Machine Learning

컴퓨테이셔널 방법론 부트캠프

2020-11-20
오종환

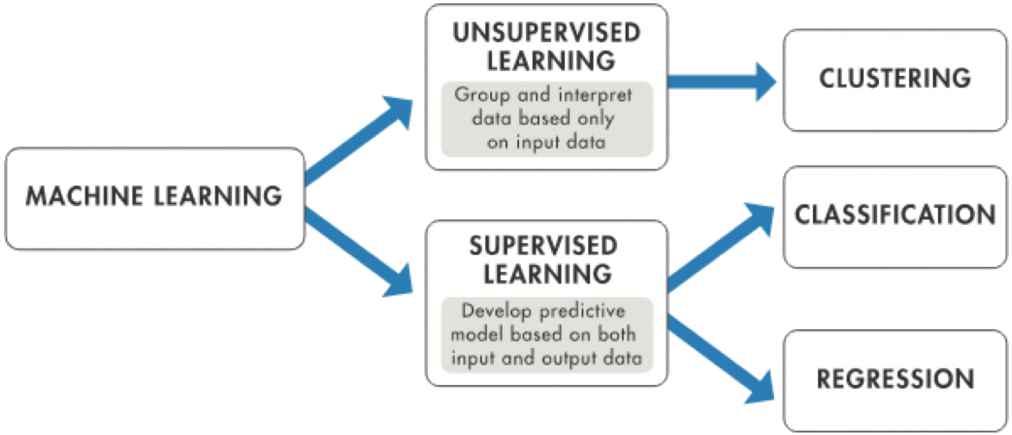
Machine Learning

What is Machine Learning?

- 컴퓨터가 데이터로부터 학습하여 의사 결정을 할 수 있도록 만드는 것
  - 인공지능의 하위 개념으로 분류되기도 함
  - 명확한 룰을 사람이 직접 프로그래밍 하지 않음
  - 예측을 할 수 있는 데이터 모델을 구축하는 수단
  - 예)
    - 이메일이 스팸인지 아닌지를 학습하는 것
    - 위키피디아의 글을 몇가지 카테고리로 clustering

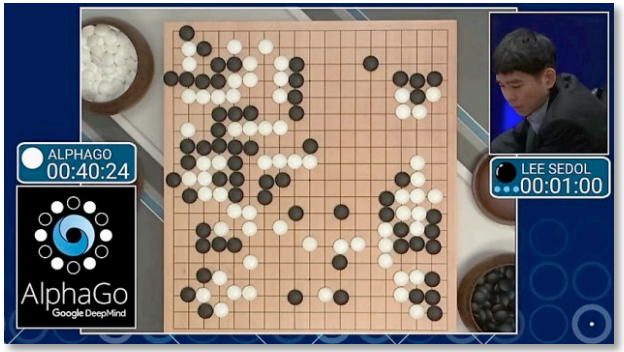
Supervised/Unsupervised Learning

- 지도학습(supervised learning): 입력과 출력값이 모두 주어진 상태에서 학습을 하여 출력 값을 예측하는 것
- 비지도학습(unsupervised learning): 입력값만 존재하는 상황에서 숨겨진 패턴을 찾아내는 것



Reinforcement Learning 강화학습

- 주변 환경과 상호작용할 수 있는 일종의 Software agent
  - 행동을 최적화하는 방법을 학습함
  - 시스템이 우연히 취한 행동에 상점 또는 벌점을 주는 것을 반복하여 특정 목적을 이끌어냄
- 경제학, 유전학, 게임플레이 등에 활용

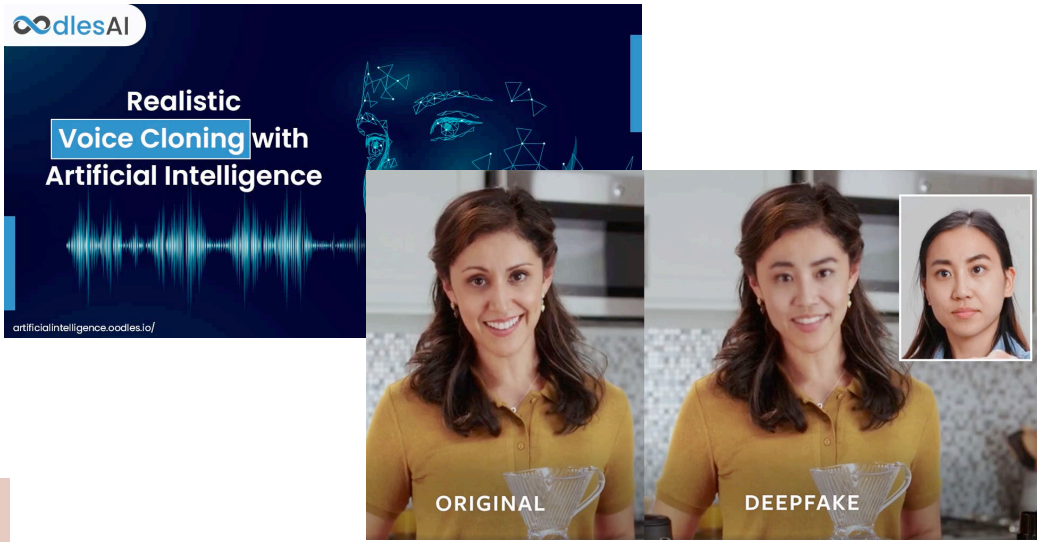


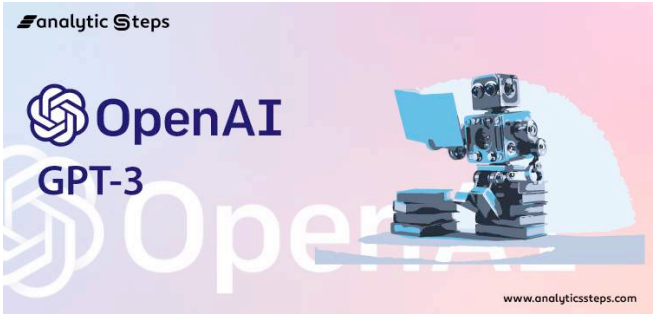
Deep Learning 심층학습

- Machine Learning의 하위 집합
- 여러 비선형 변환 기법의 조합을 통해 높은 수준의 추상화를 시도하는 머신러닝 알고리즘의 집합
    - 사람의 학습 방식을 모방하여 컴퓨터가 수행
- 생명체의 신경망(neural network)를 모방한 인공신경망에 기반하여 설계된 개념
    - 1989년 처음 소개되었을 때, 신경망 학습에 소요되는 시간이 비현실적으로 오래 걸렸음
    - 이후, 더욱 효율적인 모델에 밀려 사람들의 관심에서 멀어졌음
    - 2000년대 이후, 과적합 등 여러 가지 문제점이 해결하는 방법론이 개발. GPU의 활용 등 컴퓨팅 파워가 비약적으로 발전

