Linear Algebra review

2025. 8

References:
- Steven Skiena, The Data Science Design Manual, Springer, 2017
- Zico Kolter, CMU-388/688 Practical Data Science: Matrices, vectors, and linear algebra (review summary), 2018
- Internet sites, ChatGPT, Gemini

Contents

왜 선형대수가 중요한가?

- 데이터 과학은 선형대수 위에서 동작한다.
  - 데이터는 행렬(Matrix) 이다.
  - 우리가 다루는 대부분의 데이터(표, 이미지, 텍스트)는 결국 행렬 형태로 표현된다.
  - 행(Row): 개별 샘플, 관측치 (e.g., 고객, 상품)
  - 열(Column): 각 샘플의 특성(feature) e.g., 나이, 가격)
- 모든 머신러닝 연산은 행렬 연산이다.
- 알고리즘은 이 행렬을 변환하고, 압축하고, 새로운 정보를 추출하는 과정이다.
- 즉, 선형대수는 데이터를 이해하고 조작하는 언어이다.

데이터의 표현: 벡터와 행렬

- **벡터 (Vector)**
  - 값의 1차원 배열, 주로 하나의 데이터 샘플(관측치)을 표현한다.
  - 기본적으로 열벡터(Column Vector) 를 사용한다.
- **행렬 (Matrix)**
  - 값의 2차원 배열, 전체 데이터셋을 표현함.
  - m개의 샘플(행)과 n개의 특성(열)으로 구성된 m x n 행렬로 표현 가능

Matrices and Linear Algebra

- The most critical part of your data science project is reducing all the information you can find into one or more data matrices, ideally as large as possible.
  - Rows: examples, samples, or indices
  - Columns: distinct features or attributes
- Linear algebra: mathematics of matrices
  - Many machine learning algorithms are best understood through linear algebra

Matrices and Vectors

- A vector is a 1D array of values
  - By default, we use column vectors.

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

- A matrix is a 2D array of values
  - "Higher dimensional matrices" are called tensors.

$$A = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & \cdots & A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m1} & A_{m2} & \cdots & A_{mn} \end{bmatrix}$$

We use A_{ij} to denote the entry in row i and column j

Row and column ordering

- Matrices can be laid out in memory by row or by column

$$A = \begin{bmatrix} 100 & 80 \\ 60 & 80 \\ 100 & 100 \end{bmatrix}$$

  - Row major ordering: 100, 80, 60, 80, 100, 100
  - Column major ordering: 100, 60, 100, 80, 80, 100
- Row major ordering is default for C 2D arrays (and default for Numpy), column major is default for FORTRAN (since a lot of numerical methods are written in FORTRAN, also the standard for most numerical code)

What Can n*m Matrices Represent?

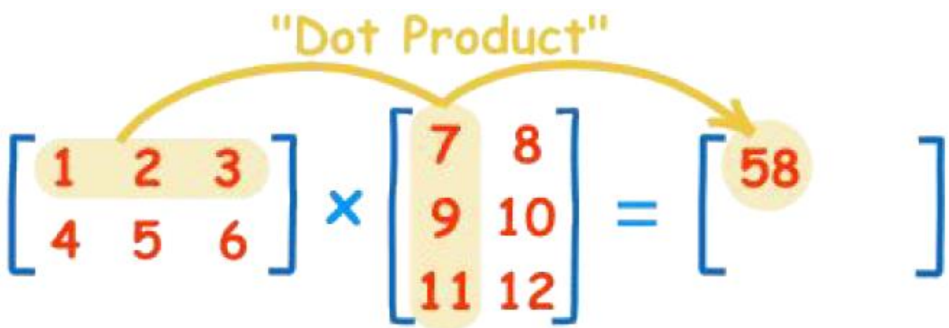
- Data: rows are objects, columns features.
- Geometric point sets: rows are points, columns are dimensions
- Systems of Equations: rows are equations, columns are coefficients for each variable.
- Graphs/Networks: M[i,j] denotes the number of edges from vertex i to vertex j.
- Vectors: any row, column or d*1 matrix
- Images: pixel (x,y)

행렬 곱셈의 의미

- **단순 계산을 넘어선 데이터 변환(Transformation)**
  - 행렬 곱셈은 데이터 포인트(벡터)를 다른 공간으로 이동시키거나 모양을 바꾸는 과정
- 기하학적 의미
  - 회전 (Rotation): 특정 행렬을 곱해 모든 데이터 포인트를 일관되게 회전시킬 수 있음.
  - 확대/축소(Scaling): 특정 축 방향으로 데이터를 늘리거나 줄임 (특성 중요도 조절)
  - 차원 변경 (Projection): n 차원 데이터를 k 차원 공간으로 사영시켜 차원을 축소한다.

Matrix multiplication/ Dot products

- The product A*B is defined by: $$C_{i,j} = \sum_{i=1}^{k} A_{i,k} \cdot B_{k,j}$$
- A*B must share inner dimensions to multiply.
- Each element of the product matrix is a dot product of row/column vectors.



- It is associative, but not commutative.
- Multiplication by the identity commutes: $IA = AI = A$

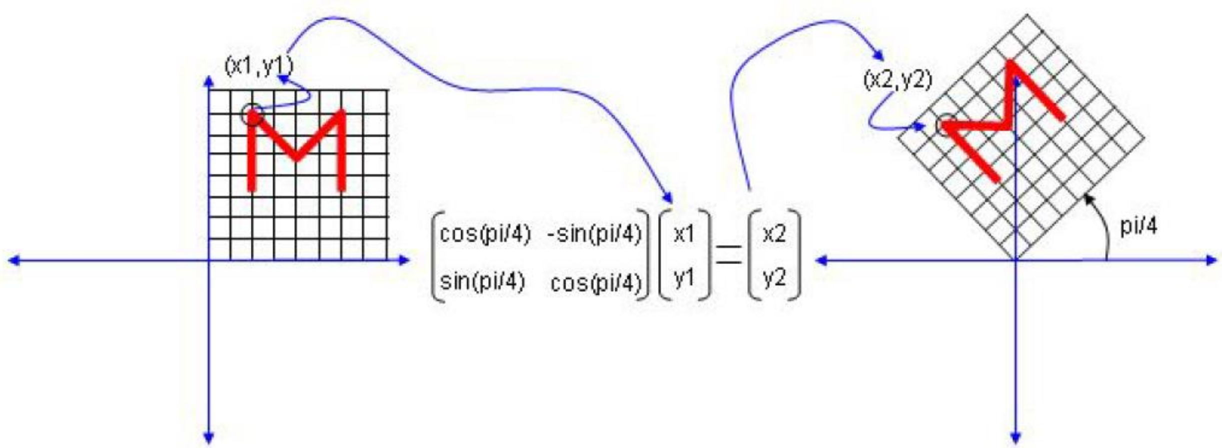
Interpreting Matrix Multiplication

- Multiplication by permutation matrices rearrange rows/columns:

$$P=\begin{pmatrix}0&0&1&0\\1&0&0&0\\0&0&0&1\\0&1&0&0\end{pmatrix}\quad M=\begin{pmatrix}m_{11}&m_{12}&m_{13}&m_{14}\\m_{21}&m_{22}&m_{23}&m_{24}\\m_{31}&m_{32}&m_{33}&m_{34}\\m_{41}&m_{42}&m_{43}&m_{44}\end{pmatrix}$$

$$PM=\begin{pmatrix}m_{31}&m_{32}&m_{33}&m_{34}\\m_{11}&m_{12}&m_{13}&m_{14}\\m_{41}&m_{42}&m_{43}&m_{44}\\m_{21}&m_{22}&m_{23}&m_{24}\end{pmatrix}$$

- Rotating point in space:



Matrix Inversion

- A^{-1} is the multiplicative inverse of A, if $A A^{-1} = I$ where I is the identity matrix.

$$\mathbf{A}^{-1} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}.$$

- If matrix A has an inverse, it can be computed by solving a linear system using Gaussian elimination.

$$[\mathbf{A} \quad \mathbf{I}] = \begin{bmatrix} 0 & 1 & 2 & 1 & 0 & 0 \\ 1 & 0 & 3 & 0 & 1 & 0 \\ 4 & -3 & 8 & 0 & 0 & 1 \end{bmatrix} \sim \begin{bmatrix} 1 & 0 & 3 & 0 & 1 & 0 \\ 0 & 1 & 2 & 1 & 0 & 0 \\ 0 & -3 & -4 & 0 & -4 & 1 \end{bmatrix}$$

$$\sim \begin{bmatrix} 1 & 0 & 3 & 0 & 1 & 0 \\ 0 & 1 & 2 & 1 & 0 & 0 \\ 0 & 0 & 2 & 3 & -4 & 1 \end{bmatrix} \sim \begin{bmatrix} 1 & 0 & 0 & -9/2 & 7 & -3/2 \\ 0 & 1 & 0 & -2 & 4 & -1 \\ 0 & 0 & 1 & 3/2 & -2 & 1/2 \end{bmatrix}$$

$$\mathbf{A}^{-1} = \begin{bmatrix} -9/2 & 7 & -3/2 \\ -2 & 4 & -1 \\ 3/2 & -2 & 1/2 \end{bmatrix}$$

Matrix Inversion and Linear Systems

- Multiplying both sides of Ax = b by the inverse of A yields: $(A^{-1}A)x = A^{-1}b$, or $x = A^{-1}b$
- Thus solving linear equations is equivalent to matrix inversion.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 5 & 1 \\ 2 & 3 & 8 \end{bmatrix}^{-1} \begin{bmatrix} 10 \\ 8 \\ 3 \end{bmatrix} = \frac{1}{25} \begin{bmatrix} -232 \\ 129 \\ 19 \end{bmatrix} = \begin{bmatrix} -9.28 \\ 5.16 \\ 0.76 \end{bmatrix}$$

- The inverse makes it cheap to evaluate many b vectors. However, Gaussian elimination is more numerically stable than inversion.

Some definitions/properties

Transpose of matrix multiplication, $A \in \mathbb{R}^{m \times n}$, $B \in \mathbb{R}^{n \times p}$

$$(AB)^T = B^T A^T$$

Inverse of product, $A \in \mathbb{R}^{n \times n}$, $B \in \mathbb{R}^{n \times n}$ *both square and invertible*

$$(AB)^{-1} = B^{-1} A^{-1}$$

Inner product: for $x, y \in \mathbb{R}^n$, special case of matrix multiplication

$$x^T y \in \mathbb{R} = \sum_{i=1}^{n} x_i y_i$$

Vector norms: for $x \in \mathbb{R}^n$, we use $\|x\|_2$ to denote Euclidean norm

$$\|x\|_2 = (x^T x)^{\frac{1}{2}}$$

Matrix Rank

- Systems are underdetermined if rows can be expressed as linear combinations of other rows.
- The rank of a matrix is a measure of the number of linearly independent rows.
- An $n*n$ matrix should be rank n for all operations to be properly defined on it.
- Some rows of the image may not be linearly independent, so it is not full rank. Adding small amounts of random noise increases rank without serious image distortion.

데이터 구조 파악: 고유값 분해(EVD)

- 정의: $Av = \lambda v$
  - 행렬 A로 변환 시, 방향은 변하지 않고(v) 크기만(λ) 변하는 특별한 벡터와 스칼라 값
- 구성 요소의 의미
  - 고유벡터 (Eigenvector, v): 변환의 주축(Principal Axis). 데이터 분산이 가장 큰 방향을 나타낸다.
  - 고유값 (Eigenvalue, λ): 해당 축 방향으로의 변환 크기(Scale). 축의 중요도를 의미한다.
- 조건: 고유값 분해는 정방행렬(n x n square matrix)에만 적용 가능함.

