Linear Algebra review

2019. 6

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

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

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

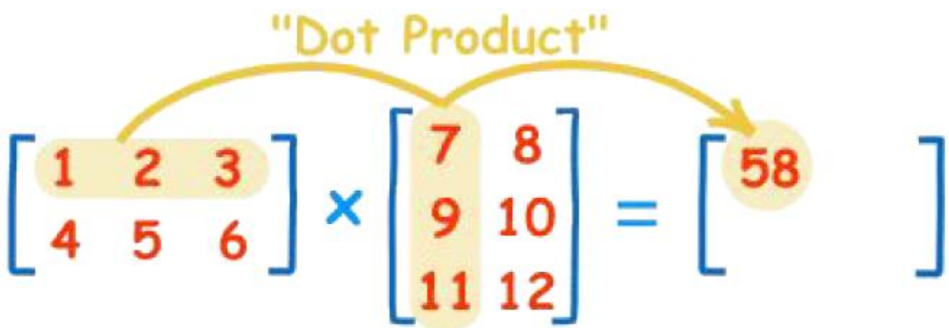
  - Row major ordering: 100, 80, 60, 80, 100, 100
  - Column major ordering: 100, 60, 100, 80, 80, 100
- Row major ordering is default for C 2D arrays (and default for Numpy), column major is default for FORTRAN (since a lot of numerical methods are written in FORTRAN, also the standard for most numerical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

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: $\mathbb{I}A = A\mathbb{I} = A$

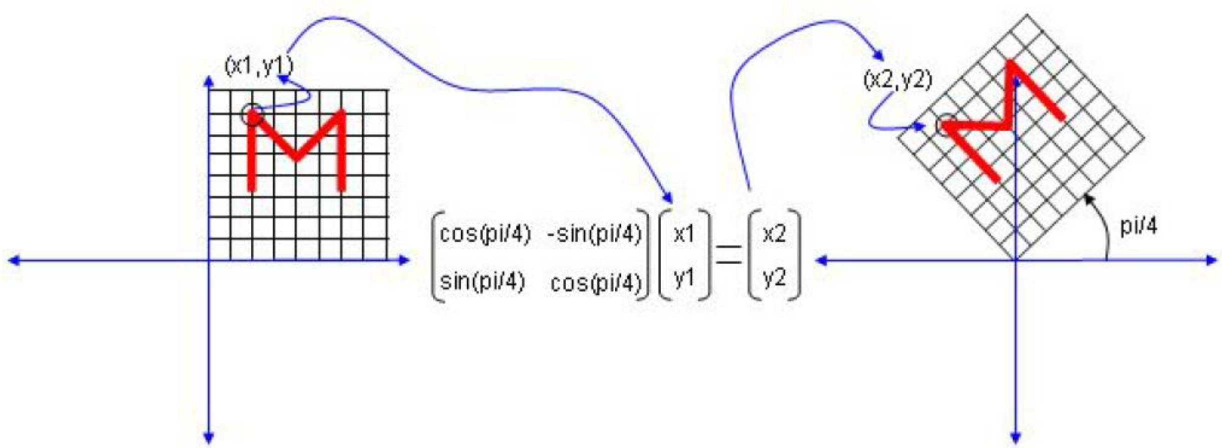
Interpreting Matrix Multiplication

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

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

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

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

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

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

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

Valid linear algebra expressions

Assume $A \in \mathbb{R}^{n\times n}, B \in \mathbb{R}^{n\times m}, C \in \mathbb{R}^{m\times n}, x \in \mathbb{R}^n$ with $m > n$.
Which of the following are valid linear algebra expressions?

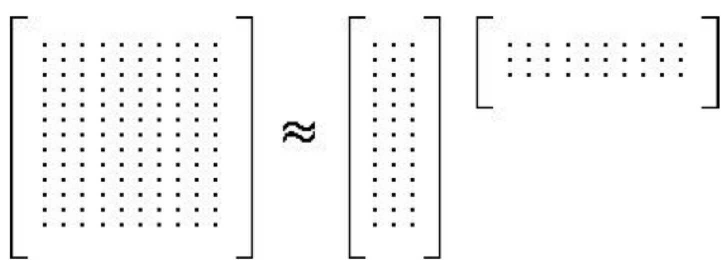
1. $A + B$
2. $A + BC$
3. $(AB)^{-1}$
4. $(ABC)^{-1}$
5. CBx
6. $Ax + Cx$

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.

