Project 3 Nakyazze Pricilla

June 22, 2025

What is Matrix factorization?

Matrix factorization is a class of collaborative filtering algorithms used in recommender systems. Matrix factorization algorithms work by decomposing the user-item interaction matrix into the product of two lower dimensionality rectangular matrices.This family of methods became widely known during the Netflix prize challenge due to its effectiveness as reported by Simon Funk in his 2006 blog post, where he shared his findings with the research community. The prediction results can be improved by assigning different regularization weights to the latent factors based on items' popularity and users' activeness for example.

The Techniques used include Funk MF, SVD++, Asymmetric SVD, Group-specific SVD, Hybrid MF and Deep-learning MF.

[4]:
```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

The goal is to find latent factors or relationships within the data and find patterns in fruit consumption over days.

[5]:
```python
data = {
    'bananas':       [3, 4, 3, 0, 2, 5, 1],
    'Apples':        [7, 2, 0, 5, 3, 1, 4],
    'Watermelon':    [0, 5, 3, 0, 6, 1, 2],
    'PassionFruit':  [6, 0, 5, 0, 1, 2, 3],
    'SugarCane':     [6, 8, 0, 5, 3, 2, 4]
}
index = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday',␣
 ↪'Sunday']
fruits = ['bananas', 'Apples', 'Watermelon', 'PassionFruit', 'SugarCane']
Fruit_df = pd.DataFrame(data, index=index)
```

Rows = days of the week (Monday to Sunday)

Columns = fruit types (bananas, Apples, etc.)

Values = Ratings of sales (1-10 scale)

```
[20]: # Convert to matrix

      print("Original Fruit df we are converting into Matrix (A):")
      print(Fruit_df)

      print("\n" + "="*50 + "\n")
      fruit_matrix = Fruit_df.to_numpy()

      print("Original Fruit Ratings Matrix (A):")
      print("Shape:", Fruit_df.shape, "(7 days × 5 Fruits)")

      print(fruit_matrix)
```

```
Original Fruit df we are converting into Matrix (A):
           bananas  Apples  Watermelon  PassionFruit  SugarCane
Monday           3       7           0             6          6
Tuesday          4       2           5             0          8
Wednesday        3       0           3             5          0
Thursday         0       5           0             0          5
Friday           2       3           6             1          3
Saturday         5       1           1             2          2
Sunday           1       4           2             3          4

==================================================

Original Fruit Ratings Matrix (A):
Shape: (7, 5) (7 days × 5 Fruits)
[[3 7 0 6 6]
 [4 2 5 0 8]
 [3 0 3 5 0]
 [0 5 0 0 5]
 [2 3 6 1 3]
 [5 1 1 2 2]
 [1 4 2 3 4]]
```

SVD breaks Fruit_df into three components:

Formula: A=U Σ VT Where A: Is the original Fruit Ratings Matrix that we are going to decompose.

U: Left singular vectors — this matrix tells you how the rows of the original matrix relate to the latent (hidden) features.

Σ (S): Singular values (in diagonal matrix) It tells the strength/importance of each latent feature.

V : Right singular vector: relates to how the columns (fruit types) relate to the same latent feature.

Singular Value Decomposition (SVD) for Fruit Ratings

```
[7]: # Perform SVD: A = U × Σ × V^T
     U, S, VT = np.linalg.svd(fruit_matrix, full_matrices=False)
```

The U matrix provides a compact representation of daily fruit consumption patterns, by capturing how each day aligns with latent features (underlying combinations of fruits).

[22]:
```python
# U matrix
print("SVD, U Matrix: Left singular vectors")
print(f"U matrix shape: {U.shape}")
for row in U:
    formatted_row = '\t'.join(f"{val:.3f}" for val in row)
    print(formatted_row)
print("\n" + "="*50 + "\n")
```

```
SVD, U Matrix: Left singular vectors
U matrix shape: (7, 5)
-0.561  -0.534  -0.331  0.067   0.041
-0.495  0.438   0.470   0.353   0.448
-0.203  0.343   -0.646  -0.183  0.314
-0.306  -0.416  0.390   -0.039  -0.205
-0.347  0.407   0.136   -0.606  -0.518
-0.246  0.212   -0.283  0.618   -0.584
-0.354  -0.148  -0.037  -0.295  0.219

