JHANVI PAREKH

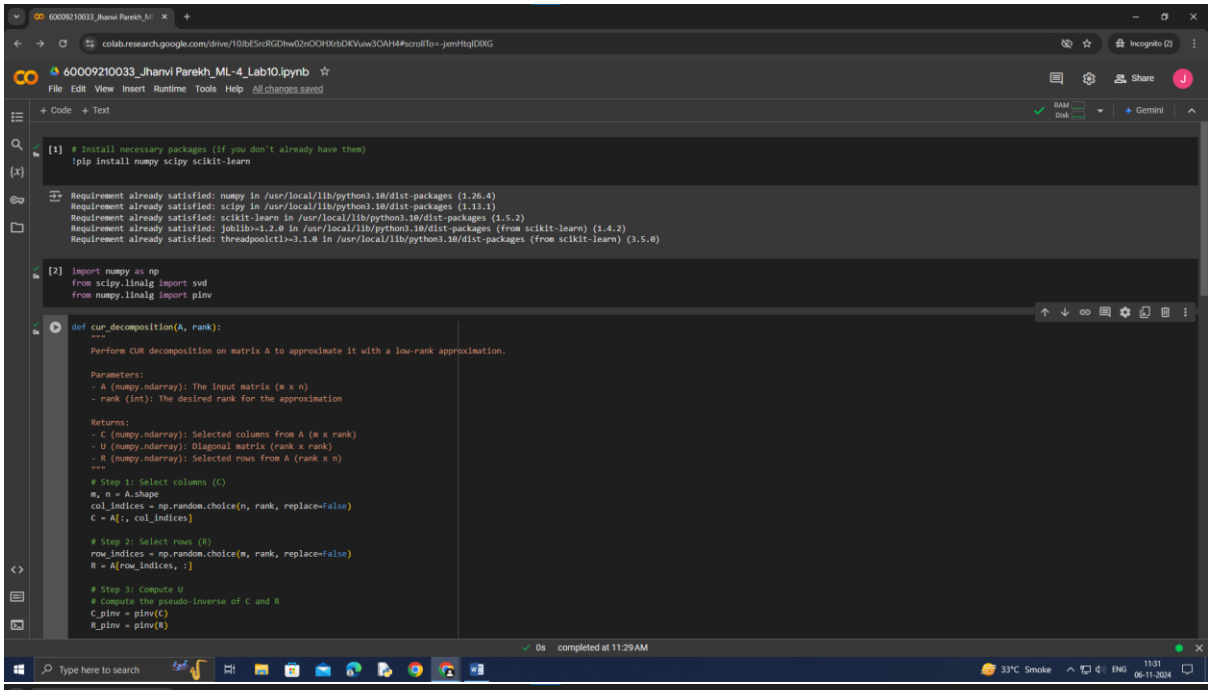
60009210033

CSE(DS)

Experiment 10

Implementation of CURE Algorithm

Link:
https://colab.research.google.com/drive/10JbESrcRGDhw02nOOHXrbDKVuiw3OAH4?usp=sharing



