□ SVM

- 관측값을 두 개의 클래스로 분류하기 쉬운 방법은 그들 사이에 선형경계를 그리는 것이다.

1. SVM의 기본적인 생각.

- 경계를 각 클래스의 각 관측값에서 최대한 멀리 두는 방법
- 단순한 최적화의 모습
- 선형 경계의 개수를 생각하여 마진 최대화
- 마진이란 경계와 가장 가까운 관측값 사이의 거리.
- 모든 관측 값이 경계의 올바른 범위 내에 있어야한다는 제약이 부과됨.
- 최적의 솔루션은 경계에 가장 가까운 관측 값에 의해서만 결정됨.

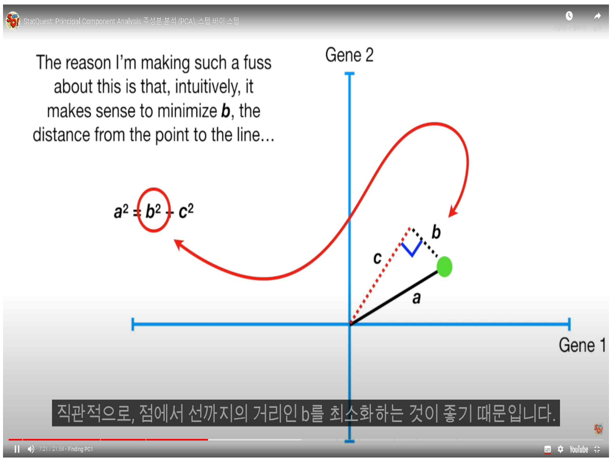
- 즉 모든 관측 값을 제대로 분류 할 수 있는 선형 경계가 존재하지 않음.
- SVM의 솔루션이 클래스 간 분리를 가능하게 하는 최적의 방법 중 하나.(마진을 최대화)
- 비선형 분류문제에서도 가능.
- 선형 분리가 가능하도록 변수를 변환하여 새로운 공간으로 이동시킴.
- SVM은 이진 클래스 분류에서만 작동한다.
- 다중 클래스의 분류 문제는 여러 이진 클래스 분류기를 결합하여 해결한다.

□ PCA(차원축소)

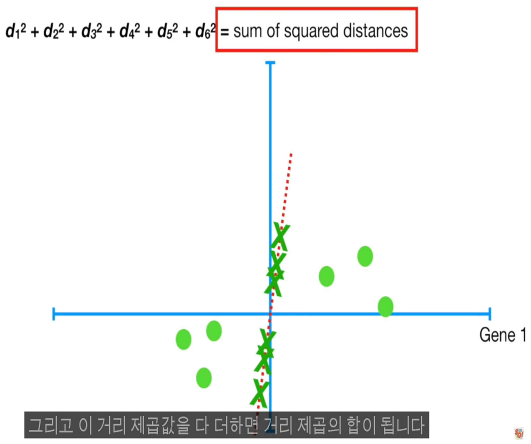
- 시각화에 많이 사용함.
- 이미지 노이즈 감소
- 적은 차원의 공간에 저장
- 주성분 분석
- Eigen Vecotr는 차원수 만큼 존재한다. ex)100 차원 100개
- 가장 넓게 퍼져있는 Eigen Vector를 선택(Eigen Value가 가장 높은 값)해서 그곳으로 데이터(점)을 모두 옮기면 PCA가 구현된다.

랜덤한 선을 그린다.(선이 데이터에 얼마나 적합한지를 정량화하기 위해 PCA는 데이터를 투영한다.)
선에서 데이터까지의 거리를 측정하고 이 거리를 최소화하는 선을 찾음.

