

사물인터넷 (Internet of Things)

김태운

목차

- Sensing Layer: 센서 및 센싱/분석기술
 - 데이터 정제

Sensing Layer: 데이터 정제

- 데이터 정제(Data cleaning/cleansing/scrubbing)
 - 데이터 정제 실습을 위한 사전 준비
 - 구글 코랩
 - pandas 라이브러리
 - 실습 환경 설정: 구글 코랩
 - 구글 코랩
 - 클라우드 기반의 주피터 노트북 개발 환경
 - 웹 브라우저에서 텍스트와 프로그램 코드를 자유롭게 작성할 수 있는 일종의 온라인 텍스트 에디터
 - 무료로 사용 가능
 - 노트북을 최대 5개만 동시에 사용(활성화)할 수 있음
 - 노트북 하나를 12시간 이상 연결(활성화 상태로 유지)할 수 없음. 12시간 후 다시 연결 해야 함

구글 코랩을 사용하지 않고, 본인의
PC/노트북을 사용해도 됩니다.



Sensing Layer: 데이터 정제

- 데이터 정제(Data cleaning/cleansing/scrubbing)
 - 실습 내용
 1. Handling missing values
 2. Scaling and normalization
 3. Inconsistent data entry
 4. Anomaly detection
 5. Noise reduction
 - 실습에 사용한 데이터 셋(csv) 출처:
 - Kaggle Data Cleaning Challenge 등

Sensing Layer: 데이터 정제

■ 데이터 정제(Data cleaning/cleansing/scrubbing)

• 1. Handling missing value (누락된 값 처리하기)

• 데이터 셋 다운로드

- <https://www.kaggle.com/code/rtatman/data-cleaning-challenge-handling-missing-values/data>
- Building_Permits.csv 및 NFL Play by Play 2009-2017 (v4).csv 파일을 구글 드라이브에 다운받고, colab에 연결하기

```
# 구글 드라이브 마운트
from google.colab import drive
drive.mount('/content/drive')
```

• 구글 드라이브에 저장된 데이터 셋 불러오기

```
[3] # 필요 라이브러리 불러오기
import pandas as pd
import numpy as np

# 데이터 읽어오기
# - events that occurred in American Football games
nfl_data = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/DataCleaningCSV/NFL Play by Play 2009-2017 (v4).csv")
# - building permits issued in San Francisco
sf_permits = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/DataCleaningCSV/Building_Permits.csv")

# set seed for reproducibility (시드값을 고정)
np.random.seed(0)
```

2009년~2017년까지의 NFL 경기 관련 데이터를 수집한 데이터 셋을 사용합니다!

```
/usr/local/lib/python3.7/dist-packages/IPython/core/interactiveshell.py:2882: DtypeWarning: Columns (25,51) have mixed types.Specify
exec(code_obj, self.user_global_ns, self.user_ns)
/usr/local/lib/python3.7/dist-packages/IPython/core/interactiveshell.py:2882: DtypeWarning: Columns (22,32) have mixed types.Specify
exec(code_obj, self.user_global_ns, self.user_ns)
```

Sensing Layer: 데이터 정제

■ 데이터 정제(Data cleaning/cleansing/scrubbing)

- 1. Handling missing value (누락된 값 처리하기)
 - NFL 데이터 중 일부만 출력하여, 데이터 값 살펴보기

```
[4] # NFL 데이터 셋에서 무작위로 5개의 샘플(행)을 추출하여 화면에 출력  
nfl_data.sample(5)
```

	Date	GameID	Drive	qtr	down	time	TimeUnder	TimeSecs	PlayTimeDiff	SideofField	...	yacEPA	Home_WP_pre	Aw
244485	2014-10-26	2014102607	18	3	1.0	00:39	1	939.0	12.0	TB	...	1.240299	0.225647	
115340	2011-11-20	2011112000	22	4	1.0	06:47	7	407.0	44.0	OAK	...	NaN	0.056036	
68357	2010-11-14	2010111401	8	2	NaN	00:23	1	1823.0	0.0	CLE	...	NaN	0.365307	
368377	2017-09-24	2017092405	24	4	1.0	08:48	9	528.0	8.0	CLE	...	1.075660	0.935995	
384684	2017-11-05	2017110505	11	2	1.0	09:15	10	2355.0	0.0	DEN	...	NaN	0.928474	

5 rows × 102 columns

- 일부 데이터가 누락 되었으며, 누락된 값은 NaN(Not a Number)으로 표기됨 // 여기가 바로 Missing values!

Sensing Layer: 데이터 정제

■ 데이터 정제(Data cleaning/cleansing/scrubbing)

- 1. Handling missing value (누락된 값 처리하기)
 - 전체 데이터에서 누락된 값이 몇개인지 확인

```
# 전체 행의 수, 열의 수 카운트하기
print("- NFL 전체 행의 수: {}, 열의 수: {}".format(nfl_data.shape[0], nfl_data.shape[1]))

# 각 column별로 missing value의 개수 카운트하기
missing_values_count = nfl_data.isnull().sum()

# 처음 10개 column에서 missing values가 몇개나 있는지를 출력하기
print("- 처음 10개 COL에서 누락된 값의 개수 확인:")
print(missing_values_count[0:10])

# total count
total_cell_cnt = nfl_data.shape[0]*nfl_data.shape[1]
print("\n")
print("- 전체 셀의 수: {}, 누락된 값/셀 수: {} ({:2.2%})".format(total_cell_cnt,
    missing_values_count.sum(), missing_values_count.sum() / (total_cell_cnt)))
```

- NFL 전체 행의 수: 407688, 열의 수: 102

- 처음 10개 COL에서 누락된 값의 개수 확인:

Date	0
GameID	0
Drive	0
qtr	0
down	61154
time	224
TimeUnder	0
TimeSecs	224
PlayTimeDiff	444
SideofField	528
dtype:	int64

- 전체 셀의 수: 41584176, 누락된 값/셀 수: 10342875 (24.87%)

Sensing Layer: 데이터 정제

■ 데이터 정제(Data cleaning/cleansing/scrubbing)

• 1. Handling missing value (누락된 값 처리하기)

• Missing values의 원인?

- 값이 존재하지 않는 경우
 - 예: 자녀가 없는 가정에서 “첫째 자녀의 키는?”이라는 질문에 답을 할 수 없음
 - 이러한 경우, missing value를 유추하는 것은 의미 없음
- 값이 누락된 경우
 - 예: 실수로 값을 입력하지 않거나, 삭제된 경우
 - 이러한 경우, 동일한 데이터 셋의 다른 값들 또는 외부의 도움 등을 통해 missing value를 유추할 수 있음

• NFL 데이터 셋에서의 missing value

- PenalizedTerm 열의 경우, 경기 중 패널티를 받지 않았다면 값 자체가 존재하지 않을 수 있음
- TimeSecs 열의 경우, 경기 남은 시간(Seconds left in game at the time of the play)을 의미하므로 (실수로) 기록되지 않은 경우로 볼 수 있음

• sf_permits 데이터 셋에서의 missing value

- Street Number Suffix (거리 이름 뒤에 붙는 접미사) 열에서 missing value 발생 원인: 실수 또는 해당 값이 원래 존재하지 않음
- Zipcode (우편번호) 열에서 missing value 발생 원인: 100% 실수

Missing value가 있는 행(row)을 삭제하고 무시할 수도 있습니다. 하지만, 전체 데이터 셋에서 삭제되는 행이 많다면, missing value를 어떻게든(?) 유추하여 사용하는 편이 좋습니다.



패널티가 없는 경기에서, 경기 데이터 기록관이 해당 값을 0으로 표기할 수도 있지만 값을 아예 기록하지 않을 수 있고, 이 경우 missing value가 발생함

Sensing Layer: 데이터 정제

■ 데이터 정제(Data cleaning/cleansing/scrubbing)

• 1. Handling missing value (누락된 값 처리하기)

- 가장 기초적인 접근법(Naïve Approach #1a): 누락된 값(NA 또는 NaN)이 포함된 행을 제거하기

```
# NA로 기록된 값이 있는 행(row)을 제거하기

# remove all the rows that contain a missing value
nfl_data_DropNaRow = nfl_data.dropna()
print("- NFL 전체 행의 수: {}, 열의 수: {}".format(nfl_data_DropNaRow.shape[0], nfl_data_DropNaRow.shape[1]))
```

- NFL 전체 행의 수: 0, 열의 수: 102

모든 행이 삭제됨...

