Vanilla matrix factorization

Matrix factorization techniques for recommender systems

By Niklas Rowohl

Probabilistic matrix factorization

By Mohamed Emarah

Non-negative Matrix Factorization with Sparseness Constraints

By Jan Forkel

Matrix factorization techniques for recommender systems

by Niklas Rowohl

Matrix factorization techniques for recommender systems

Overview of the topics

- the benefits of using recommender systems
- content filtering vs collaborative filtering
- basic matrix factorization
- learning algorithms
- refining the minimization problem
- outlook on the netflix prize

Why do users need recommender systems?

- users of electronic retailers or content providers face a large number of choices
  →matching users with most appropriate items
  raises user satisfaction and loyalty
- recommender systems analyze patterns of user interests to provide <u>personalized</u> recommendations

Content filtering

- both users and products get a profile
- user profiles might include age, demographic or answers given to a suitable questionnaire
- product profiles include attributes describing its nature (e.g. movies have genre, actors, release date)

  → based on these profiles, user/product is matched

Collaborative filtering

- relies on past user behavior (e.g. ratings, purchase history)
- no explicit profiles are necessary
- instead relationships between users and products are used to identify new user-product associations

Content filtering vs collaborative filtering (1/2)

Aspects in which content filtering is superior:

- content filtering does not suffer from the *cold start* problem as it collects data about product and user beforehand to generate profiles

- collaborative filtering has to address its *cold start* problem, because new users/products start without any information about them

Content filtering vs collaborative filtering (2/2)

Aspects in which collaborative filtering is superior:

- it is domain free
- can address data aspects that are difficult to profile, because they are too elusive

→ generally more accurate than content filtering

- easier model to expand with biases/implicit data, etc.
- content filtering needs external information about product/user which might be difficult to obtain

Collaborative filtering: Neighborhood method

- centered around relationships between users or items
- user/user relationship: highly rated items by users with similar taste are suggested for one another
- item/item relationship: similar items to the items a user liked are suggested to the user

Collaborative filtering: Latent factor models

- users and items are characterized by some (20 -100) characteristics based on rating patterns
  → each item and each user gets a vector which includes these characteristics (user vectors have preferences for certain characteristics of an item, item vectors the characteristics themself)
- the user vector is denoted as p_u, the item vector as q_i

Basic matrix factorization

- Consider a item/user matrix R that contains the ratings user u gave to item i

→ obviously does not every user rate every product

→ we try to estimate the ratings not given in R by using user and item vectors p_u and q_i

$$R \approx Q^T \cdot P = \hat{R}$$

Where Q contains the item vectors q_i and P the user vectors p_u.

Computing the item/user matrix with estimated ratings

Result: Item/user matrix with estimated ratings

input: N user vectors p_u, M item vectors q_i

```
for u=1 to u=N;u++ do
    for i=1 to i=M;i++ do
        |
        |
    end
end
Fill the item/user matrix R̂ with the estimated ratings r̂_ui
return R̂
```

$$\hat{r}_{ui} = q_i^T \cdot p_u$$

Algorithm 1: How to compute the estimated ratings

How do we estimate the entries of the user and item vector?

- as we have seen, computing the item/user matrix with estimated ratings is rather simple once we know the user and item vectors p_u and q_i
- we need to learn these vectors by analyzing the observed data
  → minimize the error between the estimated ratings and the known ratings

Minimizing the error in the training set

- K is the set of explicitly given ratings r_{ui} of user u to item i
- λ is a constant and controls the extent of regularization to avoid overfitting

→ the goal is to minimize the error in the training set K and not to overfit the training data in doing so

$$min \sum_{(u,i)\in K} (r_{ui} - q_i^T \cdot p_u)^2 + \lambda(\|q_i\|^2 + \|p_u\|^2)$$

The problem of overfitting

- Overfitting occurs when a model based on training data describes the random error or „background noise“ of the training data

→ doing so makes the model fail in predicting data outside of the training data

Learning algorithm

- learning algorithms are algorithms designed to solve the minimization problem
- there are multiple approaches to finding the minimum, in this presentation two approaches are presented and compared

Stochastic gradient descent

- basic idea: start at a random point → compute the gradient on that point → shift to a new point in the opposite direction of the gradient → repeat procedure at the new point until the new point is „good enough“
- how far can we shift the new point and what is the gradient in this case?

Stochastic gradient descent algorithm (1/2)

Result: Updated feature in the item vector, updated preference in the user vector

input: one user vector p_u, one item vector q_i, feature k that is trained, real rating r_{ui} from user u for item i, learning rate lrate

1 Error function: $e_{ui} = r_{ui} - q_i^T \cdot p_u$

2 Squared Error with regularization is therefore: $e_{ui}^2 = (r_{ui} - q_i^T \cdot p_u)^2$

3 Partial derivatives with regard to feature k:

$$\frac{\partial e_{ui}^2}{\partial p_{uk}} = 2 \cdot (r_{ui} - q_i^T \cdot p_u) \cdot (-q_{ik}) = -2 \cdot e_{ui} \cdot q_{ik}$$

$$\frac{\partial e_{ui}^2}{\partial q_{ik}} = 2 \cdot (r_{ui} - q_i^T \cdot p_u) \cdot (-p_{uk}) = -2 \cdot e_{ui} \cdot p_{uk}$$

4 Updating the item and user vector:

$$new\_p_{uk} = p_{uk} + 2 \cdot lrate \cdot e_{ui} \cdot q_{ik}$$

$$new\_q_{ik} = q_{ik} + 2 \cdot lrate \cdot e_{ui} \cdot p_{uk}$$

5 **return** p_u and q_i with updated k-th entry new_p_{uk} and new_q_{ik}.

Algorithm 1: How to train a feature k using stochastic gradient descent

Stochastic gradient descent algorithm (2/2)

- lrate stands for the learning rate (also called step size) of the algorithm and makes sure that the algorithm converges
  → can be chosen rather arbitrarily or with the Armijo algorithm
- in one iteration we calculate the error and then update the k-th entry in our vectors by going into the opposite direction of the partial derivates of the error function

Alternating least squares (ALS)

- because the minimization problem relies on two vectors (user vector and item vector) to be trained, we cant simply do a least squares approach as usual

  → however by fixing one of the vectors and only optimizing the other, we can fall back into the standard least squares approach

- then we alternate between which vector is fixed and which gets optimized

Pseudo code for ALS

Result: Optimized user and item vector

input: N user vectors p_u, M item vectors q_i, set K with the training data r_{ui}

while $\sum_K (r_{ui} - q_i^T \cdot p_u)^2 + \lambda(\|p_u\|^2 + \|q_i\|^2)$ *is not small enough* **do**

  Fix q_i and solve the least squares regression with regards to p_u.

  Update the p_us accordingly.

  Fix p_u and solve the least squares regression with regards to q_i.

  Update the q_is accordingly.

end

return $q_i, p_u \; i = 1, ..., M; u = 1, ..., N$

Algorithm 1: How to solve alternating least squares

Stochastic gradient descent vs ALS

- while in general stochastic gradient descent runs faster, there are at least two circumstances in which ALS is superior

1. parallelization: ALS can be parallelized, because each user and each item vector can be computed independently of other user/item factors
2. non sparse rating matrices: stochastic gradient descent loops over all of the training data. If there is a lot of training data, this can make the algorithm quite slow, while ALS can handle this case

Refining the model

- the formulas used to compute the estimated rating and to train the model can be expanded

→ Adding biases: Users are more or less critical in their ratings, products are better/worse

→ Implicit data: i.e. Purchase history can be used to gather further information about a user

→ Temporal dynamics: user behaviour/bias or item bias might change with time

→ Confidence in the training data: Different levels of certainty can be applied to ratings

Netflix Prize competition

- in 2006 Netflix released a training set of 100 million ratings made by 500,000 users on more than 17,000 movies
  → the item/user matrix has about 8.5 billion entries, but only 100 million of them are filled with a rating
- Netflix had a training set of about 3 million ratings on which they calculated the root mean squared error
  → the first team to beat Netflix' own recommender system by 10% on the test set wins 1 million $

How to win the Netflix Prize competition

- using the methods aforementioned and factorization models Yehuda Koren's team was the frontrunner in 2007 and 2008 and finally won in 2009
  → factorizing the user-movie matrix given by Netflix allowed the team to gain the most descriptive characteristics-preference dependencies

Conclusion

- Content filtering does not allow a look at user behaviour and only relies on profiles
- Collaborative filtering focuses on user-user, item-item (neighborhood method) or user-item (latent factors) interdependencies
- Latent factors lead directly to matrix factorization which allows us to fill in the sparse rating matrix with estimated ratings
- The item/user vectors are estimated by a learning algorithm
- The latent factor model is easy to expand

References

Koren, Yehuda, Robert Bell, and Chris Volinsky. "Matrix factorization techniques for recommender systems." Computer 42.8 (2009): 30-37.