Deep Learning 심층학습

- 영상/음성 인식, 자연어 처리 등 기존의 방식으로 할 수 없었던 많은 과제에 적용





Python Libraries

Numpy

```
[14] import numpy as np

     a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
     a

     array([[ 1,  2,  3,  4],
            [ 5,  6,  7,  8],
            [ 9, 10, 11, 12]])
```

- 고성능의 수치계산이 가능
- 벡터와 행렬의 연산
- 기본적으로 array를 기본으로 한 연산을 수행
- pandas와 matplotlib의 기반

Pandas

```
[ ]  X = iris.data
     y = iris.target
     df_f = pd.DataFrame(X, columns=iris.feature_names)
     df_f.head()
```

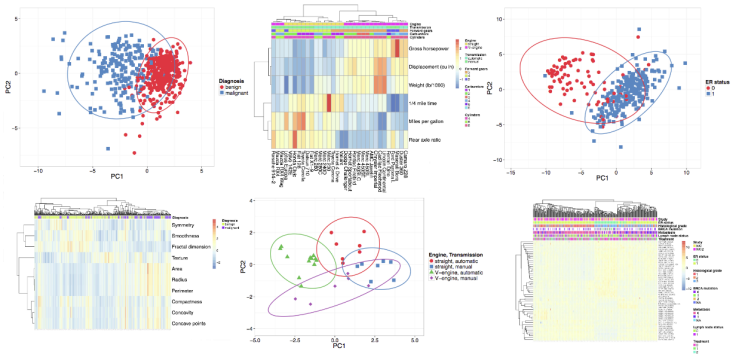
	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

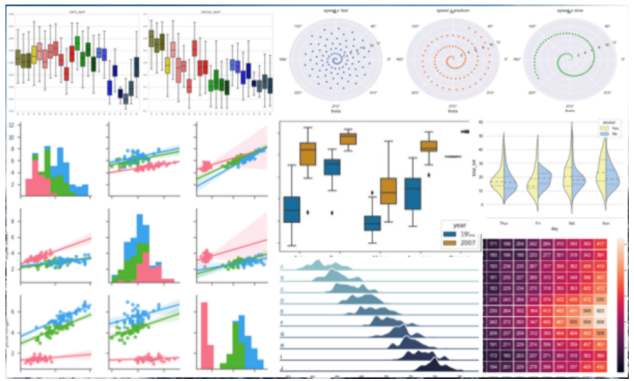
- 테이블 형태의 데이터 구조인 데이터프레임을 활용
- 데이터를 확인하거나, 선택, 변형, 병합 등의 작업을 수행할 수 있음
- Numpy array를 전달 받아 데이터 프레임을 만들거나, matplotlib으로 시각화

Matplotlib / Seaborn

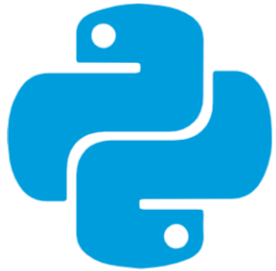
```
import matplotlib.pyplot as plt
import seaborn as sns
```

- 데이터 시각화를 위한 라이브러리
- 데이터 탐색을 위한 EDA 과정 혹은 결과물의 효율적 전달을 위한 visualization





Scikit-Learn



- Machine Learning 을 위한 라이브러리

- pandas dataframe이나 numpy array를 데이터셋으로 활용
- 모든 machine learning 모델들이 python class로 작성되어 있음

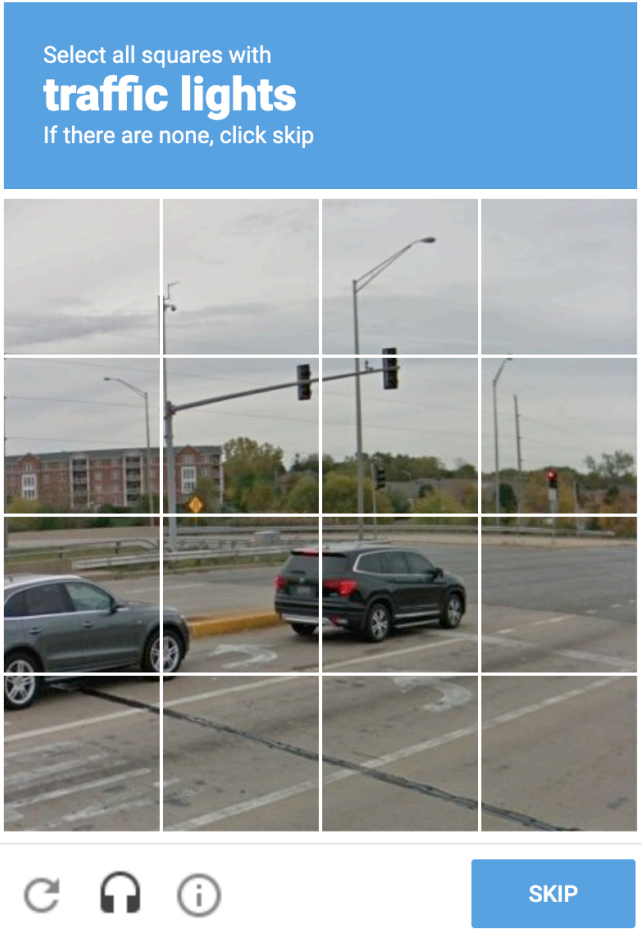
Scikit-Learn

- 일반적인 Scikit-learn API 사용단계
    1. 목적에 맞는 예측 모델을 import 한다.
    2. 모델에 원하는 하이퍼 파라미터를 설정한다.
    3. 데이터를 feature 행렬과 target 벡터로 나눈다.
    4. fit() 를 호출하여, 모델을 구성한다.
    5. 새로운 데이터를 모델에 대입한다.
        - supervised learning: predict()를 이용하여 label을 예측
        - unsupervised learing: transform() 또는 predict()를 이용하여 결과를 확인

