

데이터 전처리

데이터 전처리 과정

■ 데이터 평가 Access Data

- Dirty Data 판별 -> Data Cleaning
- Data Mining 성능 -> Data Preprocessing

■ 데이터 정제 Cleaning Data

- 누락 데이터 처리
- 이상치 처리

■ 데이터 전처리 Data Preprocessing

- Feature Scaling
- Dimensionality Reduction

데이터 전처리 분류

■ 데이터 전처리 Data Preprocessing

- Feature Scaling
: 정규화와 표준화
- Dimensionality Reduction
 - Feature Selection
 - Feature Extraction
- 변환
 - 단위환산, 자료형 변환
- 범주형 데이터처리
 - 구간분할, 더미변수

기본 용어 정리

■ Observation

- 개별 관측대상에 대한 다양한 속성 데이터들의 모음인 레코드
- Observation은 보통 데이터프레임의 행이 된다

■ Feature

- 공통의 속성을 갖는 일련의 데이터로 관측대상에 대한 정보
- Feature는 보통 데이터프레임의 열이 된다.
- 데이터 마이닝 ^{Data Mining} 분야에서는 feature라는 용어를 사용하지만, 통계 등의 분야에서 variable과 같은 의미

설명예제

■ 용어 설명예제

- Observation은 각각의 행에 있는 학생으로 관측대상이 된다
- 각각의 열에 있는 학교나 나이가 feature가 된다 즉, 관측대상인 학생의 정보들이 feature가 된다

	나이	성별	학교	수학	과학
준서	15	남	덕영중	80	90
예은	17	여	수리중	80	80
길동	16	여	연수중	80	70
희수	15	남	반포중	100	95

Feature Scaling 배경

■ Scaling 용어

- Scale : 규모(스케일이 크다), 등급-음계 , 비율, 저울
- Scaling 크기 조정

■ 배경

- 데이터 처리를 하다 보면 데이터들의 스케일링 이슈가 중요한 요소로 작용한다는 걸 쉽게 알 수 있음
- 여러 가지 데이터를 하나로 묶어서 처리하거나, 머신러닝 처럼 학습을 시키는 경우에는 보통은 유클리디안 거리(Eucledian distance)를 사용하기 때문에 스케일링을 하지 않으면 문제가 발생

유클리디안 거리

■ 유클리디안 거리 Euclidean distance

- 유클리디안 거리는 공간에서의 두 점 사이의 거리를 의미하며 수학적으로는 두 점 사이의 차이를 제곱하고 루트를 씌운 수식
cf. 피타고라스의 정리 : 2차원 vs. 유클리디안 거리 : 다차원 (일반식)

두 점 P 와 Q 가 각각 $P = (p_1, p_2, p_3, \dots, p_n)$ 와 $Q = (q_1, q_2, q_3, \dots, q_n)$ 의 좌표를 갖을 때 두 점 사이의 거리를 계산하는 유클리디안 거리 (Euclidean distance) 공식은 다음과 같습니다.

$$\sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \dots + (p_n - q_n)^2} = \sqrt{\sum_{i=1}^n (p_i - q_i)^2}$$

Feature Scaling 필요성

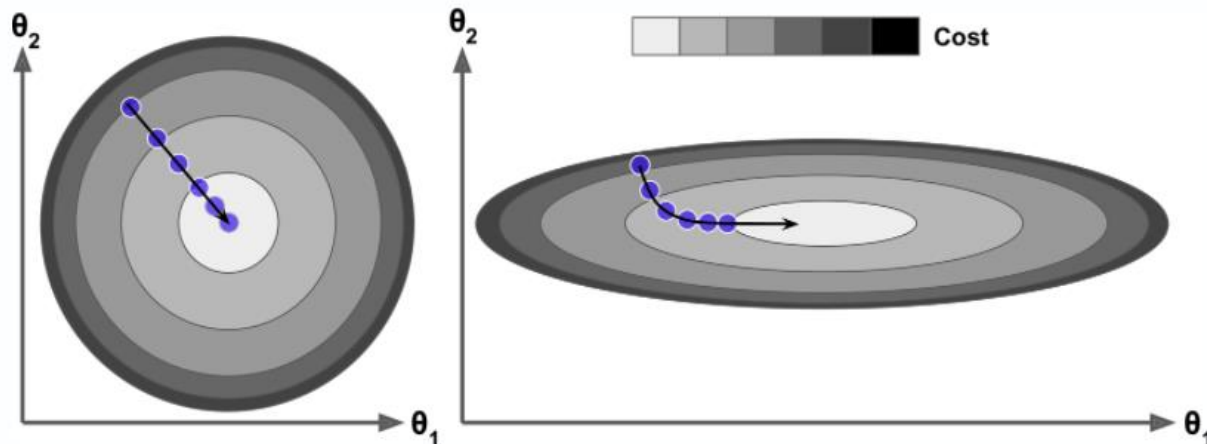
■ 필요성

- 물리적인 값이나 일상 생활의 여러 수치들에는 고유의 단위가 있거나 처리가 되는 값의 크기가 다양하다. 1의 자리 수만 존재하는 특성들과 10000의 자리에서 변화하는 특성들은 학습할 때 유클리디안 거리가 크게 다르기 때문에 원치 않은 결과를 초래할 수 있음
- 만약, 서로 다른 척도로 측정된 변수들을 같은 선상에 놓고 비교하는 것은 일종의 편의(bias)를 낳게 됨
- 예를 들면, 0 부터 1,000까지의 범위를 가지는 변수는 0부터 1 사이의 범위를 가지는 변수보다 더 큰 영향력을 가지는 것처럼 보일 수 있습니다. 스케일링을 하지 않고 이 변수들을 그대로 사용하게 되면 그 변수는 분석 상으로는 1,000이라는 더 큰 범위의 가중치를 가지게 됨

Feature Scaling 적용예

■ 경사하강법 (Gradient Descent)

- 머신러닝에서 최적해를 찾기 위해 손실함수(loss function)의 최소값을 찾는데 경사하강법이 일반적으로 사용됨.
- 이때 feature scaling을 사용한 경우에는 아래 좌측 그림과 같이 빠른 시간에 최적해를 구할 수 있어 feature scaling 기법은 선택이 아니라 꼭 구현해야 할 전처리 기법임.



Feature Scaling 정의

■ Feature Scaling

- 서로 다른 변수 `variable [feature]`의 값 범위를 일정한 수준으로 맞추는 방법
- 비교해야 할 데이터의 기준이 서로 다른 경우에 같은 기준으로 만들어서 비교
- 머신 러닝 등의 분석 성능 향상을 위해

■ 정규화^{normalization} (넓은 의미로)

- 표준화^{standardization} : 평균 0, 표준편차 1이 되도록 변환
- 정규화 : (좁은 의미로)
 - MinMax Normalization 0~1사이로 변환
 - Mean Normalization -1~1사이로 변환 (평균 0)

표준화

■ 표준화 standardization

- 표준화는 평균을 0, 표준편차를 1이 되도록 값을 스케일링하는 것
- z-score : (관측값 - 평균) / 표준편차

$$z = \frac{x - \mu}{\sigma}$$

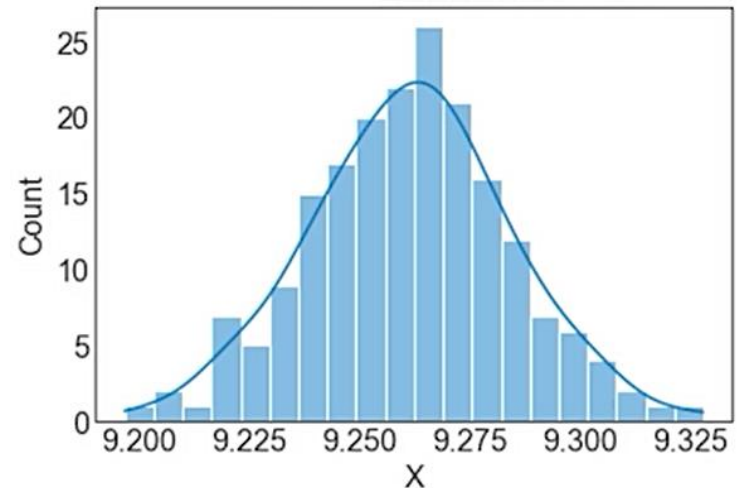
■ 표준화 는 언제 적용?