S. Funk, “Netflix Update: Try This at Home,” (1.11.16); http://sifter.org/~simon/journal/20061211.html.

https://en.wikipedia.org/wiki/Stochastic_gradient_descent (1.11.16)

https://en.wikipedia.org/wiki/Collaborative_filtering (1.11.16)

Gareth James, Daniela Witten, Trevor Hastie, R. Tibshirani (2013): An Introduction to Statistical Learning with Applications in R, Springer.

Back up slides

Every slide after this one is not part of the presentation, but may offer some insight to a topic only briefly discussed in the presentation.

Neighborhood method: User/user relationship example (1/2)

User/movie matrix filled with ratings from 1 to 5 (1=not liked,5=liked).
Ratings from Matthias are only known for „Finding Nemo“.
What other movies could be predicted to be liked by Matthias?

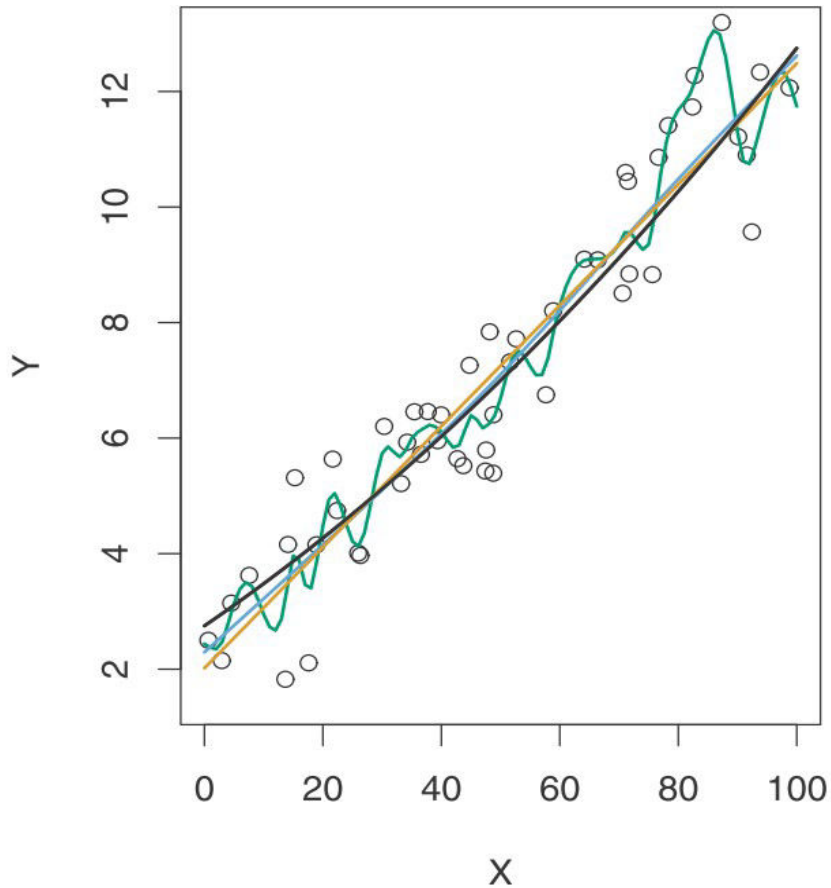
	Finding Nemo	Spiderman	Star Wars	The Hobbit
Jannis	5	2	3	4
Verena	2	3	5	1
Miriam	4	2	1	5
Matthias	5	?	?	?

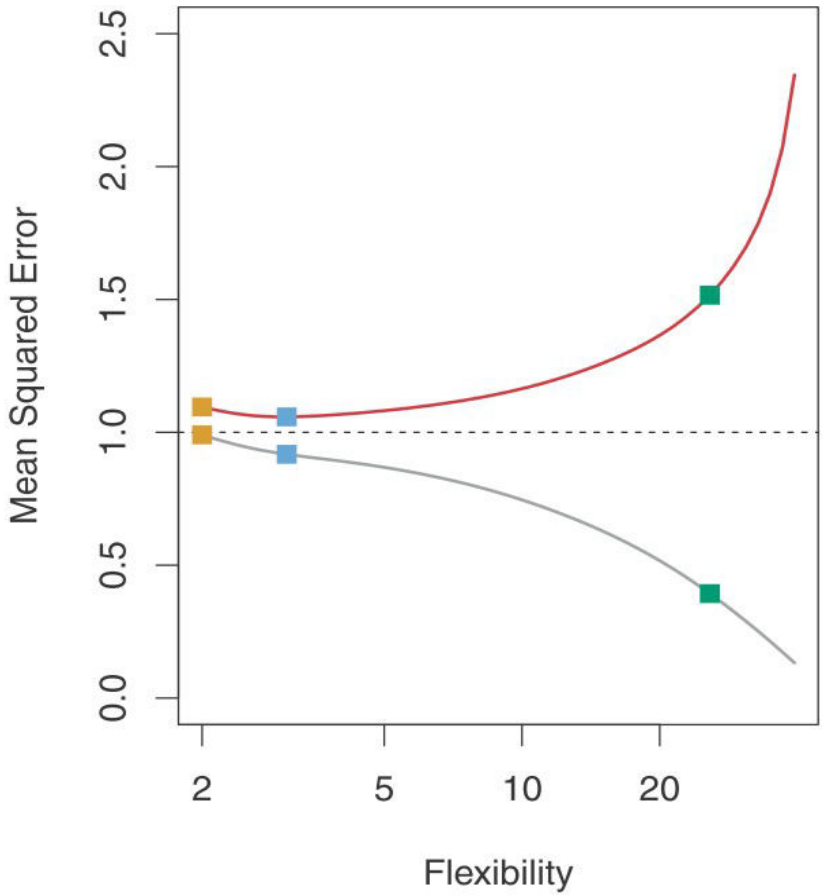
Neighborhood method: User/user relationship example (2/2)

Verena and Matthias' rating doesn't match, so we can ignore her. Jannis, Miriam and Matthias however do match. That is why we can assume that Matthias will like the movies Jannis and Miriam liked (in this example: Matthias will probably like „The Hobbit“).

	Finding Nemo	Spiderman	Star Wars	The Hobbit
Jannis	5	2	3	4
Verena	2	3	5	1
Miriam	4	2	1	5
Matthias	5	?	?	?

Example for overfitting





The left graph shows training data being estimated by different estimation methods

While the green line has the lowest mean squared error on this training data (right-hand graph, grey line), it doesnt seem to correctly reflect the „true“ function as it gets influenced too much by random error made in the training set.

