인천 진산과학과 특강
4일차

Prof. Jibum Kim (jibumkim@inu.ac.kr)

Department of Computer Science & Engineering

Incheon National University

- https://github.com/DevSlem
- **Repositories 클릭**
- **Jinsan-AI 클릭 4일차**
- **경진대회 준비, 발표, 보고서 작성을 위하여 오픈 채팅방에 들어오세요**

- **경진대회**
- **1월 6일 ~ 1월 9일까지 경진대회 운영**
- **1월 10일에는 인천대에서 오전 9시50분 ~ 12시까지 각 팀 발표 예정**
- **오늘 팀 구성 요청 4~5인**
- **팀 구성 후 팀 이름 및 팀원이 누구인지 오픈카톡방을 통해 팀원 중에 한명이 알려주세요.**

- **결과 보고서**
- **1월 17일까지 개인별로 제출 요청. 발표 자료 기반으로 보고서 작성 요청**
- **분량: 본인 학번 및 이름을 적고 한글/워드로 2장 이내로 작성**
- **구체적인 내용은 오픈카톡방 통해서 공지 예정**

- **오늘 특강 내용**
- **1. DBSCAN 군집화**
- **2. Decision tree regression**
- **3. Boosting: AdaBoost 알고리즘**
- **4. 특징 선택: RFE 알고리즘**
- **5. 실제 타이타닉 데이터셋에 대한 전처리, ML 모델 학습, ML 모델 예측 분석**

특강 내용

DBSCAN 군집화 (비지도학습)

- 군집화에서 가장 유명한 알고리즘은 지난시간에 K-means 알고리즘이다.
- 다른 유명한 군집화 알고리즘에는 DBSCAN이 있다. DBSCAN은 (Density-based spatial clustering of applications with noise)이다.
- 이는 밀도기반 clustering으로 데이터의 밀도에 따라 cluster를 형성하는 기법이다.

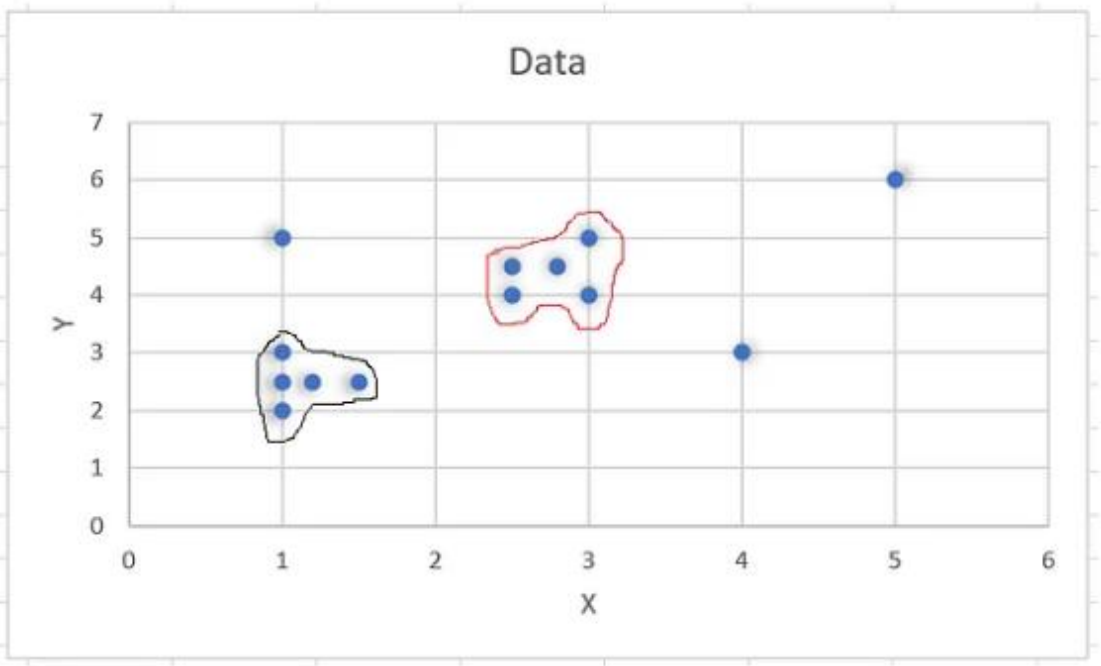
- This algorithm defines clusters as continuous regions of high density. Here is how it works:
- 1. For each instance, the algorithm counts how many instances are located within a small distance ϵ (epsilon) from it. This region is called the instances's ϵ-neighborhood
- 2. If an instance has at least min_samples instances in its ϵ-neighborhood (including itself), then it is considered a core instance. In other words, core instances are those that are located in dense regions

- **3. All instances in the neighborhood of a core instance belong to the same cluster. This neighborhood may include other core instances: therefore, a long sequence of neighboring core instances forms a single cluster.**

- **4. Any instance that is not a core instance that does not have one in its neighborhood is considered an anomaly (이상치)**

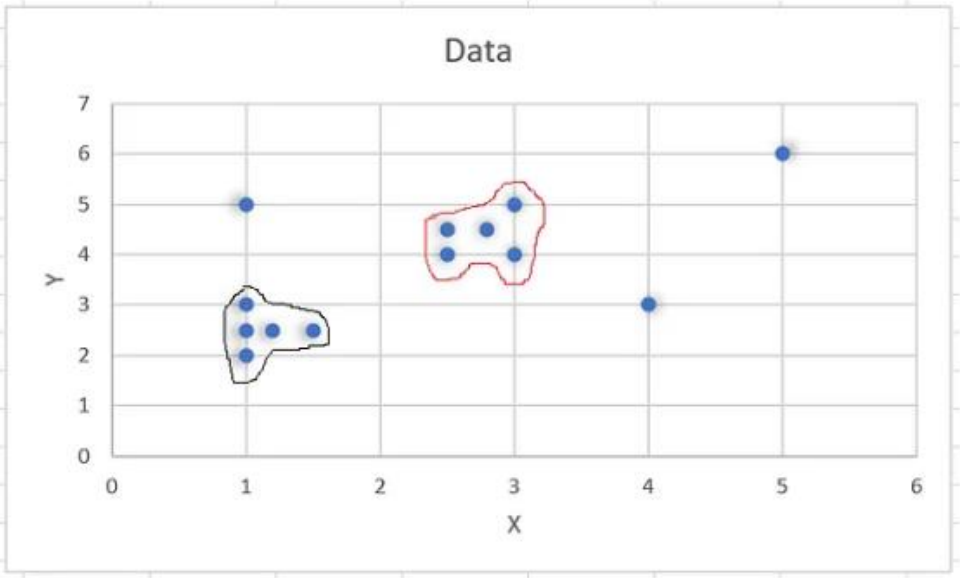
- 예: 총 13개의 데이터. DBSCAN 알고리즘을 사용하여 군집화
- 2개의 cluster 기대. (1, 5), (4, 3), (5, 6)은 outlier

X	Y
1	2
3	4
2.5	4
1.5	2.5
3	5
2.8	4.5
2.5	4.5
1.2	2.5
1	3
1	5
1	2.5
5	6
4	3



- DBSCAN의 중요한 2가지 파라미터
- 1. small distance ϵ (epsilon) 2. min_samples
- 예: $\epsilon = 0.6$, min_samples=4로 설정
- 첫 번째 점 (1, 2)에 대해서 다른 점까지의 거리 계산 (오른쪽 표)

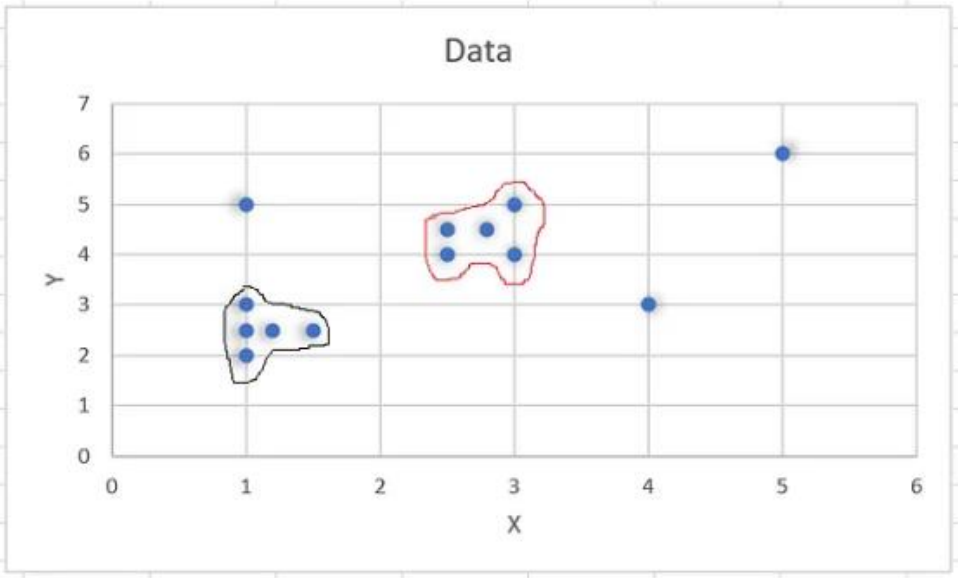
X	Y
1	2
3	4
2.5	4
1.5	2.5
3	5
2.8	4.5
2.5	4.5
1.2	2.5
1	3
1	5
1	2.5
5	6
4	3



X	Y	Distance from (1,2)
1	2	0
3	4	2.8
2.5	4	2.5
1.5	2.5	0.7
3	5	3.6
2.8	4.5	3.08
2.5	4.5	2.9
1.2	2.5	0.53
1	3	1
1	5	3
1	2.5	0.5
5	6	5.6
4	3	3.1

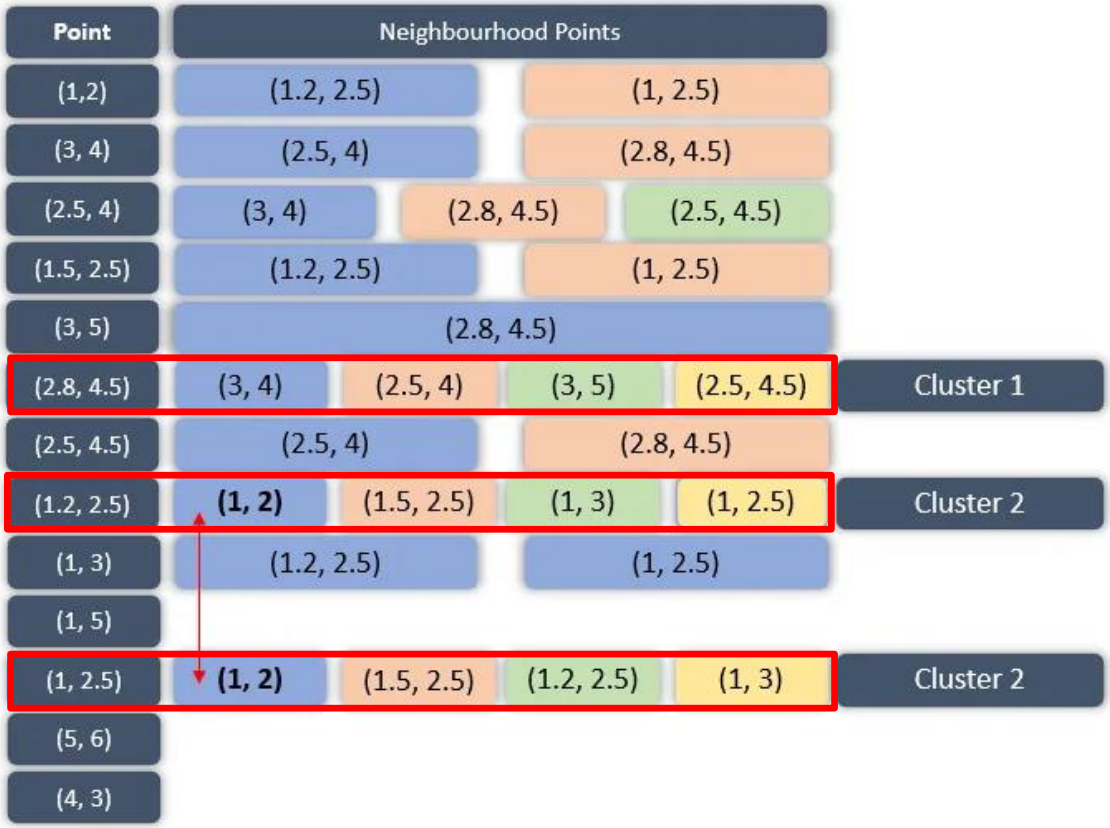
- 먼저, (1, 2)에서 $\epsilon = 0.6$ 이내의 거리에 위치하는 점을 찾는다. 2개의 점 (1, 2.5), (1.2, 2.5)
- 하지만 아직 core point (instance)는 아님
- Why? min_samples인 4보다 작아서

X	Y
1	2
3	4
2.5	4
1.5	2.5
3	5
2.8	4.5
2.5	4.5
1.2	2.5
1	3
1	5
1	2.5
5	6
4	3



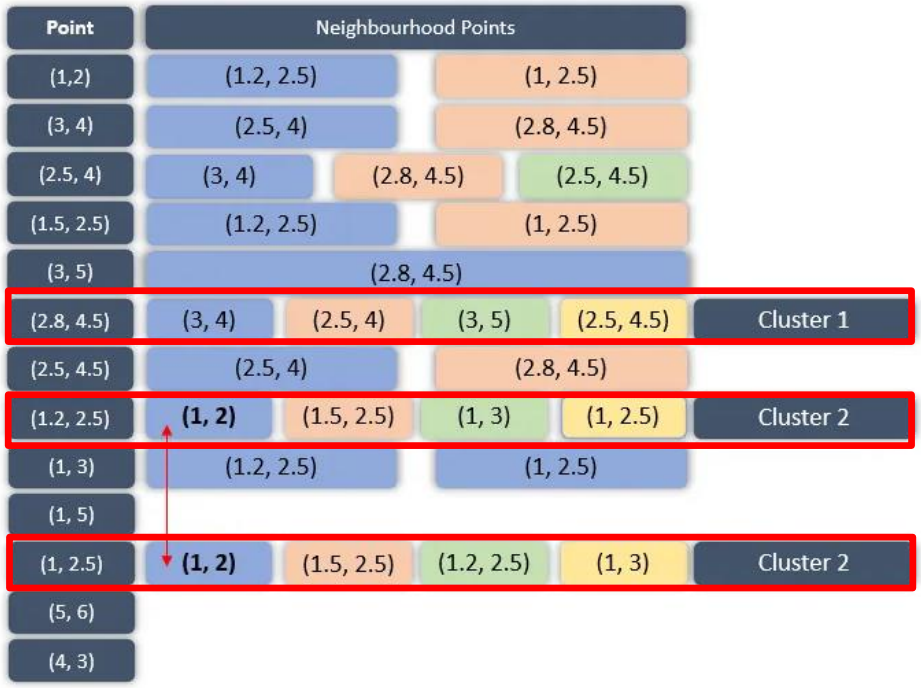
X	Y	Distance from (1,2)
1	2	0
3	4	2.8
2.5	4	2.5
1.5	2.5	0.7
3	5	3.6
2.8	4.5	3.08
2.5	4.5	2.9
1.2	2.5	0.53
1	3	1
1	5	3
1	2.5	0.5
5	6	5.6
4	3	3.1

