



수학 1팀

장이준, 박시언

다들 내용:

2.3. 고유값과 고유값 분해

2.4. 고유값과 고유값 분해 응용 및 차원 축소

2.4.1. PCA

2.4.2. ICA

2.4.3. LDA

다루면 좋을 내용:

대칭행렬의 고유벡터 행렬의 특성

2.3. 고유값과 고유값 분해

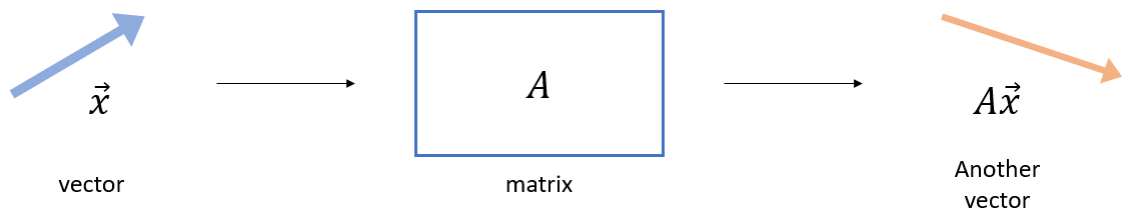
선형대수에서 가장 중요한 개념 중 하나는 행렬의 고유값과 고유벡터이다. 이들은 행렬에 대한 많은 정보를 내포하고 있기 때문에 고유값은 이론적으로 중요할 뿐만 아니라 미분방정식의 해, 행렬의 거듭제곱을 구할 때, 인터넷 검색, 이미지 압축 등 행렬과 관계된 모든 곳에서 응용된다.

2.3.1. 고유값

(1) 고유값(Eigenvalue)과 고유벡터(Eigenvector)

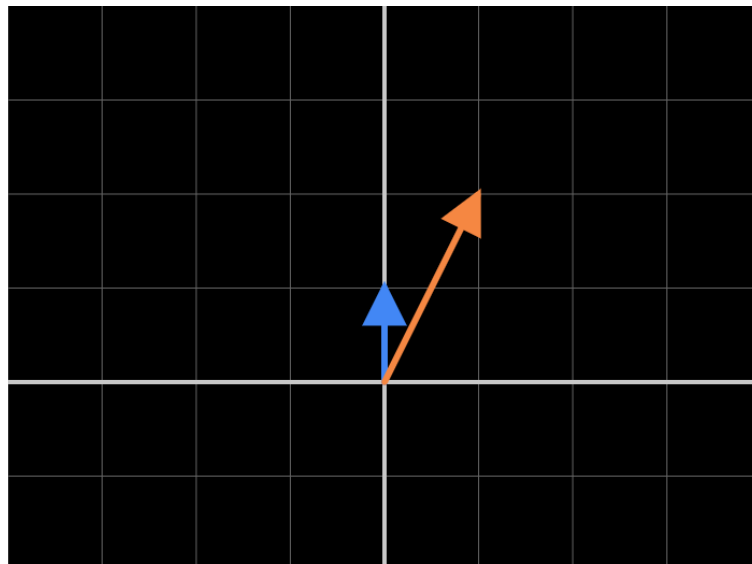
- 행렬 연산의 의미

고유값과 고유벡터에 대해 설명하기 전에 행렬 연산의 의미에 대해 알아보자. 벡터 \mathbf{x} 에 행렬 A 를 곱하여 $A\mathbf{x}$ 라는 결과를 도출하는 것은 어떤 의미일까? 아래 그림처럼 행렬은 벡터를 변환시켜 다른 벡터를 출력해준다.

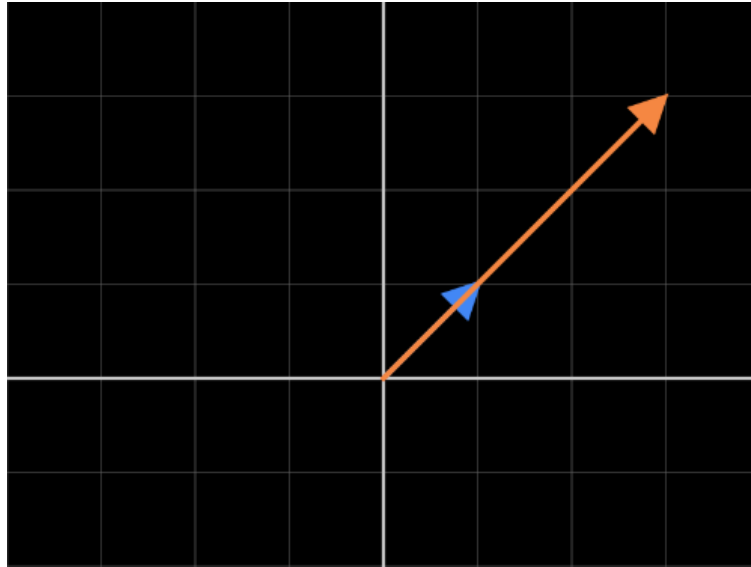


여기서 A 는 벡터 \mathbf{x} 에 함수(function)처럼 작용한다. 행렬을 이용해 벡터를 변환시키면, 변환 후의 벡터는 변환 전의 벡터와 비교했을 때, 크기와 방향이 변하게 된다.

예를 들어 벡터 $\mathbf{x} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ (파란색 화살표)에 행렬 $A = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$ 이용해 선형 변환시키면, 결과는 아래 그림과 같이 $A\mathbf{x} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$ (주황색 화살표)가 나온다.

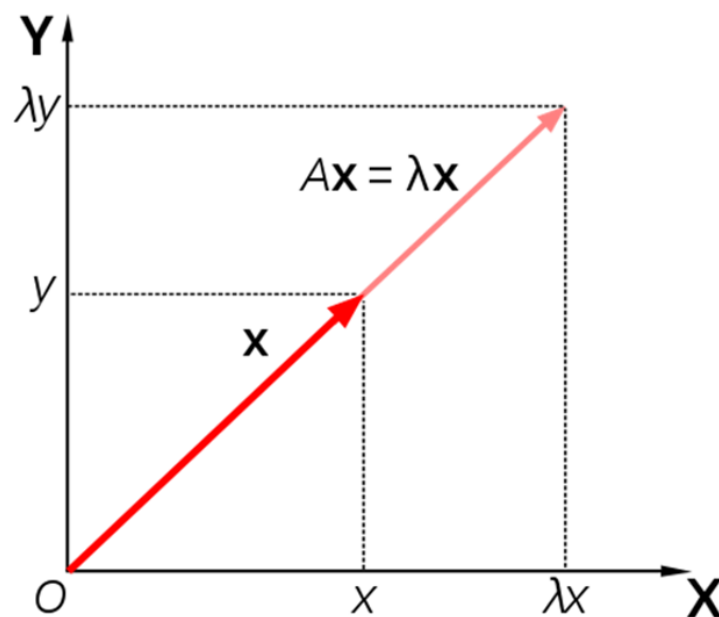


이처럼 대부분의 경우 행렬 A 에 의해 변환되는 벡터는 변환 전의 벡터와 다른 방향과 크기로 변환된다. 그러나 특정한 벡터와 행렬 연산의 경우에는 변환 후에 크기만 바뀌고 방향은 바뀌지 않을 수도 있다. 즉, 변환된 결과 ($A\mathbf{x}$)가 변환 전의 벡터(\mathbf{x})와 **평행(parallel)**한 벡터들이 존재하는 것이다. 예를 들어 벡터 $\mathbf{x} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ (파란색 화살표)에 행렬 $A = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$ 이용해 선형 변환시키면, 아래 그림과 같이 변환 전의 벡터와 평행하지만 크기만 바뀐 벡터가 나온다.



즉, 입력 벡터 \mathbf{x} 를 A 로 선형 변환시킨 결과($A\mathbf{x}$)는 \mathbf{x} 의 상수배라는 것을 의미하며, 이를 수식과 그림으로 표현하면 각각 다음과 같이 나타낼 수 있다.

$$A\mathbf{x} = \lambda\mathbf{x}$$



- 고유값과 고유벡터

앞의 예시에서 본 것과 같이 정방행렬(square matrix) A 에 곱해져서 그 위치나 크기가 바뀌어도 원래 자기 자신과 동일한 방향, 혹은 **평행한 방향**을 갖는 벡터들을 **고유벡터(\mathbf{x})**라고 한다. 이때 변환 전의 벡터와 A 에 곱해져서 변환된 벡터의 크기는 다를 수 있으며, 그 크기는 특정 상수를 곱한 만큼의 차이가 존재한다. 여기서 크기를 나타내는 특정 상수는 λ 로 표현되며 이것이 바로 **고유값**이다. 그리고 이를 한번에 수식으로 나타내면

$$A\mathbf{x} = \lambda\mathbf{x}$$

으로 표현할 수 있는 것이다. 여기서 A 는 고정된 것이고, \mathbf{x} 와 λ 가 두 개의 미지수이며, 수식을 만족시키는 벡터 \mathbf{x} 들이 고유벡터가 된다.

A 는 곱해지는 모든 벡터 \mathbf{x} 들을 변화시키지만, 그 순간에도 변하지 않는 성분이 존재하기 마련이고, 이렇게 선형시스템 A 의 변하지 않는 성분을 고유값과 고유벡터라는 형태로 표현한 것이다.



고유값과 고유벡터

임의의 $n \times n$ 행렬 A (square matrix) 에 대하여, 0 이 아닌 벡터 $\mathbf{x} \in \mathbb{R}^n$ 가 존재하고 적당한 실수 λ 에 대하여 다음을 만족하면

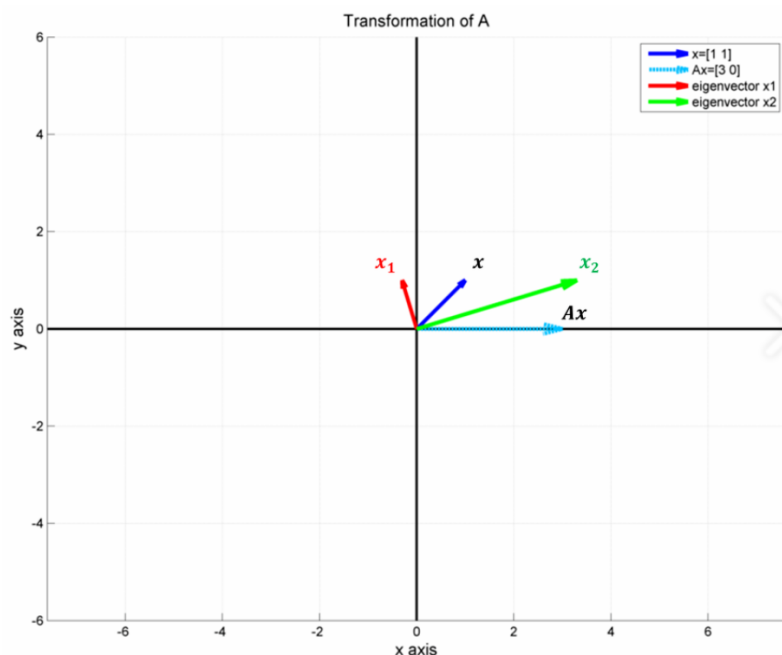
$$(1) A\mathbf{x} = \lambda\mathbf{x}$$

λ 를 A 의 **고유값(eigenvalue)**이라 하고, \mathbf{x} 를 λ 에 대응하는 A 의 **고유벡터(eigenvector)**라고 한다.

