ㅇㅇ마이스터 고등학교

# 파이썬 활용 AI 프로그래밍

『 3-2 』

2024. 05. 20. - 2024. 05. 24

Prepared by DaeKyeong Kim

Ph.D.

KOREATECH 한국기술교육대학교

# 목차

『4과목』 머신러닝

# 지도_수치 예측

# Syallabus

## 학습목표

- 이 워크샵에서는 SVM(Support Vector Machine), 인공신경망
  확인할 수 있다

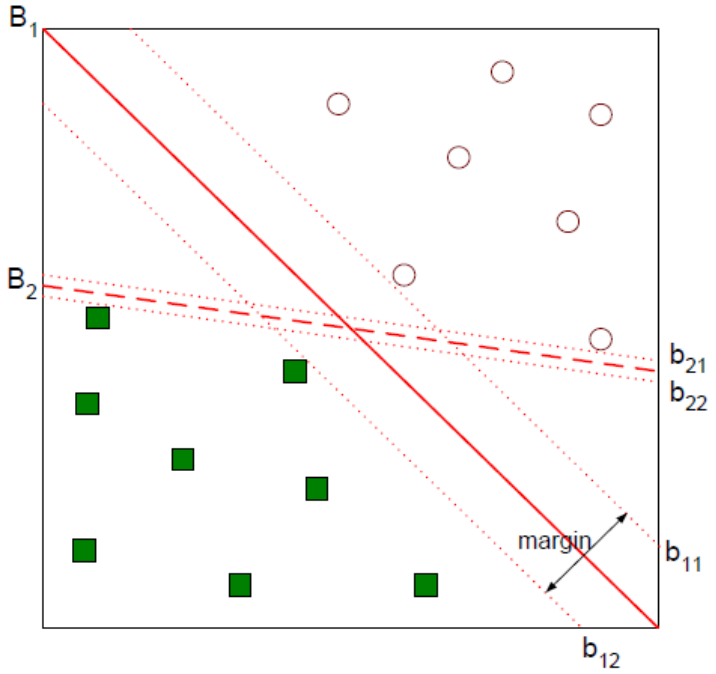
## 눈높이 체크

- SVM(Support Vector Machine), 인공신경망 을 알고 계시나요?

# 1. SVM(Support Vector Machine)

## 개념

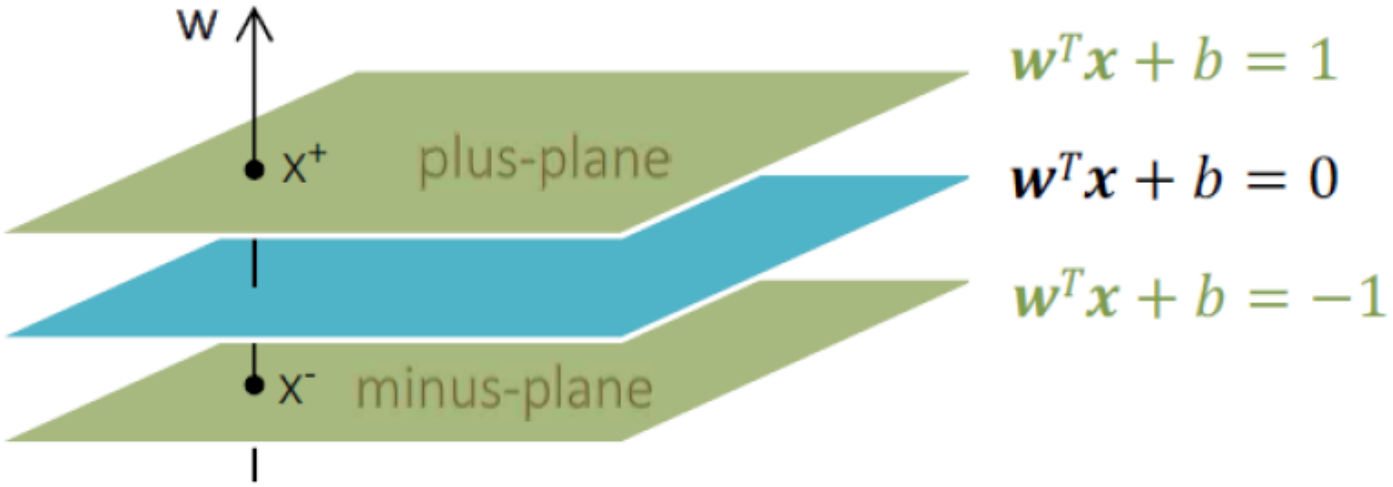
- 서포트 벡터(혹은 지지 벡터)머신은 서로 다른 분류에 속한 데이터 간에 간격(마진)이 최대가 되는 선(또는 초평면)을 찾아서 이를 기준으로 데이터를 분류하는 모델이다. 아래 그림에서 직선 B1과 B2 모두 두 클래스를 무난하게 분류하고 있음을 확인할 수 있다.



# 1. SVM(Support Vector Machine)

## 개념

- 좀 더 나은 분류경계면을 꼽으라면 B1이다. 위 그림에서 b12을 minus-plane, b11을 plus-plane, 이 둘 사이의 거리를 마진(margin)이라고 하며, SVM은 이 마진을 최대화하는 분류 경계면을 찾는 기법이다. 이를 도식적으로 나타내면 아래와 같다.




- 벡터 w는 이 경계면과 수직인 법선벡터 벡터 w는 이 경계면과 수직인 법선벡터

# 1. SVM(Support Vector Machine)

## 개념

- margin

$$
\begin{aligned}
Margin &= distance(x^+, x^-) \\
&= \|x^+ - x^-\|_2 \\
&= \|x^- + \lambda w - x^-\|_2 \\
&= \|\lambda w\|_2 \\
&= \lambda\sqrt{w^T w} \\
&= \frac{2}{w^T w}\sqrt{w^T w} \\
&= \frac{2}{\sqrt{w^T w}} \\
&= \frac{2}{\|w\|_2}
\end{aligned}
$$

벡터 $x^+$와 $x^-$ 사이의 관계

$x^+ = x^- + \lambda w$

$$
\begin{aligned}
w^T x^+ + b &= 1 \\
w^T(x^- + \lambda w) + b &= 1 \\
w^T x^- + b + \lambda w^T w &= 1 \\
-1 + \lambda w^T w &= 1
\end{aligned}
$$

$$
\lambda = \frac{2}{w^T w}
$$

# 1. SVM(Support Vector Machine)

## 개념

- SVM의 목적은 마진을 최대화하는 경계면을 찾는 것
- 계산상 편의를 위해 마진 절반을 제곱한 것에 역수를 취한 뒤 그 절반을 최소화 곧, 목적함수를 최소화

$$max \frac{2}{\|w\|_2} \rightarrow \min \frac{1}{2}\|w\|_2^2$$

- 제약식

$$y_i(w^T x_i + b) \geq 1$$

# 1. SVM(Support Vector Machine)

## 개념

- 라그랑주 승수법(Lagrange multiplier method)
- 라그랑주 승수법 (Lagrange multiplier method)은 프랑스의 수학자 조세프루이 라그랑주 (Joseph-Louis Lagrange)가 제약 조건이 있는 최적화 문제를 풀기 위해 고안한 방법이다. 라그랑주 승수법은 어떠한 문제의 최적점을 찾는 것이 아니라, 최적점이 되기 위한 조건을 찾는 방법이다. 즉, 최적해의 필요조건을 찾는 방법이다.

- 목적식과 제약식을 라그랑지안 문제로 식을 다시 쓰면

$$\min L_p(w, b, \alpha_i) = \frac{1}{2}\|w\|_2^2 - \sum_{i=1}^{n} \alpha_i (y_i (w^T x_i + b) - 1)$$

- Lp 의 제약

$$\alpha_i \geq 0, i = 1, \ldots, n$$

# 1. SVM(Support Vector Machine)

## 개념

- 라그랑주 승수 값이 0 즉, $a_i=0$ 이면 해당 데이터는 예측 모형, 즉 w 계산에 아무런 기여를 하지 않으므로 위 식을 실제로는 다음과 같다.

$$f(x) = a^+ x^T x^+ - a^- x^T x^- - w_0$$

- 여기에서 $x^Tx^+$ 는 x와 $x^+$사이의 (코사인)유사도, $x^Tx^-$ 는 x와 $x^-$사이의 (코사인)유사도이므로 결국 두 서포트 벡터와의 유사도를 측정해서 값이 큰 쪽으로 판별하게 된다.

# 1. SVM(Support Vector Machine)

## 서포트 벡터 머신 기법의 장/단점

| 장 점 | 단 점 |
| --- | --- |
| ✓ 범주분류나 수치예측 문제에 모두 활용 가능하다.<br>✓ 노이즈 데이터에 영향을 크게 받지 않고, 과적합화가 잘 되지 않는다.<br>✓ 일반적으로 분류 문제에서 다른 알고리즘 보다 분류 성능이 높은 것으로 알려져 있으며, 특히 분류 경계가 복잡한 비선형 문제일 경우 타 기법대비 성능이 좋은 것으로 알려져 있 | ✓ 최적 분류를 위해 커널함수 및 매개변수 등에 대한 반복적인 조합 테스트가 필요하다.<br>✓ 입력 데이터의 양이나 변수가 많은 경우 훈련에 오랜 시간이 소요된다.<br>✓ 배경이 되는 이론 및 알고리즘 구현 시 타 기법에 비해 상대적으로 난해한 면이 있다.<br>✓ 결과 해석이나 이유 설명 등이 쉽지 않다. |

# 1. SVM(Support Vector Machine)

## Scikit-Learn의 서포트 벡터 머신

- Scikit-Learn의 svm 서브패키지는 서포트 벡터 머신 모형인 SVC (Support Vector Classifier) 클래스를 제공한다.
- 사이킷런 SVM 주요 파라미터

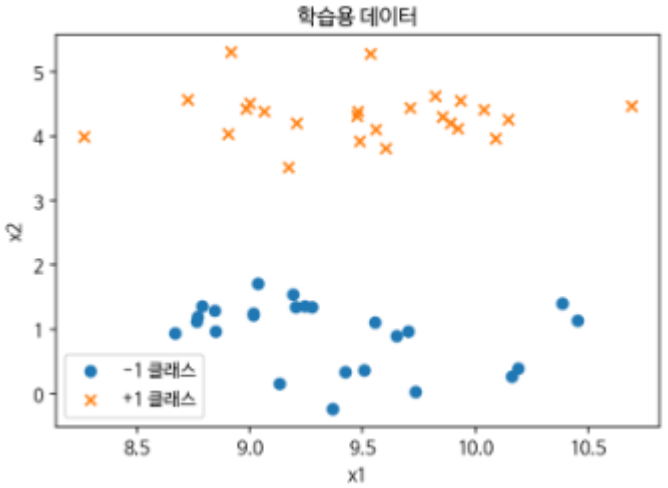
| 파라미터 | default | 설명 |
|---|---|---|
| C | 1.0 | 오류를 얼마나 허용할 것인지 (규제항)<br>클수록 하드마진, 작을수록 소프트마진에 가까움 |
| kernel | 'rbf' (가우시안 커널) | 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' |
| degree | 3 | 다항식 커널의 차수 결정 |
| gamma | 'scale' | 결정경계를 얼마나 유연하게 그릴지 결정<br>클수록 오버피팅 발생 가능성 높아짐 |
| coef0 | 0.0 | 다항식 커널에 있는 상수항 r |

# 1. SVM(Support Vector Machine)

## Scikit-Learn의 서포트 벡터 머신

- Scikit-Learn의 svm 서브패키지는 서포트 벡터 머신 모형인 SVC (Support Vector Classifier) 클래스를 제공한다.

```
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
X, y = make_blobs(n_samples=50, centers=2, cluster_std=0.5, random_state=4)
y = 2 * y - 1

plt.scatter(X[y == -1, 0], X[y == -1, 1], marker='o', label="-1 클래스")
plt.scatter(X[y == +1, 0], X[y == +1, 1], marker='x', label="+1 클래스")
plt.xlabel("x1")
plt.ylabel("x2")
plt.legend()
plt.title("학습용 데이터")
plt.show()
```



