Matrix Factorization Methods and their applications: Probabilistic & Non-Negative Factorization

Amin Anvari **SDS385- Final Project**

For the purpose of the final project for the course "SDS385: Statistical Models for big data", I am going to investigate problem of factorizing a matrix into the product of two smaller matrices (i.e., Matrix Factorization). Then I will move on to the non-negative matrix factorization, which differs from the ordinary matrix factorization by restricting all values of the data and model to be greater than zero. This allows for "parts-based learning" from data, of which topic modeling is a prime example. I will try to implement and present two standard NMF algorithms for this problem.

After that, I will cover the fundamental matrix factorization technique called principle component analysis (PCA), a very useful dimensionality reduction approach. I will also present and implement some extension to the regular PCA such as probabilistic PCA for image denoising and inpainting and also kernel PCA for nonlinear dimensionality reduction.

1. Introduction

Matrix factorization problem is an unsupervised learning problem. I will start by motivating the matrix factorization problem by first talking about collaborative filtering problem. The motivation for this type of model, is the object recommendation problem. Of course, there is no need to talk about the importance of the problem of matching consumers to products and many companies are interested to do well in this subject.

We can often make these connections using user feedback about subsets of products. for example:

- Netflix asks its users to rate the movies that they watch
- Amazon asks users to rate products and write some reviews about them.
- Yelp asks users to rate businesses, write reviews about them, and also upload pictures
- Youtube asks users to like/dislike a video and write comments about them

Recommendation systems often use this information to help recommend new things to customers that they may like. This is actually a very big complicated problem and there are many different strategies to solve this problem. One general subset of approaches is called **content filtering.**

The way content filtering performs is it uses known information about the products and users to make profiles for each of them and at the end it matches those profiles to each other. This approach is actually

not using any of the users behavioral information and also it requires a lot of information that sometimes are difficult and expensive to collect.

A fundamentally different approach to solve the problem of object recommendation is **collaborative filtering (CF).** What CF does is, it use previous user's inputs/behavior to make future recommendation, ignoring any sort of a priori user or object specific information.

For an example for collaborative filtering, we can look at neighborhood-based approaches. In this kind of methods, CF:

1. Defines some sort of similarity score between each user and other users based on solely how much their overlapping ratings agree, then
2. based on these scores, let others "vote" on what each user would probably like.

Notice that there is no background information used by this method and exactly the same sort of method can be used for any recommendation system. Note also that content filtering approaches and collaborative filtering approaches are not mutually exclusive. Content filtering information can also be built into a CF system to improve it's performance.

in location-based CF approaches, the algorithm tries to get an R^d embedding for users and objects. The way that we learn these embedding is through **Matrix Factorization.**

2. Background information:

Matrix Factorization

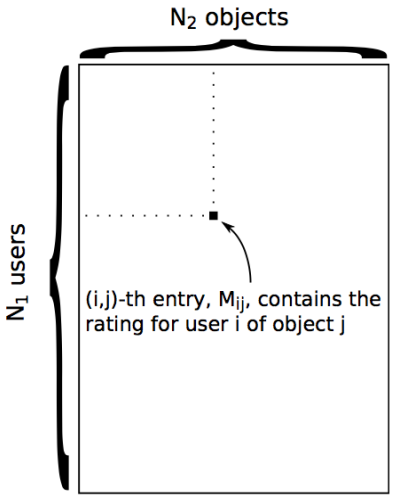
To see how we can use matrix factorization to do object recommendation in a collaborative filtering setting, first let's see how the data structure should look like. We assume that we have a matrix of ratings and by assuming that we have N_1 users and N_2 objects we setup our rating **matrix M** as below:

Notice that in this kind of data structure for the rating matrix, the matrix will have many missing values. The goal of the Matrix Factorization algorithm here is to learn a low-rank factorization of this rating matrix using only the data that we have observed and ignoring all the data that we don't have and then fill in all of these missing values. Finally we can use those predictions, and recommend highly rated objects among the predictions.

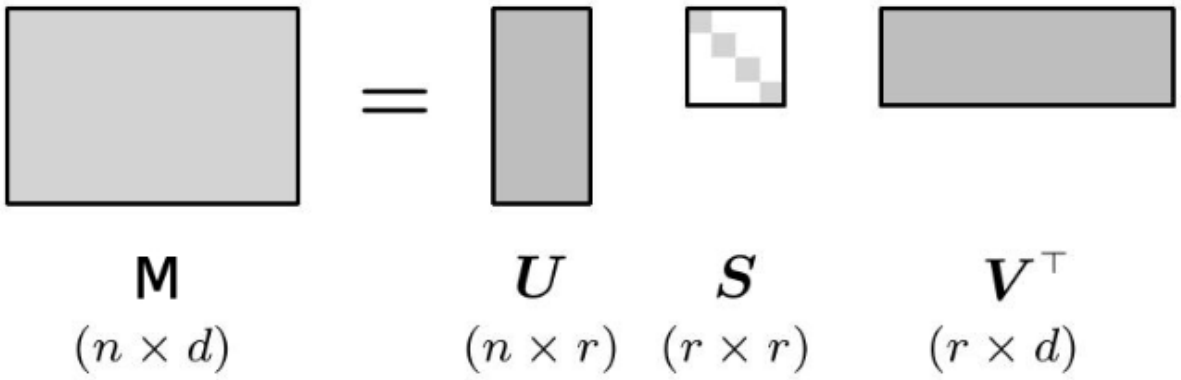
Our goal here is to factorize Matrix M. One approach to this is **Singular Value Decomposition.**

Singular Value Decompisition

Every matrix M can be written as M = USV^T, where $U^TU = I$, $V^TV = I$ and S is diagonal with $S_{ii} \geq 0$.