예시를 통해서 정확하게 이해해보자. 우선 2×2 크기의 선형시스템 방정식 A 가 있다고 할 때, 이에 대한 선형결합의 결과를 아래의 그림을 통해 설명해보고자 한다.

$$A \mathbf{x} = \mathbf{x}'$$

$$\begin{bmatrix} 2 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x_1' \\ x_2' \end{bmatrix}$$



위의 그림을 보면, A 에 벡터 $\mathbf{x} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ (파란색)를 곱하면 $A\mathbf{x} = \begin{bmatrix} 3 \\ 0 \end{bmatrix}$ 이 되고 하늘색 화살표로 표시된다. 이는 임의의 벡터를 무작위로 선택한 경우로, 대부분의 \mathbf{x} 는 A 에 곱해지면 원래의 벡터와는 다른 방향으로 변환이 이루어진다. 그러나 벡터들 중에서 \mathbf{x}_1 과 \mathbf{x}_2 는 A 에 곱해도 자기 자신과 평행한 벡터들이며, 이는 A 의 고유벡터를 의미한다. 일반적으로 $n \times n$ 크기의 행렬은 n 개의 고유벡터를 갖는다. 이때, \mathbf{x}_1 과 \mathbf{x}_2 가 고유벡터라면 A 를 곱했을 때 방향이 변하지 않는지 직접 곱해보면서 확인해보자.

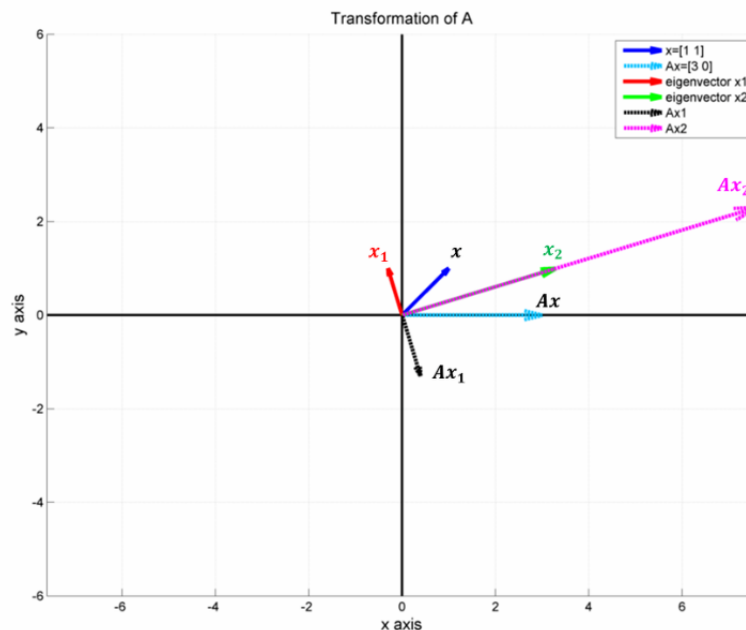
$$\mathbf{x}_1 = \begin{bmatrix} -0.3028 \\ 1 \end{bmatrix}, \mathbf{x}_2 = \begin{bmatrix} 3.3028 \\ 1 \end{bmatrix}$$

A 에 고유벡터들을 각각 곱하면 다음과 같이 결과 값이 나온다.

»

$$A\mathbf{x}_2 = \begin{bmatrix} 2 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 3.3028 \\ 1 \end{bmatrix} = \begin{bmatrix} 7.6056 \\ 2.3028 \end{bmatrix}$$

이 벡터들과 원래의 벡터들의 방향이 같은 지를 알아보기 위해 다음의 그래프를 살펴보자.

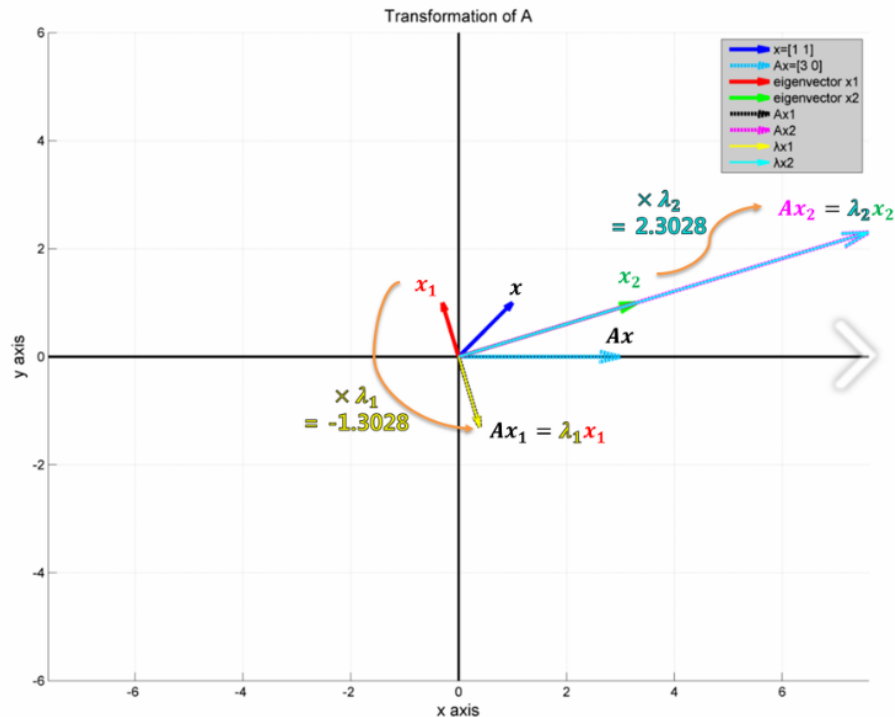


이를 통해서 A 의 고유벡터 \mathbf{x}_1 과 \mathbf{x}_2 가 A 에 의해 변화된 이후 크기와 방향이 다르기도 하지만 그 방향의 축은 변하지 않음을 알 수 있다.

그러나 누군가는 \mathbf{x}_1 에서 나온 결과를 보면서 축은 같지만 반대방향이기 때문에 이것이 고유벡터가 맞는지 의문이 들 수도 있다. 앞에서 정의한 수식에 따르면 \mathbf{x}_1 은 고유벡터가 맞다. 이때, 식을 “행렬 A 에 의해서 변환된 고유벡터 \mathbf{x} 는 원래의 벡터에 어떤 상수 람다를 곱한 것과 같다”라고 해석해 본다면, 이를 통해 아직 우리가 람다인 고유값을 사용하지 않았다는 것을 알 수 있고, 이 때문에 변환된 값이 원래의 값과 다르게 나타난 것이다. 여기에서 \mathbf{x}_1 과 \mathbf{x}_2 의 고유값은 각각

$$\lambda_1 = -1.3028, \lambda_2 = 2.3028$$

이다. 이를 원래의 벡터에 각각 곱해준 결과 값은 다음의 그래프와 같이 나타난다.



이전에 $A\mathbf{x}_1$ 과 \mathbf{x}_1 은 원래 방향이 반대이고 길이도 달랐지만, λ_1 를 곱한 뒤에 이전의 검정색 화살표($A\mathbf{x}_1$)와 노란색 화살표($\lambda_1\mathbf{x}_1$)가 완전히 겹쳐지는 것을 통해서 고유벡터와 고유값이 맞다는 것을 확인할 수 있다. 이는 \mathbf{x}_2 에서도 동일하게 적용된다.

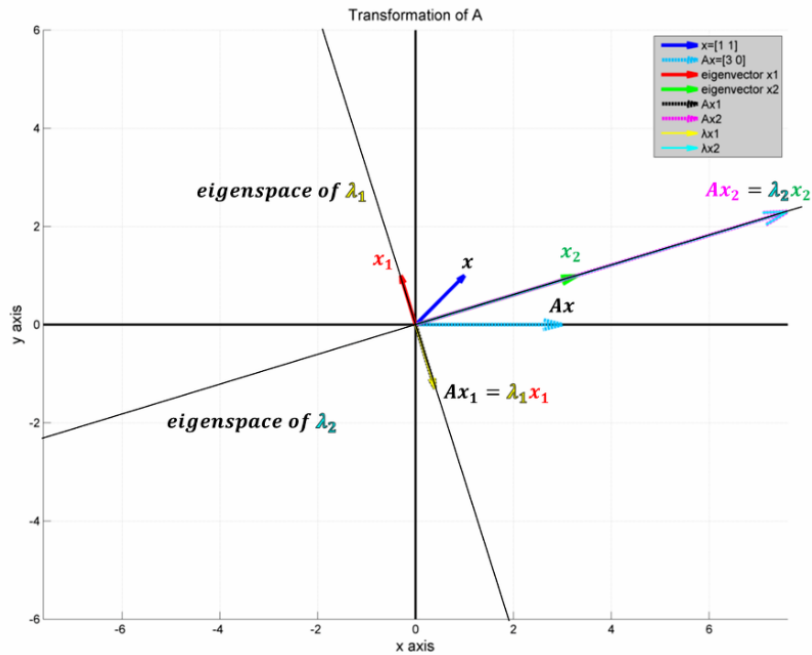
(2) 고유공간(eigenspace)

바로 위의 그래프를 통해서 고유벡터들은 고유값과 곱해지면 A 에 의해 변환된 후의 결과와 같아진다는 것을 보였다. 그렇다면 A 의 고유값과 고유벡터는 오로지 $\mathbf{x}_1, \mathbf{x}_2$ 그리고 λ_1, λ_2 로만 나타날까? 답은 **고유값**은 유일하며, **고유벡터**는 무수히 많다는 것이다. 이를 앞의 식을 활용해서 간단하게 증명해보자.

$$A\mathbf{x}=(c\lambda)\mathbf{x} \quad \cdots(7.1)$$

$$A(c\mathbf{x})=\lambda(c\mathbf{x}) \quad \cdots(7.2)$$

(7.1) 식은 고유값이 유일하지 않음을 가정하면서 고유값에 임의의 상수를 곱해서 만든 식이다. 한쪽에만 임의의 상수를 곱했기 때문에 식은 당연히 성립하지 않는다. 따라서 어떤 행렬 A 의 고유값은 유일함을 알 수 있다. (7.2)식을 보면 고유벡터가 유일하지 않음을 가정하면서 양변의 \mathbf{x} 에 임의의 상수를 각각 곱해주었다. 양변에 동일한 상수를 곱해줬기 때문에 식이 성립할 것이다. 따라서 행렬 A 의 고유벡터는 무수히 많다는 것을 알 수 있다. 이 벡터들은 부분공간을 형성하는데 이러한 공간을 **고유공간(eigenspace)**라고 한다.