# 1. SVM(Support Vector Machine)

## Scikit-Learn의 서포트 벡터 머신

- VC를 불러올 때 kernel='linear'라고 지정
- 레이블은 빨간 걸 1, 파란걸 0으로 보면 된다.그리고 .fit() 안에 학습 데이터와 레이블을 넣는다

```
from sklearn.svm import SVC
classifier = SVC(kernel = 'linear')
training_points = [[1, 2], [1, 5], [2, 2], [7, 5], [9, 4], [8, 2]]
labels = [1, 1, 1, 0, 0, 0]
classifier.fit(training_points, labels)

>>

SVC
SVC(kernel='linear')
```

# 1. SVM(Support Vector Machine)

## Scikit-Learn의 서포트 벡터 머신

- .predict() 메서드를 통해 분류를 해볼 수 있다. 예를 들어 [3, 2] 라는 데이터를 넣어 예측
- [3, 2] 좌표를 찍어봐도 알 수 있듯 빨간 점, 1로 분류

```
print(classifier.predict([[3, 2]]))

>>

[1]
```

- 서포트 벡터, 결정 경계를 정의하는 서포트 벡터를 확인

```
print(classifier.support_vectors_)
>>
[[7. 5.]
 [8. 2.]
 [2. 2.]]
```

# 1. SVM(Support Vector Machine)

## Scikit-Learn의 서포트 벡터 머신

- 서포트 벡터, 결정 경계를 정의하는 서포트 벡터를 평가 점수 확인

```
print("학습 데이터 점수: {}".format(classifier.score(training_points , labels )))
```

```
>>
```

```
학습 데이터 점수: 1.0
```

# 1. SVM(Support Vector Machine)

## 붓꽃 문제에의 응용

```
import numpy as np
import matplotlib as mp
import matplotlib.pyplot as plt


from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sk□learn.preprocessing import StandardScaler

iris = load_iris()
X = iris.data[:, [2, 3]]
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)
X_combined_std = np.vstack((X_train_std, X_test_std))
y_combined = np.hstack((y_train, y_test))
>>
```

# 1. SVM(Support Vector Machine)

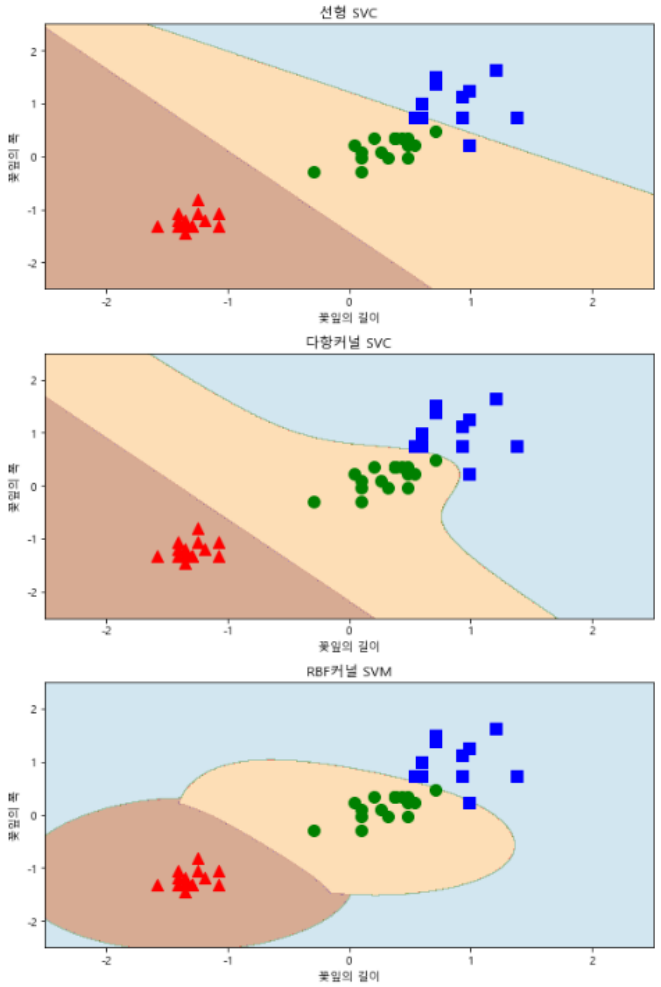
## 붓꽃 문제에의 응용

```
def plot_iris(X, y, model, title, xmin=-2.5, xmax=2.5, ymin=-2.5, ymax=2.5):
    XX, YY = np.meshgrid(np.arange(xmin, xmax, (xmax-xmin)/1000),
                         np.arange(ymin, ymax, (ymax-ymin)/1000))
    ZZ = np.reshape(model.predict(np.array([XX.ravel(), YY.ravel()]).T), XX.shape)
    plt.contourf(XX, YY, ZZ, cmap=mp.cm.Paired_r, alpha=0.5)
    plt.scatter(X[y == 0, 0], X[y == 0, 1], c='r', marker='^', label='0', s=100)
    plt.scatter(X[y == 1, 0], X[y == 1, 1], c='g', marker='o', label='1', s=100)
    plt.scatter(X[y == 2, 0], X[y == 2, 1], c='b', marker='s', label='2', s=100)
    plt.xlim(xmin, xmax)
    plt.ylim(ymin, ymax)
    plt.xlabel("꽃잎의 길이")
    plt.ylabel("꽃잎의 폭")
    plt.title(title)


model1 = SVC(kernel='linear').fit(X_test_std, y_test)
model2 = SVC(kernel='poly', random_state=0,
             gamma=10, C=1.0).fit(X_test_std, y_test)
```

# 1. SVM(Support Vector Machine)

## 붓꽃 문제에의 응용

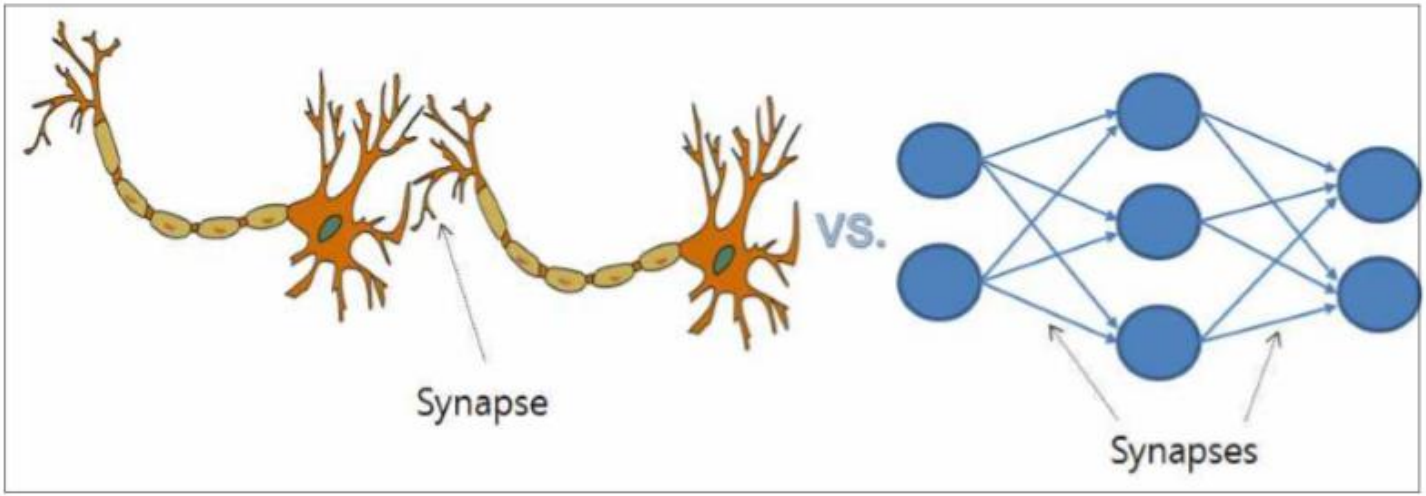
```
model3 = SVC(kernel='rbf', random_state=0, gamma=1,
             C=1.0).fit(X_test_std, y_test)

plt.figure(figsize=(8, 12))
plt.subplot(311)
plot_iris(X_test_std, y_test, model1, "선형 SVC")
plt.subplot(312)
plot_iris(X_test_std, y_test, model2, "다항커널 SVC")
plt.subplot(313)
plot_iris(X_test_std, y_test, model3, "RBF커널 SVM")
plt.tight_layout()
plt.show()
```



# 2. 인공 신경망 분석

## 개념

- 인공 신경망 분석 모델은 생물체의 뇌가 감각 입력 자극에 어떻게 반응하는지에 대한 이해로부터 얻은 힌트를 바탕으로 생물체의 신경망을 모사하여, 입력 신호와 출력 신호 간의 관계를 모델화하는 기법이다. 뇌가 뉴런이라는 세포들의 방대한 연결을 통해 신호를 처리하듯, 인공 신경망은 이를 모사한 인공 뉴런(노드)의 네트워크를 구성하여 모델화한다. 아래의 그림은 생물체의 신경망과 인공 신경망 구조를 비교한 그림이다.

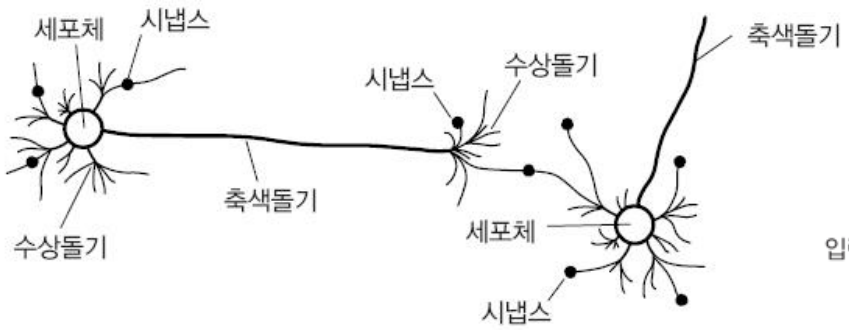


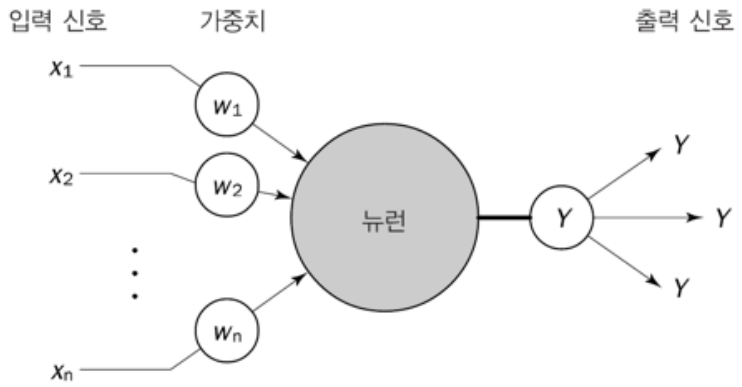
# 2. 인공 신경망 분석

## 개념

- 인간 뇌를 기반으로 한 추론 모델.
- 인간 뇌의 추론 모델 - 뉴런(neuron)
- 인간의 뇌 모델링

| 생물학적인 신경망 | 인공 신경망 |
|---|---|
| 세포체 | 뉴런 |
| 수상돌기 | 입력 |
| 축색돌기 | 출력 |
| 시냅스 | 가중치 |



# 2. 인공 신경망 분석

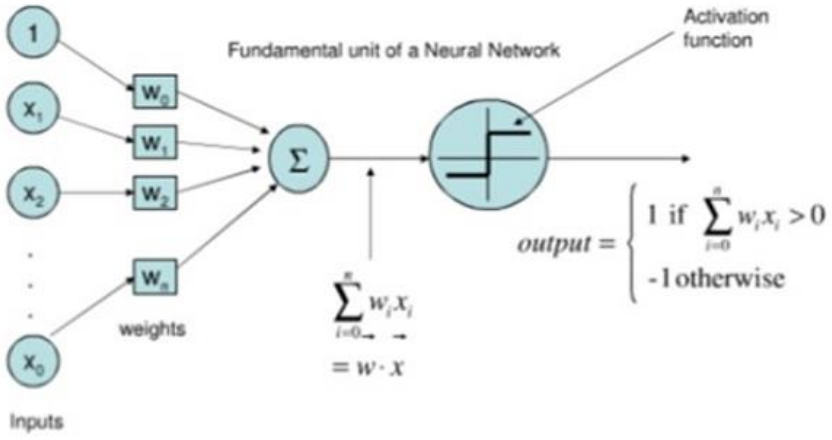
## 개념

- 뉴런의 출력 결정

• 렌 맥클록(Warren McCulloch)과 월터 피츠( Walter Pitts )가 제안(1943년).

