LINEAR ALGEBRA

<u>HACKATHON-2</u>

Date: 16/04/2025

TEAM MEMBERS:

1.M Niranjan - PES2UG23CS308.

2.Keerthan P.V – PES2UG23CS272.

Q.Customer Behavior Analysis for Retail Optimization.

Problem explination:

Here, in the given question the retail company has collected the data on 15 different behaviours from 10,000 customers. Here we need to group the customers based on the segments and target them. But here analysing the 15 different behaviour together will be hard to visualize . so we need to reduce the number dimensions but keeping the important information.

Our approach:

1.Preprocessing:

Here we center the data by subtracting the mean of each column. This helps to focus on how the data varies instead where it’s located.

2.Finding direction:

Next we use svd concept to find the direction and its called principal components. All of these are right angles to each other and provides new , non – overlapping information.

3. Reducing the Dimension:

Here, we pick the top k directions which only contains the most important changes in the data about 95%. This simplifys the data and it reduces the dimensions.

4.Reconstructing the original data.

Here, the simplified data and the top directions are used to construct something which is close to the original data.

5.Error measurement:

We compare the orginal data with the reconstructed data and

To check how much it deviates from the original data.if the error is less it indicates that we kept the important information.

How do we pick the value of k:

Here we will choose the smallest number k of directions that captures atleast 95% of the total variation in the data.by using the formula ,
Variance Retained=Total variance/Sum of top k variances

Once it reaches the ratio above 95% , we stop and it’s the number of dimenaions we keep.

The relation between full and reduced data:

The original data has 15 dimensions and the reduced data has 2 or 3 dimensions.

if the features are same for every customer , then they don’t provide anything new.after that we center the data all the features becomes zero and pca ingones them.

Code:

```
import numpy as np

import sys


class DimensionalityReducer:

    def _init_(self):

        self.mean = None

        self.basis = None  # Principal components (rows)

        self.variance_explained = None


    def preprocess(self, X):

        self.mean=np.mean(X,axis=0)

        return X-self.mean


    def compute_key_directions(self, X_centered):

        a,b,Vt=np.linalg.svd(X_centered, full_matrices=False)

        self.basis=Vt

        total_variance=np.sum(b**2)

        self.variance_explained=(b**2)/total_variance if
total_variance >0 else np.zeros_like(b)
```

```
    def reduce_dimensions(self, X_centered, k):
        return X_centered@self.basis[:k].T

    def reconstruct(self, X_reduced):
        ##HERE, FILL IN THE BLANKS
        if X_reduced.size == 0 :
            return np.empty_like(self.mean)
        return X_reduced @
self.basis[:X_reduced.shape[1]]+self.mean

    def evaluate_error(self, X_original, X_reconstructed):
        return np.linalg.norm(X_original-X_reconstructed)

def main():
    # Read input matrix from stdin
```

```
A = []
while True:
    try:
        row = input().strip()
        if row:
            A.append(list(map(float, row.split())))
        else:
            break
    except EOFError:
        break
A = np.array(A)

reducer = DimensionalityReducer()

# Step 1: Preprocess (center the data)
try:
    X_centered = reducer.preprocess(A)
except Exception as e:
    print("Error during preprocessing:", e)
```

```
        return

    print("Centered data:")
    print(X_centered)

    # Step 2: Compute key directions (SVD)
    try:
        reducer.compute_key_directions(X_centered)
    except Exception as e:
        print("Error during key directions computation:", e)
        return

    # Determine k for 95% variance
    cumulative_variance =
np.cumsum(reducer.variance_explained)
    k = np.argmax(cumulative_variance >= 0.95) + 1
    if k == 0:  # Handle case where no component meets the
threshold
        k = len(reducer.variance_explained)
    # Handle case when all variances are zero (e.g., constant
features)
    if np.allclose(X_centered, 0):
        k = 0

    # Step 3: Reduce dimensions
    try:
        if k > 0:
            X_reduced = reducer.reduce_dimensions(X_centered, k)
        else:
            X_reduced = np.zeros((X_centered.shape[0], 0))
    except Exception as e:
        print("Error during dimensionality reduction:", e)
        return

    print("\nTop directions:")
    if k > 0:
        for direction in reducer.basis[:k]:
            print(" ".join(f"{x:.2f}" for x in direction))
```

```
else:
    print("No directions (all features are constant)")

print("\nReduced data:")
if X_reduced.size > 0:
    for row in X_reduced:
        print(" ".join(f"{x:.2f}" for x in row))
else:
    print("No reduced data (all features are constant)")

# Step 4: Reconstruct data
try:
    X_reconstructed = reducer.reconstruct(X_reduced)
except Exception as e:
    print("Error during reconstruction:", e)
    return

print("\nReconstructed data:")
print(X_reconstructed)
```

```
    # Step 5: Evaluate reconstruction error
    try:
        error = reducer.evaluate_error(A, X_reconstructed)
    except Exception as e:
        print("Error during error evaluation:", e)
        return

    print(f"\nReconstruction error: {error:.2f}")

if __name__ == "__main__":
    main()
```

explaination of the return values:

```
class DimensionalityReducer:
    def _init_(self):
        self.mean = None
        self.basis = None  # Principal components (rows)
        self.variance_explained = None
```

- This function runs automatically when we create the object. It sets up empty "boxes" (self.mean, self.basis, and self.variance_explained) to store values later. These will be used to keep track of the mean of your data, the directions PCA finds (basis), and how much variance each direction explains.

```
def preprocess(self, X):
    self.mean = np.mean(X, axis=0)
    return X - self.mean
```

This function calculates the average (mean) of each column (feature) and subtracts that mean from the entire dataset. This makes sure all features are centered around zero, which is important for PCA to work properly. The centered data, where each feature has a mean of 0 is returned.

```
def compute_key_directions(self, X_centered):
    _, s, Vt = np.linalg.svd(X_centered, full_matrices=False)
    self.basis = Vt
    total_variance = np.sum(s**2)
    self.variance_explained = (s**2) / total_variance if
total_variance > 0 else np.zeros_like(s)
```

This uses a math technique called **SVD** (Singular Value Decomposition) to break the data into parts that tell us:

- The main directions in which the data varies (self.basis),
- And how much each direction matters (self.variance_explained).

Returns Nothing directly, but it saves the basis and variance info for use in the next steps.

```
def reduce_dimensions(self, X_centered, k):
    return X_centered @ self.basis[:k].T
```

This reduces the data from its original number of features to just k features (or directions). It multiplies the centered data with the first k directions found earlier. Returns the new version of the data with fewer features but still keeping the most important patterns.

```
def reconstruct(self, X_reduced):
    if X_reduced.size == 0:
        return np.empty_like(self.mean)
    return X_reduced @ self.basis[:X_reduced.shape[1]] +
self.mean
```

If the reduced data is empty (means all original data was constant), it just returns a placeholder.

Otherwise, it reverses the dimension reduction — bringing the data back to its original shape using the top k directions and adding the mean back in. returns a recreated (approximate) version of the original dataset.

```
def evaluate_error(self, X_original, X_reconstructed):
    return np.linalg.norm(X_original - X_reconstructed)
```

This checks how different the reconstructed data is from the original data. It uses something called the **Frobenius norm**, which is like measuring the total distance between all original and reconstructed values. Returns a number — smaller means better reconstruction, bigger means more info was lost.