Singular Value Decomposition (SVD) (VIMP)

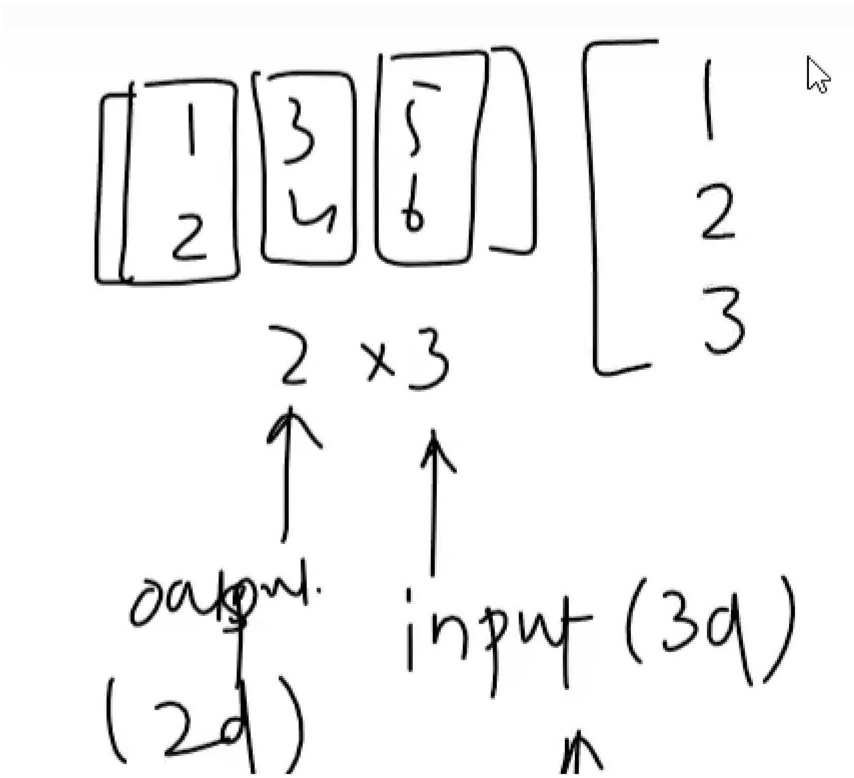
`from sklearn.decomposition import TruncatedSVD`

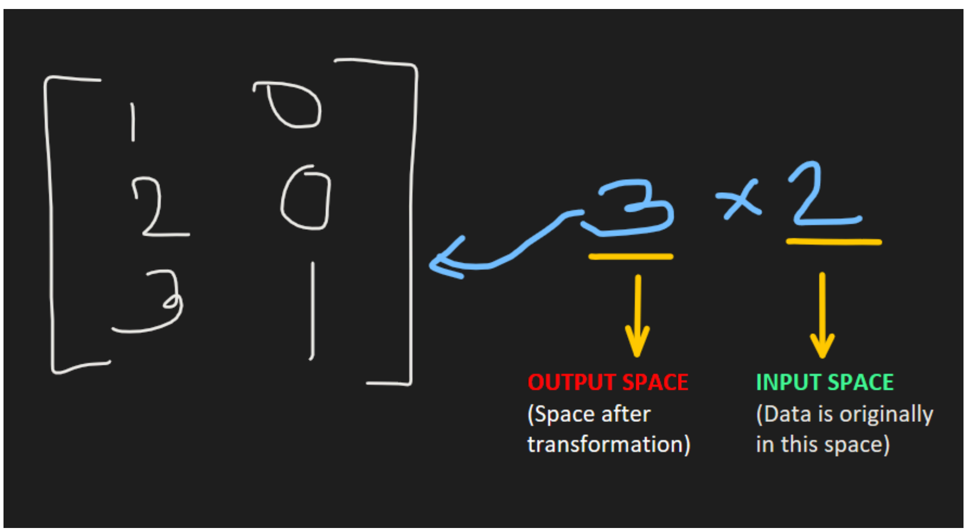
`svd = TruncatedSVD(n_components=2)`

- **SVD**: A matrix factorization technique that decomposes a matrix into three simpler matrices.
- **Purpose**: Widely used in dimensionality reduction, data compression, and noise reduction.
- **Key Applications**: Principal Component Analysis (PCA), image compression, recommendation systems

