Chris Muro

AAI 627- Homework 7

Matrix Factorization Function

Synopsis and Analysis of Matrix Factorization Implementation

This Python script implements matrix factorization, a fundamental technique in data analysis and machine learning, particularly useful for recommendation systems and pattern recognition. The script employs two distinct approaches to matrix factorization: a custom-built function and the `NMF` method from the `sklearn.decomposition` library.

Matrix Factorization Function

The custom `matrix_factorization` function iteratively updates two matrices, `P` and `Q`, to approximate the original matrix `R`. It uses a form of gradient descent, adjusting `P` and `Q` to minimize the error between the product of `P` and `Q` and the original matrix. The function takes several parameters, including the matrix to be factorized (`R`), initial guesses for `P` and `Q`, the number of latent factors (`K`), and hyperparameters for learning rate (`alpha`) and regularization (`beta`).

Implementation and Results

The script runs matrix factorization on two different datasets. In the first part, a 5x4 matrix is factorized, and in the second part, a larger 6x5 matrix is processed. The results are evaluated by comparing the original matrices with the reconstructed matrices from both the custom function and `sklearn`'s NMF method. The outcomes indicate that both methods can approximate the original matrices to a varying degree of accuracy. The custom method seems to reconstruct the non-zero values of the original matrices well but struggles to accurately model the zero entries, often replacing them with non-zero predictions. This behavior suggests a potential overfitting to non-zero entries or insufficient regularization. On the other hand, the `sklearn` NMF method, which inherently enforces non-negativity, tends to produce sparser results, more accurately reflecting the zero entries in the original matrices. This characteristic makes NMF particularly suited for applications where the data is expected to have a non-negative and sparse nature, as is common in recommendation systems and certain types of image and signal processing tasks.

Conclusion

The comparison between the custom matrix factorization function and `sklearn`'s NMF method illustrates the trade-offs between a more flexible, from-scratch implementation and a specialized, constraint-based algorithm. While the custom function provides valuable insights into the mechanics of matrix factorization and allows for more control over the process, the `sklearn` NMF method offers a robust, well-optimized, and constraint-respecting alternative that may be more suitable for specific applications, particularly those involving sparse and non-negative data. This exercise underscores the importance of understanding the underlying data structure and the implications of algorithmic choices in matrix factorization tasks, guiding the selection of the appropriate method and parameterization to achieve accurate and meaningful results.

Part 1 Output:

```
PS C:\Users\cmuro\OneDrive\Documents 1\Stevens\AAI627
 'c:\Users\cmuro\.vscode\extensions\ms-python.debugpy
I627\HW7.py'
Using matrix_factorization function:
Original Matrix R:
 [[5 3 0 1]
 [4 0 0 1]
 [1 1 0 5]
 [1 0 0 4]
 [0 1 5 4]]
Matrix Factorization:
 [[4.98140718 2.96703776 5.61801269 1.00141146]
 [3.97627543 2.38025864 4.65952676 0.99955873]
 [1.03899205 0.90175821 5.33460511 4.9686936 ]
 [0.97988833 0.80836403 4.4117145  3.97778597]
 [1.51508648 1.12261931 4.94878984 4.0093181 ]]

Using sklearn NMR method:
Original Matrix R:
 [[5 3 0 1]
 [4 0 0 1]
 [1 1 0 5]
 [1 0 0 4]
 [0 1 5 4]]
Matrix Factorization NMR Method:
 [[5.25583751 1.99314304 0.         1.45510614]
 [3.50429883 1.32891643 0.         0.97018348]
 [1.31291255 0.9441558  1.94957474 3.94614513]
 [0.98126695 0.72179626 1.52760301 3.0788861 ]
 [0.         0.65008539 2.83998144 5.21892451]]
```

Part 2 Output:

```
Using matrix_factorization function:
Original Matrix R:
 [[4 3 0 1 2]
 [5 0 0 1 0]
 [1 2 1 5 4]
 [1 0 0 4 0]
 [0 1 5 4 0]
 [5 5 0 0 1]]
Matrix Factorization:
 [[ 3.96848755  3.00430508  1.75563236  1.00130327  1.98749125]
 [ 4.97878175  3.58338528  2.76671788  1.00041964  1.28427311]
 [ 0.99818096  2.02530013  0.99826263  4.96337106  3.96605603]
 [ 0.99988102  1.39727198  2.7428391   3.98476248  0.05235106]
 [ 1.09961498  1.03308322  4.97872716  3.97953795 -3.69130458]
 [ 4.99746476  4.95392827  7.03297202  8.49857382  1.01277827]]

Using sklearn NMR method:
Original Matrix R:
 [[4 3 0 1 2]
 [5 0 0 1 0]
 [1 2 1 5 4]
 [1 0 0 4 0]
 [0 1 5 4 0]
 [5 5 0 0 1]]
Matrix Factorization NMR Method:
 [[4.13756322 2.89171283 0.         1.33051642 1.55131109]
 [3.47925069 2.25937637 0.33463975 0.29062467 0.52938315]
 [1.21681397 1.78085517 0.68478585 5.39074865 3.49498465]
 [0.32115791 0.71422622 0.91603657 2.85213327 1.4640594 ]
 [0.2026001  0.74006266 4.84834994 4.21064773 0.        ]
 [5.7992552  3.73725001 0.         0.21697975 1.03584888]]
PS C:\Users\cmuro\OneDrive\Documents 1\Stevens\AAI627>
```