Eigenvalues and Eigenvectors

- Multiplying a vector x by a matrix A can have the same effect as multiplying it by a scalar λ.
  - Av = λv (x: eigenvector, λ: eigenvalue)
  - Thus the eigenvalue-eigenvector pair (λ, v) must encode a lot of information about matrix A!

$$\begin{bmatrix} -5 & 2 \\ 2 & -2 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 2 \end{bmatrix} = -1 \begin{bmatrix} 1 \\ 2 \end{bmatrix} \qquad \begin{bmatrix} -5 & 2 \\ 2 & -2 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ -1 \end{bmatrix} = -6 \begin{bmatrix} 2 \\ -1 \end{bmatrix}$$

- The n distinct eigenvalues of a rank n matrix can be found by factoring its characteristic equation.

$$\det(A - \lambda I) = \begin{vmatrix} -\lambda & 1 & 1 \\ 1 & -\lambda & 1 \\ 1 & 1 & -\lambda \end{vmatrix} \qquad = -\lambda^3 + 3\lambda + 2$$

$$= (\lambda - 2)(\lambda + 1)^2 \qquad \lambda_1 = 2, \ \lambda_{2,3} = -1$$

Computing Eigenvectors

- The vector associated with a given eigenvalue can be computed by solving a linear system:
  - (ex)

$$\mathbf{A} \cdot \mathbf{v}_1 = \lambda_1 \cdot \mathbf{v}_1$$

$$(\mathbf{A} - \lambda_1) \cdot \mathbf{v}_1 = 0$$

-λ1 *v1,1 + v1,2 = 0

-2 * v1,1 + (-3- λ1)*v1,2 = 0

$$\begin{bmatrix} -\lambda_1 & 1 \\ -2 & -3-\lambda_1 \end{bmatrix} \cdot \mathbf{v}_1 = 0$$

- Another approach uses v’ = (A * v)/ λ to compute approximations to v until it converges.

Eigenvalue Decomposition

$$\mathbf{A}\mathbf{v} = \lambda \mathbf{v}$$
$$\mathbf{A}\mathbf{Q} = \mathbf{Q}\mathbf{\Lambda}$$
$$\mathbf{A} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^{-1}$$

$$\begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}\begin{bmatrix} -1 \\ 2 \end{bmatrix} = -2\begin{bmatrix} -1 \\ 2 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}\begin{bmatrix} -1 \\ 1 \end{bmatrix} = -1\begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}\begin{bmatrix} -1 & -1 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} -1 & -1 \\ 2 & 1 \end{bmatrix}\begin{bmatrix} -2 & 0 \\ 0 & -1 \end{bmatrix}$$

[고유값 분해 (eigenvalue decomposition)]

$$A = PDP^{-1} = PDP^{T}$$
$$= \begin{pmatrix} v_1 & v_2 & \cdots & v_n \end{pmatrix}\begin{pmatrix} \lambda_1 & 0 & 0 & 0 \\ 0 & \lambda_2 & 0 & 0 \\ 0 & 0 & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{pmatrix}\begin{pmatrix} v_1 & v_2 & \cdots & v_n \end{pmatrix}^{-1}$$

A : n * n 정방행렬 (n by n square matrix)

λ_i : 고유값 (eigenvalue)

v_i : 고유값 λ_i 에 대응하는 고유벡터 (eigenvector)

P : 고유벡터 v_i 로 이루어진 행렬

D : 고유값 λ_i 로 이루어진 대각행렬 (*diagonal matrix*)

[R 분석과 프로그래밍] http://rfriend.tistory.com

- Larger eigenvalues correspond to more important vector products.

EVD 응용: 주성분 분석 (PCA)

- **PCA (Principal Component Analysis)**
  - 고차원 데이터의 분산을 가장 잘 설명하는 새로운 축(주성분)을 찾아 차원을 축소하는 기법
- **EVD와 PCA의 관계**
  - 1단계: 데이터의 **공분산 행렬(Covariance Matrix)** 계산 (특성 간의 분산 구조 파악).
  - 2단계: 공분산 행렬에 대해 **고유값 분해(EVD)** 수행
  - 결과:
    - 고유벡터 -> 주성분 (데이터 분산이 가장 큰 방향)
    - 고유값 -> 설명된 분산 (해당 주성분의 중요도)

데이터 구조 파악: 특이값 분해(SVD)

- 정의: $A = U\Sigma V^T$
  - 모든 행렬(m x n)을 분해할 수 있는 가장 일반적이고 강력한 기법.
- 구성 요소
  - U (Left Singular Vectors): 행(Row) 공간의 새로운 직교 축. AA^T 의 고유벡터.
  - V (Right Singular Vectors): 열(Column) 공간의 새로운 직교 축. A^TA의 고유벡터.
  - Σ (Singular Values): 대각 행렬. 대각성분(특이값)은 각 축의 중요도를 나타내며, A^TA의 고유값의 제곱근과 같다.

Singular Value Decomposition (SVD)

특이값분해(singular value decomposition)

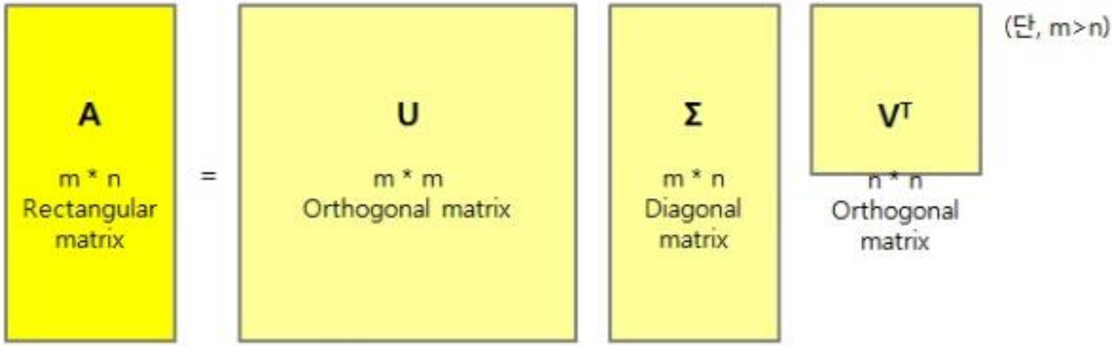
$$A = U\Sigma V^T$$

A : m*n 직사각행렬(m by n rectangular matrix)

U : A의 left singular vector로 이루어진 m*m 직교행렬(orthogonal matrix)

Σ : 주대각성분이 $\sqrt{\lambda_i}$ 로 이루어진 $m*n$ 직사각대각행렬(diagonal matrix)

V : A의 right singualr vector로 이루어진 n*n 직교행렬(orthogonal matrix)



[R 분석과 프로그래밍] http://rfriend.tistory.com

- AA^T 의 eigenvalue 의 square root – singular value
- Retaining only the rows/column with large weights permits us to compress m features with relatively little loss.