- 가장 기초적인 접근법(Naïve Approach #1b): 누락된 값이 포함된 열을 제거하기

```
# NA로 기록된 값이 있는 열(column)을 제거하기

# remove all columns with at least one missing value
nfl_data_DropNaCol = nfl_data.dropna(axis=1)
print("- NFL 전체 행의 수: {}, 열의 수: {}".format(nfl_data_DropNaCol.shape[0], nfl_data_DropNaCol.shape[1]))
```

- NFL 전체 행의 수: 407688, 열의 수: 41

행의 수는 원본과 동일하고, 열의 수가 102
에서 41로 감소함

Sensing Layer: 데이터 정제

■ 데이터 정제(Data cleaning/cleansing/scrubbing)

- 1. Handling missing value (누락된 값 처리하기)
 - 누락된 셀을 0으로 채우기

```
# 누락된 (NA) 셀을 0으로 채우기
```

```
# replace all NA's with 0
```

```
nfl_data_NaZero = nfl_data.fillna(0)
```

```
np.random.seed(0)
```

```
nfl_data_NaZero.sample(5)
```

	Date	GameID	Drive	qtr	down	time	TimeUnder	TimeSecs	PlayTimeDiff	SideofField	...	yacEPA	Home_WP_pre	Away_WP_pre	Home_WP_post
244485	2014-10-26	2014102607	18	3	1.0	00:39	1	939.0	12.0	TB	...	1.240299	0.225647	0.774353	0.245582
115340	2011-11-20	2011112000	22	4	1.0	06:47	7	407.0	44.0	OAK	...	0.000000	0.056036	0.943964	0.042963
68357	2010-11-14	2010111401	8	2	0.0	00:23	1	1823.0	0.0	CLE	...	0.000000	0.365307	0.634693	0.384697
368377	2017-09-24	2017092405	24	4	1.0	08:48	9	528.0	8.0	CLE	...	1.075660	0.935995	0.064005	0.921231
384684	2017-11-05	2017110505	11	2	1.0	09:15	10	2355.0	0.0	DEN	...	0.000000	0.928474	0.071526	0.934641

수정된 데이터

	Date	GameID	Drive	qtr	down	time	TimeUnder	TimeSecs	PlayTimeDiff	SideofField	...	yacEPA	Home_WP_pre	Away_WP_pre	Home_WP_post
244485	2014-10-26	2014102607	18	3	1.0	00:39	1	939.0	12.0	TB	...	1.240299	0.225647	0.774353	0.245582
115340	2011-11-20	2011112000	22	4	1.0	06:47	7	407.0	44.0	OAK	...	NaN	0.056036	0.943964	0.042963
68357	2010-11-14	2010111401	8	2	NaN	00:23	1	1823.0	0.0	CLE	...	NaN	0.365307	0.634693	0.384697
368377	2017-09-24	2017092405	24	4	1.0	08:48	9	528.0	8.0	CLE	...	1.075660	0.935995	0.064005	0.921231
384684	2017-11-05	2017110505	11	2	1.0	09:15	10	2355.0	0.0	DEN	...	NaN	0.928474	0.071526	0.934641

원본 데이터

Sensing Layer: 데이터 정제

■ 데이터 정제(Data cleaning/cleansing/scrubbing)

- 1. Handling missing value (누락된 값 처리하기)
 - 인접한 값으로 대체하기
 - 측정 데이터가 논리적인 순서를 가진 경우에는 누락된 값 직전 또는 직후의 값을 대체하여 사용할 수 있음
 - 예: 1초 단위로 온도를 측정하는 상황에서 $t-1$, t , $t+1$ 값 중에서 t 에 해당하는 값이 누락된 경우, $t-1$ 때의 값 또는 $t+1$ 때의 값을 t 의 값으로 사용할 수 있음
 - 누락된 값을 같은 열에서 바로 뒤따르는 값으로 대체하여 사용하기

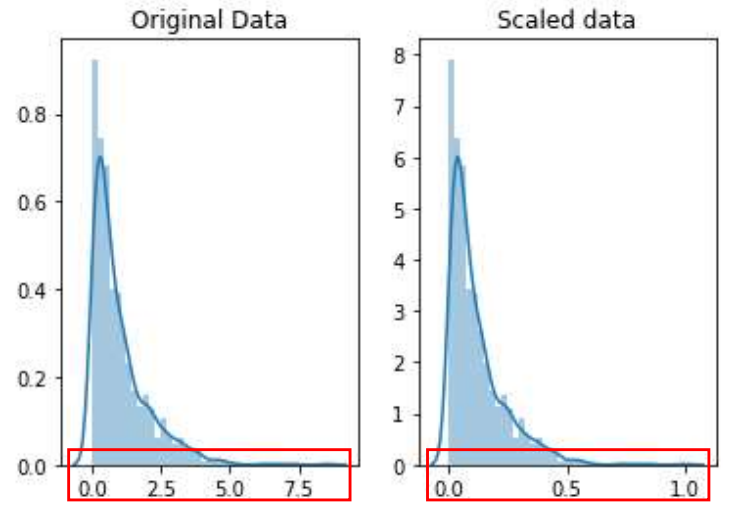
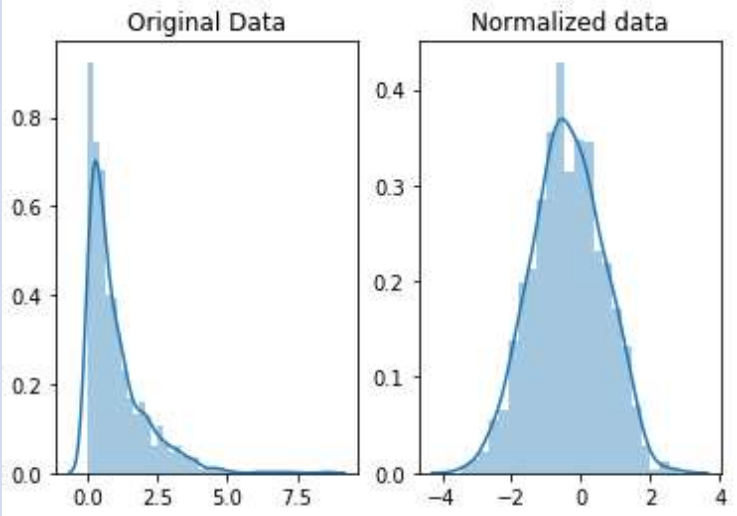
누락된 셀의 값을, 같은 열에서 바로 아래의 값으로 대체하기

```
# replace all NA's the value that comes directly after it in the same column,  
# then replace all the remaining na's with 0  
nfl_data_NaFollowingValue = nfl_data.fillna(method = 'bfill', axis=0).fillna(0)
```

- 데이터의 특성을 이해하고 있다면, 이 외에 다른 영리한(?) 방법을 사용해서 누락된 값을 복원할 수 있음

Sensing Layer: 데이터 정제

- 데이터 정제(Data cleaning/cleansing/scrubbing)
 - 2. Scaling and normalization

	Scaling	Normalization
공통점	원본 데이터를 조작/변형하여, 원하는 속성을 갖는 데이터로 바꾸는 것	
차이점	원본 데이터 값의 범위를 바꾸는 것 예: 0~100 사이의 값을 0~10 사이의 값으로 바꾸는 것	원본 데이터 값의 분포를 정규 분포로 바꾸는 것 예: Exponential 분포를 따르는 데이터를 Gaussian 분포로 변환
예시		

Sensing Layer: 데이터 정제

- 데이터 정제(Data cleaning/cleansing/scrubbing)
 - 2. Scaling and normalization
 - 필요 라이브러리 호출

```
# modules we'll use
import numpy as np

# for Box-Cox Transformation
from scipy import stats

# for min_max scaling
from mlxtend.preprocessing import minmax_scaling

# plotting modules
import seaborn as sns
import matplotlib.pyplot as plt
```

Sensing Layer: 데이터 정제

■ 데이터 정제(Data cleaning/cleansing/scrubbing)

- 2. Scaling and normalization

- Scaling 의미 및 필요성

- 원 데이터의 범위를 특정 범위로 줄이는 것(또는 늘리는 것)
- 데이터 값들 간의 거리에 기반한 분석을 수행할 때 유용함
 - 예 1: SVM(support vector machine), KNN(k-nearest neighbors)

- 서로 다른 scale의 값을 가지는 데이터를 혼용해서 사용할 때 반드시 필요함

- 예 2:

- [거래/Deal A] 1달러에 구매한 물건을 2달러에 판매한 경우, 수익 = $2 - 1 = 1$
- [거래/Deal B] 100원에 구매한 물건을 200원에 판매한 경우, 수익 = $200 - 100 = 100$

* there are two deals (A & B) available
- deal A yields \$1 profit
- deal B yields ₩100 profit

* Optimally determine which deal is better

max. profitFromA * x + profitFromB * y

subject to:

$x \in \{0, 1\}, y \in \{0, 1\}$

$x + y = 1$

where:

profitFromA = 1

profitFromB = 100

Optimal decision (from the left math. opt.):

100 > 1 이므로 [Deal B]가 더 많은 수익을 냈고,
그러므로 Deal A가 아닌 Deal B를 진행해야 한다...?

