선형대수학

분야	Mathematics

선형대수학이란?

선형대수학(Linear algebra)은 현대 대부분의 과학과 공학에서 거의 필수적으로 사용되는 수학일 것이다.

선형대수학은 수학, 물리, 생명과학, 지구과학, 화학과 같은 기초과학 뿐만 아니라, 기계공학, 로보틱스, 인공지능, 컴퓨터 공학, 컴퓨터 비전, 양자역학 등 많은 응용분야에서 사용된다.

선형대수학이 쓰이는 분야보다 오히려 쓰이지 않는 분야를 찾는 것이 더 빠를지도 모른다.

앞으로 선형대수학을 이해하고 선형대수학에서 주로 사용되는 언어들을 익히는 것은 분야를 불문하고 현대인의 가장 기초적인 지식이 되어갈 것이다.

이 자료는 선형대수학을 처음 접하는 학생들이 단기간에 주성분분석(Principal Component Analysis) 개념을 맛볼 수 있도록 하는 것을 목표로 하고 있다.

사실 주성분분석을 충분히 이해하고 계산하기 위해서는 대학교 두 학기 과정 정도의 시간이 필요하다.

특히 예상독자가 선형대수학을 처음 접하는 사람임을 고려해 내용의 수학적 엄밀함을 상당히 제하였고, 다소 직관적인 설명으로 대체한 부분이 많다.

이 자료의 부족한 부분들을 보충하고자 한다면, 다음의 자료들을 추천한다.

- 선형대수와 군 - 이인석 저, 서울대학교문화출판원
- Mathematics for Machine Learning

1. 벡터(Vector)

좌표평면 또는 좌표공간을 들어본 적이 있을 것이다.
중학교 수학에서 온전히 공리와 보조선을 이용해 논리로 증명하던 기하학은 좌표평면과 좌표공간을 도입하면서 좌표 사이의 거리를 계산하는 기하학으로 변모하게 해주었다.

좌표평면이나 좌표공간에서 우리는 점을 $a = (x, y)$ 또는 $a = (x, y, z)$ 꼴로 표현했다.

그리고 두 점 사이의 거리를 구하거나, 이로부터 직선의 방정식을 구하는 등의 계산을 해보았을 것이다.

벡터는 이러한 아이디어의 확장판이다.

벡터는 좌표공간 위에 고정된 한 점 $a = (x, y, z)$을 원점에서 출발하는 하나의 화살표로 이해하는 것이다.

점을 화살표로 표현하게 되면 생기는 이점이 있다.

이제 좌표공간 상의 한 점 $a = (x, y, z)$는 단순히 위치정보만 가진 것이 아니라, 하나의 운동량으로서 크기와 방향을 가지게 된다.

즉, 화살표의 길이를 우리는 벡터(운동)의 '크기'라고 해석하고 화살표의 방향을 벡터(운동)의 '방향'으로 해석하는 것이다.

점을 화살표로 표현하면서 생기는 이점은 해석의 확장 말고도 많다.

좌표공간상의 두 점 $a = (x_1, y_1, z_1), b = (x_2, y_2, z_2)$를 더하는 시도는 문제를 풀면서 해본적이 있을 것이다.

하지만 두 점을 고정된 위치로 해석하면 두 점의 합이 무엇을 의미해야할지 모호하다.

그러나 두 점을 각각 벡터라는 화살표로 이해하고 나면 두 벡터의 합은 원점에 동시에 가해지는 두 운동의 결합으로 이해할 수 있게 된다.

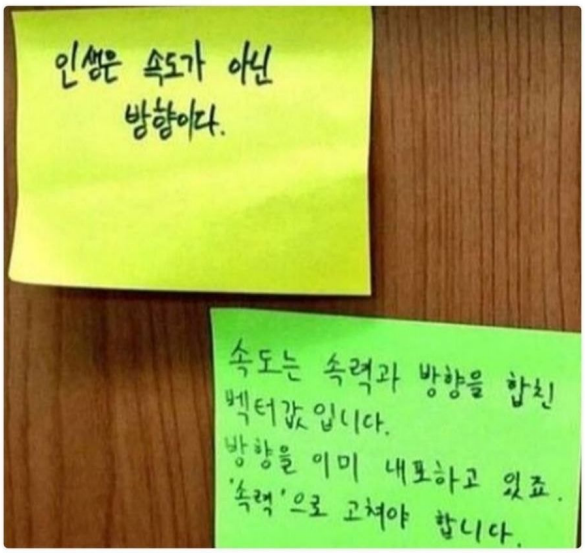
서로 방향과 크기가 다른 운동이 하나의 점에 작용했을 때, 최종적으로 어떤 크기와 방향을 가지는 운동이 되는 것인지를 계산하는 것은 굉장히 복잡한 일 같지만, 이를 벡터의 합으로 생각하면 굉장히 간단한 계산으로 얻을 수 있다.

데이터과학에서는 하나의 데이터를 표현하기 위해 벡터를 사용한다.

데이터를 벡터로 이해하면 선형대수학에서 사용하는 여러 유용한 도구를 사용해 데이터를 요약하거나 머신러닝 모델을 개발하는 데에 쓰일 수 있다.

대표적으로 데이터의 정보를 압축하는 데에 쓰이는 주성분분석(Principle Component Analysis)와 분류 예측을 수행하는 머신러닝 모델인 서포트 벡터 머신(Support Vector Machine) 등이 있다.

1.1) 벡터의 정의



'속도(velocity)'는 대표적인 벡터량이다. 속도는 '속력(speed)'과 '방향(direction)'을 모두 내포한 개념이다. 그래서 속도를 표현하기 위해 이 물체의 속도는 북쪽 방향으로 100km/h다 라고 방향과 속력을 동시에 표현해주어야 한다. 이러한 개념이 익숙한 이과생들에게 "인생은 속도가 아닌 방향이다"라는 관용구는 본능적인 거리낌이 든다. 그렇지만 사회생활에서 이런 거리낌을 여과없이 표출하면 주변에서 내가 거리끼는 사람이 될 수 있으므로 주의하도록 한다.

많은 물리량들은 크기(magnitude)와 방향(direction)의 개념을 포함하고 있다.

예를들어, 속도(velocity)는 '북쪽 방향으로 60km/h'와 같이 표현된다.

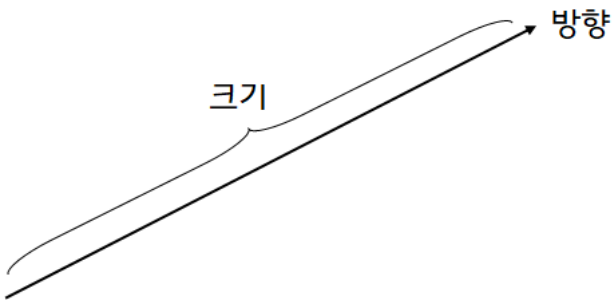
이처럼 방향과 크기로 표현되는 물리량들을 '벡터량'이라고 부른다.

반대로, 하나의 실숫값으로만 표현되는 물리량들을 '스칼라량'이라고 부른다.

예를들어, 속력(speed)는 '60km/h'와 같이 표현된다.

벡터는 크기와 방향을 나타내기에 용이한 화살표로 나타내어진다.

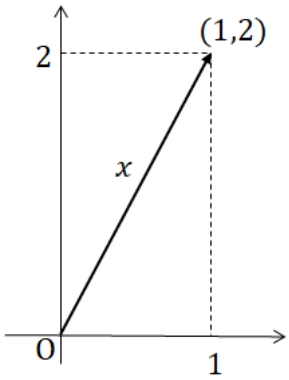
화살표의 시작점을 벡터의 시점, 화살표의 머리를 벡터의 종점이라고 한다.



벡터는 화살표로 나타내어질 수 있다. 화살표의 길이는 벡터의 크기를, 화살표의 머리는 벡터의 방향을 나타낸다.

좌표평면 또는 좌표공간의 점은 원점을 시점으로 하고, 그 점을 종점으로 하는 벡터로 표현할 수 있다.

예를들어, 좌표평면 위의 점 $x = (1, 2)$는 아래의 화살표처럼 나타낼 수 있다.



앞으로 점과 구분하기 위해 벡터는 $\mathbf{x} = [1, 2]$와 같이 나타내자.