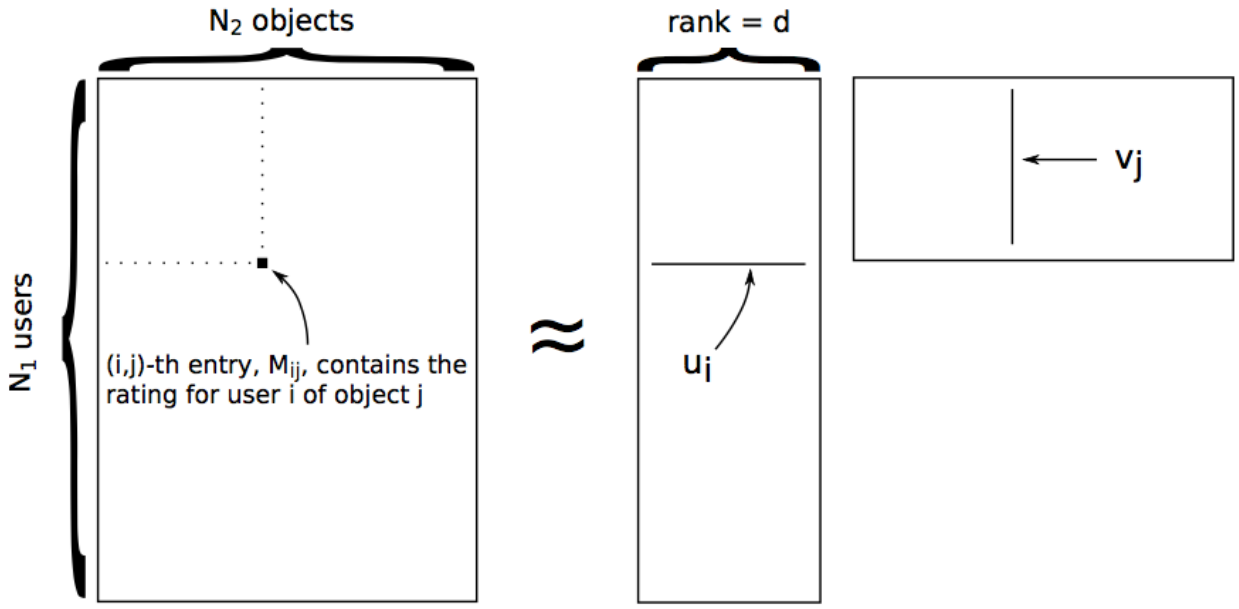
Data Structure for the rating matrix M



We assume r = rank(M). When r is small, M has fewer “degrees of freedom”. A low-rank matrix factorization can restrict the degrees of freedom of how many different types of things we can model in matrix M. We are going to exploit this idea of low-rank matrix factorization in the collaborative filtering with matrix factorization.

We want to define a model for learning a low-rank factorization of M. It should:

1. Account for the fact that most values in M are missing.
2. It should be low-rank, where d << $min\{N_1, N_2\}$
3. Learn a location $u_i \in R^d$ for user i and $v_j \in R^d$ for object j



To discuss more the idea of low-rank matrix and why we want to learn it, we can say that since there are a lot of missing data, by enforcing all the objects and users features to live in this low-rank d-dimensional space, we are in some sense enforcing there to be correlations between similar objects and similar users.

In the missing data problem, one particularly good model for learning low-rank factorization is called **Probabilistic Matrix Factorization.** Following, I am going into more details of this model.

Probabilistic Matrix Factorization

Some pieces of notation to start with:

- Let the set Ω contain the pairs (i , j) that are observed: $\Omega = \{(i,j) : M_{ij}\ is\ measured.\}$
- Let Ω_{ui} be the index set of objects rated by user i.
- Let Ω_{vj} be index set of users who rated object j.

Probabilistic Matrix Factorization assumes a **generative** model for the data. So for example, by assuming the normal prior for our model variables we would have:

For N_1 users and N_2 objects, generate

- **User locations:** $u_i \sim N(0, \gamma^{-1}I)$, i = 1, ..., N_1
- **Object locations:** $v_j \sim N(0, \gamma^{-1}I)$, j = 1, ..., N_2

Given these locations the distribution on the data is

$$M_{ij} \sim N(u_i^T v_j, \sigma^2), \text{ for each } (i,j) \in \Omega.$$

Note that since M_{ij} is a rating, the Gaussian assumption is clearly wrong. However, the Gaussian is a convenient assumption. The algorithm is going to be easy to implement, and the model works well.

Now by using this generative model, we do the inverse problem of inferring the model parameters from the data.

Model Inference

Considering the fact that there are many missing values in the matrix M, the first question that comes to mind is if there is a need for some sort of Expectation Maximization algorithm to learn all the u's and v's?

Let M_o be the part of M that is observed and M_m the missing part, then we have

$$p(M_o|U,V) = \int p(M_o, M_m|U,V)dM_m$$

if $p(M_o|U,V)$ does not present any problems for inference, then there is no need for EM. Similar scenario is correct in our MAP scenario, maximizing $p(M_o,U,V)$.