Non-square Matrix Transformation

- 2D→ 3D



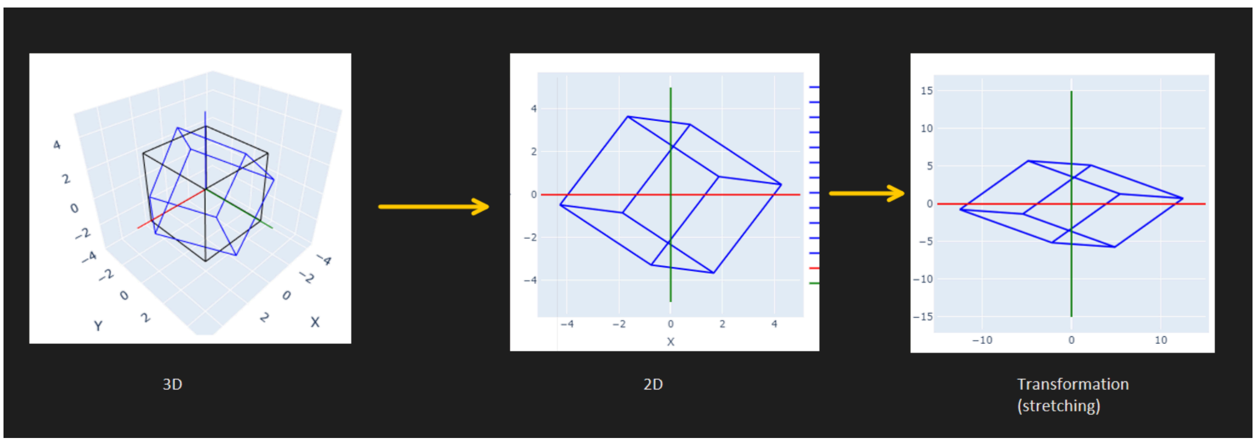


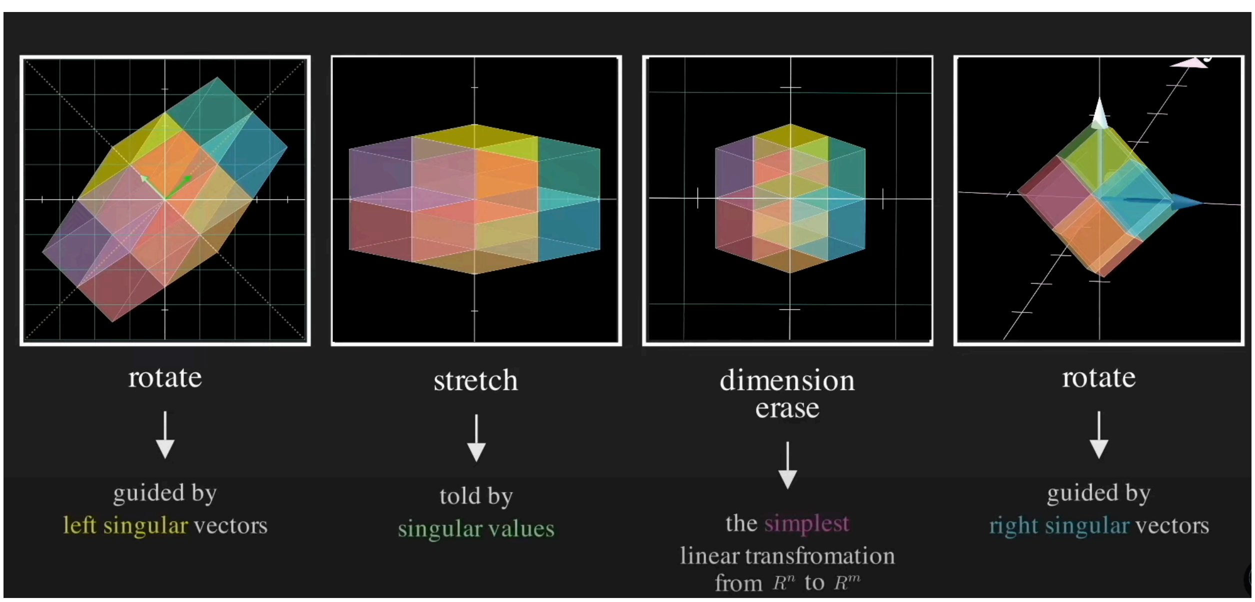
Rectangular Diagonal Matrix

- A matrix that would be diagonal if it were square, but instead is rectangular due to extra rows or columns of zeros.

$$\begin{bmatrix} a & 0 & 0 \\ 0 & b & 0 \end{bmatrix}$$

- If we remove zeros, it will be diagonal matrix
- It applies 2 transformation at same time
    - 3D → 2D
    - Stretching (or some other transformation)





SVD

Mathematical Formulation

Given a matrix A of size m×n, SVD decomposes it into:

$$A = U\Sigma V^T$$

- U: Left singular vectors (orthogonal matrix of size $m \times m$).
- Σ: Diagonal matrix of singular values (size $m \times n$).
- V^T: Right singular vectors (orthogonal matrix of size $n \times n$).