위 벡터 $\mathbf{x}$의 방향은 원점에서 $(1, 2)$로 향하는 방향(북동쪽)이고, 그 크기는 원점과 점 $(1, 2)$ 사이의 거리인 $\sqrt{5}$가 된다.

벡터와 점은 똑같은 정보를 나타내는것 같은데 왜 구분을 하느냐 질문할 수 있다.

좋은 질문이다.

가장 큰 차이점은, 기하학에서 다루는 점은 점끼리 더하거나 상수배한다는 것을 생각할 수 없지만, 벡터는 더하거나 상수배하는 것을 생각할 수 있다는 점이다.

나중에 벡터의 연산에서 이 부분을 더욱 자세히 살펴보도록 하자.

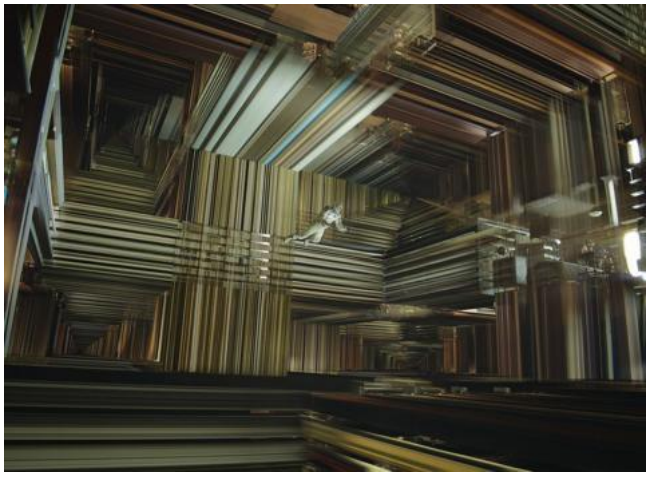
차원과 벡터공간

하나의 벡터를 표현하기 위해 필요한 숫자의 개수를 **차원(dimension)**이라고 부른다.

평면벡터는 2차원 벡터, 공간벡터는 3차원 벡터다.

당연히, 얼마든지 그 이상의 차원을 가진 벡터를 생각할 수 있다.

4차원 이상의 벡터는 시각화하기는 어렵지만, 논리적으로는 2차원, 3차원 벡터와 동일한 원리를 가지고 있다.



5차원 공간을 표현한 영화 인터스텔라의 한 장면.

n-차원 벡터들의 모임을 **벡터공간(Vector space)**라고 부른다.

벡터공간의 가장 중요한 성질은 **기저(basis)**를 가지고 있다는 점이다.

기저란, 벡터공간의 원소인 벡터를 표현하기 위한 기본단위를 의미한다.

예를들어, 우리는 보통 공간좌표 (a, b, c)를 표현하기 위해 원점에서부터 **가로**로 a만큼, **세로**로 b만큼, **높이**는 c만큼 움직여야 한다고 표현한다.

이때, 이 정보를 표현하기 위해 사용된 기본단위인 가로, 세로, 높이가 3차원 공간의 기저 역할을 한다.

다시말해, 3차원 공간의 임의의 벡터 $[a, b, c]$는 $a[1, 0, 0] + b[0, 1, 0] + c[0, 0, 1]$로 표현할 수 있으므로, 세 벡터 $[1, 0, 0], [0, 1, 0], [0, 0, 1]$는 3차원 벡터공간의 기저가 된다.

데이터 과학에서의 벡터

데이터 과학에서는 데이터를 표현하기 위해 벡터가 사용된다.

예제 1.1) 데이터의 표현

타이타닉(Titanic)은 건조 당시 세계 최대의 여객선이었지만, 1912년 최초의 항해 때 빙산과 충돌해 침몰한 비운의 여객선. 세계에서 가장 유명한 여객선이다.

다음은 타이타닉 탑승객 데이터의 일부를 나타낸 표이다.

PassengerId	Survived	Pclass	Age	SibSp	Parch	Ticket	**Fare**
887	0	2	27.0	0	0	211536	13.00
888	1	1	19.0	0	0	112053	30.00
890	1	1	26.0	0	0	111369	30.00

- PassengerId : 각 승객별 고유번호
- Survived : 생존 여부 (종속 변수)
- Pclass : 1등석, 2등석, 3등석
- Age : 나이
- SibSp : 동반한 Sibling(형제자매)와 Spouse(배우자)의 수
- Parch : 동반한 Parent(부모) Child(자식)의 수
- Ticket : 티켓의 고유번호
- Fare : 티켓의 요금

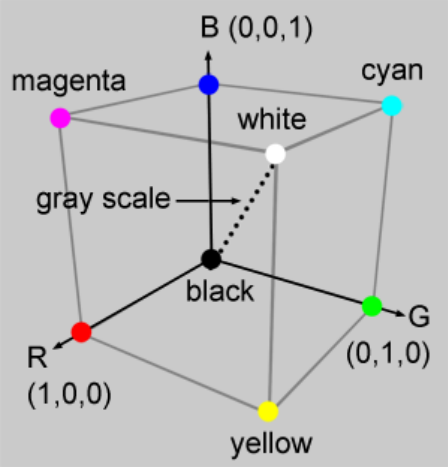
각각의 데이터 샘플(각 행)은 8차원 벡터로 표현될 수 있다.

예제 1.2) 컬러 픽셀

이미지 데이터는 여러 개의 RGB 컬러 픽셀들로 구성된다.

픽셀들은 세 가지의 숫자들로 구성되는데, 각각의 숫자는 빨강(R), 초록(G), 파랑(B)의 색 농도를 나타낸다.

따라서, RGB 컬러 벡터들이 사는 공간은 R, G, B를 기저로 가지는 3차원 벡터공간으로 생각할 수 있다.



3차원 RGB space (이미지 출처 : https://www.scratchapixel.com/lessons/digital-imaging/colors/color-space.html)

연습문제 - 벡터의 정의

1. 지구상의 각 지점에서 매 시각마다 풍향, 풍속, 기온, 기압 및 습도를 기록한다면, 이 기록은 몇 차원공간의 점으로 이해할 수 있는가? [김홍종, 미분적분학 1, 서울대학교출판문화원]
2. 과거에 공부한 교과서에서 다항식의 덧셈과 실수배, 복소수의 덧셈과 실수배, 함수의 덧셈과 실수배, 수열의 덧셈과 실수배, 미분연산자의 덧셈과 실수배, 적본연산자의 덧셈과 실수배에 대한 내용을 찾아보고 이들이 모두 벡터의 연산법칙 8가지를 만족하는 것을 살펴보라. 이들은 모두 벡터인가?

1.2) 벡터의 연산

벡터를 이용하면 한 점에 두 가지 운동이 작용하는 상황을 표현하기 좋다.

한 점에 두 가지 운동이 동시에 작용한다고 해서 그 '크기'가 더해지는 것이 아니다.

벡터의 합을 이용하면 한 점에 두 가지 운동이 작용했을 때, 그 결과 어떤 크기와 방향을 가지는 운동이 작용하는 것과 같은지를 쉽게 표현할 수 있게 된다.

벡터의 덧셈 - 평행사변형법

두 벡터의 덧셈은 아래 그림과 같이, 평행사변형법(*parallelogram law*)에 의해 표현된다.

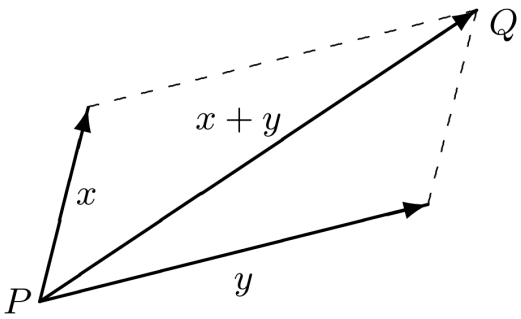


Figure 1.1

벡터는 크기와 방향만 같다면, 위치에 상관없이 동일한 것으로 간주한다.

그래서 위 그림처럼 벡터 x와 y의 덧셈은 벡터 x의 화살표 끝을 시점으로 하여 벡터 y를 옮겨도 $x+y$를 계산하는 것은 기하학적으로 동일하다.