Sensing Layer: 데이터 정제

■ 데이터 정제(Data cleaning/cleansing/scrubbing)

- 2. Scaling and normalization

- Scaling

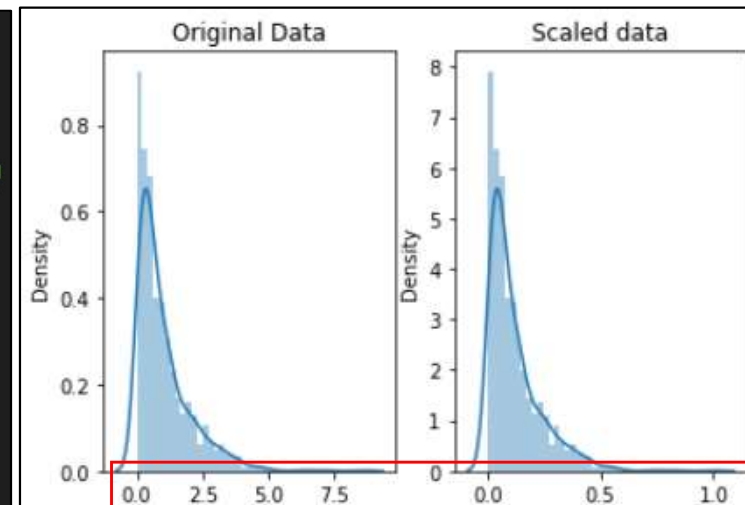
- ML Xtend 라이브러리를 사용해서 (무작위로 생성한) 데이터 값의 범위를 0.0~1.0사이로 (축소하는) scaling 하기

```
# Exponential 분포로 부터 1000개의 데이터를 무작위로 생성함
# (참고. 여기서는 어떤 분포를 썼는지는 중요하지 않음)
original_data = np.random.exponential(size = 1000)

# ML Xtend (Python library of useful tool for the data science tasks)
# min-max scale the data between 0 and 1
scaled_data = minmax_scaling(original_data, columns = [0])

# plot both together to compare
fig, ax=plt.subplots(1,2)
sns.distplot(original_data, ax=ax[0])
ax[0].set_title("Original Data")
sns.distplot(scaled_data, ax=ax[1])
ax[1].set_title("Scaled data")
```

seaborn.distplot: 데이터의 분포를 plot (histogram과 유사)



데이터 범위(x축의 값)는 바뀌었지만, 분포 형태는 바뀌지 않음

- 동작 방식

- data := 원래 데이터 셋(1-D array 라고 가정)
- HOW ?

Sensing Layer: 데이터 정제

■ 데이터 정제(Data cleaning/cleansing/scrubbing)

- 2. Scaling and normalization
- Scaling

- 동작 방식

- $data := \text{원래 데이터 셋(1-D array 라고 가정)}$
- $min_value := \min(data)$ // 주어진 데이터 셋에서의 최솟값
- $max_value := \max(data)$ // 주어진 데이터 셋에서의 최댓값

- $scaled_data[i] = (data[i] - min) / (max_value - min_value)$

Sensing Layer: 데이터 정제

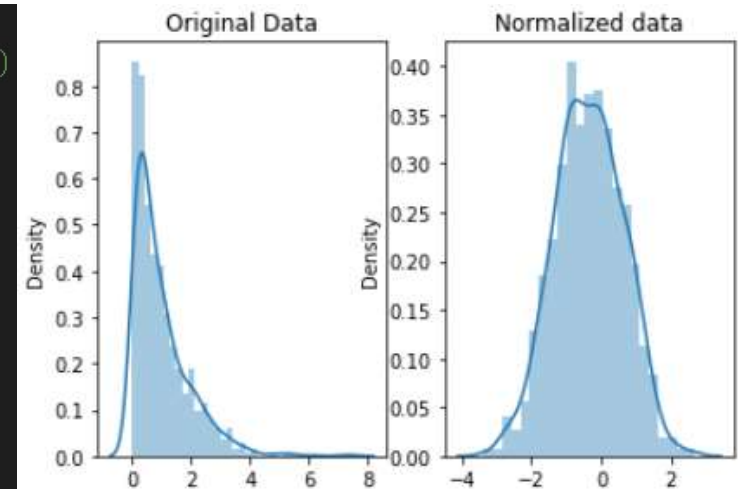
■ 데이터 정제(Data cleaning/cleansing/scrubbing)

- 2. Scaling and normalization
- Normalization(정규화) 의미 및 필요성
 - 정규 분포(Normal, Gaussian distribution)를 따르도록, 원본 데이터를 변환/변형 하는 것
 - 정규 분포를 가정하는 기계 학습 또는 통계 분석 기법을 사용할 때 유용함
 - 예: Linear regression (LR), linear discriminant analysis (LDA) and Gaussian naive Bayes.

```
# Exponential 분포로 부터 1000개의 데이터를 무작위로 생성함
# (참고. 여기서는 어떤 분포를 썼는지가 중요함. 정규화 이후에 바뀐 분포와 비교해보기!)
original_data = np.random.exponential(size = 1000)

# normalize the exponential data with boxcox
normalized_data, fitted_lambda = stats.boxcox(original_data)

# plot both together to compare
fig, ax=plt.subplots(1,2)
sns.distplot(original_data, ax=ax[0])
ax[0].set_title("Original Data")
sns.distplot(normalized_data, ax=ax[1])
ax[1].set_title("Normalized data")
```



Sensing Layer: 데이터 정제

■ 데이터 정제(Data cleaning/cleansing/scrubbing)

- 2. Scaling and normalization
- Normalization(정규화)
 - 직전 예제에서 normalization을 위해 `scipy.stats.boxcox` 함수를 사용

`scipy.stats.boxcox`

`scipy.stats.boxcox(x, lmbda=None, alpha=None, optimizer=None)`

- Box-Cox 정규화 기법:

$$y_i^{(\lambda)} = \begin{cases} \frac{y_i^\lambda - 1}{\lambda} & \text{if } \lambda \neq 0, \\ \ln y_i & \text{if } \lambda = 0, \end{cases}$$

- lambda 값을 입력하지 않고 boxcox 함수를 호출한 경우, 최적의 lambda 값을 스스로 계산함 (즉, 최적으로 Normal distribution에 fitting 하는 lambda 값을 스스로 계산)
- 참고: Box-Cox 기법의 자세한 설명은 Blog/YouTube 또는 아래의 논문을 참고

An Analysis of Transformations

By G. E. P. Box

and

D. R. Cox

University of Wisconsin

Birkbeck College, University of London

Sensing Layer: 데이터 정제

■ 데이터 정제(Data cleaning/cleansing/scrubbing)

• 3. Inconsistent data entry (데이터 일관성)

- 동일한 데이터임에도 불구하고, 서로 다른 형태로 표현된 경우
- 데이터를 처리하는 과정에서 서로 다른 데이터로 잘못 처리되는 경우가 발생할 수 있음
 - 예: “서울”, “서 울” 은 같은 데이터이지만 string compare에서 mismatch 발생

• 사전 작업: 라이브러리 설치 및 import

```
"""
https://www.kaggle.com/code/rtatman/data-cleaning-challenge-inconsistent-data-entry/notebook
"""

!pip install fuzzywuzzy

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting fuzzywuzzy
  Downloading fuzzywuzzy-0.18.0-py2.py3-none-any.whl (18 kB)
Installing collected packages: fuzzywuzzy
Successfully installed fuzzywuzzy-0.18.0

# modules we'll use
import pandas as pd
import numpy as np

# helpful modules
import fuzzywuzzy
from fuzzywuzzy import process
import chardet

# set seed for reproducibility
np.random.seed(0)
```