– <A logical calculus of ideas immanent in nervous activity> 각 신경세포 (neuron) 의 기능은 매우 단순하나, 이들이 상호 연결됨으로써 복잡한 계산을 수행하는 신경 시스템의 기초를 마련한 이 논문에서, 현대 컴퓨터의 기반을 이루는 모든 Boolean 논리 표현은 2 진 출력을 갖는 맥클로치 - 피츠 신경세포로 구현 가능함을 보여 주었다.

– 뉴런은 전이 함수, 즉 활성화 함수(activation function)를 사용




$$Y = sign\left[\sum_{i=1}^{n} x_i w_i - \theta\right]$$

$$Y = \begin{cases} +1 & \text{if } X \geq \theta \\ -1 & \text{if } X < \theta \end{cases}$$

# 2. 인공 신경망 분석

## 인공 신경망 분석기법의 장/단점

| 장 점 | 단 점 |
| --- | --- |
| ✓ 선형적으로 분류할 수 없는 복잡한 비선형 문제에 탁월한 성능을 보인다.<br>✓ 분류 및 수치예측 문제에 모두 적용 가능하다.<br>✓ 통계적 기본 가정이 적고, 유연한 모델을 만든다.<br>✓ 데이터 사이즈가 작거나, 불완전 데이터, 노이즈 데이터가 많은 경우에도 다른 모델에 비하여 예측 성능이 우수한 경우가 많다. | ✓ 모델결과 해석이 어려워서 은닉층의 노드들이 무엇 을 표현하는지, 왜 그러한 결과값이 나왔는지 설명이 필요한 모델링에는 적합하지 않다. (블랙박스 모형).<br>✓ 나이브 베이즈나 로지스틱 회귀 모형 같은 보다 단순한 분류 알고리즘에 비하여 컴퓨팅 연산에 많은 자원이 필요하다.<br>✓ 과적합화나 과소적합이 발생하기 쉽다.<br>✓ 전체적 관점에서의 최적해가 아닌 지역 최적해가 선택될 수 있다. |

# 2. 인공 신경망 분석

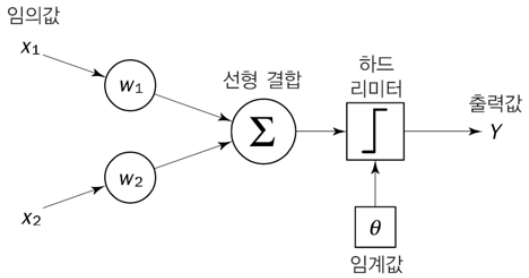
## 단일 계층 퍼셉트론 : SLP: Single Layer Perceptron (로젠블렛 퍼셉트론)

- 학습능력을 갖는 패턴분류장치. 1957년 프랭크 로젠블랫에 의해 제안
- <The perceptron : a probabilistic model for information storage and organization in the brain> 이라는 논문과 1962년에 발표된 《Principles of Neurodynamics》라는 책에서, 신경세포와 유사한 단순 계산기능을 갖는 요소로 구성된 입력층과 출력층을 갖는 perceptron 이라는 신경 시스템의 모델을 제시하고, 입력과 출력 사이의 synapse 를 출력층의 제곱오차가 최소가 되는 방법으로 학습시킬 수 있음을 보여 주었다.

# 2. 인공 신경망 분석

## 단일 계층 퍼셉트론 : SLP: Single Layer Perceptron (로젠블렛 퍼셉트론)

- 입력 노드가 두 개인 단층 퍼셉트론




- 로젠블랫퍼셉트론의 동작 원리는 맥클록과피츠의 뉴런 모델에 기반.
- 퍼셉트론은 선형 결합기와 하드 리미터로 구성.
- 입력의 가중합을 하드 리미터에 입력하고, 입력이 양이면 '+1', 음이면 '-1'을 출력.
- 기본적인 퍼셉트론의 경우, 초평면(hyperplane)으로 n차원 공간을 두 개의 결정 영역으로 나뉜다.
- 초평면을 선형 분리 함수로 정의한다.
- 선형 분리 함수 : 식 $\sum_{i=1}^{n} x_i w_i - \theta = 0$

- 임계값θ는 결정 경계를 옮기는 데 쓰인다.
- 분할 식
  - 입력이 두 개인 퍼셉트론$x_1 w_1 + x_2 w_2 - \theta = 0$
  - 입력이 세 개인 퍼셉트론$x_1 w_1 + x_2 w_2 + x_3 w_3 - \theta = 0$

# 2. 인공 신경망 분석

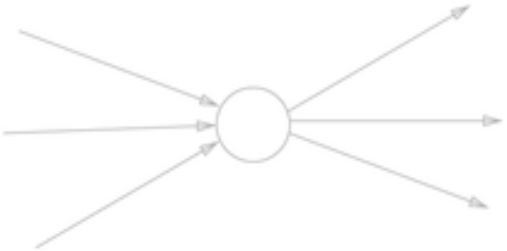
## 활성화 함수의 등장

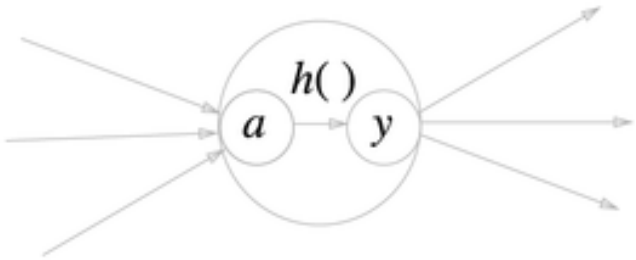
- 활성화 함수 h(x): 입력 신호의 총합을 출력 신호로 변환하는 함수. 활성화를 일으키는 지 정하는 역할.

$$y = h(b + w_1x_1 + w_2x_2)$$

$$h(x) = 0 (x <= 0)$$
$$h(x) = 1 (x > 0)$$

  - 가중치가 달린 입력 신호와 편향의 총합을 계산
    - y=h(a)
  - a를 h(a) 함수에 넣어 y를 출력하는 흐름
  - 단순 퍼셉트론은 단층 네트워크에서 계단 함수를 활성화 함수로 사용한 모델을 가리킴.
  - 다층 퍼셉트론은 신경망(여러 층으로 구성되고 시그모이드 함수 등의 매끈한 활성화 함수를 사용하는 네트워크)을 가리킴
  - 왼쪽은 일반적인 뉴런, 오른쪽은 활성화 처리 과정을 명시한 뉴런

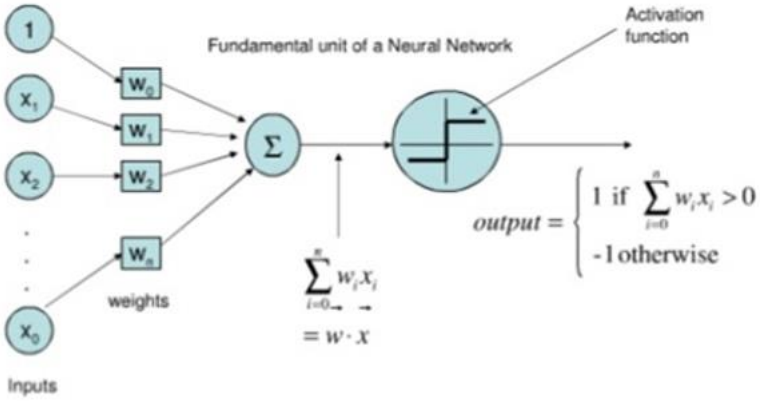




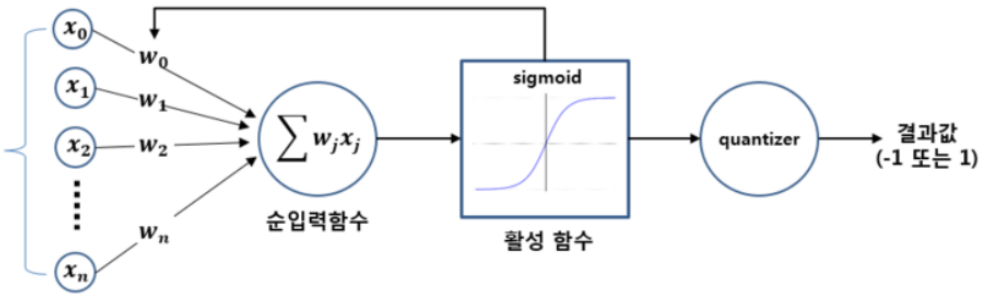
# 2. 인공 신경망 분석

## 활성화 함수의 등장

- "퍼셉트론에서는 활성화 함수로 계단 함수를 이용한다" 라고 할 수 있음. 가장 일반적인 활성화 함수로는 계단, 부호, 선형, 시그모이드 함수가 있다.
- 신경망에서는 활성화 함수로 시그모이드 함수를 이용하여 신호를 변환하고, 그 변환된 신호를 다음 뉴런에 전달




$$h(x) = \begin{cases} 0 & (x \le 0) \\ 1 & (x > 0) \end{cases}$$

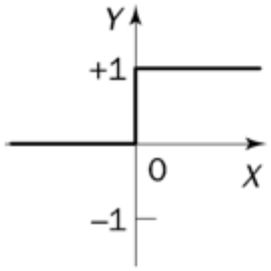


# 2. 인공 신경망 분석

## 활성화 함수의 등장

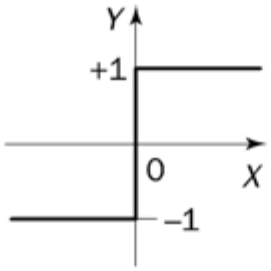
- 가장 일반적인 활성화 함수로는 계단, 부호, 선형, 시그모이드 함수가 있다.

