

분류

2025.01.23
김자영 강사

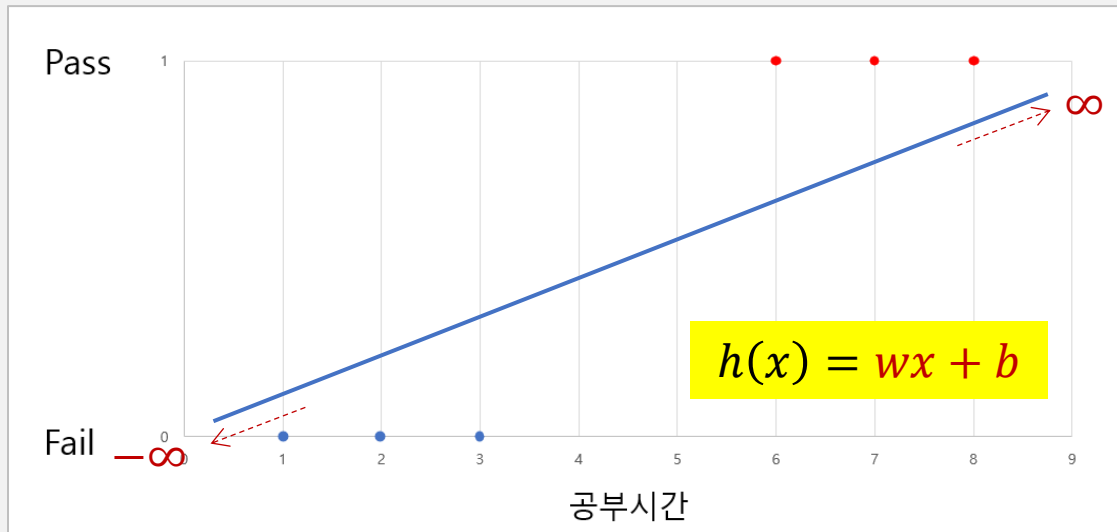
Logistic Regression

로지스틱 회귀 개념

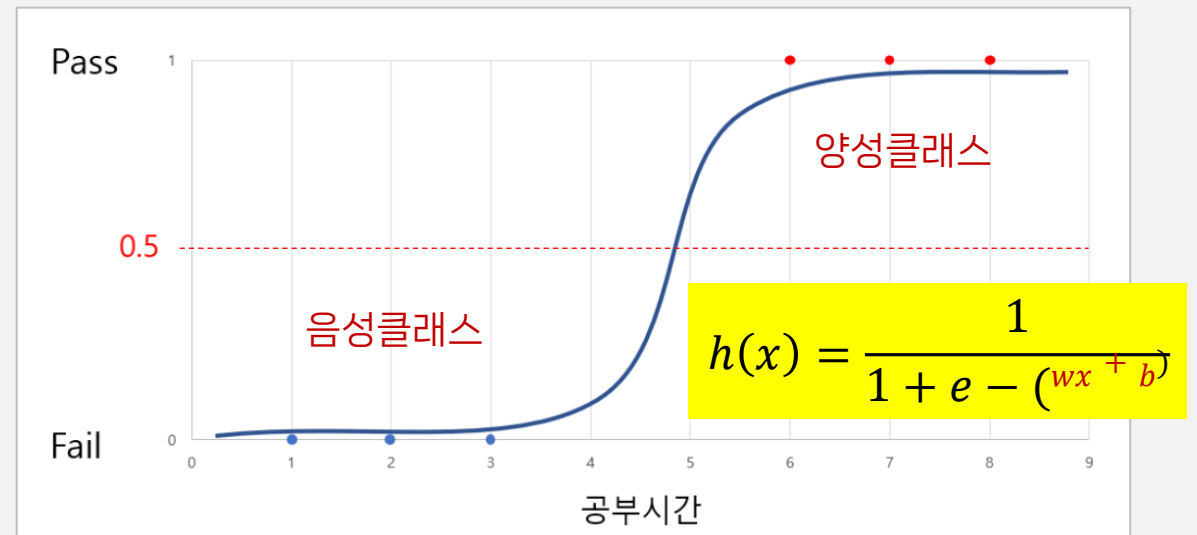
분류

- 종속 변수가 범주형일 때, 수행할 수 있는 회귀분석 기법의 한 종류
- 독립변수들의 선형 결합을 이용하여 개별 관측치가 속하는 집단을 확률로 예측
- 선형회귀의 출력값을 시그모이드 함수에 입력하여 0~1 사이의 확률로 변환