Singular Value Decomposition

$$A = U\Sigma V^{\mathrm{T}}$$

$$A^{\mathrm{T}}A = (U\Sigma V^{\mathrm{T}})^{\mathrm{T}}(U\Sigma V^{\mathrm{T}}) = V\Sigma^{\mathrm{T}}U^{\mathrm{T}}U\Sigma V^{\mathrm{T}} = V\boxed{\Sigma^{\mathrm{T}}\Sigma}V^{\mathrm{T}}$$

$$A\,A^{\mathrm{T}} = (U\Sigma V^{\mathrm{T}})(U\Sigma V^{\mathrm{T}})^{\mathrm{T}} = U\Sigma V^{\mathrm{T}}V\Sigma^{\mathrm{T}}U^{\mathrm{T}} = U\boxed{\Sigma\Sigma^{\mathrm{T}}}U^{\mathrm{T}}$$

- The right-hand sides of the relations describe the eigenvalue decompositions of the left-hand sides.
  - The columns of V are eigenvectors of $A^{\mathrm{T}}A$.
  - The columns of U are eigenvectors of AA^{T}.
  - The non-zero elements of Σ are the square roots of the non-zero eigenvalues of $A^{\mathrm{T}}A$ or AA^{T}.
- Example:

$$A = \begin{pmatrix} 3 & 6 \\ 2 & 3 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \qquad A = U\Sigma V^T = \underbrace{\begin{pmatrix} 0.881 & -0.471 & 0 & 0 \\ 0.471 & 0.881 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}}_{U} \underbrace{\begin{pmatrix} 7.605 & 0 \\ 0 & 0.394 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}}_{\Sigma} \underbrace{\begin{pmatrix} 0.471 & -0.881 \\ 0.881 & 0.471 \end{pmatrix}^{\mathrm{T}}}_{V^T}$$

AA^T 의 고유벡터
(eigenvectors of AA^T)

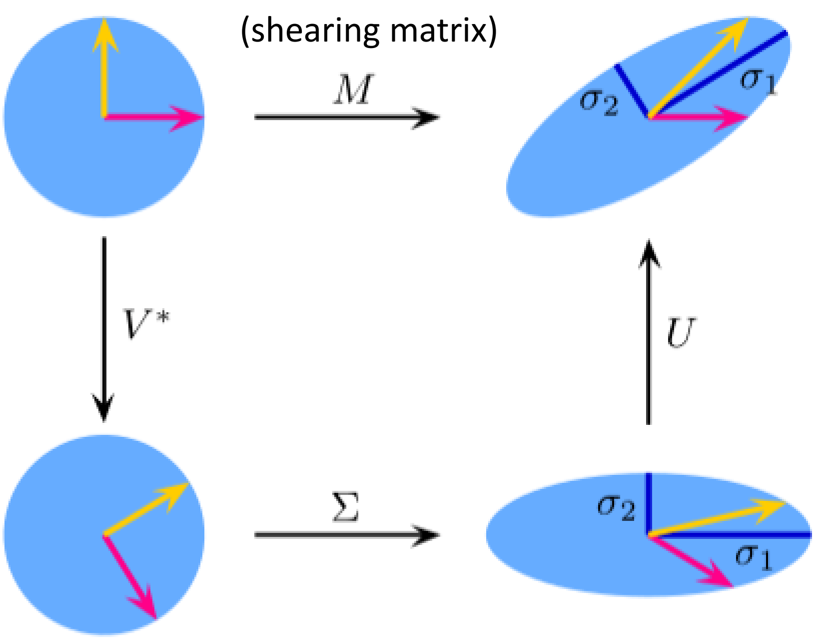
$\sigma_i = \sqrt{\lambda_i}$
($\lambda_1 \geq \lambda_2 \ldots \geq 0$ 이며,
대각원소이외 모두 0)

$A^T A$ 의 고유벡터
(eigenvectors of $A^T A$)

[R 분석과 프로그래밍] http://rfriend.tistory.com

Singular Value Decomposition

- Geometric interpretation



$$M = U \cdot \Sigma \cdot V^*$$

Illustration of the singular value decomposition **UΣV*** of a real 2×2 matrix **M**.

Top: The action of **M**, indicated by its effect on the unit disc *D* and the two canonical unit vectors e_1 and e_2.

Left: The action of **V***, a rotation, on *D*, e_1, and e_2.

Bottom: The action of **Σ**, a scaling by the singular values σ_1 horizontally and σ_2 vertically.