==================================================
```

These singular values, which are the diagonal entries of Σ, are always non-negative real numbers.They are derived from the positive square roots of the eigenvalues of the matrix (A^{T}A) (or (AA^{T})).

[23]:
```python
# S vector (Singular Values)
print("\nSingular Values in diagonal matrix (S):")
print(f"S matrix shape: {S.shape}")
for val in S:
    print(f"{val:.3f}")
print("\n" + "="*50 + "\n")
```

```

Singular Values in diagonal matrix (S):
S matrix shape: (5,)
18.510
7.622
7.133
4.109
1.880

==================================================
```

[10]:
```python
# Create full Sigma matrix for visualization
Sigma = np.zeros((U.shape[1], VT.shape[0]))
np.fill_diagonal(Sigma, S)
print(f"Full Σ Matrix - Shape: {Sigma.shape}")
print(np.round(Sigma, 3))
print()
```

```
Full Σ Matrix - Shape: (5, 5)
[[18.51   0.     0.     0.     0.   ]
 [ 0.     7.622  0.     0.     0.   ]
 [ 0.     0.     7.133  0.     0.   ]
 [ 0.     0.     0.     4.109  0.   ]
 [ 0.     0.     0.     0.     1.88 ]]
```

U,V are orthogonal matrices, that represent the rotations or reflection of the space which are eigen vectors that are orthonormal to each other.The transformations can be interpreted as pure geometric changes (no distortion)

[11]:
```python
print("SVD, VT Matrix is the Right singular vector")
print(f"VT matrix shape: {VT.shape}")
for row in VT:
    formatted_row = '\t'.join(f"{val:.3f}" for val in row)
    print(formatted_row)
print("="*50 + "\n")
```

```
SVD, VT Matrix is the Right singular vector
VT matrix shape: (5, 5)
-0.354  -0.494  -0.330  -0.339  -0.638
0.381   -0.538  0.731   -0.145  -0.096
-0.313  0.077   0.123   -0.807  0.479
0.644   -0.341  -0.582  -0.187  0.309
-0.467  -0.586  -0.038  0.421   0.509
==================================================
```

Why do we verify reconstruction of original Matrix

To Evaluate Reconstruction Quality: Matrix factorization aims to decompose a matrix into simpler matrices. Multiplying these matrices together should reconstruct or approximate the original matrix. Verifying the reconstruction assesses the approximation's effectiveness. Better reconstruction quality indicates the decomposed matrices better represent the original data.

[24]:
```python
# Verify reconstruction
print("Verification: Reconstructing the original matrix")
reconstructed = U @ Sigma @ VT
print("U × Σ × V^T =")
print(np.round(reconstructed, 3))
print()
print("\n" + "="*50 + "\n")
```

```
Verification: Reconstructing the original matrix
U × Σ × V^T =
[[ 3.  7. -0.  6.  6.]
 [ 4.  2.  5.  0.  8.]
 [ 3.  0.  3.  5.  0.]
 [-0.  5. -0. -0.  5.]
 [ 2.  3.  6.  1.  3.]
 [ 5.  1.  1.  2.  2.]
 [ 1.  4.  2.  3.  4.]]


==================================================
```

Minimizing the error

The reconstruction error is the difference between the original matrix and the result of multiplying the smaller matrices.

There Practical limitations: In most cases, it's not possible to achieve a zero reconstruction error due to dimensionality reduction and noise in the data which is random, irrelevant, or inconsistent data that can distort the true patterns you're trying to learn or model.

[29]:
```
construction_error = np.linalg.norm(fruit_matrix - reconstructed)
print(f"Reconstruction error (should be ~0): {construction_error:.10f}")
print()