Sensing Layer: 데이터 정제

- 데이터 정제(Data cleaning/cleansing/scrubbing)
 - 3. Inconsistent data entry (데이터 일관성)
 - 데이터 인코딩 방식을 확인하고, 정확한 인코딩을 적용하여 파일 읽기

```
# 구글 드라이브 마운트
from google.colab import drive
drive.mount('/content/drive')

Mounted at /content/drive

src_file = "/content/drive/MyDrive/Colab Notebooks/InconsistentDataCSV/PakistanSuicideAttacks Ver 11 (30-November-2017).csv"
```

```
# 데이터 인코딩 방식을 확인하고, 정확한 인코딩을 적용하여 파일 읽기
```

```
# look at the first ten thousand bytes to guess the character encoding
with open(src_file, 'rb') as rawdata:
    result = chardet.detect(rawdata.read(100000))
```

```
# check what the character encoding might be
print('인코딩 정보: ', result)
```

```
# 정확한 인코딩을 적용하여 파일 읽기
```

```
# read in our dat
suicide_attacks = pd.read_csv(src_file, encoding='Windows-1252')
```

```
# get all the unique values in the 'City' column
cities = suicide_attacks['City'].unique()
```

```
# sort them alphabetically and then take a closer look
cities.sort()
cities
```

```
인코딩 정보: {'encoding': 'Windows-1252', 'confidence': 0.73, 'language': ''}
array(['ATTOCK', 'Attock ', 'Bajaur Agency', 'Bannu', 'Bhakkar ', 'Buner',
      'Chakwal ', 'Chaman', 'Charsadda', 'Charsadda ', 'D. I Khan',
      'D.G Khan', 'D.G Khan ', 'D.I Khan', 'D.I Khan ', 'Dara Adam Khel',
      'Dara Adam khel', 'Fateh Jang', 'Ghalianai, Mohmand Agency ',
      'Gujrat', 'Hangu', 'Haripur', 'Hayatabad', 'Islamabad',
      'Islamabad ', 'Jacobabad', 'KURRAM AGENCY', 'Karachi', 'Karachi ',
      'Karak', 'Khanewal', 'Khuzdar', 'Khyber Agency', 'Khyber Agency ',
      'Kohat', 'Kohat ', 'Kuram Agency ', 'Lahore', 'Lahore ',
      'Lakki Marwat', 'Lakki marwat', 'Lasbela', 'Lower Dir', 'MULTAN',
      'Malakand ', 'Mansehra', 'Mardan', 'Mohmand Agency',
      'Mohmand Agency ', 'Mohmand agency', 'Mosal Kor, Mohmand Agency',
      'Multan', 'Muzaffarabad', 'North Waziristan', 'North waziristan',
      'Nowshehra', 'Orakzai Agency', 'Peshawar', 'Peshawar ', 'Pishin',
      'Poonch', 'Quetta', 'Quetta ', 'Rawalpindi', 'Sargodha',
      'Sehwan town', 'Shabqadar-Charsadda', 'Shangla ', 'Shikarpur',
      'Sialkot', 'South Waziristan', 'South waziristan', 'Sudhanoti',
      'Sukkur', 'Swabi ', 'Swat', 'Swat ', 'Taftan',
      'Tangi, Charsadda District', 'Tank', 'Tank ', 'Taunsa',
      'Tirah Valley', 'Totalai', 'Upper Dir', 'Wagah', 'Zhob', 'bannu',
      'karachi', 'karachi ', 'lakki marwat', 'peshawar', 'swat'],
      dtype=object)
```

Sensing Layer: 데이터 정제

- 데이터 정제(Data cleaning/cleansing/scrubbing)
 - 3. Inconsistent data entry (데이터 일관성)
 - 일관성 오류 타입 확인
 - 예 1: 'Lakki Marwat' vs 'Lakki marwat' (두 번째 단어의 첫문자가 대문자 또는 소문자)
 - 예 2: 'Lahore' (앞뒤 공백 없음) vs 'Lahore ' (뒤에 공백 있음)
 - 해결 방법?
 - “예 1” 경우에 대한 해결 방법?
 - “예 2” 경우에 대한 해결 방법?

Sensing Layer: 데이터 정제

■ 데이터 정제(Data cleaning/cleansing/scrubbing)

• 3. Inconsistent data entry (데이터 일관성)

• 일관성 오류 타입 확인

- 예 1: 'Lakki Marwat' vs 'Lakki marwat' (두 번째 단어의 첫문자가 대문자 또는 소문자)
- 예 2: 'Lahore' (앞뒤 공백 없음) vs 'Lahore ' (뒤에 공백 있음)

• 해결 방법?

- “예 1” 경우에 대한 해결 방법: 모든 문자를 소문자로 변경(대문자로 변경하는 것도 가능)

```
# convert to lower case  
suicide_attacks['City'] = suicide_attacks['City'].str.lower()
```

- “예 2” 경우에 대한 해결 방법: 단어 앞뒤 공백을 제거

```
# remove leading and trailing white spaces  
suicide_attacks['City'] = suicide_attacks['City'].str.strip()
```

Sensing Layer: 데이터 정제

- 데이터 정제(Data cleaning/cleansing/scrubbing)
 - 3. Inconsistent data entry (데이터 일관성)
 - 데이터 일관성 오류 타입 확인
 - 예: “d.i khan” vs “d. i khan” // 비 일관적인 표기법
 - 예: “school” vs “schol” // 단순 오타
 - 해결 방법?

Sensing Layer: 데이터 정제

■ 데이터 정제(Data cleaning/cleansing/scrubbing)

• 3. Inconsistent data entry (데이터 일관성)

• 데이터 일관성 오류 타입 확인

- 예: “d.i khan” vs “d. i khan” // 비 일관적인 표기법
- 예: “school” vs “schol” // 단순 오타

• 해결 방법?

- 유사성이 높은 string을 찾아서 통일된 방식의 string으로 치환
- fuzzywuzzy 패키지: 두 string간 유사성을 수치로 표현
- 예: cities 데이터 셋에서 “d.i khan” 과 유사성이 가장 높은 10개 데이터를 화면에 출력

```
# get the top 10 closest matches to "d.i khan"
matches = fuzzywuzzy.process.extract("d.i khan", cities, limit=10, scorer=fuzzywuzzy.fuzz.token_sort_ratio)

# take a look at them
matches

[('d. i khan', 100),
 ('d.i khan', 100),
 ('d.g khan', 88),
 ('khanewal', 50),
 ('sudhanoti', 47),
 ('hangu', 46),
 ('kohat', 46),
 ('dara adam khel', 45),
 ('chaman', 43),
 ('mardan', 43)]
```

참고: d.g khan은 d.i khan과 다른 데이터임

유사성이 90% 이상인 값을 d.i khan으로 변경하자!

Sensing Layer: 데이터 정제

■ 데이터 정제(Data cleaning/cleansing/scrubbing)

- 3. Inconsistent data entry (데이터 일관성)
- 참고:
- fuzzywuzzy 패키지의 문자열 유사성 평가 기법은 Levenshtein Distance 값을 사용 <https://pypi.org/project/fuzzywuzzy/>

FuzzyWuzzy

Fuzzy string matching like a boss. It uses [Levenshtein Distance](#) to calculate the differences between sequences in a simple-to-use package.

• Levenshtein Distance

- 하나의 문자열을 다른 문자열로 바꾸기 위해서 필요한 단일 char 수정의 횟수를 측정함

Levenshtein distance

From Wikipedia, the free encyclopedia

In [information theory](#), [linguistics](#), and [computer science](#), the **Levenshtein distance** is a [string metric](#) for measuring the difference between two sequences. Informally, the Levenshtein distance between two words is the [minimum number of single-character edits \(insertions, deletions or substitutions\) required to change one word into the other](#). It is named after the Soviet mathematician [Vladimir Levenshtein](#), who considered this distance in 1965.^[1]

```
FuzzyWuzzy의 리턴값(% 유사도)  
= 100%(String * ength-LevenshteinDistance) / StringLength
```

Sensing Layer: 데이터 정제

- 데이터 정제(Data cleaning/cleansing/scrubbing)
 - 3. Inconsistent data entry (데이터 일관성)
 - 유사성이 90% 이상인 값을 d.i khan으로 변경하기

```
# function to replace rows in the provided column of the provided dataframe
# that match the provided string above the provided ratio with the provided string
def replace_matches_in_column(df, column, string_to_match, min_ratio = 90):
    # get a list of unique strings
    strings = df[column].unique()

    # get the top 10 closest matches to our input string
    matches = fuzzywuzzy.process.extract(string_to_match, strings,
                                         limit=10, scorer=fuzzywuzzy.fuzz.token_sort_ratio)

    # only get matches with a ratio > 90
    close_matches = [matches[0] for matches in matches if matches[1] >= min_ratio]

    # get the rows of all the close matches in our dataframe
    rows_with_matches = df[column].isin(close_matches)

    # replace all rows with close matches with the input matches
    df.loc[rows_with_matches, column] = string_to_match

    # let us know the function's done
    print("All done!")

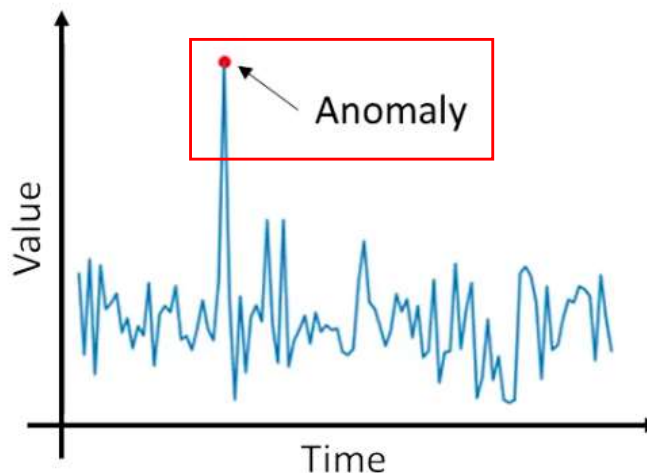
# use the function we just wrote to replace close matches to "d.i khan" with "d.i khan"
replace_matches_in_column(df=suicide_attacks, column='City', string_to_match="d.i khan")
```