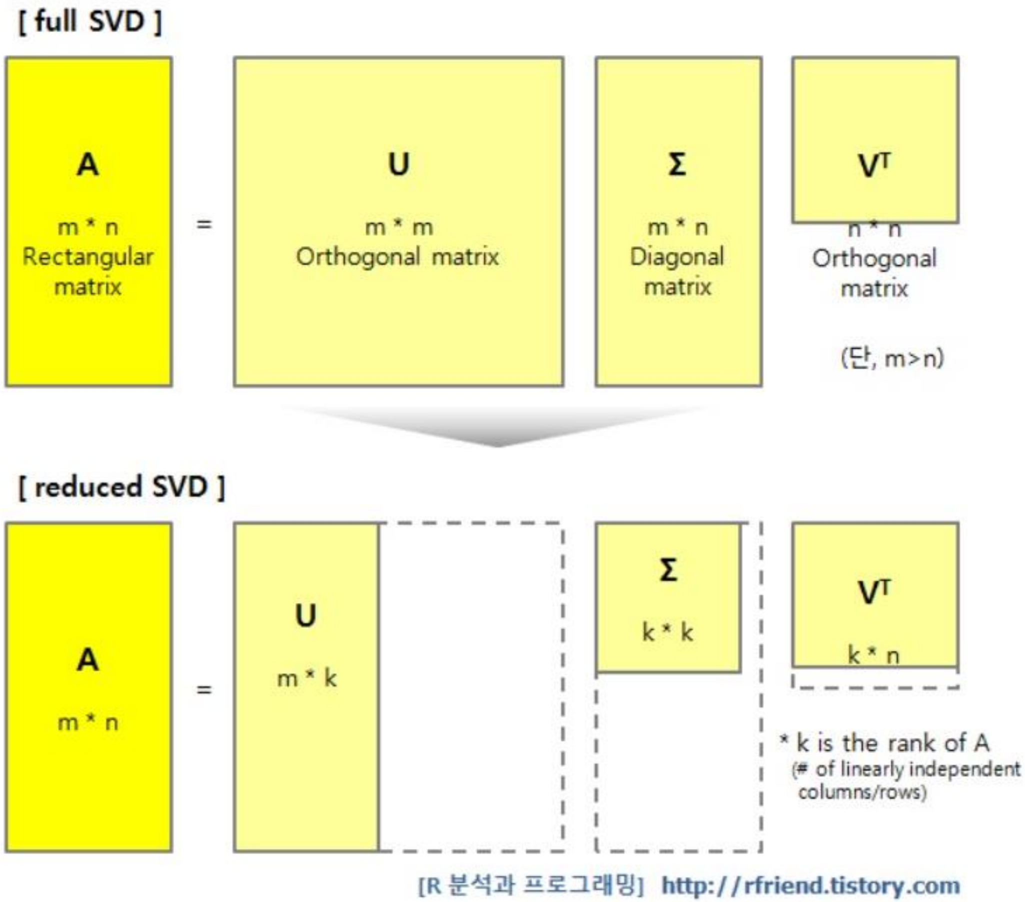
Right: The action of **U**, another rotation.

https://en.wikipedia.org/wiki/Singular_value_decomposition

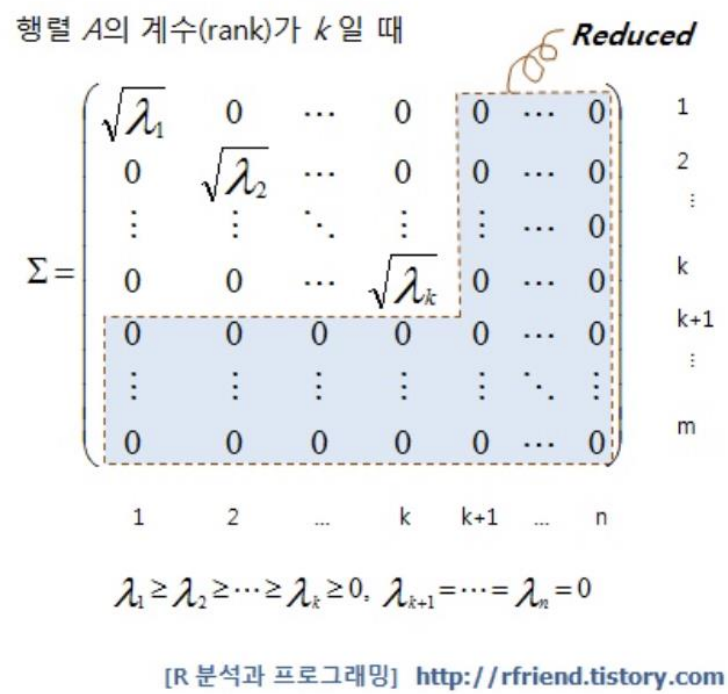
SVD 응용: 이미지 압축 및 차원축소

- 원리: 중요도가 높은(값이 큰) 상위 k개의 특이값과 해당 벡터들만 사용하여 원본 행렬을 근사함.
  - 이를 Reduced SVD 라고 함.
  - 원본 데이터의 손실을 최소화하면서 차원을 크게 줄일 수 있음.
- 활용 예시: 이미지 압축
  - 이미지(행렬)를 SVD로 분해한 후, 일부 특이값만으로 이미지를 재구성함.
  - 적은 수의 특이값으로도 원본과 유사한 이미지를 복원 가능.

Reduced SVD



[full SVD]
A
m * n
Rectangular matrix
=
U
m * m
Orthogonal matrix
Σ
m * n
Diagonal matrix
V^T
n * n
Orthogonal matrix
(단, m>n)
[reduced SVD]
A
m * n
=
U
m * k
Σ
k * k
V^T
k * n
* k is the rank of A
(# of linearly independent columns/rows)
[R 분석과 프로그래밍] http://rfriend.tistory.com



행렬 A의 계수(rank)가 k 일 때
Reduced
Σ =
1
2
k
k+1
m
1
2
...
k
k+1
...
n
$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_k \geq 0, \ \lambda_{k+1} = \cdots = \lambda_n = 0$
[R 분석과 프로그래밍] http://rfriend.tistory.com

Reduced SVD

- 데이터 압축: 전송할 데이터 절감 효과 (예)