계단 함수



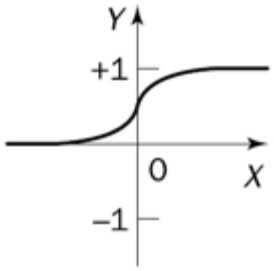

$$Y^{step} = \begin{cases} 1, \text{ if } X \geq 0 \\ 0, \text{ if } X < 0 \end{cases}$$

부호 함수



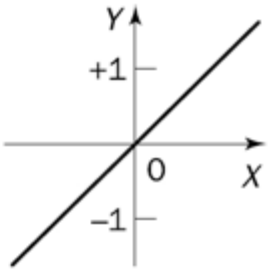

$$Y^{sign} = \begin{cases} +1, \text{ if } X \geq 0 \\ -1, \text{ if } X < 0 \end{cases}$$

시그모이드 함수




$$Y^{sigmoid} = \frac{1}{1 + e^{-X}}$$

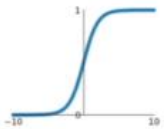
선형 함수




$$Y^{linear} = X$$

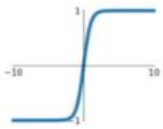
**Sigmoid**

$\sigma(x) = \frac{1}{1+e^{-x}}$



**tanh**

$\tanh(x)$

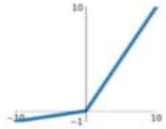


**ReLU**

$\max(0, x)$

**Leaky ReLU**

$\max(0.1x, x)$

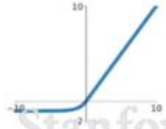


**Maxout**

$\max(w_1^T x + b_1, w_2^T x + b_2)$

**ELU**

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



# 2. 인공 신경망 분석

## 인공신경망을 이용한 데이터분석

● 데이터 셋 로드

```
# sklearn의 datasets에서 load_iris를 로드
from sklearn.datasets import load_iris
# iris데이터셋을 iris라는 변수에 저장
iris = load_iris()

# iris에 있는 key값을 나타냄
iris.keys()
```

```
dict_keys(['data', 'target', 'target_names', 'DESCR', 'feature_names'])
```

```
# iris의 데이터에 해당하는 부분의 X와 Y의 크기를 나타냄
iris['data'].shape
```

```
(150, 4)
```

# 2. 인공 신경망 분석

## 인공신경망을 이용한 데이터분석

- 데이터 셋 로드

```
# iris데이터셋의 0번째부터 9번째까지를 슬라이싱해서 나타냄
iris['data'][0:10]
```

`

```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1]])
```

`

# 2. 인공 신경망 분석

## 인공신경망을 이용한 데이터분석

- 데이터 셋 분리

```
# X에는 iris데이터의 값 150x4의 크기를 입력
# y에는 분류하고자 하는 target변수를 입력
# target변수는 데이터가 무엇인지에 대해 판별하는 값
# iris target의 경우 0, 1, 2로 구분됨
X = iris['data']
y = iris['target']

# 위의 데이터를 train과 test로 구분
# sklearn의 model_selection 내에 train_test_split를 로드
# train_test_split를 이용해 위의 X변수에 선언한 data값과 y변수에 선언한 target
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y)
```

## 인공신경망을 이용한 데이터분석

- 데이터 셋 표준화와 정규화

```
# sklearn 내에 preprocessing의 StandardScaler를 로드
# StandardScaler는 정규화를 시키는 함수
# StandardScaler는 데이터의 범위를 평균 0, 표준편차 1의 범위로 바꿔주는 함수
# 그리고 StandardScaler를 scaler라는 변수에 저장해 사용
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

# StandardScaler를 담은 변수에 X_train을 학습해 데이터를 정규화
scaler.fit(X_train)

# X_train과 X_test를 StandardScaler를 이용해 정규화
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

# 2. 인공 신경망 분석

## 인공신경망을 이용한 데이터분석

- MLP 알고리즘 로드 및 히든 레이어 할당

```
# 다중인공신경망(MLP) 분류 알고리즘을 sklearn의 neural_network에서 로드
from sklearn.neural_network import MLPClassifier

# MLP 알고리즘의 히든레이어를 3계층(10,10,10)으로 할당
mlp = MLPClassifier(hidden_layer_sizes=(10,10,10))
```

## 인공신경망을 이용한 데이터분석

- 데이터 학습과 학습 성능 평가

```
# 위에서 분류한 X_train과 y_train을 MLP를 이용해 학습
mlp.fit(X_train, y_train)

# mlp로 학습한 내용을 X_test에 대해 예측하여 predictions변수에 저장
predictions = mlp.predict(X_test)

# sklearn.metrics의 confusion_matrix와 classification_report를 로드
# confusion_matrix는 데이터가 맞는지의 유무를 판단
# classification_report는 precision과 recall 그리고 f1_score등을 계산해 정확률에 대해 계산
from sklearn.metrics import classification_report, confusion_matrix

# confusion_matrix를 이용해 실제값 y_test와 예측값에 대해 비교
print(confusion_matrix(y_test, predictions))

`

[[10  0  0]
 [ 0 10  0]
 [ 0  3 15]]`
```

# 2. 인공 신경망 분석

## 인공신경망을 이용한 데이터분석

- 정확률, 재현율, f1-score를 확인

```
# classification_report를 이용해 정확률, 재현율, f1-score를 확인
print(classification_report(y_test, predictions))

>>
 precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       0.77      1.00      0.87        10
           2       1.00      0.83      0.91        18

    accuracy                           0.92        38
   macro avg       0.92      0.94      0.93        38
weighted avg       0.94      0.92      0.92        38
```

『5과목』 Data Analysis과 머신러닝

# 비지도-자율학습

# Syallabus

## 학습목표

- 이 워크샵에서는 자율학습 모델, K-means 클러스터링(K-Means Clustering)기법에 대해 알 수 있습니다.

## 눈높이 체크

- K-means 클러스터링(K-Means Clustering) 을 알고 계시나요?

# 1. 자율학습 개념

## 자율학습?

- 자율학습 혹은 비지도 학습(Unsupervised Learning)머신러닝 기법은 데이터 세트에 목적변 수(혹은 반응변수)(Y)가 없이, 일련의 변수들 X1, X2, X3, ..., Xp만 주어진 경우에 시행하게 되는 머신러닝 기법들이다. 일련의 설명변수들과 연관된 목적변수(혹은 반응변수)가 없기 때문에, 무엇을 예측한다기보다는 주어진 데이터에서 특정한 패턴이나 알려지지 않은 지
- 식을 발견하고자 하는 것이 목표라고 할 수 있다. 앞서 다룬 지도학습 머신러닝 기법들이 목적변수(혹은 반응변수)의 형태에 따라 분류 및 수치예측 수행이라는 명확한 목표가 있고, 데이터 세트 분할 통한 교차검증 등의 분석결과에 대한 명확한 평가 기준을 가지고 분석을 진행할 수 있는 것에 비해 자율학습 기법은 무엇을 발견하고자 하는 것인지 명확하지 않으며, 예측 대상을 '지도'할 수 없으므로, 컴퓨터 프로그램은 실제로 정답을 알 수 없다. 이로 인해 머신러닝 결과가 만족스러운 것인지 점검하기가 곤란하다는 문제가 있다.

# 1. 자율학습 개념

## 자율학습?

- 이러한 자율학습 머신러닝 기법은 분석의 목적 및 알고리즘에 따라, 유사한 개체나 사람들을 그룹 짓는 군집화(Clustering), 특정 대상들 간의 발생 관련성을 파악하는 연관성 분석 (Association), 주어진 변수 세트를 효과적으로 설명 가능한 더 적은 수의 대표적인 변수들로 요약하는 차원축소 (Dimension Reduction) 등의 기법이 있다.

# 2. 클러스터링(군집)

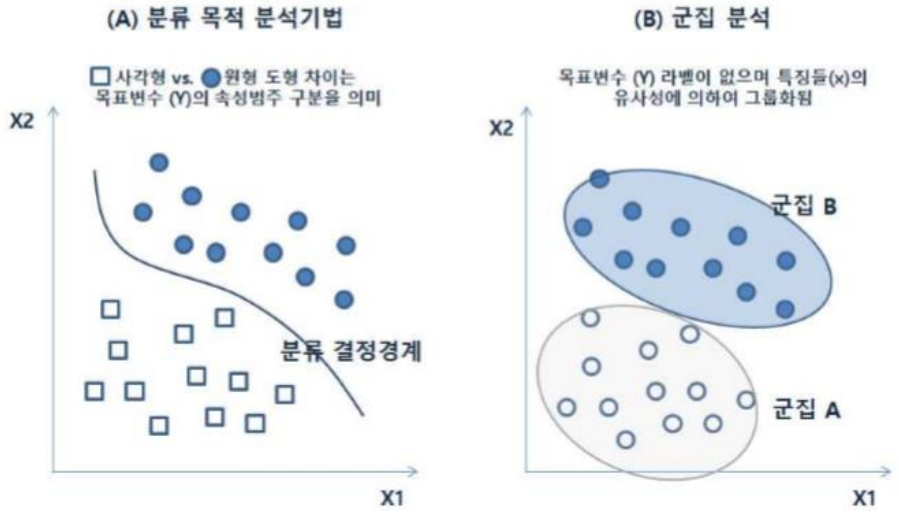
## 클러스터링(군집) 분석?

- 군집분석 혹은 클러스터링(Clustering) 분석은 대표적인 자율학습(혹은 비지도 학습)으로서, 일련의 관측값들을 적절한 기준으로 서로 유사한 관측값끼리 그룹으로 묶는(군집화) 기법을 말한다. 군집화는 사전에 그룹이 어떤 형태인지 모르는 상태에서 실행하게 된다. 분석가가 찾고 있는 것이 무엇인지 모르는 상태에서 실행하기 때문에 당연히 분석가가 컴퓨터 프로그램에게 무엇을 지도할 수 없다. 그런 의미에서 군집분석이 자율학습(비지도 학습)으로 불리고 있는 것이다. 또한, 동일한 이유로 군집분석은 무엇을 예측하기 위해 실행하기보다는 지식 발견 그 자체를 위한 목적으로 주로 활용된다. 그런 측면에서 보자면 데이터를 통해 학습하여 무엇을 예측하려는 머신러닝이라기 보다는 지식발견 그 자체 혹은 알려지지 않았던 통찰을 발견하려는 데이터 마이닝에 좀 더 가깝다고 할 수 있다.

# 2. 클러스터링(군집)

## 클러스터링(군집) 분석 원리

- 군집분석은 입력된 데이터가 어떤 형태로 그룹을 형성하는지가 분석의 핵심 목적이다. 따라서 군집분석에서는 입력된 데이터를 어떤 기준으로 그룹화 하는지가 첫 번째 질문이 되는데, 일반적으로는 각 데이터 간의 유사성을 기준으로 그룹화를 짓게 된다. 즉, 군집 내의 데이터는 서로 매우 유사하지만, 다른 군집과는 다르다는 원칙으로 그룹화를 진행하게 된다. 군집분석은 목적변수가 없는 상태에서 유사한 특성을 가진 개체끼리 그룹화를 한다.



# 2. 클러스터링(군집)

## 클러스터링(군집) 분석 주요 활용 분야

- 군집분석은 데이터들에 존재하는 유사성 혹은 비유사성에 근거해서 패턴화를 하는 특성으로 인해, 매우 다양한 영역에서 활용할 수 있다. 군집분석이 주로 활용되는 분야에 대한 대표적인 예시는 다음과 같다.

1. 마케팅 등 분야에서의 고객 세분화(Segmentation)
2. 질병 및 환자 특성에 따른 유사 그룹화
3. 개체 유사성에 근거한 문서 분류
4. 디지털 이미지 인식 통한 사물 및 안면 인식
5. 금융 분야에서의 알려진 군집 이외의 사용 패턴 식별 (신용카드 사기, 보험료 과다 청구 등)
6. 공학 분야에서의 이상치 탐지 (제조 과정에서의 불량제품 자동 탐지, 통화 음질 개선을 위한 노이즈 구별 등)
7. 컴퓨터 네트워크에 비인가 된 침입 등의 비정상적 행위 탐지

- 상기 몇 가지 예시에서 볼 수 있는 바와 같이, 주로 유사한 사람들을 그룹화 및 세분화하여 비즈니스에 활용하거나, 유사한 개체들을 묶어 필요한 정보를 압축하여 활용하거나, 유사성 기반의 그룹화를 응용하여 역으로 이상치(Outlier) 등의 특이점 혹은 비정상 패턴을 찾는 분야 등에 활용되고 있다. 이 외에도 매우 다양한 영역에서 응용 및 활용되고 있는 중요한 분석 기법이라고 할 수 있다.

# 2. 클러스터링(군집)

## 클러스터링(군집) 분석의 주요 종류

| 구분 | 기법 | 주요 내용 |
|---|---|---|
| 비계층 군집 (분할 기반 군집 | K-평균(K-Means) 클러스터링 | 주어진 군집 수 k 에 대해서 군집 내 거리 제곱 합 의 합을 최소화하는 형태로 데이터 내의 개체들을 서로 다른 군집으로 그룹화하는 기법 |
| | K-Medoids 클러스터링 혹은 (PAM : Partitioning Around Method) | K-평균 클러스터링의 보완한 기법으로서, 모든 형태 의 유사성(비유사성) 측도를 사용하며, 좌표평면상 임의의 점이 아닌 실제 데이터 세트 내의 값을 사 용하여 클러스터 중심을 정하므로 노이즈나 이상치 처리에 강건한 군집화 기법 |
| | DBSCAN (Density Based Spatial Clustering of Application with Noise) | K-평균 기법이 K개의 평균과 각 데이터 점들 간의 거리를 계산하여 그룹화를 하는 반면, DBSCAN은 밀 도개념을 도입하여 일정한 밀도로 연결된 데이터집 합은 동일한 그룹으로 판정하여 노이즈 및 이상치 식별에 강한 군집화 기법 |
| | 자기 조직화 지도 (Self Organizing Map) | 자율학습 목적의 머신러닝에 속하는 인공 신경망의 한 기법으로서 벡터 수량화 네트워크를 이용한 군 집화 기법 |
| | Fuzzy 군집 | K-평균 기법이 하나의 데이터 개체는 하나의 군집 에만 배타적으로 속하는 독점적 군집인데 반해, 퍼 지군집은 하나의 데이터 개체가 여러 개의 군집에 중복해서 속할 수 있도록 하는 중복 군집화 기법 |
| 계층적 군집 | 병합적(Agglomerative) 혹은 상 향식(Bottom-up) 군집화 | 모든 데이터 객체를 별개의 그룹으로 구성한 뒤, 단 하나의 그룹화가 될 때까지 각 그룹을 단계적으 로 합쳐가는 계층적 군집기법 |
| | 분할식(Divisive) 혹은 하향식(Top-down) 군집화 | 모든 데이터 객체를 하나의 그룹으로 구성한 뒤, 각 데이터 점이 하나의 그룹으로 될 때까지 단계적으 로 분할에 가는 계층적 군집기법 |
| 확률 기반 군집 | 가우스 혼합 모형 | EM (Expectation Maximization) 알고리즘 혹은 MCMC (Markov Chain Monte Carlo) 등의 알고리즘을 사용하 여 모수를 추정하는 확률 기반의 군집분석 |

# 2. 클러스터링(군집)

## 군집화 성능기준

- 군집화의 경우에는 분류문제와 달리 성능기준을 만들기 어렵다. 심지어는 원래 데이터가 어떻게 군집화되어 있었는지를 보여주는 정답(groundtruth)이 있는 경우도 마찬가지이다. 따라서 다양한 성능기준이 사용되고 있다. 다음의 군집화 성능기준의 예다.

1. 조정 랜드지수(Adjusted Rand Index)
2. 조정 상호정보량 (Adjusted Mutual Information)
3. 실루엣계수 (Silhouette Coefficient)

- 일치행렬
- 랜드지수를 구하려면 데이터가 원래 어떻게 군집화되어 있어야 하는지를 알려주는 정답(groundtruth)이 있어야 한다. N 개의 데이터 집합에서 i , j 두 개의 데이터를 선택하였을 때 그 두 데이터가 같은 군집에 속하면 1 다른 데이터에 속하면 0이라고 하자. 이 값을 N×N 행렬 T 로 나타내면 다음과 같다.

$$T_{ij} = \begin{cases} 1 & i\text{와 } j\text{가 같은 군집} \\ 0 & i\text{와 } j\text{가 다른 군집} \end{cases}$$

# 2. 클러스터링(군집)

## 군집화 성능기준

- 예를 들어 {0,1,2,3,4} 라는 5개의 데이터 집합에서 {0,1,2} 와 {3,4} 가 각각 같은 군집라면 행렬 T 는 다음과 같다.