- 데이터가 정규분포 (가우시안 분포 Gaussian Distribution 라고도 함)를 따르고 있다고 가정할 때 유용 (p12~14)
- 예를 들어, 선형 회귀분석(linear regression), 로지스틱 회귀분석(logistic regression) (p15~18)

정규분포의 모양

■ 정규분포 Normal Distribution 의 모양

- 가우시안 분포라고도 함
- 종모양
- 중앙에 빈도가 몰려 있음
- 좌우 대칭
- 양쪽 꼬리 부분이 빨리 사라짐



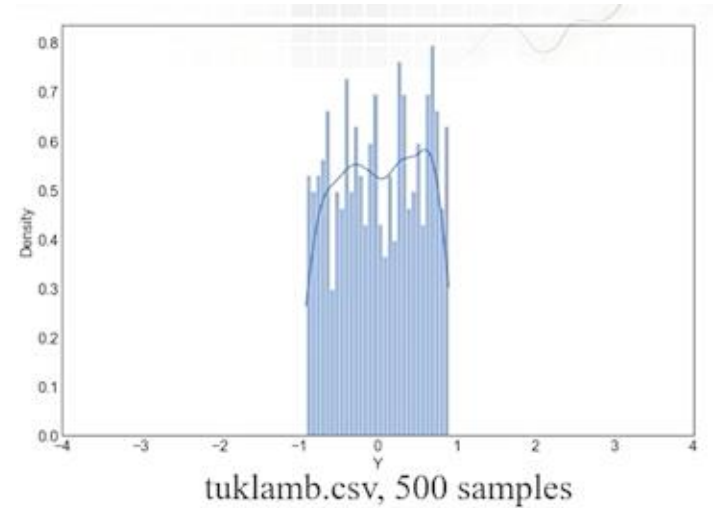
■ 다른 분포와 모양 비교

- 균일분포 : 꼬리가 거의 없거나 양쪽 꼬리가 마치 잘린 것 같음
- 코시분포 : 양쪽 꼬리가 길고 두꺼움

다른 분포의 모양

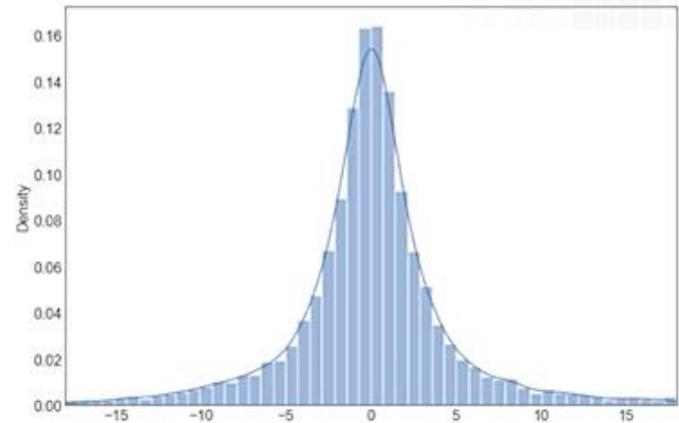
■ 균일분포 Uniform Distribution

- 정규형이 아님
- 대칭형
- 꼬리가 거의 없거나 양쪽 꼬리가 마치 잘린 것 같음



■ 코시분포 Cauchy Distribution

- 정규형이 아님
- 대칭형
- 양쪽 꼬리가 길고 두꺼움



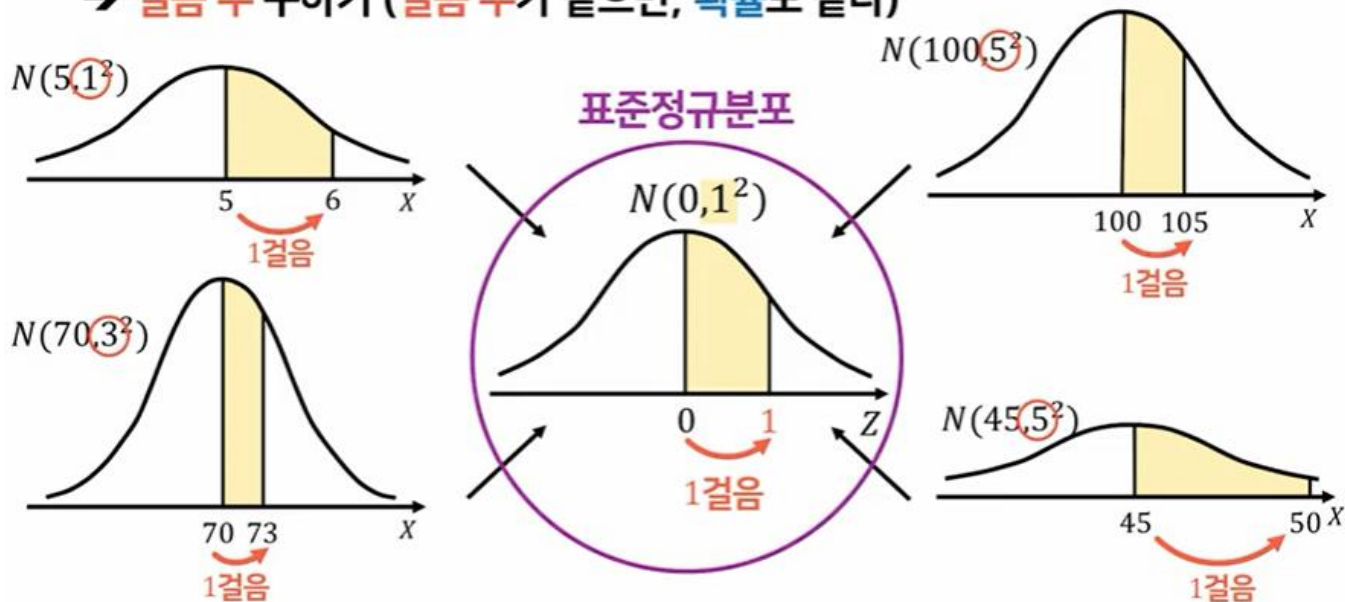
표준정규분포

■ 표준정규분포

- 정규분포의 표준화

* **표준화** ($X \rightarrow Z$)

→ **걸음 수** 구하기 (걸음 수가 같으면, 확률도 같다)