$$A \approx A_k = U_k \Sigma_k V_k^T$$

여기서 U_k, Σ_k, V_k는 기존 행렬에서 상위 k개에 해당하는 부분만 잘라낸 작은 행렬들

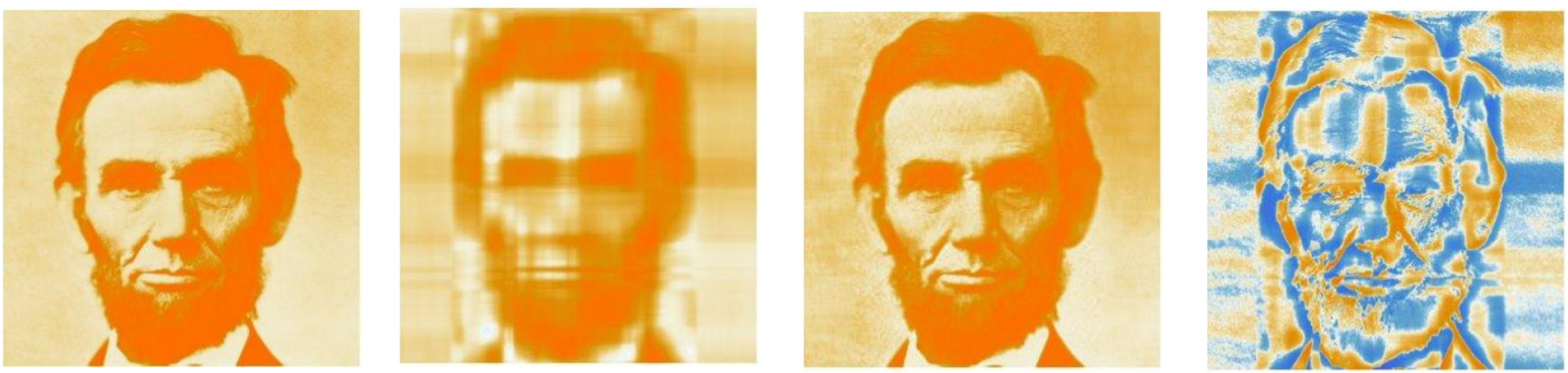
예를 들어, 1000 x 800 픽셀의 이미지 행렬(총 800,000개의 값)이 있다고 가정하자.

- **원본 데이터**: 800,000개의 값을 저장해야 한다.
- **Reduced SVD (k=50)**: 상위 50개의 특이값만 사용해 이미지를 근사하면, 저장할 데이터는 다음과 같다.
  - U_{50} (1000 x 50 행렬) = 50,000개
  - Σ_{50} (50 x 50 대각행렬, 실제론 50개) = 50개
  - V_{50}^T (50 x 800 행렬) = 40,000개
  - **총합**: 50,000 + 50 + 40,000 = **90,050개**

결과적으로 원본 데이터(800,000개)의 약 11% 크기만으로도 원본과 매우 유사한 이미지를 표현할 수 있게 된다. 이렇게 압축된 데이터는 저장, 처리, 전송 등 모든 면에서 훨씬 효율적이다.

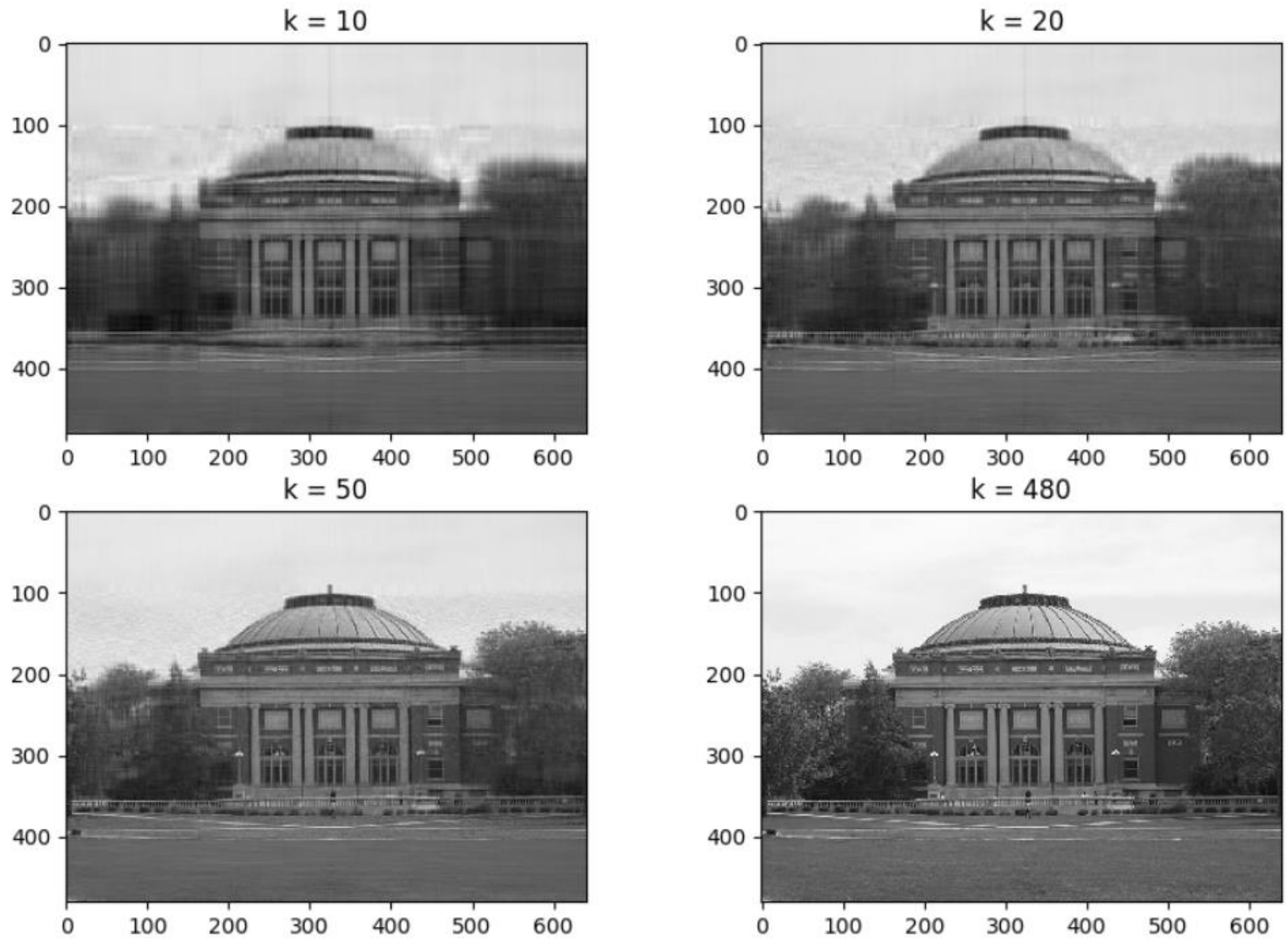
Reconstructing image from SVD

- Lincoln’s face from 5 and 50 singular values, a substantial compression of the original matrix.
  - (a) original (b) k=5 © k=50 (d) error for k=50



Reconstructing image from SVD

- Rank-k approximations of an image



https://courses.grainger.illinois.edu/cs357/fa2021/notes/ref-16-svd.html

SVD 응용: 추천 시스템 및 자연어 처리

- 추천 시스템 (Recommender Systems)
  - 사용자-아이템 행렬을 SVD로 분해.
  - 분해된 행렬들은 사용자의 잠재적 취향(Latent Factor)과 아이템의 잠재적 특성을 나타냄.
  - 이를 통해 사용자가 평가하지 않은 아이템의 평점을 예측하고 추천.
- 자연어 처리 (NLP): 잠재 의미 분석(LSA)
  - 단어-문서 행렬을 SVD로 분해 (단어-주제, 주제-문서 행렬을 얻음).
  - 문서에 잠재된 주제(Topic)를 추출하고, 단어 간의 의미적 유사도를 파악하는 데 사용.