Therefore the mean squared error of the green line in the test data is actually bigger than the error made by the other estimations

Stochastic gradient descent with regularization

Result: Updated feature in the item vector, updated preference in the user vector

input: one user vector p_u, one item vector q_i, feature k that is trained, real rating r_{ui} from user u for item i, learning rate lrate, regularization constant λ

1 Error function: $e_{ui} = r_{ui} - q_i^T \cdot p_u$

2 Squared Error with regularization is therefore:

$$e_{ui}^2 = (r_{ui} - q_i^T \cdot p_u)^2 + \lambda(\|p_u\|^2 + \|q_i\|^2)$$

3 Partial derivatives with regard to feature k:

$$\frac{\partial e_{ui}^2}{\partial p_{uk}} = 2 \cdot (r_{ui} - q_i^T \cdot p_u) \cdot (-q_{ik}) + 2\lambda \cdot p_{uk} = -2 \cdot e_{ui} \cdot q_{ik} + 2\lambda \cdot p_{uk}$$

$$\frac{\partial e_{ui}^2}{\partial q_{ik}} = 2 \cdot (r_{ui} - q_i^T \cdot p_u) \cdot (-p_{uk}) + 2\lambda \cdot q_{ik} = -2 \cdot e_{ui} \cdot p_{uk} + 2\lambda \cdot q_{ik}$$

4 Updating the item and user vector:

$$new\_p_{uk} = p_{uk} + 2 \cdot lrate \cdot (e_{ui} \cdot q_{ik} - \lambda \cdot p_{uk})$$

$$new\_q_{ik} = q_{ik} + 2 \cdot lrate \cdot (e_{ui} \cdot p_{uk} - \lambda \cdot q_{ik})$$

5 **return** p_u and q_i with updated k-th entry new_p_{uk} and new_q_{ik}.

Algorithm 1: How to train a feature k using stochastic gradient descent and considering regularization

Adding biases

- expanding the model by 3 types of biases:

1. user bias: b_u
2. item bias: b_i
3. overall average rating: μ

$$\hat{r}_{ui} = \mu + b_i + b_u + q_i^T \cdot p_u$$

$$\min_{p*,q*,b*} \sum_{(u,i)\in K} (r_{ui} - \mu - b_u - b_i - q_i^T \cdot p_u)^2 + \lambda(\|p_u\|^2 + \|q_i\|^2 + b_u^2 + b_i^2)$$

Additional input sources (1/2)

- additional information can be gathered through implicit feedback or known user attributes (age, gender, etc) and refine our model
- helps with the cold start problem and with users not willing to give explicit ratings

Additional input sources (2/2)

(Normalized) Implicit feedback:

$$|N(u)|^{-0.5} \sum_{i \in N(u)} x_i$$

Known user information:

$$\sum_{a \in A(u)} y_a$$

$$\Rightarrow \hat{r}_{ui} = \mu + b_i + b_u + q_i^T \cdot (p_u + |N(u)|^{-0.5} \sum_{i \in N(u)} x_i + \sum_{a \in A(u)} y_a)$$

$N(u)$ is the set of items user u showed implicit preference for, x_i is the vector associated with item i, $A(u)$ is the set of attributes that correspond with user u, y_a corresponds corresponds to each attribute to describe a user/item association.

Temporal dynamics

- specifically the user bias, item bias and the user preference can change over time
  → modeling these parameters as functions over time, change our model from a static to a dynamic one

$$\hat{r}_{ui}(t) = \mu + b_i(t) + b_u(t) + q_i^T \cdot p_u(t)$$

Inputs with varying confidence levels

- massive advertising for a certain item might influence ratings not correctly reflecting its longterm ratings
  → how sure can one be in given ratings?
- our model can reflect the confidence in ratings by giving user/item pairs weights (c_{ui})

$$\min_{p*,q*,b*} \sum_{(u,i)\in K} c_{ui}(r_{ui} - \mu - b_u - b_i - q_i^T \cdot p_u)^2 + \lambda(\|p_u\|^2 + \|q_i\|^2 + b_u^2 + b_i^2)$$

Probabilistic Matrix Factorization

Collaborative-filtering technique for Recommender systems

By: Mohamed Emara

Contents

Introduction

- The PMF model that is basically based on the assumption that users who have rated similar sets of movies are likely to have similar preferences.

- This model scales linearly with the number of observations and, more importantly, performs well on the large, sparse, and very imbalanced Netflix dataset. It also emphasize the regularization concept which solve all ill-posed problems or prevent any overfitting.

Current techniques

- There are many different approaches and algorithms in collaborative-filtering technique, some of them are based on low-dimensional factor.The idea behind such models is that attitudes or preferences of a user are determined by a small number of unobserved factors.

- A variety of probabilistic factor-based models has been proposed recently, in addition to low-rank approximations based on minimizing the sum-squared distance can be found using Singular Value Decomposition (SVD).

- SVD finds the matrix $\hat{\hat{R}} = U^T V$ of the given rank which minimizes the sum-squared distance to the target matrix R.

Current techniques

Unfortunately all of the mentioned techniques have a lot of disadvantages and drawbacks. i.e.

- Performance
- Scalability
- Sparseness (because of users who have few ratings records)

Except for the matrix-factorization-based ones none of the above mentioned have proved an accurate predictions.

PMF technique

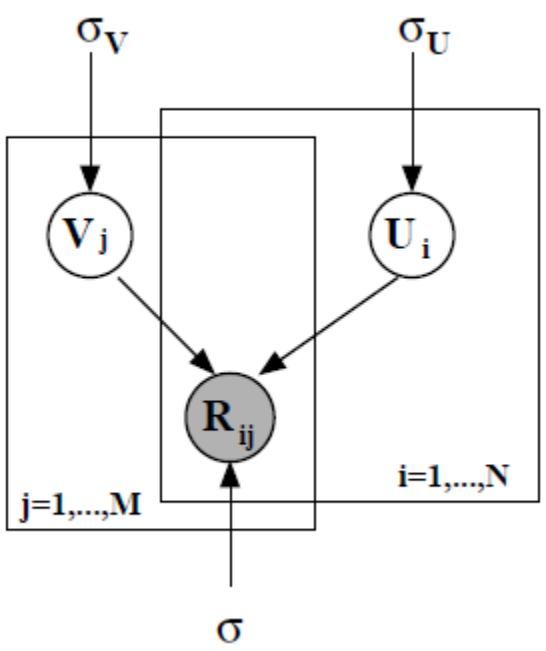
- PMF is a simple probabilistic linear model with Gaussian observation noise.

- Given the feature vectors for the user and the movie, the distribution of the corresponding rating is:

$$p(R_{ij}|U_i, V_j, \sigma^2) = \mathcal{N}(R_{ij}|U_i^T V_j, \sigma^2)$$

- The user and movie feature vectors are given zero-mean spherical Gaussian priors:

$$p(U|\sigma_U^2) = \prod_{i=1}^{N} \mathcal{N}(U_i|0, \sigma_U^2 \mathrm{I}), \quad p(V|\sigma_V^2) = \prod_{j=1}^{M} \mathcal{N}(V_j|0, \sigma_V^2 \mathrm{I})$$



PMF technique

- Maximize the log posterior over movie and user features with fixed hyperparameters.
- Equivalent to minimizing the sum-of-squared-errors with quadratic regularization terms:

$$E = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{M} I_{ij}\left(R_{ij} - U_i^T V_j\right)^2 + \frac{\lambda_U}{2}\sum_{i=1}^{N} \| U_i \|_{Fro}^2 + \frac{\lambda_V}{2}\sum_{j=1}^{M} \| V_j \|_{Fro}^2,$$

where $\lambda_U = \sigma^2/\sigma_U^2$, $\lambda_V = \sigma^2/\sigma_V^2$, and $\| \cdot \|_{Fro}^2$ denotes the Frobenius norm.

- Find a local minimum by performing gradient descent in U and V.
- If all ratings were observed, the objective to the SVD objective in the limit of prior variance going to infinity.

PMF technique

- Instead of using a simple linear-Gaussian model, which can make predictions outside of the range of valid rating values, the dot product between user- and movie-specific feature vectors is passed through the logistic function g(x) = 1/(1+exp(−x)), which bounds the range of predictions:

$$p(R|U,V,\sigma^2) = \prod_{i=1}^{N}\prod_{j=1}^{M}\left[\mathcal{N}(R_{ij}|g(U_i^T V_j),\sigma^2)\right]^{I_{ij}}.$$

- We map the ratings 1,...,K to the interval [0,1] using the function t(x) = (x − 1)/(K − 1), so that the range of valid rating values matches the range of predictions our model makes.