Here, the best approach would be to write down the joint natural logarithm of the likelihood function and try to take derivatives with respect to u_i and v_j and see if we can solve for them or no. If yes, then there would not be any need for EM algorithm.

The joint likelihood of P(M_o, U, V) can be factorized as follows:

$$p(M_o, U, V) = \underbrace{\Big[\prod_{(i,j)\in\Omega} p(M_{ij}|u_i, v_j)\Big]}_{\text{conditionally independent likelihood}} \times \underbrace{\Big[\prod_{i=1}^{N_1} p(u_i)\Big]\Big[\prod_{j=1}^{N_2} p(v_j)\Big]}_{\text{independent priors}}.$$

Log joint likelihood and Maximum A Posteriori

The MAP solution for U and V is the maximum of the log joint likelihood

$$U_{\text{MAP}}, V_{\text{MAP}} = \arg\max_{U,V} \sum_{(i,j)\in\Omega} \ln p(M_{ij}|u_i, v_j) + \sum_{i=1}^{N_1} \ln p(u_i) + \sum_{j=1}^{N_2} \ln p(v_j)$$

Calling the MAP objective function L, we want to maximize

$$\mathcal{L} = -\sum_{(i,j)\in\Omega} \frac{1}{2\sigma^2}\|M_{ij} - u_i^T v_j\|^2 - \sum_{i=1}^{N_1} \frac{\lambda}{2}\|u_i\|^2 - \sum_{j=1}^{N_2} \frac{\lambda}{2}\|v_j\|^2 + \text{constant}$$

To update each u_i and v_j, we take the derivative of L and set to zero.

$$\nabla_{u_i}\mathcal{L} = \sum_{j\in\Omega_{u_i}} \frac{1}{\sigma^2}(M_{ij} - u_i^T v_j)v_j - \lambda u_i = 0$$

$$\nabla_{v_j}\mathcal{L} = \sum_{i\in\Omega_{v_j}} \frac{1}{\sigma^2}(M_{ij} - v_j^T u_i)u_i - \lambda v_i = 0$$

We can solve for each u_i and v_j individually:

$$u_i = \left(\lambda\sigma^2 I + \sum_{j\in\Omega_{u_i}} v_j v_j^T\right)^{-1}\left(\sum_{j\in\Omega_{u_i}} M_{ij}v_j\right)$$

$$v_j = \left(\lambda\sigma^2 I + \sum_{i\in\Omega_{v_j}} u_i u_i^T\right)^{-1}\left(\sum_{i\in\Omega_{v_j}} M_{ij}u_i\right)$$

Note that we can not solve for all u_i and v_j at once to find the MAP solution. Thus we need to use a coordinate ascent kind algorithm to optimize the objective function L.

Algorithm:

MAP inference coordinate ascent algorithm

Input: An incomplete ratings matrix M, as indexed by the set Ω. Rank d.

Output: N_1 user locations, $u_i \in \mathbb{R}^d$, and N_2 object locations, $v_j \in \mathbb{R}^d$.

Initialize each v_j. For example, generate $v_j \sim N(0, \lambda^{-1}I)$.

for each iteration **do**

- **for** $i = 1, \ldots, N_1$ **update user location**

$$u_i = \left(\lambda\sigma^2 I + \sum_{j\in\Omega_{u_i}} v_j v_j^T\right)^{-1}\left(\sum_{j\in\Omega_{u_i}} M_{ij}v_j\right)$$

- **for** $j = 1, \ldots, N_2$ **update object location**

$$v_j = \left(\lambda\sigma^2 I + \sum_{i\in\Omega_{v_j}} u_i u_i^T\right)^{-1}\left(\sum_{i\in\Omega_{v_j}} M_{ij}u_i\right)$$

Predict that user i rates object j as $u_i^T v_j$ rounded to closest rating option

We can assess the convergence of the algorithm by calculating the log of joint likelihood and monitor that to see if it is converging to a value.

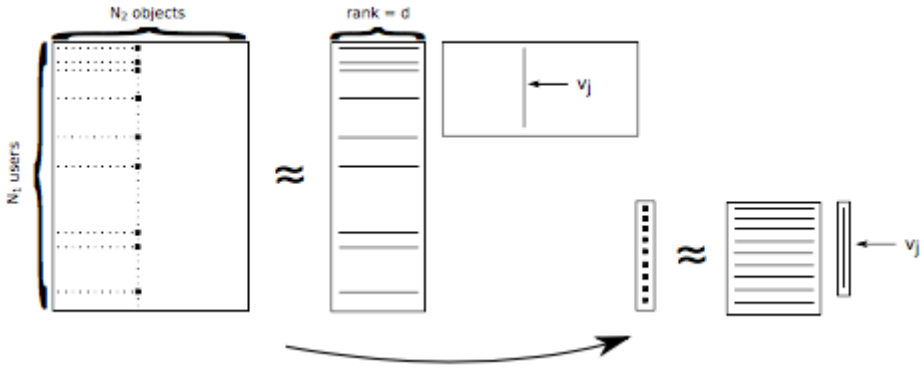
Matrix Factorization and Ridge Regression:

There is a very close relationship between CF using Matrix Factorization algorithm and ridge regression algorithm.