- 모든 점에 대해서 반복하여 eps보다 같거나 작은 점들을 표시
- 표를 보면 총 3개의 점. (2.8, 4.5), (1.2, 2.5), (1, 2.5)이 eps 이내에 4개의 점 (min_samples보다 크거나 같은)이 있음. 즉, core point

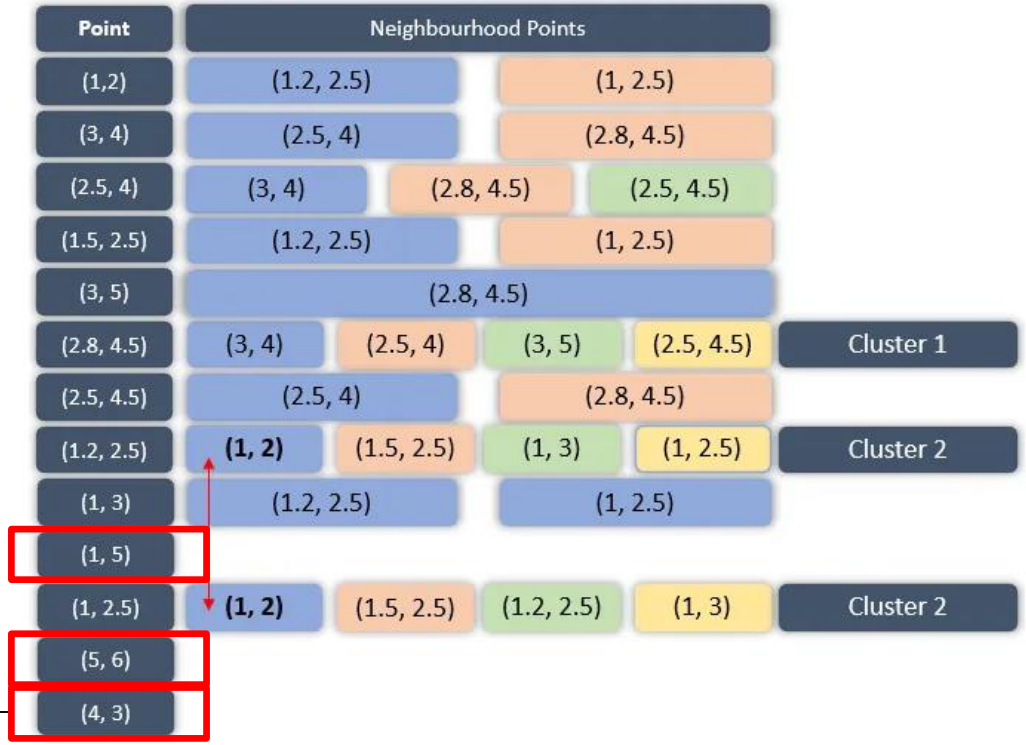


Point	Neighbourhood Points				
(1,2)	(1.2, 2.5)	(1, 2.5)			
(3, 4)	(2.5, 4)	(2.8, 4.5)			
(2.5, 4)	(3, 4)	(2.8, 4.5)	(2.5, 4.5)		
(1.5, 2.5)	(1.2, 2.5)	(1, 2.5)			
(3, 5)	(2.8, 4.5)				
(2.8, 4.5)	(3, 4)	(2.5, 4)	(3, 5)	(2.5, 4.5)	Cluster 1
(2.5, 4.5)	(2.5, 4)	(2.8, 4.5)			
(1.2, 2.5)	**(1, 2)**	(1.5, 2.5)	(1, 3)	(1, 2.5)	Cluster 2
(1, 3)	(1.2, 2.5)	(1, 2.5)			
(1, 5)					
(1, 2.5)	**(1, 2)**	(1.5, 2.5)	(1.2, 2.5)	(1, 3)	Cluster 2
(5, 6)					
(4, 3)					

- Core point가 어떤 cluster에도 할당이 안됨, 새로운 cluster 만들어짐
- 따라서, (2.8, 4.5)에는 새로운 cluster 할당 (cluster 1)
- (1.2, 2.5)에도 새로운 cluster 할당 (cluster 2)
- (1, 2.5)와 (1.2, 2.5)는 하나의 공통된 이웃점 (1, 2) 공유. 같은 cluster로 배정



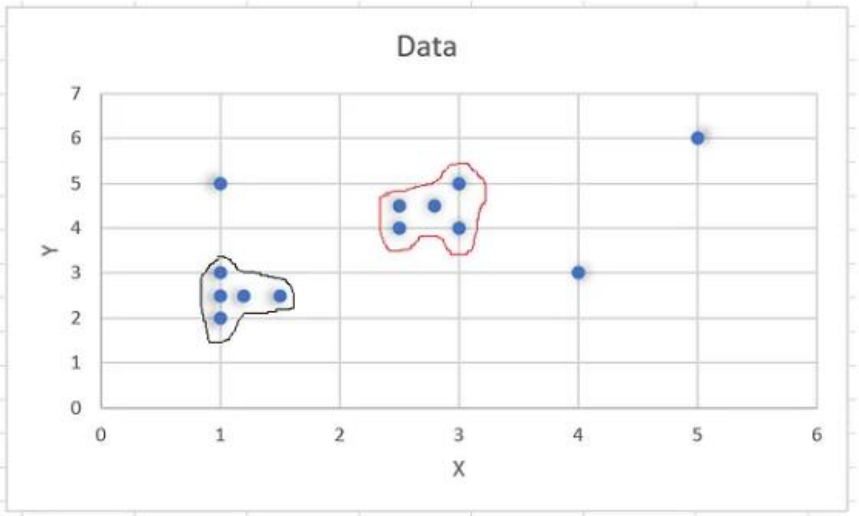
- 즉, 최종 cluster는 cluster 1과 cluster 2로 2개
- Outlier: 세 점 (2.8, 4.5), (1.2, 2.5), (1, 2.5)가 아니거나 이웃 점으로 없는 점은 outlier. 즉, **(1, 5), (5, 6), (4,3)**



- **DBSCAN 결과**

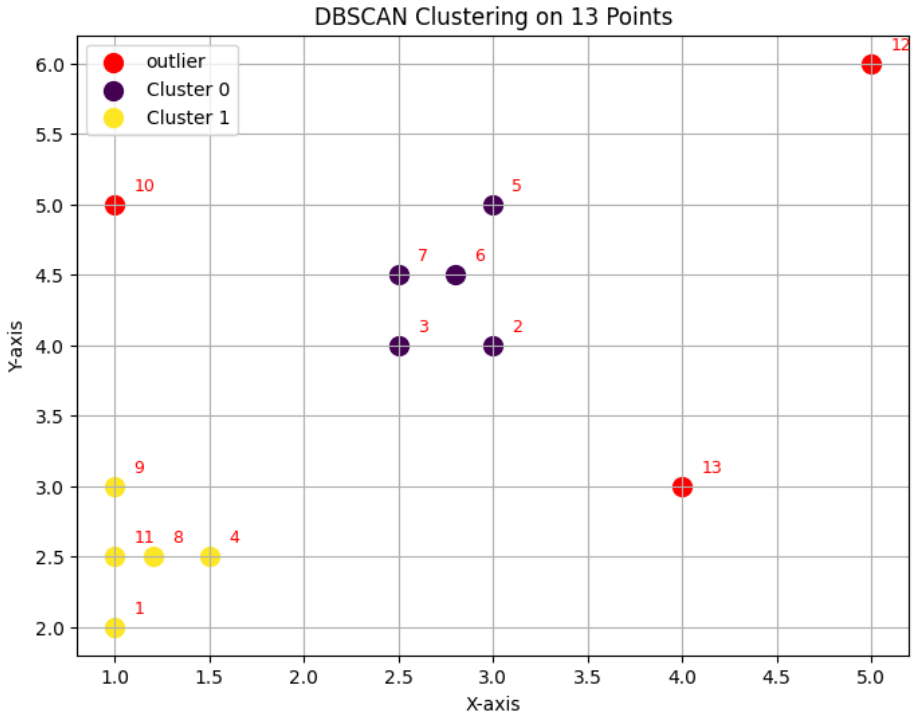
Cluster 1	Cluster 2	Outliers
(3,4)	(1, 2)	(1, 5)
(2.5, 4)	(1.5, 2.5)	(5, 6)
(3,5)	(1.2, 2.5)	(4, 3)
(2.8, 4.5)	(1, 3)	
(2.5, 4.5)	(1, 2.5)	

X	Y
1	2
3	4
2.5	4
1.5	2.5
3	5
2.8	4.5
2.5	4.5
1.2	2.5
1	3
1	5
1	2.5
5	6
4	3



실습:

```
# DBSCAN 알고리즘 적용 (eps: 반경, min_samples: 최소 샘플 수)
dbscan = DBSCAN(eps=0.6, min_samples=4)
labels = dbscan.fit_predict(points)
```

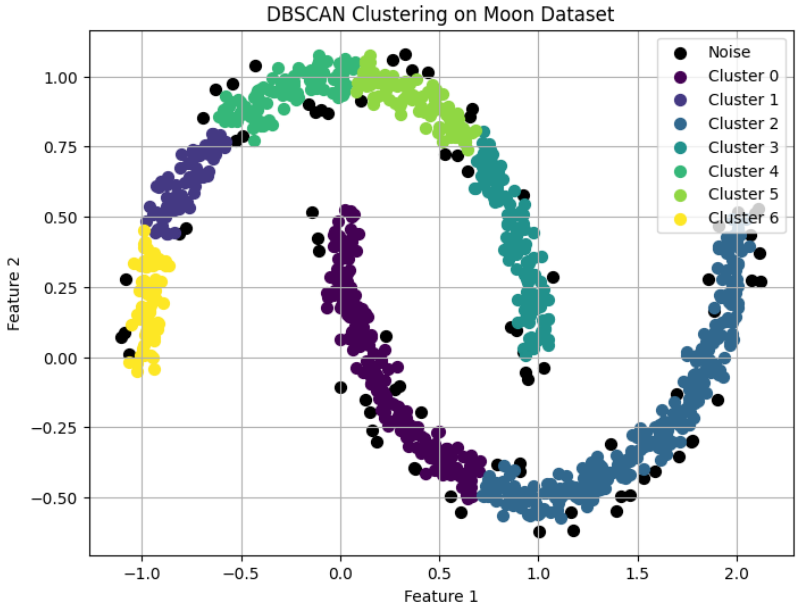


- **DBSCAN 알고리즘의 장점**
- **1. cluster 수를 미리 정의할 필요가 없음**
- **2. 임의의 모양을 가진 cluster를 탐지할 수 있음**
- **3. 이상치 (outlier)를 효과적으로 처리**
- **4. 밀도가 다른 cluster를 잘 처리**

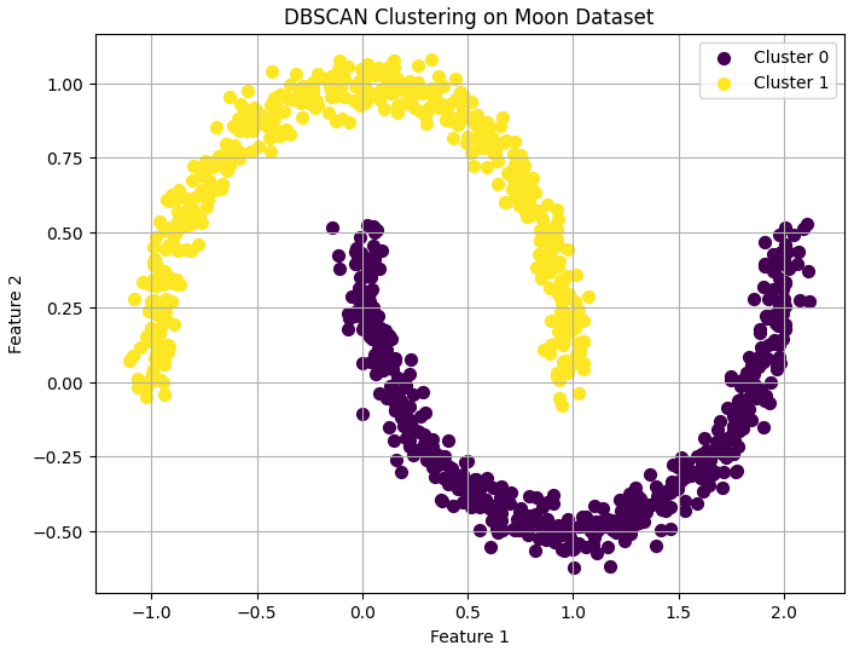
- **DBSCAN 알고리즘의 단점**
- **eps와 min_samples 설정이 까다로울 수 있음**

- **K-means**
  - 주어진 데이터셋을 `k` 개의 클러스터로 나눔.
  - 각 클러스터는 중심(centroid)으로 정의되고, 데이터를 가장 가까운 중심으로 할당.
  - 중심을 반복적으로 업데이트하여 클러스터링 수행.
  - 데이터가 구형(구조적으로 동그란 형태)이고, 클러스터 간 크기와 밀도가 유사한 경우 잘 작동.
- **DBSCAN (Density-Based Spatial Clustering of Applications with Noise)**
  - 데이터의 밀도(density)에 기반하여 클러스터를 정의.
  - 특정 반경(ε) 내에 최소 포인트(minPts) 이상의 데이터가 존재하면 하나의 클러스터로 간주.
  - 밀집되지 않은 데이터는 노이즈로 간주.
  - 다양한 밀도를 가진 클러스터와 복잡한 모양의 클러스터링에 적합.