이렇게 P에서 출발해 두 화살표를 따라가게 되면 Q에 도착하게 된다.

그래서 두 벡터 x, y의 합 $x+y$은 점 P에서 출발해 점 Q에 도착하는 하나의 벡터로 나타나게 된다.

벡터의 덧셈 - 좌표별 덧셈

벡터의 덧셈은 대수적으로도 쉽게 계산될 수 있다.

좌표평면 위의 두 벡터 $\mathbf{x} = [a_1, a_2]$와 $\mathbf{y} = [b_1, b_2]$가 있다고 하자.

그러면 두 벡터의 합은 각 성분별로 더한 $\mathbf{x} + \mathbf{y} = [a_1 + b_1, a_2 + b_2]$가 된다.

일반적으로 n-차원 공간에서 벡터의 합은 대수적으로 다음과 같이 정의된다.

▼ ***Definition) 벡터의 덧셈***

n-차원 공간 $\mathbb{R}^n$의 두 벡터 $[a_1, a_2, \ldots, a_n]$, $[b_1, b_2, \ldots, b_n] \in \mathbb{R}^n$의 덧셈은 각 좌표별(componentwisely) 덧셈으로 정의된다.

즉, $[a_1, a_2, \ldots, a_n] + [b_1, b_2, \ldots, b_n] = [a_1 + b_1, a_2 + b_2, \ldots, a_n + b_n]$이다.

아래의 그림 (a)를 보면, 대수적인 벡터의 덧셈과 평행사변형법이 동일함을 관찰할 수 있다.

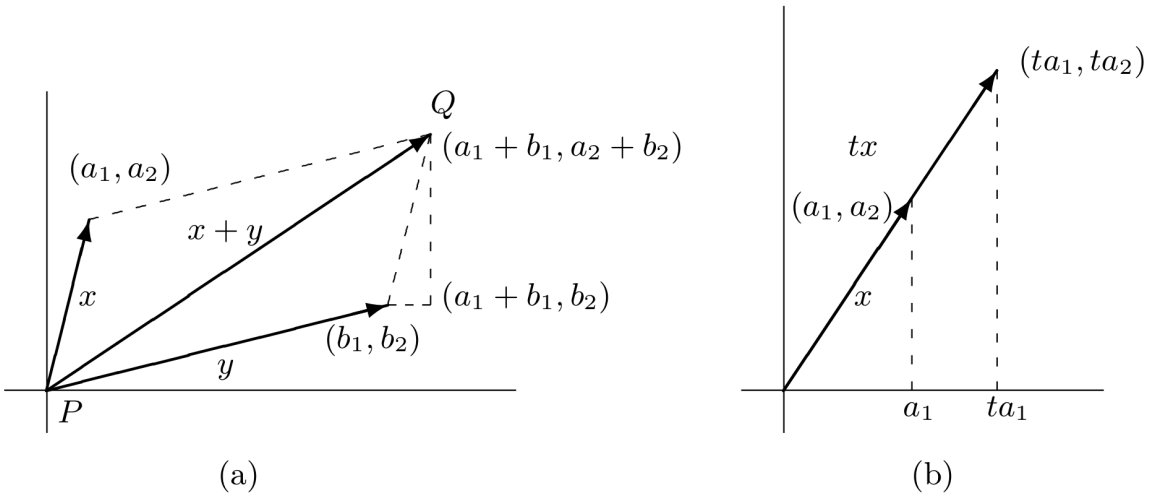


Figure 1.2

벡터의 스칼라배 - 크기의 변화

벡터에는 스칼라배(곱)도 존재한다.

스칼라배는 벡터에 실수 t를 곱해, 방향은 변하게하지 않고 오직 크기만을 변하게 만드는 연산이다.

스칼라배 연산도 마찬가지로 실수 t를 성분별로 곱하기만 하면 된다.

그 기하학적 해석은 위 그림 (b)와 같다.

사실 벡터의 방향이라는 것은 스칼라배에 의해 정해진다.

두 벡터가 x, y가 방향이 같다는 것은 임의의 스칼라 $t \geq 0$이 존재하여, $y = tx$가 만족하는 것을 말한다.

유사하게, 두 벡터의 방향이 반대라는 것은 임의의 스칼라 $t < 0$이 존재하여, $y = tx$가 만족하는 것을 말한다.

이 두 가지 경우 모두 두 벡터가 평행(parallel)하다고 한다.

벡터의 스칼라배 - 좌표별 상수배

벡터의 스칼라배는 벡터 $\mathbf{x}$와 스칼라(상수) r 사이의 곱셈이다.

일반적으로 n-차원 공간벡터의 스칼라배는 다음과 같이 정의된다.

▼ ***Definition) 벡터의 스칼라배***

n-차원 공간 $\mathbb{R}^n$의 벡터 $[a_1, a_2, \ldots, a_n] \in \mathbb{R}$와 실수 $t \in \mathbb{R}$의 스칼라배는 각 좌표별(componentwisely) 상수배로 정의된다.

즉, $t[a_1, a_2, \ldots, a_n] = [ta_1, ta_2, \ldots, ta_n]$이다.

벡터의 연산성질

대수적으로 벡터의 연산은 다음 8가지 성질이 성립한다.

💡 ***벡터의 연산성질***

1. **(덧셈에 대한 교환법칙)** 임의의 두 벡터 x, y에 대하여, $x + y = y + x$
2. **(덧셈에 대한 결합법칙)**임의의 세 벡터 x, y, z에 대하여, $x + (y + z) = (x + y) + z$
3. **(덧셈에 대한 항등원)** $\mathbf{0}$로 표기되는 벡터가 존재하여, 각 벡터 x에 대해 $x + \mathbf{0} = x$가 성립한다.
4. **(덧셈에 대한 역원)** 각 벡터 x에 대하여, 벡터 y가 존재하여, $x + y = \mathbf{0}$가 성립한다.
5. **(스칼라배에 대한 항등원)** 각 벡터 x에 대하여, $1x = x$이다.
6. **(스칼라배에 대한 결합법칙)** 임의의 두 실수 a, b와 벡터 x에 대하여, $(ab)x = a(bx)$가 성립한다.
7. **(분배법칙 1)** 임의의 실수 a와 두 벡터 x, y에 대하여, $a(x + y) = ax + ay$가 성립한다.
8. **(분배법칙 2)** 임의의 두 실수 a, b와 벡터 x에 대하여, $(a + b)x = ax + bx$가 성립한다.

수학적으로는 어떤 집합에 덧셈과 스칼라배 연산이 주어지고 위 8가지 성질을 만족할 때, 그 집합을 **벡터공간(Vector space)**라고 정의한다.

예제 1.3) 평면벡터의 연산

원점을 시점으로 하는 두 벡터 $\mathbf{x} = [1, 1]$과 $\mathbf{y} = [-1, 1]$에 대하여, 다음을 계산하고 그 결과를 좌표평면 위에 나타내시오.

1. $2\mathbf{x}$
   ▼ ***Solution)***
   $2\mathbf{x} = 2[1, 1] = [2, 2]$
2. $\mathbf{x} + \mathbf{y}$
   ▼ ***Solution)***
   $\mathbf{x} + \mathbf{y} = [1, 1] + [-1, 1] = [1 - 1, 1 + 1] = [0, 2]$
3. $2\mathbf{x} + 2\mathbf{y}$
   ▼ ***Solution)***
   $2\mathbf{x} + 2\mathbf{y} = 2[1, 1] + 2[-1, 1] = [2 - 2, 2 + 2] = [0, 4]$
4. $\mathbf{x} - \mathbf{y}$
   ▼ ***Solution)***
   $\mathbf{x} - \mathbf{y} = [1, 1] - [-1, 1] = [1 - (-1), 1 - 1] = [2, 0]$
5. $\mathbf{x} + 2\mathbf{y}$
   ▼ ***Solution)***
   $\mathbf{x} + 2\mathbf{y} = [1, 1] + 2[-1, 1] = [1 - (-2), 1 - 2] = [3, -1]$

1.3) 벡터의 내적

벡터를 이용해 기하학 문제를 풀 수 있게 해주는 세 가지 개념이 있다.

크기(norm), 거리(distance), 내적(inner product)가 바로 그것들이다.