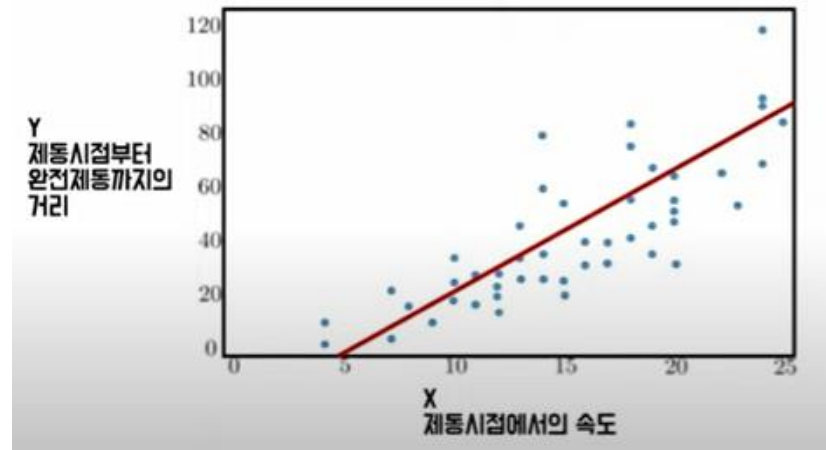
회귀분석

■ 회귀분석 Regression

- 인과관계를 수학적으로 분석하는 것
- 즉, 원인 (독립변수)과 결과 (종속변수)의 관계를 분석

■ 선형회귀분석 Linear Regression

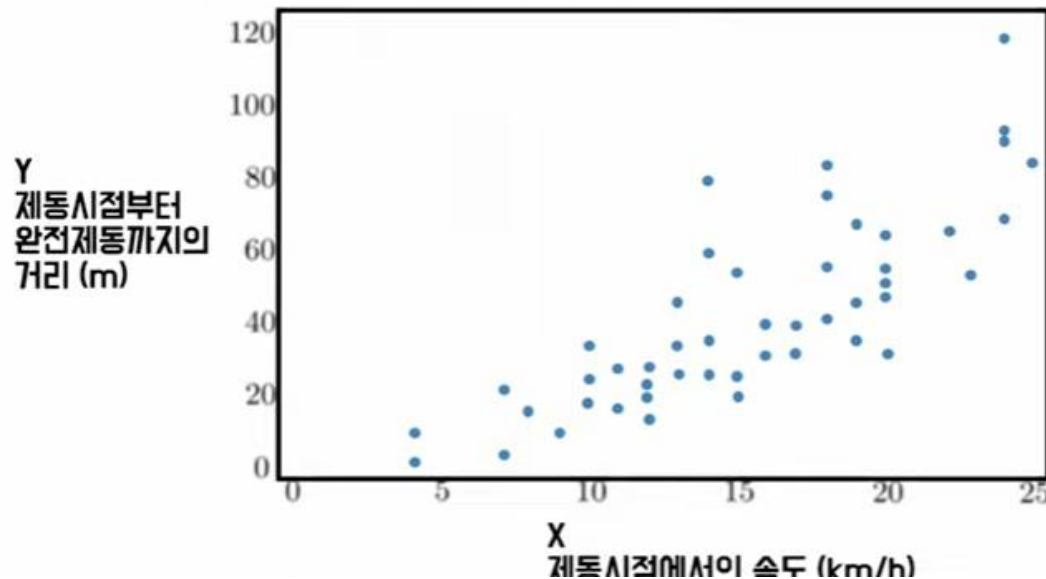
- 두 변수의 관계를 가장 잘 설명할 수 있는 직선을 찾는 것
- 예를 들어, 제동시점의 속도와 완전제동까지 거리의 관계를 가장 잘 설명할 직선



선형회귀분석 사례

■ 자동차 제동 시험 사례

- 자동차의 제동시점의 속도와 제동시점부터 완전제동까지의 거리의 인과관계를 가장 잘 설명할 수 있는 직선을 찾자
- 50번의 실험을 통해 X, Y를 표시한 그래프

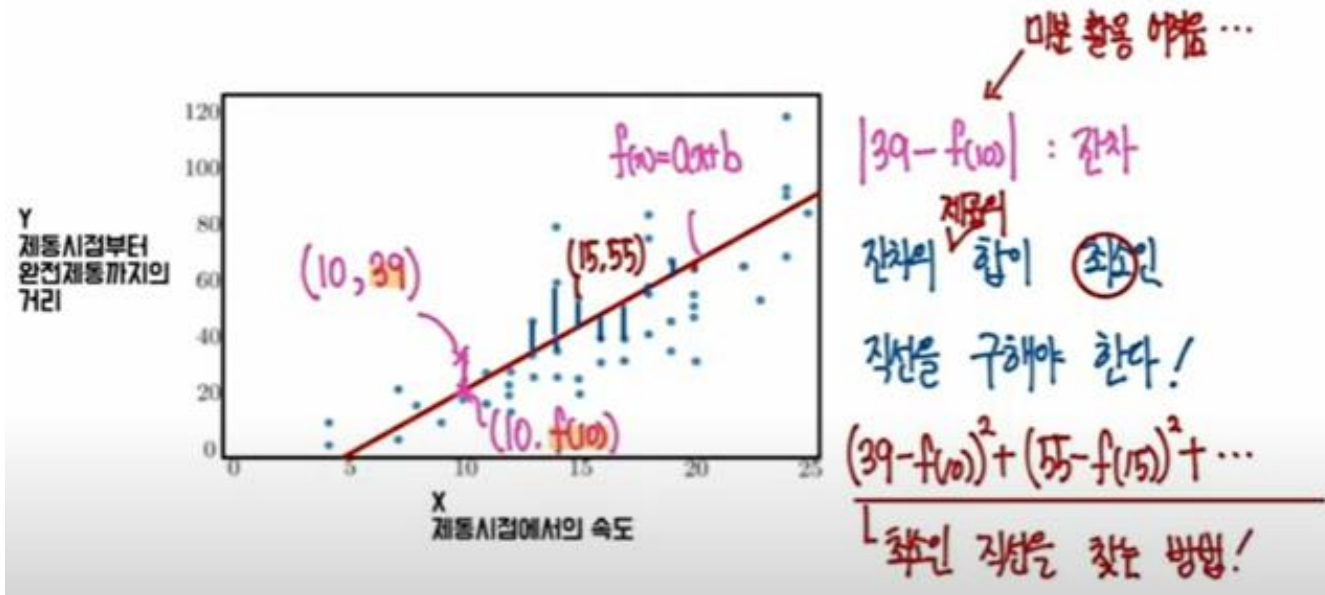


거리 예측?