```
import numpy as np

groundtruth = np.array([
    [1, 1, 1, 0, 0],
    [1, 1, 1, 0, 0],
    [1, 1, 1, 0, 0],
    [0, 0, 0, 1, 1],
    [0, 0, 0, 1, 1],
])
groundtruth
```

- 이제 군집화 결과를 같은 방법으로 행렬 C 로 표시하자. 만약 군집화이 정확하다면 이 행렬은 정답을 이용해서 만든 행렬과 거의 같은 값을 가져야 한다. 만약 군집화 결과 {0,1} 과 {2,3,4} 가 같은 군집라면 행렬 C 는 다음과 같다.

```
clustering = np.array([
    [1, 1, 0, 0, 0],
    [1, 1, 0, 0, 0],
    [0, 0, 1, 1, 1],
    [0, 0, 1, 1, 1],
    [0, 0, 1, 1, 1],
])
clustering
```

# 2. 클러스터링(군집)

## 군집화 성능기준

- 이 두 행렬의 모든 원소에 대해 값이 같으면 1 다르면 0으로 계산한 행렬을 일치행렬(incidence matrix)이라고 한다. 즉 데이터 집합에서 만들수 있는 모든 데이터 쌍에 대해 정답과 군집화 결과에서 동일한 값을 나타내면 1, 다르면 0이 된다.

$$R_{ij} = \begin{cases} 1 & \text{if } T_{ij} = C_{ij} \\ 0 & \text{if } T_{ij} \neq C_{ij} \end{cases}$$

- 즉, 원래 정답에서 1번 데이터와 2번 데이터가 같은(다른) 군집인데 군집화 결과에서도 같은(다른) 군집이라고 하면 $R_{12}=1$ 이다.
- 위 예제에서 일치행렬을 구하면 다음과 같다.

```
incidence = 1 * (groundtruth == clustering)  # 1*는 True/False를 숫자 0/1로 바꾸기 위한 계산
incidence
>>
array([[1, 1, 0, 1, 1],
       [1, 1, 0, 1, 1],
       [0, 0, 1, 0, 0],
       [1, 1, 0, 1, 1],
       [1, 1, 0, 1, 1]])
```

# 2. 클러스터링(군집)

## 군집화 성능기준

- 이 일치 행렬은 두 데이터의 순서를 포함하므로 대칭행렬이다. 만약 데이터의 순서를 무시한다면 위 행렬에서 대각성분과 아래쪽 비대각 성분은 제외한 위쪽 비대각 성분만을 고려해야 한다. 위쪽 비대각 성분에서 1의 개수는 다음과 같아진다.

- a=T에서 같은 군집에 있고 C에서도 같은 군집에 있는 데이터 쌍의 수
- b=T에서 다른 군집에 있고 C에서도 다른 군집에 있는 데이터 쌍의 수

- 일치행렬 위쪽 비대각 성분에서 1의 개수=a+b

```
np.fill_diagonal(incidence, 0) # 대각성분 제외
a_plus_b = np.sum(incidence) / 2 # 대칭행렬이므로 절반만 센다.
a_plus_b

>>

6.0
```

# 2. 클러스터링(군집)

## 군집화 성능기준

- 랜드지수
- 랜드지수(Rand Index, RI)는 가능한 모든 데이터 쌍의 개수에 대해 정답인 데이터 쌍의 개수의 비율로 정의한다.

$$\text{Rand Index} = \frac{a+b}{{}_N C_2}$$

- shape[0], shape[1]를 이용하여 전체 행의 갯수와 열의 갯수를 반환

```
from scipy.special import comb
rand_index = a_plus_b / comb(incidence.shape[0], 2)
rand_index

>>

0.6
```

# 2. 클러스터링(군집)

## K-평균 군집화(K-Means Clustering)

- K-평균 클러스터링은 주어진 군집 수 k 에 대하여 군집 내 거리 제곱 합의 합을 최소화하는 것을 목적으로 하며, 다음과 같은 목적함수 값이 최소화될 때까지 군집의 중심위치와 각 데이터가 소속될 군집를 반복해서 찾는다. 이 값을 관성(inertia)이라 한다.

$$J = \sum_{k=1}^{K} \sum_{i \in C_k} d(x_i, \mu_k)$$

- 이 식에서 K 는 군집의 갯수이고 Ck 는 k 번째 군집에 속하는 데이터의 집합, μk 는 k 번째 군집의 중심위치(centroid), d 는 xi,μk 두 데이터 사이의 거리 혹은 비유사도(dissimilarity)로 정의한다. 만약 유클리드 거리를 사용한다면 다음과 같다.

$$d(x_i, \mu_k) = ||x_i - \mu_k||^2$$

- 위 식은 다음처럼 표현할 수도 있다.

$$J = \sum_{i=1}^{N} \min_{\mu_j \in C} (||x_i - \mu_j||^2)$$

# 2. 클러스터링(군집)

## K-평균 군집화(K-Means Clustering)

- 세부 알고리즘은 다음과 같다.

1. 임의의 중심위치 $\mu_k(k=1,\ldots,K)$ 를 고른다. 보통 데이터 표본 중에서 K 개를 선택한다.
2. 모든 데이터 $x_i(i=1,\ldots,N)$ 에서 각각의 중심위치 $\mu_k$ 까지의 거리를 계산한다.
3. 각 데이터에서 가장 가까운 중심위치를 선택하여 각 데이터가 속하는 군집을 정한다.
4. 각 군집에 대해 중심위치 $\mu_k$ 를 다시 계산한다.
5. 2 ~ 4를 반복한다.

- K-평균 군집화란 명칭은 각 군집의 중심위치를 구할 때 해당 군집에 속하는 데이터의 평균(mean)값을 사용하는데서 유래하였다. 만약 평균 대신 중앙값(median)을 사용하면 K-중앙값(K-Median) 군집화라 한다.

# 2. 클러스터링(군집)

## K-평균 군집화(K-Means Clustering)

- scikit-learn의 cluster 서브패키지는 K-평균 군집화를 위한 KMeans 클래스를 제공한다. 다음과 같은 인수를 받을 수 있다.

  - ✓ n_clusters: 군집의 갯수
  - ✓ init: 초기화 방법. "random"이면 무작위, "k-means++"이면 K-평균++ 방법. 또는 각 데이터의 군집 라벨.
  - ✓ n_init: 초기 중심위치 시도 횟수. 디폴트는 10이고 10개의 무작위 중심위치 목록 중 가장 좋은 값을 선택한다.
  - ✓ max_iter: 최대 반복 횟수.
  - ✓ random_state: 시드값.

- 다음은 make_blobs 명령으로 만든 데이터를 2개로 K-평균 군집화하는 과정을 나타낸 것이다. 각각의 그림은 군집을 정하는 단계 3에서 멈춘 것이다. 마커(marker)의 모양은 소속된 군집을 나타내고 크기가 큰 마커가 해당 군집의 중심위치다. 각 단계에서 중심위치는 전단계의 군집의 평균으로 다시 계산되는 것을 확인할 수 있다.

# 2. 클러스터링(군집)

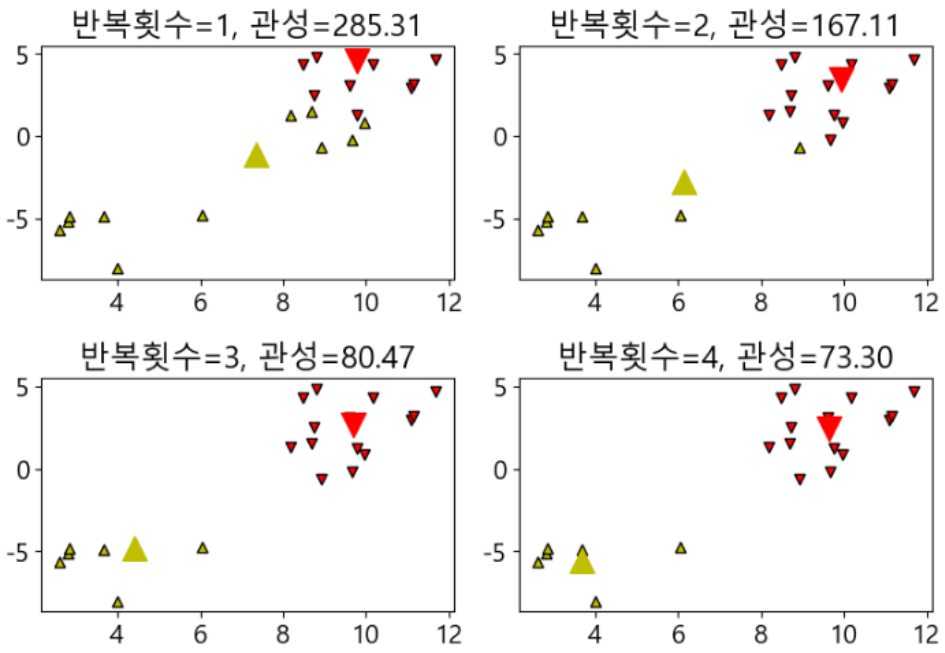
## K-평균 군집화(K-Means Clustering)