Linear Regression



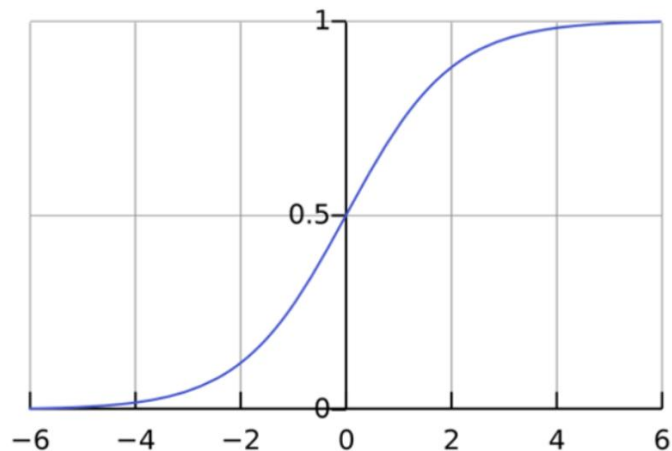
Logistic Regression



Logistic Regression

■ 시그모이드 함수(로지스틱 함수)

- 결과를 0과 1 사이의 확률로 변환하는 함수
- 로지스틱 회귀에서는 종속 변수의 값이 0과 1 사이의 확률을 나타내야 하기 때문에, 예측 값이 이 범위 내에 있도록 제한한다.
- 이를 위해 시그모이드 함수라는 S-형 곡선을 사용한다. 이 함수는 모든 입력값을 0과 1 사이의 값으로 대응시킨다.



<https://aws.amazon.com/ko/what-is/logistic-regression/>

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

x가 ∞ 에 가까워질수록 0에 가까워짐
x가 $-\infty$ 에 가까워질수록 1에 가까워짐

x가 ∞ 에 가까워질수록 1에 가까워짐
x가 $-\infty$ 에 가까워질수록 0에 가까워짐

K-Nearest Neighbors, KNN 알고리즘

K-Nearest Neighbors, KNN 알고리즘

▪ KNN알고리즘 개념

- 기본 원리

- ✓ 새로운 데이터 포인트의 클래스나 값을 결정할 때, **가장 가까운 K개의 이웃 데이터 포인트를 참조하는 방식**
- ✓ 학습 과정 없이 훈련 데이터에서 가장 가까운 K개의 이웃을 찾아 그들의 정보를 기반으로 분류나 회귀 수행