30km/h

선형회귀분석 접근법

- 잔차의 제곱의 합이 최소가 되는 직선 $f()$ 을 구하자!
- 예를 들어, 속도 10km/h에 대해 실험한 거리 값은 39m이고, 예측한 결과는 $f(10)$ 이므로 잔차는 $39 - f(10)$



오차분포 vs. 정규분포

■ 오차분포

- 정규분포를 도출하게 된 근원 분포

오차분포



<https://www.youtube.com/watch?v=ZzGtRmGvLWQ>



이상치 문제

■ 이상치 문제

- outlier에 대해 좋은 결과를 내는 모델은 상황에 따라, 평균 근처의 값들 (대부분의 값들)에 대한 좋지 못한 결과를 낼 수도 있음
- 대표적으로 MinMax를 사용한 정규화의 경우, 이상치에 약함. 즉, 1000개의 데이터 중에서 999개가 0~50사이 값이고 최대값이 100으로 한 개 있을 때, 정규화에 의해 999개의 값은 0.5이하가 됨

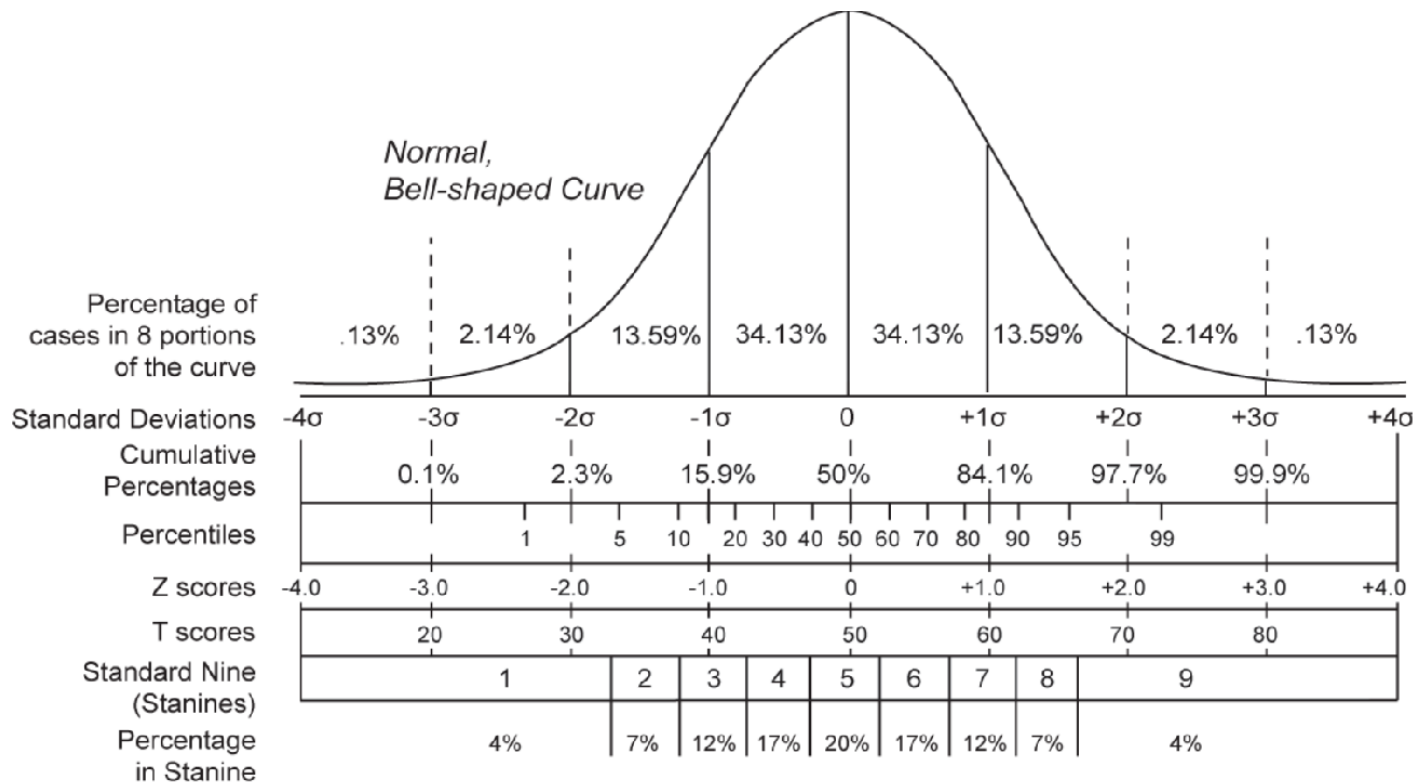
■ 표준화로 이상치 제거

- z-score (Standardization수식) 의 절대값이 2 또는 3을 넘는 값들을 제거함 (즉, 평균에서 크게 벗어난 outlier를 제거함)
- 이로 인해 대부분의 경우 모델이 좋은 성능을 갖게 함

표준정규분포

■ 표준정규분포 곡선

- z-score가 -2~2 사이에 값이 95%, -3~3 사이에 99% 존재



실습1.1 표준화

■ 평균, 표준편차 구하기

- housingsample.csv데이터
: 집의 가격,크기, 연식
- method : mean(), std()

```
import pandas as pd
```

```
df = pd.read_csv('housingsample.csv')  
df
```

	Price	Sq Feet	Age
0	90300	1700	14
1	150500	1518	27
2	269500	2708	22
3	98000	830	15
4	244650	2550	28

```
price_mean = df.Price.mean()  
price_std = df.Price.std()  
print('mean=',price_mean,', std=',price_std)
```

```
mean= 170590.0 , std= 82745.80351897007
```

실습1.2 표준화

■ z-score

- z-score를 직접 연산으로
- `df.Price - price_mean`
: Price열 모두에 대한 연산
- `df.shape`
: (5,3)
- `df.iloc[:, i]`
: i번째 열을 의미

```
(df.Price-price_mean)/price_std
```

```
0    -0.970321  
1    -0.242792  
2     1.195348  
3    -0.877265  
4     0.895030  
Name: Price, dtype: float64
```

```
for i in range(df.shape[1]):  
    col = df.iloc[:, i]  
    col = (col-col.mean())/col.std()  
    df.iloc[:, i]=col  
    #print(col)  
df
```

	Price	Sq Feet	Age
0	-0.970321	-0.208166	-1.101840
1	-0.242792	-0.443193	0.887593
2	1.195348	1.093519	0.122427
3	-0.877265	-1.331645	-0.948807
4	0.895030	0.889485	1.040627

scikit-learn library

■ 파이썬으로 머신러닝을

- scikit-learn 핵심개발자

Andreas Mueller 자료

<https://scikit-learn.org/stable/index.html#>



Machine Learning with Scikit-Learn

Andreas Mueller (NYU Center for Data Science, scikit-learn)

■ scikit-learn 설치방법

- <https://scikit-learn.org/stable/install.html>

실습2.1 Library 준비

■ scikit-learn 라이브러리 import하기

```
from sklearn import preprocessing
```

■ scikit-learn 라이브러리 설치 안 된 경우 (아래 오류 발생)

```
from sklearn import preprocessing
```

```
ModuleNotFoundError                                Traceback (most recent call last)
<ipython-input-2-d27a4e3c0526> in <module>
----> 1 from sklearn import preprocessing

ModuleNotFoundError: No module named 'sklearn'
```

scikit-learn library 설치

■ Anaconda Prompt에서

- 가상환경 목록 확인
: conda env list
- 가상환경 활성화
: conda activate data_eng
- 설치된 library 목록 확인
: conda list
(scikit-learn library 없는 것 확인)
- scikit-learn library 설치
: conda install [scikit-learn](#)

실습2.2 표준화

■ scikit-learn의 preprocessing

- scale() 메서드
: 표준화 메소드

```
import pandas as pd  
from sklearn import preprocessing
```

```
housing = pd.read_csv('housing.csv')
```

```
standardized = preprocessing.scale(housing)
```

```
pd.DataFrame(standardized, columns=['Price', 'Sq.feet', 'Age'])
```

	Price	Sq.feet	Age
0	-1.084852	-0.232737	-1.231895
1	-0.271449	-0.495505	0.992360
2	1.336439	1.222592	0.136877
3	-0.980812	-1.488824	-1.060798
4	1.000674	0.994475	1.163456

정규화

■ 정규화normalization

- 정규화는 값들을 특정 범위, 주로 [0,1]로 스케일링하는 것
- MinMax Scaling $x' = (x - \min) / (\max - \min)$
: 값을 0 ~ 1 사이로 변환

■ 정규화는 언제 적용?