직관적으로 점에서 선까지의 거리인 b를 최소화하는 것이 좋음.
a의 크기는 바뀌지 않음(원점에서 점까지의 거리)
b를 최소화하면 c의 길이는 최대화됨. a^2 = b^2 + c^2

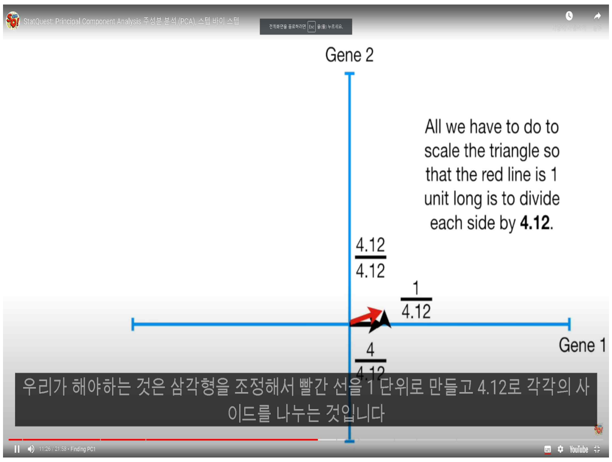


투영된 선에서부터 원점까지의 거리 제곱을 다 더한다.

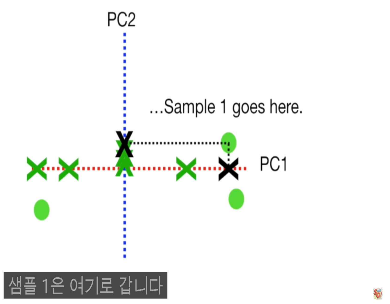


a^2 = b^2 +c^2

(a,b,c) /a => 1, b/a, c/a 로 만든다.



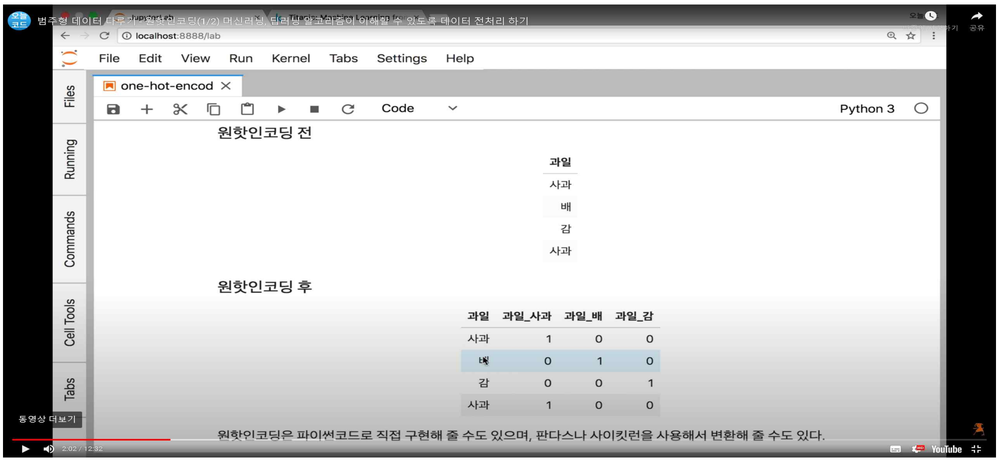
PCA가 특이값 분해를 사용하는 방식.



◘ One-Hot Encoding

- 데이터에는 수치형 데이터와 텍스트 데이터나 범주형 데이터가 있다.
- 머신러닝, 딥러닝 알고리즘은 수치 데이터만 이해가 가능함.
- 기계가 이해할 수 있는 형태로 데이터를 변환해 주어야함.
- 범주형 데이터는 원핫 인코딩 형태로 변환.

예시



과일 컬럼에 사과, 배, 감이 들어있다고 하면

각각의 과일인 사과, 배, 감으로 컴럼으로 만들어주고 해당되는 과일에만 1로 표기

나머지 과일은 0으로 표기.

◘ 코드

1. 전처리

```
import pandas as pd
import numpy as np

train = pd.read_csv('titanic/train.csv')
test = pd.read_csv('titanic/test.csv')

print(train.shape) # (891, 12)
print(test.shape) # (418, 11)

# 데이터 타입 확인
train.dtypes

test.dtypes

train.columns # 컬럼 정보 확인

# 컬럼 정보 :
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')

train.describe() # 컬럼 상태 확인

train # 데이터 프레임 정보 확인
```

```
# 오브젝트 타입의 데이터만 따로 추출해 본다.
# 이 데이터 중에 카테고리 형태의 데이터가 무엇인지 보고 인코딩 해준다.
# 원핫인코딩 뿐만 아니라 자연어처리(NLP)에서 배웠던 TF, TF-IDF의 인코딩도 해줄 수 있음.
obj_df = train.select_dtypes(include=['object']).copy()
obj_df.head()

# 처리전과 비교해 보기위해 데이터를 복사
train_c_df = train.copy()
test_c_df = train.copy()
```