안정적 해법: QR분해와 선형 회귀

- 주요 용도: 선형 회귀(Linear Regression) 모델의 해를 수치적으로 안정적이게 계산.
- 기존 방식의 문제점:
  - 표준 선형 회귀 해법($\beta=(X^TX)^{-1} X^Ty$)은 역행렬 계산을 요구함.
  - 특성 간 상관관계가 높으면 역행렬 계산이 불안정해져 결과의 신뢰도가 하락할 수 있음.
- QR 분해를 통한 해결:
  - 데이터 행렬 A를 직교행렬 Q와 상삼각행렬 R로 분해 (A=QR).
  - 역행렬 계산을 피하면서 더 빠르고 안정적으로 해를 구할 수 있음.

QR Decomposition

- Theorem 6.3.7 (QR-Decomposition)
  - If A is an $m \times n$ matrix with linearly independent column vectors, then A can be factored as

$$A = QR$$

where Q is an $m \times n$ matrix with orthonormal column vectors, and R is an $n \times n$ invertible upper triangular matrix.

Q is orthonormal matrix: $Q^T Q = I$

$$\underset{A}{\begin{bmatrix} 1 & \sqrt{5} \\ 2 & 0 \\ 0 & -\sqrt{5} \end{bmatrix}} = \underset{Q}{\begin{bmatrix} \frac{1}{\sqrt{5}} & \frac{4}{3\sqrt{5}} \\ \frac{2}{\sqrt{5}} & -\frac{2}{3\sqrt{5}} \\ 0 & -\frac{5}{3\sqrt{5}} \end{bmatrix}} \underset{R}{\begin{bmatrix} \sqrt{5} & 1 \\ 0 & 3 \end{bmatrix}}$$

- Application: often used to estimate linear regressions.
  - Ordinally Least Squares (OLS) estimator

$X=QR$

$$\begin{aligned} \beta &= (X^\top X)^{-1} X^\top y \\ &= (R^\top Q^\top Q R)^{-1} R^\top Q^\top y \\ &= (R^\top R)^{-1} R^\top Q^\top y \\ &= R^{-1} (R^\top)^{-1} R^\top Q^\top y \\ &= R^{-1} Q^\top y \end{aligned}$$

$$\hat{y} = X\beta \quad \Rightarrow \quad \beta = (X^\top X)^{-1} X^\top y \quad \Rightarrow$$

Best estimator to minimize MSE

Well-known solution (OLS estimator)

To avoid inversion and reduce computational burden (R is upper triangular)

대용량 데이터 처리: 희소 행렬

- 희소 행렬 (Sparse Matrix)
  - 대부분의 원소가 0으로 채워진 행렬.
  - 예시: 소셜 네트워크 연결망, 전자상거래 구매 내역, 문서 내 단어 등장 빈도.
- 필요성
  - 수많은 0을 모두 저장하는 것은 메모리 및 계산 자원의 심각한 낭비.
- 해결책
  - 0이 아닌 값만 위치 정보와 함께 효율적으로 저장하는 특별한 자료 구조 사용.
  - 예시: COO(Coordinate) 형식, CSC(Compressed Sparse Column) 형식.

Sparse matrices

- Many matrices are *sparse* (contain mostly zero entries, with only a few non-zero entries)
- Examples: matrices formed by real-world graphs, document-word count matrices (more on both of these later)
- Storing all these zeros in a standard matrix format can be a huge waste of computation and memory
- Sparse matrix libraries provide an efficient means for handling these sparse matrices, storing and operating only on non-zero entries

Coordinate format

- There are several different ways of storing sparse matrices, each optimized for different operations
- Coordinate (COO) format: store each entry as a tuple (row-index, col-index, value)

$$A = \begin{bmatrix} 0 & 0 & 3 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 4 & 0 & 1 & 0 \end{bmatrix}$$

$$\begin{aligned} \text{data} &= [2\ 4\ 1\ 3\ 1\ 1] \\ \text{row-indices} &= [1\ 3\ 2\ 0\ 3\ 1] \\ \text{col-indices} &= [0\ 0\ 1\ 2\ 2\ 3] \end{aligned}$$

- A good format for constructing sparse matrices

Compressed sparse column format

- Compressed sparse column (CSC) format
- *Ordering is important (always column-major ordering)*
- Faster for matrix multiplication, easier to access individual columns
- Very bad for modifying a matrix, to add one entry need to shift all data
- Example:

$$A = \begin{pmatrix} 0 & 0 & 3 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 4 & 0 & 1 & 0 \end{pmatrix}$$

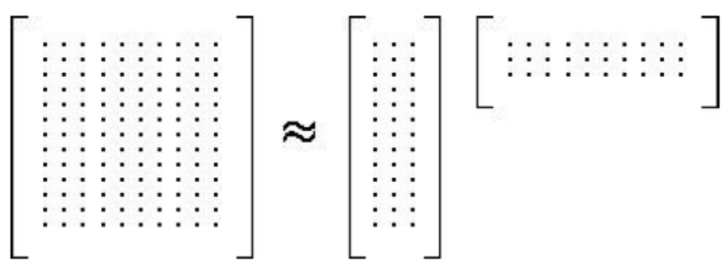
CSC 포맷은 아래 3개의 배열로 표현된다.