- 작동 방식

- ✓ 분류 : K개 이웃의 다수결로 클래스 결정
- ✓ 회귀 : K개 이웃의 목표 변수 평균으로 예측

- 거리측정

- ✓ 주로 유클리드 거리 사용
- ✓ 다른 거리 측정법(맨해튼, 민코프스키 등)도 가능

- K값 선택

- ✓ 홀수 선택 (분류 시 동점 방지)
- ✓ 작은 K : 과적합 위험. 노이즈에 민감
- ✓ 큰 K : 과소적합 위험

- 장점

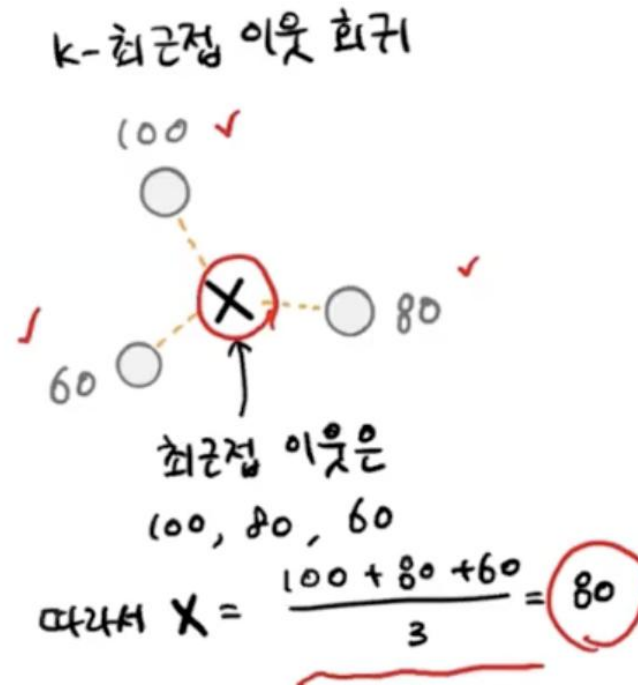
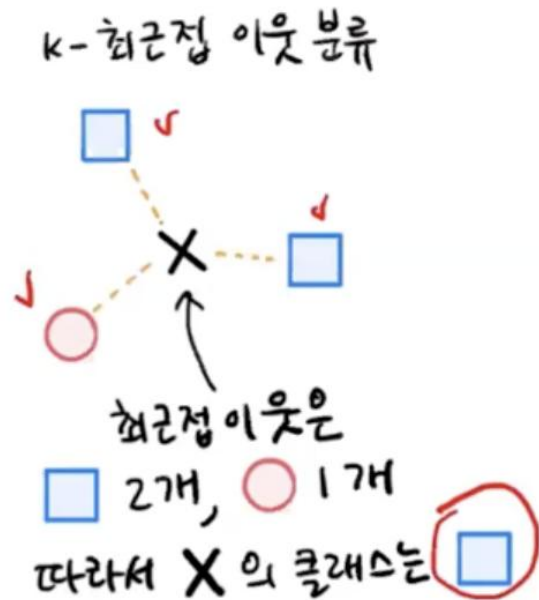
- ✓ 단순성 : 구현이 간단하고 이해하기 쉬움
- ✓ 훈련 과정이 빠름

