1 Non-negative Matrix Factorization Recommender

Lab Objective: *Understand and implement the non-negative matrix factorization for recommendation systems.*

Introduction

Collaborative filtering is the process of filtering data for patterns using collaboration techniques. More specifically, it refers to making prediction about a user's interests based on other users' interests. These predictions can be used to recommend items and are why collaborative filtering is one of the common methods of creating a recommendation system.

Recommendation systems look at the similarity between users to predict what item a user is most likely to enjoy. Common recommendation systems include Netflix's Movies you Might Enjoy list, Spotify's Discover Weekly playlist, and Amazon's Products You Might Like.

Non-negative Matrix Factorization

Non-negative matrix factorization is one algorithm used in collaborative filtering. It can be applied to many other cases, including image processing, text mining, clustering, and community detection. The purpose of non-negative matrix factorization is to take a non-negative matrix V and factor it into the product of two non-negative matrices.

For $V \in \mathbb{R}^{m\times n}, 0 \preceq W$,

$$\begin{aligned} \text{minimize} \quad & ||V - WH|| \\ \text{subject to} \quad & 0 \preceq W, 0 \preceq H \\ \text{where} \quad & W \in \mathbb{R}^{m\times k}, H \in \mathbb{R}^{k\times n} \end{aligned}$$

k is the rank of the decomposition and can either be specified or found using the Root Mean Squared Error (the square root of the MSE), SVD, Non-negative Least Squares, or cross-validation techniques.

For this lab, we will use the Frobenius norm, given by

$$||A||_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a|_{ij}^2}.$$

It is equivalent to the square root of the sum of the diagonal of $A^H A$

Problem 1. Create the `NMFRecommender` class, which will be used to implement the NMF algorithm. Initialize the class with the following parameters: `random_state` defaulting to 15, `tol` defaulting to $1e-3$, `maxiter` defaulting to 200, and `rank` defaulting to 3.

Add a method called `initialize_matrices` that takes in m and n, the dimensions of V. Set the random seed so that initializing the matrices can be replicated

```
>>> np.random.seed(self.random_state)
```

Initialize W and H using randomly generated numbers between 0 and 1, where $W \in \mathbb{R}^{m\times k}$ and $H \in \mathbb{R}^{k\times n}$. Return W and H.

Finally, add a method called `compute_loss()` that takes as parameters `V`, `W`, and `H` and returns the Frobenius norm of $V - WH$.

Multiplicative Update

After initializing W and H, we iteratively update them using the multiplicative update step. There are other methods for optimization and updating, but because of the simplicity and ease of this solution, it is widely used. As with any other iterative algorithm, we perform the step until the `tol` or `maxiter` is met.

$$H_{ij}^{n+1} = H_{ij}^{n}\frac{((W^n)^T V)_{ij}}{((W^n)^T W^n H^n)_{ij}} \tag{1.1}$$

and

$$W_{ij}^{n+1} = W_{ij}^{n}\frac{(V(H^{n+1})^T)_{ij}}{(W^n H^{n+1}(H^{n+1})^T)_{ij}} \tag{1.2}$$

Problem 2. Add a method to the `NMF` class called `update_matrices` that takes as inputs matrices V, W, H and returns W_{n+1} and H_{n+1} as described in Equations 1.1 and 1.2.

Problem 3. Finish the `NMF` class by adding a method `fit` that finds an optimal W and H. It should accept `V` as a numpy array, perform the multiplicative update algorithm until the loss is less than `tol` or `maxiter` is reached, and return W and H.

Finally add a method called `reconstruct` that reconstructs and returns `V` by multiplying `W` and `H`.

Using NMF for Recommendations

Consider the following marketing problem where we have a list of five grocery store customers and their purchases. We want to create personalized food recommendations for their next visit. We start by creating a matrix representing each person and the number of items they purchased in different grocery categories. So from the matrix, we can see that John bought two fruits and one sweet.

$$V = \begin{array}{c} \begin{matrix} John & Alice & Mary & Greg & Peter & Jennifer \end{matrix} \\ \begin{pmatrix} 0 & 1 & 0 & 1 & 2 & 2 \\ 2 & 3 & 1 & 1 & 2 & 2 \\ 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 2 & 3 & 4 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \end{array} \begin{matrix} \\ Vegetables \\ Fruits \\ Sweets \\ Bread \\ Coffee \end{matrix}$$

After performing NMF on V, we'll get the following W and H.

