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 2.

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)
```

Then, using `np.random.random`, initialize W and H with 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.

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

and

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

Problem 2. Add a method to the `NMF` class called `_update_matrices` that takes as inputs matrices V, W, H and returns W_{s+1} and H_{s+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 as matrices, and the number of people who have higher weights in component 2 than in component 1 as a float.

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`$= 2, 3, 4, 5, \ldots$. For each iteration, run NMF using `n_components`=rank and reconstruct the matrix V. Calculate the root mean square error (RMSE) by taking the square root of the MSE – calculated by `sklearn.metrics.mean_squared_error` – of the original dataframe and the reconstructed matrix V. If the RMSE is less than the benchmark value, stop. Return the rank and the reconstructed matrix of this rank.

Hint: the optimal rank can be found between 2 and 25, so you only actually need to iterate through `rank`$= 2, \ldots, 25$.

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 as strings.

This list of strings 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']