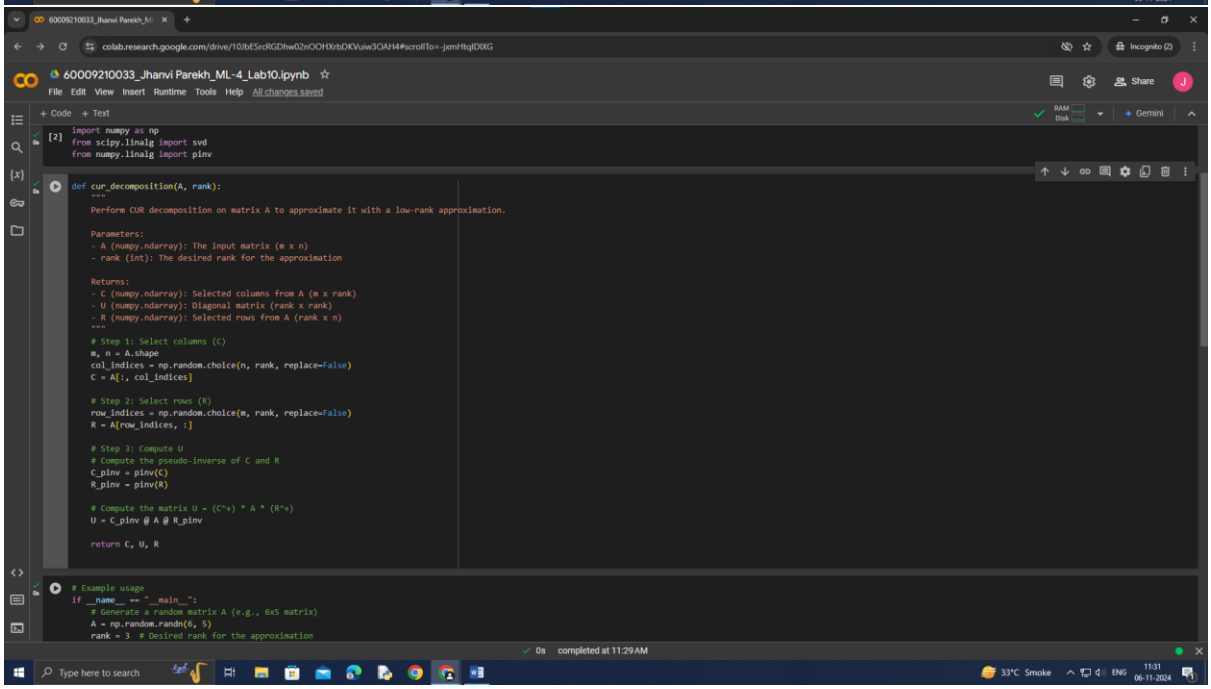
```
# Install necessary packages (if you don't already have them)
!pip install numpy scipy scikit-learn
```

```
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (1.26.4)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (1.13.1)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (1.5.2)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (3.5.0)
```

```
import numpy as np
from scipy.linalg import svd
from numpy.linalg import pinv
```

```
def cur_decomposition(A, rank):
    """
    Perform CUR decomposition on matrix A to approximate it with a low-rank approximation.

    Parameters:
    - A (numpy.ndarray): The input matrix (m x n)
    - rank (int): The desired rank for the approximation

    Returns:
    - C (numpy.ndarray): Selected columns from A (m x rank)
    - U (numpy.ndarray): Diagonal matrix (rank x rank)
    - R (numpy.ndarray): Selected rows from A (rank x n)
    """
    # Step 1: Select columns (C)
    m, n = A.shape
    col_indices = np.random.choice(n, rank, replace=False)
    C = A[:, col_indices]

    # Step 2: Select rows (R)
    row_indices = np.random.choice(m, rank, replace=False)
    R = A[row_indices, :]

    # Step 3: Compute U
    # Compute the pseudo-inverse of C and R
    C_pinv = pinv(C)
    R_pinv = pinv(R)
```