- 단점

- ✓ 이상치에 민감 (피쳐 스케일링이 중요함)
- ✓ 피쳐의 중요도를 반영하지 않음

K-Nearest Neighbors, KNN 알고리즘

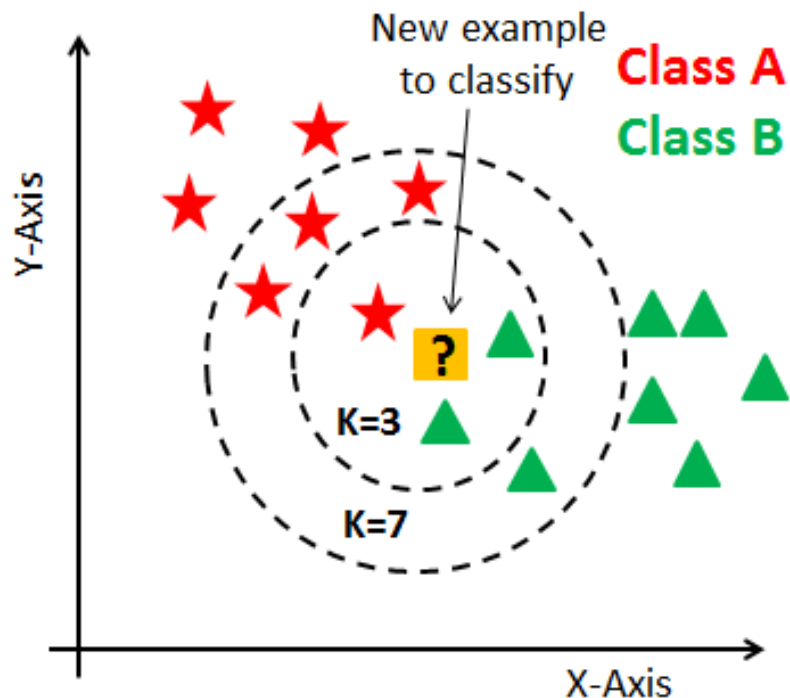
주변의 가장 가까운 K개의 샘플을 통해 값을 예측



[그림출처] <https://velog.io/@happyhilll/혼공-6장-회귀-문제-이해-k-최근접-이웃-알고리즘으로-풀어보기>

K-Nearest Neighbors, KNN 알고리즘

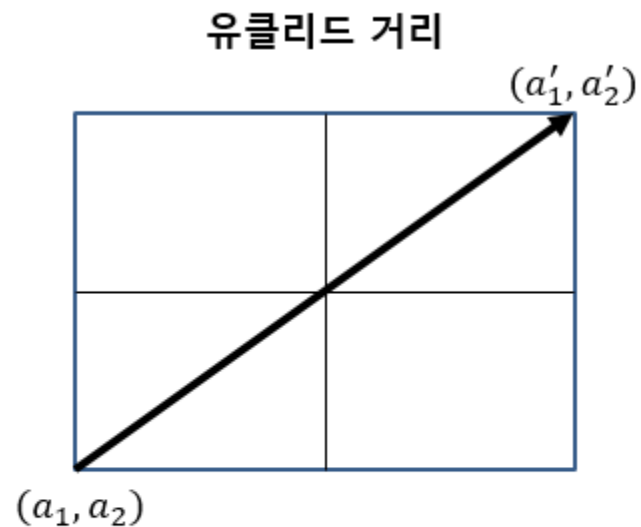
▪ KNN알고리즘 분류



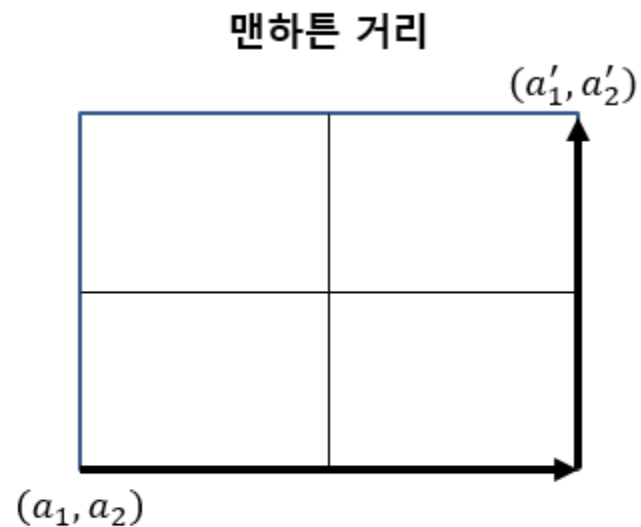
- ① **데이터 준비:** 학습 데이터셋 준비. 각 데이터 포인트는 다차원 공간의 점으로 표현
 - ② **거리 측정:** 새로운 데이터 포인트가 주어지면, 학습 데이터셋의 각 데이터 포인트와의 거리를 계산. 거리 측정을 위해 주로 유클리드 거리(Euclidean Distance)를 사용하지만, 맨해튼 거리(Manhattan Distance)나 다른 거리 측정 방법을 사용할 수도 있다.
 - ③ **K개의 이웃 선택:** 계산된 거리 중 가장 짧은 거리의 K개의 이웃을 선택
 - ④ **다수결 투표:** 선택된 K개의 이웃 중 가장 빈번하게 등장하는 클래스를 새로운 데이터 포인트의 클래스로 결정
- ※ **회귀의 경우 평균 계산:** 선택된 K개의 이웃의 값을 평균 내어 새로운 데이터 포인트의 값을 결정