Sensing Layer: 데이터 정제

■ 데이터 정제(Data cleaning/cleansing/scrubbing)

• 4. Anomaly detection (이상 탐지)

- 예상과 다르거나, 또는 대부분의 데이터와는 다른 패턴을 보이는 값을 구분하는 것
- 기존에 수집한 데이터를 사용하여 모델을 생성하고, 이를 기반으로 정상 값과 비정상 값을 구분
- 사물인터넷 센서의 일시적 오류 또는 비정상적인 이벤트의 발생으로 인해 대부분의 데이터와 다른 특징을 가지는 값이 관측된 경우, 이러한 값을 비정상적으로 분류하기 위함
- ‘비정상’ 또는 ‘이상’ 이라는 표현은 영어로 anomaly, outlier 등으로 표현함
 - 그 외에도 분야에 따라 discordant observations, exceptions, aberrations, surprises, peculiarities, contaminants 로도 표현함
 - 단, anomaly와 outlier가 가장 널리 사용됨
- 필요성
 - 비정상 값/패턴을 구분해서 제거해야 하는 경우: 기계학습/딥러닝 등의 학습 모델을 개발할 때, 일반화 성능을 높이기 위해서는 이러한 비정상적인 값을 배제할 필요가 있음 => 노이즈 또는 오류 데이터 제거
 - 비정상 값/패턴을 탐지할 필요가 있는 경우: 보안(신용카드 사기, 사이버 침입, 사이버 테러 등), 시스템 관제(시스템 고장, 물리적/논리적 오류 탐지), 의료(MRI 사진에서 종양 탐지) 등의 분야에서는 평소와 다른 비정상 패턴을 탐지하는 것을 목적으로 함 => 이상 탐지

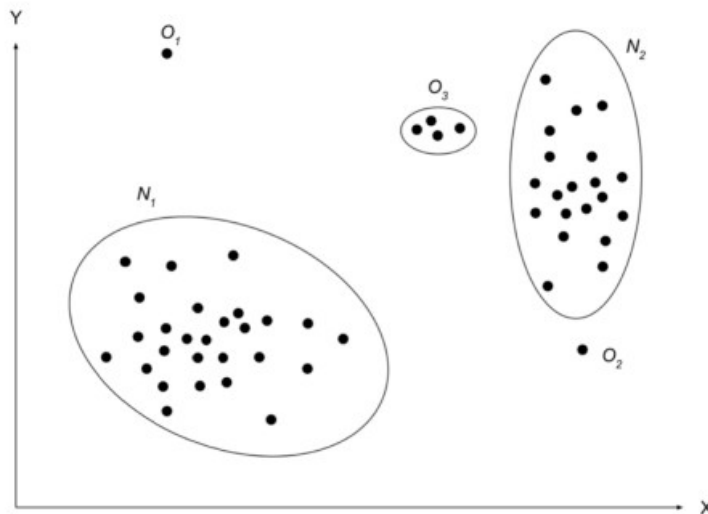


Sensing Layer: 데이터 정제

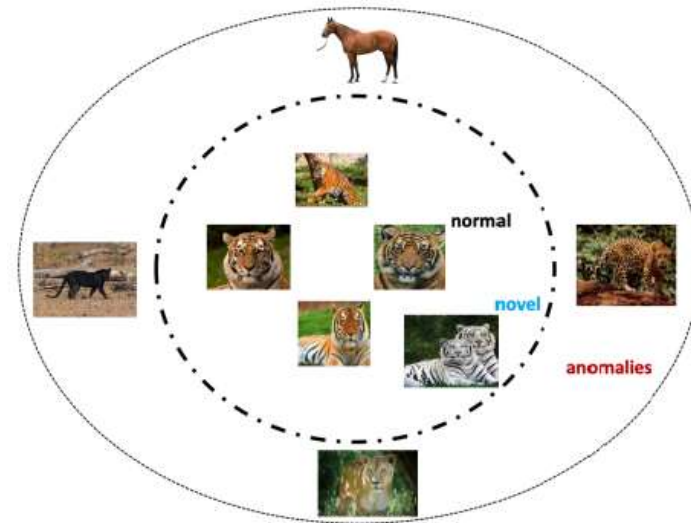
■ 데이터 정제(Data cleaning/cleansing/scrubbing)

• 4. Anomaly detection (이상 탐지)

- Anomaly (이상, 이상치, 비정상) 은 expected normal/general pattern 을 따르지 않는 데이터이고, 다르게 표현하면, 대부분의 데이터와는 다른 특징을 가지는 값을 말함



대부분의 측정 값이 N1 또는 N2 그룹에 속함. 따라서 그 외 O1, O2, O3에 속하는 값을 anomaly로 볼 수 있음



호랑이 사진을 모은 데이터 셋에서, 그 외 동물 사진은 anomaly로 볼 수 있음

(참고: novel/novelty라는 것은 normal group 내에서 새로운 특징을 가지고 있는 데이터를 말함)

Sensing Layer: 데이터 정제

■ 데이터 정제(Data cleaning/cleansing/scrubbing)

- 4. Anomaly detection (이상 탐지)
 - 이상 탐지 알고리즘

순서	알고리즘	설명
1	Normal pattern 정의하기	<ul style="list-style-type: none">• (대부분의 데이터가 normal 이라는 가정 하에) 기존에 획득한 데이터를 활용하여, 대부분의 데이터가 보여주는 패턴을 찾고 이를 normal pattern으로 정의하기• 만약, 사전에 정의된 normal pattern이 있다면, 이를 사용하기
2	이상 탐지를 위한 기준 설정	<ul style="list-style-type: none">• 임의의 데이터 값에 대해, normal pattern 에 속하는지 여부를 판단하기 위한 기준 및 임계치 설정 <p>* Normal pattern에 속하는지 여부가 binary(0/1)로 결정되지 않고, 확률적으로 계산되는 것이 일반적이는데, 이 때 몇 %의 확률 이상/미만이면 anomaly라고 판단할지에 대한 기준을 설정할 필요가 있음</p>
3	신규 데이터에 대한 이상 탐지	<ul style="list-style-type: none">• 신규로 획득한 데이터에 대해, anomaly 인지 여부를 판단

- 이상 탐지 알고리즘은 unsupervised learning (clustering) 또는 supervised learning으로 구현할 수 있음
- 이후에 등장하는 실습 코드는 supervised learning 으로 구현한 예임
 - Labelled data set을 통해서 이상 탐지를 위한 기준을 설정하고, 이를 사용해서 신규 데이터에 대한 이상 탐지를 수행함

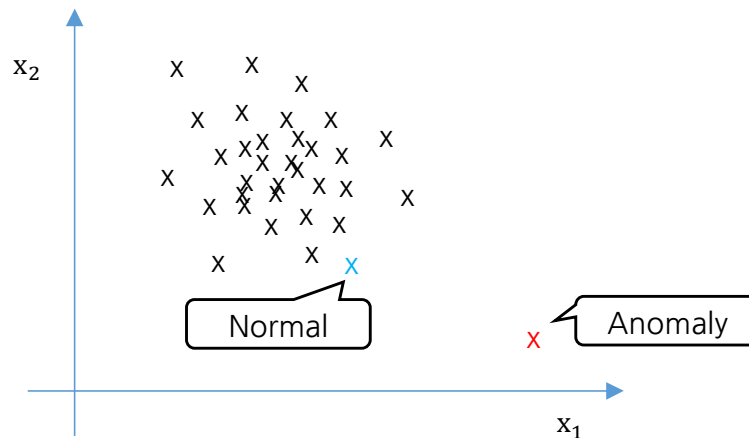
Sensing Layer: 데이터 정제

■ 데이터 정제(Data cleaning/cleansing/scrubbing)

• 4. Anomaly detection (이상 탐지)

- m 개의 값으로 구성된 데이터 셋을 가정: $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$
- 데이터 셋에 포함된 각 $x^{(m)}$ 은 n 개의 feature를 가짐: $x^{(m)} = [x_1, x_2, \dots, x_n]$
- 예: aircraft engine features
 - Feature #1: x_1 = heat generated (averaged over s consecutive samples)
 - Feature #2: x_2 = vibration intensity (averaged over s consecutive samples)

각 샘플은 s 개의 연속된 센싱 데이터의 평균에 해당함 (노이즈 영향을 줄이는 한가지 방법)



Normal 데이터의 pattern을 찾고, anomaly를 탐지하는 방법은?