2. 성별

```
train['Sex'].value_counts() # 성별 확인 (남자 몇 명, 여자 몇 명인지)


# 남자는 0 여자는 1로 변환. (원 핫 인코딩)
train.loc[train["Sex"] == "male", "Sex"] = 0
train.loc[train["Sex"] == "female", "Sex"] = 1


test.loc[test["Sex"] == "male", "Sex"] =0
test.loc[test["Sex"] == "female", "Sex"] =1
```

3. 사이킷런의 LabelEncoder로 원핫인코딩해준다

```
# 카테고리 데이터를 인코딩 해준다.
from sklearn.preprocessing import LabelEncoder


# 성별을 0과 1로 인코딩
def gender_to_int(data):
    le = LabelEncoder()
    le.fit(["male", "female"]) # male 1, female 0
    data["Sex"] = le.transform(data["Sex"])
    return data


train_c_df = gender_to_int(train_c_df)
test_c_df = gender_to_int(test_c_df)
train_c_df.head()
```

4. 승선위치

```
train['Embarked'].value_counts()
train_c_df["Embarked_C"] = train_c_df["Embarked"] == "C"
train_c_df["Embarked_S"] = train_c_df["Embarked"] == "S"
train_c_df["Embarked_Q"] = train_c_df["Embarked"] == "Q"

print(train.shape)
print(train_c_df.shape)
train_c_df[["Embarked","Embarked_C","Embarked_S","Embarked_Q"]].head(10)
```

5. 판다스의 get_dummies로 원핫 인코딩

```
def dummy_data(data, columns):
    for column in columns:
        data = pd.concat([data, pd.get_dummies(data[column], prefix = column)], axis = 1)
        data = data.drop(column, axis=1)
    return data

dummy_columns = ["Sex", "Pclass", "Embarked"]
train_dummy = dummy_data(train, dummy_columns)
test_dummy = dummy_data(test, dummy_columns)

print('원핫인코딩 전 shape')
print(train.shape)
print(test.shape)

print('get_dummies로 원핫인코딩 후 shape')
print(train_dummy.shape)
print(test_dummy.shape)

train_dummy.head()
```

○ 인코딩된 데이터를 그대로 사용하게 된다면 사용하지 않는 컬럼을 drop 해주는 방법으로 피처를 생성

```
def drop_not_concerned(data, columns):
    return data.drop(columns, axis=1)
not_concerned_columns = ["PassengerId", "Name", "Ticket","Cabin"]
X_train = drop_not_concerned(train_dummy, not_concerned_columns)
X_train = X_train.drop('Survived', axis=1)
X_test = drop_not_concerned(test_dummy, not_concerned_columns)
```

ㅁ KNN

1. 기본적인 생각

ㅇ 샘플을 분류하는 가장 쉽고 효과적인 방법이 무엇일까?

- 가장 가까운 샘플을 찾고, 그 샘플과 같은 클러스터로 분류
- 인접한 데이타에 대해 매우 민감하고, 또 실제 카테고리는 이렇게 깨끗하게 분리되지 않음.
- 만약 가장 인접한 샘플이 이상값이라면, 새로운 샘플은 잘못 분류 될 가능성이 높아짐.
- 이런 문제를 해결하는 가장 쉬운 방법은 더 많은 샘플을 관찰하여 다수를 이루는 클러스터로 분류

ㅁ K-Means-Clustering

- 관측 값을 지정된 몇개의 그룹으로 나누는 방법.
- K개 그룹의 중심 위치를 결정.
- 각 관측 값에 대해 어떤 중심에 가장 가까운지 여부에 따라 그룹을 할당함.
- 처음에는 중심 위치를 무작위로 설정, 관측 값들은 가장 가까운 중심의 그룹에 할당.
- 각 그룹의 중심 위치는 그룹에 속한 관측 값들을 평균하여 업데이트됨.
- 그룹을 결정하는데 사용되는 것은 중심위치입니다.
- K-Means는 임의의 중심위치를 사용
- 최적의 클러스터링은 그룹의 중심을 기준으로 조밀한 결과를 나타낼 때.
- 각 관측에서 해당 그룹의 중심까지의 총 거리가 클러스터링 품질의 척도입니다.
- 여러 시작 위치를 시도할 수 있는 옵션을 실행하여 총 거리가 가장 짧은 결과가 채택됩니다.