위의 그림을 보면 고유값 λ_1 에 대응되는 고유벡터 \mathbf{x}_1 을 기준으로 형성된 직선과 고유값 λ_2 에 대응되는 고유벡터 \mathbf{x}_2 를 기준으로 형성된 직선 2개가 행렬 A 의 부분공간이 된다.

행렬 A 의 고유벡터들은 고유공간 상에 존재하는 수많은 벡터들이다. 다시 정리하면 고유벡터는 변환 전과 변환 후의 벡터가 동일한 부분공간에 존재하는 벡터들을 의미하는 것이다. 그리고 그 변하는 정도를 나타내는 상수값을 고유값이라고 한다.

참고로 $n \times n$ 행렬에서 고유공간의 수는 동일한 고유값에 대응되는 고유벡터들의 집합의 수만큼 정의될 수 있다.

(3) 고유값과 고유벡터의 계산방법

그렇다면 이제는 고유값과 고유벡터를 어떻게 계산하는지 알아보려고 한다. 이는 앞에서 고유값과 고유벡터를 이해하는데 가장 중요했던 식인

$$A\mathbf{x} = \lambda\mathbf{x}$$

을 통해 유도할 수 있다. 이 식을 우변을 좌변으로 이항시키면 다음과 같이 나오는데

$$\begin{aligned} A\mathbf{x} - \lambda\mathbf{x} &= 0 \\ (A - \lambda I)\mathbf{x} &= 0 \dots (*) \end{aligned}$$

(*) 식은 결국 $A\mathbf{x} = 0$ 의 꼴이며, 식 푸는 것은 행렬 A 의 null space를 찾는 과정이 된다. 이때 행렬 A 의 null space가 존재하기 위해서는 행렬 A 가 **특이행렬(singular matrix)**이라는 조건을 반드시 만족해야 한다. 따라서 (*) 식에서 괄호 안에 있는 행렬, 즉 행렬 A 를 람다만큼 이동시킨 행렬이 특이행렬이 되어야 한다는 것을 의미한다. 이때, **특이행렬은 행렬식이 0**이라는 특징을 지니고 있다. 이 특징을 활용하면 아래와 같이 쓸 수 있다.

$$\det(A - \lambda I) = 0$$

이 식을 보면 (*) 식의 미지수가 두개에서 하나(λ)로 줄어든 것을 확인할 수 있다. 따라서 위의 식을 풀면
 람다, 즉 고유값을 구할 수 있게 된다.

고유값을 구한 후에는 고유벡터를 구해야 한다. 고유벡터는 (*) 식에서 null space 를 찾는 일이기 때문에
 앞선 과정에서 구한 고유값을 각각 대입하여 가우스 소거를 통해 null space 를 찾으면 고유값에 대한 고
 유벡터를 구할 수 있다.

지금까지는 고유값과 고유벡터를 구하는 흐름에 대해 설명하였고 이제 예시를 통해 직접 계산해보고자 한
 다.

»

이 있다고 하면, 고유값과 고유벡터의 정의에 따라 다음의 식을 만족한다.

$$\begin{aligned} A\mathbf{x} &= \lambda\mathbf{x} \\ (A - \lambda I)\mathbf{x} &= 0 \dots (*) \end{aligned}$$

먼저 고유값을 구해보자

$$\begin{aligned} \det(A - \lambda I) &= 0 \\ \det\left(\begin{bmatrix} 2 - \lambda & 1 \\ 1 & 2 - \lambda \end{bmatrix}\right) &= 0 \\ \Rightarrow (2 - \lambda)^2 - 1 & \\ = (4 - 4\lambda + \lambda^2) - 1 & \\ = \lambda^2 - 4\lambda + 3 = 0 & \end{aligned}$$

이차방정식의 해는 $\lambda_1 = 1$, $\lambda_2 = 3$ 이기 때문에 행렬 A 의 고유값은 1과 3 이다.

그 다음으로는 고유벡터를 구해야 하는데 우선 $\lambda_1 = 1$ 인 경우에서 고유벡터 \mathbf{x}_1 을 구해보자. (*) 식에 그
 대로 대입하면 다음과 같다.

$$\begin{aligned} \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} &= 1 \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \\ \Rightarrow \begin{bmatrix} 2 - 1 & 1 \\ 1 & 2 - 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} &= 0 \end{aligned}$$

가우스소거법을 통해 null space를 찾아 고유벡터를 구하면 $\mathbf{x}_1 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$ 이 나온다. 같은 방식으로

$\lambda_2 = 3$ 인 경우를 구하면 고유 벡터는 $\mathbf{x}_2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ 이다.

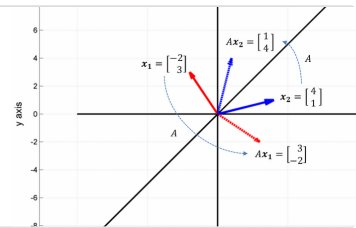
(4) 여러가지 행렬의 고유값과 고유벡터

일반적인 행렬 외에 몇 가지 특수한 형태의 행렬에서는 고유값과 고유벡터에 대한 특별한 성질들이 존재한다. 이번 스터디에서는 다음 파트에서 대칭행렬을 소개할 것이다. 혹시 치환행렬, 회전행렬, 삼각행렬에 대한 고유값과 고유벡터에 대해 더 알아보고 싶다면 아래 레퍼런스를 참고!

[Linear Algebra] Lecture 21-(2) 고유값(eigenvalues)과 고유 벡터(eigenvectors)

지난 강의 에 이어 고유값(eigenvalue)과 고유벡터(eigenvector)에 대한 두 번째 강의 다. 이번 포스팅에서는 고유값과 고유벡터에 대한 실제 계산법과 유용한 공식, 그리고 여러 가지 행렬들에 대한 고유값/고유벡터에 대해 알아보도록 하겠다. 지난 첫 번째 강의에서 주로 개념 위주의

☞ <https://twlab.tistory.com/47?category=668741>



2.3.2 고유값 분해

(1) 고유값 분해

대부분의 $n \times n$ 행렬 A 에 대해서는 일부 경우를 제외하고는 아래의 식을 만족하는 n 개의 고유값과 n 개의 고유벡터를 얻을 수 있다.

$$Av_i = \lambda_i v_i \text{ for } i = 1, 2, \dots, n$$

그 다음으로 모든 고유벡터들을 열벡터로 두면 행렬 V 를 다음과 같이 구할 수 있다.

$$V = \begin{bmatrix} | & | & \cdots & | \\ v_1 & v_2 & \cdots & v_n \\ | & | & \cdots & | \end{bmatrix}$$

이 때, 식 $Ax = \lambda x$ 를 행렬 A 와 V , 상수 λ 이용하여 나타낸다면 다음과 같이 표현할 수 있다.

$$AV = \begin{bmatrix} | & | & \cdots & | \\ \lambda_1 v_1 & \lambda_2 v_2 & \cdots & \lambda_n v_n \\ | & | & \cdots & | \end{bmatrix}$$

여기서 λ_i for $i = 1, 2, \dots, n$ 은 실수이기 때문에 행렬 AV 는 다음과 같이 열벡터 행렬과 대각행렬로 인수분해 할 수 있다.

$$AV = \begin{bmatrix} | & | & & | \\ v_1 & v_2 & \cdots & v_n \\ | & | & & | \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & 0 & 0 \\ & & \ddots & \\ 0 & 0 & \cdots & \lambda_n \end{bmatrix}$$

고유값들을 대각성분에 모아둔 행렬을 Λ 로 다음과 같이 나타낸다면

$$\Lambda = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & 0 & 0 \\ & & \ddots & \\ 0 & 0 & \cdots & \lambda_n \end{bmatrix}$$

위의 식을 다음과 같이 나타낼 수 있고,

$$\Rightarrow AV = V\Lambda$$

만약 모든 고유벡터들이 선형독립이라면 행렬 A 를 다음과 같이 나타낼 수 있다.

$$\Rightarrow A = V\Lambda V^{-1}$$

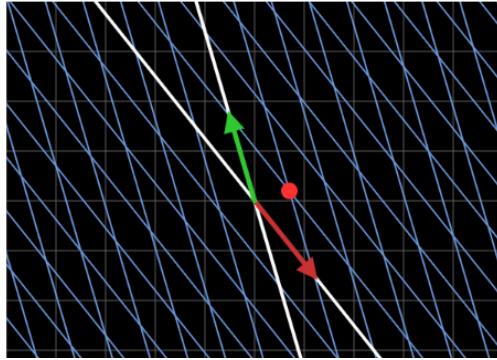
이는 행렬 A 를 고유벡터 행렬인 V 와 고유값 행렬인 Λ , 그리고 고유벡터행렬의 역행렬의 곱으로 정의한 것이다. 이는 행렬을 분해하는 방법들 중 하나이며, 이러한 대각화 분해를 **고유값 분해 (eigendecompoistion)**라고 한다.

(2) 고유값 분해의 기하학적 의미

고유값 분해의 기하학적인 의미를 설명하기 위해 예시를 들어보고자 한다.

$$A = \begin{bmatrix} 1.2 & -0.5 \\ -1.5 & 1.7 \end{bmatrix}$$

우선 행렬 A 의 선형변환을 시각화하면 아래와 같다고 할 때



행렬 A 의 고유값과 고유벡터는 아래와 같다.

»

이를 행렬 A 를 고유값 분해할 때 A 를 구성하는 성분들을 행렬로 나타내면 다음과 같다.