Classification

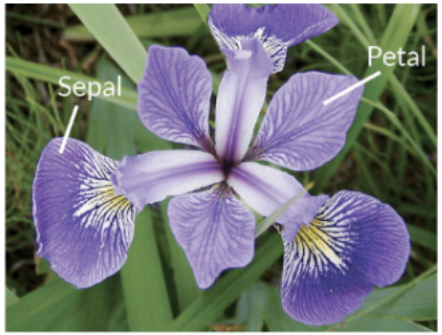
Supervised Learning

- 시간이 많이 들거나 비용이 많이 드는 작업을 대신함
  - 의사의 진단
- 미래에 대한 예측
  - 클릭할 것인가 하지 않을 것인가?
- 라벨이 있는 (labeled) 데이터가 필요
  - 기존의 기록된 데이터를 활용
  - 실험을 통해 모음 (예: 크라우드 소싱)



Exploratory Data Analysis

- EDA (Exploratory Data Analysis, 탐색적 데이터 분석)
  - 데이터의 구조와 특징을 파악
  - 통계적 정보와 시각화를 통해 데이터의 패턴 또는 인사이트를 발견
  - 분석 과정에서 유연하게 가설을 설정할 수 있음
  - 자료를 직관적으로 바라볼 수 있음
  - 이상치(outliers)를 발견
  - 선 데이터 탐색, 후 분석



Iris Versicolor

Iris Setosa

Iris Virginica

Iris Dataset

Classification

- Predictor variables = Independent variables = Features
- Response variable = Dependent variables = Target variable

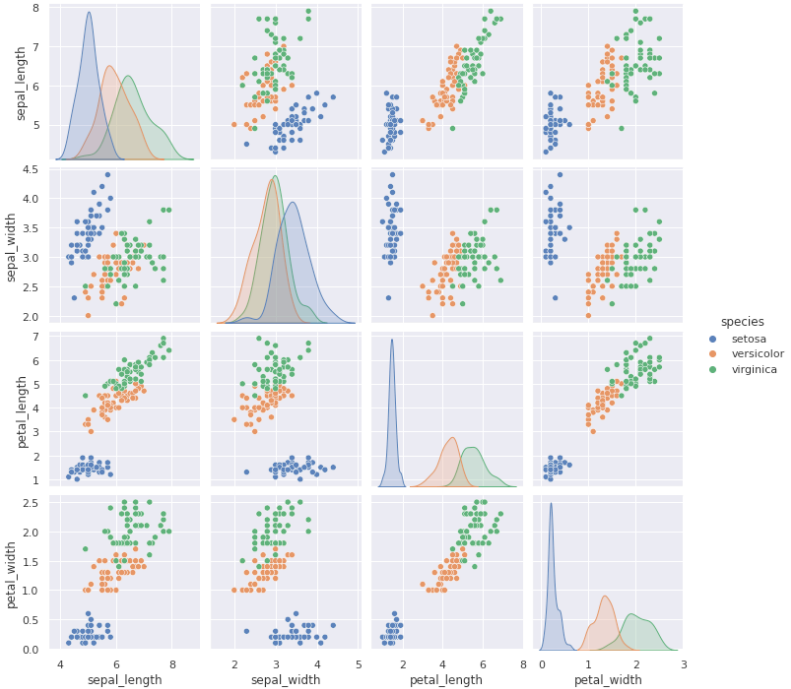
Predictor variables

Target variable

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	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

Exploratory Data Analysis

```
import seaborn as sns
iris = sns.load_dataset('iris')
sns.pairplot(iris, hue='species');
```



EDA

```
[8] type(iris)
```

```
pandas.core.frame.DataFrame
```

```
[12] iris.shape
```

```
(150, 5)
```

```
[9] iris.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

```
[10] iris.describe()
```

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

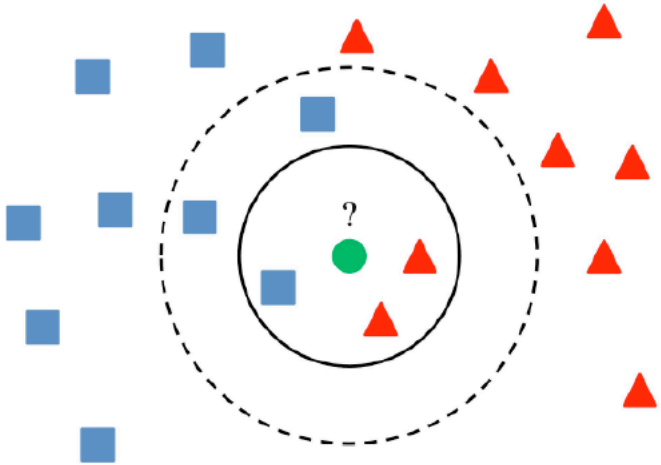
```
[11] iris.info()
```

```
<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
```

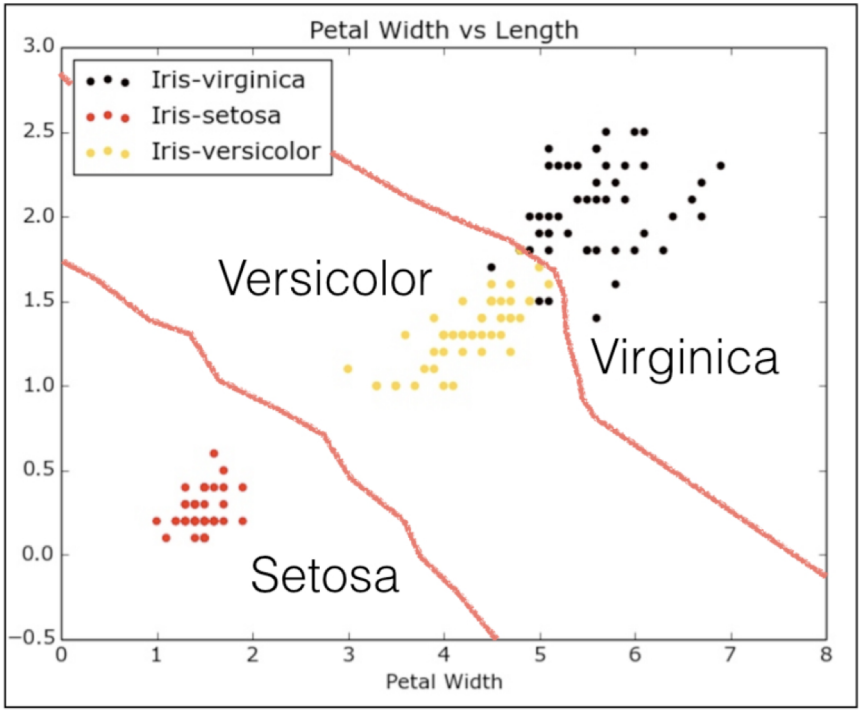
k-Nearest Neighbors

- 새로운 데이터가 주어졌을 때 기존 데이터 가운데 가장 가까운 k개 이웃의 정보로 새로운 데이터를 예측하는 방법
    - 주변에 있는 가까운 labeled data들의 다수결 투표로 분류
    - k의 값과 거리의 측정방법이 결정해야할 요소(hyperparameter)

k–Nearest Neighbors



k–Nearest Neighbors



Iris 데이터셋으로 모델 구성하기

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=6)
knn.fit(iris['data'], iris['target'])
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30,
    metric='minkowski',metric_params=None, n_jobs=1,
    n_neighbors=6, p=2,weights='uniform')
```

```
iris['data'].shape
```

```
(150, 4)
```

```
iris['target'].shape
```

```
(150,)
```