- 데이터의 분포를 모르거나 변수의 분포가 종의 형태(a bell curve)를 띄는 가우시안 분포가 아니라는 것을 알았을 때 사용하기 좋은 기법
- 예를 들어, 최근접 이웃알고리즘(K-nearest neighbors)이나 인공신경망 같은 방법

실습3. 정규화

■ MinMax 스케일링 연산 구현

- 최대값, 최소값 구해서
- $x' = (x - \min) / (\max - \min)$
구현하기

```
for i in range(df.shape[1]):  
    col = df.iloc[:, i]  
    col = (col - col.min()) / (col.max() - col.min())  
    df.iloc[:, i] = col  
    #print(col)  
df
```

	Price	Sq Feet	Age
0	0.000000	0.463259	0.000000
1	0.335938	0.366347	0.928571
2	1.000000	1.000000	0.571429
3	0.042969	0.000000	0.071429
4	0.861328	0.915868	1.000000

실습4. 정규화

■ scikit-learn의 preprocessing

- MinMaxScaler()
- fit_transform()
- 실습3과 동일한 연산 구현

```
import pandas as pd
from sklearn import preprocessing
```

```
housing = pd.read_csv('housing.csv')
```

```
normalized = preprocessing.MinMaxScaler().fit_transform(housing)
```

```
pd.DataFrame(normalized, columns=['Price', 'Sq.feet', 'Age'])
```

	Price	Sq.feet	Age
0	0.000000	0.463259	0.000000
1	0.335938	0.366347	0.928571
2	1.000000	1.000000	0.571429
3	0.042969	0.000000	0.071429
4	0.861328	0.915868	1.000000

정규성 검정

■ 정규성 검정 Normality Test 이란

- 데이터가 정규분포를 따르는지 검정하는 것

■ 정규성 검정 방법

- 시각화 : histogram, Q-Q plot 등
- 통계 검정 : 샤피로-윌크 검정(Shapiro-Wilk test) 등

시각화를 이용한 정규성 검정

■ 정규분포 모양 비교법

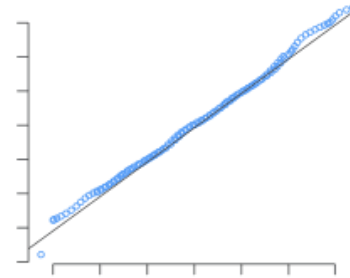
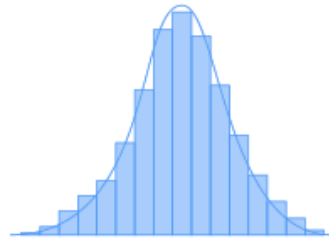
- 좌우대칭, 종모양에 가까운가?
- 시각화 도구로 histogram이나 kde (kernel density estimate) line으로 모양 비교

■ Q-Q (Quantile-Quantile) plot

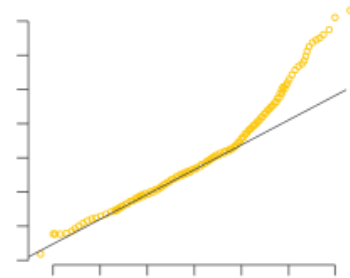
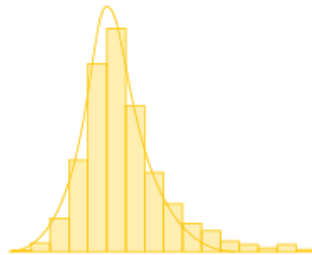
- 점들이(실데이터) 기준 선분 주위에 모여있는지 여부로 판단

시각화 사례 분석

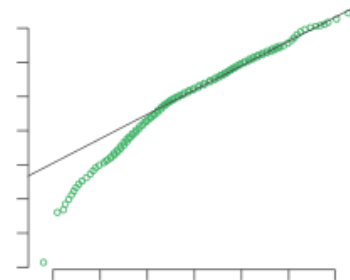
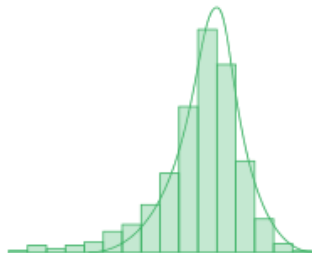
Normally distributed data



Right-skewed data



Left-skewed data



정규분포 모양 비교법

■ 정규분포 모양 비교

- Histogram이나 ked (kernel density estimate) line을 그려 정규분포 모양과 비교

■ Library

- seaborn의 histplot() : histogram 그리기
- seaborn의 kdeplot() : kde line 그리기
- seaborn의 distplot() : histogram과 kde line을 함께 그리기
(단, deprecated function)

실습5.1 데이터셋

■ seaborn titanic 데이터셋

```
import pandas as pd
import seaborn as sns
```

```
df = sns.load_dataset('titanic')
df.head()
```

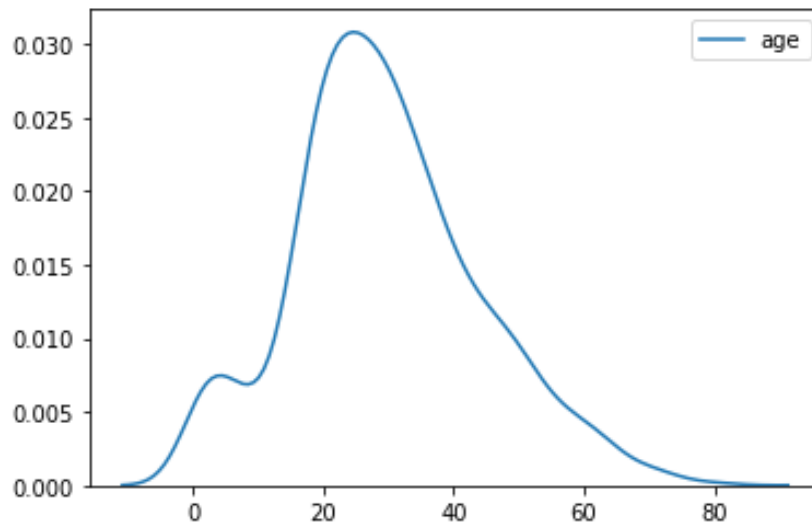
	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True

실습5.2 kdeplot()