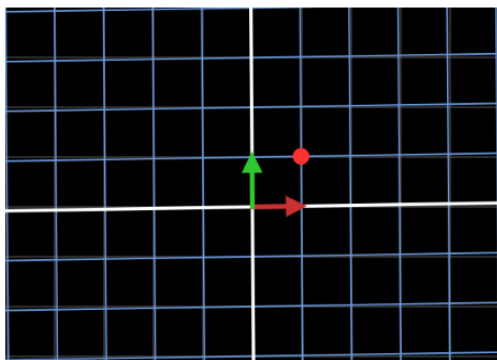
$$V = \begin{bmatrix} 0.6089 & -0.3983 \\ 0.7933 & 0.9172 \end{bmatrix}$$

$$\Lambda = \begin{bmatrix} 0.5486 & 0 \\ 0 & 2.3514 \end{bmatrix}$$

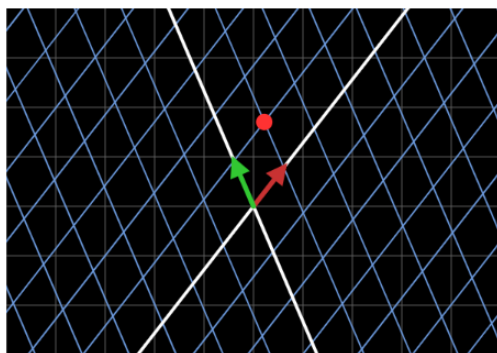
$$V^{-1} = \begin{bmatrix} 1.0489 & 0.4555 \\ -0.9072 & 0.6963 \end{bmatrix}$$

따라서 행렬 A 는 아래와 같이 분해하여 나타낼 수 있다.

»



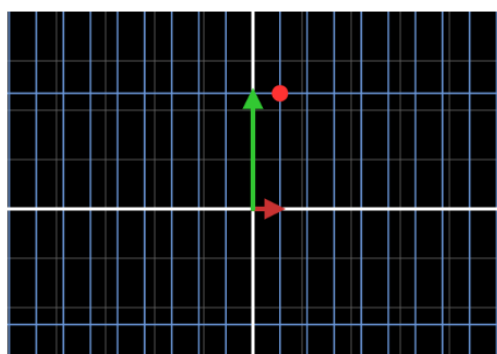
먼저 고유벡터로 구성된 행렬 $V = \begin{bmatrix} 0.6089 & -0.3983 \\ 0.7933 & 0.9172 \end{bmatrix}$ 의 선형변환을 살펴보자



행렬 V 의 선형변환은 회전과 유사한 형태를 보인다. 왜냐하면 V 에 들어간 열벡터들은 모두 고유벡터이고, 이 고유벡터는 방향만을 표시하기 때문에 그 길이는 1이 되기 때문이다. 따라서 원점에서부터 시작하는 길이가 1인 벡터 두 개가 서로 다른 방향을 보여주고 있으므로 회전 변환과 유사한 역할을 한다.

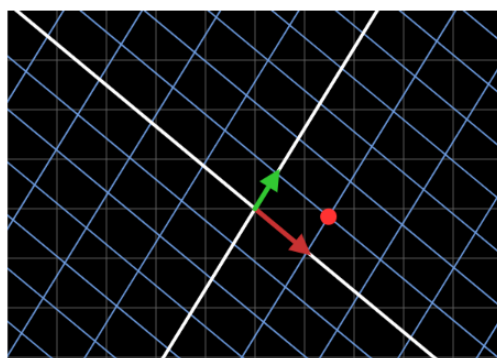
다만 변환 후의 기저 벡터의 길이가 모두 같은 것은 아니고 변환 시 뒤집어 질 가능성이 있기 때문에 완전히 회전변환과 같다고는 할 수 없다. 또한 변환 후 기저벡터들의 각도는 90도를 이루지 않는다.

그리고 다음은 $\Lambda = \begin{bmatrix} 0.5486 & 0 \\ 0 & 2.3514 \end{bmatrix}$ 의 선형변환이다.

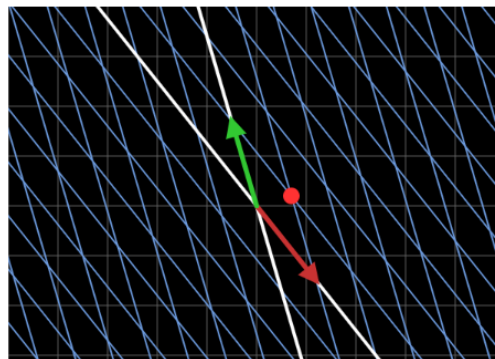
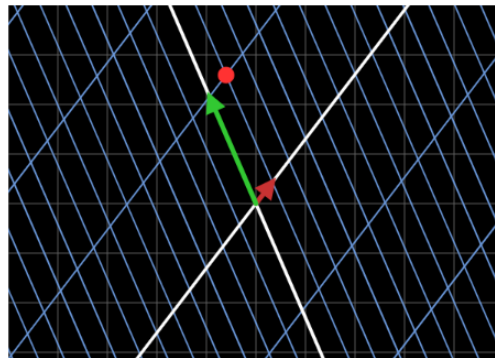
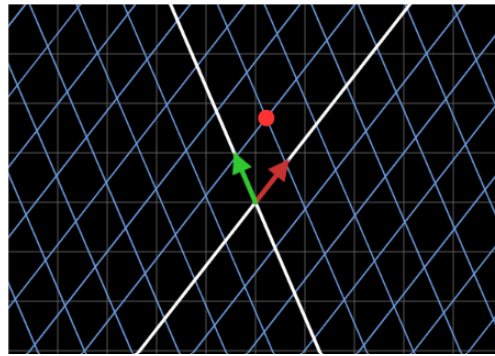


대각 성분만 존재하기 때문에 위 아래, 혹은 양 옆으로 늘어나거나 줄어드는 변환만 보인다.

$V^{-1} = \begin{bmatrix} 1.0489 & 0.4555 \\ -0.9072 & 0.6963 \end{bmatrix}$ 의 선형변환은 V 와 비슷하게 회전과 유사한 형태를 보인다. 다만 회전 방향이 V 와 반대이다.



따라서 V, Λ, V^{-1} 의 선형 변환을 차례로 적용하면 A 의 선형변환과 같은 것을 알 수 있다.



(3) 대칭행렬의 고유값 분해

대칭 행렬의 고유값 분해는 일반정방행렬의 고유값 분해와 비교했을 때 특이한 성질을 보인다. 이는 실원소(real-valued) **대칭행렬**은 항상 고유값 대각화가 가능하며 **직교행렬**(orthogonal matrix)로 대각화(분해)가 가능하다는 것이다.

대칭행렬은 $A = A^T$ 를 만족하는 행렬인데 만약 행렬 A 가 고유값 분해를 할 수 있다면 아래의 내용이 성립한다.

$$A = V\Lambda V^{-1} = A^T = (V\Lambda V^{-1})^T = (V^{-1})^T \Lambda^T V^T$$

여기서, Λ 는 대각행렬이기 때문에 $\Lambda^T = \Lambda$ 이고, $(V^{-1})^T = (V^T)^{-1}$ 이 성립하기 때문에

$$V\Lambda V^{-1} = (V^T)^{-1}\Lambda V^T$$

으로 나타낼 수 있다. 그리고 다음을 만족하기 때문에

▮

대칭행렬의 고유벡터를 모아둔 행렬 V 는 **직교행렬**이다. 따라서 모든 대칭행렬은 아래와 같이 직교행렬을 이용하여 고유값 분해가 가능하다.

$$A = V\Lambda V^T$$

일반적으로 대칭행렬의 고유값을 모아둔 행렬은 V 대신에 Q 를 사용하기 때문에 대칭행렬 A 의 고유값 분해는

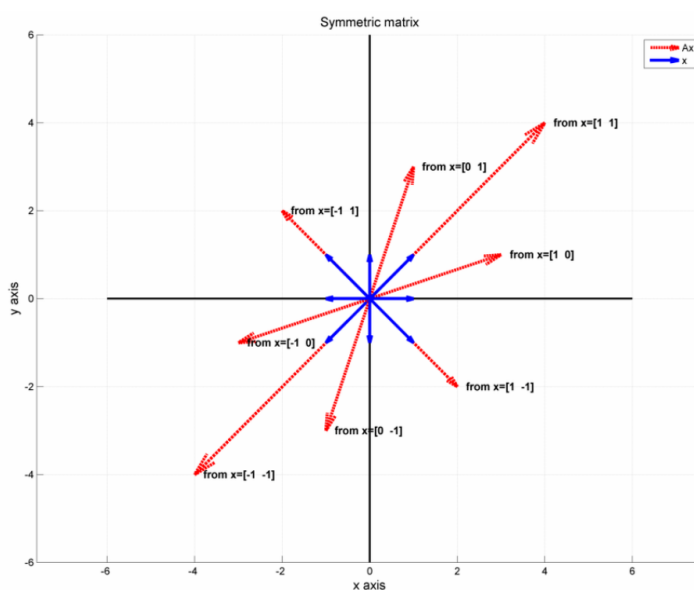
$$A = Q\Lambda Q^T$$

으로 나타낸다.

대칭행렬의 고유값 분해는 선형대수에서 중요한 성질인데 이러한 성질은 특이값 분해(SVD), 주성분 분석(PCA) 등에서 가장 기본이 되는 성질로 활용되기 때문이다. 따라서 대칭행렬은 항상 고유값분해가 가능하며 그리고 직교행렬로 대각화가 가능함을 기억하자.

참고

대칭행렬의 예시로 $A = \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix}$ 는 아래와 같이 행렬 변환을 하게 된다.



이를 통해 A 는 원래 벡터를 원점을 중심으로 양쪽 대각선 방향으로 잡고 늘리는 변환을 수행하는 것을 알 수 있다.

2.4. 고유값과 고유값 분해 응용 및 차원 축소

원래는 eigenvector decomposition의 응용파트인 PCA만 다룰려다, PCA와 함께 거론되는 ICA, 다른 차원축소기법으로 자주 언급되는 LDA도 함께 다뤄볼 예정이다. 우선은 PCA부터 보도록 하자.

2.4.1. PCA

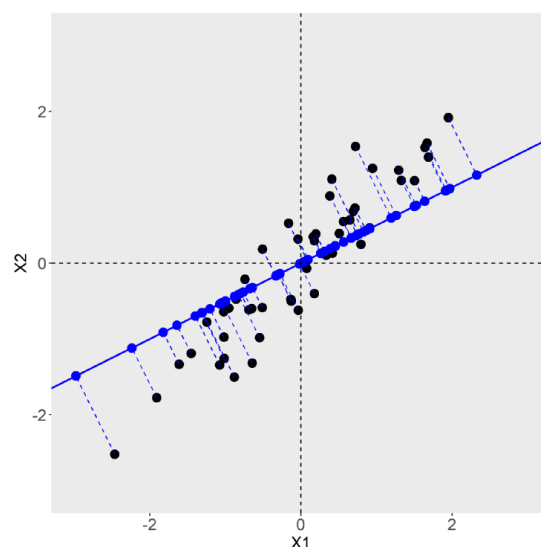
데이터 분석을 진행할 때 우리는 “적절한 변수의 개수”가 몇 개인지를 항상 주의깊게 봐야한다. 데마 1주차 클린업에서 이미 보았듯이, 변수의 개수가 많아지게 되면 데이터가 표현되는 공간이 기하급수적으로 넓어지며, 그만큼 유의미한 학습을 진행하기 위한 데이터의 수도 급격하게 증가한다. 이러한 연유로 충분치 않은 데이터 개수를 갖고 있을 때에는 차원축소가 더욱 중요한 과제로 다가오게 된다.