새로운 데이터를 넣어 예측하기

```
X_new = np.array([[5.6, 2.8, 3.9, 1.1],
       [5.7, 2.6, 3.8, 1.3],
       [4.7, 3.2, 1.3, 0.2]])

prediction = knn.predict(X_new)
X_new.shape
```

```
(3, 4)
```

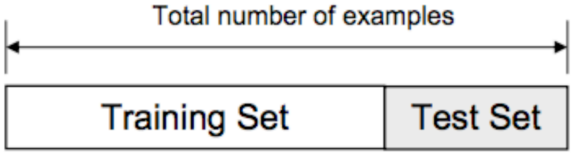
```
print('Prediction: {}'.format(prediction))
```

```
Prediction: [1 1 0]
```

모델의 성능 측정하기

- 분류에서는 정확도(accuracy)가 기본적인 성능 측정 지표
- Accuracy: 테스트를 위해 입력한 데이터 중에 맞게 예측한 것의 비율
    - 성능 측정에는 모델 구성에 사용하지 않은 데이터를 넣어봐야 함
    - 균일한 label을 가진 데이터를 가지고 측정해야한다

Training Set, Test Set



- 전체 데이터를 미리 학습을 위한 데이터(training set)와 모델 성능 측정을 위한 데이터(test set)로 나눔
- test set의 데이터를 입력하여 예측을 해본 뒤, 실제 label과 비교하여, 정확도를 측정함
    - 랜덤하게 추출하되, 나눌 때에도 각 class 의 비율을 고려해야함
- 일반적으로 7:3 정도이나 상황에 따라 다름
    - training set이 많으면, 더 정확한 모델을 만들 수 있으나, test set이 적어지면 측정이 부정확할 수 있음

Training Set, Test Set

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =
    train_test_split(X, y, test_size=0.3,
                     random_state=21, stratify=y)
knn = KNeighborsClassifier(n_neighbors=8)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
print(\"Test set predictions:\\n {}\".format(y_pred))
```

```
Test set predictions:
 [2 1 2 2 1 0 1 0 0 1 0 2 0 2 2 0 0 0 1 0 2 2 2 0 1 1 1 0 0
 1 2 2 0 0 2 2 1 1 2 1 1 0 2 1]
```

```
knn.score(X_test, y_test)
```

```
0.9555555555555556
```

실습: 정치인의 정당 소속 분류하기

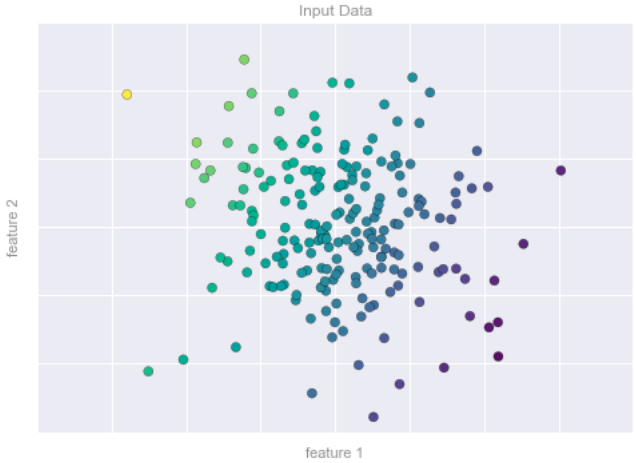
- 정치인들 활동 내역이 담긴 데이터 세트에 kNN을 적용
- 기록에 따라 미국 의원의 정당 소속을 분류

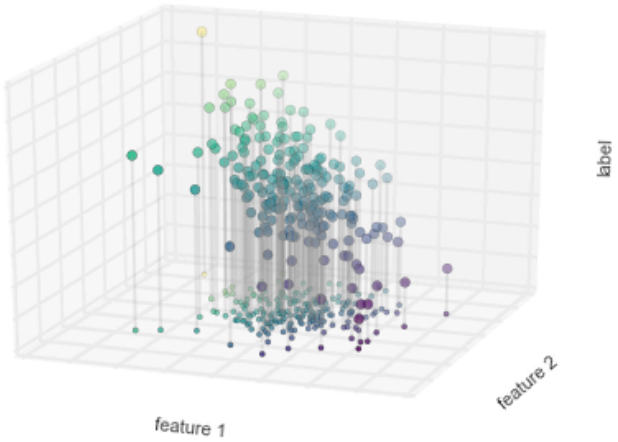
1. Data set 준비
    - Google colab에 접속하여 주어진 데이터셋을 업로드
    - 필요한 라이브러리를 Import
    - EDA를 통한 데이터 탐색
    - 전처리(preprocessing): 분석에 맞게 데이터셋의 구조를 변화하고 결측치를 처리
2. training set, test set 나누기
3. k-NN분류기를 구성하고, training set으로 fit()
4. 모델 성능 측정
    - 새로운 데이터를 정치 활동 데이터를 입력하여, 어느 정당 소속인지 예측해보기
    - Test set 입력하여, 정확도 측정

Regression

Regression 회귀분석

- 분류(Classification) : 불연속적인 값을 예측
- 회귀(Regression) : 연속적인 값을 예측





Boston Housing Data