If we look at only object location v_j in the problem of Matrix factorization, we can see that we want to minimize the sum of squared error: $\frac{1}{\sigma}(M_{ij} - u_i^T v_j)^2$ with penalty $\lambda||v_j||^2$. The update for this would be:

$$v_j = \left(\lambda\sigma^2 I + \sum_{i\in\Omega_{v_j}} u_i u_i^T\right)^{-1}\left(\sum_{i\in\Omega_{v_j}} M_{ij}u_i\right)$$

Which is exactly the same update rule for the ridge regression. The following figure shows that the miniature problem is very similar to the ridge regression problem.



Therefore, we can actually view the matrix factorization problem as a sequence of $N_1 + N_2$ coupled ridge regression problem that can be solved iteratively.

Note that if we remove the Gaussian priors assumption for u_i and v_j we can also connect Matrix Factorization to the least square models and in that case the update rule would be:

$$v_j = \left(\sum_{i\in\Omega_{v_j}} u_i u_i^T\right)^{-1}\left(\sum_{i\in\Omega_{v_j}} M_{ij}u_i\right)$$

This is exactly the least square solution. Note that this requires that every user has rated at least d objects and every object is rated at least d users and since this is not always the case, we have to impose the priors to the model.

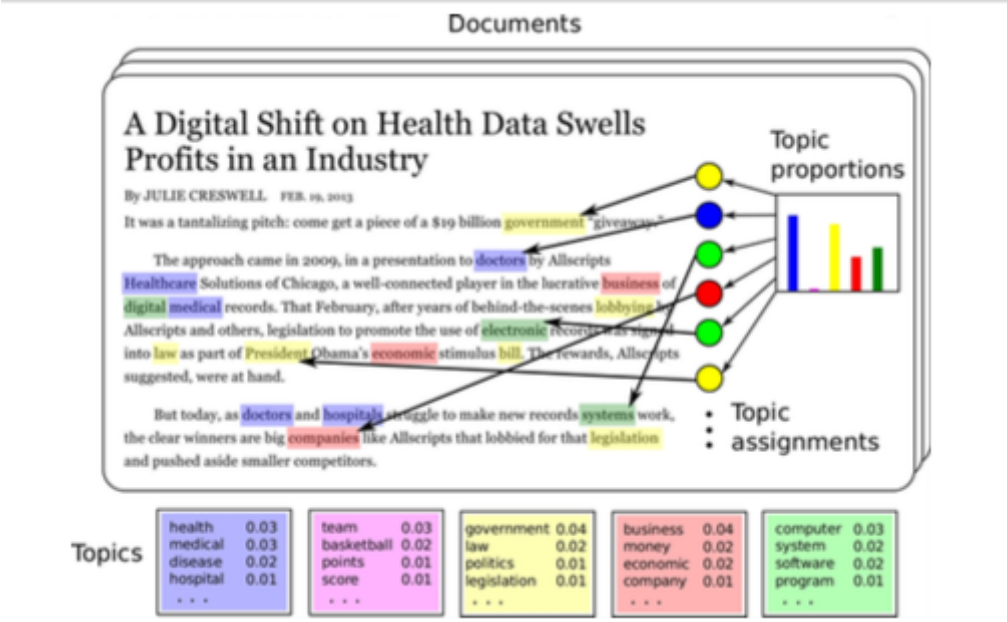
3. Topic modeling and non-negative matrix factorization

The subject of topic modeling is, given text data, when we want to organize, visualize, summarize, search, predict, or in general understand the data, we need to do some kind of topic modeling. Topic models allows us to discover theme in text data, annotate documents, organize and summarize the data.

A probabilistic topic model:

- Learns distributions on words called “topics” shared by documents

- Learns a distribution on topics for each document
- Assigns every word in a document to a topic



An example of a document and a learned topic model on that

Latent Diritchet Allocation (LDA):

LDA is a Bayesian model and therefore can be represented through a generative process. i.e, we hypothesize a generative process for the observed data, and then derive an inference algorithm for doing the inverse problem of learning the actual parameters or the posterior distribution of those parameters that could explain the data.

The way that we frame the generative process for LDA for topic modeling is a three step process:

1. Generate each topic, which is a distribution on words
   $\beta_k \sim Dirichlet(\gamma)$, k = 1, ..., K
2. For each document, generate a distribution on topics
   $\theta_d \sim Dirichlet(\alpha)$. d=1, ..., D
3. For the nth word in the dth document,
   a. Allocate the word to a topic, $c_{dn} \sim Discrete(\theta_d)$
   b. Generate the word from the selected topic, $x_{dn} \sim Dicscrete(\beta_{c_{dn}})$

Note that here, we are modeling language of the documents a bag-of-word approach and the ordering of words are not taken into consideration. In practice, this assumption is good enough to pick up the underlying theme of the document.