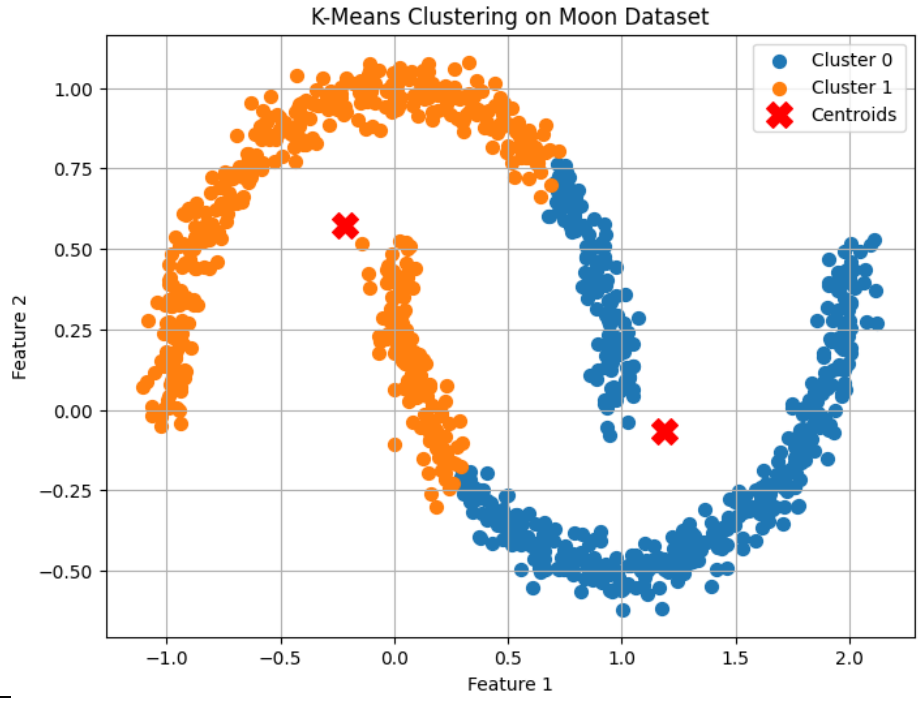
- 실습:
- moon dataset에 대하여 DBSCAN의 2개의 파라미터 값에 대하여 비교
- 1. dbscan = DBSCAN(eps=0.05, min_samples=5)
- How disappointing! 이웃을 포함하는 반경을 너무 작게 작음



- **2.**
- **dbscan = DBSCAN(eps=0.2, min_samples=5)**
- **Looks perfect!**



- **3. K-means clustering 결과. K=2**
- **Centroid 표시. 좋은 결과? 나쁜 결과?**



- **실습 예제**

특강 내용

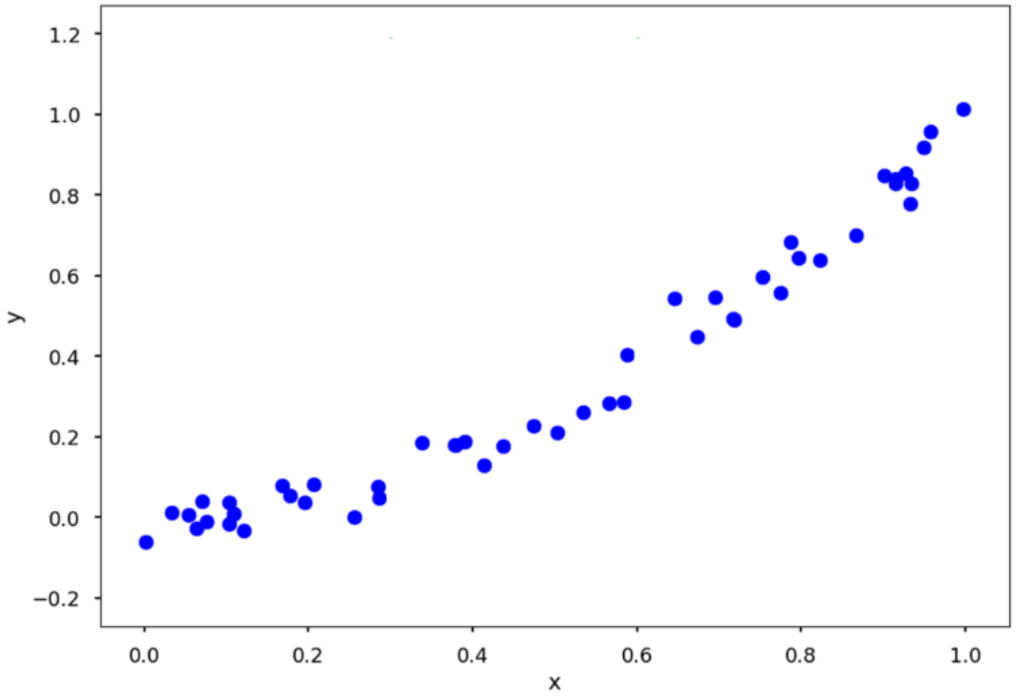
Decision tree regression

- Decision trees are also capable of performing regression tasks

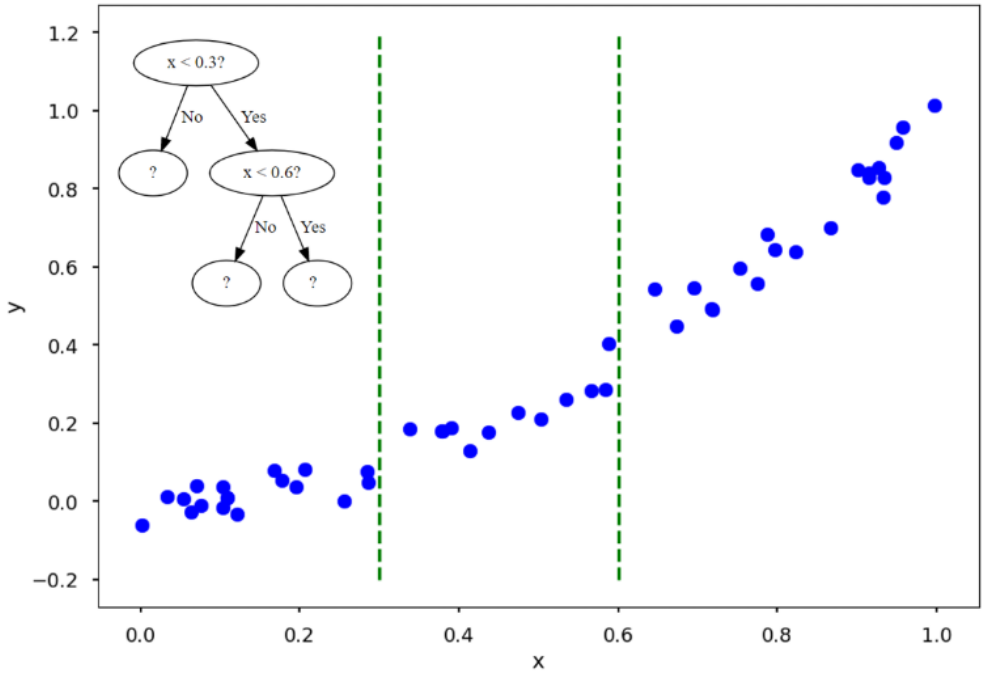
- The CART algorithm works mostly the same as earlier, except that instead of trying to split the training set in a way that minimizes Gini impurity (지니 불순도),
- it now tries to split the training set in a way that minimizes the MSE

- 회귀문제에서는 각 분할에서 MSE를 최소화하도록 분할

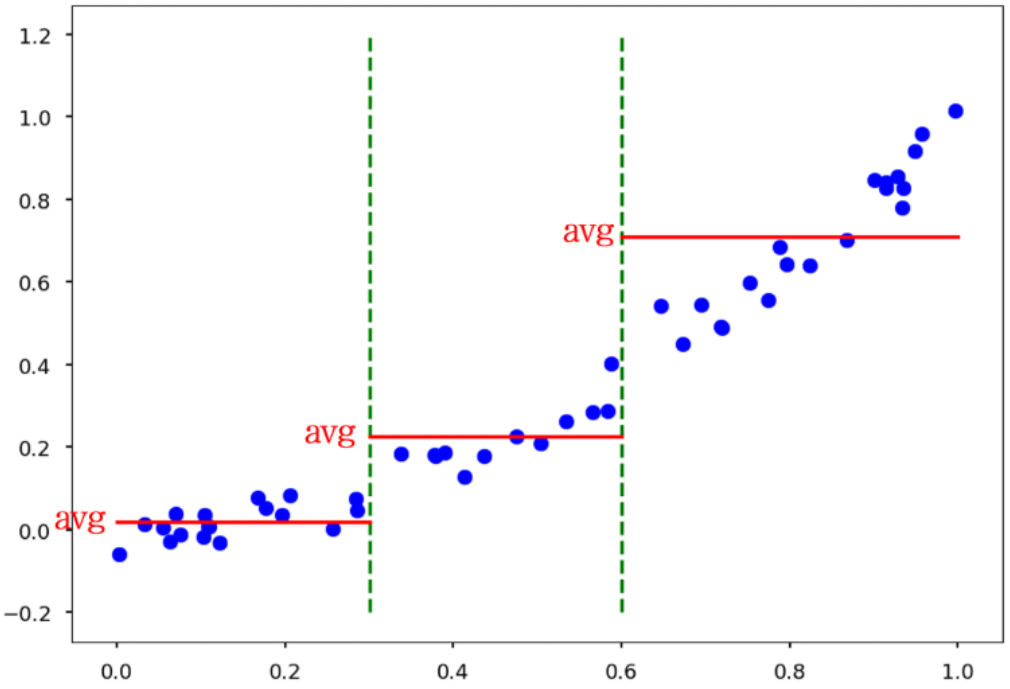
- 목표: 아래와 같은 분포를 가진 데이터에 대하여 decision tree regression 모델을 학습하고 임의의 x값에 대하여 y값 예측



- Decision tree에서 아래 그림처럼
- 첫번째 질문 $x < 0.3$
- 두번째 질문 $x > 0.6$을 사용하면
- 데이터가 아래 초록색 점선처럼 영역을 나눠짐



- 각 구간안의 x값에 대해서 훈련 데이터의 평균값 (빨간 실선)을 모델의 예측값으로 출력
- 예: x=0이면 그것이 속한 영역의 평균값인 0.02정도로 y값을 예측
- 오차: x=0일때의 실제 y값 (-0.02)과 예측값 0.02의 차이=> 0.04



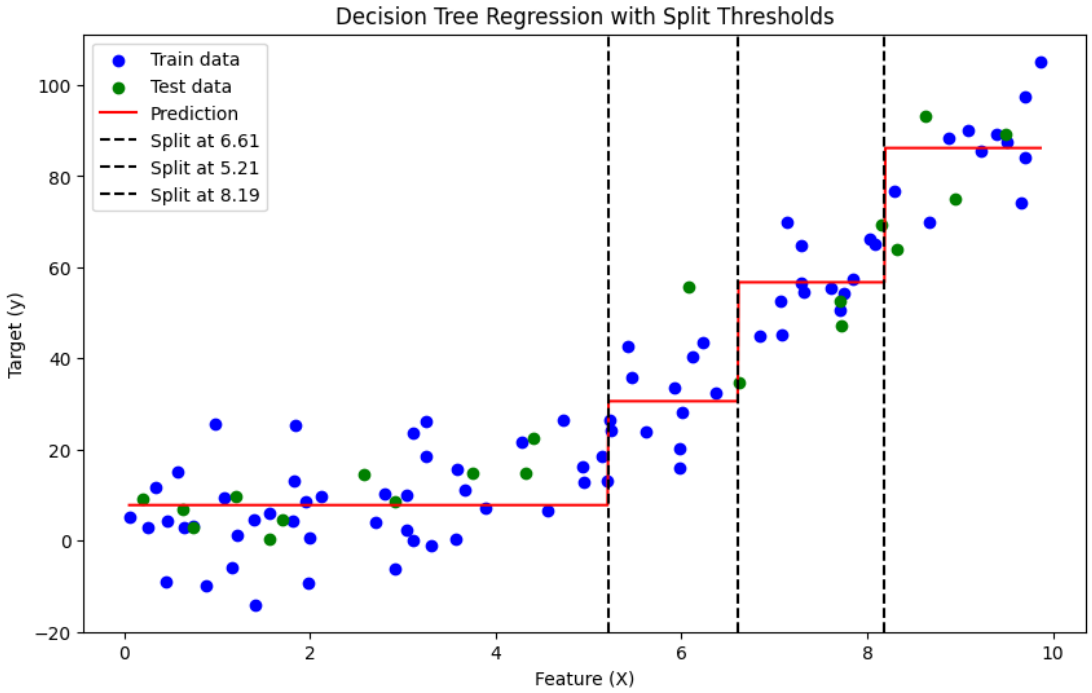
- 실제 값과 예측 값의 오차는 MSE와 같은 값으로 나타낼 수 있음

- Decision tree regression에서의 CART 알고리즘

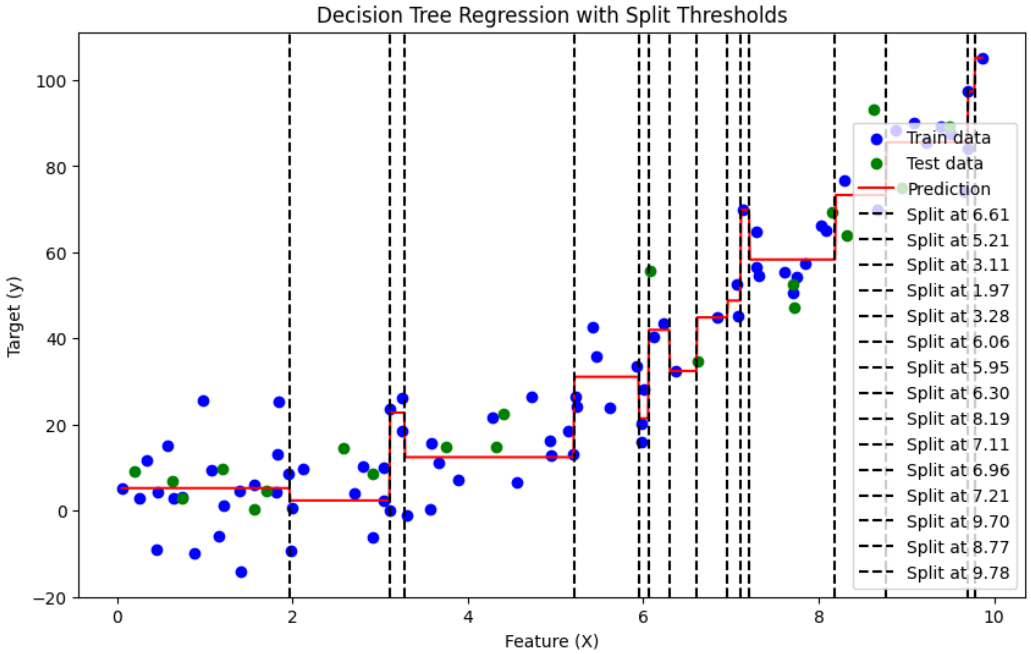
$$J(k, t_k) = \frac{m_{left}}{m} MSE_{left} + \frac{m_{right}}{m} MSE_{right}$$