```
boston = pd.read_csv('boston.csv')
print(boston.head())
```

```
      CRIM    ZN  INDUS  CHAS     NX     RM   AGE     DIS  RAD    TAX  \\
0  0.00632  18.0   2.31     0  0.538  6.575  65.2  4.0900    1  296.0
1  0.02731   0.0   7.07     0  0.469  6.421  78.9  4.9671    2  242.0
2  0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671    2  242.0
3  0.03237   0.0   2.18     0  0.458  6.998  45.8  6.0622    3  222.0
4  0.06905   0.0   2.18     0  0.458  7.147  54.2  6.0622    3  222.0
   PTRATIO       B  LSTAT  MEDV
0     15.3  396.90   4.98  24.0
1     17.8  396.90   9.14  21.6
2     17.8  392.83   4.03  34.7
3     18.7  394.63   2.94  33.4
4     18.7  396.90   5.33  36.2
```

Feature/Target variable 나누기

```
X = boston.drop('MEDV', axis=1).values
y = boston['MEDV'].values
```

```
X_rooms = X[:,5]
type(X_rooms), type(y)
```

```
(numpy.ndarray, numpy.ndarray)
```

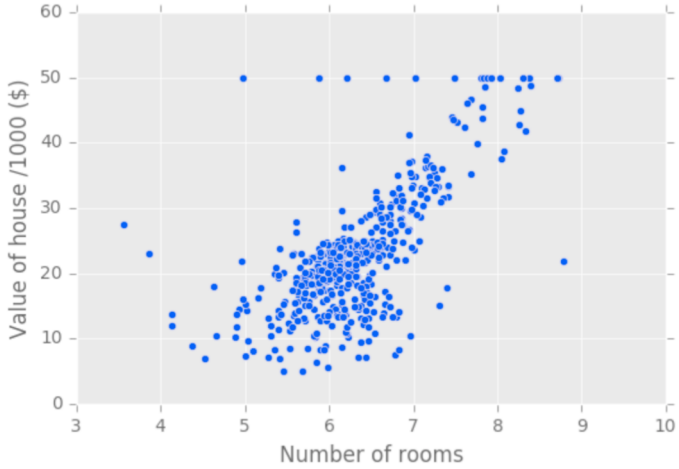
```
y = y.reshape(-1, 1)
X_rooms = X_rooms.reshape(-1, 1)
```

산점도(scatter plot)로 시각화

```
X_rooms = X[:,5]
type(X_rooms), type(y)
```

```
(numpy.ndarray, numpy.ndarray)
```

```
y = y.reshape(-1, 1)
X_rooms = X_rooms.reshape(-1, 1)
```

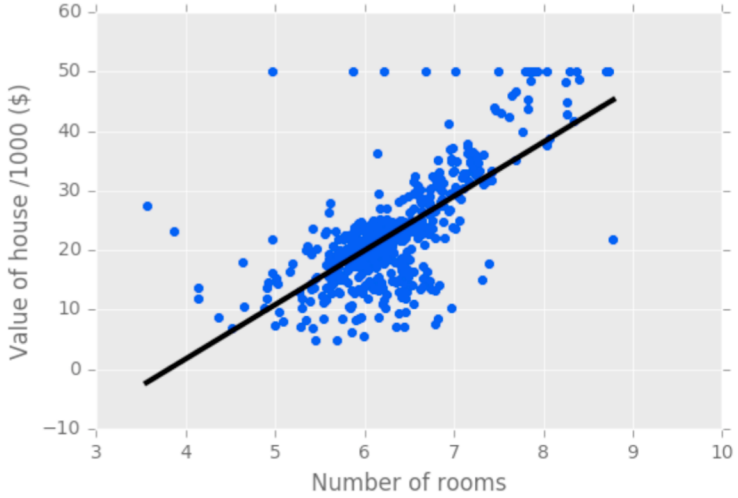


Linear Regression Model

```
import numpy as np
from sklearn.linear_model import LinearRegression

reg = LinearRegression()
reg.fit(X_rooms, y)
prediction_space = np.linspace(min(X_rooms),
                               max(X_rooms)).reshape(-1, 1)
```

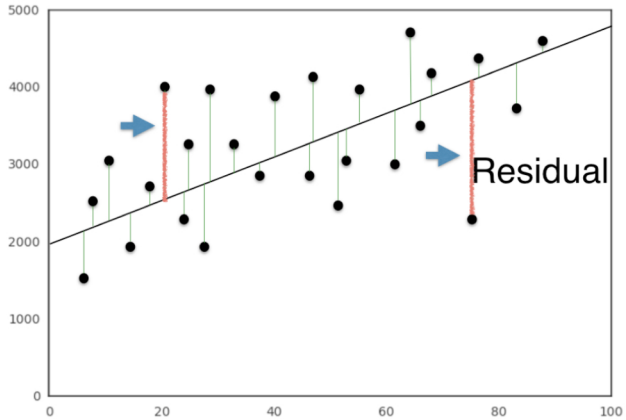
```
plt.scatter(X_rooms, y, color='blue')
plt.plot(prediction_space, reg.predict(prediction_space),
         color='black', linewidth=3)
plt.show()
```



Linear Regression Model

- y = ax + b
  - y : target
  - x : feature
  - a, b : 모델이 결정하는 parameter.
- 모델을 구성하는 것은 오류를 가정 적게 만드는 model parameters를 결정하는 일

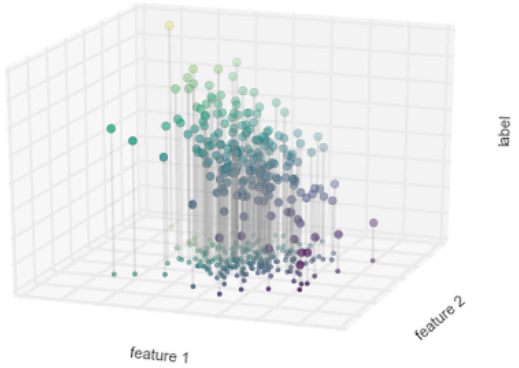
The Loss function



- 손실함수(loss function)을 정의하고, 이 값을 최소화하는 값을 계산

높은 차원의 linear regression

- 차원을 높이면 (=feature의 수를 늘리면),
  - 결정해야할 parameter의 숫자가 늘어남
  - label을 예측할 수 있는 평면을 만들어 내는 것



$$y = a_1 x_1 + a_2 x_2 + b$$

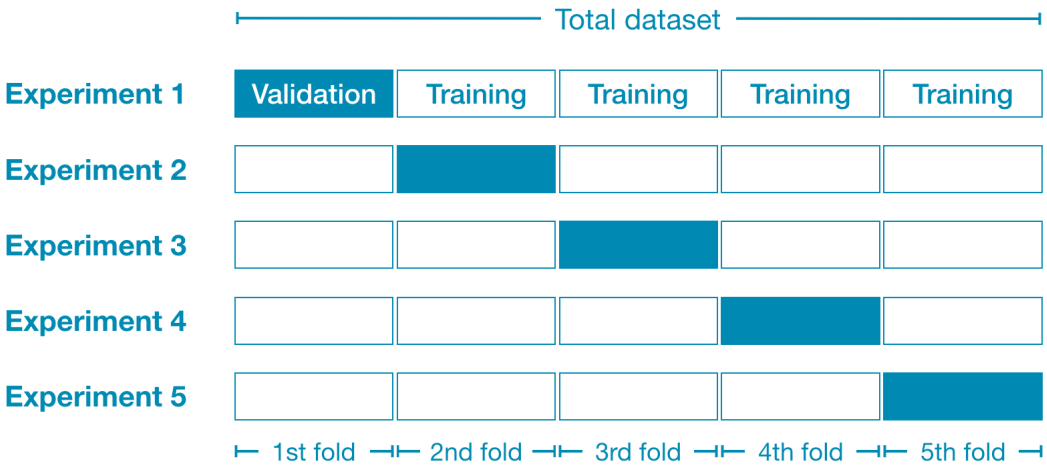
높은 차원의 linear regression

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size = 0.3, random_state=42)
reg_all = LinearRegression()
reg_all.fit(X_train, y_train)
y_pred = reg_all.predict(X_test)
reg_all.score(X_test, y_test)
```

```
0.711226005748495 26
```