```
import matplotlib as mpl
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans

X, _ = make_blobs(n_samples=20, random_state=4)

def plot_KMeans(n):
    model = KMeans(n_clusters=2, init="random", n_init=1, max_iter=n,
random_state=6).fit(X)
    c0, c1 = model.cluster_centers_
    plt.scatter(X[model.labels_ == 0, 0], X[model.labels_ == 0, 1], marker='v',
facecolor='r', edgecolors='k')
    plt.scatter(X[model.labels_ == 1, 0], X[model.labels_ == 1, 1], marker='^',
facecolor='y', edgecolors='k')
    plt.scatter(c0[0], c0[1], marker='v', c="r", s=200)
    plt.scatter(c1[0], c1[1], marker='^', c="y", s=200)
    plt.grid(False)
    plt.title("반복횟수={}, 관성={:5.2f}".format(n, -model.score(X)))
```

# 2. 클러스터링(군집)

## K-평균 군집화(K-Means Clustering)

```
plt.figure(figsize=(8, 8))
plt.subplot(321)
plot_KMeans(1)
plt.subplot(322)
plot_KMeans(2)
plt.subplot(323)
plot_KMeans(3)
plt.subplot(324)
plot_KMeans(4)
plt.tight_layout()
plt.show()
```




- plot_KMeans(n)은 KMeans 모델을 훈련하고, 특정 반복 횟수에 따른 클러스터링 결과를 시각화

# 2. 클러스터링(군집)

## K-평균++ 알고리즘

- K-평균++ 알고리즘은 초기 중심위치를 설정하기 위한 알고리즘이다. 다음과 같은 방법을 통해 되도록 멀리 떨어진 중심위치 집합을 찾아낸다.
  1. 중심위치를 저장할 집합 M 준비
  2. 일단 하나의 중심위치 $\mu_0$ 를 랜덤하게 선택하여 M 에 넣는다.
  3. M 에 속하지 않는 모든 표본 $x_i$ 에 대해 거리 $d(M,x_i)$ 를 계산. $d(M,x_i)$ 는 M 안의 모든 샘플 $\mu_k$ 에 대해 $d(\mu_k,x_i)$ 를 계산하여 가장 작은 값 선택
  4. $d(M,x_i)$ 에 비례한 확률로 다음 중심위치 $\mu$ 를 선택.
  5. K 개의 중심위치를 선택할 때까지 반복
  6. K-평균 방법 사용

- 다음은 K-평균++ 방법을 사용하여 MNIST Digit 이미지 데이터를 군집화한 결과이다. 각 군집에서 10개씩의 데이터만 표시하였다.

# 2. 클러스터링(군집)

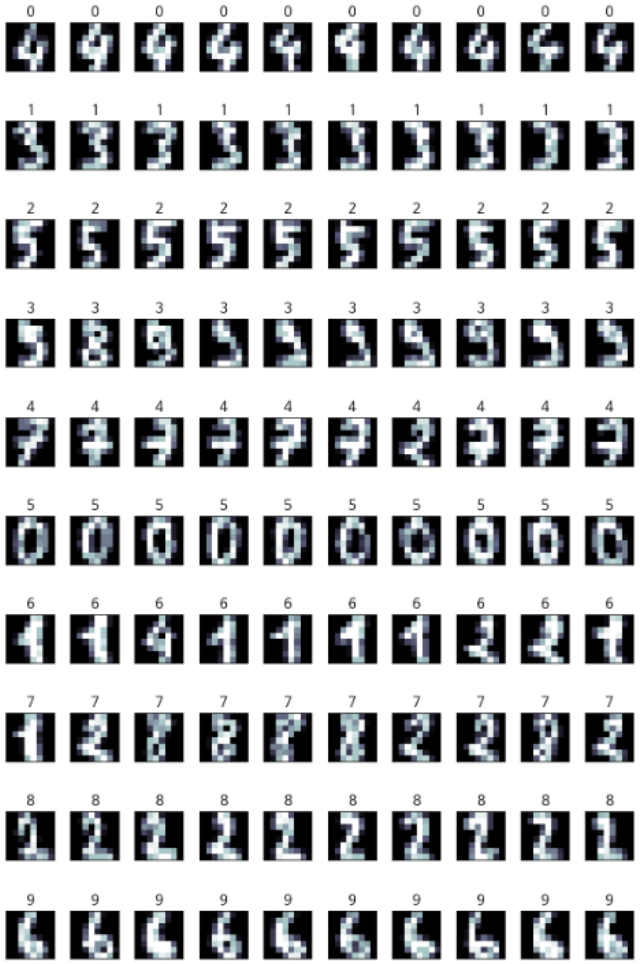
## K-평균++ 알고리즘

```
from sklearn.datasets import load_digits

digits = load_digits()

model = KMeans(init="k-means++", n_clusters=10, random_state=0)
model.fit(digits.data)
y_pred = model.labels_

def show_digits(images, labels):
    f = plt.figure(figsize=(8, 2))
    i = 0
    while (i < 10 and i < images.shape[0]):
        ax = f.add_subplot(1, 10, i + 1)
        ax.imshow(images[i], cmap=plt.cm.bone)
        ax.grid(False)
        ax.set_title(labels[i])
        ax.xaxis.set_ticks([])
        ax.yaxis.set_ticks([])
        plt.tight_layout()
        i += 1

def show_cluster(images, y_pred, cluster_number):
    images = images[y_pred == cluster_number]
    y_pred = y_pred[y_pred == cluster_number]
    show_digits(images, y_pred)