차원축소라고 하면 1) Feature selection, 2) Feature extraction으로 나뉘볼 수 있는데, 오늘 우리는 eigenvector decomposition의 응용파트를 주의깊게 보기로 하였으므로 후자인 Feature extraction의 대표적인 기법인 PCA를 다뤄보도록 하겠다.

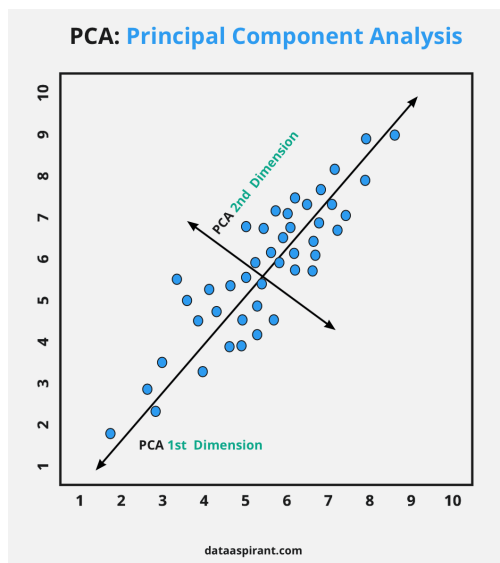
우선 PCA에 들어가기 전, 이해를 돕기 위한 간단한 예시를 먼저 봐보도록 하겠다.

PCA의 가장 큰 핵심은, 차원축소를 하되, 데이터의 구조를 잘 유지하면서 축소를 하자는 것이 가장 큰 핵심이다. 그렇다면 이때 데이터의 구조를 잘 유지한다는 것이 어떤 의미일까? 아래의 사진을 봐도록 하자.

PCA는 origin을 지나는 subspaces(n 차원에서 서로 독립인 $n-1$ 개의 basis로 spanned된 공간)에 데이터를 정사영시킴으로써 차원축소를 한다고 보아도 된다.



그때, 아래의 사진처럼 두 종류의 subspaces(1st dimension, 2nd dimension)가 있다고 가정하자. 1st dimension으로 데이터를 정사영시켰을 시에는 데이터간의 분산이 크고, 2nd dimension으로 데이터를 정사영시켰을 때는 데이터의 분산이 작게 정사영이 되는 것을 볼 수 있다. 이때, 데이터의 구조를 잘 유지하면서 정사영되었다고 말하는 경우는 1st dimension을 활용한 경우이다. 데이터간의 분산이 크다는 것은 곧 데이터들간의 차이가 명확하다고 볼 수 있고, 이는 곧 데이터들을 잘 설명할 수 있음을 의미하기 때문이다. 즉, 데이터간의 분산을 데이터의 설명력이라고 보면 이해하기 쉬울 것이다!



그렇다면 이제는 데이터를 어떤 벡터에 내적(혹은 정사영)하는 것이 최적의 결과를 내주는가에 대한 고민만 하면 된다. 그리고 이때 사용되는 것이 바로 eigenvector decomposition이다.

PCA 작동 원리

본격적으로 원리를 살펴해보도록 하자.

① 우선 데이터들을 정규화시킨다.

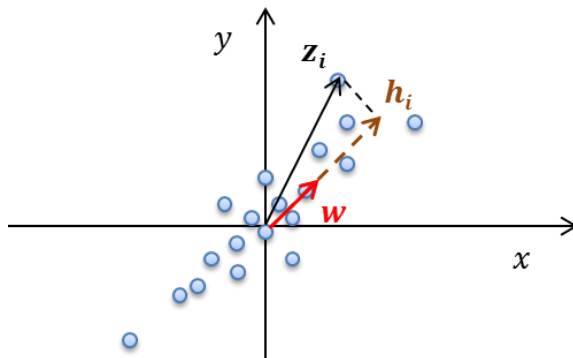
$$x_j^{(i)} \leftarrow \frac{x_j^{(i)} - \mu_j}{\sigma_j}$$

$$\mu_j = \frac{1}{n} \sum_{i=1}^n x_j^{(i)}$$

$$\sigma_j = \frac{1}{n} \sum_{i=1}^n (x_j^{(i)} - \mu_j)^2$$

데이터들간의 단위가 다를 수 있으므로 단위를 상쇄시키기 위해 정규화를 시킨다.

② 데이터 분산을 최대화시키는 subspaces를 찾자.



정사영 시킨 후의 데이터들간의 분산을 최대화한다는 것은 origin과 정사영된 점들간의 거리의 제곱을 최대화시키겠다는 것과 동일한 의미이다. $\nu \in \mathbb{R}^d$ 인 unit vector ν 가 있다고 가정하자. 이때 unit vector ν 에 정사영된 vector $x^{(i)}$ 와 원점간의 거리는 $proj(\nu)x^{(i)}$ 로 표현할 수 있다. ($\nu \in \mathbb{R}^d$ 인 unit vector ν 의 dot product는 1이다. $\rightarrow \nu^\top \nu = 1$)

즉 우리는 분산을

$$\frac{1}{n} \sum_{i=1}^n \|proj(\nu)x^{(i)}\|^2 = \frac{1}{n} \sum_{i=1}^n \|(x^{(i)}\nu)\vec{\nu}\|^2$$

으로 표현할 수 있고,

$$\arg \max_{\nu} \frac{1}{n} \sum_{i=1}^n \|(x^{(i)}\nu)\vec{\nu}\|^2$$

를 최대화하는 ν 를 찾아야 한다.

더 정리를 이어가보겠다. 자세한 과정은 적어두었으니 시간이 나면 살펴봐도 좋다.

$$\begin{aligned}
\frac{1}{n} \sum_{i=1}^n \|(x^{(i)} \nu) \nu\|^2 &= \frac{1}{n} \sum_{i=1}^n (x^{(i)} \nu)^2 \\
&= \frac{1}{n} \sum_{i=1}^n x^{(i)\top} \nu x^{(i)\top} \nu \\
&= \frac{1}{n} \sum_{i=1}^n x^{(i)\top} \nu \nu^\top x^{(i)} \\
&= \frac{1}{n} \sum_{i=1}^n \nu^\top x^{(i)} x^{(i)\top} \nu \\
&= \nu^\top \left(\frac{1}{n} \sum_{i=1}^n x^{(i)} x^{(i)\top} \right) \nu
\end{aligned}$$

이때, $\frac{1}{n} \sum_{i=1}^n x^{(i)} x^{(i)\top}$ 는 covariance matrix Σ 이므로

$$\arg \max_{\nu} \nu^\top \left(\frac{1}{n} \sum_{i=1}^n x^{(i)} x^{(i)\top} \right) \nu = \arg \max_{\nu} \nu^\top \Sigma \nu$$

로 정리할 수 있다.

이 문제를 해결하려면 앞선 스터디에서 배웠던 라그랑지안 form으로 바꿔주면 분산을 최대화하는 ν 를 구할 수 있다.

제약 조건이 $\|\nu\|^2 = 1$ 이고 목적함수인 $\nu^\top \Sigma \nu$ 를 최대화하는 상황에서 라그랑주 승수법을 이용해보자. 이때 보조방정식은 하단과 같다.

$$L = \nu^\top \Sigma \nu - \lambda (\|\nu\|^2 - 1)$$

보조방정식 L 을 ν 로 편미분하면 하단과 같이 정리가 된다.

$$\frac{\partial L}{\partial \vec{\nu}} = 2\Sigma \vec{\nu} - 2\lambda \vec{\nu}$$

$$\Sigma \vec{\nu} = \lambda \vec{\nu}$$

어디서 많이 보지 않았는가? 맞다. 위에서 보았던 eigenvector decomposition 꼴과 같은 꼴이고, 해당 조건을 만족하는 $\vec{\nu}$ 을 구할 수 있다.

즉, $\vec{\nu}$ 는 $x^{(i)\top} x^{(i)}$, Σ 의 eigenvector이고, λ 는 eigenvalue에 해당된다.

③ eigenvector를 이용하여 PC를 구한다.

이때 eigenvector와 eigenvalue는 여러 개가 구해진다. eigenvalue λ_i 에 대응되는 eigenvector ν_i 를 구하고, 이를 내림차순으로 정렬한다.

$$\lambda_1 = 10 \leftrightarrow \nu_1$$

$$\lambda_2 = 8 \leftrightarrow \nu_2$$

$$\lambda_3 = 5 \leftrightarrow \nu_3$$

X 를 i 번째 eigenvector ν_i 에 정사영시킨 것을 i 번째 PC라고하고, $z_i = X\vec{\nu}_i$ 로 표현할 수 있다.

$$z_i = \nu_{i1}x_1 + \nu_{i2}x_2 + \nu_{i3}x_3 + ..$$

이때, 이 z_i 을 기존 변수들을 선형결합하여 새로 탄생한 변수라고 보고, z_i 를 몇 개를 사용함에 따라 차원이 달라지게 된다. (2개를 사용하게 되면 데이터의 차원은 이제 2차원이 되는 것이다.)

$$Var(X\vec{\nu}) = \vec{\nu}^T \Sigma \vec{\nu} = \vec{\nu}^T \lambda \vec{\nu} = \lambda \vec{\nu}^T \vec{\nu} = \lambda$$

추가적으로 λ_i 는 하단의 수식에 의해 eigenvalue가 분산, 즉 설명력을 뜻함을 알 수 있다. 이는 적절한 PC개수를 선택할 때 중요한 기준이 된다.

PC 개수 선택 기준

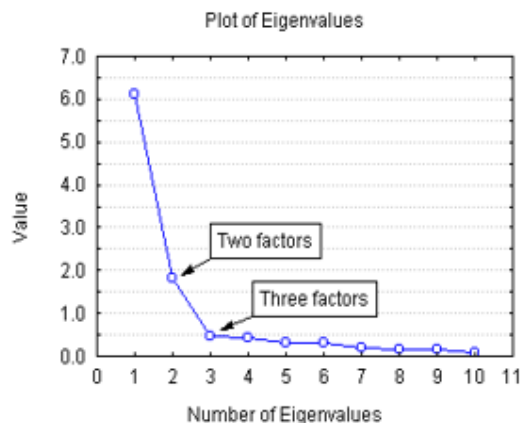
eigenvalue를 내림차순으로 정렬하여 $\lambda_1, \lambda_2, ..$ 로 표현해주자 (여기서 $\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq \lambda_d$)

이 때, 논리적인 방법 중 하나는 전체 데이터의 variance 중 가령 90%만큼을 설명하는 차원까지 감소시켜 주는 것이다. 즉,

$$\frac{m\text{차원까지 축소했을때의 분산}}{\text{전체데이터의 분산}} = \frac{\sum_{i=1}^m \lambda_i}{\sum_{j=1}^d \lambda_j} = 0.9$$

인 적절한 m 을 찾아 그 차원까지 감소시켜주는 것이다. (실제 R을 돌려볼때 각 변수들이 몇 %의 설명력을 갖는지에 대한 것이 기술되어져 있다.)