$$W = \begin{array}{c} \begin{matrix} Component1 & Component2 & Component3 \end{matrix} \\ \begin{pmatrix} 2.1 & 0.03 & 0. \\ 1.17 & 0.19 & 1.76 \\ 0.43 & 0.03 & 0.89 \\ 0.26 & 2.05 & 0.02 \\ 0.45 & 0. & 0. \end{pmatrix} \end{array} \begin{matrix} Vegetables \\ Fruits \\ Sweets \\ Bread \\ Coffee \end{matrix}$$

$$H = \begin{array}{c} \begin{matrix} John & Alice & Mary & Greg & Peter & Jennifer \end{matrix} \\ \begin{pmatrix} 0.00 & 0.45 & 0.00 & 0.43 & 1.0 & 0.9 \\ 0.00 & 0.91 & 1.45 & 1.9 & 0.35 & 0.37 \\ 1.14 & 1.22 & 0.55 & 0.0 & 0.47 & 0.53 \end{pmatrix} \end{array} \begin{matrix} Component1 \\ Component2 \\ Component3 \end{matrix}$$

W represents how much each grocery feature contributes to each component; a higher weight means it's more important to that component. For example, component 1 is heavily determined by vegetables followed by fruit, then coffee, sweets and finally bread. Component 2 is represented almost entirely by bread, while component 3 is based on fruits and sweets, with a small amount of bread. H is similar, except instead of showing how much each grocery category affects the component, it shows a much each person belongs to the component, again with a higher weight indicating that the person belongs more in that component. We can see the John belongs in component 3, while Jennifer mostly belongs in component 1.

To get our recommendations, we reconstruct V by multiplying W and H.

$$WH = \begin{array}{c} \begin{matrix} John & Alice & Mary & Greg & Peter & Jennifer \end{matrix} \\ \begin{pmatrix} 0.0000 & 0.9723 & 0.0435 & 0.96 & 2.1105 & 1.9011 \\ 2.0064 & 2.8466 & 1.2435 & 0.8641 & 2.0637 & 2.0561 \\ 1.0146 & 1.3066 & 0.533 & 0.2419 & 0.8588 & 0.8698 \\ 0.0228 & 2.0069 & 2.9835 & 4.0068 & 0.9869 & 1.0031 \\ 0.0000 & 0.2025 & 0.0000 & 0.1935 & 0.45 & 0.405 \end{pmatrix} \end{array} \begin{matrix} Vegetables \\ Fruits \\ Sweets \\ Bread \\ Coffee \end{matrix}$$

Most of the zeros from the original V have been filled in. This is the `collaborative filtering` portion of the algorithm. By sorting each column by weight, we can predict which items are more

attractive to the customers. For instance, Mary has the highest weight for bread at 2.9835, followed by fruit at 1.2435 and then sweets at .533. So we would recommend bread to Mary.

Another way to interpret WH is to look at a feature and determine who is most likely to buy that item. So if we were having a sale on sweets but only had funds to let three people know, using the reconstructed matrix, we would want to target Alice, John, and Jennifer in that order. This gives us more information that V alone, which says that everyone except Greg bought one sweet.

Problem 4. Use the `NMFRecommender` class to run NMF on V, defined above, with 2 components. Return W, H, and the number of people who have higher weights in component 2 than in component 1.

Sklearn NMF

Python has a few packages for recommendation algorithms: Surprise, CaseRecommender and of course SkLearn. They implement various algorithms used in recommendation models. We'll use SkLearn, which is similar to the NMFRecommender class, for the last problems.

```
from sklearn.decomposition import NMF

>>> model = NMF(n_components=2, init='random', random_state=0)
>>> W = model.fit_transform(X)
>>> H = model.components_
```

As mentioned earlier, many big companies use recommendation systems to encourage purchasing, ad clicks, or spending more time in their product. One famous example of a Recommendation system is Spotify's Discover Weekly. Every week, Spotify creates a playlist of songs that the user has not listened to on Spotify. This helps users find new music that they enjoy and keeps Spotify at the forefront of music trends.

Problem 5. Read the file `artist_user.csv` as a pandas dataframe. The rows represent users, with the user id in the first column, and the columns represent artists. For each artist j that a user i has listened to, the ij entry contains the number of times user i has listened to artist j.

Identify the rank, or number of components to use. Ideally, we want the smallest rank that minimizes the error. However, this rank may be too computationally expensive, as in this situation. We'll choose the rank by using the following method. First, calculate the frobenius norm of the dataframe and multiply it by .0001. This will be our benchmark value. Next, iterate through $rank = 3, 4, 5, \ldots$. For each iteration, run NMF using `n_components`=rank and reconstruct the matrix V. Calculate the root mean square error using `sklearn.metrics.mean_squared_error` of the original dataframe and the reconstructed matrix V. When the RMSE is less than the benchmark value, stop. Return the rank and the reconstructed matrix of this rank.