Cross-Validation (k-fold CV)



- 우연히 선택된 set의 분할이 결과의 모델 성능을 크게 좌우할 수 있으므로, 이를 보정하기 위한 방법
- 샘플의 크기가 작을 때 유용함
- 결과를 평균 내어 모델 성능을 결정

Cross-Validation (k-fold CV)

```
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
reg = LinearRegression()
cv_results = cross_val_score(reg, X, y, cv=5)
print(cv_results)
```

```
[ 0.63919994  0.71386698  0.58702344  0.07923081 -0.25294154]
```

```
np.mean(cv_results)
```

```
0.353275924395870 58
```

실습: Gapminder

- Gapminder Dataset:여러 국가의 GDP, 출산율, 인구, 기대수명 등이 담긴 데이터
- Feature들을 사용하여 기대수명을 예측하는 모델을 구성

1. EDA
    - 데이터를 탐색해서 변수들간의 상관관계를 알아보기
2. 한 가지 변수만으로 예측하기
3. 여러 변수를 사용하여 예측하기
4. 모델 성능 측정
    - R square 값을 확인하여, 모델이 정확하게 구성되었는지 확인하기
    - Cross Validation으로 모델 성능 확인하기

더 좋은 모델 만들기

Accuracy in Classification

- Label된 Class간의 비율이 지나치게 차이가 나면 accuracy 사용이 맞지 않음
  - 스팸 메일 1%, 정상 메일 99%인 데이터 셋에서, 무조건 정상 메일로 분류한다면?
    - 99%의 accuracy → 쓸모 없는 분류기
    - 다른 척도를 사용해야함
- class를 구성하는 비율이 서로 비슷할 때만 사용!

Confusion Matrix

$$Accuray = \frac{TP+FN}{TP+FP+TN+FN}$$

	Predicted: Spam Email	Predicted: Real Email
Actual: Spam Email	True Positive	False Negative
Actual: Real Email	False Positive	True Negative

$$Recall = \frac{TP}{TP+TN}$$

$$Precision = \frac{TP}{TP+FP}$$

$$F_1 = 2 * \frac{Precision*Recall}{Precision+Recall}$$

- precision이 높으면, “스팸으로 예측된 것들 중에 정상 메일은 별로 없음”
- recall이 높으면, “실제 스팸메일들을 대부분 모델이 예측했음”

```
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
knn = KNeighborsClassifier(n_neighbors=8)
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.4, random_state=42)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
```

```
print(confusion_matrix(y_test, y_pred))
```

```
[[52  7]
 [ 3 112]]
```

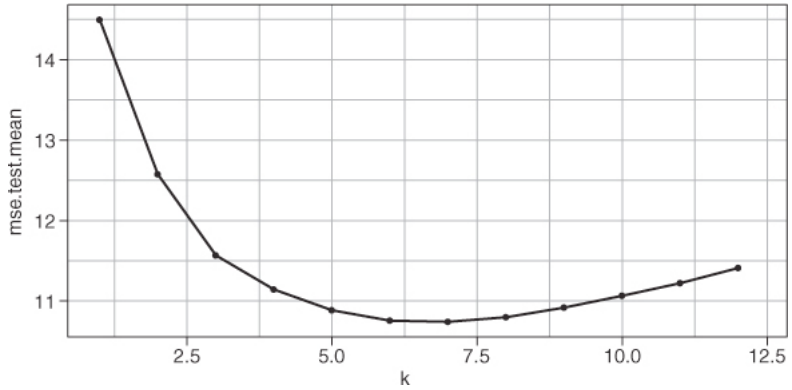
```
print(classification_report(y_test, y_pred))
```

```
             precision    recall  f1-score   support

          0       0.95      0.88      0.91        59
          1       0.94      0.97      0.96       115
avg / total       0.94      0.94      0.94       174
```

