updates are defined for a single training example:

Large-scale learning algorithms

- The traditional least squares algorithms can be inefficient in the large-scale problems.
- A good solution in this case is the stochastic gradient descent.
- The stochastic gradient updates are defined for a single training example:

$$u_{ik} \leftarrow u_{ik} + \nu(y_{ij} - \hat{y}_{ij})m_{jk}\,,$$
$$m_{jk} \leftarrow m_{jk} + \nu(y_{ij} - \hat{y}_{ij})u_{ik}\,,$$

where ν is the learning rate and $\hat{y}_{ij} = \sum_k u_{ik}m_{jk}$.

Large-scale learning algorithms

- The traditional least squares algorithms can be inefficient in the large-scale problems.
- A good solution in this case is the stochastic gradient descent.
- The stochastic gradient updates are defined for a single training example:

$$u_{ik} \leftarrow u_{ik} + \nu(y_{ij} - \hat{y}_{ij})m_{jk} \, ,$$
$$m_{jk} \leftarrow m_{jk} + \nu(y_{ij} - \hat{y}_{ij})u_{ik} \, ,$$

  where ν is the learning rate and $\hat{y}_{ij} = \sum_k u_{ik} m_{jk}$.
- There is also a question about ordering the updates: the approaches discussed earlier can be used here as well.

Large-scale learning algorithms

- The stochastic gradient descent can also be used with regularization.

- The stochastic gradient descent can also be used with regularization.
- The update has then the following form:

$$u_{ik} \leftarrow u_{ik} + \nu((y_{ij} - \hat{y}_{ij})m_{jk} - \lambda u_{ik}),$$
$$m_{jk} \leftarrow m_{jk} + \nu((y_{ij} - \hat{y}_{ij})u_{ik} - \lambda m_{jk}).$$

where λ is the regularization parameter.

Ending the attempt at optimization

- Updates of u_{ik} and m_{jk} are usually repeated until the error improves (more than a given threshold).

Ending the attempt at optimization

- Updates of u_{ik} and m_{jk} are usually repeated until the error improves (more than a given threshold).
- Try different setting in the experiments (different values λ, ν, etc.).

Ending the attempt at optimization

- Updates of u_{ik} and m_{jk} are usually repeated until the error improves (more than a given threshold).
- Try different setting in the experiments (different values λ, ν, etc.).
- Start several times as there are no guarantees to find the global minimum.

Ending the attempt at optimization

- Updates of u_{ik} and m_{jk} are usually repeated until the error improves (more than a given threshold).
- Try different setting in the experiments (different values λ, ν, etc.).
- Start several times as there are no guarantees to find the global minimum.
- However, we should try to avoid overfitting:

Ending the attempt at optimization

- Updates of u_{ik} and m_{jk} are usually repeated until the error improves (more than a given threshold).
- Try different setting in the experiments (different values λ, ν, etc.).
- Start several times as there are no guarantees to find the global minimum.
- However, we should try to avoid overfitting:
  - Stop revising elements of $\mathbf{U}$ and $\mathbf{M}$ before the process has converged,

Ending the attempt at optimization

- Updates of u_{ik} and m_{jk} are usually repeated until the error improves (more than a given threshold).
- Try different setting in the experiments (different values λ, ν, etc.).
- Start several times as there are no guarantees to find the global minimum.
- However, we should try to avoid overfitting:
  - Stop revising elements of $\mathbf{U}$ and $\mathbf{M}$ before the process has converged,
  - Average over different runs of the algorithm,

Ending the attempt at optimization

- Updates of u_{ik} and m_{jk} are usually repeated until the error improves (more than a given threshold).
- Try different setting in the experiments (different values λ, ν, etc.).
- Start several times as there are no guarantees to find the global minimum.
- However, we should try to avoid overfitting:
  - Stop revising elements of $\mathbf{U}$ and $\mathbf{M}$ before the process has converged,
  - Average over different runs of the algorithm,
  - Smaller steps in optimization algorithms,

Ending the attempt at optimization

- Updates of u_{ik} and m_{jk} are usually repeated until the error improves (more than a given threshold).
- Try different setting in the experiments (different values λ, ν, etc.).
- Start several times as there are no guarantees to find the global minimum.
- However, we should try to avoid overfitting:
  - Stop revising elements of $\mathbf{U}$ and $\mathbf{M}$ before the process has converged,
  - Average over different runs of the algorithm,
  - Smaller steps in optimization algorithms,
  - Regularization.

Matrix factorization extensions

- Different loss functions.

Matrix factorization extensions

- Different loss functions.
- Visualization of features.

Matrix factorization extensions

- Different loss functions.
- Visualization of features.
- The use of regular features.

Matrix factorization extensions

- Different loss functions.
- Visualization of features.
- The use of regular features.
- A quite general learning framework . . .

Beyond matrix factorization

- Relational learning,
- Tensor factorization.

Beyond matrix factorization

		$t_1(y)$	4	5	$\cdots$	7	8	6
		$t_2(y)$	10	14	$\cdots$	9	21	12
$u_1(x)$	$u_2(x)$	x/y	y_1	y_2	$\cdots$	y_m	y_{m+1}	y_{m+2}
1	1	x_1	10	?	$\cdots$	1	?	?
3	5	x_2	?	0.1	$\cdots$	0		?
7	0	x_3	?	?	$\cdots$	1	?	?
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
3	1	x_n	-5	0.9	$\cdots$	1	?	?
2	3	x_{n+1}	?	?	$\cdots$	?	?	?
3	1	x_{n+2}	?	?	$\cdots$	?	?	?

Outline

Summary

- Recommender systems:
  - Content-based systems,
  - Collaborative filtering.
- Collaborative filtering
  - Similarity-based,
  - Clustering,
  - Matrix factorization.
- Matrix factorization:
  - Matrix factorization with more features,
  - Regularization,
  - Stochastic gradient optimization.

Bibliography

- A. Rajaraman and J. D. Ullman. *Mining of Massive Datasets*. Cambridge University Press, 2011 `http://www.mmds.org`