PMF with adaptive prior

- Capacity control is essential to making PMF models generalize well.
- The simplest way to control the capacity of a PMF model is by changing the dimensionality of feature vectors.
- The complexity of the model is controlled by the hyperparameters:

$$\ln p(U, V, \sigma^2, \Theta_U, \Theta_V | R) = \ln p(R|U, V, \sigma^2) + \ln p(U|\Theta_U) + \ln p(V|\Theta_V) + \ln p(\Theta_U) + \ln p(\Theta_V) + C,$$

- the noise variance σ2 and the parameters of the priors.
- Find a MAP estimate for the hyperparameters after introducing priors for them.
- Find a point estimate of parameters and hyperparameters by maximizing the log-posterior.

PMF with adaptive prior

- Can use more sophisticated regularization methods than simple penalization of the Frobenius norm of the feature matrices.
- priors with diagonal or full covariance matrices and adjustable means, or even mixture of Gaussians priors.
- Using spherical Gaussian priors for feature vectors leads to the standard PMF with λU and λV chosen automatically.
- selection of the hyperparameter values worked considerably better than the manual approach that used a validation set.

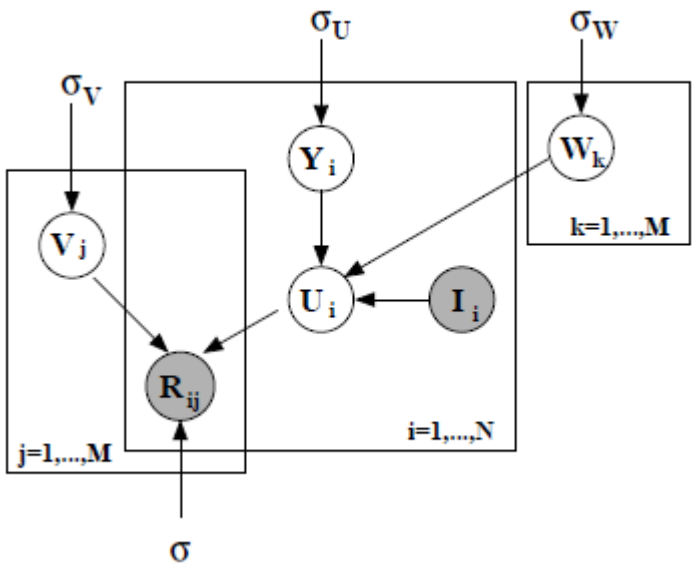
Constraint PMF

- Once a PMF model has been fitted, users with very few ratings will have feature vectors that are close to the prior mean, or the average user, so the predicted ratings for those users will be close to the movie average ratings.
- Two users that have rated similar movies are likely to have preferences more similar than two randomly chosen users.
- We are introducing a way of constraining user-specific feature vectors that has a strong effect on infrequent users.

Constraint PMF

- Let W ∈ RD×M be a latent, so the feature vector for user i is:

$$U_i = Y_i + \frac{\sum_{k=1}^{M} I_{ik} W_k}{\sum_{k=1}^{M} I_{ik}}.$$



where I is the observed indicator matrix with Iij taking on value 1 if user i rated movie j and 0 otherwise.

- This should be performing better on infrequent user.

Constraint PMF

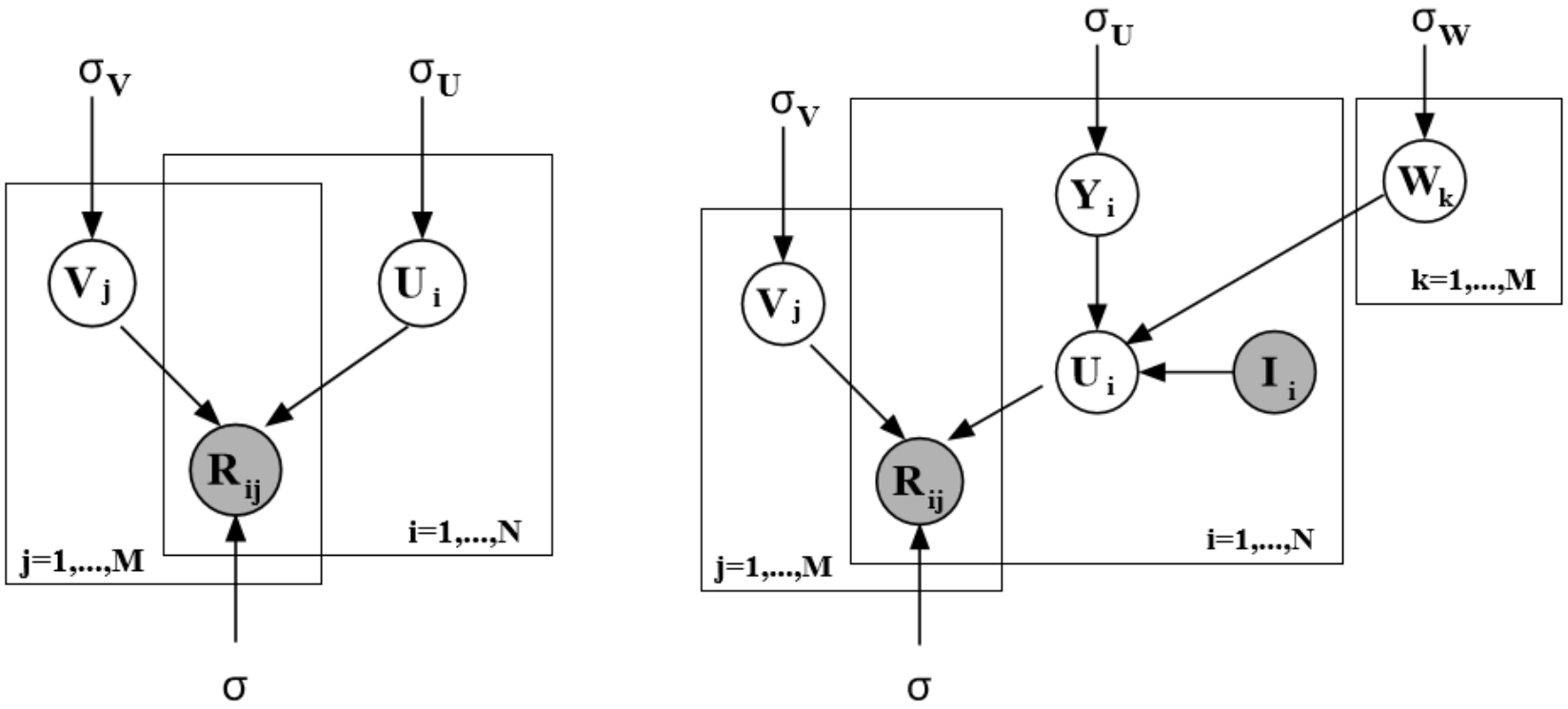
- We now define the conditional distribution over the observed ratings as:

$$p(R|Y,V,W,\sigma^2) = \prod_{i=1}^{N}\prod_{j=1}^{M}\left[\mathcal{N}(R_{ij}|g([Y_i + \frac{\sum_{k=1}^{M} I_{ik}W_k}{\sum_{k=1}^{M} I_{ik}}]^T V_j), \sigma^2)\right]^{I_{ij}}.$$