- `Data` : `[2, 4, 1, 3, 1, 1]` (1열부터 순서대로 0이 아닌 값을 읽음)
- `Row Indices` : `[1, 3, 2, 0, 3, 1]` (Data 값에 해당하는 행 인덱스)
- `Column Pointers` : `[0, 2, 3, 5, 6]`
  - 0번 열은 `Data` 배열의 0번 인덱스에서 시작한다.
  - 1번 열은 `Data` 배열의 2번 인덱스에서 시작한다.
  - 2번 열은 `Data` 배열의 3번 인덱스에서 시작한다.
  - 3번 열은 `Data` 배열의 5번 인덱스에서 시작한다.
  - 마지막 값(6)은 0이 아닌 원소의 총개수이다.

Matrix Decomposition (Factoring)

Factoring Matrices

- Many important machine learning algorithms can be viewed as factoring a matrix. Suppose *n*m* matrix *A* can be expressed as the product *BC*, i.e. an *n*k* matrix times a *k*m* matrix.
- (ex) factoring Word-Document Matrices
  - If A is a document/word co-occurrence matrix, and A=BC, where B is d*k and C is k*w:
  - B,C are compressed feature vectors for docs and words



LU Decomposition

LU 분해는 하나의 정방행렬 A를 **아래 삼각행렬(Lower triangular matrix, L)**과 **위 삼각행렬(Upper triangular matrix, U)**의 곱으로 나누는 기법이다.

- **핵심 목적**: 컴퓨터가 선형 방정식 시스템($Ax = b$)을 매우 빠르고 효율적으로 풀도록 하는 데 있다. 복잡한 역행렬을 한 번에 구하는 대신, 두 개의 단순한 삼각 시스템을 순서대로 푸는 방식으로 계산을 단순화한다.
- **구성 요소**:
  - **L (아래 삼각행렬)**: 주대각선 위쪽 원소가 모두 0인 행렬.
  - **U (위 삼각행렬)**: 주대각선 아래쪽 원소가 모두 0인 행렬.

- **Example Matrix**:
  The matrix $A = \begin{pmatrix} 2 & 1 \\ 4 & 5 \end{pmatrix}$ can be decomposed as follows:
  $$A = L \cdot U = \begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} 2 & 1 \\ 0 & 3 \end{pmatrix}$$

QR Decomposition

QR 분해는 행렬 A를 **직교 행렬(Orthogonal matrix, Q)**과 **위 삼각행렬(Upper triangular matrix, R)**의 곱으로 나누는 방법이다.

- **핵심 목적**: 선형 회귀 분석에서 **최소 제곱법(Least Squares)** 문제의 해를 수치적으로 매우 안정적이게 찾는 데 주로 사용된다. 데이터의 기하학적 속성(거리, 각도)을 보존하는 '회전' 변환을 통해 문제를 단순화한다.
- **구성 요소**:
  - **Q (직교 행렬)**: 모든 열벡터가 서로 수직이고 크기가 1인 행렬. 이 행렬은 데이터를 회전시키는 역할을 한다. ($Q^T Q = I$)
  - **R (위 삼각행렬)**: LU 분해의 U와 같다.

- **Example Matrix**:

  The matrix $A = \begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 1 & 1 \\ 1 & 0 \end{pmatrix}$ can be decomposed as follows:

  $$A = Q \cdot R = \begin{pmatrix} 1/2 & 1/2 \\ 1/2 & -1/2 \\ 1/2 & 1/2 \\ 1/2 & -1/2 \end{pmatrix} \begin{pmatrix} 2 & 1 \\ 0 & 1 \end{pmatrix}$$

Eigenvalue Decomposition

고유값 분해는 **정방행렬(Square matrix)** A가 가진 고유한 '성질'을 **고유벡터(Eigenvector) 행렬**과 **고유값(Eigenvalue) 대각행렬**로 분해하는 것이다.

$$A = Q\Lambda Q^{-1}$$

- **핵심 목적**: 특정 행렬이 가하는 변환의 **핵심 축(방향)**과 그 **축 방향으로의 변환 크기(힘)**를 알아내는 데 있다. **주성분 분석(PCA)**에서 데이터의 분산이 가장 큰 방향(주성분)을 찾는 데 사용되는 핵심 원리이다.
- **구성 요소**:
  - **Q**: 행렬 A의 고유벡터들을 열로 가지는 행렬.
  - **Λ (람다)**: 행렬 A의 고유값들을 대각 원소로 가지는 대각 행렬.

- **Example Matrix**:
  The matrix $A = \begin{pmatrix} 4 & -2 \\ 1 & 1 \end{pmatrix}$ can be decomposed as follows:
  $$A = Q\Lambda Q^{-1} = \begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 3 & 0 \\ 0 & 2 \end{pmatrix} \begin{pmatrix} 1 & -1 \\ -1 & 2 \end{pmatrix}$$

Singular Value Decomposition

SVD는 **정방행렬뿐만 아니라 모든 직사각 행렬(m x n matrix)**에 대해 적용할 수 있는, 가장 범용적이고 강력한 분해 기법이다.

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

- **핵심 목적**: 데이터에 숨겨진 잠재적인 구조를 파악하는 데 매우 효과적이다. **차원 축소, 데이터 압축, 노이즈 제거, 추천 시스템, 자연어 처리** 등 데이터 과학 전반에서 가장 널리 활용된다. 모든 행렬 변환을 '회전 → 확대/축소 → 다시 회전'이라는 세 단계로 해석할 수 있게 해준다.
- **구성 요소**:
  - **U**: 왼쪽 특이벡터로 구성된 직교 행렬.
  - **Σ (시그마)**: 특이값을 대각 원소로 가지는 직사각 대각 행렬.
  - **V**: 오른쪽 특이벡터로 구성된 직교 행렬.
- **Example Matrix**:
  The matrix $A = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$ can be decomposed as follows:

$$A = U\Sigma V^T = \begin{pmatrix} 0.71 & -0.71 \\ 0.71 & 0.71 \end{pmatrix} \begin{pmatrix} 1.73 & 0 & 0 \\ 0 & 0.58 & 0 \end{pmatrix} \begin{pmatrix} 0.41 & 0.82 & 0.41 \\ 0.71 & 0 & -0.71 \\ -0.58 & 0.58 & -0.58 \end{pmatrix}^T$$