For the details of Dirichlet distribution, please see https://en.wikipedia.org/wiki/Dirichlet_distribution

LDA is very close related to matrix factorization. For example, we can see this relation by posing the question of calculating the probability of n-th word in the d-th document being equal to index i, given the

set of topics and the distributions on those topics. We can rewrite the following probability in the following form:

$$P(x_{dn} = i|\boldsymbol{\beta}, \theta) = \sum_{k=1}^{K} P(x_{dn} = i, c_{dn} = k|\boldsymbol{\beta}, \theta_d)$$
$$= \sum_{k=1}^{K} \underbrace{P(x_{dn} = i, |\boldsymbol{\beta}, c_{dn} = k)}_{=\beta_{ki}} \underbrace{P(c_{dn} = k|\theta_d)}_{=\theta_{dk}}$$

Let's write the same equation in matrix form:

$$B = [\beta_1, \beta_2, ..., \beta_K], \Theta = [\theta_1, \theta_2, ..., \theta_D] \qquad \therefore P(x_{dn} = i|\beta, \theta) = (B\Theta)_{id}$$

We can see that, in this form, topic modeling can be thought of as a non-negative matrix factorization. We want to learn a factorization of the Matrix P of non-negative probabilities, and represent that as the product of two matrices that are also include non-negative values in them. i.e, we can obtain the aforementioned probabilities, just by forming a matrix which is the product of two matrices that have non-negative entries. In fact, it can be shown that the divergence objective that I am going to talk about in the next section is closely related to LDA.

Simple algorithm to do topic modeling using NMF:

Step 1. Form the term-frequency matrix X. (X_{ij} = # times word i in doc j)

Step 2. Run NMF to learn W and H using $D(X\|WH)$ penalty

Step 3. As an added step, after Step 2 is complete, for $k = 1, \ldots, K$

1. Set $a_k = \sum_i W_{ik}$
2. Divide W_{ik} by a_k for all i
3. Multiply H_{kj} by a_k for all j

Notice that this is does not change the matrix multiplication WH.

Using NMF to do Topic Modeling

Non-negative Matrix Factorization

Non-negative Matrix factorization is a very useful method that can be used for topic modeling of text data, as we discussed before, or even it can be used for "parts-based" learning of image data for use cases such as factorizing face images. Each column of the factors learns something interpretable and this is a direct result of the non-negativity constraint. Some other usages of NMF are:

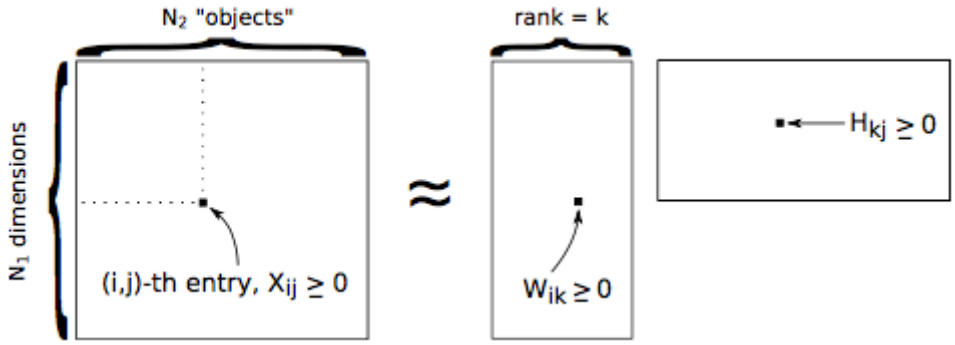
- feature learning
- clustering

- temporal segmentation
- filtering and source seperation
- etc

NMF is a non-exact factorization that factors into one skinny positive matrix and one short positive matrix. The important point is NMF is NP-hard and non-unique. There are a number of variations on it, each by considering some extra different constraints.

I have already discussed LDA as an instance of nonnegative matrix factorization. In this section I will go through the details of two other NMF models and their algorithms. Note that although “non-negative matrix factorization“ is a general technique, “NMF” usually just refers to the following two methods.

Unlike the probabilistic matrix factorization (PMF) that we discussed before, in NMF, data matrix X has nonnegative entries and none of the values are missing, though it is likely that many of the entries are zero. The goal is to learn factorization matrices W and H that are also only consist of nonnegative entries.



Non-negative Matrix Factorization

To obtain the non-negative factorization of data matrix X into W and H matrices, NMF minimizes one of the following two objective functions over W and H that leads into two different algorithms. In both algorithms, we have to impose the non-negative values constraint on W and H.