K-Nearest Neighbors, KNN 알고리즘

- 거리 측정 방법



$$d = \sqrt{(a'_1 - a_1)^2 + (a'_2 - a_2)^2}$$



$$d = (a'_1 - a_1) + (a'_2 - a_2)$$

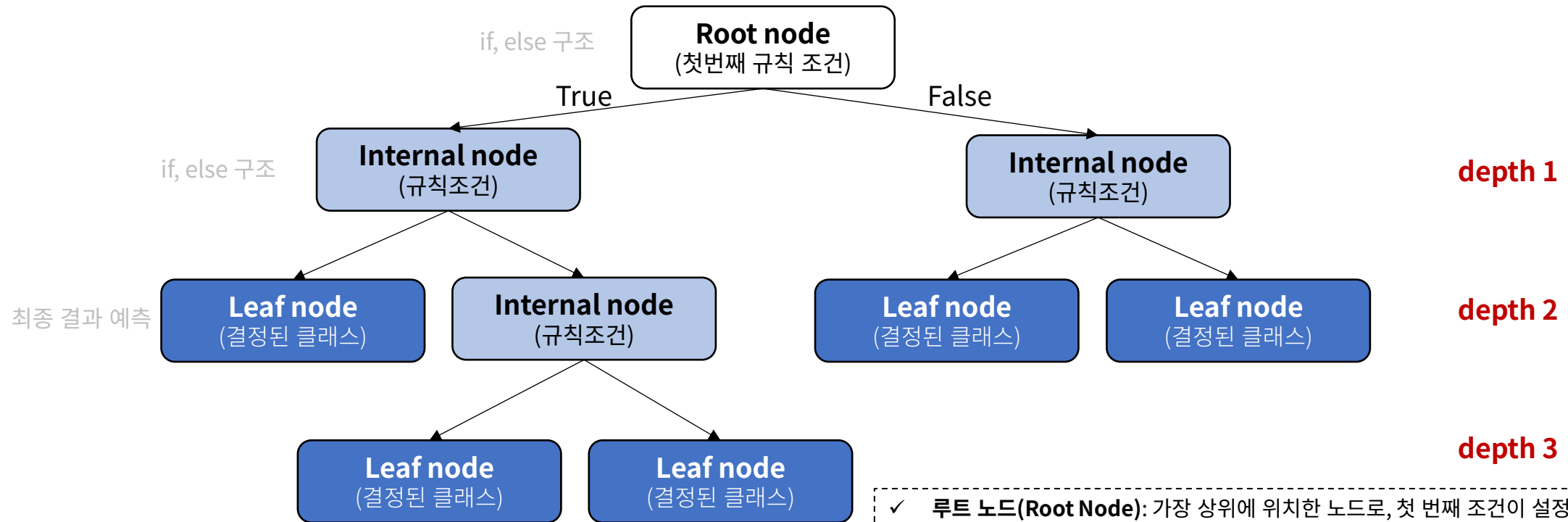
머신러닝 알고리즘

Decision Tree

Decision Tree (의사결정 트리)

■ Decision Tree 개요

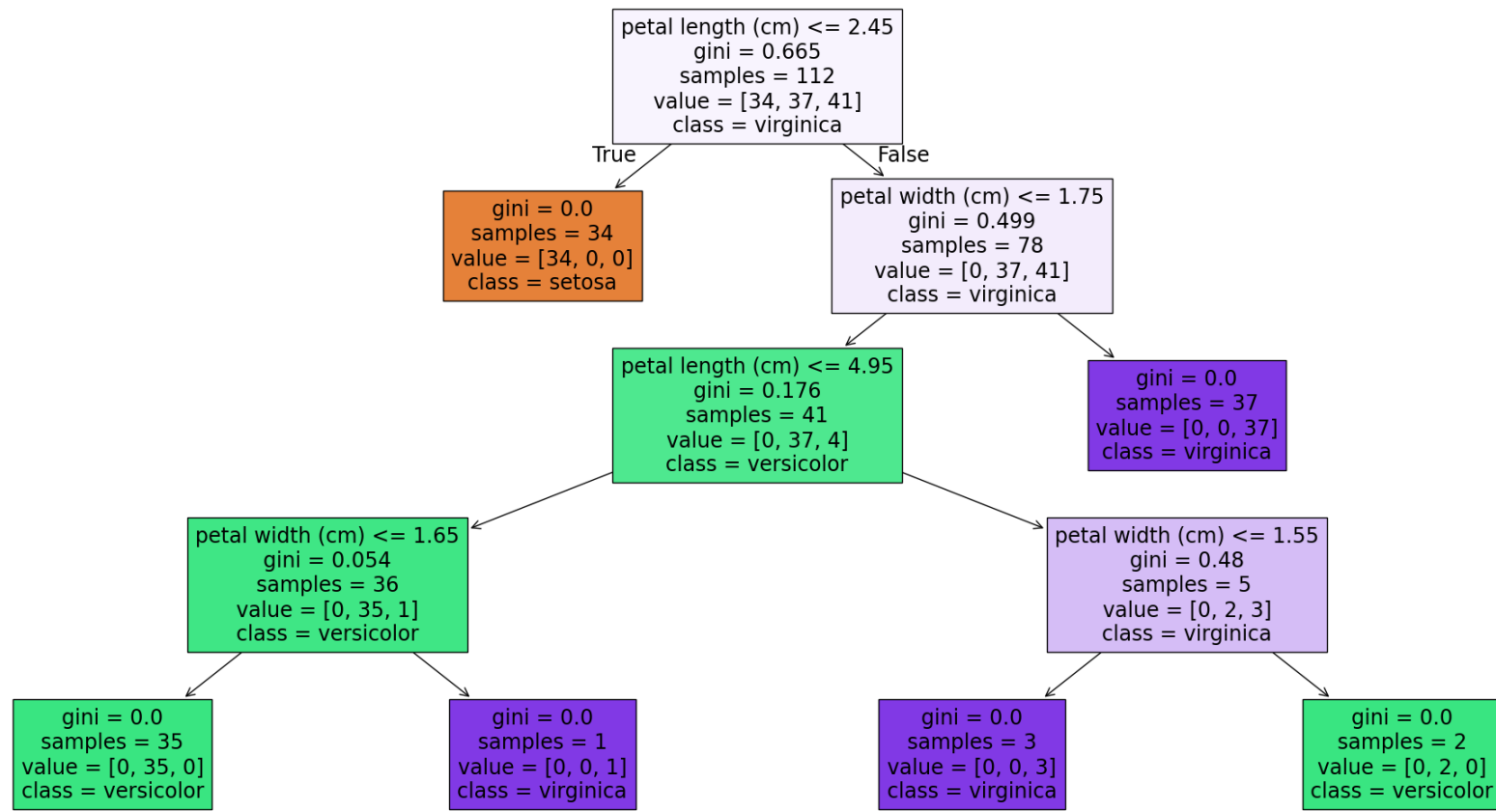
- 데이터의 특성을 바탕으로 분기를 만들어 트리 구조로 의사결정 과정을 모델링하는 방법
 - ✓ Root node에서 시작하여 조건에 따라 데이터를 분할하고, 이 과정을 반복하여 Leaf node에서 예측 결과를 도출한다.
 - ✓ 분류 및 회귀를 모두 지원한다.