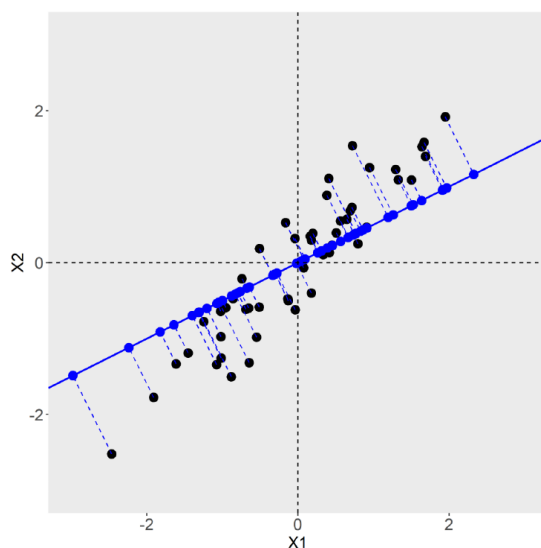
또 다른 방법으로는 scree plot을 이용하는 방법이 있다. 이 방법은 다소 주관적일 수 있는데 2차원 plot을 그리는데 x 축에는 dimensions, y축에는 해당 dimension의 eigenvalue를 기재한다. 예를 들어 다음과 같은 그림일 수 있다.



여기서 보면 세 번째 eigenvalue부터 갑자기 꺾이는 현상이 보인다. 그러면 이때는 3차원까지 차원 감소를 시켜준다는 식으로 결정하는 것이 scree plot을 이용한 방법이다.

PCA 다른 해석

PCA를 정사영한 점들과 origin간의 거리 최대화, 혹은 residual 거리의 최소화라고 보아도 된다.

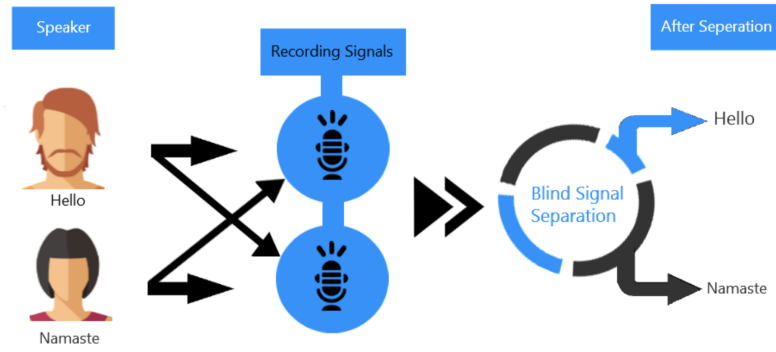


PCA 사용 조건

PCA자체가 상관관계가 있는 변수들을 선형결합하여 변수를 축약하는 기법이기 때문에 PCA는 주로 상관성을 띄는 변수들이 다수 분포할 때 쓰이게 된다. (즉 PCA를 적용하기 전에 heatmap을 그려봐서 각 변수들의 상관성이 존재하는지 보는 것을 추천한다.)

2.4.2. ICA

PCA와 자주 거론되는 ICA에 대해 봐보도록 하겠다. ICA는 독립성분분석으로, 음성인식에서 패턴을 인식할 때 자주 쓰인다고 한다.



방에서 두사람이 동시에 말을 하고 있고, 마이크도 동시에 두개가 존재한다고 생각해보자. 이때 두 개의 마이크로 녹음된 신호를 각각 $x_1(t)$, $x_2(t)$ 라고 하고 음성신호를 각각 $s_1(t)$, $s_2(t)$ 라고 하면, 이 식들의 관계를 하단과 같이 표현 가능하다. 이때 a_{ij} 는 마이크로부터 발화자까지의 거리와 관련된 변수라고 하자.

$$x_1(t) = a_{11}s_1(t) + a_{12}s_2(t)$$

$$x_2(t) = a_{21}s_1(t) + a_{22}s_2(t)$$

우리가 원하는 것은 두 녹음 음원으로부터 원래의 음원을 분리해내는 일을 해야한다. 즉 $s_1(t)$, $s_2(t)$ 을 복원해야한다. 이를 식으로 표현하면 아래와 같다.

$$x = As$$

이때 A 가 무엇인지만 알면 구하고자하는 s 를 쉽게 구해낼 수 있다.

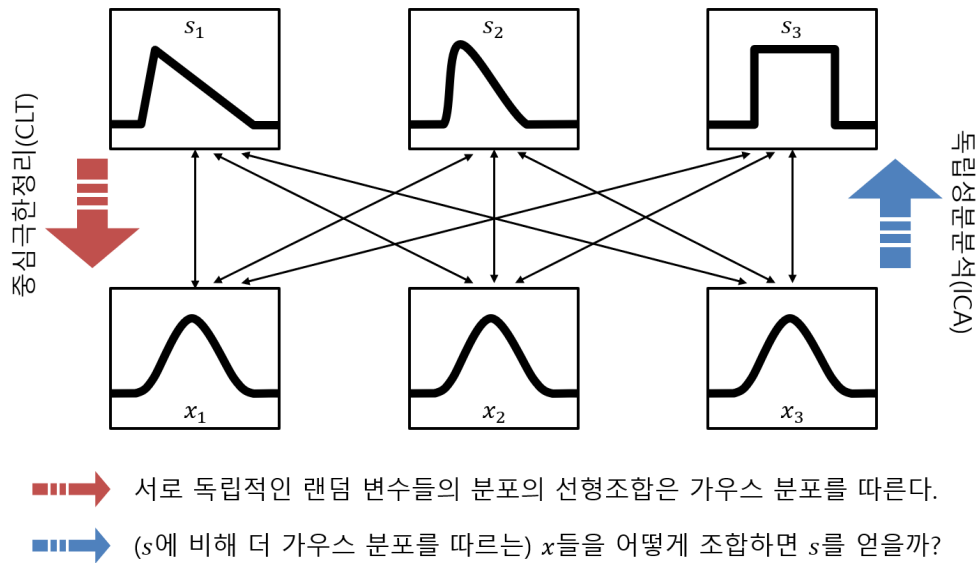
$$s = A^{-1}x = Wx$$

결국 우리의 목적은 $W = A^{-1}$ 찾는 것이다. 과연 어떻게 찾을 수 있을까?

이에 대해 공부하기 이전에 몇가지 개념을 보고 넘어가야 한다.

1) 독립성분분석

중심극한정리(CLT)와 독립성분분석(ICA) 관계 개요도



ICA는 독립 랜덤 변수들의 조합으로 얻어진 x 에 적절한 행렬 $W = A^{-1}$ 을 곱해 원래의 독립 랜덤 변수들인 source들인 S 를 찾는 과정이다.

다시 말해 ICA에서는 **source들이 서로 독립적이라는 가정을 최대한 만족할 수 있도록하는 $W = A^{-1}$ 을 찾는 것**을 목적으로 한다고 생각할 수 있다.

2) Densities and linear transformation

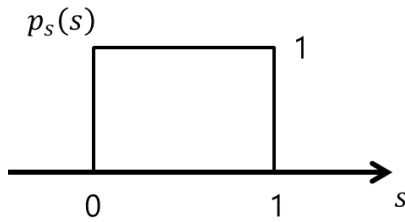
독립성분분석에 대한 알고리즘을 이해하기 위해 필요한 마지막 개념으로 랜덤변수에 선형변환을 적용했을 때 변환 적용 전후 변수의 확률밀도간의 관계를 생각해보자. 확률밀도함수(cdf)의 면적의 총합은 1이 되어야 하므로 어떤 행렬로 random variable를 선형변환해주면 그 확률밀도 함수는 행렬식을 이용해 보정해주어야 한다.

$$s \longrightarrow \boxed{A} \longrightarrow x$$

Ex) $A = 2$

$$s: [0, 1]$$

$$p_s(s) = 1\{0 \leq s \leq 1\}$$



$$x: [0, 2]$$

$$p_x(x) = (0.5)1\{0 \leq x \leq 2\}$$



따라서, $x = As$ 이고, $s = A^{-1}x = Wx$ 라는 점을 생각했을 때, 아래와 같이 선형 변환 후의 random variable의 확률밀도함수를 표시할 수 있다. 다시 쓰자면 아래와 같은 방식으로 $p_x(x)$ 를 구할 수 있다.

$$\begin{aligned} p_x(x) &= p_s(A^{-1}x) \cdot |A^{-1}| \\ &= p_s(Wx) \cdot |W| \end{aligned}$$

▼ transformation of random variable

$$\begin{aligned} P(y < Y \leq y + dy) &= P(x < X \leq x + dx) \\ \rightarrow f_Y(y)|dy| &= f_X(x)|dx| \rightarrow f_Y(y) = \frac{f_X(x)}{\left|\frac{dy}{dx}\right|} \rightarrow f_Y(y) = \frac{f_X(x)}{\left|\frac{dg}{dx}\right|} \\ \therefore f_Y(y) &= \frac{f_X(x)}{|g'(x)|} \Big|_{x=g^{-1}(y)} \end{aligned}$$

ICA algorithm: Bell-sejnowski algorithm

각 source s_i 의 density function을 p_s 라고 하면, 모든 source로 구성된 joint distribution은 다음과 같다.