0s completed at 11:29 AM

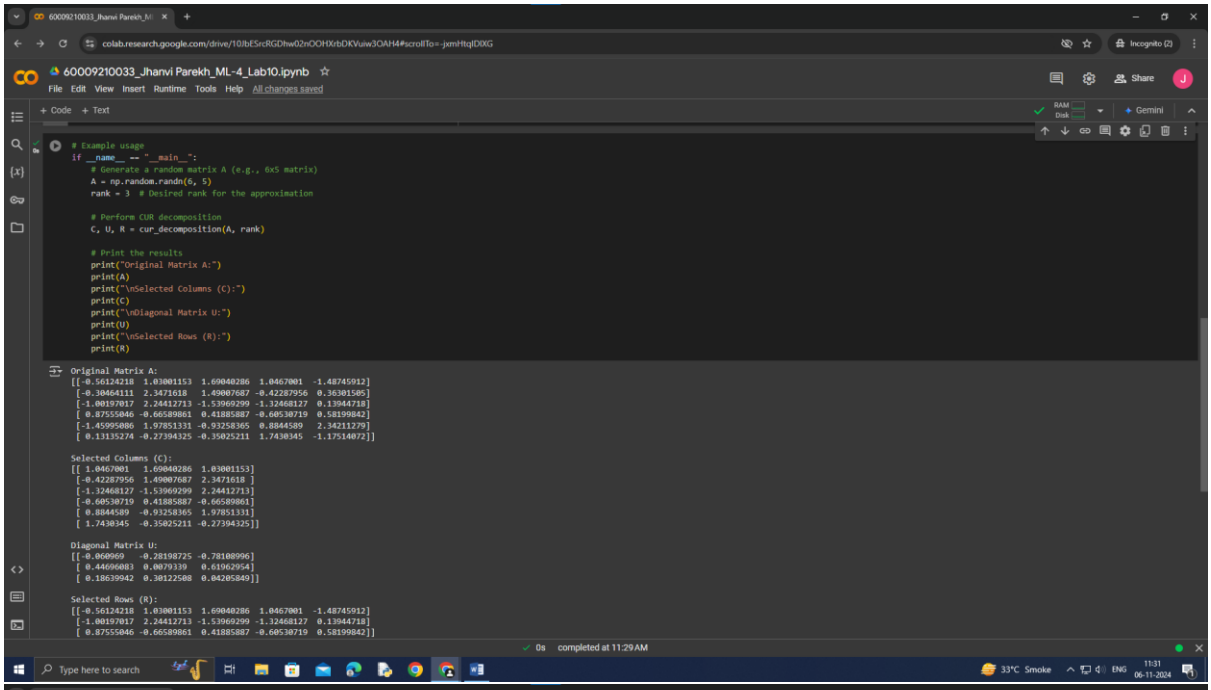


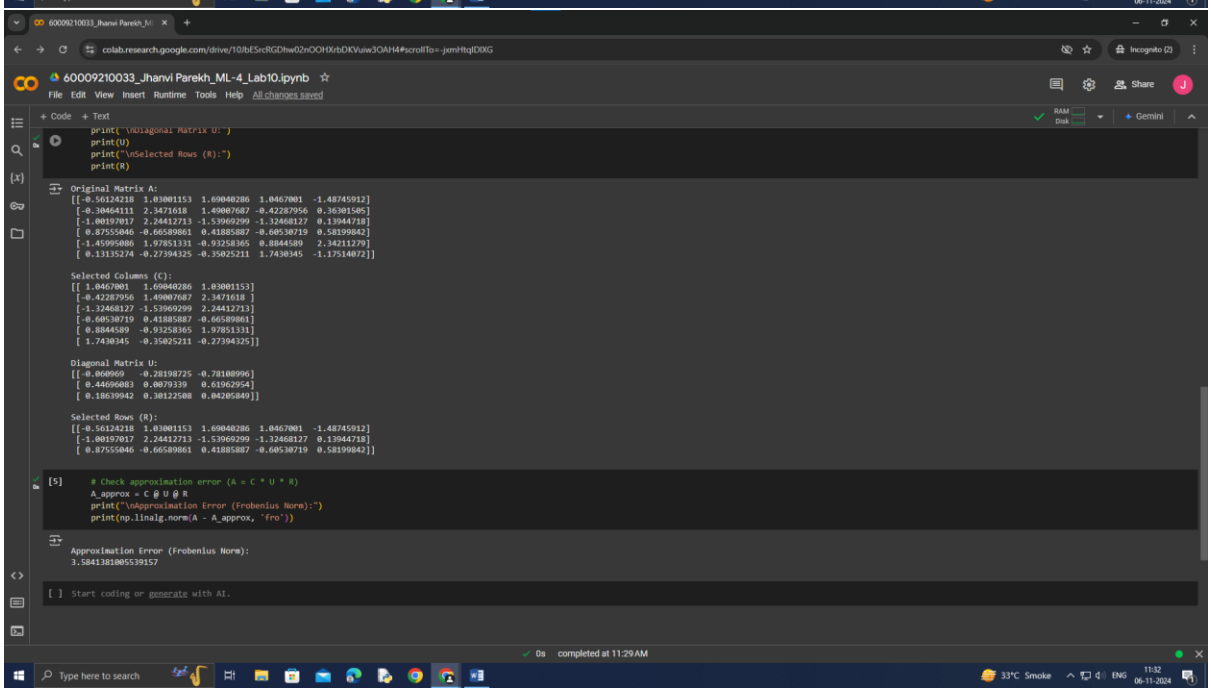
```
import numpy as np
from scipy.linalg import svd
from numpy.linalg import pinv
```

```
def cur_decomposition(A, rank):
    """
    Perform CUR decomposition on matrix A to approximate it with a low-rank approximation.

    Parameters:
    - A (numpy.ndarray): The input matrix (m x n)
    - rank (int): The desired rank for the approximation

    Returns:
    - C (numpy.ndarray): Selected columns from A (m x rank)
    - U (numpy.ndarray): Diagonal matrix (rank x rank)
    - R (numpy.ndarray): Selected rows from A (rank x n)
    """
    # Step 1: Select columns (C)
    m, n = A.shape
    col_indices = np.random.choice(n, rank, replace=False)
    C = A[:, col_indices]

    # Step 2: Select rows (R)
    row_indices = np.random.choice(m, rank, replace=False)
    R = A[row_indices, :]

    # Step 3: Compute U
    # Compute the pseudo-inverse of C and R
    C_pinv = pinv(C)
    R_pinv = pinv(R)

    # Compute the matrix U = (C^+) * A * (R^+)
    U = C_pinv @ A @ R_pinv

    return C, U, R
```

```
# Example usage
if __name__ == "__main__":
    # Generate a random matrix A (e.g., 6x5 matrix)
    A = np.random.randn(6, 5)
    rank = 3  # Desired rank for the approximation
```