ㅁ ICA(독립성분분석)

1. 필요 개념

- 중심극한정리

독립 확률 변수 n개의 평균의 분포는 n이 적당히 크다면 정규분포에 가까워진다는 정리이다.
충분히 많은 수의 iid랜덤변수의 표본평균이 정규 분포를 따른다.

- 최대 우도법(MLE)

모수적인 데이터 밀도 추정방법
파라미터으로 구성된 어떤 확률밀도함수 P에서 관측된 표본 데이터 집합을
x=(x1, x2, ... ,x(n))이라 할 때, 이 표본들에서 파라미터를 추정하는 방법.

- 경사 하강법

함수의 기울기(경사)를 구하고 경사의 절댓값이 낮은쪽으로 계속 이동시켜 극값에 이를 때까지 반복시키는 것이다.
모든 차원과 모든 공간에서 적용 가능.

ICA 모델과 목적
x1(t), x2(t)라고 하고, 두 사람의 음성 신호를 s1(t),s2(t)라고 하면 다음과 같은 관계로 모델링 할 수 있다.
```
x1(t) = a11*s1(t) +a12*s2(t)
x2(t) = a21*s1(t) +s22*s2(t)
x = As
```

- 이 때 우리가 원하는 것은 녹음 음원으로부터 원래 source의 음원을 분리해내는 일.
s = A^(-1)*x = W*x # A^(-1)을 찾는 행위 => ICA

ㅁ 중심극한정리와 독립성분분석
1. 독립성분분석은 중심극한정리를 거꾸로 생각하는 것.
2. 중심극한정리
- 서로 독립적인 랜덤 변수들의 선형조합 => 가우스 분포를 따른다.
- 선형 조합에 들어가는 독립 변수들이 많을 수록 더 가우스 분포에 가까워진다.

3. 독립성분분석
- 선형조합을 통해 더 가우스 분포에 가까운 분포를 따르는 결과물들을 어떻게 조합하면 원래의 독립적인 source를 얻을 수 있을까?

=> ICA에서는 source들이 서로 독립적이라는 가정을 최대한 만족할 수 있도록 하는 W = A^(-1)을 찾는 것이 목적.

랜덤변수에 선형변환을 적용했을 때의 density function의 변화
- 랜덤 변수에 선형 변환을 적용 했을 때 변환 적용 전, 후 변수의 확률밀도간의 관계를 생각해보자.

- 확률밀도 함수의 전체 면적은 1이 되어야 하므로, 어떤 행렬로 랜덤 변수를 선형변환 해주면 그 확률밀도 함수는 행렬식을 이용해 보정 해야 함.

``` 수식
s:[0,1]
ps(s) = 1{0 <= s <=1}

x:[0,2]
px(x) =(0.5)1{0<=x<=2} |W| = |A^(-1)|

x의 확률밀도 함수:px(x)= ps(Wx)*|W|
```

ㅁ 신호처리 (FFT and STFT)

Signal Representation by Harmonic Sinusoids

ㅁ FFT(Fast Fourier Transform)

FFT with Smapling Frequency
적은 계산량으로 이산 푸리에 변환값을 계산하는 알고리즘.

x[n] = 2cos(2*pi*60t) => f = 60   60t => frequency(주기)

```python
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

from scipy.fftpack import fft, fftshift
from scipy.signal import spectrogram

Fs = 2**10 # Sampling frequency
T = 1/Fs # Sampling period (or sampling interval)

N = 5000 # Total data points (signal length)

t = np.arange(0, N) * T # Time vector (time range)

k = np.arange(0, N) # vector from 0 to N-1
f = (Fs/N) *k # frequency range

x = 2*np.cos(2*np.pi*60*t) + np.random.randn(N) # 신호 + 랜덤 노이즈
```

```python

# original fft
xt = fft(x)/N
xtshift = ffshift(xt)

kr = np.hstack([np.arange(0, N/2), np.arange(-N/2, 0)])
fr = (Fs/N)*kr
fs = fftshift(fr)

# single-sides fft

xt = fft(x)/N
xtss = xt[0:int(N/2)+1]
xtss[1:-1] = 2*xtss[1:-1]

fss = f[0:int(N/2)+1]
```

```python
x1 = np.cos(2*np.pi*50*t)
x2 = np.cos(2*np.pi*100*t)

x = np.zeros(t.shape)
x[0:int(N/2)] = x1[0:int(N/2)]
x[int(N/2):-1] = x2[int(N/2):-1]

z = 1/2*(x1 + x2)
```