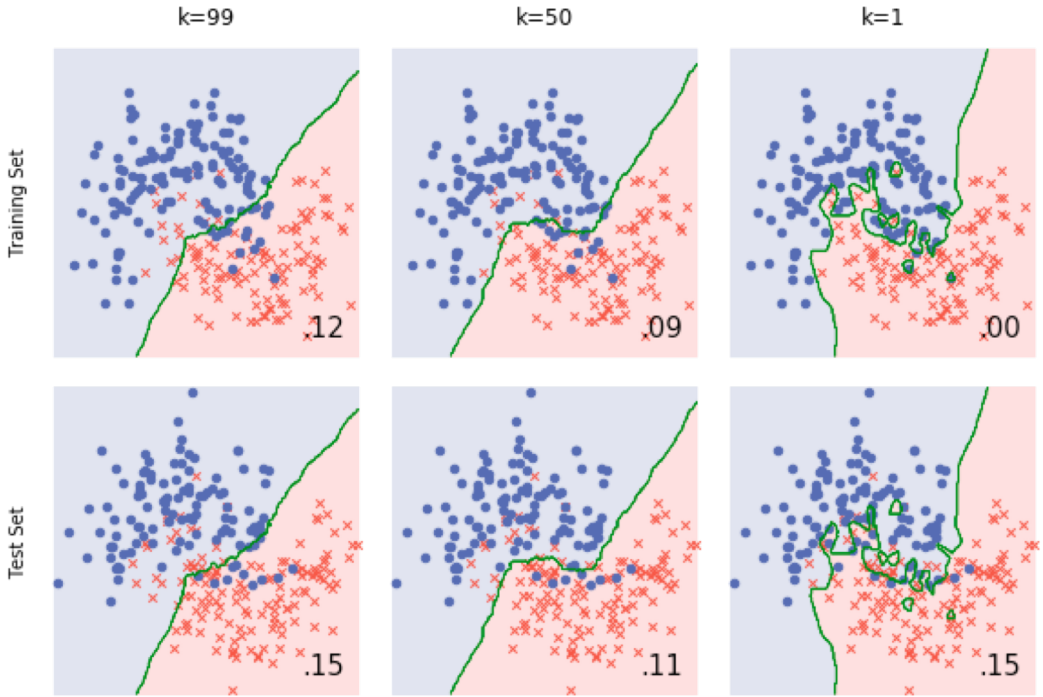
Hyperparameter 조절하기

- Hyperparameter는 모델이 학습할 수 없으며, 직접 선택해주어야 하는 값
    - 예: k-Nearest Neighbors: k 고르기 (n_neighbors)
- Deep Learing 에서는 훨씬 많은 수의 hyperparameter 존재



Under-/Over-Fit

- 적당한 k값 찾기
  - k가 너무 크면, 경계선이 단순한 모델이 되어 정확도가 떨어짐
  - k가 너무 작으면, training set에만 잘 맞아 새로운 데이터 입력시 정확도가 떨어짐



https://cambridgecoding.wordpress.com/2016/03/24/misleading-modelling-overfitting-cross-validation-and-the-bias-variance-trade-off/

Hyperparameter

1. 여러 가지 다른 hyperparameter 값을 대입
2. 각각 모델을 구성해 보기
3. 모델 성능 비교해보기 (cross-validation 포함)
4. 가장 높은 성능의 모델 채택

C	Alpha			
	0.1	0.2	0.3	0.4
0.5	0.701	0.703	0.697	0.696
0.4	0.699	0.702	0.698	0.702
0.3	0.721	0.726	0.713	0.703
0.2	0.706	0.705	0.704	0.701
0.1	0.698	0.692	0.688	0.675

- 반복적으로 hyperparameter의 값을 변화하면서 가장 낮은 오류율을 가진 모델을 선택
- Grid search CV 등의 방법 사용

Preprocessing

Preprocessing

- **“Garbage in, Garbage out”**
- 데이터를 다루는 과정에서 가장 힘들고 오래 걸리는 과정
- 내가 분석하고자 하는 데이터는 예제와 같이 항상 깔끔하게 정리되어 있지 않다!

범주형 변수 (categorical feature)

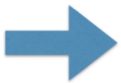
- Scikit-learn 은 범주형 변수를 학습할 수 없음
- 문자를 숫자로 바꾸는 encoding 과정이 필요

	mpg	displ	hp	weight	accel	origin	size
0	18.0	250.0	88	3139	14.5	US	15.0
1	9.0	304.0	193	4732	18.5	US	20.0
2	36.1	91.0	60	1800	16.4	Asia	10.0
3	18.5	250.0	98	3525	19.0	US	15.0
4	34.3	97.0	78	2188	15.8	Europe	10.0

범주형 변수 (categorical feature)

- 각각의 category를 feature로 정의 하고 0 또는 1로 코딩 (dummy variable)
- pandas 의 get_mummies() 이용

Origin
US
Europe
Asia



origin_Asia	origin_US
0	1
0	0
1	0

결측치(missing value) 처리

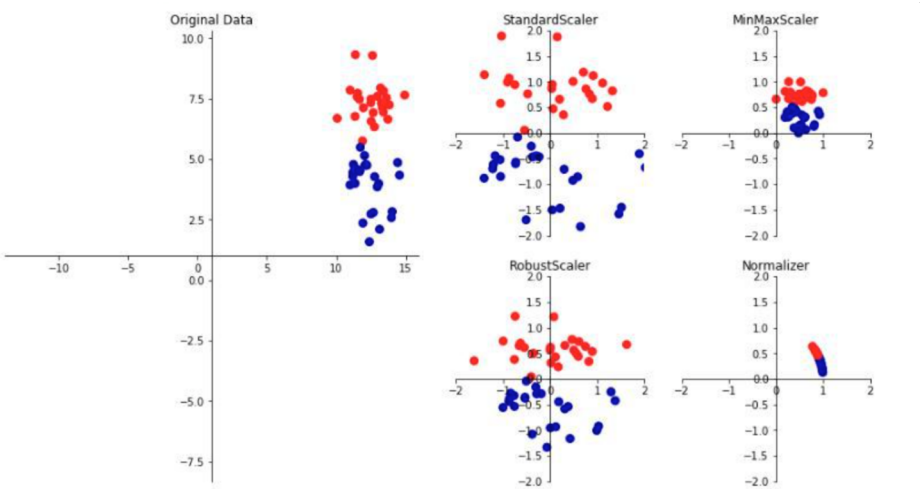
- 어떤 이유에서건 측정되지 않아, 데이터가 비어있는 경우가 있음
- 데이터 원본에는 어떻게 표기되어 있는지 직접 확인해야함
    - NaN 으로 표기되거나, 다르게 표기되어 있을 경우도 있음
- 모델의 종류에 따라, 제대로 학습할 수 없는 경우가 많아 처리가 필요함
- 결측치의 패턴, 결측치의 양을 파악하는 것에서 시작