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 *B*C*, 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

- $A = LU$
  - Factoring a matrix M representing lower and upper triangular matrices L and U prove useful in solving linear systems.
  - The determinant of M is the product of the main diagonal elements of U.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}$$

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.

$$\det(A - \lambda I) = \begin{vmatrix} -\lambda & 1 & 1 \\ 1 & -\lambda & 1 \\ 1 & 1 & -\lambda \end{vmatrix} \qquad \begin{aligned} &= -\lambda^3 + 3\lambda + 2 \\ &= (\lambda - 2)(\lambda + 1)^2 \qquad \lambda_1 = 2,\ \lambda_{2,3} = -1 \end{aligned}$$

Computing Eigenvectors

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

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

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

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

-λ1 *v1,1 + v1,2 = 0

-2 * v1,1 + (-3- λ1)*v1,2 = 0

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

Eigenvalue Decomposition

[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*)

- Larger eigenvalues correspond to more important vector products.

Singular Value Decomposition

특이값분해(singular value decomposition)

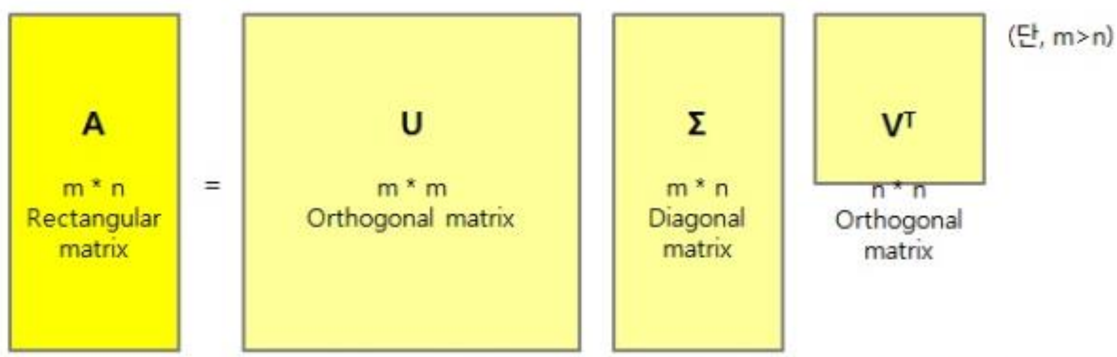
$$A = U\Sigma V^T$$

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)

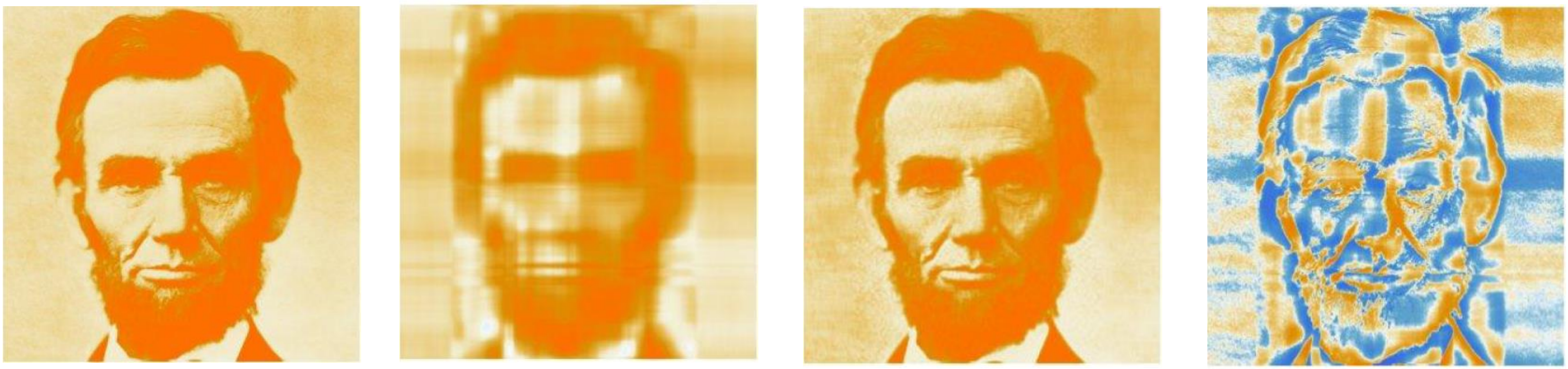


[R 분석과 프로그래밍] http://rfriend.tistory.com

- Retaining only the rows/column with large weights permits us to compress m features with relatively little loss.

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



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
  - Note: this is important from the first (storage-based) perspective of matrices, the linear algebra is the same (mostly)

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

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

$$\begin{array}{l} \text{data} = [2\ 4\ 1\ 3\ 1\ 1] \\ \text{row-indices} = [1\ 3\ 2\ 0\ 3\ 1] \\ \cancel{\text{col-indices} = [0\ 0\ 1\ 2\ 2\ 3]} \\ \qquad\qquad\Downarrow \\ \text{col-indices} = [0\ 2\ 3\ 5\ 6] \end{array}$$

- *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