ㅁ STFT (Short-Time Fourier Transformation)
FFT는 시간의 흐름에 따라 신호의 주파수가 변했을 때, 어느 시간대에 주파수가 변하는지 모르게 됨. 이러한 한계를 극복하기 위해서, STFT는 시간의 길이를 나눠서 푸리에 변환을 하게됨.

ㅁ 웨이블릿
급격한 변화가 있는 신호 및 이미지를 정확하게 분석하려면 시간과 주파수에
대해 국한되어 있는 새로운 함수클래스를 사용해야함.
웨이블릿은 유한기간 동안 존재.
웨이블릿은 모양과 크기가 다름.
- 웨이블릿의 종류
Morlet, Daubechies, Coiflets, Biorthogonal, Mexican Hat, Symiets

1. 웨이블릿 변환 개념에서 중요한 것
웨이블릿은 비례 비율이 일정한 스케일과 주파수 사이에 상호 관계가 있음.
이 비례 상수를 웨이블릿의 중심 주파수(Center Frequency)라고 함.

- 스케일링(Scaling)
방정식을 사용하여 표현할 수 있는 시간에 따라 신호를 늘리거나 줄이는 과정.
예시로 사인파의 10Hz를 2만큼 스케일링하면 원래 주파수가 절반 또는 한 옥타브 감소함. 10Hz => 5Hz

- 이동(Shifting)
신호 길이에 따라 웨이블릿의 시작을 지연시키거나 진행하는것.

2. 웨이블릿 분석의 두 가지 주요 변환
- 연속 웨이블릿 변환
- 이산 웨이블릿 변환

3. 이산 웨이블릿 변환(Discrete wavelet transform)
- 주요 응용 분야
신호 및 이미지의 노이즈 제거 및 압축

- 적은 계수로 자연 발생 신호와 이미지를 표현하는데 적합

- 서로 다른 해상도에서 점차 좁은 서브밴드에서 신호를 분석하는데 도움을 줌.

4. 순서
- 근사와 세부 계수를 얻는다.(다단계 웨이블릿 분해를 수행)
- 세부 계수를 분석하고 적절한 임계 값을 선별.
- 세부 계수를 임계값으로 구분하고 신호를 재구성.

ㅁ 임계값 방식
- 범용 임계값 계산 공식(The universal threshold)
sqrt(2*log(length(X))) * median(abs(D))/ 0.6745

1. Soft Thersholding
- 임계 값보다 작은 크기의 계수는 0으로 설정
- 임계 값보다 큰 계숭에서 임계 값을 빼서 계수의 크기가 줄어듦.

2. Hard Thersholding
- 임계 값보다 작은 크기의 계수는 0으로 설정
- 임계 값보다 큰 계수는 변경되지 않음.

ㅁ 연속 웨이블릿 변환(Continuous wavelet transform)
- 주요 웨이블릿 분석
1. Morse Wavelets
2. Analytical Morlet
3. Bump Wavelet

- 주요 응용 분야
시간 주파수 분석 및 시간적으로 국부화된 주파수 성분의 필터링

- 웨이블릿의 스케일과 등가 주파수는 반비례함.