- 좌/우 가지에 해당하는 데이터의 비율을 가중치로 곱하여 전체 오차값 $J(k, t_k)$를 계산하여 최적의 분기를 결정

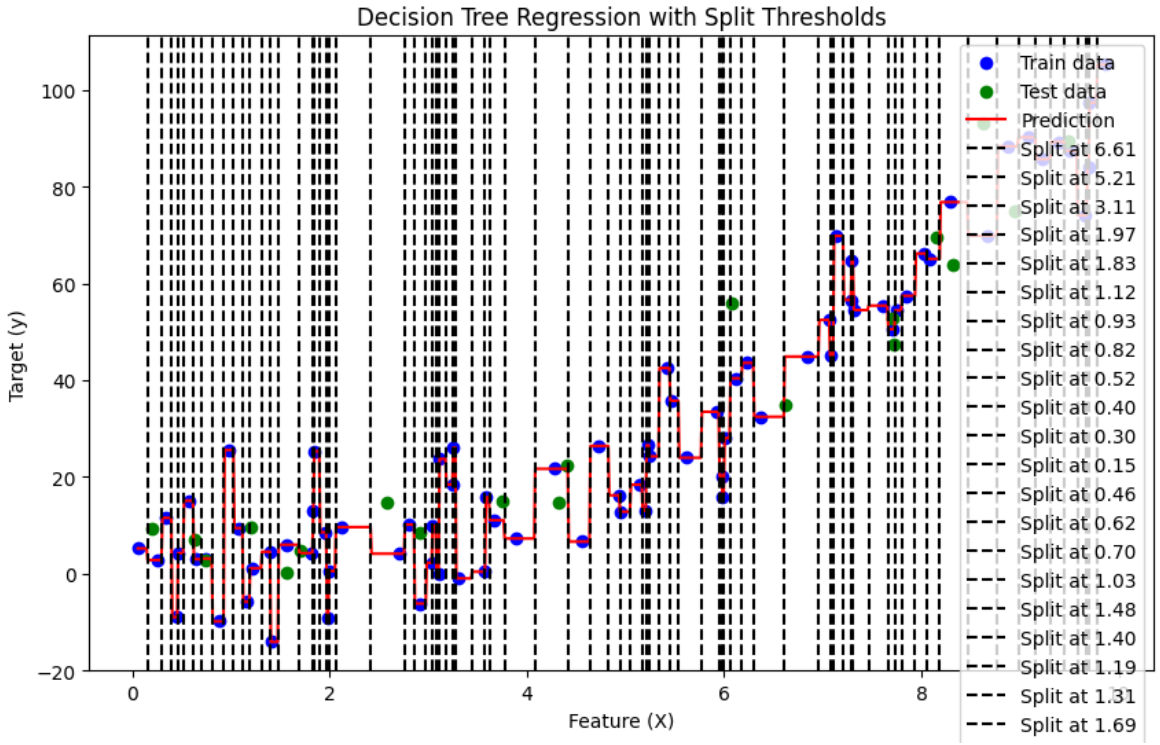
- 실습: $y = x^2$에 noise를 추가하여 가상의 학습/테스트 데이터 생성
- Train data로 max_depth=2인 decision tree regression으로 학습
- 결정 경계 (3개): 점선, 예측 값: 빨간색.
- 테스트 데이터 예측 예: x=1.70일 때 y=7.78로 예측



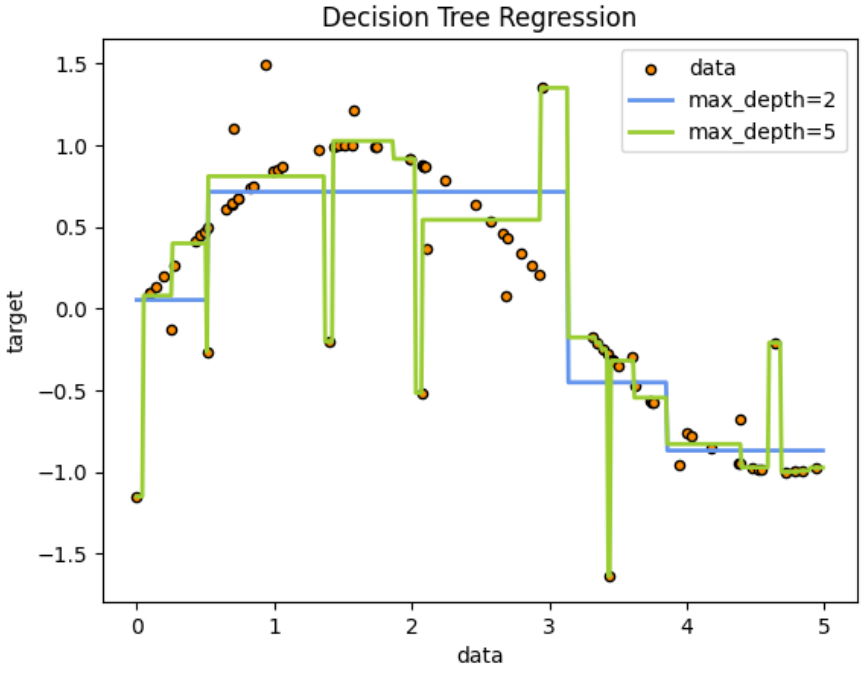
- max_depth=4로 주었을 경우
- Tree의 depth가 깊어질 수록, 데이터를 나누는 점선이 촘촘해지고 예측 값과 실제 값의 오차는 줄어든다
- 하지만, overfitting되기가 쉽다



- **Regularization 없음. Overfitting**



- **(실습) We can see that if the maximum depth of the tree (controlled by the max_depth parameter) is set too high, the decision trees learn too fine details of the training data and learn from the noise, i.e. they overfit**



- Decision tree regression에서 overfitting을 줄이는 방법
- Decision tree classifier와 유사
- 1. tree의 최대 깊이 제한 (max_depth)
- 2. leaf node의 최대 샘플 수 제한 (min_samples_leaf)
- 3. 내부 노드 분할을 위한 최소 샘플 수 제한 (min_samples_split)
- 등
- Decision Tree Regression — scikit-learn 1.5.2 documentation

- **Decision tree regression에서도 GridSearchCV를 사용하여 조절 가능한 하이퍼 파라미터 최적화 가능**
- **실습**

```
# 하이퍼파라미터 그리드 정의
param_grid = {
    'max_depth': [3, 5, 7, 10, None],  # 트리 깊이
    'min_samples_split': [2, 5, 10],  # 분할을 위한 최소 샘플 수
    'min_samples_leaf': [1, 2, 4],    # 리프 노드의 최소 샘플 수
    'max_features': [1, 2, None]      # 분할에 사용할 최대 특성 수
}
```

특강 내용

분류기 성능 지표

- **분류기 성능 지표에는 정확도와 confusion matrix외에 precision, recall, F1-score, 등 많은 지표가 있다**
- **특히 class imbalanced dataset에서는 정확도 대신에 precision, recall, F1-score 등이 널리 쓰인다**

2. 예시: 스팸 이메일 필터링

- **문제**: 이메일이 스팸인지 아닌지 분류하는 문제에서, 전체 이메일 중 대부분은 정상 이메일이고, 소수의 이메일만 스팸입니다.
  - 정상 이메일: 95%
  - 스팸 이메일: 5%
- **정확도(Accuracy)**를 기준으로 모델을 평가한다고 가정해봅시다. 만약 모델이 모든 이메일을 정상(negative)으로 예측한다면, 정확도는 95%로 나옵니다. 하지만 **이 모델은 스팸 이메일을 전혀 걸러내지 못**하는 것입니다.

- **Confusion 행렬에서 TP/FP/FN/TN 의미**

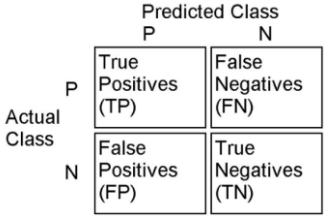
		Predicted Class: P	Predicted Class: N
Actual Class	P	True Positives (TP)	False Negatives (FN)
	N	False Positives (FP)	True Negatives (TN)

- 코로나 19 검사 예
- True positive (TP): 실제 병에 걸렸는데 모델이 양성으로 예측
- True negative (TN): 실제 병에 걸리지 않았는데 모델이 음성으로 예측
- False positive (FP,위양성): 실제 병에 걸리지 않았는데 모델이 양성으로 잘못 예측
- False negative (FN): 실제 병에 걸렸는데 모델이 음성으로 잘못 예측

- 코로나19에서 위양성은 실제로는 감염되지 않은 사람을 잘못된 검사 결과로 코로나 19에 감염된 것으로 판정
- 이유: 검사 키트, 기술적 오류 등

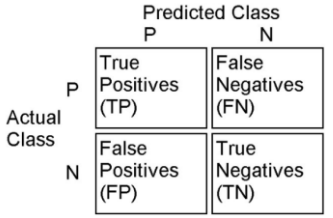
- **Classification 성능 지표: 각 class에 대해서 다음을 계산한다**
- **1. 정밀도 (precision)**
- **정의: 해당 클래스에 대해 모델이 양성이라고 예측한 샘플 중 실제 양성의 비율**

$$\text{Precision} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Positives (FP)}}$$



- **2. 재현율 (recall), 민감도 (sensitivity)**
- **정의: 해당 클래스에 대해 실제 양성 샘플 중에서 모델이 올바르게 예측한 비율**

$$\text{Recall} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Negatives (FN)}}$$



- 코로나19 자가검사키트 5품목, 차이는 '민감도' < 친절한 데이터 < 식약처 < 정책 < 기사본문 - 히트뉴스

- **3. F1-score**
- **정의: Precision과 Recall의 조화 평균**

$$\text{F1-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \qquad \text{산술 평균} = \frac{\sum_{i=1}^{n} x_i}{n} \qquad \text{조화 평균} = \frac{n}{\sum_{i=1}^{n} \frac{1}{x_i}}$$

- **조화 평균은 데이터의 역수를 기반으로 작은 값에 더 큰 가중치 부여**

정밀도와 재현율 중 어느 한 쪽이 높고 다른 쪽이 낮으면, 산술 평균을 사용했을 때 한쪽 값이 과대평가될 위험이 있습니다.

- 예를 들어, Precision = 1.0, Recall = 0.1인 경우:
  - 산술 평균: $\frac{1.0+0.1}{2} = 0.55$
  - F1 (조화 평균): $2 \cdot \frac{1.0 \cdot 0.1}{1.0+0.1} = 0.18$

- **4. support:** 각 class에 포함된 샘플 데이터 수
- **5. Weighted avg:** class별 샘플 수를 고려한 가중치 평균
- **6. Macro avg:** 모든 class의 단순 평균

```
Best Parameters: {'estimator': DecisionTreeClassifier(max_depth=1), 'learning_rate': 0.1, 'n_estimators': 200}
Test Accuracy: 0.93

Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       0.90      0.90      0.90        10
   virginica       0.90      0.90      0.90        10

    accuracy                           0.93        30
   macro avg       0.93      0.93      0.93        30
weighted avg       0.93      0.93      0.93        30
```

3개의 클래스: **Class A, Class B, Class C**

Confusion Matrix:

Predicted/Actual	Class A	Class B	Class C
Class A	40	5	5
Class B	2	30	8
Class C	1	4	35

행: 실제 클래스

열: 예측된 클래스

Class A:

- True Positives (TP): 40
- False Positives (FP): $5 + 5 = 10$ (Class B와 C로 잘못 예측된 Class A 샘플)
- False Negatives (FN): $2 + 1 = 3$ (Class A가 다른 클래스(Class B, C)로 잘못 예측됨)

$$\text{Precision (A)} = \frac{TP}{TP + FP} = \frac{40}{40 + 10} = 0.8$$

$$\text{Recall (A)} = \frac{TP}{TP + FN} = \frac{40}{40 + 3} \approx 0.93$$

$$\text{F1-Score (A)} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \approx 2 \cdot \frac{0.8 \cdot 0.93}{0.8 + 0.93} \approx 0.86$$

Class B:

- True Positives (TP): 30
- False Positives (FP): $5 + 4 = 9$ (Class A와 C로 잘못 예측된 Class B 샘플)
- False Negatives (FN): $5 + 8 = 13$ (Class B가 다른 클래스(Class A, C)로 잘못 예측됨)

$$\text{Precision (B)} = \frac{TP}{TP + FP} = \frac{30}{30 + 9} \approx 0.77$$

$$\text{Recall (B)} = \frac{TP}{TP + FN} = \frac{30}{30 + 13} \approx 0.70$$

$$\text{F1-Score (B)} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \approx 2 \cdot \frac{0.77 \cdot 0.70}{0.77 + 0.70} \approx 0.73$$

Class C:

- True Positives (TP): 35
- False Positives (FP): $5 + 8 = 13$ (Class A와 B로 잘못 예측된 Class C 샘플)
- False Negatives (FN): $5 + 4 = 9$ (Class C가 다른 클래스(Class A, B)로 잘못 예측됨)

$$\text{Precision (C)} = \frac{TP}{TP + FP} = \frac{35}{35 + 13} \approx 0.73$$

$$\text{Recall (C)} = \frac{TP}{TP + FN} = \frac{35}{35 + 9} \approx 0.80$$

$$\text{F1-Score (C)} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \approx 2 \cdot \frac{0.73 \cdot 0.80}{0.73 + 0.80} \approx 0.76$$