벡터의 크기와 거리 개념은 중고등학교 기하학에서도 유사하게 다루어보았기 때문에 익숙할 것이다.

그러나, 내적의 개념은 다소 직관적으로 받아들이기 어렵다.

크기(norm)

벡터의 크기(norm)는 이름이 시사하는 바대로 벡터의 시점부터 종점까지의 길이를 의미한다.

일반적으로는 다음과 같이 정의된다.

▼ *Definition) 크기*

벡터 $\mathbf{x} = [a_1, a_2, \dots, a_n] \in \mathbb{R}^n$의 크기는 $\|\mathbf{x}\|$로 표기하고, $\|\mathbf{x}\| = \sqrt{a_1^2 + a_2^2 + \cdots + a_n^2}$으로 정의된다.

크기가 1인 벡터를 단위벡터라고 한다.

거리(distance)

거리는 이름 그대로 두 벡터의 종점 사이의 거리를 의미한다.

이때, 두 벡터의 시점은 같은 것으로 본다.

벡터의 거리 개념은 두 점 사이의 거리 개념과 같다.

그러나, 거리를 벡터의 크기를 이용해 정의할 수도 있다.

▼ *Definition) 거리*

두 벡터 $\mathbf{x} = [a_1, a_2, \dots, a_n], \mathbf{y} = [b_1, b_2, \dots, b_n]$의 거리는 $d(\mathbf{x}, \mathbf{y})$로 표기하고, $d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\| = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \cdots + (a_n - b_n)^2}$으로 정의된다.

내적(inner product)

벡터의 내적(inner product)은 두 벡터 사이의 관계를 하나의 스칼라값으로 나타내주는 함수이다.

벡터의 내적은 두 가지 방법으로 정의할 수 있다.

먼저 크기와 삼각함수를 이용해 정의할 수 있다.

두 벡터 $\mathbf{x} = [a_1, a_2, \dots, a_n], \mathbf{y} = [b_1, b_2, \dots, b_n]$의 내적은 $\mathbf{x} \cdot \mathbf{y} = \|\mathbf{x}\|\|\mathbf{y}\| \cos\theta$로 정의된다.

여기서 θ는 두 벡터 $\mathbf{x}$와 $\mathbf{y}$ 사이의 각이다.

이 정의를 이용하면 각과 관련된 기하학 문제를 벡터를 이용해 풀 수 있을 것이다.

그러나, 실제로 두 벡터 사이의 내적을 계산하고자 하면 두 벡터 사이의 각을 알고 있어야 한다는 문제가 생긴다.

다음 정의는 놀랍게도 위 정의와 동일한 값을 산출하지만, 계산이 훨씬 편리하다.

두 벡터 $\mathbf{x} = [a_1, a_2, \dots, a_n], \mathbf{y} = [b_1, b_2, \dots, b_n]$의 내적은 $\mathbf{x} \cdot \mathbf{y} = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$으로도 정의된다.

지금 바로 각과 크기를 계산하기 쉬운 두 벡터를 잡아 계산해보라.

이 계산 덕분에 역으로 두 벡터 사이의 각에 대한 정보를 다음과 같이 계산할 수 있다.

$\cos\theta = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\|\|\mathbf{y}\|}$.

두 벡터의 성분들만 알고 있으면 우변을 정확하게 계산할 수 있기 때문에, 각도기가 없어도 각을 계산할 수 있게 된다.

또한 좌변이 코사인 함수로 표현된다는 것을 상기해보자.

코사인 함수는 $\theta = \frac{\pi}{2} = (90^\circ)$일 때 0의 값을 가진다.

즉, 두 벡터가 수직일 때 내적의 값은 0이 된다.

거꾸로 내적의 값이 0이면 항상 두 벡터는 수직이다.

따라서, 내적을 이용하면 두 벡터가 수직인 벡터인지 아닌지 알 수 있다.

예제 1.4) 계산

다음을 계산하시오.

1. $\mathbf{x} = [2, 2, -1]$일 때, $\|\mathbf{x}\|$.
2. $\mathbf{x} = [1, 3, -2, 7], \mathbf{y} = [0, 7, 2, 2]$일 때, $d(\mathbf{x}, \mathbf{y})$.
3. $\mathbf{x} = [-1, 3, 5, 7], \mathbf{y} = [5, -4, 7, 0]$일 때, $\mathbf{x} \cdot \mathbf{y}$.

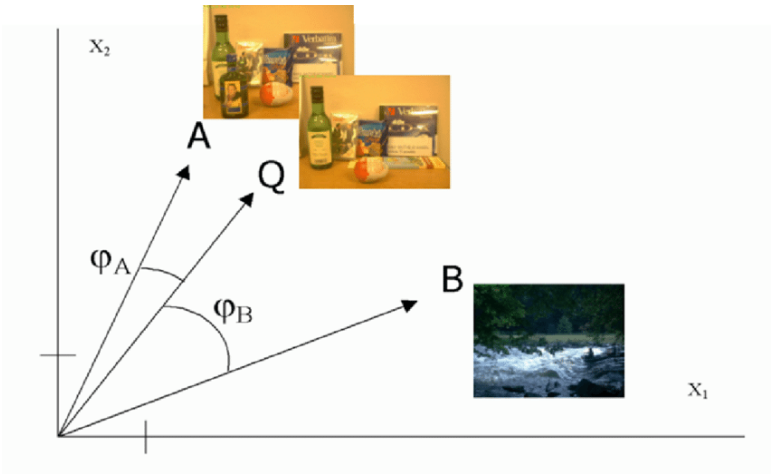
예제 1.5) 코사인 유사도

내적의 개념을 이용하면 두 데이터 사이의 유사도(Similarity)를 정의할 수 있다.

그 방법은 내적을 이용해 두 데이터 사이의 코사인 값을 계산하는 것이다.

만약, 두 데이터가 유사하다면 코사인 값이 1에 가까울 것이고 두 데이터가 유사하지 않다면 코사인 값이 0에 가까울 것이다.

이러한 아이디어를 기반으로 닮은 데이터와 그렇지 않은 데이터를 분류할 수 있다.



코사인 유사도를 이용한 유사 이미지 분류의 스키마틱. (이미지 출처 : https://www.researchgate.net/figure/b-The-cosine-similarity-example-in-2-D-space-Here-A-Q-B-represents-the-vectors_fig2_267819405)

2. 행렬 (Matrix)

영화 '매트릭스'(1999)는 인간이 가상세계에 살면서 기계에 의해 지배되는 세계관을 그리고 있다. 영화에서 기계가 만든 가상세계는 사람이 사는 세계와 구분하기 어렵게 만들어져 있다. 기계가 사람이 사는 세계의 원리를 잘 구현해내었기 때문이다. 행렬(Matrix)는 사람이 사는 세계의 원리인 함수를 기계가 이해할 수 있는 숫자로 표현해주는 역할을 하는 도구라는 점에서, 영화 매트릭스의 제목은 또 다른 의미로 다가온다.

행렬은 숫자들을 직사각형 모양의 틀에 행(가로)과 열(세로)을 맞추어 배열한 것에 지나지 않다.

그러나, 다소 시시해보이는 겉모습과는 달리, 행렬은 현대 과학의 가장 중요한 도구다.

행렬은 사람의 언어로 표현된 함수를 컴퓨터가 이해할 수 있는 숫자로 바꾸어 나타낸 것이라 이해할 수 있다.

예를들면, "좌표평면 위의 임의의 점을 x축에 대하여 대칭시키는 함수"는 인간의 언어로 표현된 함수다.

이 함수는 행렬로 $\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$와 같이 표현될 수 있다.

행렬을 이용하면 생기는 이점 중 하나는 복잡한 인간의 언어로 표현된 함수를 컴퓨터에 입력할 수 있게 된다는 점일 것이다.

컴퓨터를 이용해 복잡한 함수 계산을 쉽게 수행하는 비결이 행렬에 있다고 할 수 있다.

사실 컴퓨터의 발전 역사는 행렬 계산의 발전 역사와 함께 해왔다고해도 과언이 아니다.

행렬에 대한 철학을 잘 이해만 해도 수학의 절반은 이해했다고 할 수 있을 것이다.

행렬을 잘 이해하게 되면, 현대의 어떤 낯선 과학을 만나더라도 행렬이라는 도구로 해석할 수 있게 된다.