0s completed at 11:29 AM


60009210033_Jhanvi Parekh_ML-4_Lab10.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Share
+ Code + Text
Gemini
# Example usage
if __name__ == "__main__":
# Generate a random matrix A (e.g., 6x5 matrix)
A = np.random.randn(6, 5)
rank = 3 # Desired rank for the approximation
# Perform CUR decomposition
C, U, R = cur_decomposition(A, rank)
# Print the results
print("Original Matrix A:")
print(A)
print("\nSelected Columns (C):")
print(C)
print("\nDiagonal Matrix U:")
print(U)
print("\nSelected Rows (R):")
print(R)
Original Matrix A:
[[-0.56124218 1.03001153 1.69040286 1.0467001 -1.48745912]
[-0.30464111 2.3471618 1.49007687 -0.42287956 0.36301505]
[-1.00197017 2.24412713 -1.53969299 -1.32468127 0.13944718]
[0.87555046 -0.66589861 0.41885887 -0.60530719 0.58199842]
[-1.45995086 1.97851331 -0.93258365 0.8844589 2.34211279]
[0.13135274 -0.27394325 -0.35025211 1.7430345 -1.17514072]]
Selected Columns (C):
[[1.0467001 1.69040286 1.03001153]
[-0.42287956 1.49007687 2.3471618]
[-1.32468127 -1.53969299 2.24412713]
[-0.60530719 0.41885887 -0.66589861]
[0.8844589 -0.93258365 1.97851331]
[1.7430345 -0.35025211 -0.27394325]]
Diagonal Matrix U:
[[-0.060969 -0.28198725 -0.78108996]
[0.44696083 0.0079339 0.61962954]
[0.18639942 0.30122508 0.84205849]]
Selected Rows (R):
[[-0.56124218 1.03001153 1.69040286 1.0467001 -1.48745912]
[-1.00197017 2.24412713 -1.53969299 -1.32468127 0.13944718]
[0.87555046 -0.66589861 0.41885887 -0.60530719 0.58199842]]
0s completed at 11:29AM
Type here to search
33°C Smoke
ENG
11:31
06-11-2024


60009210033_Jhanvi Parekh_ML-4_Lab10.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Share
+ Code + Text
Gemini
print(U)
print("\nSelected Rows (R):")
print(R)
Original Matrix A:
[[-0.56124218 1.03001153 1.69040286 1.0467001 -1.48745912]
[-0.30464111 2.3471618 1.49007687 -0.42287956 0.36301505]
[-1.00197017 2.24412713 -1.53969299 -1.32468127 0.13944718]
[0.87555046 -0.66589861 0.41885887 -0.60530719 0.58199842]
[-1.45995086 1.97851331 -0.93258365 0.8844589 2.34211279]
[0.13135274 -0.27394325 -0.35025211 1.7430345 -1.17514072]]
Selected Columns (C):
[[1.0467001 1.69040286 1.03001153]
[-0.42287956 1.49007687 2.3471618]
[-1.32468127 -1.53969299 2.24412713]
[-0.60530719 0.41885887 -0.66589861]
[0.8844589 -0.93258365 1.97851331]
[1.7430345 -0.35025211 -0.27394325]]
Diagonal Matrix U:
[[-0.060969 -0.28198725 -0.78108996]
[0.44696083 0.0079339 0.61962954]
[0.18639942 0.30122508 0.84205849]]
Selected Rows (R):
[[-0.56124218 1.03001153 1.69040286 1.0467001 -1.48745912]
[-1.00197017 2.24412713 -1.53969299 -1.32468127 0.13944718]
[0.87555046 -0.66589861 0.41885887 -0.60530719 0.58199842]]
[5]
# Check approximation error (A ≈ C * U * R)
A_approx = C @ U @ R
print("\nApproximation Error (Frobenius Norm):")
print(np.linalg.norm(A - A_approx, 'fro'))
Approximation Error (Frobenius Norm):
3.5841381005539157
[ ] Start coding or generate with AI.
0s completed at 11:29AM
Type here to search
33°C Smoke
ENG
11:32
06-11-2024