Problem 6. Write a function `discover_weekly` that takes in a user id and the reconstructed matrix from Problem 5, and returns a list of 30 artists to recommend.

The list should be sorted so that the first artist is the recommendation with the highest

weight and the last artist is the least, and it should not contain any artists that the user has already listed to. Use the file `artists.csv` to match the artist ID to their name.

As a check, the Discover Weekly for user 2 should return

[['Britney Spears'],['Avril Lavigne'],['Rihanna'],
['Paramore'],['Christina Aguilera'],['U2'],
['The Devil Wears Prada'],['Muse'],['Hadouken!'],
['Ke$ha'],['Good Charlotte'],['Linkin Park'],
['Enter Shikari'],['Katy Perry'],['Miley Cyrus'],
['Taylor Swift'],['Beyoncé'],['Asking Alexandria'],
['The Veronicas'],['Mariah Carey'],['Martin L. Gore'],
['Dance Gavin Dance'],['Erasure'], ['In Flames'],
['3OH!3'],['Blur'],['Kelly Clarkson'],
['Justin Bieber'],['Alesana'],['Ashley Tisdale']]

2 Inverse Problems

An important concept in mathematics is the idea of a well posed problem. The concept initially came from Jacques Hadamard. A mathematical problem is *well posed* if

1. a solution exists,
2. that solution is unique, and
3. the solution is continuously dependent on the data in the problem.

A problem that is not well posed is *ill posed*. Notice that a problem may be well posed, and yet still possess the property that small changes in the data result in larger changes in the solution; in this case the problem is said to be ill conditioned, and has a large condition number.

Note that for a physical phenomena, a well posed mathematical model would seem to be a necessary requirement! However, there are important examples of mathematical problems that are ill posed. For example, consider the process of differentiation. Given a function u together with its derivative u', let $\tilde{u}(t) = u(t) + \varepsilon \sin(\varepsilon^{-2} t)$ for some small $\varepsilon > 0$. Then note that

$$\|u - \tilde{u}\|_\infty = \varepsilon,$$

while

$$\|u' - \tilde{u}'\|_\infty = \varepsilon^{-1}.$$

Since a small change in the data leads to an arbitrarily large change in the output, differentiation is an ill posed problem. And we haven't even mentioned numerically approximating a derivative!

For an example of an ill posed problem from PDEs, consider the backwards heat equation with zero Dirichlet conditions:

$$\begin{aligned} &u_t = -u_{xx}, \quad (x,t) \in (0,L) \times (0,\infty), \\ &u(0,t) = u(L,t) = 0, \quad t \in (0,\infty), \\ &u(x,0) = f(x), \quad x \in (0,L). \end{aligned} \tag{2.1}$$

For the initial data $f(x)$, the unique[1] solution is $u(x,t) = 0$. Given the initial data $f(x) = \frac{1}{n}\sin(\frac{n\pi x}{L})$, one can check that there is a unique solution $u(x,t) = \frac{1}{n}\sin(\frac{n\pi x}{L})\exp((\frac{n\pi}{L})^2 t)$. Thus, on a finite interval $[0,T]$, as $n \to \infty$ we see that a small difference in the initial data results in an arbitrarily large difference in the solution.

[1] See *Partial Differential Equations* by Lawrence C. Evans, chapter 2.3, for a proof of uniqueness.

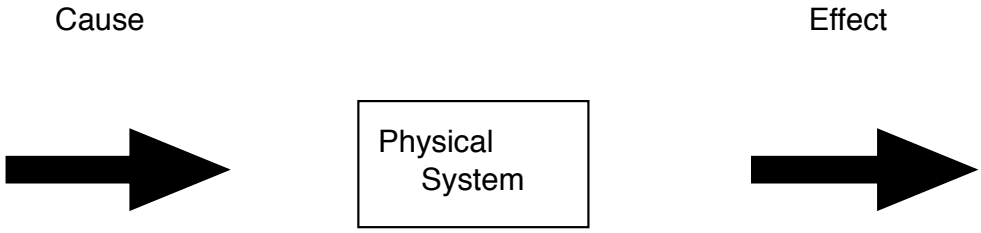

Figure 2.1: Cause and effect within a given physical system.

Inverse Problems

As implied by the name, inverse problems come in pairs. For example, differentiation and integration are inverse problems. The easier problem (in this case integration) is often called the direct problem. Historically, the direct problem is usually studied first.

Given a physical system, together with initial data (the "cause"), the direct problem will usually predict the future state of the physical system (the "effect"); see Figure 2.1. Inverse problems often turn this on its head - given the current state of a physical system at time T, what was the physical state at time $t = 0$?