- We regularize the latent similarity constraint matrix W by placing a zero-mean spherical Gaussian prior on it:

$$p(W|\sigma_W) = \prod_{k=1}^{M}\mathcal{N}(W_k|0,\sigma_W^2\mathbf{I}).$$

Constraint PMF

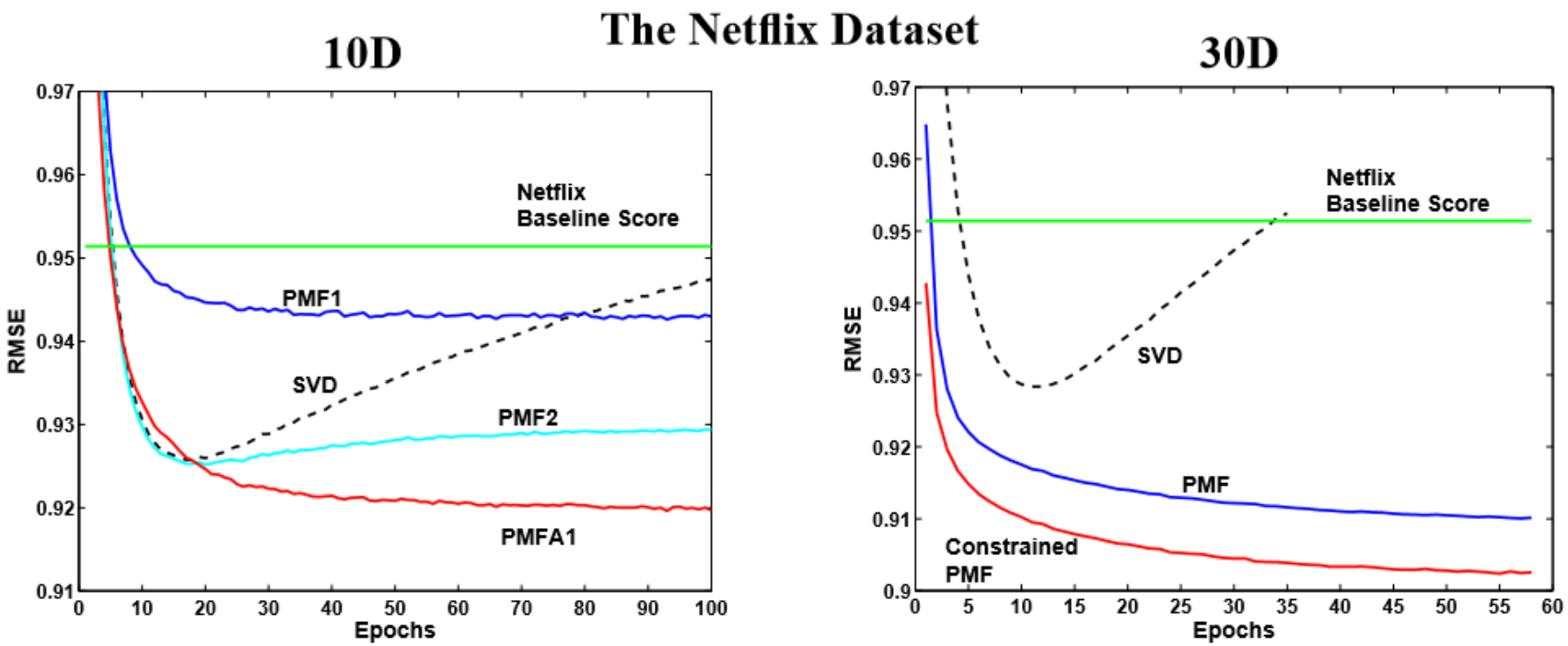


- The left panel shows the graphical model for Probabilistic Matrix Factorization (PMF). The right panel shows the graphical model for constrained PMF.

Experimental result and conclusion

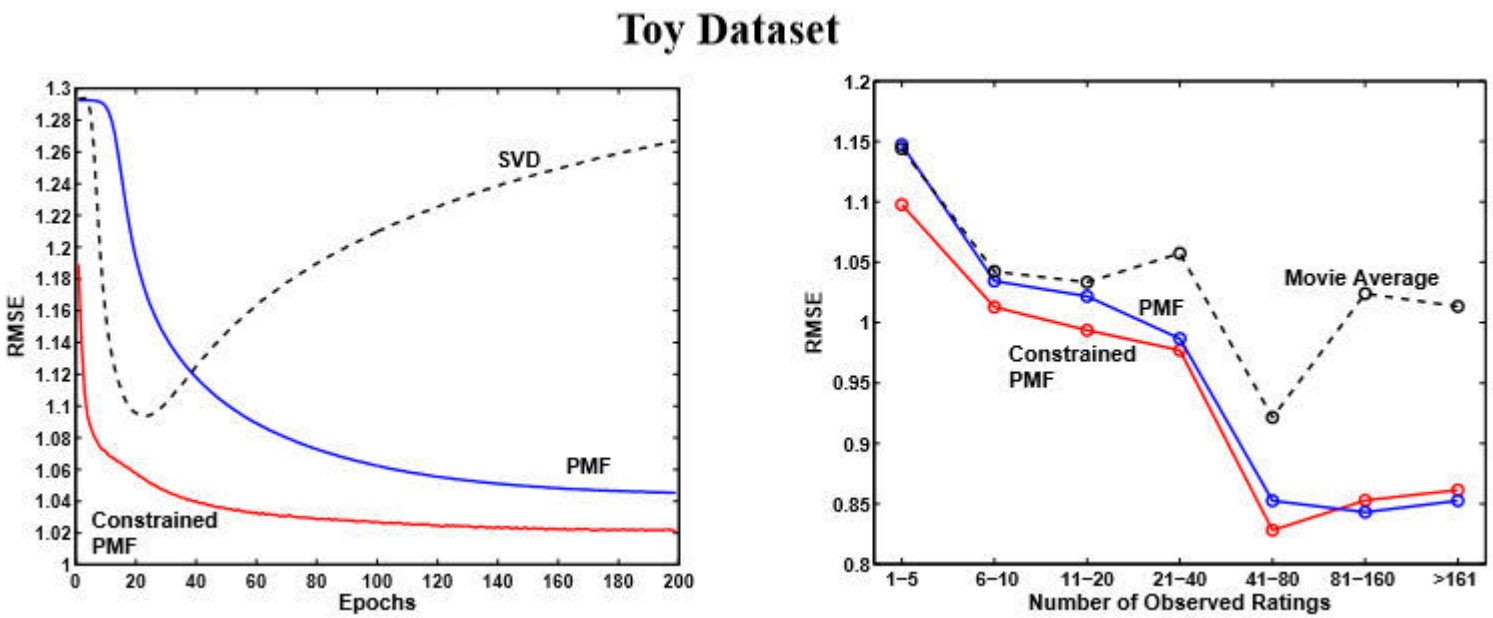
- According to Netflix, the data were collected between October 1998 and December 2005 and represent the distribution of all ratings Netflix obtained during this period. The training dataset consists of 100,480,507 ratings from 480,189 randomly-chosen, anonymous users on 17,770 movie titles. As part of the training data, Netflix also provides validation data, containing 1,408,395 ratings. In addition to the training and validation data, Netflix also provides a test set containing 2,817,131 user/movie pairs with the ratings withheld.

Experimental result and conclusion



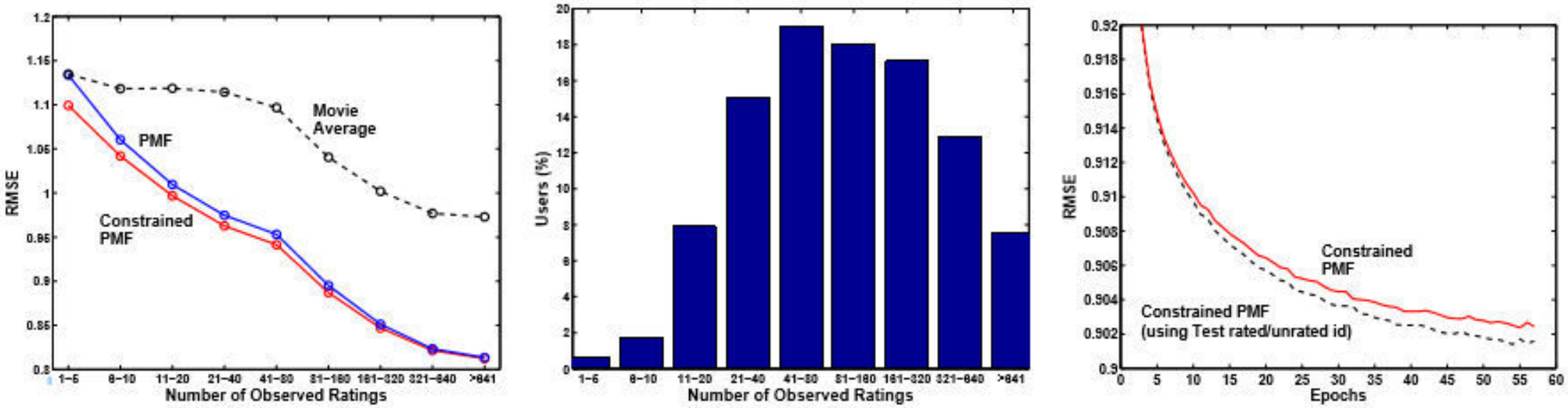
- Left panel: Performance of SVD, PMF and PMF with adaptive priors, using 10D feature vectors, on the full Netflix validation data. Right panel: Performance of SVD, Probabilistic Matrix Factorization (PMF) and constrained PMF, using 30D feature vectors, on the validation data. The y-axis displays RMSE (root mean squared error), and the x-axis shows the number of epochs, or passes, through the entire training dataset.