for i in range(10):
    show_cluster(digits.images, y_pred, i)
```

# 2. 클러스터링(군집)

## K-평균++ 알고리즘

- 각 클러스터마다 어떤 숫자들이 모여 있는지를 시각적으로 확인할 수 있습니다. 이 예제는 KMeans 알고리즘이 숫자 이미지 데이터를 어떻게 클러스터링하는지를 시각화하여 보여줍니다.



『5과목』 Data Analysis과 머신러닝

# Association Rule Analysis

# Syallabus

## 학습목표

- 이 워크샵에서는 Association Rule Analysis 확인할 수 있다

## 눈높이 체크

- Association Rule Analysis 을 알고 계시나요?

# 1. 연관성 분석(장바구니 분석)

## 연관성 분석(장바구니 분석) 란?

- 연관성 분석 기법은 방대한 데이터 세트에서 객체나 아이템 간의 연관관계를 찾아내는 분석기법이다. Y값(종속변수, 목표변수)이 없는 상태에서 데이터 속에 숨겨져 있는 패턴, 규칙을 찾아내는 비지도학습(unsupervised learning)의 하나인 '연관규칙분석(Association Rule Analysis)', 혹은 유통업계에서 사용하는 용어로 '장바구니분석(Market Basket Analysis)'이라고도 한다.
- 상품 추천에 사용하는 분석기법

1. 연관규칙분석, 장바구니분석 (Association Rule Analysis, Market Basket Analysis) : 고객의 대규모 거래데이터로부터 함께 구매가 발생하는 규칙(예: A à 동시에 B)을 도출하여, 고객이 특정 상품 구매 시 이와 연관성 높은 상품을 추천
2. 순차분석 (Sequence Analysis) : 고객의 시간의 흐름에 따른 구매 패턴(A à 일정 시간 후 B)을 도출하여, 고객이 특정 상품 구매 시 일정 시간 후 적시에 상품 추천
3. Collaborative Filtering : 모든 고객의 상품 구매 이력을 수치화하고, 추천 대상이 되는 고객A와 다른 고객B에 대해 상관계수를 비교해서, 서로 높은 상관이 인정되는 경우 고객B가 구입 완료한 상품 중에 고객A가 미구입한 상품을 고객A에게 추천

# 1. 연관성 분석(장바구니 분석)

## 연관성 분석(장바구니 분석) 란?

4. Contents-based recommendation : 고객이 과거에 구매했던 상품들의 속성과 유사한 다른 상품 아이템 중 미구매 상품을 추천 (⇔ Collaborative Filtering은 유사 고객을 찾는 것과 비교됨)
5. Who-Which modeling : 특정 상품(군)을 추천하는 모형을 개발 (예: 신형 G5 핸드폰 추천 스코어모형)하여 구매 가능성 높은(예: 스코어 High) 고객(군) 대상 상품 추천

- 넥플릭스에서 상품추천에 이용하는 알고리즘
- 넥플릭스는 이용자들이 동영상에 매긴 별점과 위치정보, 기기정보, 플레이버튼 클릭 수, 평일과 주말에 따른 선호 프로그램, 소셜 미디어 내에서 언급된 횟수 등을 분석해 알고리즘을 개발했다.
- 출처 : 넷플릭스의 빅데이터, 인문학적 상상력과의 접점, 조영신, KISDI 동향 Focus
- 규칙(rule)이란 "if condition then result" (if A --> B) 의 형식으로 표현을 합니다.
- 연관규칙(association rule)은 특정 사건이 발생하였을 때 함께 (빈번하게) 발생하는 또 다른 사건의 규칙을 말합니다.

# 1. 연관성 분석(장바구니 분석)

## 연관성 분석(장바구니 분석) 란?

- 대량의 트랜잭션 데이터에서의 규칙성이나 패턴화 도출의 목적이 주어진 경우, 이의 문제 해결을 위해 다양한 연관 규칙 알고리즘을 비교하고 적용할 수 있다.
- 연관성 분석 기법은 방대한 데이터 세트에서 객체나 아이템 간의 연관관계를 찾아내는 분석기법이다. 연관관계는 빈발 아이템이나 연관규칙의 형태로 표현되는데 빈발 아이템은 {X,Y}처럼 표현하고, 연관 규칙은 {X}→{Y}처럼 특정 아이템 'X'가 발생하면, 'Y'가 함께 발생한다는 형태로 표현한다. 예를 들어 {순대, 족발}→{보쌈}처럼 순대와 족발을 구매하면 보쌈도 함께 구매한다는 규칙을 찾아내는 것이다. 여기서 {순대, 족발}은 해당 규칙에서의 '조건'(antecedent)에 해당하고, {보쌈}은 '결과'(consequence)에 해당하게 된다. 이러한 연관성 분석은 사전에 목표변수(Y)가 주어지지 않으며, 각 트랜잭션(혹은 거래데이터)에서 객체나 아이템 간의 패턴을 찾아내는 기법이므로 자율학습(비지도 학습) 기법에 속하게 된다.

# 1. 연관성 분석(장바구니 분석)

## 연관성 분석 주요 측도

- 연관성 분석에서 가장 핵심적인 개념은 각 아이템 간의 연관성을 파악하는 주요 3개 측도인 지지도(Support), 신뢰도(Confidence), 향상도(Lift)이다.

• 지지도(Support)

– 지지도는 전체 데이터 세트에서 해당 아이템 집합이 포함된 비율을 말하며 아래의 S(X) 혹은 S(X,Y)와 같이 표현된다.

$$S(X) = \frac{count(X)}{N} \qquad (4.11)$$

$$S(X,Y) = \frac{count(X,Y)}{N} = P(X \cap Y)$$

– 즉, 지지도는 빈도적 관점에서 확률을 정의할 때, 전체 데이터 세트 중 아이템 집합 {X,Y}가 발생할 확률과 같다.

# 1. 연관성 분석(장바구니 분석)

## 연관성 분석 주요 측도

- 신뢰도(Confidence)

- 신뢰도는 연관규칙 {X}→{Y}에서 '조건' X를 포함한 아이템 세트 중에서 X, Y 둘 다 포함된 아이템 세트가 발생한 비율을 말하는데, 규칙의 왼쪽에 있는 '조건 X'가 발생했다는 조건하에 규칙의 오른쪽에 있는 '결과 Y'가 발생할 확률을 의미한다. 신뢰도는 특정 연관 규칙의 예측력이나 정확도에 대한 측정이다. 사실 이 신뢰도는 조건부 확률 P(Y|X) 와 동일한 의미이다.

$$Conf(X \Rightarrow Y) = \frac{S(X,Y)}{S(X)} = \frac{\frac{count(X,Y)}{N}}{\frac{count(X)}{N}} = \frac{count(X,Y)}{count(X)} \qquad (4.12)$$

$$= \frac{P(X \cap Y)}{P(X)} = P(Y|X)$$

- 신뢰도 (X→Y)와 신뢰도 (Y→X)는 서로 같지 않다. 즉, 두 신뢰도 모두 X, Y가 모두 포함된 지지도(X,Y)가 분자에 포함되어 있지만 신뢰도 (X→Y)는 지지도(X)가 분모에 들어가게 되고, 신뢰도 (Y→X)는 지지도(Y)가 분모에 들어가게 되는 점이 다르다. 결국 이는 조건부 확률 P(Y|X)와 P(X|Y)가 서로 다른 결과를 가져오는 것과 동일한 의미라고 할 수 있다.

# 1. 연관성 분석(장바구니 분석)

## 연관성 분석 주요 측도

- 향상도(Lift)
  - 지지도와 신뢰도는 연관규칙 생성과 탐사에 있어서 매우 중요한 핵심개념임에는 틀림없지만, 지지도(X,Y)와 신뢰도(X→Y)가 높았다는 것만으로 유의미한 규칙이라고 결론 내리기는 어렵다. 그 이유는 만일 전체 데이터 세트에서 원래부터 아이템 세트 {Y}가 포함된 경우의 수가 많았다면, 아이템 세트 {Y}와 함께 아이템 세트 {X}가 포함되어 있을 가능성도 그만큼 커지고, 지지도(X,Y)와 신뢰도(X→Y)는 높게 나타날 수밖에 없기 때문이다.
  - 즉, 연관규칙 {X}→{Y}가 탐색 되었을 때 정말로 조건 {X}가 발생했을 때 결과 {Y}가 함께 나타나는 경우의 수가 많아서 그런 것인지, 아니면 원래부터 {Y}가 많이 포함되어 있어 연관규칙 {X}→{Y}가 탐색 된 것인지에 대한 보다 명확한 측정 지표가 필요하게 된다. 이럴 때 이를 측정하는 지표가 바로 향상도(Lift)이다.
  - 향상도(Lift)가 의미하는 바는 조건 {X}가 주어지지 않았을 때의 결과 {Y}가 발생할 확률 대비, 조건 {X}가 주어졌을 때의 결과 {Y}의 발생 확률의 증가 비율을 의미한다. 즉, 아이템 집합 {Y}가 원래 발생된 경우의 수보다 연관규칙 {X}→{Y}가 탐색 되었을 때 조건에 해당하는 아이템 집합 {X}가 주어졌다는 "정보"가 결과 아이템 집합 {Y}가 발생하게 되는 경우의 수를 예상하는 데 얼마나 유용하느냐를 나타낸다고 할 수 있다.

# 1. 연관성 분석(장바구니 분석)

## 연관성 분석 주요 측도

- 향상도(Lift)
  - 향상도(Lift)는 다음과 같이 측정된다.

$$Lift(X \Rightarrow Y) = \frac{Conf(X \Rightarrow Y)}{S(Y)} \tag{4.13}$$

$$= \frac{\frac{S(X,Y)}{S(X)}}{S(Y)} = \frac{S(X,Y)}{S(X)S(Y)} = \frac{\frac{count(X,Y)}{N}}{\frac{count(X)}{N}\frac{count(Y)}{N}}$$

$$= \frac{P(Y|X)}{P(Y)} = \frac{\frac{P(X \cap Y)}{P(X)}}{P(Y)} = \frac{P(X \cap Y)}{P(X)P(Y)}$$

  - 따라서 향상도가 1을 넘어가면 조건과 결과 아이템 집합 간에 서로 양의 상관관계가 있으며, 1보다 작으면 서로 음의 상관관계 만일 향상도가 1로 나타나면 조건과 결과 아이템 집합은 서로 독립적인 관계라고 할 수 있다.

# 1. 연관성 분석(장바구니 분석)

## 연관성 분석 주요 측도

- 예

지지도(Support), 신뢰도(Confidence), 향상도(Lift) 예시

| Customer ID | Transaction ID | Items |
|---|---|---|
| 1131 | 1번 | 계란, 우유 |
| 2094 | 2번 | 계란, 기저귀, 맥주, 사과 |
| 4122 | 3번 | 우유, 기저귀, 맥주, 콜라 |
| 4811 | 4번 | 계란, 우유, 맥주, 기저귀 |
| 8091 | 5번 | 계란, 우유, 맥주, 콜라 |

N = 5 (전체 transaction 개 수)

s(Y) = n(Y)/ N
= n{2번, 3번, 4번}/N = 3/5 = 0.6

연관규칙 {계란, 맥주} → {기저귀} 에 대해
X Y

▪ 지지도(Support)

s(X→Y) = n(X∪Y)/N
= n{2번, 4번}/ N
= 2/5 = 0.4

▪ 신뢰도(Confidence)

c(X→Y) = n(X∪Y)/ n(X)
= n{2번, 4번}/ n{2번, 4번, 5번}
= 2/3 = 0.667

▪ 향상도(Lift)

Lift(X→Y) = c(X→Y)/ s(Y)
= 0.667 / 0.6 = 1.111

# 1. 연관성 분석(장바구니 분석)

## 연관성 분석 주요 측도

- 연관성 분석 기법의 주요 활용 분야

• 연관성 분석은 소매업 혹은 쇼핑몰 등의 거래데이터에서 물품 간에 연관관계를 파악하는 '장바구니 분석'에 가장 많이 사용되어 왔기 때문에 마케팅 영역에서는 연관성 분석 = 장바구니 분석으로 거의 동일한 용어로 사용되기도 한다. 그러나 연관성 분석이 비단 장바구니 분석에만 활용되는 것은 아니며, 객체(혹은 문서) 및 아이템 간의 연관 관계성 혹은 빈출 패턴을 파악할 필요가 있는 다양한 영역에서 활용되고 있다.

1. 쇼핑/유통 분야에서 구매된 물품 간의 장바구니 분석
2. 금융상품이나 서비스 간의 가입 패턴의 연관성 분석
3. 인터넷/모바일 서비스 상품 추천 시스템의 구매 항목 간 연관성 알고리즘으로 사용
4. 웹 페이지 간의 링크 관계성 분석 (웹 사용자의 행동 추적 및 패턴 예측)
5. 가맹점 간의 방문 연관성 및 순서 패턴 도출 통한 쿠폰 마케팅 등 활용
6. 생물 정보학 분야에서의 단백질과 유전자 패턴 분석
7. 신용카드 사기 및 보험청구 사기 등 부정거래 패턴 발견

• 즉, 방대한 데이터베이스나 인터넷 로그 데이터 등에서 아이템, 객체들 간의 발생 연관성을 파악하거나 발생 순서 간 패턴을 파악하려는 거의 모든 분야에서 활용 가능한 현실 세계의 실무 활용성이 높은 기법이라고 할 수 있다.

# 1. 연관성 분석(장바구니 분석)

## 연관성 분석 주요 측도

- 연관성 분석 알고리즘
- 연관성 분석의 대표적인 알고리즘에는 아프리오리(Apriori) 알고리즘과 이를 개선한 빈출 패턴 성장(FP-Growth) 알고리즘이 있다.

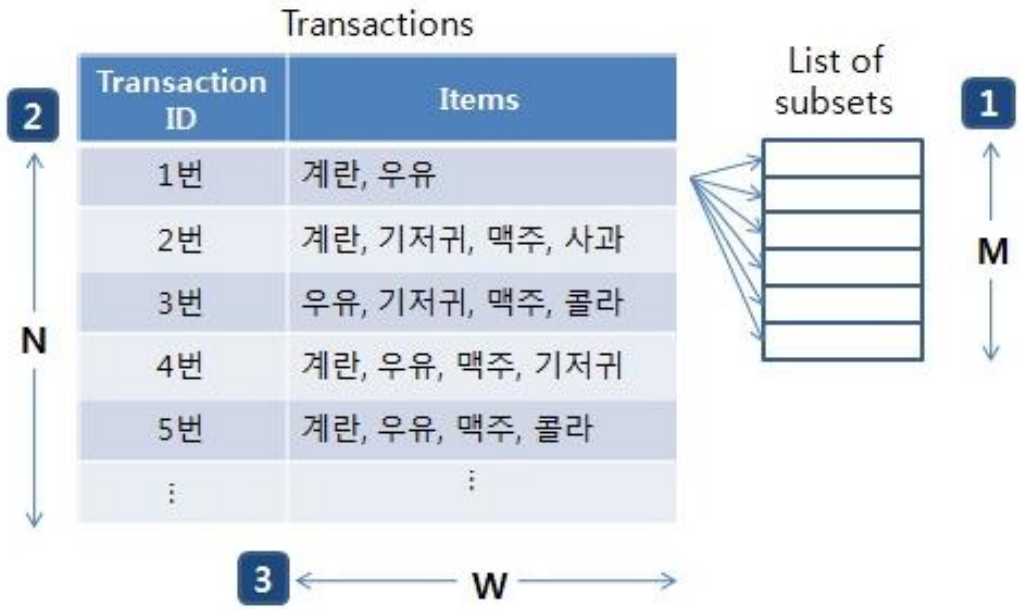
| 알고리즘 | 알고리즘 설명 |
|---|---|
| 아프리오리(Apriori) 알고리즘 | 빈출 아이템 집합을 효과적으로 계산하기 위하여 검토해야 할 대상 집합 Pool을 효과적으로 줄여주는 기법. 즉, 특정 집합 {1,2}가 빈발하지 않는다면, 이들을 포 함한 {0,1,2}, {1,2,3}, {1,2,4}, {0,1,2,3,4} 등도 빈발하지 않은 것이 되어, 지지도 계 산 검토 대상에서 제외할 수 있다. 이런 식으로 검토대상의 아이템 데이터 집합 을 효과적으로 줄여서 연관규칙을 계산해내는 기법. 데이터가 커질 경우 계산량이나 속도 면에서 비효율적이라는 단점이 있음 |
| 빈출패턴 성장 (FP-Growth) | 아프리오리 알고리즘의 빈발 아이템을 찾는 방법을 개선한 알고리즘으로서 데이 터베이스를 모든 중요한 정보를 가진 FP-Tree라는 구조로 압축한 뒤, 높은 빈도 의 세트에 관련한 조건부 데이터 세트로 분할하여 규칙을 만들어 낸다. 전체 데 이터베이스를 검색하는 작업이 아프리오리 알고리즘보다 현저하게 줄어들어서 높은 성능과 처리속도를 보이는 알고리즘임. 반면, 아프리오리 알고리즘보다 구 현이 어렵다는 단점이 있음. |
| DHP algorithm | Transaction 개수(N)을 줄이는 전략 |

# 1. 연관성 분석(장바구니 분석)

## 연관성 분석 주요 측도

- 연관성 분석 알고리즘

**Association Rule : strategy and algorithm**




1 모든 가능한 항목집합 개수(M)를 줄이는 방식 ➡ Apriori algorithm

2 Transaction 개수(N)를 줄이는 방식 ➡ DHP Algorithm

3 비교하는 수(W)를 줄이는 방식 ➡ FP-growth Algorithm

# 1. 연관성 분석(장바구니 분석)

## Apriori algorithm

- 최소지지도 이상을 갖는 항목집합을 빈발항목집합(frequent item set)이라고 합니다. 모든 항목집합에 대한 지지도를 계산하는 대신에 최소 지지도 이상의 빈발항목집합만을 찾아내서 연관규칙을 계산하는 것이 Apriori algorithm의 주요 내용입니다.
- 빈발항목집합 추출의 Apriori Principle

1. 한 항목집합이 빈발(frequent)하다면 이 항목집합의 모든 부분집합은 역시 빈발항목집합이다.(frequent item sets -> next step)
2. 한 항목집합이 비빈발(infrequent)하다면 이 항목집합을 포함하는 모든 집합은 비빈발항목집합이다. (superset -> pruning)

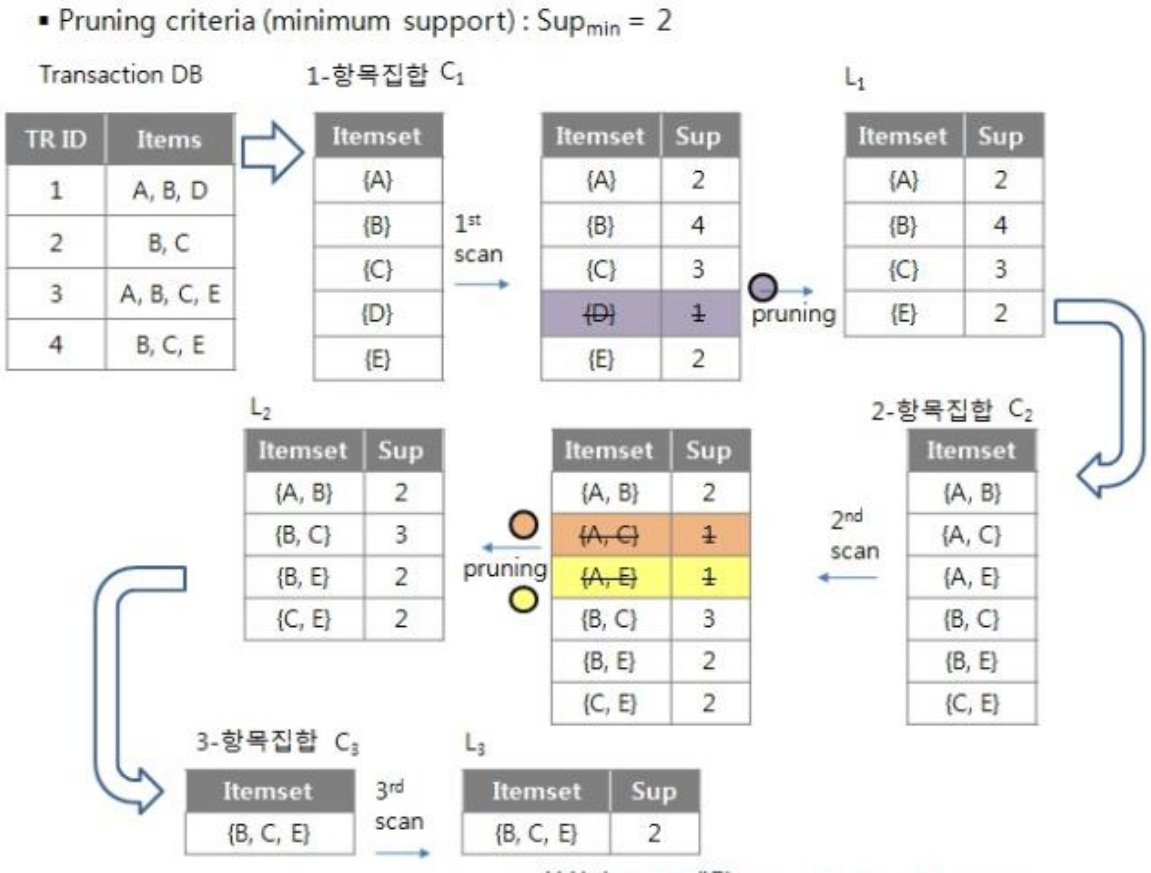
- Apriori Pruning Principle

If there is any itemset which is infrequent, its superset should not be generated/tested!
(Agrawal & Srikant @VLDB’94, Mannila, et al. @ KDD’ 94)

# 1. 연관성 분석(장바구니 분석)

## Apriori algorithm

- {A, B, C, D, E}의 5개 원소 항목을 가지는 4건의 transaction에서 minimum support 2건 (=2건/총4건=0.5) 기준으로 pruning 하는 예



# 1. 연관성 분석(장바구니 분석)

## Apriori algorithm

- 빈발항목집합 추출 Pseudo Aprior algorithm

1) $L_1 = \{\text{large 1-itemsets}\};$
2) for $(k = 2; L_{k-1} \neq 0; k++)$ do begin
3) $C_k = apriori\text{-}gen(L_{k-1})$; // New candidates
4) forall transactions $t \in T$ do begin
5) $C_t = subset(C_k, t)$; // Candidates contained in t
6) forall candidates $c \in C_t$ do
7) c.count ++;
8) end
9) $L_k = \{c \in C_k \mid c.count \geq \text{min sup}\}$
10) end
11) Answer $= \bigcup_k L_k$;

| k-itemset | [Notation] An itemset having k items |
|---|---|
| $L_k$ | Set of large k-itemsets (those with minimum support)<br>Each member of this set has two fields: i) itemset and ii) support count. |
| $C_k$ | Set of candidate k-itemsets (potentially large itemsets)<br>Each member of this set has two fields: i) itemset and ii) support count |
| $\overline{C}_k$ | Set of candidate k-itemsets when the TIDs of the generating transactions are kept associated with the candidates |

[Reference] "Fast Algorithms for Mining Association Rules", Rakesh Agrawal, Ramakrishman Srikant, 1994

# 1. 연관성 분석(장바구니 분석)

## Apriori 예

### 1. mlxtend 패키지 설치