■ kde line 그리기

- 'age'열에 대해서 누락데이터 클리닝 후 kde line 그리기

```
sns.kdeplot(df['age'], dropna())
```

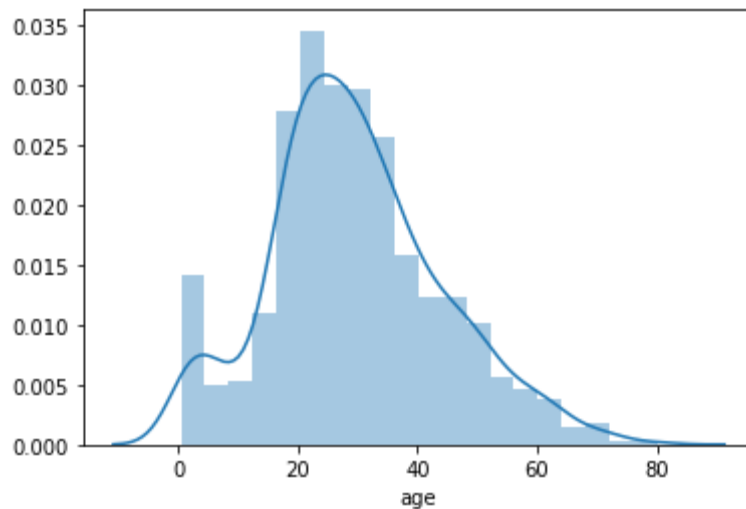


실습5.3 distplot()

■ histogram과 kde line 그리기

- ‘age’열에 대해서 그리기

```
#histogram & kde line  
#           kernel density estimate line  
sns.distplot(df['age'].dropna()) #, kde=False)
```



Q-Q plot

■ Q-Q plot Quantile-Quantile plot

- x축이 이론적 분포 즉, 기준이 되는 분포
: 즉, 정규분포를 따르는 데이터 z-score
- y 축이 비교하고자 하는 분포, 비교 대상 분포(실제 값)

■ Library

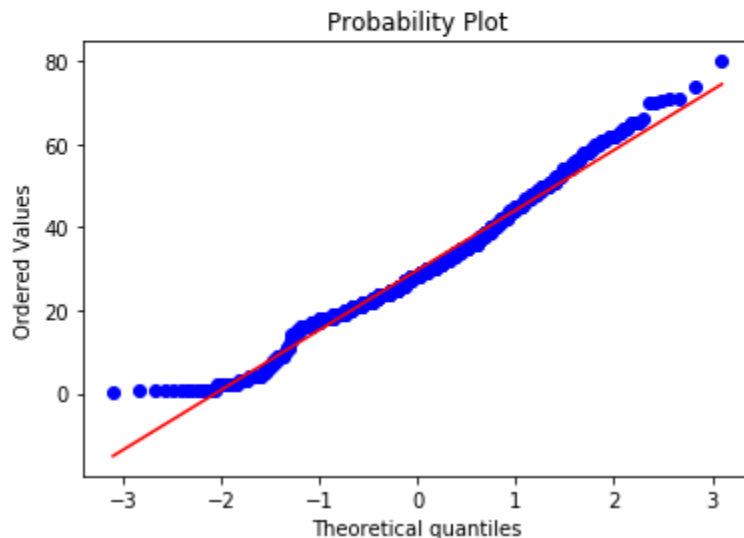
- `scipy` : 과학 관련 컴퓨팅에 사용되는 오픈 소스 파이썬 라이브러리
- `scipy.stats` : 다양한 통계적 기능을 제공하는 모듈
- `scipy.stats.probplot( )` : `matplotlib` 기반으로 Q-Q plot을 그린다.

실습5.4 Q-Q plot 그리기

■ `scipy.stats.probplot()`

- 실제값이 기준선 가까이 분포하는지 여부 판단

```
#Q-Q plot
import matplotlib.pyplot as plt
import scipy.stats as stats
stats.probplot(df['age'].dropna(), plot=plt)
plt.show()
```



Dimensionality Reduction

■ 차원 Dimension

- 데이터 마이닝 관점에서는 Observation을 표현하기 위한 Feature의 개수가 차원이 된다

■ ‘차원이 증가한다는 것’의 의미

- 개, 고양이를 분류하기 위한 머신러닝 모델을 설계할 때, 눈 모양과 귀 모양 2개의 특성으로 분류하면 2차원이 된다
- 이때 성능 향상을 위해 털 색상을 추가하면 3차원으로 차원이 증가하게 된다
- 타이타닉 데이터세트에 적용해보면?

분류 예제

■ 개와 고양이 분류

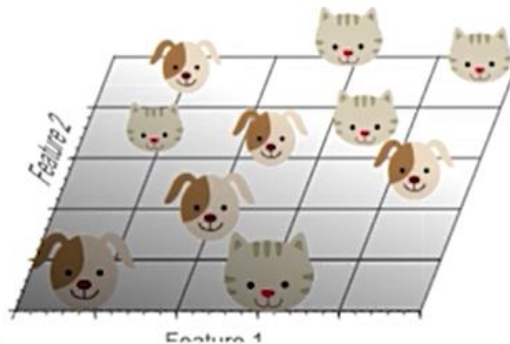
- 1,2 차원에서는 선형 분류가 어려운 경우

- [1 feature]: 1D



➡ 선형 분리 불가능

- [2 feature]: 2D



➡ 선형 분리 불가능

출처

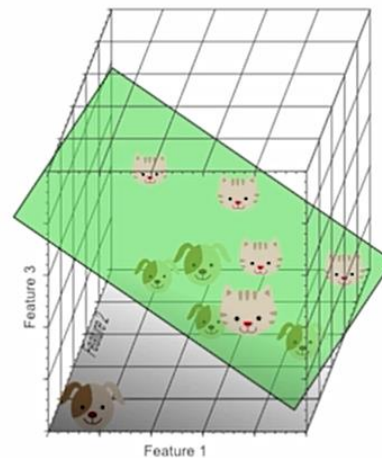
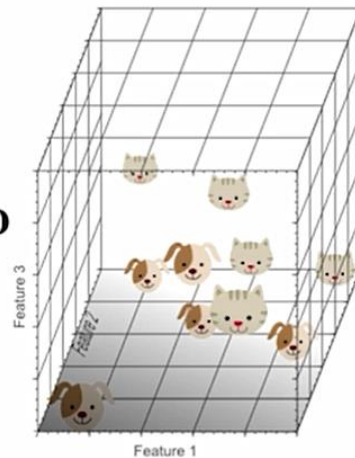
<https://www.visiondummy.com/2014/04/curse-dimensionality-affect-classification/>

차원 증가

■ 개와 고양이 분류

- 3차원에서는 선형 분류가 가능해진다.
- 즉, 차원을 증가시켜 선형 분류할 수 있도록 모델의 성능이 향상된다.

• [3 feature]: 3D



선형 분리 가능 !

출처

<https://www.visiondummy.com/2014/04/curse-dimensionality-affect-classification/>

차원의 저주

■ 차원의 저주 Curse of Dimensionality

- 변수가 늘어나고 차원이 커지면서 발생하는 문제
- -> 차원 축소 필요

■ 차원을 증가시키고 싶은 유혹

- 개, 고양이 분류 모델을 설계할 때, 눈 모양, 귀 모양 2개의 특성으로 훈련시키고 분류했더니 원하는 충분한 성능이 나오지 않으면, 털 색상, 발톱 모양을 추가하여 2차원에서 4차원으로 차원을 증가시키고 싶은 유혹이 생긴다.
- 그러면, 언제까지 차원을 증가시키는 것이 좋은가?
- 마냥 증가시키는 것이 정답은 아님을 지적하는 것이 ‘차원의 저주’ 개념이다.

차원의 저주-정리1

■ 차원이 증가하면

- 필요한 데이터 수가 지수함수적으로 증가해야 한다는 것을 의미
- 즉, 머신러닝 관점에서는 차원 증가에 따라 성능 향상에 필요한 최소한의 훈련데이터의 규모가 구하기 어려울 정도로 커진다고 보면 됨

ex) 각 변수가 가질 수 있는 값이 **3개**일 때

1	2	3
---	---	---

1차원 : 3개

1	2	3
4	5	6
7	8	9

2차원 : 9개

19	20	21
22	23	24
25	26	27

3차원 : 27개

...