Experimental result and conclusion



- Left panel: Performance of SVD, Probabilistic Matrix Factorization (PMF) and constrained PMF on the validation data. The y-axis displays RMSE (root mean squared error), and the x-axis shows the number of epochs, or passes, through the entire training dataset. Right panel: Performance of constrained PMF, PMF, and the movie average algorithm that always predicts the average rating of each movie. The users were grouped by the number of observed ratings in the training data.

Experimental result and conclusion



- Left panel: Performance of constrained PMF, PMF, and the movie average algorithm that always predicts the average rating of each movie. The users were grouped by the number of observed rating in the training data, with the x-axis showing those groups, and the y-axis displaying RMSE on the full Netflix validation data for each such group. Middle panel: Distribution of users in the training dataset. Right panel: Performance of constrained PMF and constrained PMF that makes use of an additional rated/unrated information obtained from the test dataset.

Conclusion

- In this paper we presented Probabilistic Matrix Factorization (PMF) and its two derivatives: PMF with a learnable prior and constrained PMF. We also demonstrated that these models can be efficiently trained and successfully applied to a large dataset containing over 100 million movie ratings.

- Efficiency in training PMF models comes from finding only point estimates of model parameters and hyperparameters, instead of inferring the full posterior distribution over them. If we were to take a fully Bayesian approach, we would put hyperpriors over the hyperparameters and resort to MCMC methods [5] to perform inference. While this approach is computationally more expensive, preliminary results strongly suggest that a fully Bayesian treatment of the presented PMF models would lead to a significant increase in predictive accuracy.

References

[1] Delbert Dueck and Brendan Frey. Probabilistic sparse matrix factorization. Technical Report PSI TR 2004-023, Dept. of Computer Science, University of Toronto, 2004.
[2] Thomas Hofmann. Probabilistic latent semantic analysis. In Proceedings of the 15th Conference on Uncertainty in AI, pages 289–296, San Fransisco, California, 1999. Morgan Kaufmann.
[3] Benjamin Marlin. Modeling user rating profiles for collaborative filtering. In Sebastian Thrun, Lawrence K. Saul, and Bernhard Scholkopf, editors, NIPS. MIT Press, 2003.
[4] Benjamin Marlin and Richard S. Zemel. The multiple multiplicative factor model for collaborative filtering. In Machine Learning, Proceedings of the Twenty-first International Conference (ICML 2004), Banff, Alberta, Canada, July 4-8, 2004. ACM, 2004.
[5] Radford M. Neal. Probabilistic inference using Markov chain Monte Carlo methods. Technical Report CRG-TR-93-1, Department of Computer Science, University of Toronto, September 1993.
[6] S. J. Nowlan and G. E. Hinton. Simplifying neural networks by soft weight-sharing. Neural Computation, 4:473–493, 1992.
[7] MATRIX FACTORIZATION METHODS FOR COLLABORATIVE FILTERING Andriy Mnih and Ruslan Salakhutdinov