Alternatively, suppose we measure the current state of the system, and we then measure the state at some future time. An important inverse problem is to determine an appropriate mathematical model that can describe the evolution of the system.

Another look at heat flow through a rod

Consider the following ordinary differential equation, together with natural boundary conditions at the ends of the interval[2]:

$$\begin{cases} -(au')' = f, & x \in (0,1), \\ a(0)u'(0) = c_0, & a(1)u'(1) = c_1. \end{cases} \tag{2.2}$$

This BVP can, for example, be used to describe the flow of heat through a rod. The boundary conditions would correspond to specifying the heat flux through the ends of the rod. The function $f(x)$ would then represent external heat sources along the rod, and $a(x)$ the density of the rod at each point.

Typically, the density $a(x)$ would be specified, along with any heat sources $f(x)$, and the (direct) problem is to solve for the steady-state heat distribution $u(x)$. Here we shake things up a bit: suppose

[2]This example of an ill-posed problem is given in *Inverse Problems in the Mathematical Sciences* by Charles W Groetsch.

the heat sources f are given, and we can measure the heat distribution $u(x)$. Can we find the density of the rod? This is an example of a *parameter estimation problem.*

Let us consider a numerical method for solving (2.2) for the density $a(x)$. Subdivide $[0,1]$ into N equal subintervals, and let $x_j = jh$, $j = 0, \ldots, N$, where $h = 1/N$. Let $\phi_j(x)$ be the tent functions (used earlier in the finite element lab), given by

$$\phi_j(x) = \begin{cases} (x - x_{j-1})/h & x \in [x_{j-1}, x_j], \\ (x_{j+1} - x)/h & x \in [x_j, x_{j+1}], \\ 0 & \text{otherwise.} \end{cases}$$

We look for an approximation $a^h(x)$ that is a linear combination of tent functions. This will be of the form

$$a^h = \sum_{j=0}^{N} \alpha_j \phi_j, \quad \alpha_j = a(x_j). \tag{2.3}$$

The h in this equation indicates that each of the tent functions in the linear combination rely on $h = 1/N$, and that a different h or N will result in different tent functions, so a^h will be different. The second half of (2.3) says that a good choice of a^h is found by taking $\alpha_j = a(x_j)$. Integrating (2.2) from 0 to x, we obtain

$$\begin{aligned} \int_0^x -(au')'\, ds &= \int_0^x f(s)\, ds, \\ -\left[a(x)u'(x) - c_0\right] &= \int_0^x f(s)\, ds, \\ u'(x) &= \frac{c_0 - \int_0^x f(s)\, ds}{a(x)}. \end{aligned} \tag{2.4}$$

Thus for each x_j

$$\begin{aligned} u'(x_j) &= \frac{c_0 - \int_0^{x_j} f(s)\, ds}{a(x_j)}, \\ &= \frac{c_0 - \int_0^{x_j} f(s)\, ds}{\alpha_j}. \end{aligned}$$

The coefficients α_j in (2.3) can now be approximated by minimizing

$$\left(\frac{c_0 - \int_0^{x_j} f(s)\, ds}{\alpha_j} - u'(x_j) \right)^2.$$

Problem 1. Solve (2.2) for $a(x)$ using the following conditions:
$c_0 = 3/8$, $c_1 = 5/4$, $u(x) = x^2 + x/2 + 5/16$, $x_j = .1j$ for $j = 0, 1, \ldots, 10$, and

$$f = \begin{cases} -6x^2 + 3x - 1 & x \le 1/2, \\ -1 & 1/2 < x \le 1, \end{cases}$$

Produce the plot shown in Figure 2.2.