1. **Macro Average**:
   - 단순 평균:

$$\text{Macro Precision} = \frac{0.8 + 0.77 + 0.73}{3} \approx 0.77$$

$$\text{Macro Recall} = \frac{0.93 + 0.70 + 0.80}{3} \approx 0.81$$

$$\text{Macro F1-Score} = \frac{0.86 + 0.73 + 0.76}{3} \approx 0.78$$

2. **Weighted Average**:
   - 각 클래스의 샘플 수를 고려한 가중 평균: (Support: Class A = 43, Class B = 43, Class C = 44)

$$\text{Weighted Precision} = \frac{43 \cdot 0.8 + 43 \cdot 0.77 + 44 \cdot 0.73}{43 + 43 + 44} \approx 0.77$$

$$\text{Weighted Recall} = \frac{43 \cdot 0.93 + 43 \cdot 0.70 + 44 \cdot 0.80}{43 + 43 + 44} \approx 0.81$$

$$\text{Weighted F1-Score} = \frac{43 \cdot 0.86 + 43 \cdot 0.73 + 44 \cdot 0.76}{43 + 43 + 44} \approx 0.78$$

특강 내용

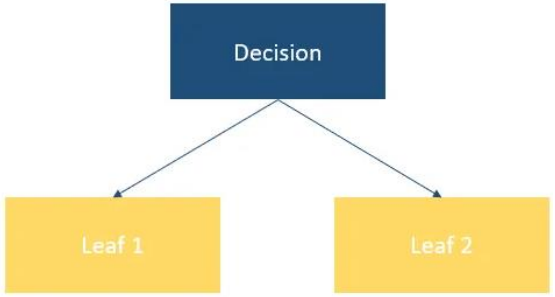
Boosting: AdaBoost

참고: tyami's study blog

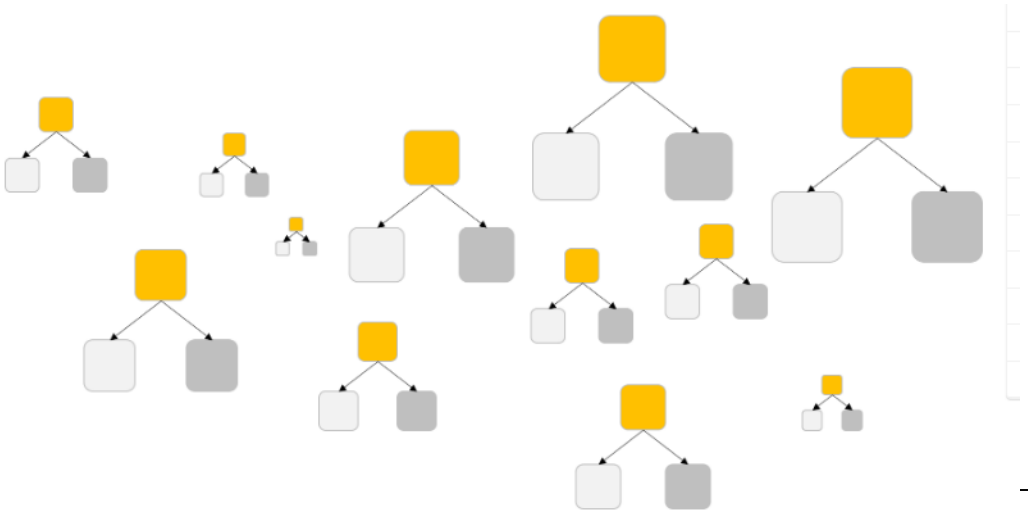
참고: AdaBoost, Clearly Explained

- **Boosting refers to any Ensemble method that can combine several weak learners into a strong learner**

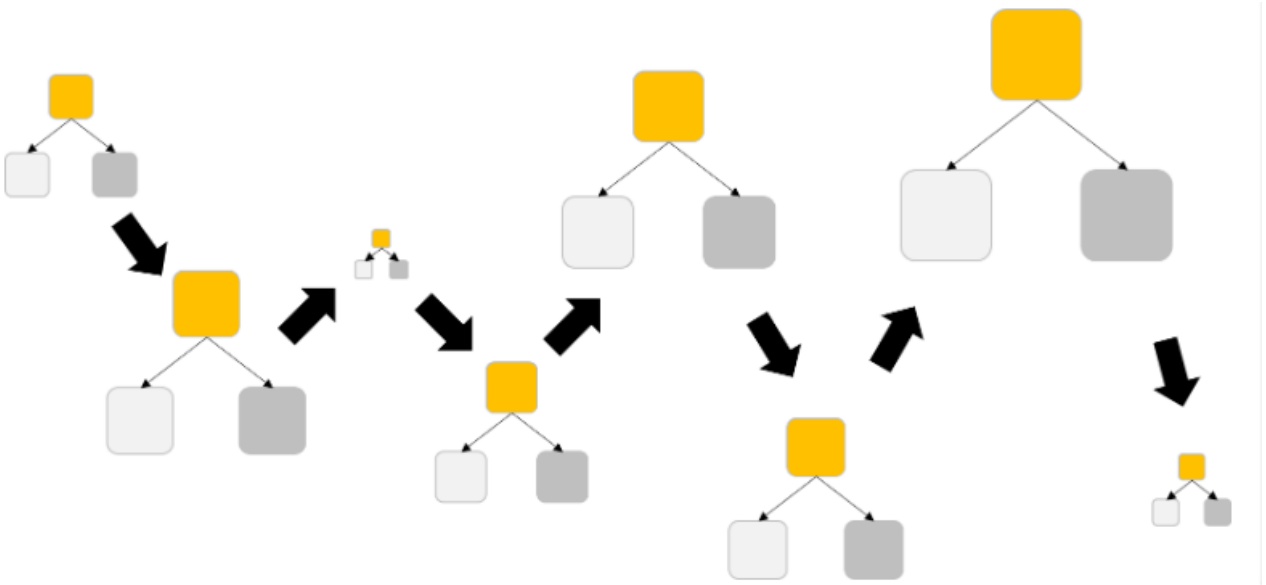
- **Weak learner? 약한 학습기**
- **이진 분류기에서 랜덤 분류기의 정확도가 50%라면 weak learner는 정확도가 50% 이상인 ML 모델로 단순하기 때문에 학습 속도가 빠르고, 계산 비용이 적음**
- **예: 깊이가 1인 decision tree가 대표적인 예**
- **이를 stump라고 부른다**



- Random forest 분류기에서 모든 tree는 동일한 가중치를 갖는다
- AdaBoost에서는 아래와 같이 개별 모델로 기본적으로 깊이가 1인 decision tree인 stump를 사용하고 각 stump는 서로 다른 weight (amount of say)를 갖는다
- 최종 ensemble (결합)시 예측 결과값에 가중치를 다르게 부여함
- 이유: 올바르게 분류하는 stump에는 높은 가중치를 주기 위함이다



- Random forest와 달리 AdaBoost에서 stump들은 순서대로 sequential하게 만들어진다
- 즉, 기존에 만들어진 stump의 결과가 이후 만들어질 stump 생성에 영향을 준다



- AdaBoost 분류기에서는
- 샘플 데이터와 각 분류기 모두 가중치를 갖고 있다.
- 샘플 데이터의 가중치 ($w_{i,t}$), 분류기의 가중치 (amount of say, α_t)
- 1. 각 stump에서 잘못 분류한 샘플 데이터에 가중치 ($w_{i,t}$)를 상대적으로 크게 하고 제대로 분류한 샘플 데이터는 상대적으로 가중치 ($w_{i,t}$)를 작게 한다.
- 가중치가 상대적으로 커진 샘플 데이터는 다음 stump에서 학습 시에 학습에 더 중요하게 반영 한다.
- 가중치를 고려하여 Gini 불순도를 최소화. 즉, 가중치가 커진 샘플 데이터를 더 올바르게 분류하도록 stump 만듬 (개선?)

- **2. 특정 stump의 분류기 성능에 따라서 그 분류기의 가중치 (amount of say)를 준다.**
- **각 stump에서는 얼마나 그 stump가 분류기의 오류가 큰지 오류율을 계산한다**
- **오류율이 작은 stump 는 amount of say가 상대적으로 커진다.**
- **무작위 분류기 정도의 성능을 보이는 stump는 amount of say를 0으로 (무시? 제외?)**
- **반대로, 오류율이 큰 stump는 amount of say를 음의 값으로 줘서 반대의 class로 예측하도록 한다**

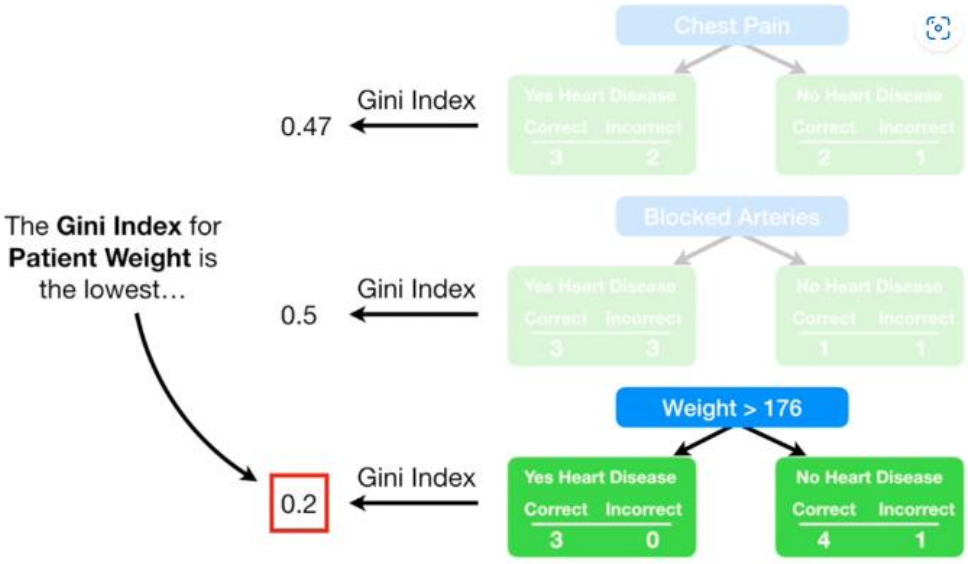
- **AdaBoost 학습 예**
- **총 8개의 학습 데이터, 3개의 특징, class (심장병 있음 +1, 없음 -1)**
- **1. 최초 각 sample의 weight는 동일하게 1/8로 부여**

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8

$$Initial\ sample\ weight = w_{i,1} = \frac{1}{total\ number\ of\ samples}$$

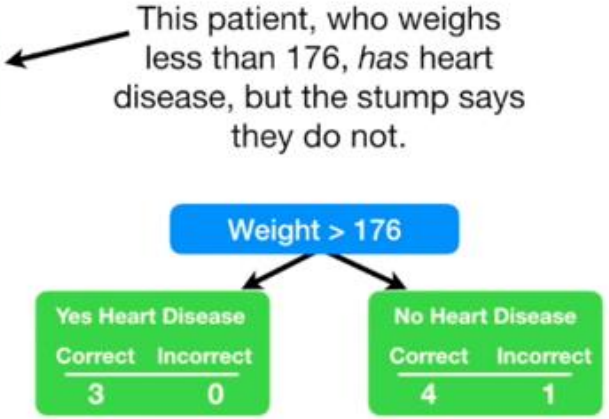
- **2. 최초 stump**
- **Patient weight 특징을 사용하고 weight >176으로 기준을 정하면 Gini 불순도가 0.2로 가장 크게 낮아짐**

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8

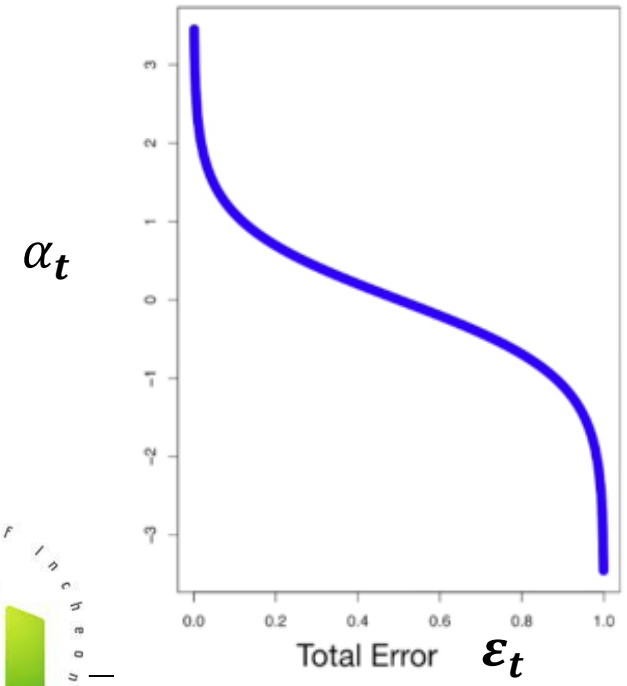


- **3. Amount of say (이 stump의 가중치)**
- **완성된 stump의 분류 결과에 따라 amount of say α_t 를 계산.**
- **Amount of say= $\alpha_t = \frac{1}{2} log\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$.**
- **ε_t: 잘못 분류된 샘플의 weight 총합, 예에서 $\varepsilon_t = \sum_{i=1}^{n} w_{i,t} = 1/8$**

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8



- Amount of say= $\alpha_t = \frac{1}{2} log\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$.
- ε_t: 잘못 분류된 샘플의 weight 총합
- $\varepsilon_t = 1/8$로 작은 편이다. 이 경우 이 stump의 가중치 $\alpha_t = 0.97$



ε_t <0.5 이면 α_t는 양의 값 (높은 신뢰)

ε_t=0.5 (랜덤 분류기)이면 α_t=0 (무시)

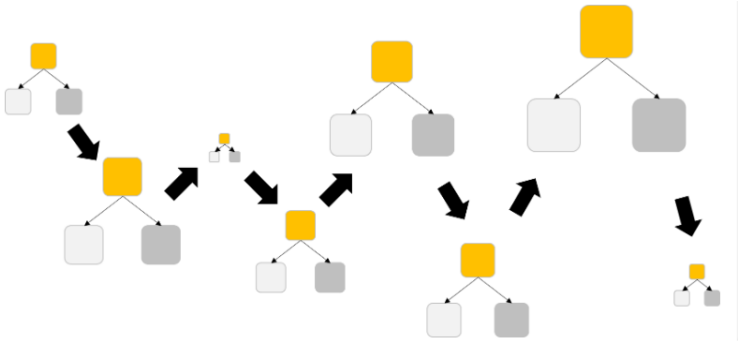
$\varepsilon_t > 0.5$ 이면 α_t는 음의값

- **4. 샘플 weight 업데이트**
- **잘못 분류한 샘플의 가중치가 커지고 나머지 샘플들은 가중치 감소**
- **제대로 분류한 샘플** $w_{i,t+1} = w_{i,t} \times e^{-\alpha_t}$ **(가중치 상대적 감소)**
- **잘못 분류한 샘플** $w_{i,t+1} = w_{i,t} \times e^{\alpha_t}$ **(가중치 상대적 증가)**
- **그 이후에 sample weight의 총합이 1이 되도록 정규화**

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight	New Weight	Norm. Weight
Yes	Yes	205	Yes	1/8	0.05	0.07
No	Yes	180	Yes	1/8	0.05	0.07
Yes	No	210	Yes	1/8	0.05	0.07
Yes	Yes	167	Yes	1/8	0.33	0.49
No	Yes	156	No	1/8	0.05	0.07
No	Yes	125	No	1/8	0.05	0.07
Yes	No	168	No	1/8	0.05	0.07
Yes	Yes	172	No	1/8	0.05	0.07

- **5. 다음 stump 생성**
- **업데이트된 sample weight를 적용한 새로운 stump 만듬**
- **가중치가 상대적으로 커진 샘플 데이터는 다음 stump에서 학습 시에 학습에 더 중요하게 반영 한다**
- **가중치를 고려하여 Gini 불순도를 최소화.**
- **즉, 가중치가 커진 샘플 데이터를 올바르게 분류하도록 stump 만듬**

- **6. Repetition**
- 정해진 반복 횟수까지 앞의 3에서 5까지 과정 반복하여 stump 계속 생성 (최초 설정한 stump 개수까지)



- **7. 최종 분류 예측**
- 모든 stump의 예측값 $h_t(x)$ (-1 또는 1)에 amount of say α_t 를 가중치로 곱하여 부호로 최종 예측. 부호가 양수면 +1로 예측. 부호가 음수면 -1로 예측

$$F(x) = \sum_{t=1}^{N} \alpha_t h_t(x)$$

- **클래스 1**: +1
- **클래스 2**: -1

이진 분류에서 AdaBoost는 여러 개의 "stump"(간단한 결정 트리 분류기)를 사용하여 최종 예측을 생성합니다.