print("\n" + "="*50 + "\n")
```

```
Reconstruction error (should be ~0): 0.0000000000


==================================================
```

Dimensionality Reduction with SVD:

Reduces storage requirements by representing data with fewer dimensonsi, It can s speed up computation and potentiallyismprove the performance of machine learning models by reducing noise and irrelevant feates.n . SVD can reveal underlying patterns and relationships in the data by identifying nt factors.late

Approximation A rank-2 approximation is a simplified version of a matrix that captures its most important structure using only 2 latent features (or components) The equation Ak=Uk.Sigmak.Vk^T represents a low-rank approximation of a matrix A using the first k singular values and vectors from its Singular Value Decomposition. If you keep just the top 2 singular values, you're keeping the 2 strongest trenrs.

[27]:
```python
 # Keep only top 2 components
k = 2
U_reduced = U[:, :k]
sigma_reduced = np.diag(Sigma[:k])
Vt_reduced = VT[:k, :]

print(f"U_reduced shape: {U_reduced.shape}")
print(f"sigma_reduced shape: {sigma_reduced.shape}")
print(f"Vt_reduced shape: {Vt_reduced.shape}")

 # Reconstruct with reduced dimensions
sigma_reduced = np.diag(sigma_reduced)
approximated = U_reduced @ sigma_reduced @ Vt_reduced  # shape: (7, 5)
print("\nRanking reveals the underlying structure and simplifies data␣
 ↪representation")
print("\nRank-2 Approximation:")
df_approx = pd.DataFrame(np.round(approximated, 2), index=index, columns=fruits)
print(df_approx)
```

```
U_reduced shape: (7, 2)
sigma_reduced shape: (2,)
Vt_reduced shape: (2, 5)

Ranking reveals the underlying structure and simplifies data representation

Rank-2 Approximation:
           bananas  Apples  Watermelon  PassionFruit  SugarCane
Monday        2.12    7.32        0.45          4.11       7.01
Tuesday       4.51    2.73        5.47          2.62       5.52
Wednesday     2.32    0.45        3.15          0.89       2.14
Thursday      0.79    4.50       -0.45          2.38       3.91
Friday        3.45    1.50        4.39          1.73       3.80
Saturday      2.22    1.38        2.68          1.31       2.74
Sunday        1.89    3.85        1.34          2.39       4.29
```

Based on the approximation above on Tuesday, the expected quantity (or rating) for Watermelon consumption is approximately 5.47. So a manager would stock up due to relatively high expected sales on that day based on previous data.

Approximation Error

The approximation error is the difference between the original matrix Fruit_matrixX) and its truncated SVD approximation Rank_2 ApproximationX). It is commonly measured using the Frobenius norm. This norm represents the “size” of the difference between the two matrices. The Eckart-Young theorem proves that the truncated SVD provides the best low-rank approximation of a matrix, minimizing the approximation error as measured by the Frobenius norm. The approximation error is equal to the sum of the squares of the discarded singular valu

The Frobenius norm tells you how far your rank-2 approximation is from the original matrix. A defined as the square root of the sum of the absolute squares of its elements,

||A||*F=sqrt(sum*(i=1)$^{\text{msum_(j=1)}}$n|a_(ij)|^2)

Smaller = better approximation.

In SVD, the energy of the matrix is the sum of squares of all singular .

es.

[30]:
```
approximation_error = np.linalg.norm(fruit_matrix - approximated)
print(f"\nApproximation error with Smaller = better approximation.:␣
↪{approximation_error:.3f}")

# Show which singular values we kept vs discarded
print(f"\nSingular values kept: {np.round(sigma_reduced, 3)}")
print(f"Singular values discarded: {np.round(Sigma[k:], 3)}")
print(f"Energy retained: {np.sum(sigma_reduced**2) / np.sum(Sigma**2):.1%}")
```