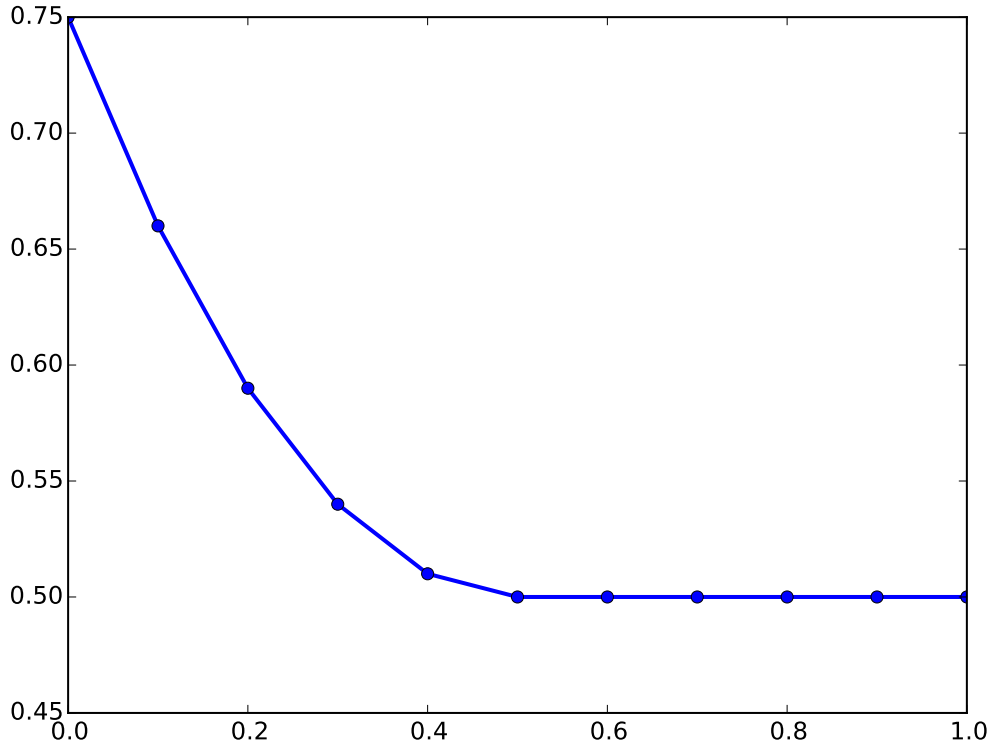

Figure 2.2: The solution $a(x)$ to Problem 1

Hint: use the `minimize` function in `scipy.optimize` and some initial guess to find the a_j.

Problem 2. Find the density function $a(x)$ satisfying

$$\begin{cases} -(au')' = -1, & x \in (0,1), \\ a(0)u'(0) = 1, & a(1)u'(1) = 2. \end{cases} \tag{2.5}$$

where $u(x) = x+1+\varepsilon\sin(\varepsilon^{-2}x)$. Using several values of $\varepsilon > 0.66049142$, plot the corresponding density $a(x)$ for x in `np.linspace(0,1,11)` to demonstrate that the problem is ill-posed.

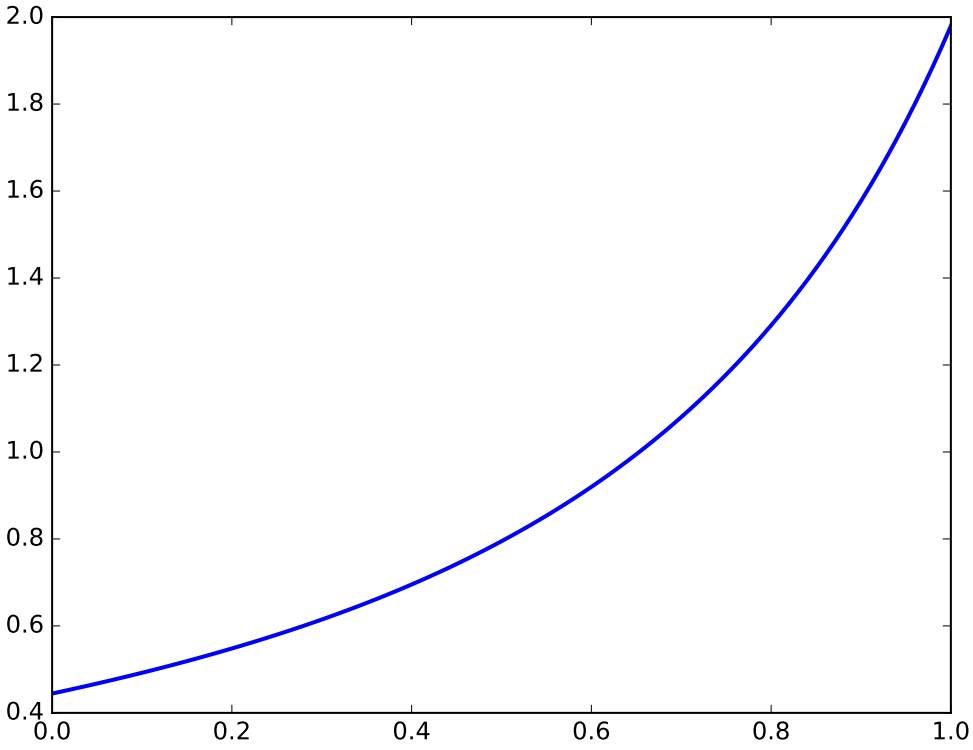

Figure 2.3: The density function $a(x)$ satisfying (2.5) for $\varepsilon = .8$.