Step 1: 첫 번째 stump 예측

- 첫 번째 stump가 예측한 값: +1 (class 1 예측)
- 이 stump의 가중치(학습된 중요도): 0.6

Step 2: 두 번째 stump 예측

- 두 번째 stump가 예측한 값: -1 (class 2 예측)
- 이 stump의 가중치: 0.4

Step 3: 최종 예측 계산

최종 예측은 각 stump의 예측값에 가중치를 곱한 후 모두 더한 값으로 결정됩니다.

$$\text{최종 예측} = (0.6 \times +1) + (0.4 \times -1) = 0.6 - 0.4 = 0.2$$

- 이 값이 **양수**이므로, 최종 예측은 **+1 (class 1)**로 분류됩니다.

- Adaboost의 중요 파라미터
- 1. base_estimator: weak learner 정의. 기본적으로 decision tree. 각 decision tree의 max_depth 조절 가능

estimator : *object, default=None*

The base estimator from which the boosted ensemble is built. Support for sample weighting is required, as well as proper `classes_` and `n_classes_` attributes. If `None`, then the base estimator is `DecisionTreeClassifier` initialized with `max_depth=1`.

- 2. n_estimators : 사용할 weak learner의 개수
- 큰 값을 사용하면 더 많은 weak learner 사용하여 복잡한 패턴 학습. Overfitting 위험. 작은 값은 underfitting 위험
- => GridSearchCV 사용하여 최적의 파라미터 찾을 수 있음
- 3. learning_rate: 각 weak learner의 가중치 기여도를 조절하는 파라미터로 amount of say에 곱해지는 값. 기본값은 1. 이 값이 작아지면 모델이 천천히 학습.

- **실습: random dataset에 대하여 AdaBoost를 사용하여 classification tuning 예**

```
param_grid = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.1, 0.5, 1.0],
    'estimator': [DecisionTreeClassifier(max_depth=1), DecisionTreeClassifier(max_depth=3)]
}
```

```
# 최적의 파라미터 출력
print("Best Parameters:", clf.best_params_)
```

```
Best Parameters: {'estimator': DecisionTreeClassifier(max_depth=3), 'learning_rate': 0.5, 'n_estimators': 200}
```

- **실습: Iris dataset에 대해서도 Adaboost를 사용하고 최적의 파라미터 tuning**

```
Best Parameters: {'estimator': DecisionTreeClassifier(max_depth=1), 'learning_rate': 0.1, 'n_estimators': 200}
Test Accuracy: 0.93

Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       0.90      0.90      0.90        10
   virginica       0.90      0.90      0.90        10

    accuracy                           0.93        30
   macro avg       0.93      0.93      0.93        30
weighted avg       0.93      0.93      0.93        30
```

- Adaboost는 tuning할 파라미터가 적고, overfitting에도 강인하다고 알려져 있다. 또한, weak learner의 결합 덕에 outlier에도 비교적 강인하다고 알려져 있다

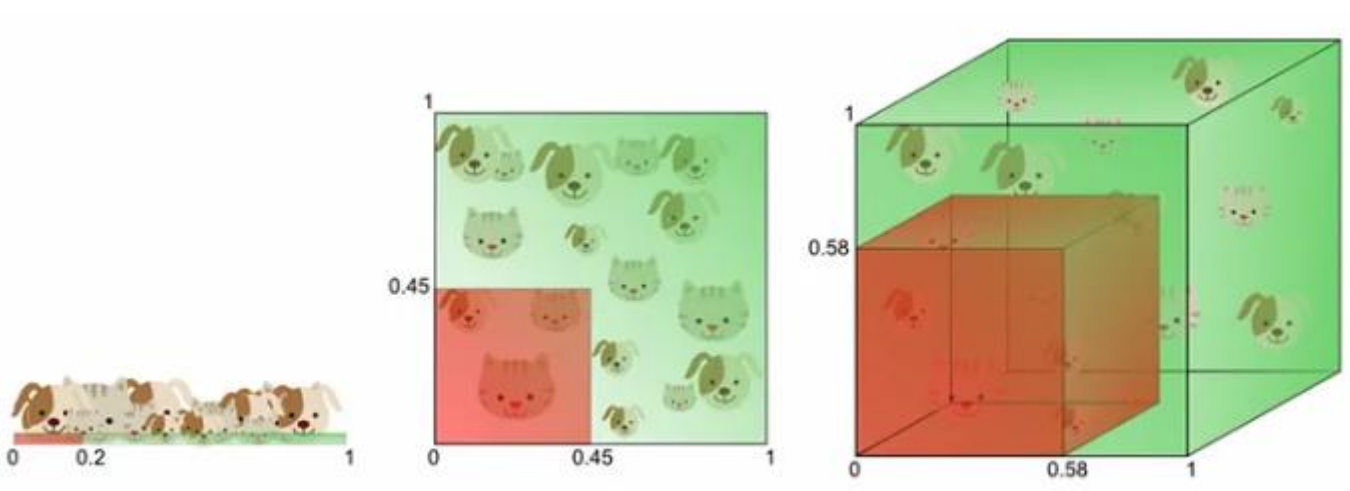
- 하지만, 대규모로 큰 dataset에는 학습 시간이 길어질 수 있고 weak learner에 의존하므로 weak learner를 어떻게 설계하는지가 중요하다. 이는 파라미터로 조정 가능하다.

특강 내용

Feature selection

- **Feature selection**
- ML모델에서 학습시에 사용하는 features (특징들) 중에서 중요한 특징들만 선택하는 과정이다
- 이를 통해 ML 모델의 성능 (예: 정확도)을 개선
- 주로, 학습 데이터 수에 비해서 특징의 수가 많을 때 사용
- 전처리로 ML 모델을 학습하기 전에 사용한다
- 대표적으로 RFE (recursive feature elimination) 방법이 있다

- **차원의 저주:** 데이터의 차원 (특징의 개수)이 증가할수록 데이터 분석과 ML 모델 학습이 어려워짐
- 고차원 공간에서는 데이터가 서로 멀리 떨어지고 학습 데이터에 over-fitting될 가능성이 높음
- RFE는 차원의 저주를 완화할 수 있는 한 방법



curse of dimensionality

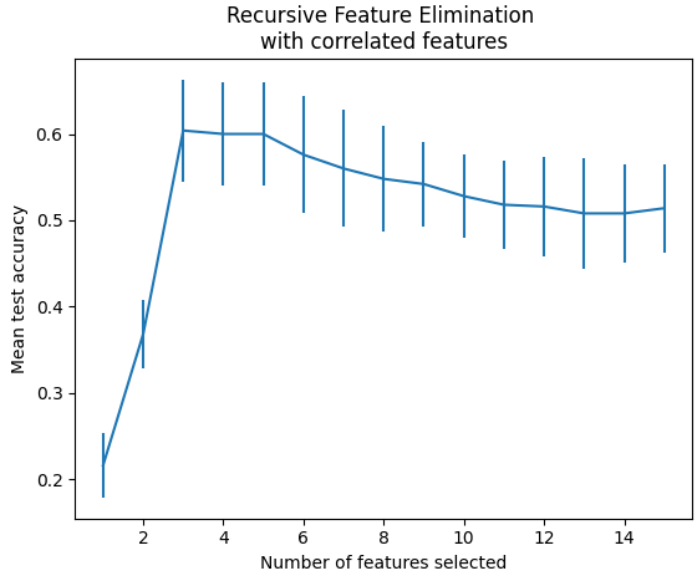
- **RFE의 작동 원리**
- **1. 전체 특징을 사용하여 ML 학습 모델을 만듬**
- **2. 특징의 중요도 평가 (예: 트리 기반 모델에서는 feature importance)**
- **3. 최소 중요 특징 제거: 중요도가 가장 낮은 특징을 하나 혹은 여러 개 제거**
- **4. 반복: 남은 특징을 사용하여 ML 모델을 다시 학습하고 위 과정을 반복**
- **5. 최종 특징 선택: 사전 정의된 목표에 도달하거나 성능이 더 이상 개선되지 않을 때 종료**

- **Scikit-learn의 ‘RFE’ 함수를 가면 몇 개의 특징을 선택할지 파라미터로 줄 수 있다**

n_features_to_select : *int or float, default=None*

The number of features to select. If `None`, half of the features are selected. If integer, the parameter is the absolute number of features to select. If float between 0 and 1, it is the fraction of features to select.

- **무작위 데이터에 대하여 RFE 적용 예. 특징 수에 따른 정확도 그래프. Optimal 특징 수?**

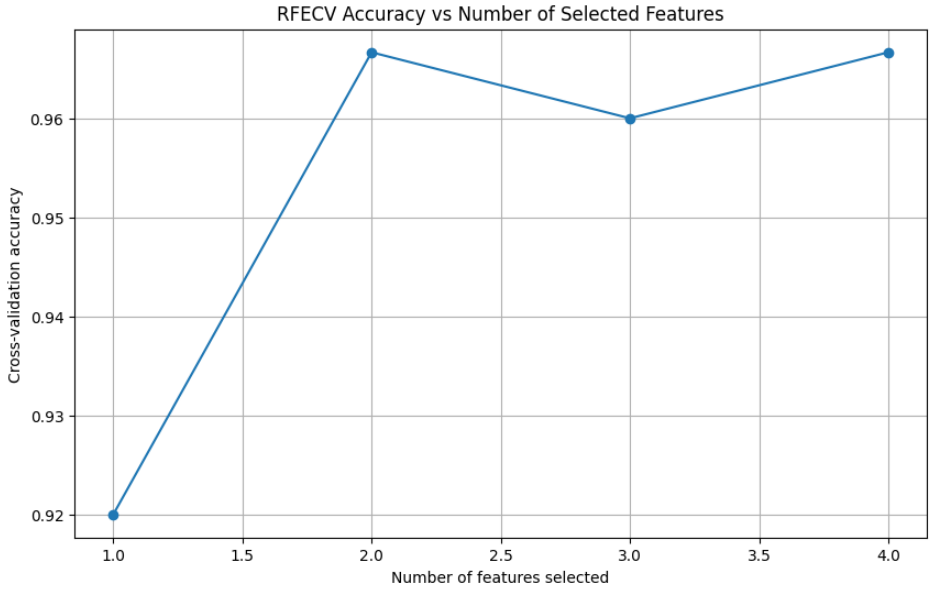


- Scikit-learn에는 ‘RFECV’라는 함수가 있다
- 이는 RFE와 Cross-validation을 결합한 함수로 교차 검증을 통하여 최적의 특징 개수를 자동으로 선택

- 실습: iris dataset에 대하여 random forest 분류기를 사용하여 ‘RFECV’ 적용 하여 최적의 특징 개수를 찾은 예

```
Optimal number of features: 2
Selected features: [False False  True  True]
Feature ranking: [2 3 1 1]
```

```python
# 최적의 특징 개수와 선택된 특징 출력
print(f"Optimal number of features: {rfecv.n_features_}")
print("Selected features:", rfecv.support_)
print("Feature ranking:", rfecv.ranking_)
```



```
Selected features: ['petal length (cm)', 'petal width (cm)']
Feature: sepal length (cm), Selected: False, Rank: 2
Feature: sepal width (cm), Selected: False, Rank: 3
Feature: petal length (cm), Selected: True, Rank: 1
Feature: petal width (cm), Selected: True, Rank: 1
```