```
Approximation error with Smaller = better approximation.: 8.443

Singular values kept: [[18.51   0.   ]
 [ 0.     7.622]]
Singular values discarded: [[0.    0.    7.133 0.    0.   ]
 [0.    0.    0.    4.109 0.   ]
 [0.    0.    0.    0.    1.88 ]]
Energy retained: 84.9%
```

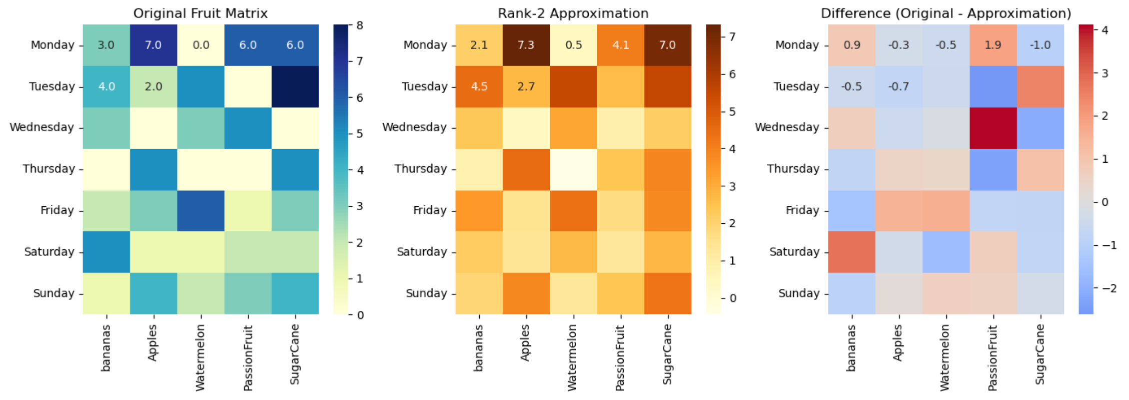
8.443 Approximation error represents the total reconstruction loss across all 35 matrix entries (7×5). 84.9% of energy retained means only 15% of variation is not captured. I think this is a good error. ”===

[32]:
```
# Define prediction function
A_k = U_reduced @ sigma_reduced @ Vt_reduced
A_k_df = pd.DataFrame(np.round(A_k, 2), index=index, columns=fruits)

def predict_rating(day, fruit):
    try:
        return round(A_k_df.loc[day, fruit], 2)
    except KeyError:
        return "Invalid day or fruit name"

# Example usage
print("Predicted rating for Friday - PassionFruit:", predict_rating("Friday",␣
↪"PassionFruit"))
```

```
Predicted rating for Friday - PassionFruit: 1.73
```

The model predicts that on Friday, the expected quantity (or rating) for PassionFruit consumption is approximately 1.73. So a manager would choose not to stock based on daily preference.

```
[17]: # Step 5: Visualization

# a) Original vs Reconstructed Matrix Heatmap
plt.figure(figsize=(14, 5))

plt.subplot(1, 3, 1)
sns.heatmap(Fruit_df, annot=True, cmap="YlGnBu", fmt=".1f")
plt.title("Original Fruit Matrix")

plt.subplot(1, 3, 2)
sns.heatmap(A_k_df, annot=True, cmap="YlOrBr", fmt=".1f")
plt.title("Rank-2 Approximation")

# b) Difference Heatmap (Error Visualization)
difference = Fruit_df - A_k_df
plt.subplot(1, 3, 3)
sns.heatmap(difference, annot=True, cmap="coolwarm", fmt=".1f", center=0)
plt.title("Difference (Original - Approximation)")

plt.tight_layout()
plt.show()
```



The Rank-2 Approximation captures the trends in fruit rating across days. Compared to Monday's original rating of 7.0 the approximation is 7.3 with just a difference of -0.3. This is pretty close and if I was a store manager going by past ratings and sales to determine stocking needs this would be a sufficient recommendation.

SVD's lower ranks ignore minor, high-frequency detail to highlight global patterns. ”===

Conclusion

Why do we need SVD? In addition to its use in recommender systems, matrix factorization has several other practical applications. It's commonly used for dimensionality reduction, where large data sets are simplified by breaking them into lower-rank matrice. The SVD decomposition function simplifies complex data by breaking it into three smaller parts. This helps uncover hidden patterns and relationships, making it easier to analyze and work with large datasets. SVD is useful in, data compression, and finding important features, making data simpler and more manageable.

What are the disadvantages of matrix factorization?

Matrix Factorization has some limitations: Sparse Data Dependency where Performance may degrade with very sparse matrices. Computational Costs with Large datasets making factorization computationally expensive. Overfitting Risks Without regularization, factorized models may overfit the data.

Citation https://en.wikipedia.org/wiki/Matrix_factorization_(recommender_systems)#SVD++