Non-negative Matrix Factorization with Sparseness Constraints

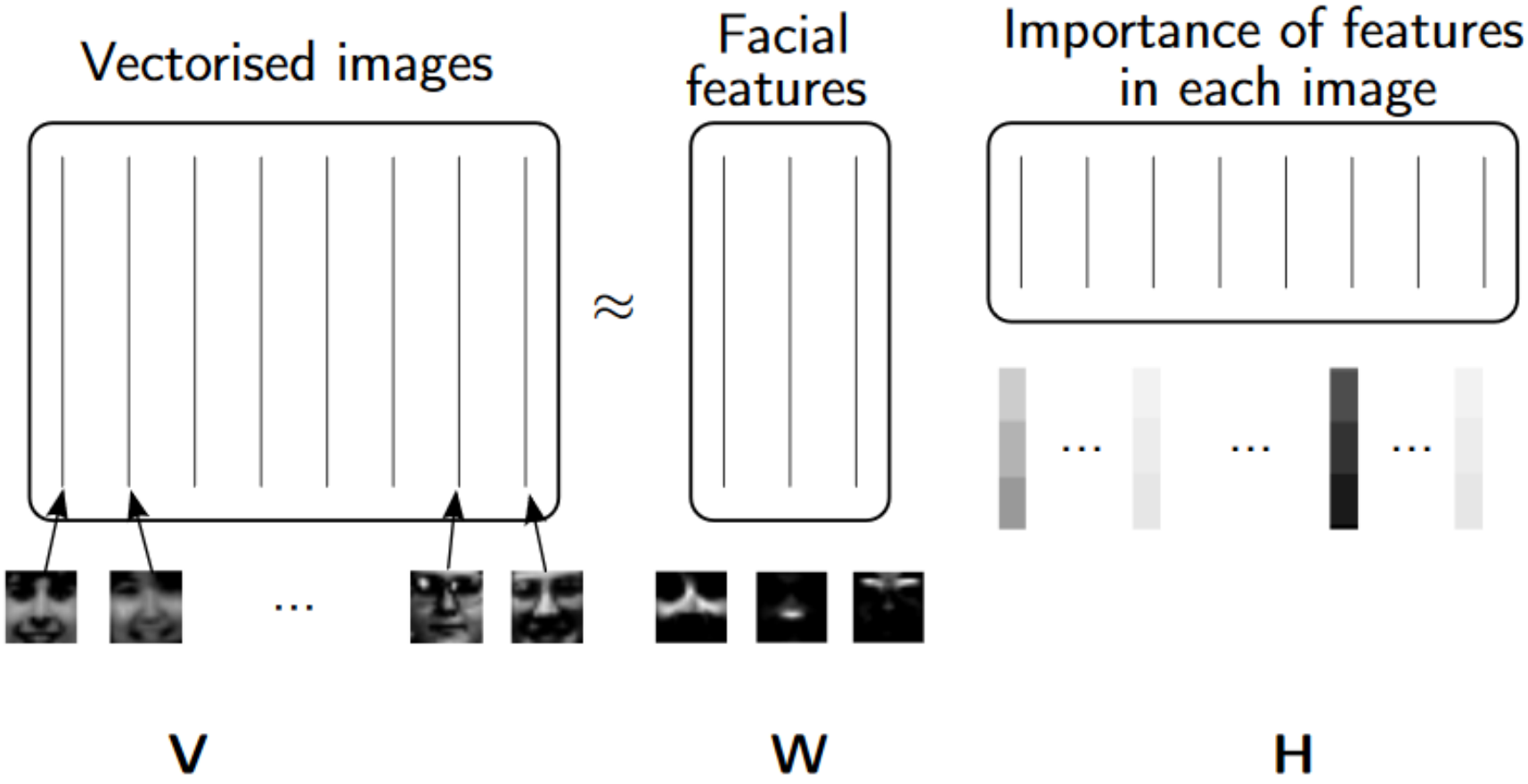
By Jan Forkel

Table of contents

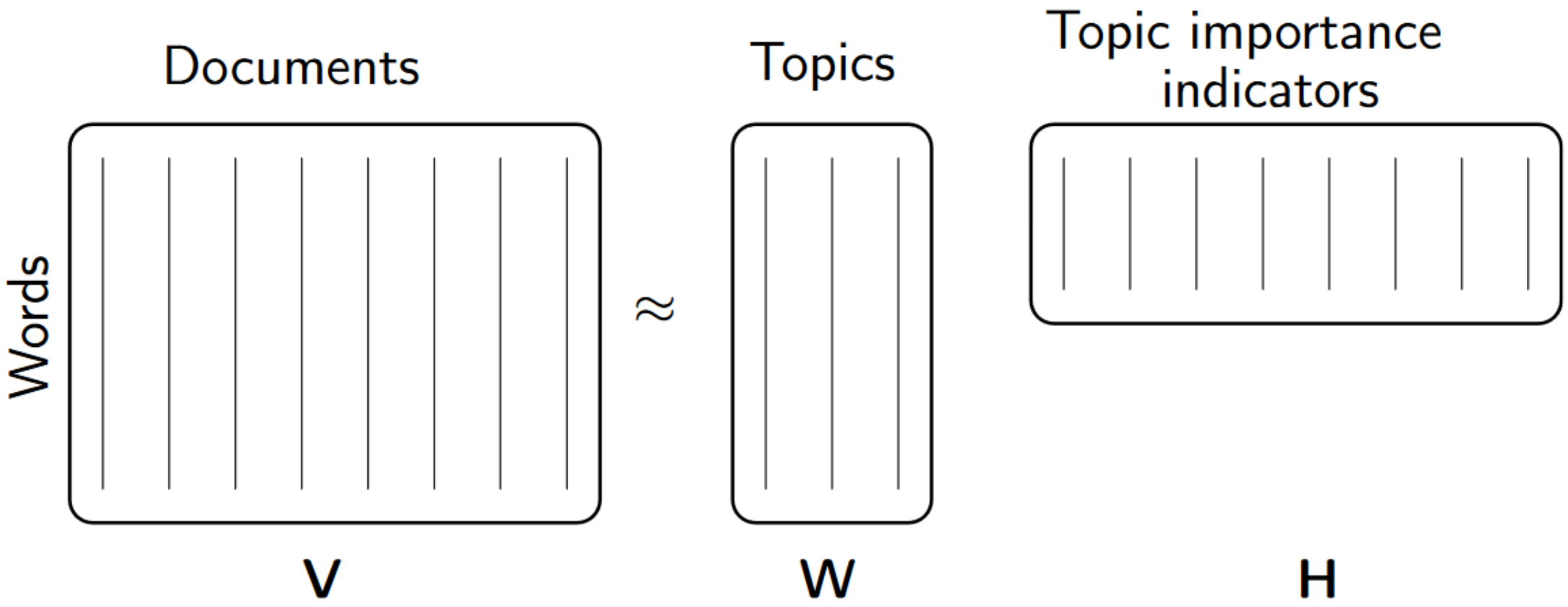
General information

- NMF decomposes a matrix $V \in \mathbb{R}_{+}^{NxT}$ into 2 matrices $W \in \mathbb{R}_{+}^{NxM}$ and $H \in \mathbb{R}_{+}^{MxT}$. All three are positive matrices.
- $V \xrightarrow{NMF} V \approx WH$
- Matrix W contains „basis-vectors“ (features).
- Matrix H contains weightings (importance of basis)
- The non negativity property produces sparse matrices
- Used in computer vision, document clustering, audio signal processing and recommender systems

Example 1


Vectorised images
Facial features
Importance of features in each image
≈
...
...
...
...
V
W
H

Example 2



NMF without sparseness constraints

- NMF is used for parts-based approximative linear representations of non-negative data
- It is called „parts-based” because the basis-vectors represent features of the initial data. Can be seen as „building-blocks“.
- Thus the information of the initial matrix is split into its different features
- Due to sparseness the resulting decomposed Matrixes are easy to interpret and further computational methods are easy to apply
- Sometimes more sparseness is required to improve the solution

Which matrices should be chosen?

- W and H must be chosen, such that the reconstruction error $WH = V$ is minimal.
- Often the squared error (euclidean distance) is used
- $E(W,H) = \|V - WH\|^2 = \sum_{i,j}\left(V_{i,j} - (WH)_{i,j}\right)^2$
- Algorithms for minimization:
  - Gradient algorithm
  - Multiplicative algorithm (simpler implementation, also good performance)

Sparseness

- Sparseness means, that only few units of a large set are effectively used to present a data-vector. Most are zero or close to zero.
- There exist many different sparseness measures.
- Normalized scale: sparseness $\in [0,1]$
- Example: sparseness(x)=$\frac{\sqrt{n}-\sum |x_i| / \sqrt{\sum x_i^2}}{\sqrt{n}-1}$, n=dim(x)
- Measurement for amount of “energy“ which is packed in a few components: $\mathbb{R}^n \to \mathbb{R}$

Definition: NMD with sparseness constraints

Given a non-negative data-matrix $V \in \mathbb{R}_+^{NxT}$ matrixes $W \in \mathbb{R}_+^{NxM}, H \in \mathbb{R}_+^{MxT}$ must be found, such that $\mathrm{E}(W,H) = \|V - WH\|^2$ is minimized under optimal constraints

- Sparseness(w_i)=S_w , $\forall\ i \in \mathbb{N}$
- Sparseness(h_i)=S_h , $\forall\ i \in \mathbb{N}$

➢ h_i and w_i are not constrained ⇒ free to fix any norm of eighter one

➢ W or H sparse? Depends on specific application/experiment

Pseudo code(1/2)

1. Initialize W and H to random positive matrices
2. If sparseness constraints on W apply, then project each column of W to be non-negative, have unit L_2 norm, but L_1 norm set to achieve desired sparseness
3. If sparseness constraints on H apply, then project each row of H to be non-negative, have unit L_2-norm, and L_1-norm set to achieve desired sparseness

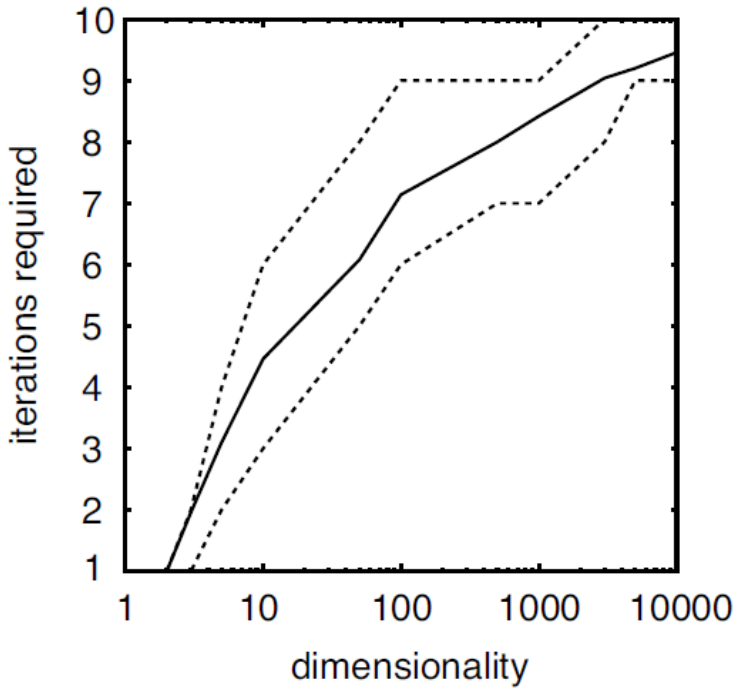
Pseudo code(2/2)

4. Iterate:
   i. If sparseness constraints on W apply,
      - Set $W := W - \mu_W (WH - V) H^T$
      - Project every column of W to be non-negative, have unchanged L_2-norm, but L_1 norm set to achieve desired degree of sparseness
   ii. Else take standard multiplicative step $W \coloneqq W \otimes (W^T V) \oslash (WHH^T)$
   iii. If sparseness constraints on H apply,
      - Set $H \coloneqq H - \mu_H W^T (WH - V)$
      - Project each row of H to be non-negative, have unit L_2-norm, and L_1 norm set to achieve desired degree of sparseness
   iv. Else take standard multiplicative step $H \coloneqq H \otimes (W^T V) \oslash (W^T WH)$