다음 명제는 조금 과장되어 보이지만, 연구자의 경험으로서는 매번 놀랍도록 공감이 된다.

"우리가 아는 것은 행렬뿐이다." [이인석, 선형대수와 군, 서울대학교출판문화원]

행렬이 담고 있는 철학은 단순명료하지만, 그것을 삶으로 체화하는 데에는 적지않은 노력과 경험이 필요하다.

단숨에 이를 얻어내기는 어렵겠지만, 어딘가 지향점이 있다는 것을 먼저 상기하고서 그 지향점을 찾아 천천히 음미하며 공부해나가길 바란다.

2.1) 행렬의 정의

다음 미지수가 x_1, x_2, x_3 세 개인 연립일차방정식을 생각해보자.

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 &= 0 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 &= 0. \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 &= 0 \end{aligned}$$

이 연립방정식을 특징짓는 것은 9개의 계수다.

그러므로, 이들만 하나로 같이 묶어서 표현하면 이 연립방정식 자체를 표현할 수 있을 것이다.

이들 미지수는 다음과 같이 묶어서 표현할 수 있다.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}.$$

위와 같이 숫자를 직사각형 안에 행과 열로 배열한 것을 행렬(matrix)라고 한다.

행렬은 일반적으로 다음과 같이 정의된다.

▼ ***Definition) 행렬***

$i = 1, \dots, m$과 $j = 1, \dots, n$에 대하여, $a_{ij} \in \mathbb{R}$일 때,

$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{33} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{bmatrix}$ 을 $(m \times n)$-행렬이라고 부른다.

행렬 A를 간단히 $A = (a_{ij})_{m \times n} = (a_{ij})$로 표기하기도 한다.

이때, 각 a_{ij}를 행렬 A의 (i, j)-성분이라고 부른다.

행렬 A의 i-번째 가로 줄을 행(row)이라고 부르며, $[A]_i$로 표기한다.

행렬 A의 j-번째 세로 줄을 열(column)이라고 부르며, $[A]^j$로 표기한다.

벡터도 행렬처럼 생각할 수 있다.

지금까지 우리는 벡터를 $\mathbf{x} = [1, 2, 3]$처럼 숫자가 가로로 나열된 것으로 생각했다.

이렇게 하나의 행으로 나타낸 벡터를 행벡터라고 부른다.

행벡터는 행이 하나이고, 열이 n개인 행렬이라고 생각할 수 있다.

동일한 벡터를 $\mathbf{x} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$ 처럼 숫자가 세로로 나열된 것으로 표현할 수 있다.

이렇게 하나의 열로 나타낸 벡터를 열벡터라고 부른다.

열벡터는 열이 하나이고, 행이 m개인 행렬이라고 생각할 수 있다.

지금까지는 벡터가 행벡터인 것처럼 표현했지만, 이제부터는 모든 벡터는 열벡터로 생각하기로 한다.

예제 1.6) 데이터셋의 표현

예제 1.1에서 하나의 데이터 샘플은 벡터로 표현할 수 있다고 했다.

일반적으로 데이터셋은 수천, 수만개 이상의 샘플로 구성된다.

따라서, 데이터셋은 데이터 샘플 수 만큼의 행과 데이터를 표현하는 데에 필요한 특성 수 만큼의 열을 가지는 행렬로 나타낼 수 있다.

2.2) 행렬의 연산

행렬의 기본연산은 벡터와 마찬가지로 덧셈과 스칼라곱이 있다.

그 원리도 동일하다.

행렬의 덧셈과 스칼라곱은 모두 성분별(componentwisely)로 수행된다.

이때, 주의할 점은 두 행렬을 더할 때, 두 행렬은 크기가 같아야 한다는 점이다.

▼ ***Definition) 행렬의 덧셈***

크기가 같은 두 행렬 $A = (a_{ij})$와 $B = (b_{ij})$에 대하여, 두 행렬의 합 $A + B$는 (i, j)-성분이 $a_{ij} + b_{ij}$인 행렬이다.

▼ *Definition) 행렬의 스칼라곱*

행렬 $A=(a_{ij})$와 스칼라 t에 대하여, 스칼라곱 tA는 $tA=(ta_{ij})$로 정의된다.

행렬의 덧셈과 스칼라곱에 대해서도 벡터의 연산이 가지고 있는 8가지 연산성질이 모두 만족한다.

예제 1.7) 행렬의 연산

행렬 $A=\begin{bmatrix}1 & 2\\ 0 & 1\end{bmatrix}$과 $B=\begin{bmatrix}1 & 0\\ -1 & 1\end{bmatrix}$에 대하여, 다음을 계산하라.

1. $A+B$
2. $A-B$
3. $2A$
4. $A-2B$

2.3) 행렬의 곱셈

$a_{11}x_1+a_{12}x_2+a_{13}x_3=0$
$a_{21}x_1+a_{22}x_2+a_{23}x_3=0$
$a_{31}x_1+a_{32}x_2+a_{33}x_3=0$

위 연립방정식의 예제를 다시 생각해보자.

행렬을 사용하여 우린 이 방정식의 계수를 하나의 대상으로 나타낼 수 있었다.

그렇다면 거꾸로 행렬에서 다시 위 연립방정식을 얻으려면 어떻게 해야 할까?

우리가 나타낸 행렬 $A=\begin{bmatrix}a_{11} & a_{12} & a_{13}\\ a_{21} & a_{22} & a_{23}\\ a_{31} & a_{32} & a_{33}\end{bmatrix}$는 계수에 대한 정보를 담고 있으므로, 미지수에 대한 정보를 적절히 표현해주면 될 것이다.

미지수 x_1, x_2, x_3은 하나의 열벡터 $\mathbf{x}=\begin{bmatrix}x_1\\ x_2\\ x_3\end{bmatrix}$로 나타낼 수 있다.

이때, 행렬 A의 각 행과 열벡터 $\mathbf{x}$를 내적하면 $a_{i1}x_1+a_{i2}x_2+a_{i3}x_3$이 되는 것을 관찰할 수 있을 것이다.

따라서, 행렬 A와 벡터 $\mathbf{x}$의 곱 $A\mathbf{x}$를 "행렬 A의 각 행과 열벡터 $\mathbf{x}$의 내적"으로 정의하면 위 연립방정식을 다시 복구할 수 있다.

이 연산은 행렬 A의 행의 개수만큼 수행될 것이므로, 그 결과물은 행이 3개이고 열이 1개인 벡터다.

위 연립방정식은 이 결과물 벡터가 모든 값이 0인 영벡터가 되길 원하는 방정식이다.

(열)벡터는 열이 하나인 행렬이라고 생각할 수 있다고 했으므로, 이러한 곱셈을 행렬과 행렬의 곱셈으로 확장할 수 있다.

이때, 유의할 점은 두 행렬 A와 B를 곱할 때, 행렬 A의 "열"의 개수와 행렬 B의 "행"의 개수가 같아야 곱셈이 정의된다는 것이다.

▼ *Definition) 행렬의 곱셈*

$(m\times n)$-행렬 $A=(a_{ij})$와 $(n\times l)$-행렬 $B=(b_{jk})$에 대하여, 행렬곱 AB는 $(m\times l)$-행렬로서, (i,k)-성분이 A의 i번째 "행" $[A]_i$와 B의 k번째 "열" $[B]^k$의 내적

$[A]_i\cdot[B]^k=a_{i1}b_{1k}+a_{i2}b_{2k}+\cdots+a_{in}b_{nk}$으로 정의한다.

행렬의 곱셈은 성분별로 곱하는 것이 아님에 유의하자.

행렬의 곱셈은 교환법칙이 성립하지 않음도 대표적인 유의사항이다.

행렬에 대해 처음 소개할 때, 행렬은 인간의 언어로 된 함수를 컴퓨터가 이해할 수 있는 숫자로 나타낸 것이라고 했다.

행렬의 곱셈이 바로 이 함수를 작용시키는 방법이다.

예를들어보자.

도입부에서 행렬 $A = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$는 평면 위의 점을 x축에 대하여 대칭하는 함수를 나타낸다고 했다.

실제로 그렇게 작동하는지 확인하기 위해, 평면 위의 벡터 $\mathbf{x} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$를 생각하자.