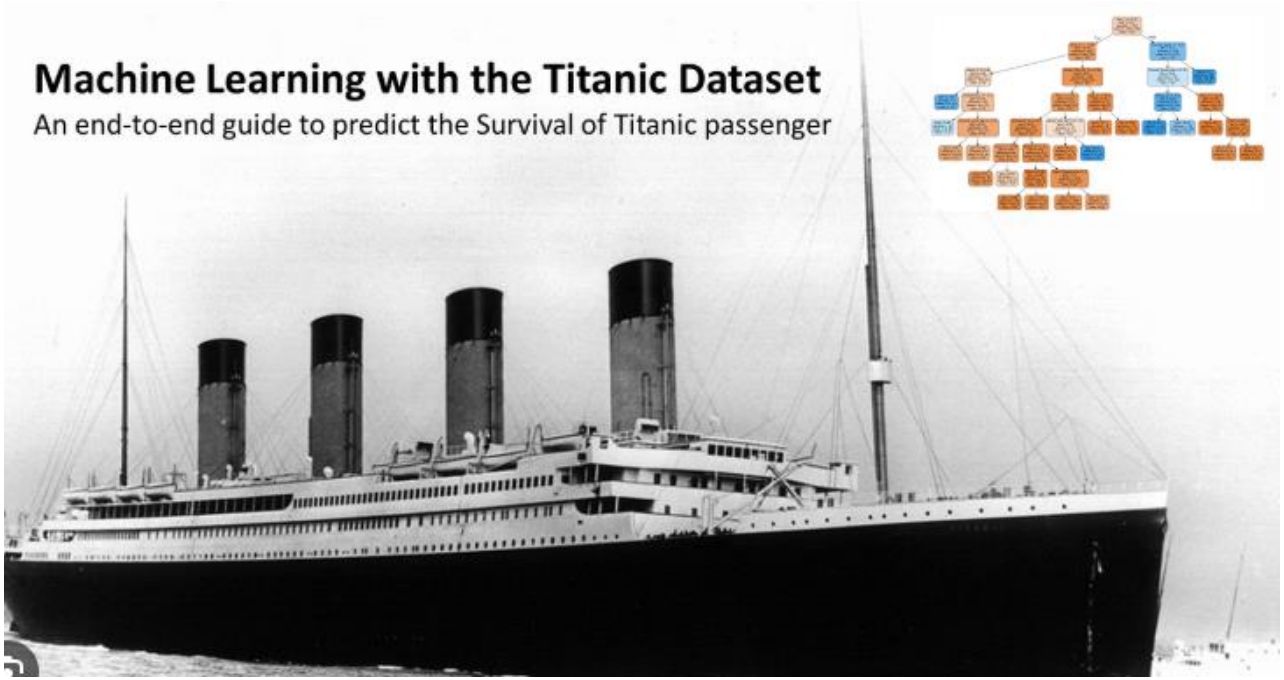
- RFE가 필요한 대표적인 데이터셋: Arrhythmia 데이터셋
- https://www.dropbox.com/scl/fi/ti5y10hn6l2wdl0tsj1h0/arrhythmia.data?rlkey=1gvntap3de2l6kqhodj6s0sn7&dl=0
- 특징 개수, 데이터 수
- 심장 부정맥(Arrhythmia)을 진단하는 데 사용
- 데이터: 452개 샘플
- 특징: 279개의 특징 (심박수, 심박 변동성등)
- Class: 부정맥이 있다 (1), 부정맥이 없다 (0)

특강 내용

타이타닉 데이터셋을 이용한 분류기 실습

- **Titanic dataset:** 타이타닉 호의 비극적인 침몰 사건(1912년)을 기반으로 만들어진 데이터셋으로, 데이터 분석과 머신러닝 연습을 위한 가장 유명한 데이터셋
- Titanic - Machine Learning from Disaster | Kaggle
- 목적: 주어진 정보를 바탕으로 승객의 생존 여부 예측
- 생존 여부는 이진 분류 문제 1: 생존, 0: 사망
- 데이터셋의 특징
- 결측치 전처리 필요 (예: age, cabin)
- 범주형 데이터: sex (성별), embarked (탑승 항구), 좌석 등급 (Pclass)들이 있어서 ML 모델 사용전에 encoding 필요

- **Titanic dataset**
- 다운로드: Titanic - Machine Learning from Disaster | Kaggle



- **PassengerId**: 승객의 고유 식별자
- **Survived**: 생존 여부 (목표 변수)
- **Pclass**: 좌석 등급 (1, 2, 3등석)
- **Name**: 승객 이름
- **Sex**: 성별 (male, female)
- **Age**: 나이
- **SibSp**: 함께 탑승한 형제자매/배우자의 수
- **Parch**: 함께 탑승한 부모/자녀의 수
- **Ticket**: 티켓 번호
- **Fare**: 운임 요금
- **Cabin**: 객실 번호 (종종 결측치가 많음)
- **Embarked**: 탑승 항구 (C = Cherbourg, Q = Queenstown, S = Southampton)

- 아래 링크를 클릭해서 코랩 열기:
- titanic.ipynb - Colab

데이터 형태

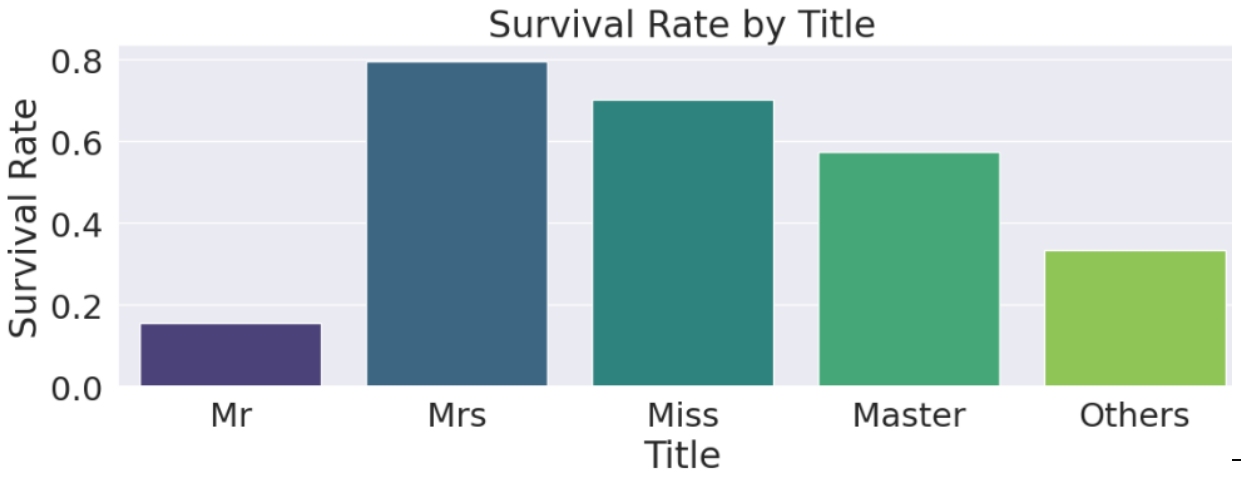
	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

- 결측치가 있는 특징들 확인:
- Train: Age, Cabin, Embarked
- Test: Age, Fare, Cabin

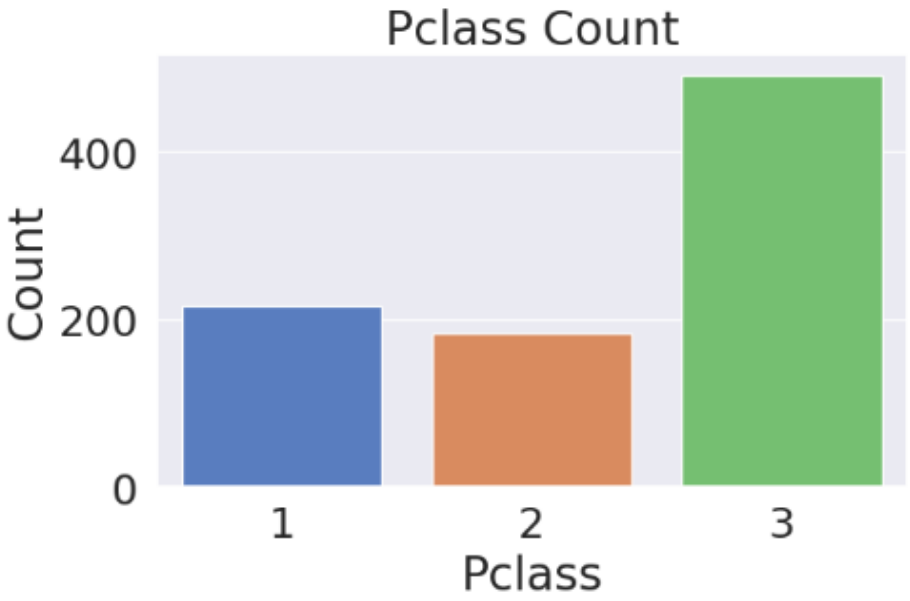
```
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
```

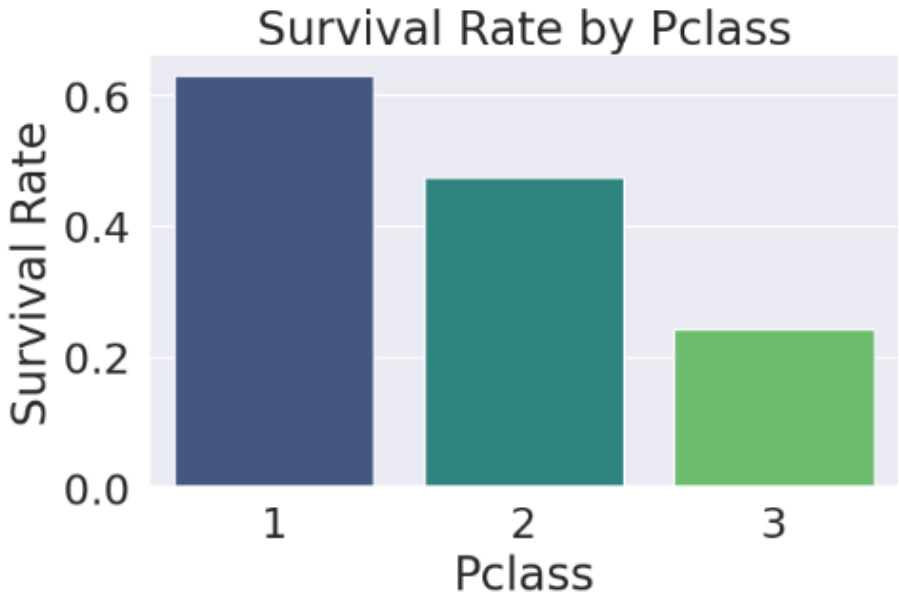
```
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  418 non-null    int64
 1   Pclass       418 non-null    int64
 2   Name         418 non-null    object
 3   Sex          418 non-null    object
 4   Age          332 non-null    float64
 5   SibSp        418 non-null    int64
 6   Parch        418 non-null    int64
 7   Ticket       418 non-null    object
 8   Fare         417 non-null    float64
 9   Cabin        91 non-null     object
 10  Embarked     418 non-null    object
```

- **Name 특징 전처리 및 시각화**
- **1. 이름에서 title 추출: Mr., Mrs, Miss..등**
- **4개의 title로 나누고 one-hot encoding**



- **Pclass 특징: 승객의 티켓 등급 (Pclass) 별 데이터 개수 및 등급 별 생존률 시각화**

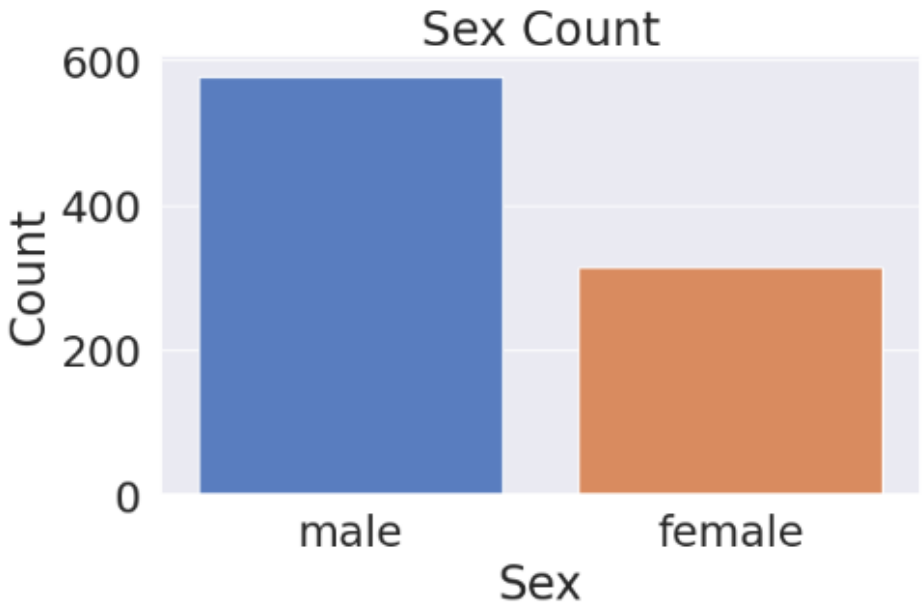


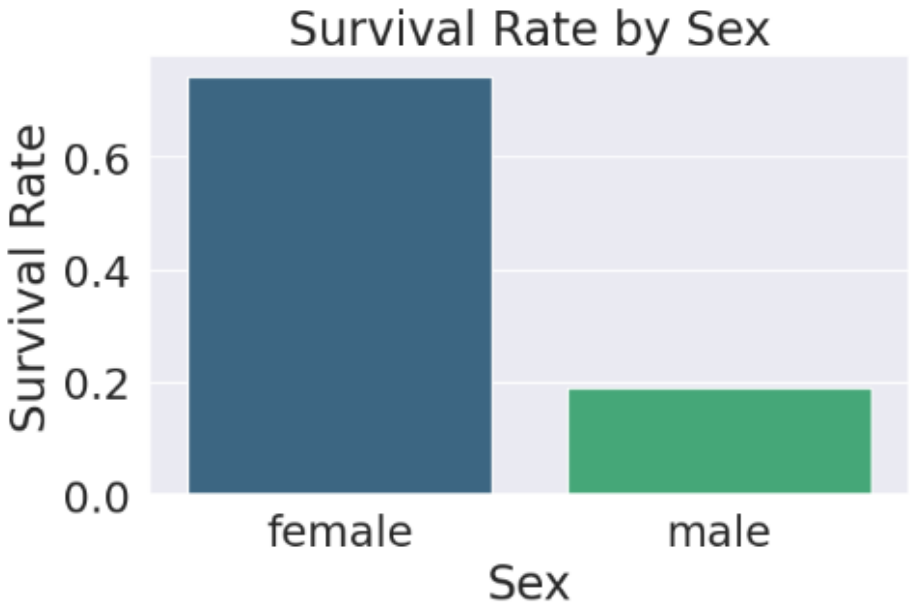


인천대학교

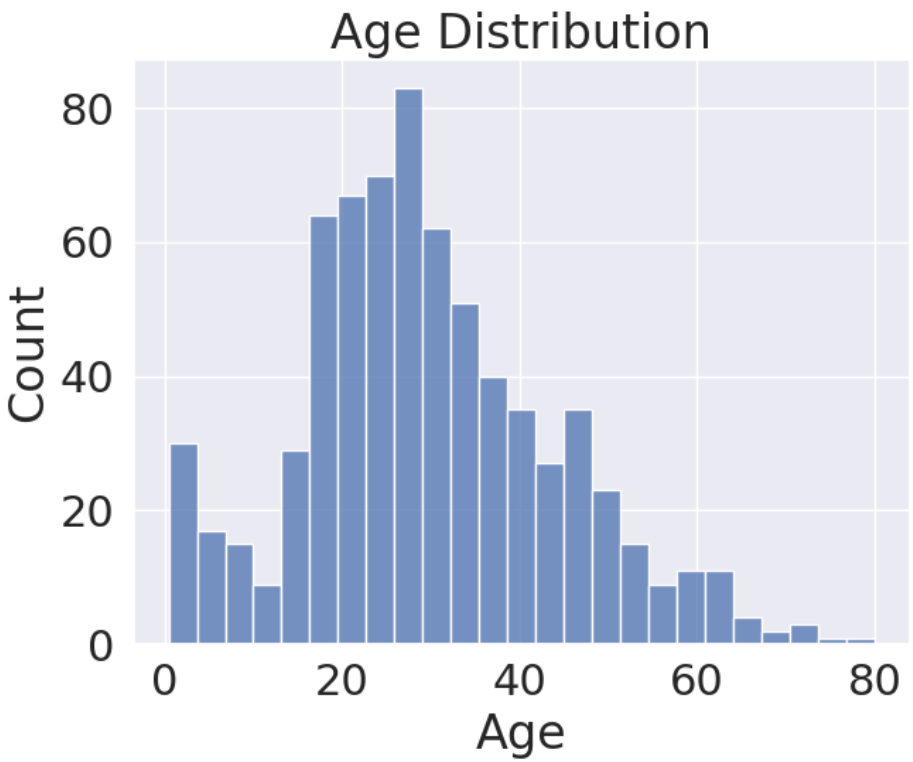
- **Sex 특징: 성별 별 데이터 개수 및 생존률 시각화**
- **Sex 특징에 대하여 label encoding을 수행**