- A is the original matrix (of size $m \times n$).
- U is an $m \times m$ orthogonal matrix (its columns are the left singular vectors of A).
- Σ is an $m \times n$ diagonal matrix (its diagonal elements are the singular values of A).
- V^T is the transpose of an $n \times n$ orthogonal matrix V (its rows are the right singular vectors of A).

Key Components

Singular Values (Σ):

- Represent the **strength (magnitude)** of each component.
- Larger singular values correspond to more important components.

Left Singular Vectors (U):

- Represent the **directions** in the **row** space of A.

Right Singular Vectors (*V*):

- Represent the **directions** in the **column** space of A.

Steps in SVD

1. **Compute A^TA and AA^T**:
    - Eigenvalues and eigenvectors of these matrices are used to derive UU and VV.
2. **Decompose *A***:
    - Factorize A into U, Σ, and V^T.
3. **Truncate Σ**:
    - Keep only the top k singular values for dimensionality reduction.

$$\underset{(m \times n)}{A} = \underset{(m \times m)}{U}\;\underset{(m \times n)}{\Sigma}\;\underset{(n \times n)}{V^T}$$

- **You can convert any matrix into symmetric matrix by multiplying it with its inverse.**

U is eigenvector of AA^T

V is **eigenvector of A^TA**

$$\Sigma \rightarrow \sqrt{a^2}, \sqrt{b^2}$$

💡 **If we directly apply A, multiple steps are applied at the same time. SVD breaks down these steps.**

Transformation of any matrix A can be divided into 4 parts:

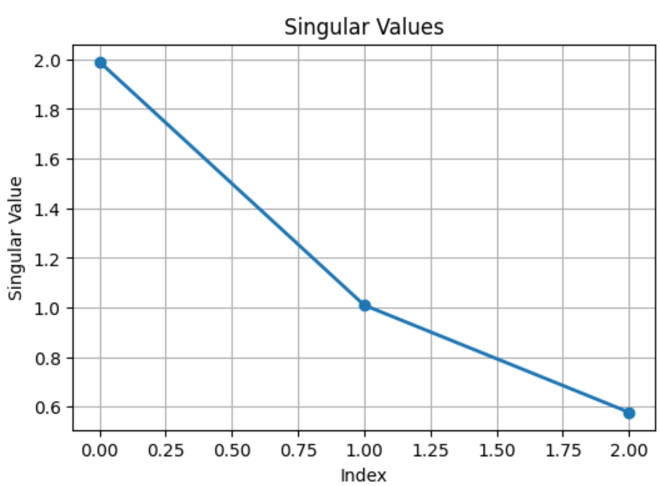
1. Counter-clockwise rotation in the input space (V^T)
2. Dimensionality reduction/increment (Σ_1)
3. Stretching (Σ_2)
4. Clockwise rotation in output (U)

Python Code:

```
import numpy as np
import matplotlib.pyplot as plt

# Create a random matrix A (e.g., 5×3 matrix)
np.random.seed(42)
A = np.random.rand(5, 3)
print("Matrix A:\n", A)

# Perform SVD on matrix A
U, s, VT = np.linalg.svd(A, full_matrices=False)

# Create the diagonal matrix of singular values
```

```
Sigma = np.diag(s)

print("\nMatrix U:\n", U)
print("\nSingular Values:\n", s)
print("\nMatrix V^T:\n", VT)

# Verify that A = U * Sigma * V^T
A_approx = U @ Sigma @ VT
print("\nReconstructed Matrix A:\n", A_approx)

# Plot singular values to see their distribution
plt.figure(figsize=(6,4))
plt.plot(s, 'o-', linewidth=2)
plt.title('Singular Values')
plt.xlabel('Index')
plt.ylabel('Singular Value')
plt.grid(True)
plt.show()
```

```
Matrix A:
 [[0.37454012 0.95071431 0.73199394]
 [0.59865848 0.15601864 0.15599452]
 [0.05808361 0.86617615 0.60111501]
 [0.70807258 0.02058449 0.96990985]
 [0.83244264 0.21233911 0.18182497]]

Matrix U:
 [[-0.5991048  -0.38620771 -0.12988737]
 [-0.25170251  0.32375656 -0.38389036]
 [-0.4495347  -0.55516825  0.01152904]
 [-0.51180949  0.4814656   0.71001691]
 [-0.33717783  0.45387706 -0.57576083]]

Singular Values:
 [1.99063285 1.0096001  0.57767497]

Matrix V^T:
 [[-0.52458829 -0.54271957 -0.65594405]
 [ 0.72866708 -0.6846751  -0.01625695]
 [-0.44028559 -0.48649304  0.75463443]]

Reconstructed Matrix A:
 [[0.37454012 0.95071431 0.73199394]
 [0.59865848 0.15601864 0.15599452]
 [0.05808361 0.86617615 0.60111501]
 [0.70807258 0.02058449 0.96990985]
 [0.83244264 0.21233911 0.18182497]]
```



```
import numpy as np

# Example matrix
A = np.array([
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
])

# Perform SVD
U, S, VT = np.linalg.svd(A)

print("U (Left Singular Vectors):")
print(U)
print("\nS (Singular Values):")
print(S)
print("\nVT (Right Singular Vectors):")
print(VT)
```