Objective function 1: Squared error objective

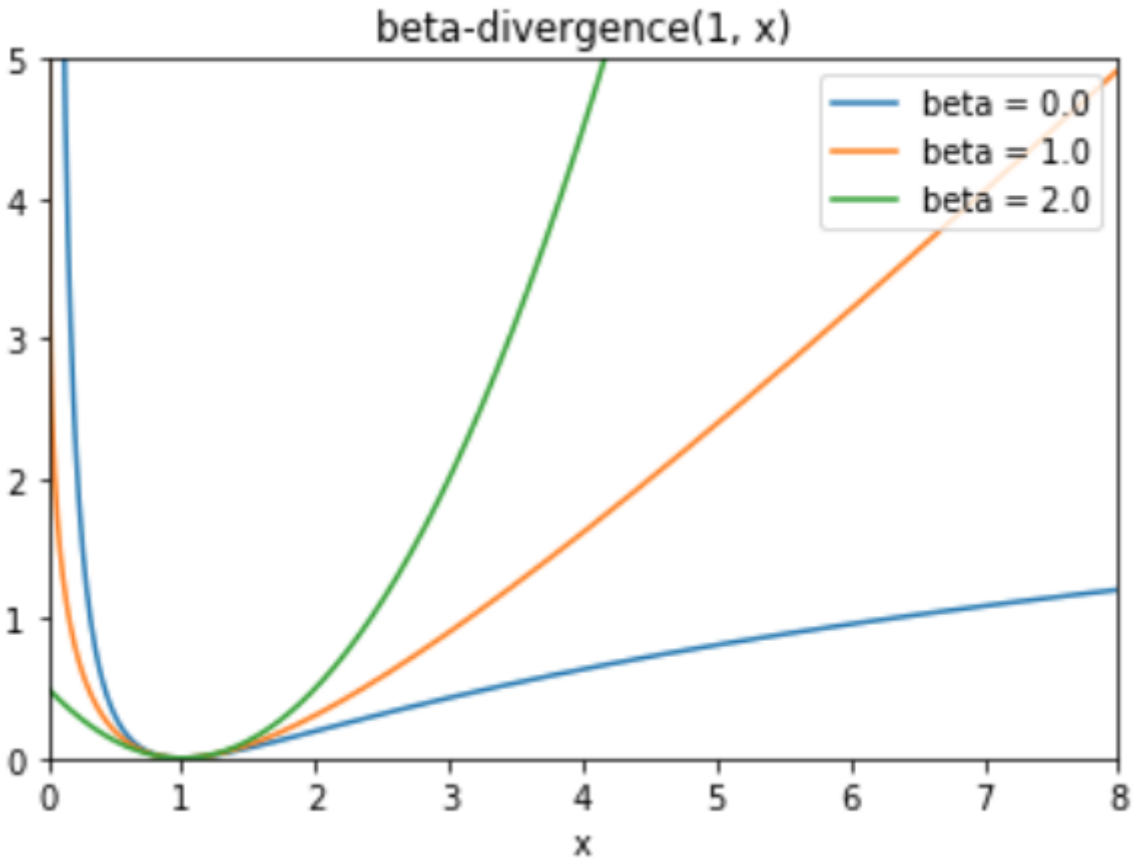
$$||X - WH||_{Fro}^2 = \sum_i \sum_j (X_{ij} - (WH)_{ij})^2$$

Objective function 2: Divergence objective

$$D(X||WH) = -\sum_i \sum_j [X_{ij} ln(WH)_{ij} - (WH)_{ij}]$$

Note: Other distance functions can also be used. For example, KL divergence, or Itakura-Saito (IS) divergence.These are all special cases of β-divergance family. I wrote a small piece of code that plots

different β-divergence functions for β = 0, 1, and 2 (Frobenius norm). See the code for the details.



beta-divergence function for different beta values

NMF uses a very fast multiplicative update rule for optimizing these two objectives.

Regularization:

Similar to many other objective functions, here also we can add L1 and L2 priors to the objective function in order to regularize the model. L2 penalty uses the Frobenius norm and L1 prior uses an elementwise L1 norm. we can control the combination of these two penalty term using another parameter l1_ratio. So in the most general case, we can have these penalty terms added to the above objective functions:

$$\alpha\rho||W||_1 + \alpha\rho||H||_1 + \frac{\alpha(1-/rho)}{2}||W||^2_{Fro} + \frac{\alpha(1-\rho)}{2}||H||^2_{Fro}$$

and the most general regularized objective function for the above objective function 1 would be:

$$||X - WH||^2_{Fro} + \alpha\rho||W||_1 + \alpha\rho||H||_1 + \frac{\alpha(1-/rho)}{2}||W||^2_{Fro} + \frac{\alpha(1-\rho)}{2}||H||^2_{Fro}$$

Multiplicative Algorithms

(see D.D. Lee and H.S. Seung (2001). "Algorithms for non-negative matrix factorization." *Advances in Neural Information Processing Systems.)*

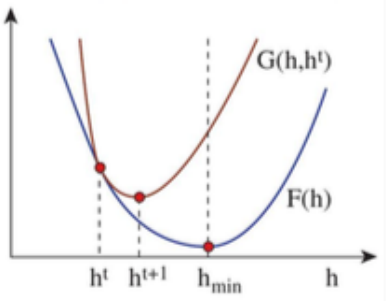
Here I will small details and mostly talk about the multiplicative algorithm to minimize the objectives that we already discussed.

The idea is when minimizing the objective "$\min_h$ F(h)", we should look for:

1. A way to generate a sequence of values $h^1, h^2, \ldots$ such that
$$F(h^1) \geq F(h^2) \geq F(h^3) \geq \ldots$$
2. Convergence of the sequence to a local minimum of F

The multiplicative algorithm that we are going to discuss fulfill these requirements. In this algorithm,

- Minimization is done using an "auxiliary function."
- It leads to a "multiplicative algorithm" for W and H.



Algorithm 1:

$\min \sum_{ij}(X_{ij} - (WH)_{ij})^2$ subject to $W_{ik} \geq 0, H_{kj} \geq 0$

- Randomly initialize H and W with non-negative values.
  - There are many more complicated ways to initialize H and W, such as Non-negative double singular value decomposition(nndsvd), or nndsvd with zeros filled with the average of X(nndsvda)
- Iterate the following, first for all values in H, then all in W:

$$H_{kj} \leftarrow H_{kj} \frac{(W^T X)_{kj}}{(W^T WH)_{kj}}$$

$$W_{ik} \leftarrow W_{ik} \frac{(XH^T)_{ik}}{(WHH^T)_{ik}}$$

loop until the change in $||X - WH||^2$ is "small".