$$p(s) = \prod_{i=1}^n p_s(s_i)$$

바로 앞에서 확인한 바와 같이 $s = A^{-1}x = Wx$ 라는 관계에서 $p(x)$ 를 구하면,

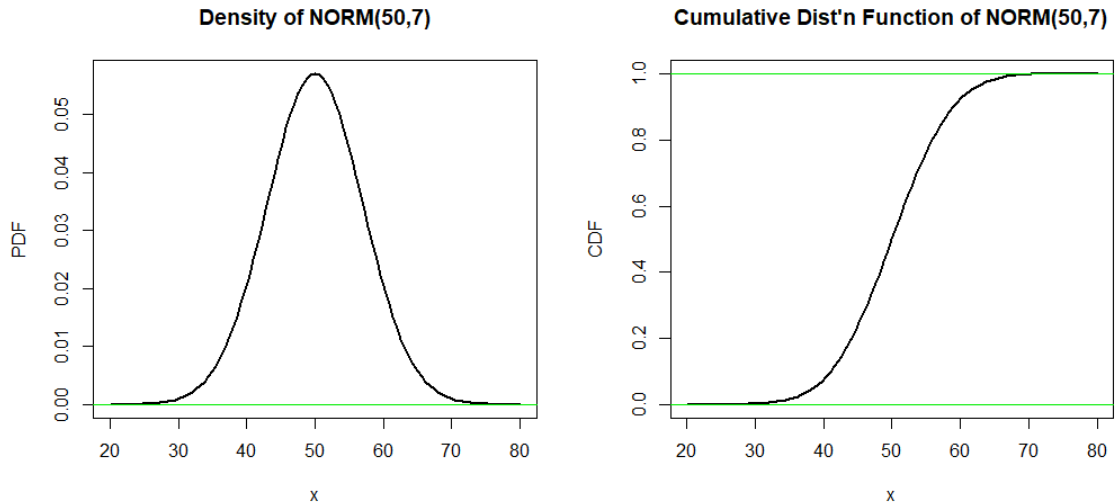
$$p(x) = \prod_{i=1}^n p_s(w_i^\top x) \cdot |W|$$

이제 우리는 해당 식에서 W 에 대한 값을 찾으려 한다. 그 방법으로는 최대우도법, 즉 mle를 사용하도록 하겠다. 위 식의 log likelihood는 아래와 같다.

$$l(w) = \sum_{i=1}^m \left(\sum_{j=1}^n \log p_s(w_j^\top x^{(i)}) + \log |W| \right)$$

또, 여기서 우리는 p_s 에 대한 구체적인 확률밀도함수를 결정해볼 수 있다. 일반적으로의 CDF로 $g(s) = \frac{1}{(1+e^{-s})}$ (sigmoid 함수)를 사용했을 때 알고리즘의 사용에 문제가 없다고 알려져 있다.

▼ CDF: sigmoid



(만약 source의 density에 대한 사전지식이 있다면 해당 density function을 사용하면 된다.)

따라서, pdf인 $p(s)$ 는 $g'(s)$ 이므로, $l(s)$ 는 다음과 같이 쓸 수 있다.

$$l(w) = \sum_{i=1}^m \left(\sum_{j=1}^n \log g'(w_j^\top x^{(i)}) + \log |W| \right)$$

이제, $\frac{d \log |W|}{d|W|} = \frac{1}{|W|} |W| (W^{-1})^\top$ 라는 사실을 이용하여 l 에 대해 편미분 해보자.

$$\begin{aligned} \frac{\partial l}{\partial W} &= \sum_{i=1}^m \left(\sum_{j=1}^n \frac{1}{g'(s_j)} g''(s_j) x^{(i)\top} + \frac{1}{|W|} |W| (W^{-1})^\top \right) \\ &= \sum_{i=1}^m \left(\sum_{j=1}^n (1 - 2g(s_j)) x^{(i)\top} + (W^{-1})^\top \right) \end{aligned}$$

Gradient Ascent 방법을 이용하면 W 는 아래와 같이 update 할 수 있다.

$$W = W + \alpha \frac{\partial l}{\partial W}$$

따라서, 각 $j=1, 2, \dots, n$ 에 대해,

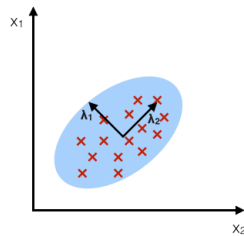
$$W = W + \alpha \begin{bmatrix} 1 - 2g(w_1^\top x^{(i)}) \\ 1 - 2g(w_2^\top x^{(i)}) \\ \dots \\ 1 - 2g(w_n^\top x^{(i)}) \end{bmatrix} x^{(i)\top} + (W^{-1})^\top$$

위의 알고리즘이 수렴한 뒤에 얻은 W 를 이용하면 원래의 sources인 s 를 얻을 수 있게 된다.

2.4.3. LDA(Linear Discriminant Analysis)

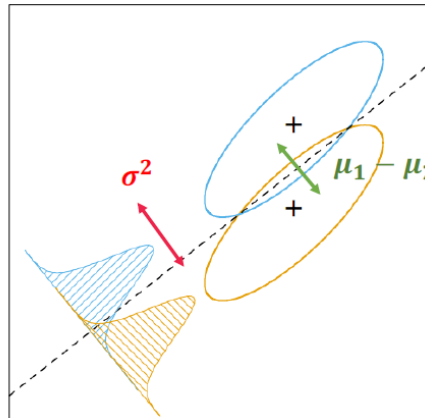
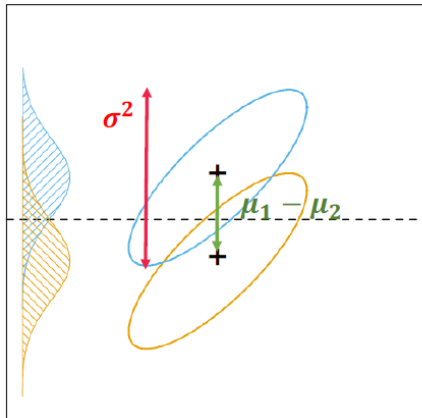
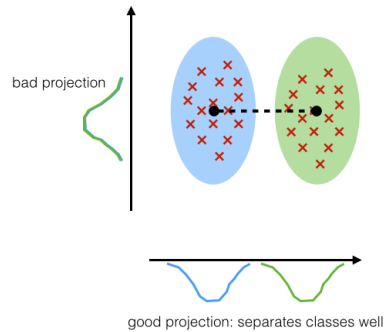
PCA:

component axes that maximize the variance



LDA:

maximizing the component axes for class-separation

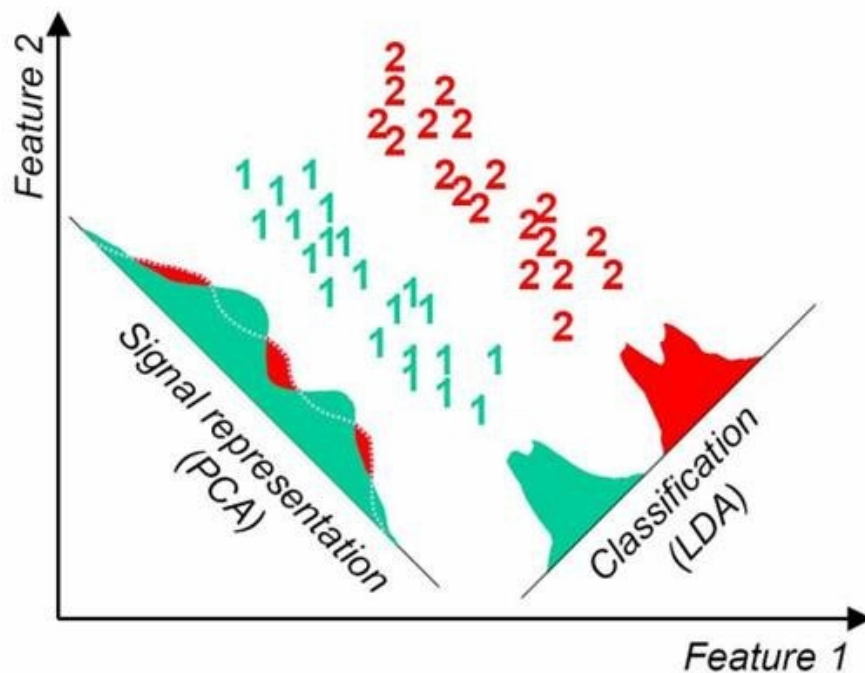


LDA란 데이터가 주어졌을 때 이를 특정 기저에 정사영을 시켜서 데이터를 분류해내는 모델이라고 보면 된다. (LDA하면 토픽모델링 기법인 Latent Dirichlet Allocation가 자주 등장하는데, 이와는 완전히 다른 분류 모델이다.)

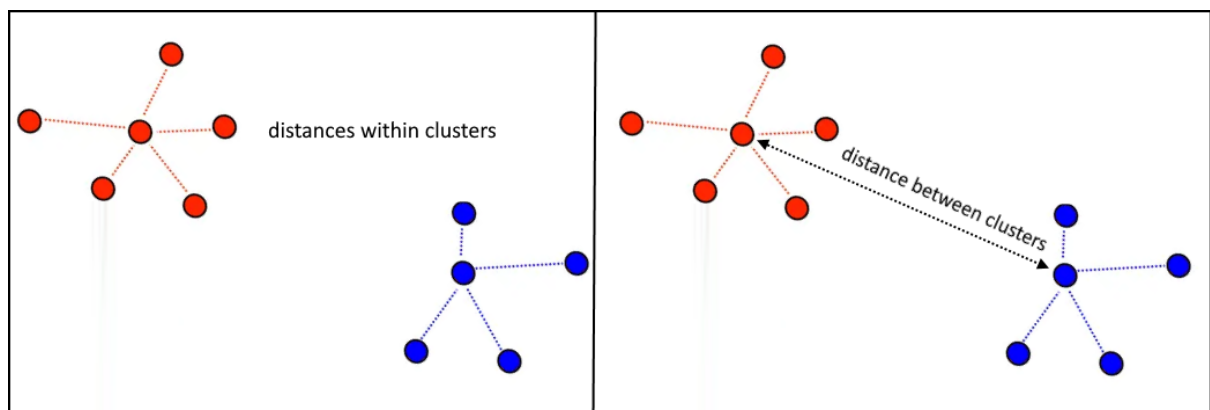
LDA는 데이터를 차원축소시키면서 Y값에 대한 분류를 극대화하려는 것이 목표이다. 예를 들어, 왼쪽과 같은 경우는 정사영시킬 시 데이터 간의 분산은 크지만 주황색 데이터와 파란색 데이터 분포 간의 중복된

파트가 큰 경우를 보아 분류가 잘 되지 않았음을 알 수 있다. 반대로 오른쪽과 같은 경우에는 정사영시킨 후의 데이터 간의 분산이 상대적으로 작다고 볼 수 있지만 Y값이 분류되게끔 정사영되어 LDA는 최종적으로 오른쪽과 같은 정사영 기저를 택할 것이다.

정사영시켜 차원 축소한다는 점에서는 PCA와 굉장히 유사한 것 같은데, 과연 어떤 점이 다른 것일까? 이미 눈치챈 분들도 있겠지만, PCA는 정사영시킬 기저를 정할 때 “분산이 가장 큰 쪽으로” 데이터가 분포하게끔 하는 것이 목표이다. 그렇지만 LDA는 projection 이후에도 두 범주가 잘 분류되게끔 하는 것이 목표이다. 그래서 하단의 사진과 같은 차이가 등장하게 되는 것이다.



LDA에 대해 본격적으로 알아가기에 앞서 clustering에서도 몇 번 등장한 Between distance, within distance에 대한 개념을 알아야 한다.