6. Example Output

U (Left Singular Vectors):

```
[[-0.21483724  0.88723069  0.40824829]
 [-0.52058739  0.24964395 -0.81649658]
 [-0.82633754 -0.38794278  0.40824829]]
```

S (Singular Values):

```
[1.68481034e+01 1.06836951e+00 4.41842475e-16]
```

VT (Right Singular Vectors):

```
[[-0.47967118 -0.57236779 -0.66506441]
 [-0.77669099 -0.07568647  0.62531805]
 [ 0.40824829 -0.81649658  0.40824829]]
```

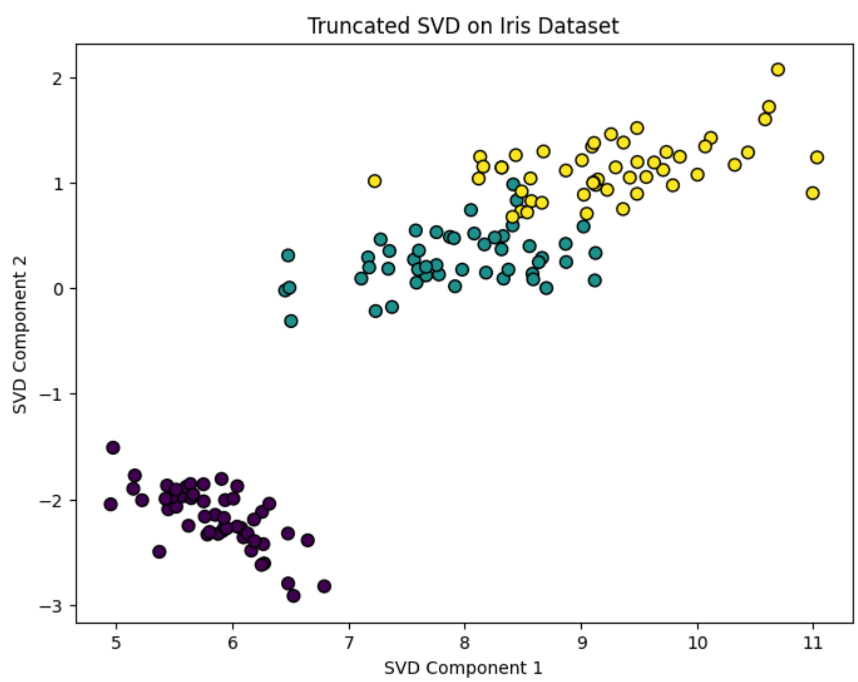
SVD in PCA

- SVD is faster
- You can calculate PC without covariance matrix & its eigen decomposition.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.decomposition import TruncatedSVD

# Load the Iris dataset
iris = load_iris()
X = iris.data  # shape (150, 4)
y = iris.target

# Apply SVD via TruncatedSVD to reduce dimensions to 2
svd = TruncatedSVD(n_components=2, random_state=42)
```

```
X_svd = svd.fit_transform(X)

# Print explained variance ratio and singular values
print("Explained Variance Ratio:", svd.explained_variance_ratio_)
print("Singular Values:", svd.singular_values_)

# Visualize the 2D SVD result
plt.figure(figsize=(8,6))
plt.scatter(X_svd[:, 0], X_svd[:, 1], c=y, cmap='viridis', edgecolor='k', s=50)
plt.xlabel("SVD Component 1")
plt.ylabel("SVD Component 2")
plt.title("Truncated SVD on Iris Dataset")
plt.show()
```

Explained Variance Ratio: [0.52875361 0.44845576]
Singular Values: [95.95991387 17.76103366]



- The

`c=y` argument in `plt.scatter()` maps these labels to colors using a colormap (`cmap='viridis'`).

- **`X_svd[:, 0]`**: The first component (x-axis) of the reduced data.
- **`X_svd[:, 1]`**: The second component (y-axis) of the reduced data.
- **`c=y`**: Colors the points based on the true labels (`y`).