Algorithm to find the closest non-negative vector(in euclidean sense) with given $\boldsymbol{L_1}$ and $\boldsymbol{L_2}$ norm

1. Set $s_i \coloneqq x_i + \frac{L_1 - \sum x_i}{\dim(x)}, \forall i$
2. Set $Z \coloneqq \{\,\}$
3. Iterate
   i. Set $m_i \coloneqq \begin{cases} \frac{L_1}{\dim(x) - size(Z)}, if\ i \notin Z \\ 0 \qquad\qquad\qquad , if\ i \in Z \end{cases}$
   ii. Set $s \coloneqq m + \alpha(s - m)$, where $\alpha \geq 0$ is selected such that the resulting s satisfies the L_2 norm constraint. This requires solving a quadratic equation.
   iii. If all components of s are non-negative, return s, end
   iv. else
      a) Set $Z \coloneqq Z \cup \{i, s_i < 0\}$
      b) Set $s_i \coloneqq 0, \forall i \in Z$
      c) Calculate $c \coloneqq \frac{\sum s_i - L_1}{\dim(x) - size(Z)}$
      d) Set $s_i \coloneqq s_i - c, \forall i \notin Z$
      e) Restart by setting m_i in i.)

Iterations of algorithms depending on dimension

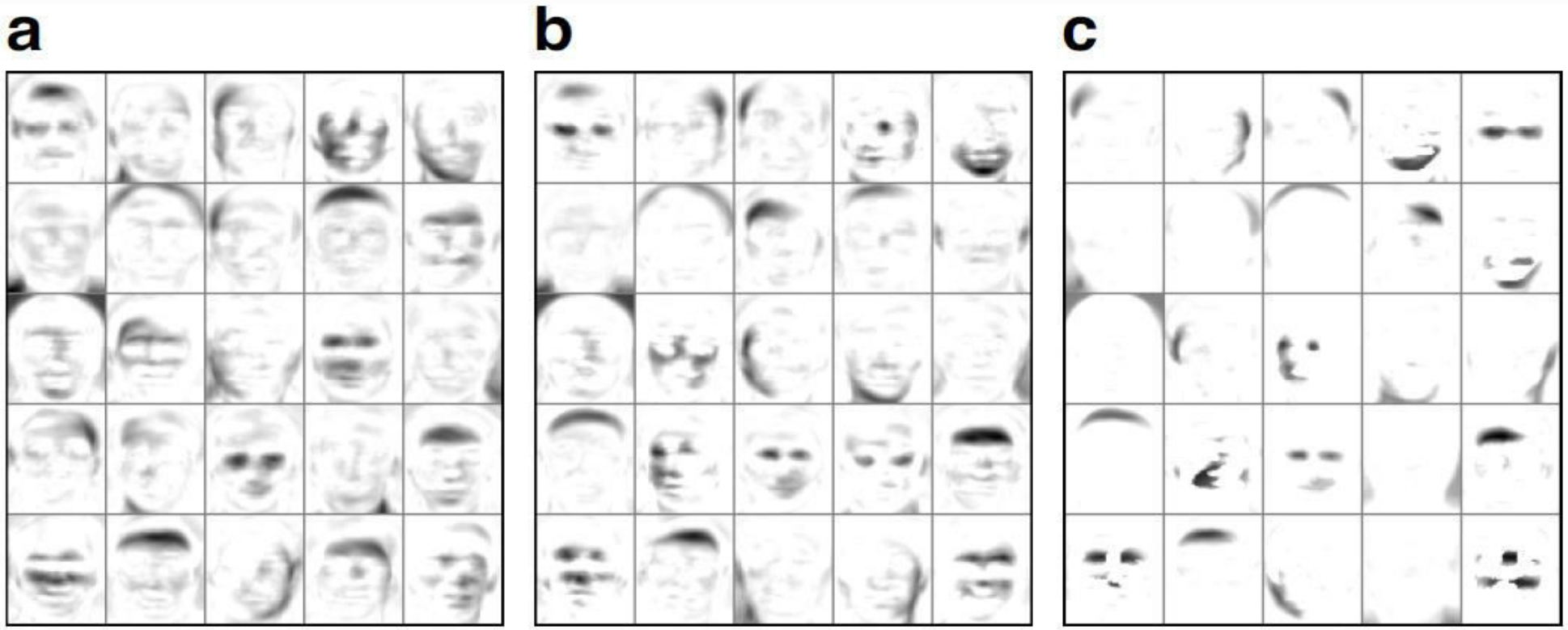


Amount of iterations such that the projection algorithm converges. The solid line shows the average, the dotted lines are the maximum and minimum amount of iterations. (Worst scenario: desired sparseness 0.9, initial sparseness 0.1)

Comparing NMF and NMF with constraints

- NMF with constraints is similar to the NMF
- Both create a positive sparse decomposition of an initial matrix
- In the one with constraints the sparseness can be chosen
- This offers the chance to experiment with different levels of sparseness to gain a better solution depending on what the matrixes are needed for
- In many experiments the NMF with constraints produced a better solution

Using NMF on ORL image database



Features learned from ORL face image database using NMF with sparseness constraints. When increasing the sparseness of the basis images, the representation switches from a global one to a local one. Sparseness levels were set to (a) 0.5, (b) 0.6, (c) 0.75

References

- Patrik O. Hoyer; Non-negative Matrix Factorization with Sparseness Constraints; Journal of Machine Learning Research 5; 2004
- Daniel D. Lee, H. Sebastian Seung; Algorithms for Non-negative Matrix Factorization
- https://en.wikipedia.org/wiki/Non-negative_matrix_factorization; looked up at 31.10.2016, 3pm
- http://perso.telecom-paristech.fr/~essid/teach/NMF_tutorial_ICME-2014.pdf

Comparing the different methods

- The MF and PMF have the same goal
- While MF and PMF tried to fill the initially sparse matrix, the NMF was about setting a specific degree of sparseness on the decomposed ones.
- The NMF with constraints changes the values itself by setting a specific degree of sparseness
- The MF and PMF defined sparseness as empty spaces in the matrices, while the NMF defined them as zero or close to zero
- The NMF only used positive values in the matrices. In the MF and PMF there were also negative ones allowed
- The MF and PMF use stochastic gradient descent, while the NMF uses the gradient descent and multiplicative algorithm
- All approaches use the euclidean distance to minimize the error

Back up slides

Every slide after this one is not part of the presentation, but may offer some insight to a topic only briefly discussed in the presentation.

Current research

- Algorithms: calculation of global minima of factors and factor initialization
- Scalability: Methods to factorize huge matrices
- Online: Updating factorization without recomputing the matrixes
- Collective factorization: factorization of multiple interrelated matrices for multiple-view learning
- Cohen and Rothblum 1993 problem: Does a rational matrix always have a NMF of minimal inner dimension whose factors are also rational? No!

Different norms

- $L_1 = \|x\|_1 = \sum_{i=1}^{N} |x_i|$
- $L_2 = \|x\|_2 = \sqrt[2]{\sum_{i=1}^{N} x_i^2}$
- $L_p = \|x\|_p = \sqrt[p]{\sum_{i=1}^{N} |x|_i^p}$
- x is a vector, N is the dimension of x and $p \in \mathbb{N}$

What do data-matrixes include

- Given data: T measurements and N scalar-variables
- $v^t = \sum_{i=1}^{M} w_i h_i^t = W h^t \,, \forall t \in \{1, \dots, T\}$
- v are data-vectors, w_i are basis-vectors and h_i are weightings
- If many measurements exist, it becomes $V \approx WH$
- Thus a factorization of an initial data matrix can be approximated
- Examples:
  1. w_i=movie information-vector, h_i=user-vector (Movie-rating)
  2. w_i=picture information-vector, h_i=intensity-vector (Image-construction)