```
pip install mlxtend
```

### 2. 패키지 로드

```
# 패키지 로드
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori
```

# 1. 연관성 분석(장바구니 분석)

## Apriori 예

### 3. 데이터셋 생성

```
# 데이터셋 생성
dataset=[['사과','치즈','생수'],
['생수','호두','치즈','고등어'],
['수박','사과','생수'],
['생수','호두','치즈','옥수수']]
dataset
```

>>

```
[['사과', '치즈', '생수'],
 ['생수', '호두', '치즈', '고등어'],
 ['수박', '사과', '생수'],
 ['생수', '호두', '치즈', '옥수수']]
```

# 1. 연관성 분석(장바구니 분석)

## Apriori 예

4. 가나다 순으로 Column값을 생성해서(고등어,사과,생수,수박,옥수수,치즈,호두) 첫번째 (사과,치즈.생수) 데이터에 고등어가 표시되어이있으면 True값으로 없으면 False값으로 표시한다. 다음 사과도 첫번째 데이터에 사과가 있으면 1값으로 없으면 0값으로 표시한다. 이러한 것을 반복하면 4X7행렬이 생성된다

```
te = TransactionEncoder()
te_ary = te.fit(dataset).transform(dataset)
df = pd.DataFrame(te_ary, columns=te.columns_) #위에서 나온걸 보기 좋게 데이터프레임으로 변경
df
>>
```

| | 고등어 | 사과 | 생수 | 수박 | 옥수수 | 치즈 | 호두 |
|---|---|---|---|---|---|---|---|
| 0 | False | True | True | False | False | True | False |
| 1 | True | False | True | False | False | True | True |
| 2 | False | True | True | True | False | False | False |
| 3 | False | False | True | False | True | True | True |

# 1. 연관성 분석(장바구니 분석)

## Apriori 예

**5. 지지도를 0.5로 놓고 apriori를 돌려보면 아래와 같이 결과가 출력된다. 사과를 살 확률 0.5, 치즈 등을 함께 살 확률 0.75**

```
frequent_itemsets = apriori(df, min_support=0.5, use_colnames=True)
frequent_itemsets
```

```
>>
```

| | support | itemsets |
|---|---|---|
| 0 | 0.50 | (사과) |
| 1 | 1.00 | (생수) |
| 2 | 0.75 | (치즈) |
| 3 | 0.50 | (호두) |
| 4 | 0.50 | (생수, 사과) |
| 5 | 0.75 | (치즈, 생수) |
| 6 | 0.50 | (호두, 생수) |
| 7 | 0.50 | (치즈, 호두) |
| 8 | 0.50 | (치즈, 호두, 생수) |

# 1. 연관성 분석(장바구니 분석)

## Apriori 예

### 6. 신뢰도가 0.3인 항목만 보기

```
from mlxtend.frequent_patterns import association_rules
association_rules(frequent_itemsets, metric="confidence", min_threshold=0.3
```

>>

| | antecedents | consequents | antecedent support | consequent support | support | confidence | lift | leverage | conviction | zhangs_metric |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | (생수) | (사과) | 1.00 | 0.50 | 0.500000 | 1.000000 | 1.000000 | 0.000 | 1.0 | 0.0 |
| 1 | (사과) | (생수) | 0.50 | 1.00 | 0.500000 | 1.000000 | 1.000000 | 0.000 | inf | 0.0 |
| 2 | (치즈) | (생수) | 0.75 | 1.00 | 0.750000 | 1.000000 | 1.000000 | 0.000 | inf | 0.0 |
| 3 | (생수) | (치즈) | 1.00 | 0.75 | 0.750000 | 0.750000 | 1.000000 | 0.000 | 1.0 | 0.0 |
| 4 | (호두) | (생수) | 0.50 | 1.00 | 0.500000 | 1.000000 | 1.000000 | 0.000 | inf | 0.0 |
| 5 | (생수) | (호두) | 1.00 | 0.50 | 0.500000 | 0.500000 | 1.000000 | 0.000 | 1.0 | 0.0 |
| 6 | (치즈) | (호두) | 0.75 | 0.50 | 0.500000 | 0.666667 | 1.333333 | 0.125 | 1.5 | 1.0 |
| 7 | (호두) | (치즈) | 0.50 | 0.75 | 0.500000 | 1.000000 | 1.333333 | 0.125 | inf | 0.5 |
| 8 | (치즈, 호두) | (생수) | 0.50 | 1.00 | 0.500000 | 1.000000 | 1.000000 | 0.000 | inf | 0.0 |
| 9 | (치즈, 생수) | (호두) | 0.75 | 0.50 | 0.500000 | 0.666667 | 1.333333 | 0.125 | 1.5 | 1.0 |

# 1. 연관성 분석(장바구니 분석)

## FP-Growth 예

```
pip install mlxtend –upgrade

from sklearn.preprocessing import MultiLabelBinarizer

data = [
    ['휴지', '물티슈', '샴푸'],
    ['수세미', '물티슈', '비누'],
    ['휴지', '수세미', '물티슈', '비누'],
    ['수세미', '비누']
]

mlb = MultiLabelBinarizer()
encoded_data = mlb.fit_transform(data)
df_encoded = pd.DataFrame(encoded_data, columns=mlb.classes_)

print(df_encoded)
```

```
>>
   물티슈  비누  샴푸  수세미  휴지
0    1   0   1    0   1
1    1   1   0    1   0
2    1   1   0    1   1
3    0   1   0    1   0
```

# 1. 연관성 분석(장바구니 분석)

## FP-Growth 예

```
from mlxtend.frequent_patterns import fpgrowth
fpgrowth(df_encoded, min_support=0.5, use_colnames=True)
>>
```

| | support | itemsets |
|---|---|---|
| 0 | 0.75 | (물티슈) |
| 1 | 0.50 | (휴지) |
| 2 | 0.75 | (수세미) |
| 3 | 0.75 | (비누) |
| 4 | 0.50 | (비누, 물티슈) |
| 5 | 0.50 | (물티슈, 수세미) |
| 6 | 0.50 | (비누, 물티슈, 수세미) |
| 7 | 0.50 | (휴지, 물티슈) |
| 8 | 0.75 | (비누, 수세미) |