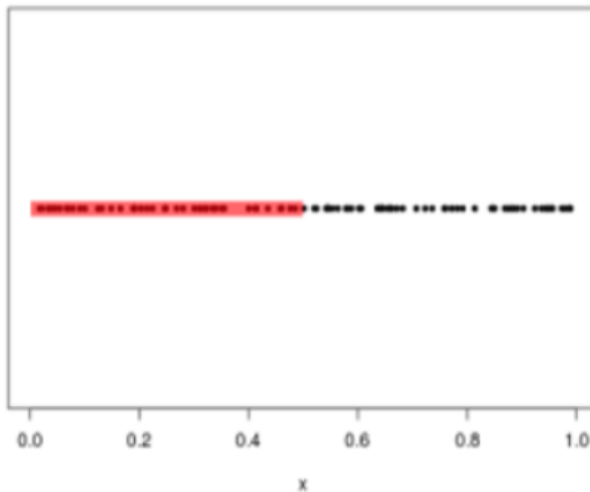
N차원 : **3의 n승** 개

차원의 저주-정리2

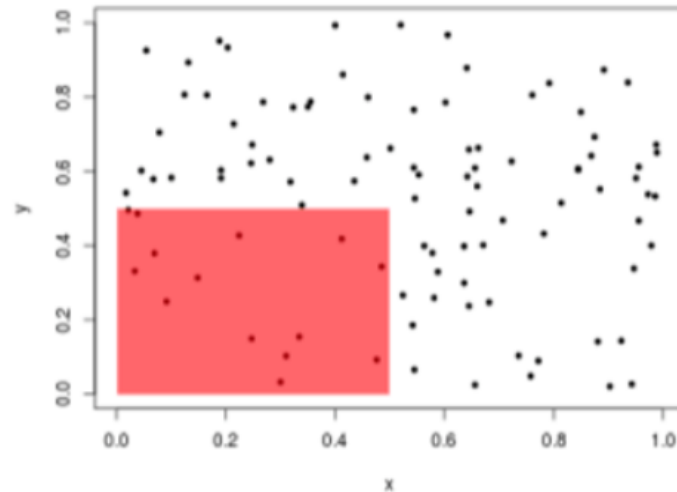
■ 데이터의 크기가 정해진 경우

- 정해진 데이터 크기에서 차원이 증가한다는 것은 공간의 희소성이 증가하며 데이터의 밀도가 감소하는 것을 의미

1-D: 42% of data captured.



2-D: 14% of data captured.

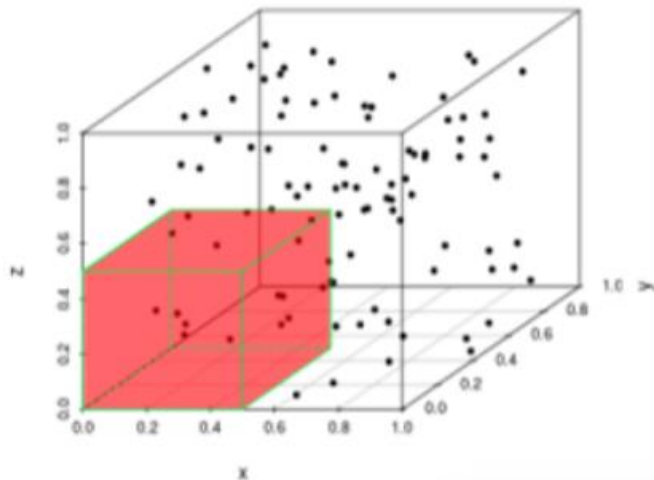


차원의 저주-정리2

■ 데이터의 크기가 정해진 경우

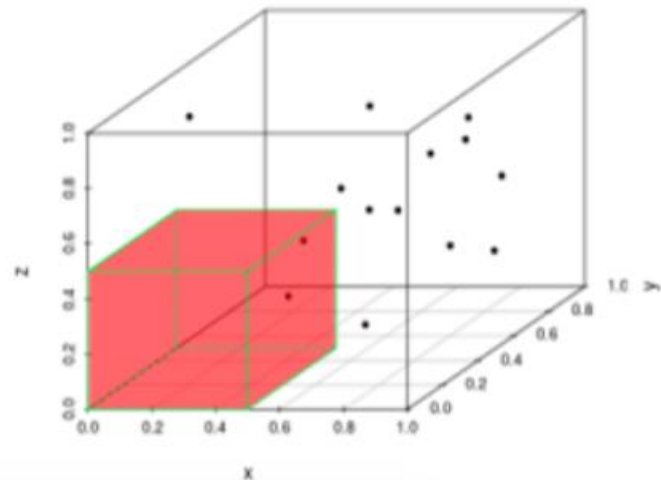
- 차원이 증가한다는 것은 공간의 희소성이 증가하며 데이터의 밀도가 감소하는 것을 의미

3-D: 7% of data captured.



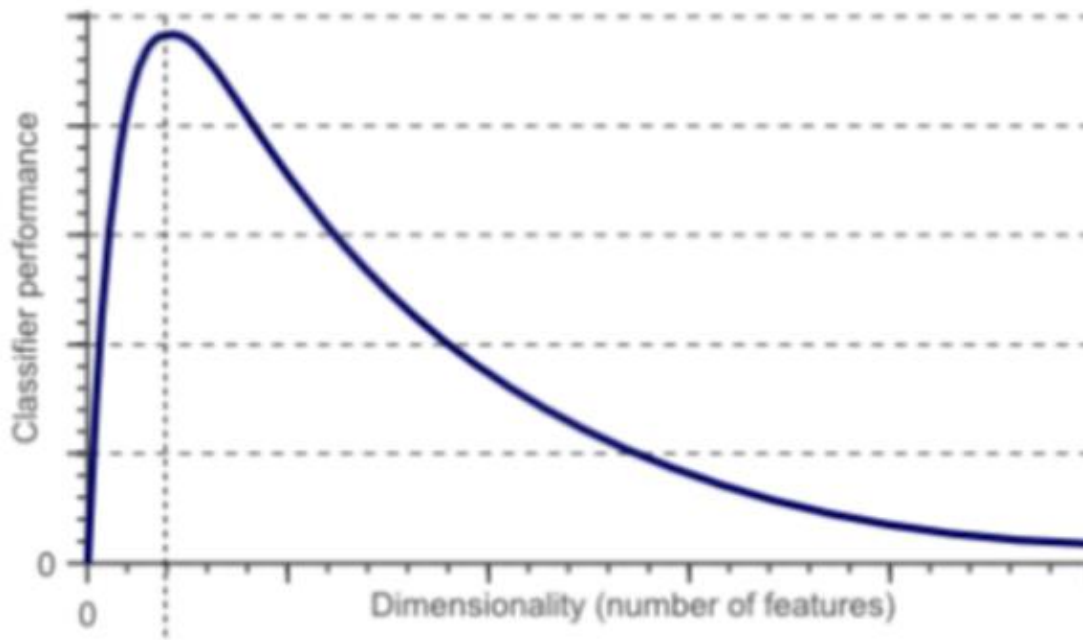
4-D: 3% of data captured.