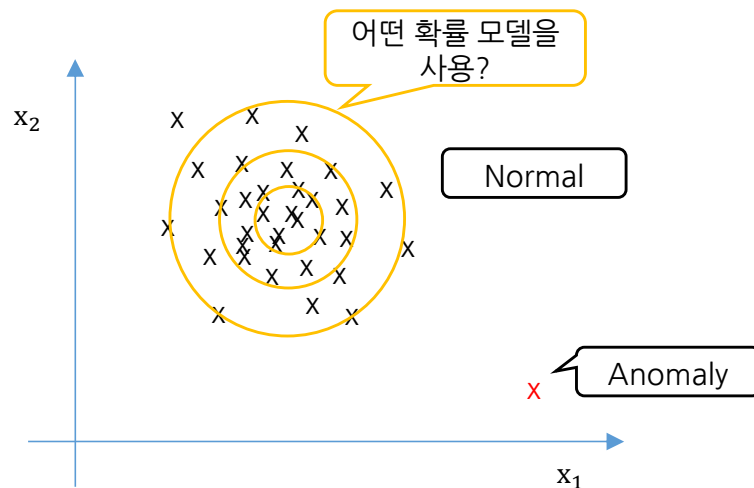
Sensing Layer: 데이터 정제

■ 데이터 정제(Data cleaning/cleansing/scrubbing)

• 4. Anomaly detection (이상 탐지)

• Density estimation

- 데이터 셋의 model을 나타내는 pdf (probability density function) $p(x)$ 를 찾고, 새롭게 획득한 x_{new} 값에 대해서 아래와 같이 anomaly 여부를 판단
- IF $p(x_{\text{new}}) < \text{epsilon}$: anomaly detected
- IF $p(x_{\text{new}}) \geq \text{epsilon}$: it's a normal data
- 여기서 epsilon은 임계치(threshold)로서, normal data와 anomaly data를 구분하는 기준이 됨



Sensing Layer: 데이터 정제

■ 데이터 정제(Data cleaning/cleansing/scrubbing)

• 4. Anomaly detection (이상 탐지)

• Gaussian distribution (정규 분포)

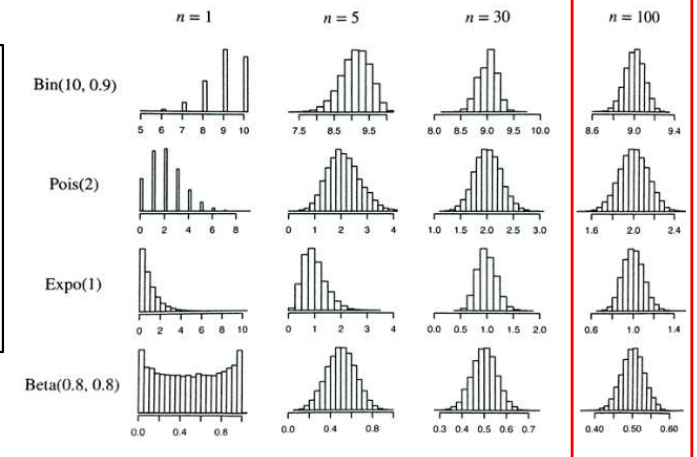
- 가장 널리 사용되는 확률 분포 중 하나로, 자연에서 관찰 가능한 현상들이 대부분 정규분포를 따름
- 참고: 중심극한정리(CLT, Central Limit Theorem)

Theorem 46.2 (Central Limit Theorem for Means) Let X_1, \dots, X_n be independent and identically distributed (i.i.d.) random variables. Then, for n large, their (sample) mean

$$\bar{X}_n = \frac{X_1 + \dots + X_n}{n}$$

is approximately normally distributed.

표본이 많아질수록 표본평균은 정규분포에 근사함



- 모집단의 표본 분포에 관계 없이, sample 들의 mean 값이 정규 분포를 따름
- 예:
 - 일반적으로, 샘플 값을 획득할 때 하나의 값 만을 사용하면 오차가 크게 발생할 수 있어서, 연속적으로 n개의 값을 측정한 뒤 그 평균값을 사용함
 - Feature #1: x_1 = heat generated (averaged over s consecutive samples)
 - Feature #2: x_2 = vibration intensity (averaged over s consecutive samples)
 - 이렇게 획득한 평균값들은 중심극한정리에 따라 정규 분포를 따르게 됨
- 따라서, 획득한 데이터를 정규 분포로 모델링 한 후, 새로 획득한 정보가 모델링한 정규 분포에 얼마나 벗어나 있는지를 기준으로 anomaly 여부를 판단하자!
- 획득한 데이터가 정규 분포를 따르지 않는다면, 'normalization' 기법을 사용하여 정규 분포로 변환

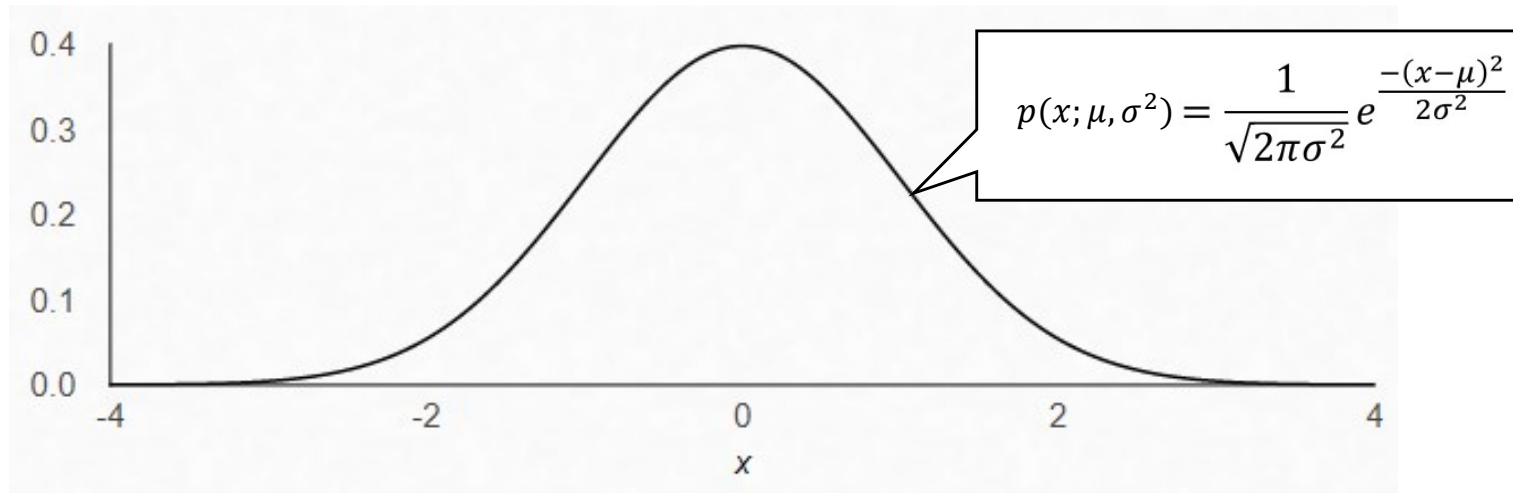
Sensing Layer: 데이터 정제

■ 데이터 정제(Data cleaning/cleansing/scrubbing)

• 4. Anomaly detection (이상 탐지)

• Gaussian distribution (정규 분포)

- 임의의 $x \in \mathbb{R}$ 이 mean μ 및 variance (분산) σ^2 을 가지는 정규 분포를 따를 경우, 다음과 같이 표기: $x \sim N(\mu, \sigma^2)$ 또는 $x \sim N(\mu, \sigma)$. 이 때, σ 를 표준편차(standard deviation, SD)라고 함
- 예: $X \sim N(0,1)$ 을 따르는 랜덤 변수의 pdf (probability density function, 확률 밀도 함수) $p(x)$