여기서 $A\mathbf{x}$를 계산하면, $A\mathbf{x} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$가 되어, 실제로 그 결과물이 점 $(1, 2)$를 x축에 대하여 대칭시킨 것과 같음을 알 수 있다.

예제 1.7) 행렬의 곱셈

$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$와 $B = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$에 대하여, 다음을 구하시오

1. AB
2. BA
3. A^2
4. B^2.

예제 1.8) 행렬의 곱셈 2

$A = \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix}$이고, $\mathbf{x} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$일 때, 다음을 구하시오.

1. A^2
2. A^3
3. $A^{100}\mathbf{x}$

3. 행렬의 분해 (Matrix Decomposition)

행렬 곱셈이 굉장히 복잡하기 때문에 행렬의 크기가 조금만 커져도 사람이 손으로 계산하기가 어렵다.

불행하게도 실제 현장에서 행렬을 사용할 때에는 굉장히 큰 행렬을 다루어야한다.

그래서 사람들은 행렬의 곱셈을 어떻게 하면 쉽게 할 수 있을까 오랜 시간 연구해왔다.

그 대표적인 방법이 하나의 행렬을 곱하기 쉬운 행렬의 곱으로 분해하는 것이다.

큰 숫자를 거듭제곱할 때, 그 숫자를 소인수분해한 다음 소인수 단위로 거듭제곱하면 그 결과를 다소 쉽게 계산할 수 있는 것과 비슷한 아이디어이다.

이때 사용되는 개념이 고윳값과 고유벡터다.

고윳값과 고유벡터는 행렬마다 가지는 고유한 단짝이다.

서로 마음이 잘 맞는 친구와 대화를 하면, 복잡하게 이야기할 것 없이 이야기의 흐름이 한 방향으로 흘러가곤 하는 것을 경험해 보았을 것이다.

이처럼 어떤 행렬에 고유벡터를 곱하면 마치 고유벡터에 고윳값만큼의 상수배를 한 것이 되어 방향의 변화가 없고 크기만 변하게 된다.

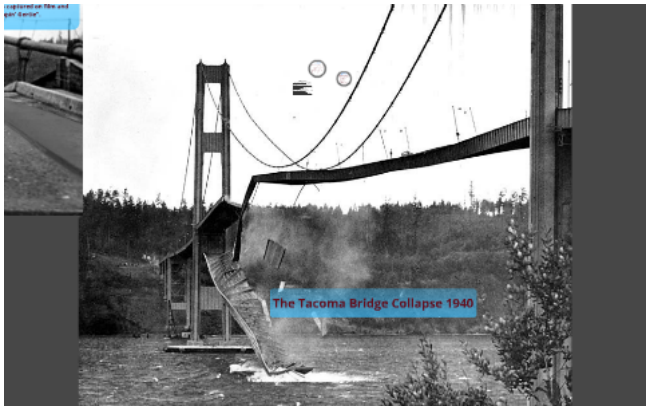
행렬이 인간 혹은 자연의 언어로 표현된 함수를 숫자로 나타낸 것이라고 했던 것을 기억해보자.

예를들면, 외부 진동에 따른 건축물의 진동변화는 행렬로 표현될 수 있다.

이때 건축가가 고려해야할 점은 이 건축물에 불어오는 바람 벡터가 이 행렬의 고유벡터가 되지 않도록 하는 것이다.

만약, 일반적으로 불어올 수 있는 바람 벡터가 건축물 진동 행렬의 고유벡터가 되면 바람에 의해 생기는 진동은 고윳값만큼 증폭될 것이고 이는 건축물이 버티기 힘들만큼 흔들리는 결과를 초래할 수 있다.

실제로 1940년 Tacoma 다리 붕괴사건은 바람이 다리의 고유진동수에 맞추어 불면서 강철로 만든 다리가 엿가락처럼 휘어 무너진 참사다.



강풍에 의해 붕괴된 Tacoma 다리 (1940)

이처럼 행렬은 단순한 숫자들의 모임 같아보여도, 그 안에 숨겨진 특성들이 있어서 이를 잘 이용하면 실생활의 많은 문제들을 해결하는데 큰 도움을 준다.

데이터 과학에서도 데이터를 행렬로 표현한 뒤 고윳값과 고유벡터를 찾아 데이터의 핵심요소를 찾아내는 방법이 흔히 사용된다.

3.1) 고윳값과 고유벡터

행렬의 곱셈은 수의 곱셈과는 완전히 다른 성격을 가지고 있어서 굉장히 신기한 일이 일어나기도 한다.

다음 계산을 잘 관찰해보자.

$$\begin{bmatrix} 2 & 1 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 1 \\ 3 \end{bmatrix} = \begin{bmatrix} 5 \\ 15 \end{bmatrix}.$$

단순한 행렬과 벡터의 곱셈처럼 보인다.

그런데, 우변을 잘 살펴보면 곱했던 벡터 $\begin{bmatrix} 1 \\ 3 \end{bmatrix}$의 5배인 5 $\begin{bmatrix} 1 \\ 3 \end{bmatrix}$와 같다는 것을 관찰할 수 있다.

다시말해, 이 행렬에 벡터 $\begin{bmatrix} 1 \\ 3 \end{bmatrix}$를 곱한 것과 상수배 5를 곱한 것과 같다는 것이다.

어째서 이런 일이 일어난 것일까?

우연일지도 모르니, 다른 벡터를 곱해보라.

이런 일이 발생하는가?

아마 이런 유사한 일이 일어나는 벡터를 쉽게 찾지 못했을 것이다.

행렬마다 어떤 고유한 벡터가 있어서, 그 벡터를 곱했더니 그 벡터의 상수배가 나오는 경우가 있다.

이때, 그 상수를 고윳값(eigenvalue), 그 벡터를 고유벡터(eigenvector)라고 부른다.

행렬의 고윳값과 고유벡터를 찾는 문제는 다소 계산이 복잡하기 때문에 여기서 다루지 않는다.

다만, 몇 가지 예제를 보면서 고윳값과 고유벡터가 무엇인지 느껴보도록 하자.

예제 1.9) 고윳값과 고유벡터

행렬 $A = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$에 대하여, $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$는 고유벡터임을 보이고, 그 고윳값을 구하여라.

3.2) 고윳값 분해 (Eigen Decomposition)

여러가지 모양의 행렬

몇 가지 특이한 모양의 행렬을 소개한다.

$(m \times n)$-크기의 직사각행렬 X에 대하여, 행과 열을 뒤집은 $(n \times m)$-크기의 행렬을 X의 전치행렬이라고 하고 X^T로 표기한다.

예를들어, $X = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$이라면, $X^T = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$이다.

어떤 행렬은 전치를 해도 자기자신이 나오는 행렬이 있다.

물론, 그런 행렬은 정사각행렬이어야만 할 것이다.

전치를 해도 자기자신이 나오는 행렬은 대각선을 기준으로 모든 성분이 같아야 한다.

이를 대칭행렬이라고 한다.

예를들어, $X = \begin{bmatrix} 1 & -1 \\ -1 & 2 \end{bmatrix}$는 대칭행렬이다.

대칭행렬 중, 대각성분만 값이 존재하고 나머지는 0인 행렬을 대각행렬이라고 한다.

예를들어, $X = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$는 대각행렬이다.

대각행렬 중, 대각성분이 모두 1인 행렬을 항등행렬이라고 한다.

항등행렬은 주로 I로 표기한다.

항등행렬은 행렬 곱셈의 항등원 역할을 한다.

즉, 임의의 (사이즈가 맞는) 행렬 A에 대하여, $AI = IA = A$가 성립한다.

일부 행렬들은 곱셈에 대한 항등원으로 자신의 전치행렬을 가지곤 한다.

즉, 어떤 행렬들은 $A^TA = AA^T = I$가 성립한다.

이런 행렬을 직교행렬이라고 한다.

예를들어, $A = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{bmatrix}$는 $A^TA = AA^T = I$ 가 성립하므로 직교행렬이다.

고윳값 분해

고윳값 분해는 정사각행렬 행렬 A를 직교행렬 P와 대각행렬 D를 이용하여, $A = P^TDP$ (또는 $A = PDP^T$)꼴로 분해하는 것을 말한다.