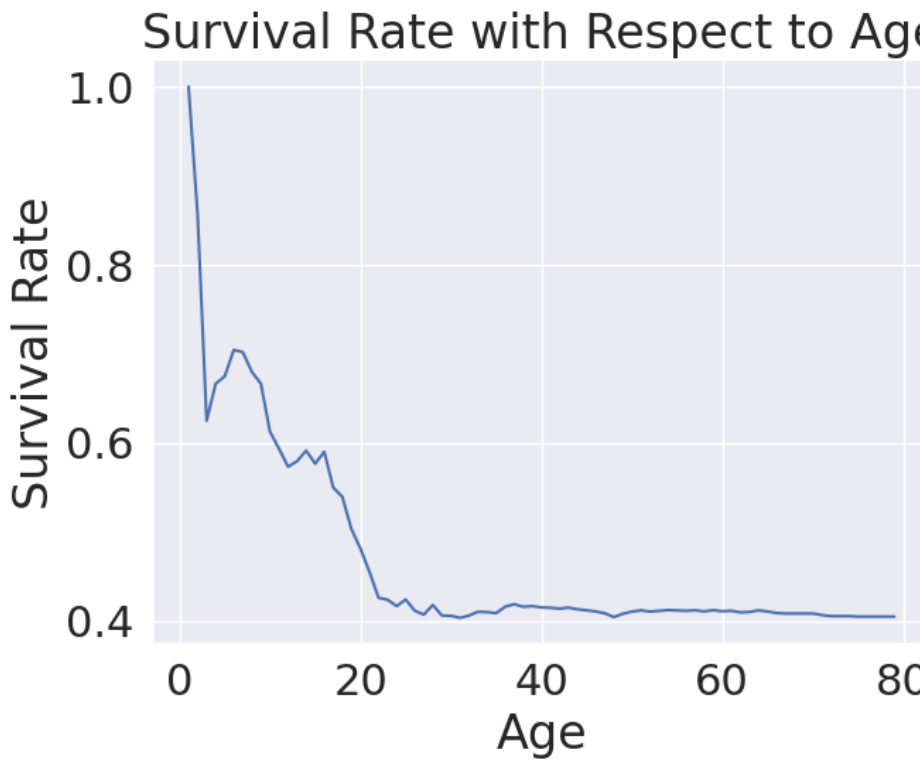
```
for data in data_train_test:
    data['Sex'] = data['Sex'].map({'male':0, 'female':1}).astype(int)
```





- **Age 특징: Age 데이터 개수 및 Age 별 생존률 시각화**

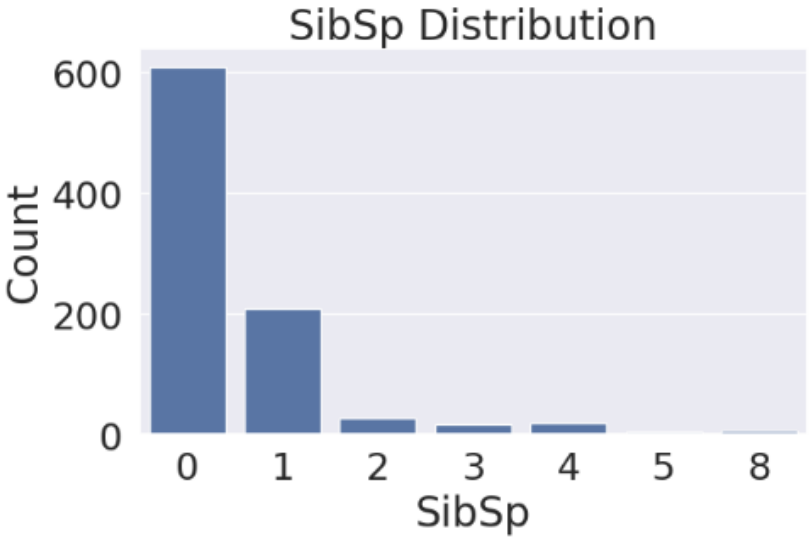


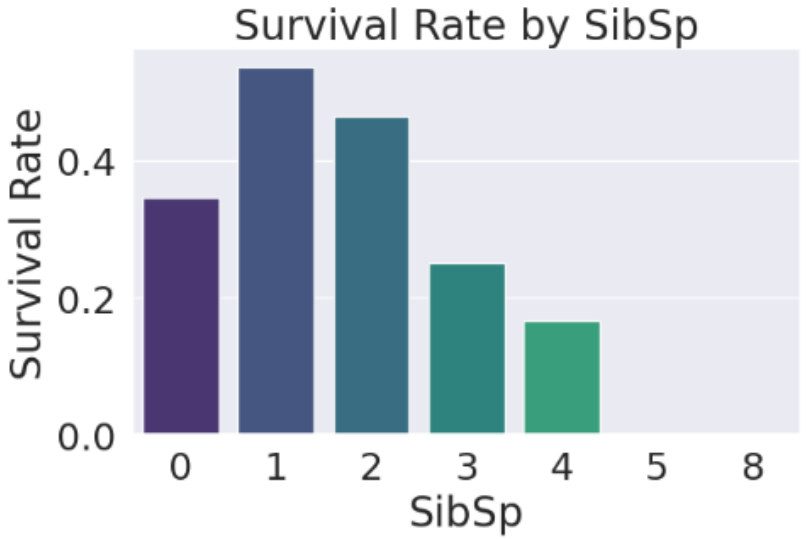


- **Age를 N등분하여 새로운 특징 만듬**

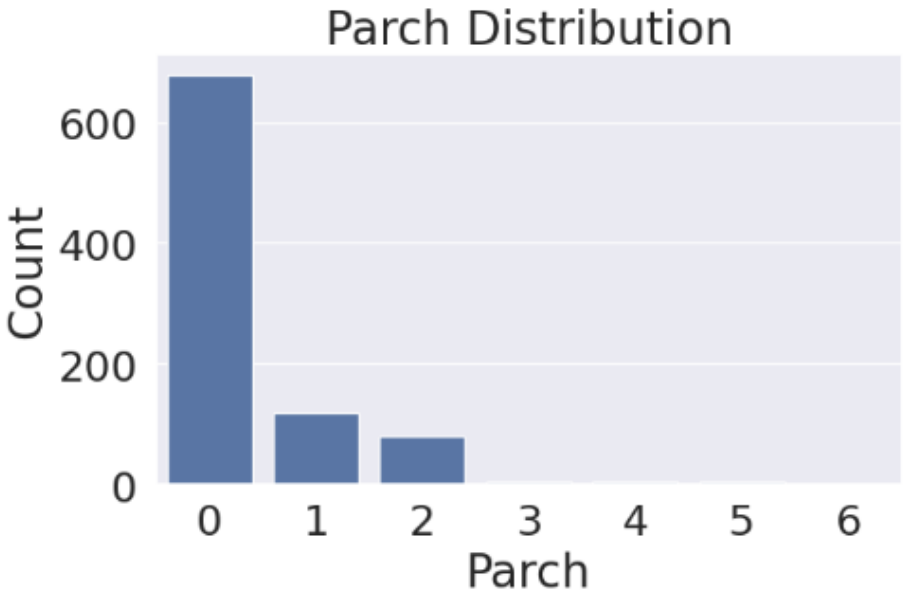
```
# AgeRange 범위 대로 값 변경
for data in data_train_test:
    data.loc[ data['Age'] <= 16, 'Age'] = 0
    data.loc[(data['Age'] > 16) & (data['Age'] <= 32), 'Age'] = 1
    data.loc[(data['Age'] > 32) & (data['Age'] <= 48), 'Age'] = 2
    data.loc[(data['Age'] > 48) & (data['Age'] <= 64), 'Age'] = 3
    data.loc[ data['Age'] > 64, 'Age'] = 4
```

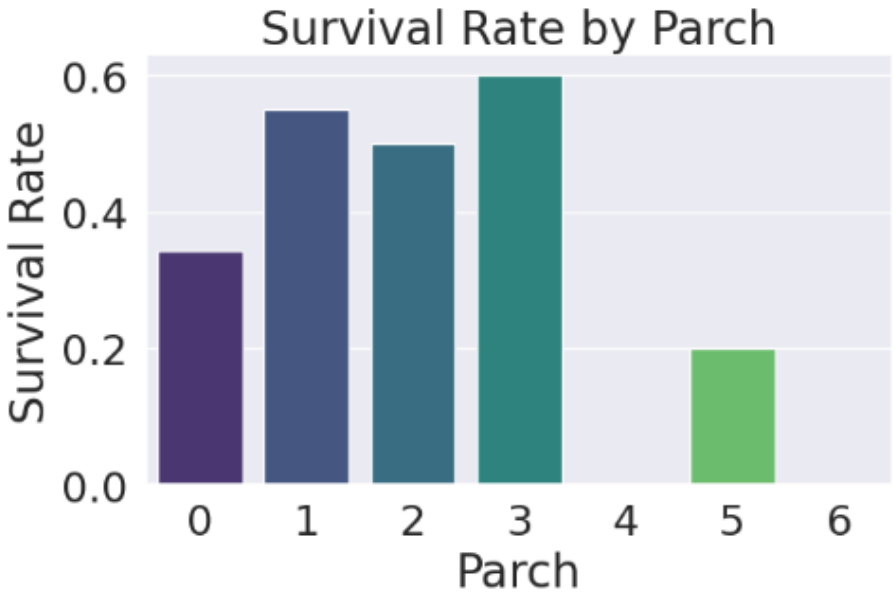
- **SibSp 특징: 승객과 함께 탑승한 형재자매 또는 배우자 수**





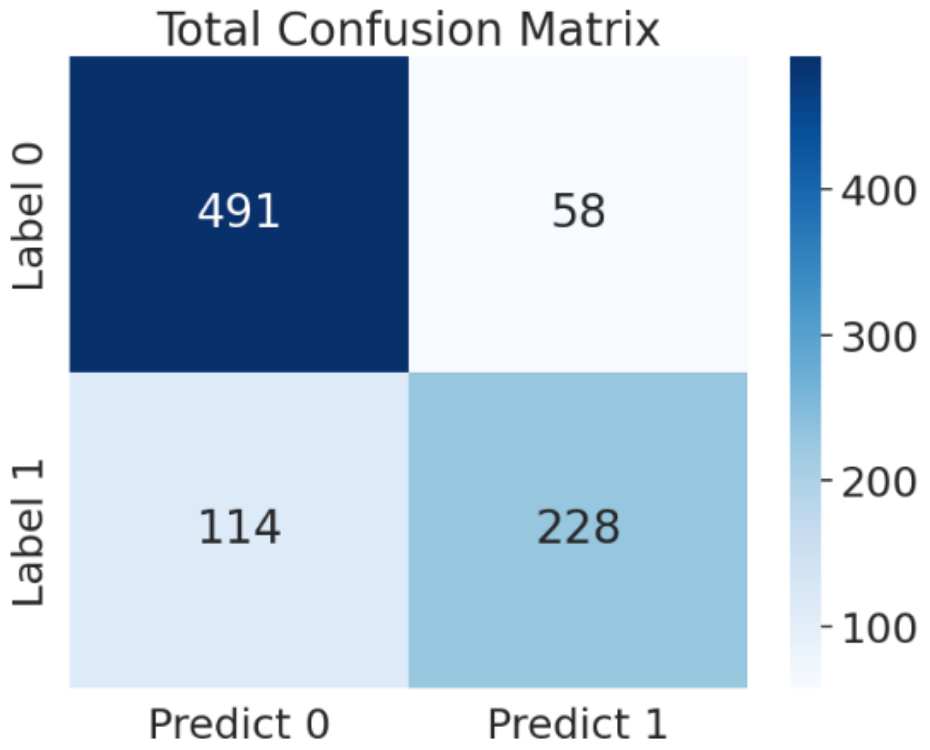
- **Parch: 승객과 함께 탑승한 부모 또는 자녀의 수**





- **Familysize라는 새로운 특징을 만들고 앞의 SibSp와 Parch 특징 삭제**

Decision tree와 cross validation 사용



Total Classification Report:

	precision	recall	f1-score	support
0	0.81	0.89	0.85	549
1	0.80	0.67	0.73	342
accuracy			0.81	891
macro avg	0.80	0.78	0.79	891
weighted avg	0.81	0.81	0.80	891

- **Q: 전처리, ML 모델 종류, 최적의 파라미터 등을 사용하여 분류기 성능을 더 높일 수 있나?**

- **Q: feature importance를 사용시 어떠한 특징이 생존자 예측에 가장 중요한 특징인가?**
- **성별 (sex), 티켓 클래스 (Pclass), 티켓 요금(fare)**