- 신호의 진동 동작을

- spectrogram() : 신호와 샘플링 주파수를 전달.
사용 예시 : spectrogram(kobe,128, [], [], Fs, 'yaxis')

- cwt() : 웨이블릿 계수와 등가 주파수를 출력으로 반환
사용 예시 : cwt(kobe, Fs, 'NumOctaves', No, 'VoicesPerOctave', Nv);

ㅁ AI 신호처리

ㅁ 인공신경망의 구조
1. Input Layer
- 데이터를 넣어주는 과정

2. Hidden Layer
- 데이터의 특성을 학습하는 과정.

3. Output Layer
- 분류나 회귀문제의 정답을 알려줌.

ㅁ 대표적인 인공 신경망

1. CNN
- Convolution 연산을 통해 결과 도출.
- 현재출력이 현재 입력만 영향.
- 일차원의 신호를 이차원으로 변환해서 데이터를 인풋값으로 넣음.

2. LSTM
- RNN의 일종
- 현재출력이 이전의 입력까지 고려함.
- 가지고 있는 데이터의 시간에 따라 변화하는 특성들을 인풋값으로 넣음

ㅁ Deep Learning Workflow

1. Create And Access Datasets
- 좋은데이터가 좋은 결과를 만든다.
- 연구목적

모델 개발에 압도적으로 많은 시간 소요가들어감.

- 실제 산업

데이터개발에 많은 시간 소요가 들어감.

```MatLab
Datastore() : 대용량의 데이터셋을 가져 와서 처리를 쉽게 함.
사용 예시
a = audioDatastore(pwd, 'IncludeSubfolders', true...
, "LabelSource", "foldernames")
```

- 데이터를 강화함으로써 대용량의 데이터를 만듬으로써 더 견고한 모델 생성 가능.

2. PREPROCESS AND TRANSFORM DATA

3. DEVELOP PREDICTIVE MODELS
- Design
- Train
- Optimize

4. ACCELRATE AND DEPLOY

ㅁ 신호처리 - 푸리에 트랜스폼.

- sparse domain => frequency domain

- Discrete fourier transform

- 사용처

각각의 통신사들(SK, KT, LG) 자신들이 활용하는 대역폭으로 전달받은 영상 신호를 전송시킨후에 inverse system을 이용해원본으로 복원한다.

원본신호 => SK, KT, LG 대역폭으로 변조 신호 전달.(Forward transform 원본신호 => 변조 신호)
 => 각 통신사마다 다른 대역폭으로 신호 변조
 => 받은 신호를 사용자들의 단말기로 전달
 => 신호를 받은 단말기는 다시 신호를 변조(inverse transform 변조신호 => 복조신호)

1. 1D Fourier transform

```MatLab
N= 100;

xn = zeros(N,1);
xn(1) = 1;
xk = fft(xn);

figure(1)
subplot(131): stem(xn):title('spatial domain')
subplot(132): stem(real(xk):title('fourier domain: real')
subplot(133): stem(image(xk):title('fourier dom ain: imaginary')
```

ㅁ 컨볼루션 연산과 푸리에 트랜스폼의 관계

```MatLab
## 1D Convolution
Ny = 64;
Nx = 1;

My = 64;
Mx = 1;

f = zeros(Ny, Nx);
g = zeros(My, Mx);

f((1:Ny/4) + Ny*3/8, :) = 1;
g((1:My/4) + My*3/8, :) = 1;

pad_pre = [floor((My-1)/2), floor((Mx -1)/2)];
pad_post = [floor(My/2), floor(Mx/2)];

f_pad = padarray(f, pad_pre, 'pre');
f_pad = padarray(f_pad, pad_post, 'post');
```

□ Separability와 Dimension Embedding, MobileNet에 어떻게 적용되는가?