오른쪽 사진에 보이는 것처럼 between distance란 각 범주간 데이터의 중심인 centroid간의 거리를 말한다. (군집 간 거리). within distance란 반대로 각각 객체들과 해당 범주의 centroid간의 거리, 즉 군집 내 거리를 나타낸 것이다.

만약 데이터를 잘 분리를 해내려면 clustering의 objective와 비슷하게 between distance는 최대화, within distance는 최소화하는 방향으로 학습이 이루어져야만 한다.

본격적으로 LDA의 수식에 대해 알아보도록 하겠다. 우리는 Fisher's의 LDA를 기준으로 설명할 것이다.

$$y = w^T x \quad m_1 = \frac{1}{N_1} \sum_n x_n \quad m_2 = \frac{1}{N_2} \sum_n x_n$$

y 는 x 와 w 에 내적한 것이고, m_1 은 첫번째 군집의 centroid, m_2 는 두번째 군집의 centroid이다.

이때, 첫번째 목적 함수는 projection 이후 Between distance를 최대한 크게 만드는 것과 관련되어 있다.

$$M_2 - M_1 = w^T (m_2 - m_1) \Rightarrow M_k = w^T m_k$$

두번째 목적 함수는 projection 이후 Within distance를 최대한 작게 만드는 것과 관련되어 있다.

$$S_k = \sum_n (y_n - m_k)^2$$

이 두 가지를 모두 고려하여 $J(w)$ 라는 새로운 목적 함수가 탄생하였다.

$$J(w) = \frac{(m_2 - m_1)^2}{s_1^2 + s_2^2} = \frac{W^T S_B W}{W^T S_w W}$$

$$S_B = (m_2 - m_1)(m_2 - m_1)^T$$

$$S_W = \sum_n (x_n - m_1)(x_n - m_1)^T + \sum_n (x_n - m_2)(x_n - m_2)^T$$

$J(w)$ 의 분자에는 군집 간의 거리, 분모에는 군집 내의 거리이고, 군집 간의 거리는 증가시키고 군집 내의 거리는 줄이는 방향으로 학습해야 하므로 $J(w)$ 를 최대화시키는 w 를 찾아내면 된다.

목적함수인 $J(w)$ 를 w 로 편미분하게 되면

$$\frac{\partial J(w)}{\partial w} = \frac{(W^T S_B W) S_w W - (W^T S_w W) S_B W}{W^T S_w W * W^T S_w W} = 0$$

꼴이 나오고, 해당 식을 0으로 만드는 w 를 찾으면 된다. 이때 분모는 0이 되면 안되므로

$$\begin{aligned} (W^T S_B W) S_w W - (W^T S_w W) S_B W &= 0 \\ \Downarrow \\ (W^T S_B W) S_w W &= (W^T S_w W) S_B W \end{aligned}$$

가 성립하게 된다. 이를 변형해주면

$$\frac{(W^T S_B W)}{W^T S_w W} S_w W = S_B W$$

이렇게 완성이 되는데, 이때

$$\frac{(W^T S_B W)}{W^T S_w W} = \lambda$$

로 각각의 요소들을 부분적으로 λ 라는 scalar 형태로 바꿔줄 수 있다. ($W^T S_B W$: (1xd) (dxd) (dx1)로 scalar 형태)

$$\begin{aligned} S_w W &= \lambda S_B W \\ S_B^{-1} S_w W &= \lambda W \end{aligned}$$

즉 이 문제 또한 고유값 분해는 통해 새로운 축 w 를 구할 수 있다. LDA를 통해 데이터를 변형해주고, 특정 threshold보다 높으면 A라는 범주, 낮으면 B라는 범주 라는 식의 분류가 가능해진다.

Reference

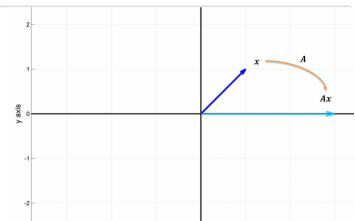
2.3.1. 고유값

linear algebra and its application 4th ed - gilbert strang : chapter 5 Eigenvalues and Eigenvectors

[Linear Algebra] Lecture 21-(1) 고유값(eigenvalues)과 고유 벡터(eigenvectors)

이번 시간에 다룰 내용은 고유값(eigenvalues)과 고유벡터(eigenvectors)이다. 고유값과 고유벡터에 대한 관련 내용도 꽤 방대하기 때문에 몇 개로 나누어서 다루도록 하겠다. 고유값과 고유벡터에 대한 중요성이나 필요성은 이 포스팅을 찾아온 분들이라면 굳이 설명하지 않아도

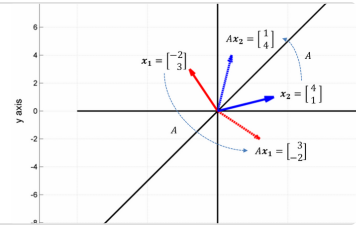
🔗 <https://twlab.tistory.com/46>



[Linear Algebra] Lecture 21-(2) 고유값(eigenvalues)과 고유 벡터(eigenvectors)

지난 강의 에 이어 고유값(eigenvalue)과 고유벡터(eigenvector)에 대한 두 번째 강의 다. 이번 포스팅에서는 고유값과 고유벡터에 대한 실제 계산법과 유용한 공식, 그리고 여러 가지 행렬들에 대한 고유값/고유벡터에 대해 알아보도록 하겠다. 지난 첫 번째 강의에서 주로 개념 위주의

📄 <https://twlab.tistory.com/47?category=668741>



[선형대수학 #3] 고유값과 고유벡터 (eigenvalue & eigenvector)

선형대수학에서 고유값(eigenvalue)과 고유벡터(eigenvector)가 중요하다고 하는데 왜 그런 것인지 개인적으로도 참 궁금합니다. 고유값, 고유벡터에 대한 수학적 정의 말고 이런 것들이 왜 나왔고 그 본질이 무엇인지에 대한 직관이 있으면 좋을텐데요..

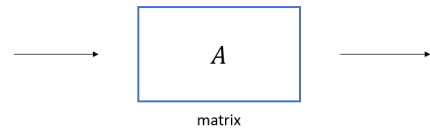
📄 <https://darkpgmr.tistory.com/105?category=460967>



고유값과 고유벡터

고유벡터와 고유값이 물어보는 것: "벡터 x 에 어떠한 선형변환 A 를 했을 때, 그 크기만 변하고 원래 벡터와 평행한 벡터 x 는 무엇인가요?" "그렇다면, 그 크기는 얼마만큼 변했나요?" 행렬은 선형 변환 연산이다. '선형'이라는 말이 어렵게 들릴 수 있으니, 일

📄 https://angeloyeo.github.io/2019/07/17/eigen_vector.html



Machine Learning & Linear Algebra-Eigenvalue and eigenvector

Eigenvalue and eigenvector are probably one of the most important concepts in linear algebra. Who can expect a simple equation like $Av = \lambda v$ is so significant? From machine learning to quantum computing, many problems

📄 <https://jonathan-hui.medium.com/machine-learning-linear-algebra-eigenvalue-and-eigenvector-f8d0493564c9>



2.3.2 고유값 분해

고유값 분해(eigen-value decomposition)

※ 시각화와 이해의 편의를 도모하기 위해 벡터와 행렬이 정의되는 체(field)는 실수(real number)로 한정함. 원래의 선형변환 A 의 EVD를 이용해 세 개의 단계로 분해한 선형변환 A 의 EVD는 기존의 선형변환을 '돌리기', '늘리기', '돌리기'의 세 과정으로 분해해...

📄 https://angeloyeo.github.io/2020/11/19/eigen_decomposition.html

2.4.1. 고유값과 고유값 분해 응용 : PCA

주성분 분석(PCA)

PCA가 말하는 것: 데이터들을 정사영 시켜 차원을 낮춘다면, 어떤 벡터에 데이터들을 정사영 시켜야 원래의 데이터 구조를 제일 잘 유지할 수 있을까? ※ 본 article에서는 열벡터(column vector) convention을 따릅니다. 100명의 학생들이 국어 시험과 영어 시험을 봤다고 생각해봅시다. 영어 시험이 조금 더 어려웠고 그 결과 중 일부는 대략적으로 다음과 같았다고 하자.

📄 <https://angeloyeo.github.io/2019/07/27/PCA.html>

CS229: Machine Learning - The Summer Edition!

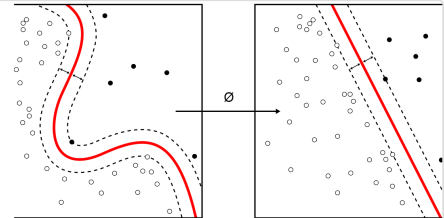
Introduction and Pre-requisites review (3 lectures) Supervised Learning (8 lectures) Lecture 4 [YouTube] 7/1 Linear Regression [Stochastic] Gradient Descent ([S]GD) Normal Equations Probabilistic Interpretation Maximum
<https://cs229.stanford.edu/syllabus-summer2020.html>



2.4.2. 고유값과 고유값 분해 응용 : ICA


독립 성분 분석 - 위키백과, 우리 모두의 백과사전

독립 성분 분석(Independent Component Analysis, ICA)은 다변량의 신호를 통계적으로 독립적인 하부 성분으로 분리하는 계산 방법이다. 각 성분은 비 가우스 성 신호로서 서로 통계적 독립을 이루는 성분으로 구성되어 있다. 독립 성분 분석은 블라 W https://ko.wikipedia.org/wiki/%EB%8F%85%EB%A6%BD_%EC%84%B1%EB%B6%84_%EB%B6%84%EC%84%9D




독립 성분 분석 (ICA)

아래의 내용에 대해 잘 알고 오는 것을 추천드립니다. 위키피디아에 따르면 독립 성분 분석(Independent Component Analysis, ICA)은 다변량의 신호를 통계적으로 독립적인 하부 성분으로 분리하는 계산 방법이라고 되어 있다. 말이 어려워 보이지만, 아래의 예시를 보면서 ICA가 할 수 있는 일이 어떤 것인지 알아보도록 하자. 그림 1. ICA가 할 수 있는 일을 block diagram으로 나타낸 것.

 <https://angeloyeo.github.io/2020/07/14/ICA.html>

2.4.3. 고유값과 고유값 분해 응용 : LDA

머신러닝 - LDA (Linear Discriminant Analysis)

선형판별분석(Linear Discriminant Analysis, LDA)는 Classification(분류모델)과 Dimensional Reduction(차원 축소)까지 동시에 사용하는 알고리즘이다. LDA는 입력 데이터 세트를 저차원 공간으로 투영(projection)해 차원을 축소하는 기법이며 지  <https://vimeo.io/@swan9405/LDA-Linear-Discriminant-Analysis>