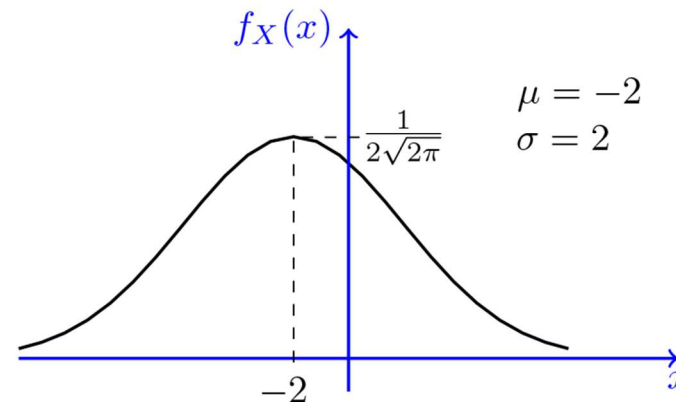
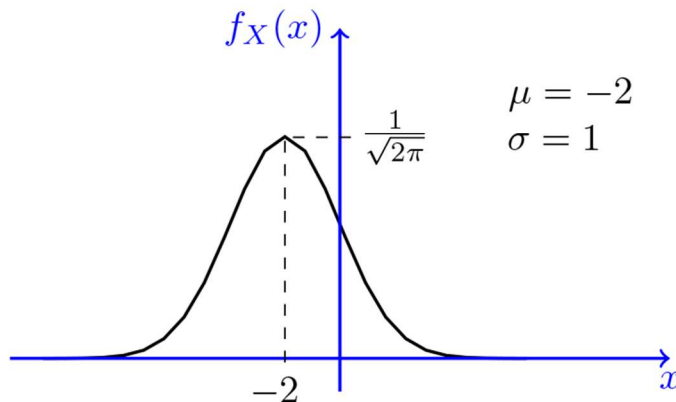
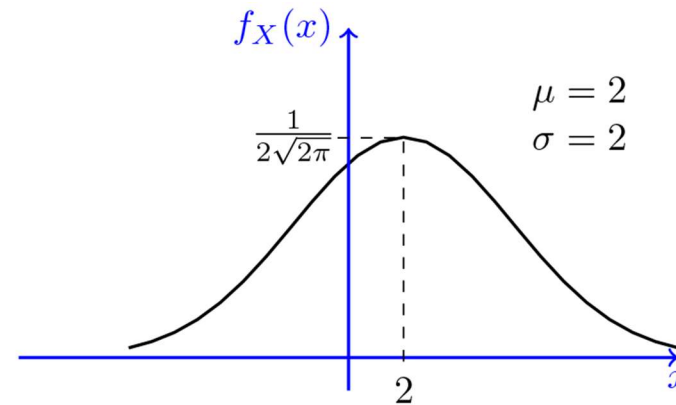
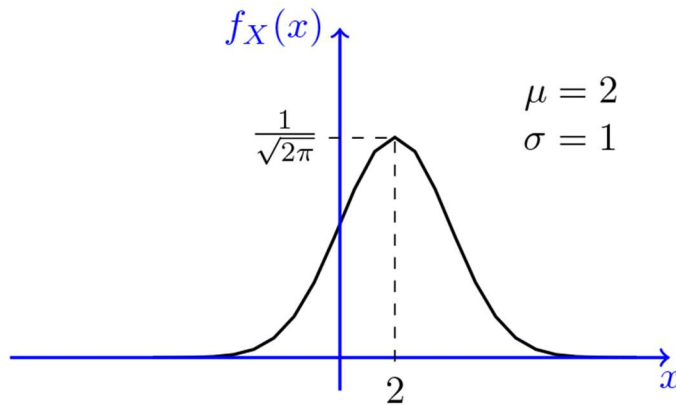


- $N(\mu, \sigma^2)$ 에서 샘플링 한 임의의 x 에 대해, x 가 x' 근처에서 샘플링 되었을 확률 $p(x = x'; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$
 - 따라서
 - $p(x=x')$ 의 값이 크면 x' 값이 p 로 묘사되는 모델에 적합하고(normal samples)
 - $p(x=x')$ 의 값이 작으면 x' 값이 p 로 묘사되는 모델에 부적합하다(anomaly samples)
 - 즉, $p(x=x')$ 를 x' 값이 모델 $p(x; \mu, \sigma^2)$ 에 적합한 정도를 표현하는 값으로 해석할 수 있음

Sensing Layer: 데이터 정제

■ 데이터 정제(Data cleaning/cleansing/scrubbing)

- 4. Anomaly detection (이상 탐지)
 - Gaussian distribution (정규 분포)
 - mean, std_dev 에 따른 pdf 변화
 - mean 값에 따라서 중심 위치가 좌/우로 이동함
 - std_dev 값이 커지면 곡선의 최대 높이가 낮아지고, 좌우로 퍼지는 형태로 변화함



Sensing Layer: 데이터 정제

■ 데이터 정제(Data cleaning/cleansing/scrubbing)

• 4. Anomaly detection (이상 탐지)

- (수집한 샘플이 정규 분포를 따른다는 가정 하에)

수집한 샘플 데이터로 부터 정규 분포 파라미터 예측하기(μ, σ parameter estimation)

- $x \in \mathbb{R}^1$ 인 경우 (즉, x 는 스칼라 값이고, 각 데이터 샘플은 하나의 feature로 구성됨)

- $\mu = \frac{1}{m} \sum_{i=1}^m x^{(i)}$

- $\sigma = \frac{1}{m} \sum_{i=1}^m (x^{(i)} - \mu)^2$

- 정확하게는 $m-1$ 로 나눠야 unbiased estimation 이지만, m 이 충분히 크다는 가정하에 $m-1$ 대신 m 을 사용

- 신규 샘플 x' 에 대해서 모델 적합도 평가 방법: (즉, abnormal/normal 판단 방법)

- IF $p(x = x'; \mu, \sigma^2) < \varepsilon$: 이상(abnormal)

- ELSE: normal

- $x \in \mathbb{R}^n$ 인 경우 (즉, x 는 스칼라 값이고, 각 데이터 샘플은 n 개의 feature로 구성됨)

- 각 x 에 대해서 feature j 별로 μ_j, σ_j 계산

- $\mu_j = \frac{1}{m} \sum_{i=1}^m x_j^{(i)}$ 를 $j = 1, \dots, n$ 에 대해서 반복해서 각 feature에 대한 mean을 계산

- $\sigma_j = \frac{1}{m} \sum_{i=1}^m (x_j^{(i)} - \mu_j)^2$ 를 $j = 1, \dots, n$ 에 대해서 반복해서 각 feature에 대한 std_dev을 계산

- 정확하게는 $m-1$ 로 나눠야 unbiased estimation 이지만, m 이 충분히 크다는 가정하에 $m-1$ 대신 m 을 사용

- 신규 샘플 $x' = [x'_1, x'_2, \dots, x'_n]$ 에 대해서 모델 적합도 평가 방법: (즉, abnormal/normal 판단 방법)

- $p(x = x'; \mu_1, \dots, \mu_n, \sigma_1^2, \dots, \sigma_n^2) = p(x_1 = x'_1; \mu_1, \sigma_1^2) \times \dots \times p(x_n = x'_n; \mu_n, \sigma_n^2) = \prod_{j=1}^n p(x_j = x'_j; \mu_j, \sigma_j^2)$ // 각 feature별로 적합도를 계산해서 결과를 곱함

- IF $p(x = x') < \varepsilon$: 이상(abnormal)

- ELSE: normal

Sensing Layer: 데이터 정제

■ 데이터 정제(Data cleaning/cleansing/scrubbing)

• 4. Anomaly detection (이상 탐지)

• 알고리즘 정리

1. 비정상 샘플 탐지에 적합한 feature 를 선택: x_1, x_2, \dots, x_n

2. Cross-Validation sample을 사용하여 parameter fitting (mean, std_dev 계산)

- 각각의 샘플 x 에 대해서 feature 별로 μ_j, σ_j 계산

- $\mu_j = \frac{1}{m} \sum_{i=1}^m x_j^{(i)}$ 를 $j = 1, \dots, n$ 에 대해서 반복해서 각 feature에 대한 mean을 계산

- $\sigma_j = \frac{1}{m} \sum_{i=1}^m (x_j^{(i)} - \mu_j)^2$ 를 $j = 1, \dots, n$ 에 대해서 반복해서 각 feature에 대한 std_dev를 계산

3. Cross-Validation sample을 사용한 임계치 계산: 사전에 획득한 샘플 및 샘플에 대한 normal/anomaly 여부를 판단한 결과를 기반으로 적합도 기준치(threshold, ϵ)을 계산

4. 새로 획득한 샘플에 대해서 모델 적합도 $p(x)$ 계산

- $$p(x) = \prod_{j=1}^n p(x_j; \mu_j, \sigma_j^2) = \prod_{j=1}^n \frac{1}{\sqrt{2\pi}\sigma_j} \exp\left(-\frac{(x_j - \mu_j)^2}{2\sigma_j^2}\right)$$

5. 새로 획득한 샘플에 대해서

- IF $p(x) < \epsilon$: 비정상(abnormal)

- ELSE : 정상(normal)