Algorithm2:

min $\sum_{ij}[X_{ij} ln \frac{1}{(WH)_{ij}} + (WH)_{ij}]$ subject to $W_{ik} \geq 0, H_{kj} \geq 0$

- Randomly initialize H and W with nonnegative values.
  - There are many more complicated ways to initialize H and W, such as Non-negative double singular value decomposition(nndsvd), or nndsvd with zeros filled with the average of X(nndsvda)
- Iterate the following, first for all values in H, then all in W:

$$H_{kj} \leftarrow H_{kj} \frac{\sum_i W_{ik} X_{ij}/(WH)_{ij}}{\sum_i W_{ik}}$$

$$W_{ik} \leftarrow W_{ik} \frac{\sum_j H_{kj} X_{ij}/(WH)_{ij}}{\sum_j H_{kj}}$$

Loop until the change in $D(X||WH)$ is “small”.

Note: Here, in both cases, the nonnegativity of W and H is guaranteed by construction

4. Non-negative MF Implementation details:

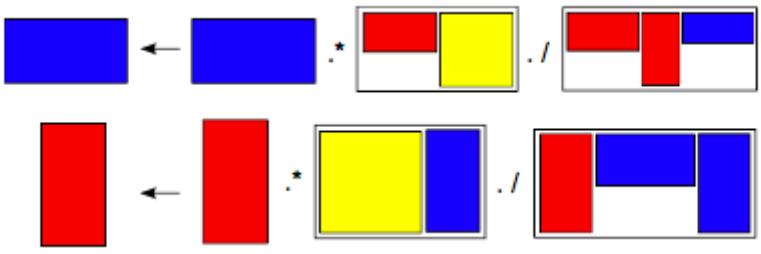
As mentioned before, the initialization method for W and H can have an important effect on the performance of NMF. Here I implemented random initialization to begin with. In the random intialization, I simply initialize W and H randomly and then scale them with sqrt(X.mean() / n_components).

If time permits, I will also implement Non-negative double singular value decomposition as well to compare with random initialization. NNDSVD is a very good choice for sparse factorization. Apparently, the multiplicative update is not able to update zeros present in the intialization and would lead to very poor results. In this case, other initialization methods should be used.

To be able to implement the two NMF algorithms correctly, I use a simple visualization. This way we are able to implement both methods in a vectorized way that helps speed up the algorithm many degrees of magnitude. This kind of visualization would be very helpful specially for implementing the second algorithm in a vectorized fashion.

Algorithm 1 :

Color-coded definition



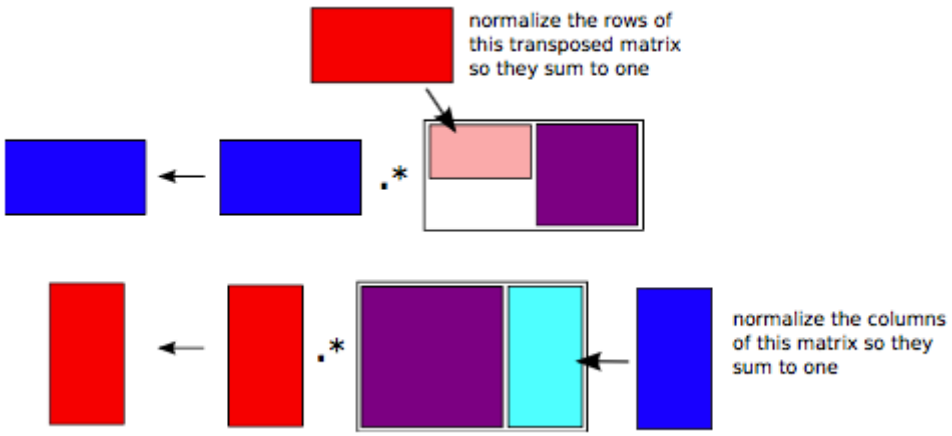
Here is how I implement this update rule in Python:

```
# update H
# H=H.*(W'*X)./(W'*W*H+eps)
H = H * ((A.T * W).T) / (W.T.dot(W).dot(H) + eps)
# update W
# W=W.*(X*H')./(W*(H*H')+eps);
W = W * (X * H.T) / (W.dot(H.dot(H.T)) + eps)
```

Algorithm 2:

- Here, note that “purpule box” is the data matrix “dot-devided” by the approximation of it.
- Here we iterate between : update purple → update blue → update purple → update red

Color-coded defintion



Principal Component Analysis and it’s variants

Principle component analysis (PCA) is an automatic way to map data $x \in R^d$ into a lower dimensional coordinate system, such that it captures most of the information embedded in the data. Extensions to PCA would also help us work with missing data, and “unwrap the data”.

The problem of solving for principle component analysis is very closely related to the eigen-decomposition problem of finding the largest eigenvalue and the corresponding eigenvector.

The optimization problem for PCA that considers K eigenvectors is:

$$q = \arg\min_{q} \sum_{i=1}^{n} \| x_i - \underbrace{\sum_{k=1}^{K} (x_i^T q_k) q_k}_{\text{approximates } x} \|^2 \qquad \text{s.t. } q_k^T q_{k'} = \begin{cases} 1, & k = k' \\ 0, & k \neq k' \end{cases}$$

$$= \arg\min_{q} \sum_{i=1}^{n} x_i^T x_i - \sum_{k=1}^{K} q_k^T \underbrace{\left(\sum_{i=1}^{n} x_i x_i^T \right)}_{= XX^T} q_k$$

The vectors in Q = $[q_1, q_2, ..., q_k]$ give us a K dimensional subspace with which we can easily represent the $x \in R^d$ with a lower dimensional vector $x_{proj} \in R^K$.

$$x_{\text{proj}} = \begin{bmatrix} q_1^T x \\ \vdots \\ q_K^T x \end{bmatrix}, \qquad x \approx \sum_{k=1}^{K} (q_k^T x) q_k = Q x_{\text{proj}}$$

Probabilistic PCA

In probabilistic PCA, we want to take PCA and view it as a probability model and learn the parameters using a probabilistic representation.