이때, 대각행렬 D의 대각성분은 행렬 A의 고윳값이 되고, 행렬 P는 행렬 A의 고유벡터가 된다.

이러한 분해의 장점은 행렬의 거듭제곱을 계산할 때 나타나게 된다.

앞선 예제에서 행렬의 거듭제곱을 계속해서 반복해나가는 것이 쉽지 않다는 것을 느꼈을 것이다.

그러나, 대각행렬의 거듭제곱은 대각성분만 거듭제곱하면 되기 때문에 상당히 계산하기가 쉽다.

그래서 만약 행렬 A를 고윳값 분해하여 $A = P^T DP$로 나타내고나면, A^{100}을 계산한다고 할 때, $A^{100} = (P^T DP)^{100} = P^T D^{100} P$가 되어 상당히 계산이 편리해진다.

이때, P가 직교행렬이므로 $P^T P = I$가 됨을 이용했다.

고윳값 분해를 직접 구하는 과정은 다소 복잡하기 때문에 여기서 다루지 않는다.

다만, 예제 하나를 살펴보면서 고윳값 분해의 이점을 살펴보자.

예제) 고윳값 분해 (직교대각화)

$A = \begin{bmatrix} 4 & 2 & 2 \\ 2 & 4 & 2 \\ 2 & 2 & 4 \end{bmatrix}$ 가 주어졌을 때, $\mathbf{x}_1 = \begin{bmatrix} -1/\sqrt{2} \\ 1/\sqrt{2} \\ 0 \end{bmatrix}$ 와 $\mathbf{x}_2 = \begin{bmatrix} -1/\sqrt{2} \\ 0 \\ 1/\sqrt{2} \end{bmatrix}$ 이 A의 고유벡터임을 관찰하라.

각각의 벡터는 고윳값을 얼마로 가지는가?

$P = \begin{bmatrix} -1/\sqrt{2} & -1/\sqrt{6} & 1/\sqrt{3} \\ 1/\sqrt{2} & -1/\sqrt{6} & 1/\sqrt{3} \\ 0 & 2/\sqrt{6} & 1/\sqrt{3} \end{bmatrix}$ 이고, $D = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 8 \end{bmatrix}$ 일 때, $A = PDP^T$ 가 성립함을 보여라.

이때, A^3을 계산하여 보아라.

3.3) 특잇값 분해 (Singular Value Decomposition)

고윳값 분해는 굉장히 강력한 행렬 분해 도구이지만, 정사각행렬에 대해서만 사용할 수 있다는 단점이 있다.

그리고 데이터과학에서 만나는 행렬들은 대부분 정사각행렬이 아니다.

$(m \times n)$-크기의 직사각행렬 X에 대하여, 행과 열을 뒤집은 $(n \times m)$-크기의 행렬을 X의 전치행렬이라고 하고 X^T로 표기했다.

그러면, $X^T X$는 크기가 $(n \times n)$인 정사각행렬이 된다.

이때, $X^T X$의 고윳값(의 제곱근)을 X의 특잇값(singular value)라고 부른다.

직사각행렬은 고윳값 분해를 할 수 없었지만, 직사각행렬 X로부터 얻은 $X^T X$는 고윳값 분해가 가능하다.

이를 이용해 우리는 X를 최대한 고윳값 분해와 유사하게 분해할 수 있고, 이를 특잇값 분해라고 부른다.

특잇값 분해는 $(m \times n)$-행렬 X를 $(m \times m)$-직교행렬 U, 주대각선에 특잇값이 들어있는 $(m \times n)$-행렬 Σ, $(n \times n)$-직교행렬 V를 이용해 $X = U^T \Sigma V$로 분해하는 것이다.

그 철학은 고윳값 분해와 같다.

특잇값 분해를 손수 계산하는 것은 굉장히 버거운 일이다.

다만, 그 개념의 핵심 아이디어는 고윳값 분해와 동일하다는 점을 상기하고 후에 파이썬을 이용해 특잇값 분해를 수행해보도록 하자.

4. 주성분분석(Principal Component Analysis; PCA)

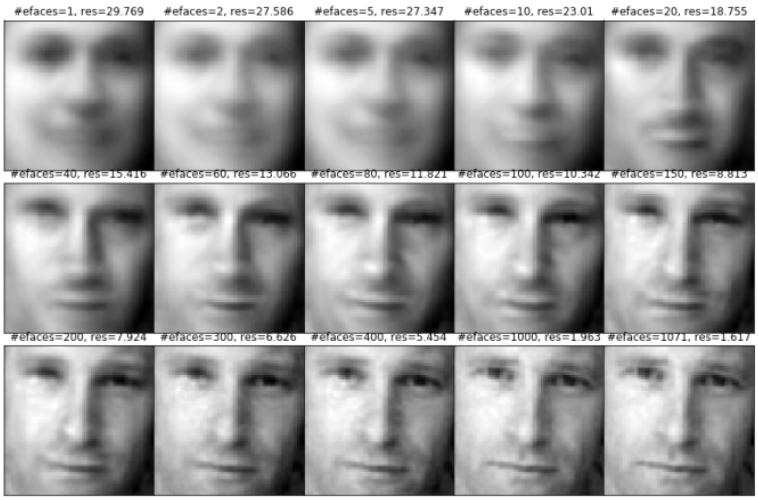
주성분 분석(Principal Component Analysis; PCA)는 데이터에 가장 가까운 초평면(hyperplane)을 정의한 다음, 이 초평면에 데이터를 사영시키는 방법이다.

PCA를 이용하면 데이터에서 모든 정보를 사용하지 않고 핵심정보만 사용해서 문제를 해결할 수 있다.

아래의 그림은 Eigenface라 불리는 PCA의 대표적인 응용이다.

Eigenface는 얼굴 이미지의 핵심 요소만을 추출하여 효율적으로 얼굴 인식 모델을 만들 수 있다.

아래 그림에서 efaces 값이 낮을 수록 데이터가 많이 압축되어 있기 때문에 이미지의 선명도는 낮지만, 이 사람의 핵심적인 얼굴 윤곽은 다 포함하고 있다는 것을 살펴볼 수 있다.



이미지 출처 : https://sandipanweb.wordpress.com/2018/01/06/eigenfaces-and-a-simple-face-detector-with-pca-svd-in-python/

4.1) 분산 보존

데이터에 가장 가까운 초평면을 정의하려면 '가깝다'의 의미를 명확히 해야할 것이다.

PCA에서는 이 가깝다의 기준을 사영했을 때, 원래 데이터가 가지고 있던 '분산을 최대로 보존하는' 평면으로 생각한다.

이것을 조금 더 명료하게 말하자면, 데이터의 평균점을 지나고 사영된 점들의 분산이 원래 데이터의 분산에 가장 가까운, 즉 **평균 제곱 거리(Mean Squared Distance)**가 최소가 되게 하는 방향을 가지는 초평면으로 정의하는 것이다.

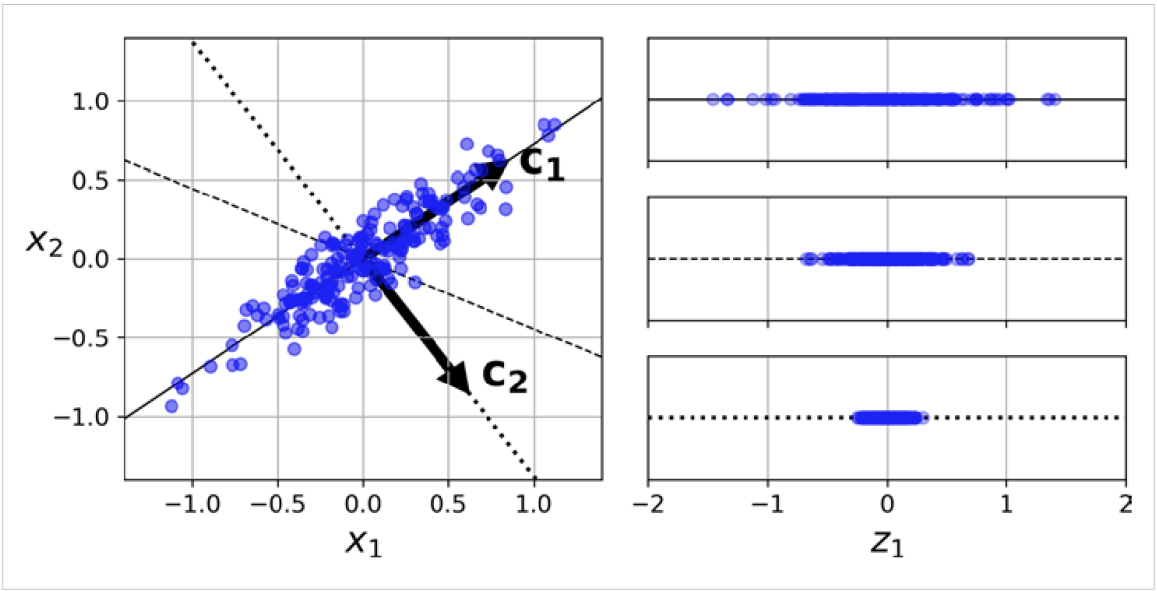


그림 8-7 투영할 부분 공간 선택하기

위 그림에서 보면 실선 C1으로 데이터를 사영한 경우 (우측 그림의 최상단) 분산이 원래 분산에 가장 가깝다는 것을 관찰할 수 있다.