t = 0



차원의 저주-정리3

- 공간을 설명하기 위한 데이터가 부족할 경우
 - 데이터 크기가 고정되어 있을 때, 차원이 증가할 수록 성능이 감소



Dimensionality Reduction

■ 차원 축소 Dimensionality Reduction

- Feature Selection (특성선택, 변수선택)
: 가지고 있는 특성 중에서 훈련에 가장 유용한 특성을 선택하는 것을 말한다. (다른 표현으로는 원본 데이터의 불필요한 특징을 제거한다)
- Feature Extraction (특성추출, 변수추출)
: 원본 데이터의 특징들의 조합으로 새로운 특징을 생성한다

■ 차원 축소 평가방법

- 변수간의 correlation을 구하여 일반적으로 그 크기가 0.7이상이면 차원 축소가 필요한 것으로 평가 → DataFrame corr() 실행

실습6.1 데이터셋

■ seaborn iris 데이터셋

```
import pandas as pd
import seaborn as sns
```

```
df = sns.load_dataset('iris')
df.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

실습6.2 누락데이터 확인

■ 누락데이터 없어 그대로 적용

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 150 entries, 0 to 149
```

```
Data columns (total 5 columns):
```

#	Column	Non-Null Count	Dtype
0	sepal_length	150 non-null	float64
1	sepal_width	150 non-null	float64
2	petal_length	150 non-null	float64
3	petal_width	150 non-null	float64
4	species	150 non-null	object

```
dtypes: float64(4), object(1)
```

```
memory usage: 6.0+ KB
```

실습6.3 correlation

■ DataFrame corr()

- 수치형 변수간의 correlation을 -1~1 사이의 값으로 표현하여 1에 가까울수록 상관관계가 높음 (음수는 반비례)

```
corr = df.corr().round(2)
corr
```

	sepal_length	sepal_width	petal_length	petal_width
sepal_length	1.00	-0.12	0.87	0.82
sepal_width	-0.12	1.00	-0.43	-0.37
petal_length	0.87	-0.43	1.00	0.96
petal_width	0.82	-0.37	0.96	1.00

차원 축소 사례

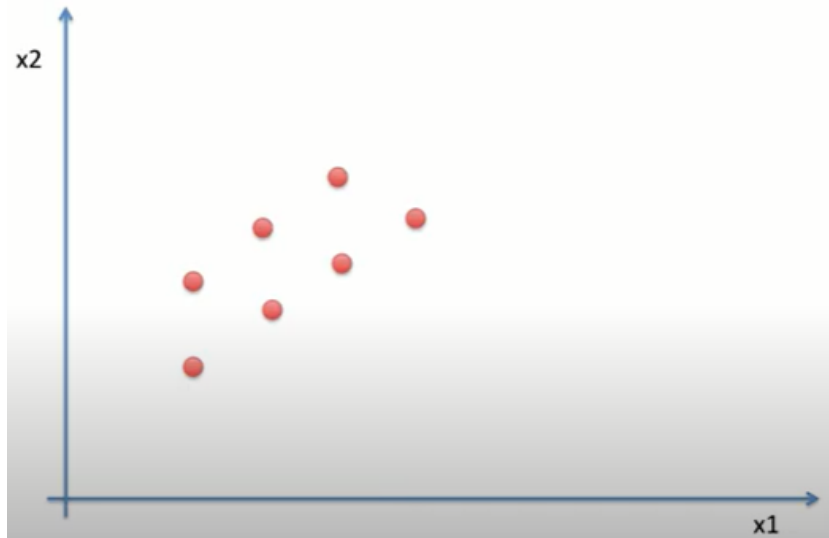
■ Housing 데이터에서

- 방의 개수가 많을 수록 집의 평수가 큰 경우, 두 변수는 상관관계가 높다고 할 수 있음
- Feature Selection (특성선택)
: 방의 개수를 변수에서 제거하는 방법
- Feature Extraction (특성추출)
: 방의 개수와 집의 평수를 연산하여 새로운 집의 크기 변수를 추출하는 방법
* 키와 몸무게가 독립이 아니라고 판단하여 체구라는 새로운 파생변수 생성 -> $\text{체구} = 0.2 * \text{키} + 0.8 * \text{몸무게}$

주성분 분석 (PCA)

■ PCA (Principal Component Analysis)

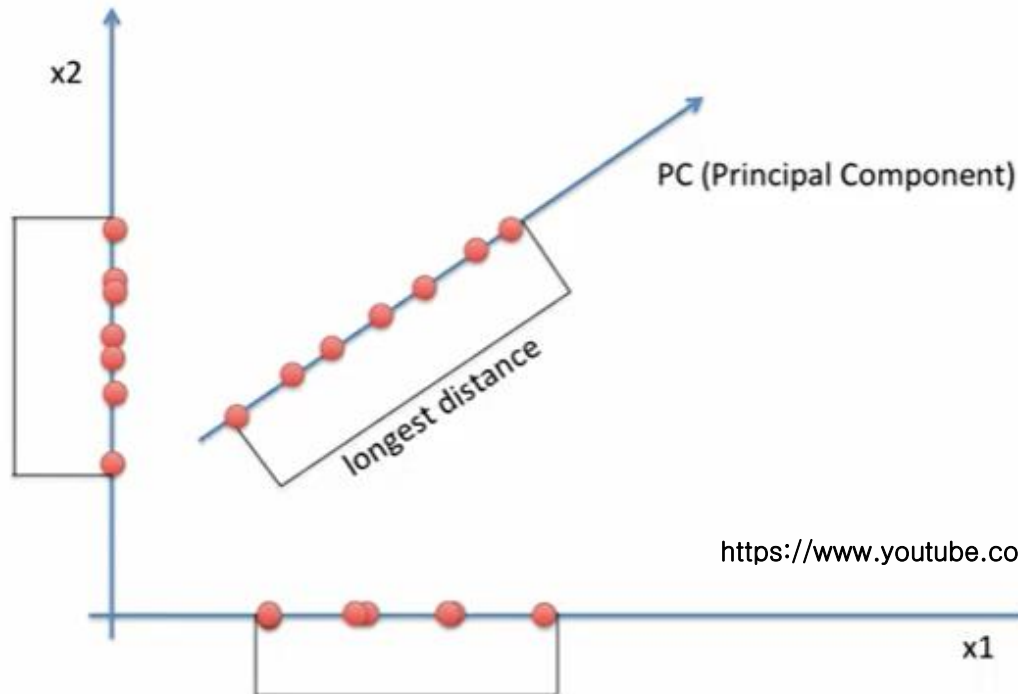
- 측정된 변수들의 선형 조합(Linear Combination)에 의해 대표적인 주성분을 만들어 차원(Dimension)을 줄이는 방법
- 사례 : 2차원을 1차원으로 축소하기



PCA 2차원 사례

■ 주성분(PC) 구하기

- 2차원의 점들을 1차원으로 축소하였을 때 중복 없이 최대한 잘 퍼져있도록 하는 1차원 주성분 (Principal Component)을 구하기



<https://www.youtube.com/watch?v=DUJ2vwjRQag>

참고도서

➤reference

[1] 파이썬을 활용한 데이터길들이기

- 프로그래밍인사이트

[2] 모두의 데이터분석

- 길벗

[3] 파이썬머신러닝 판다스데이터분석

- 정보문화사

End