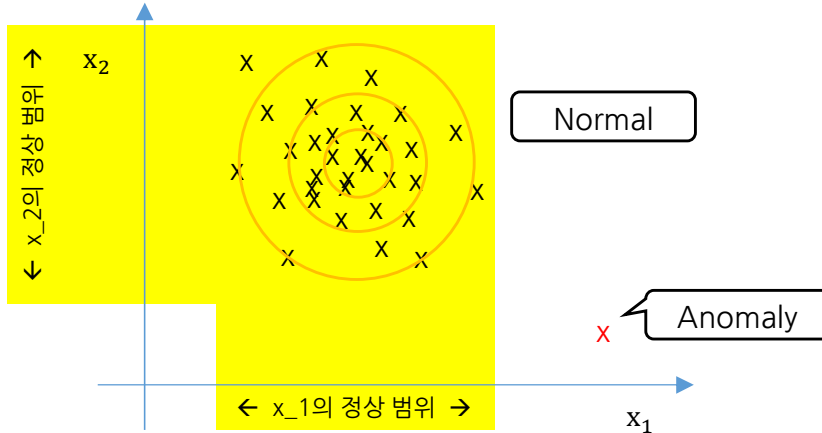
- 로 판단

Sensing Layer: 데이터 정제

■ 데이터 정제(Data cleaning/cleansing/scrubbing)

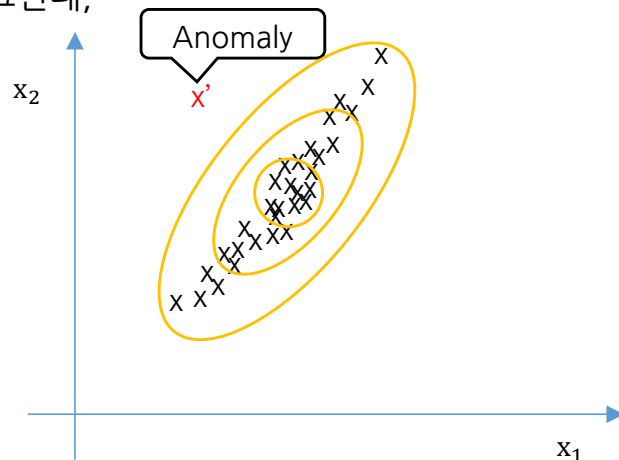
• 4. Anomaly detection (이상 탐지)

- 알고리즘 : 노란색 원의 중심에 가까울 수록 정상으로, 멀어질 수록 비정상으로 판단



- x₁ feature 만을 고려하여 모델 적합도 평가
- x₂ feature 만을 고려하여 모델 적합도 평가
- 위의 두 결과를 곱하여 최종 적합도 평가

- 그런데,



- x' 샘플의 경우,
- x₁ 만을 고려했을 때, 정상 범위에서 크게 벗어나지 않음
- x₂ 만을 고려했을 때, 정상 범위에서 크게 벗어나지 않음
- 하지만, 전체적인 데이터의 분포를 고려하면, 정상 범위에서 벗어남

Sensing Layer: 데이터 정제

- 데이터 정제(Data cleaning/cleansing/scrubbing)
 - 4. Anomaly detection (이상 탐지)
 - 각 데이터 샘플에 다수의 feature 가 포함된 상황에서, feature 들 간의 상관관계까지 동시에 고려할 수 있는 방법은?

Multi-Variate Gaussian Distribution을 모델로 사용
(다음 시간에...)

Sensing Layer: 데이터 정제

■ 데이터 정제(Data cleaning/cleansing/scrubbing)

• 4. Anomaly detection (이상 탐지): 구현

- 사전 작업: 필수 라이브러리 import

```
[ ] # 필수 라이브러리 import
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from scipy.io import loadmat

[ ] # 구글 드라이브 마운트
from google.colab import drive
drive.mount('/content/drive')
```

- 데이터 셋 불러오기

```
data1_path = '/content/drive/MyDrive/Colab Notebooks/AnomalyDetection/data1.mat'
data2_path = '/content/drive/MyDrive/Colab Notebooks/AnomalyDetection/data2.mat'

data1 = loadmat(data1_path)
print('Data set 1: ', data1.keys())

#data2 = loadmat(data2_path)
#print(data2.keys())

Data set 1: dict_keys(['__header__', '__version__', '__globals__', 'X', 'Xval', 'yval'])
```

실습에 필요한 데이터 값

Sensing Layer: 데이터 정제

■ 데이터 정제(Data cleaning/cleansing/scrubbing)

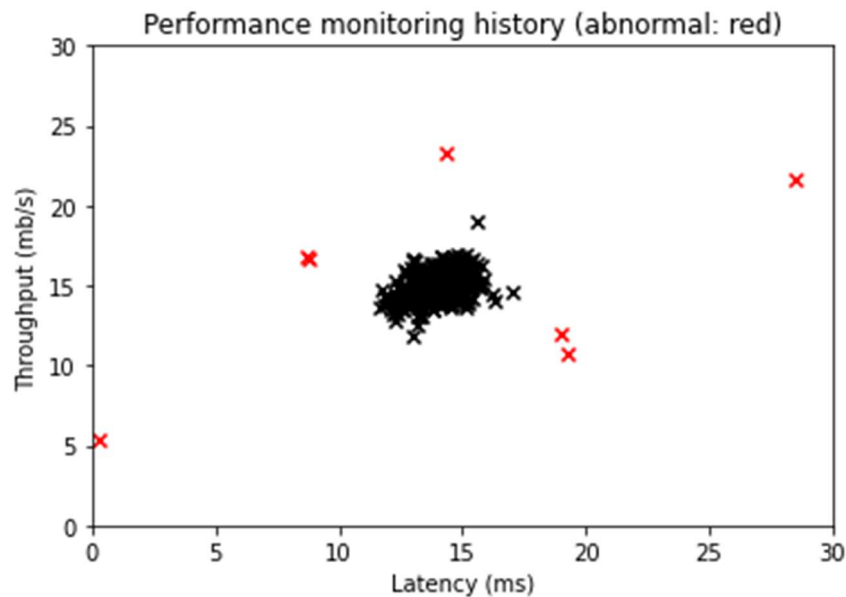
- 4. Anomaly detection (이상 탐지) : 구현
 - 데이터 설명
 - 운영자가 관리하는 시스템의 성능을 측정한 데이터를 모은 데이터 셋
 - Latency (ms) 및 throughput (mb/s) 을 측정한 값

Column	설명
X	<u>새롭게 획득한</u> Latency, Throughput 데이터 셋 // Label이 없음 <ul style="list-style-type: none">• X.shape = (307, 2)• 즉, (Latency, Throughput) 쌍으로 구성된 데이터가 총 307개
Xval yval	Xval: 기존에 획득한 Latency, Throughput 데이터 셋 yval: 기존에 획득한 데이터 각각에 대해 정상/비정상 여부를 표시한 값 (Label) <ul style="list-style-type: none">• yval[i]=1: i번째 데이터는 비정상(anomaly)• yval[i]=0: i번째 데이터는 정상(normal)
공통	X와 Xval 모두 동일한 시스템을 대상으로 측정한 값이기 때문에, 동일한 특성을 가짐

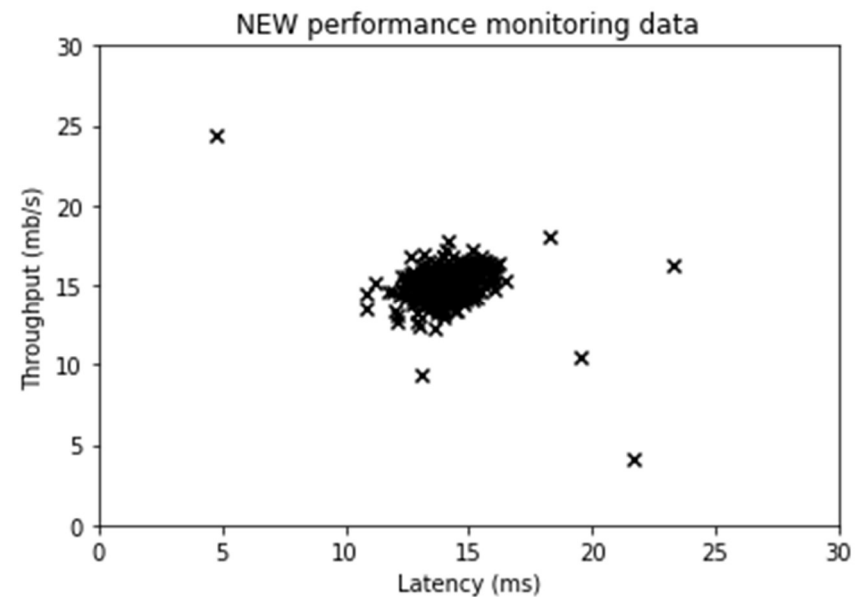
Sensing Layer: 데이터 정제

- 데이터 정제(Data cleaning/cleansing/scrubbing)
 - 4. Anomaly detection (이상 탐지) : 구현
 - 데이터 시각화 하기

정상/비정상 분류가 완료된 기존 데이터셋



신규로 획득한 데이터 셋(정상/비정상 분류 필요함)



이상탐지 기법 구현은 다음
시간에...