결측치(missing value) 처리

- 제거 : 결측치가 있는 데이터를 버림. 결측된 데이터가 사라져 편향이 생길 수 있으므로 매우 주의해야 함
- 다른 값을 로 치환: 데이터의 특성을 살펴서 평균값, 최빈값 등 합리적인 값을 선택해 채움
- 모델 기반 처리: 결측치 예측 모델을 구성하여, 채워나가는 방식
    - Scikit-Learn의 pipleline 이용하기

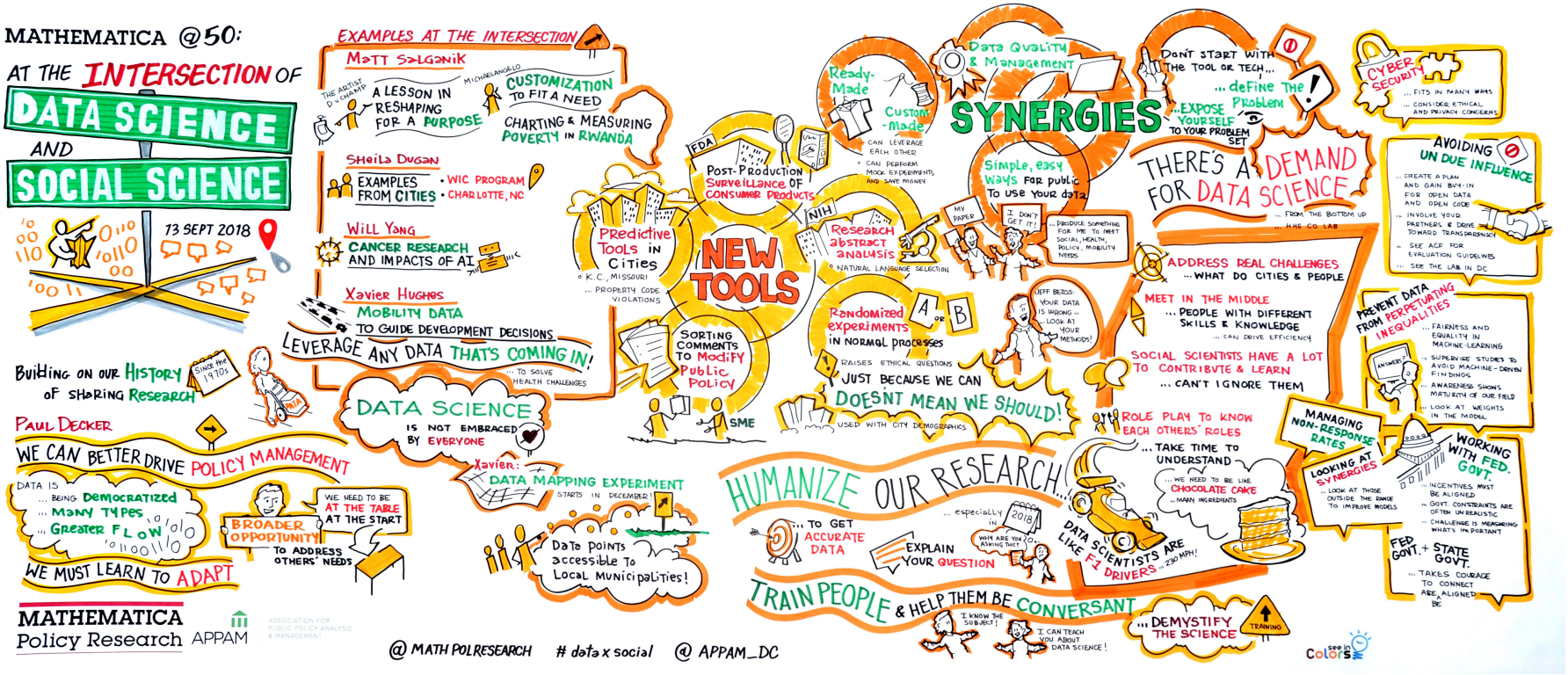
Normalizing

- Feature 간의 스케일이 달라 무시되는 경우도 있음
- 값의 범위를 비슷한 스케일의 수준으로 변환

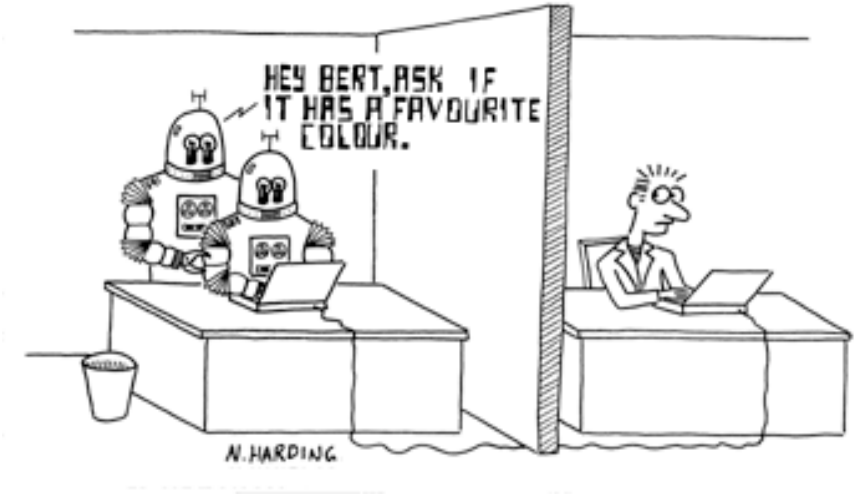


Normalizing

- 정규분포를 따르는 경우 (EDA로 확인)
  - 모든 feature가 평균 1, 표준편차 1가 되도록 만듦
  - 경우에 따라 Log를 취해 정규분포를 따른 경우도 있음 (Feature engineering)
- 정규분포는 아니나, 고르게 분포하는 경우
  - 최솟값을 0으로, 최댓값을 1로 매핑함
- 데이터의 분포나 특성에 따라 다양한 방법을 사용



https://www.mathematica.org/-/media/internet/data-analytics-graphics-and-photos/data-analytics-graphics/mathematica-at-50-data-science-and-social-science-graphic-recording-13-sept-2018-4464x1912-72-dpi.jpg



이 자료는 Datacamp의 "Supervised Learning with scikit-learn"과 Jake VanderPlas 의 "Python Data Science Handbook"의 데이터와 구성을 참조하여 제작하였습니다.