1. Separability : the Concept of MobilenetV1

```MatLab
## Full-rank matrix
N = 4; # matrix 개수
mat = zeros(N,N);

for i =1;N
    mat(i, i) = N - (i - 1);
end

figure(1);
imagesc(mat); axis image off; title('Ground Truth');

## SVD
[U, S, V] = svd(mat);

figure(2);
subplot(141); imagesc(mat); axis image off; title('Ground Truth');
subplot(142); stem(diag(S)); title('Singular value');
subplot(143); imagesc(mat); axis image off; title('U matrix of SVD');
subplot(144); imagesc(mat); axis image off; title('V matrix of SVD');

## Rank1 matrix
BSS1 = U(:, 1) * V(:, 1)';
BSS2 = U(:, 2) * V(:, 2)';
BSS1 = U(:, 3) * V(:, 3)';
BSS2 = U(:, 4) * V(:, 4)';

COEF1 = S(1,1);
COEF2 = S(2,2);
COEF3 = S(3,3);
COEF4 = S(4,4);

## Convolution operator
## Gaussian kernel

# Input image matrix size
Ny = 255;
Nx = 301;

# gausian kernel size
My = 10;
Mx = 13;

# 2D gausian scaling factor
a = 1;

# gausian kernel sigma parameter
sgmy = 3;
sgmx = 3;
```

```
# gausian kenel center position
y0 = 0;
x0 = 0;

## 2D Gaussian kernel
ly = linspace(-(My-1)/2, (My - 1)/2, My);
lx = linspace(-(Mx-1)/2, (Mx - 1)/2, Mx);

[mx, my] = meshgrid(lx, ly);

W = a * exp(-((mx - x0).^2/(2*sgmx^2) + (my-y0).^2/(2*sgmy^2)) )
W = W / norm(W);

## 1D Gaussian kernel

Wy = a * exp(-((ly - y0).^2 / (2*sgmy^2)));
Wx = a * exp(-((lx - x0).^2 / (x*sgmx^2)));

Wy = Wy/norm(Wy);
Wx = Wx/norm(Wx);

Wxy = Wy * Wx;

## 2D Convolution vs. separable 1D convolution

x = imresize(phantom(max(Ny, Nx)), [Ny, Nx]);

x_conv2d = conv2(x, W, 'same');

x_conv1dy = conv2(x, Wy, 'same');
x_conv1dxy = conv2(x_conv1dy, Wx, 'same');


## 2D Fourier transform vs. 1D Separable Fourier transform
x_ft2d = fftshift(fft2(ifftshift(x)));
x_ift2d = real(fftshift(ifft2(ifftshift(x_ft2d))));

x_ft1dy = fft(ifftshift(x), [], 1);
x_ft1dxy = fftshift(fft(x,ft1dy, [], 2));

x_ift1dy = ifft(ifftshift(x_ft1dxy), [], 1);
x_ift1dxy = real(fftshift(ifft(x_ift1dy, [], 2)));
```

ㅁ 콘볼루션과 푸리에 트랜스폼, 매트릭스 곱으로 표현

```MatLab
%% 1D Fourier transform
Ny = 128;
Nx = 1;

x = rand(Ny, Nx);

%% 1D operator
x_ft1d = fftshift(fft(ifftshift(x)));
x_ift1d = real(fftshift(ifft(ifftshift(x_ft1d))));

%% 1D matrix multiplication
ny = linspace(0, Ny - 1, Ny);
ky = linspace(0, Ny - 1, Ny);

kyny = ky(:) * ny(:)';

ft1d_mtx = exp(-1j * 2 * pi * kyny/Ny);
## ift1d_mtx = 1/N * exp(1j * 2 * pi * kyny/Ny);
ifft1d_mtx = 1/N * ft1d_mtx;

x_ft1d_mtx = fftshift(ft1d_mtx * ifftshift(x));
x_ift1d_mtx = fftshift(ift1d_mtx * ifftshift(x_ft1d_mtx));

%% 2D Fourier transform
Ny = 128;
Nx = 108;

x = imresize(phantom(max(Ny, Nx)), [Ny, Nx], 'nearest');
```

□ 함수의 tranpose

```MatLab

% matrix version
% <A * x, y> = <x, A^T * y>

% operator & function version
% <A(x), y> = <x, A^T(y)>

%% Transpose of matrix
% A in N * M
% x in M * K
% y in N * K

N = 100;
M = 80;
K = 30;

A = randn(N, M);
x = randn(M, K);
y = randn(N, K);

AT = A';

% lhs = <A*x, y>
Ax = A * x;
lhs = Ax(:)' * y(:);

% rhs = <x, A^T*y>
ATy = AT * y
rhs = x(:)' * ATy(:);


## Transpose of Fourier transform function
Ny = 100;
Nx = 100;

A = @(x) fft2(x);
x = randn(Ny, Nx);
y = randn(Ny, Nx);

AT = @(y) Ny * Nx * ifft2(y);

% lhs = <A(x), y>
Ax = A(x);
lhs = Ax(:)' * y(:);

% rhs = <x, A^T(y)>
ATy = AT(y);
rhs = x(:)' * ATy(:);
```