EM for Probabilistic PCA

Given: Data $x_{1:n}$, $x_i \in \mathbb{R}^d$ and model $x_i \sim N(Wz_i, \sigma^2)$, $z_i \sim N(0, I)$, $z \in \mathbb{R}^K$
Output: Point estimate of W and posterior distribution on each z_i

E-Step: Set each $q(z_i) = p(z_i|x_i, W) = N(z_i|\mu_i, \Sigma_i)$ where

$$\Sigma_i = (I + W^T W/\sigma^2)^{-1}, \quad \mu_i = \Sigma_i W^T x_i/\sigma^2$$

M-Step: Update W by maximizing the objective $\mathcal{L}$ from the E-step

$$W = \left[\sum_{i=1}^{n} x_i \mu_i^T\right] \left[\sigma^2 I + \sum_{i=1}^{n} (\mu_i \mu_i^T + \Sigma_i)\right]^{-1}$$

Iterate E and M steps until increase in $\sum_{i=1}^{n} \ln p(x_i|W)$ is "small."

EM for Probabilistic PCA

Kernel PCA

Kernel PCA is a very useful extension to PCA algorithm that can be used for nonlinear dimensionality reduction by embedding different kernel functions. It can be used for image denoising, and many other applications. Here, I don't want to go through the derivation of the kernel PCA and just add the algorithm for the completion of the discussion.

Kernel PCA

Given: Data $x_1, \ldots, x_n$, $x \in \mathbb{R}^d$, and a kernel function $K(x_i, x_j)$.

Construct: The kernel matrix on the data, e.g., $K_{ij} = b \exp\left\{-\frac{\|x_i - x_j\|^2}{c}\right\}$.

Solve: The eigendecomposition

$$\boldsymbol{K}\boldsymbol{a}_k = \lambda_k \boldsymbol{a}_k$$

for the first $r \ll n$ eigenvector/eigenvalue pairs $(\lambda_1, \boldsymbol{a}_1), \ldots, (\lambda_r, \boldsymbol{a}_r)$.

Output: A new coordinate system for x_i by (implicitly) mapping $\phi(x_i)$ and then projecting $q_k^T \phi(x_i)$

$$x_i \xrightarrow{\text{projection}} \begin{bmatrix} \lambda_1 a_{1i} \\ \vdots \\ \lambda_r a_{ri} \end{bmatrix}$$

where a_{ki} is the ith dimension of the kth eigenvector $\boldsymbol{a}_k$.

Kernel PCA algorithm

Question: How we handle new data, x_0? ...

Sparse PCA

Sparse PCA is also an extension of the plain-vanilla PCA that is pursuing the goal of extracting only a set of sparse components that can reconstruct the data efficiently. Sparse PCA can yield a much more interpretable representation compared to regular PCA. There are many different formulation of sparse PCA. Here, as requested in Assignment 9, I am going to implement an L1 penalized variant of PCA. Specifically, I am going to incorporate regularization on the principal components, in the form of a penalty function. This would lead to a **biconvex** optimization problem that can be solved quite rapidly. Therefore, the objective function would be:

$$(U, V) = argmin_{U,V} \tfrac{1}{2}||X - UV||_2^2 + \alpha||V||_1$$
subject to $||U_k||_2 = 1 \; \forall 0 \leq k \leq n_{components}$

Refrences:

- A review of principal components analysis in Introduction to Statistical Learning, Section 10.2.
- The paper "A penalized matrix decomposition, with applications to sparse principal components and canonical correlation analysis", by Witten, Tibshirani, and Hastie.
- Optional: more background on the singular value decomposition.

- Koren, Y., Robert B., and Volinsky, C.. “Matrix factorization techniques for recommender systems.” Computer 42.8 (2009): 30-37.
- “Learning the parts of objects by non-negative matrix factorization” D. Lee, S. Seung, 1999
- D.D. Lee and H.S. Seung (2001). “Algorithms for non-negative matrix factorization.” *Advances in Neural Information Processing Systems.*
- Cichocki, Andrzej, and P. H. A. N. Anh-Huy. “Fast local algorithms for large scale nonnegative matrix and tensor factorizations.” IEICE transactions on fundamentals of electronics, communications and computer sciences 92.3: 708-721, 2009.
- Fevotte, C., & Idier, J. (2011). Algorithms for nonnegative matrix factorization with the beta-divergence. Neural Computation, 23(9).
- “Structured Sparse Principal Component Analysis” R. Jenatton, G. Obozinski, F. Bach, 2009
- “Probabilistic Matrix Factorization for Automated Machine Learning”, Nicole Fusi, 2017
- “Non-negative Matrix Factorization with Sparseness Constraints” P. Hoyer, 2004
- http://scikit-learn.org/stable/modules/decomposition.html#decompositions