4.2) 주성분분석 알고리즘

PCA의 알고리즘은 다음과 같은 과정을 거친다.

> 📌 *PCA 알고리즘*
>
> 1. 훈련 데이터에서 분산이 최대가 되게 하는 축을 찾는다.
> 2. 해당 축에 대하여 데이터를 사영한다.
> 3. 사영된 데이터를 다시 훈련 데이터로 생각하여, 1과 2를 원하는 만큼 반복한다. 이때, i번째 반복에서 찾은 분산이 최대가 되게 하는 축을 **i번째 주성분(i-th principal component)**라고 한다.

PCA를 이용하면, 데이터가 가지고 있는 중요한 정보를 최대한 보존하면서 데이터를 표현하기 위해 필요한 차원을 줄이는 데에 사용할 수 있다.

이를 **차원축소(Dimensional reduction)**이라고 한다.

4.3) 차원축소 (Dimensional Reduction)

위 숫자 이미지들은 28*28 사이즈의 데이터 크기를 가지고 있다.

이 숫자들은 각각 784 차원의 벡터로 생각할 수 있다.

이처럼 일반적으로 이미지 데이터는 고차원 데이터다.

데이터의 차원이 너무 크게되면 분석하고, 해석하고, 이해하기 어렵다.

뿐만 아니라, 이렇게 큰 차원의 데이터를 사용하는 알고리즘은 상당히 큰 메모리를 필요로 하기 때문에 프로그램이 무거워지게 된다.

따라서 데이터가 가진 주요한 정보를 충분히 보존하면서도 차원을 축소하는 것은 굉장히 유용한 방법일 것이다.

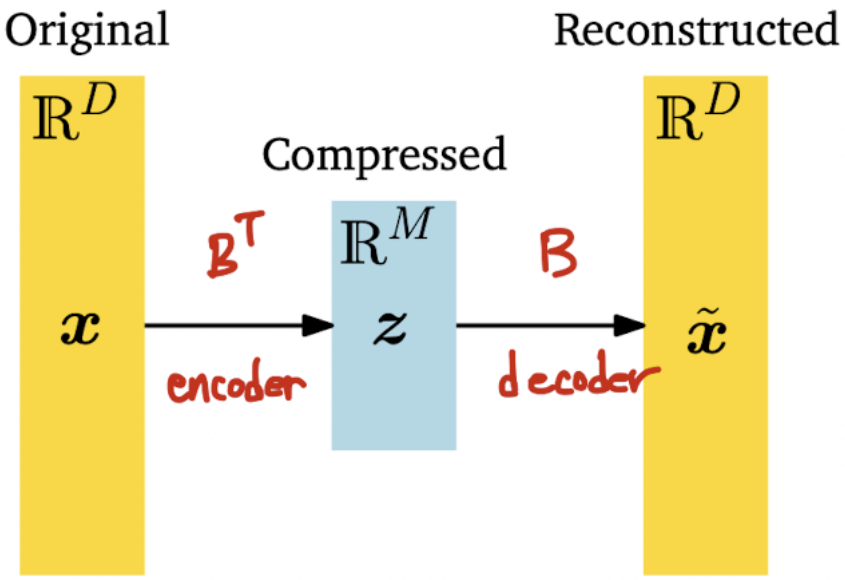
그렇다면, 데이터가 가진 주요한 정보란 무엇일까?

먼저 주요하지 않은 정보가 무엇인지부터 살펴보자.

숫자 데이터의 배경이 되는 흰색 픽셀들은 그다지 의미가 없는 것처럼 보인다.

즉, 784차원 중에 상당수 차원들은 무의미하다.

그러므로 이러한 정보들은 최대한 배제하고, 검은 픽셀들이 많이 위치한 영역만 추출한다면 상당히 많은 차원을 줄이면서 숫자의 모양 정보는 보존할 수 있을 것이다.



이는 '압축'의 개념과 흡사하다.

우리가 듣는 mp3 음원 파일은 녹음실에서 초 고용량으로 녹음된 소리를 아주 저용량으로 압축하여 보급한 것이다.

원본 음원 파일은 녹음실의 환경에 따라 흡입되는 소리, 인간이 듣지 못할 음역대의 소리 등도 포함되어 있을 것이다.

그래서 mp3 파일은 이러한 정보들을 걸러내고 우리가 듣는 데에 '필요한 정보'들만 압축한 것이다.

차원 축소도 이와 유사한 개념의 기술이다.

결론적으로 차원축소는 고차원 공간상에 존재하는 데이터들을 그것을 나타내기에 최적화된 저차원 공간으로 놓는 것을 말한다.

4.4) 수학적 이론 설명

이론적으로 PCA를 살펴보자.

PCA는 데이터를 특정한 초평면으로 사영 시키는 것이다.

이때, 특별히 사영된 벡터들의 분산을 최대한으로 유지도록 한다.

데이터를 X라는 행렬로 나타내어보자.

그러면, X는 대게 정사각행렬이 아닐 것이다.

그러므로 우리는 X의 특잇값 분해(SVD)를 염두해둘 것이다.

계산의 편의를 위해 X의 평균이 영벡터가 되도록 조율했다고 하자.

그러면 데이터 X의 분산을 나타내는 행렬 S는 $S = \frac{1}{n}X^TX$로 간단하게 계산될 수 있다.

여기서 X^TX의 모양을 보면, 특잇값 분해의 모티브가 떠오를 것이다.

정사각행렬 S의 고윳값들은 분명 데이터의 분산에 대한 정보들을 담고 있을 것이다.

이 말은 S를 고윳값 분해하여 $S = P^TDP$로 나타내면, 대각행렬 D에는 분산의 정보가 얼마나 담겨있는지를 나타내는 고윳값들이, 직교행렬 P의 각 열에는 각 고윳값에 대응하는 축 정보가 들어있다는 것이다.

이때, 큰 고윳값에 대응하는 고유벡터가 바로 우리가 원하는 분산을 최대로 보존하는 축이다.

그런데 이 정보는 S를 고윳값 분해하지 않고 X를 특잇값 분해 하는 것으로 얻을 수 있다.

X의 특잇값 분해을 $X = U^T\Sigma V$로 나타내자.

그러면, $X^TX = (U^T\Sigma V)^T(U^T\Sigma V) = V^T\Sigma^TUU^T\Sigma V = V^T\Sigma^T\Sigma V$ 임을 알 수 있다.

이때, Σ에는 X^TX의 고윳값의 제곱근인 X의 특잇값들이 들어있으므로, 대각행렬 $\Sigma^T\Sigma$는 대각성분이 X^TX의 고윳값들로 구성된다.

그러므로 $\Sigma^T\Sigma = D$이며, $X^TX = V^TDV$가 된다.

따라서, S를 고윳값 분해하여 얻은 분산을 최대로 보존하는 축은 X를 특잇값 분해하여 V의 열벡터를 취하는 것으로 얻을 수 있다.

그러므로 PCA를 수행하기 위해서는 X의 특잇값분해 $U^T\Sigma V$에서 V의 열들을 기저로 선택하기만 하면 된다는 것을 알 수 있다.