- ✓ **루트 노드(Root Node)**: 가장 상위에 위치한 노드로, 첫 번째 조건이 설정된 지점.
- ✓ **내부 노드(Internal Node)**: 각 분할의 중간 지점으로, 더 세부적인 조건을 설정하는 부분.
- ✓ **단말 노드(Leaf Node)**: 더 이상 분할이 일어나지 않고, 최종적으로 결과를 예측하는 지점.

Decision Tree Classifier

■ Decision Tree Classifier 예시



Decision Tree Classifier

■ 데이터 분할 기준

Information Gain

정보이득

- 엔트로피 개념을 기반으로 한다.
- **엔트로피(Entropy) : 데이터의 혼잡도**
 - ✓ 각 클래스의 비율에 로그를 적용하여 계산

$$-\sum (p_i \log_2 p_i)$$
 - ✓ 최솟값 : 0
 - ✓ 최댓값 : $\log_2 k$ (k는 클래스 수)
- **정보이득(Information Gain)**
: 분할 전후의 엔트로피의 차이
- 정보이득이 높은 속성을 기준으로 분할
- 더 세밀하게 측정하지만 계산이 복잡할 수 있다.

Gini Impurity

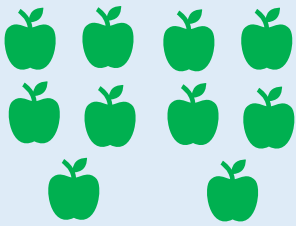
지니 불순도

- **데이터의 혼합 정도**를 나타내는 지표
 - ✓ 노드에서 무작위로 선택된 샘플이 잘못 분류될 확률
- 각 클래스의 비율의 제곱합을 뺀 값으로 계산

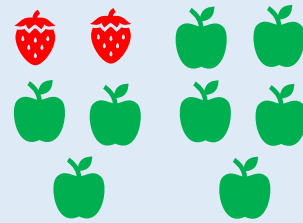
$$G = 1 - \sum (p_i^2)$$
 - ✓ 최솟값 : 0
 - ✓ 최댓값 : $1 - \frac{1}{n}$ (클래스가 2개일 때 0.5, 3개일 때 0.667, 4개일 때 0.75, ...)
- Gini Impurity가 낮은 속성을 기준으로(순도가 높은 쪽으로) 분할 기준을 선택한다.
- 계산이 간단하고 빠르며, 더 자주 사용된다.

Decision Tree Classifier

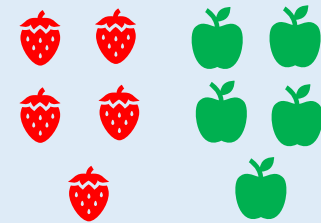
■ 지니 불순도(Gini Impurity)



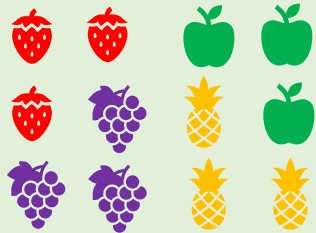
$$GI = 1 - 1^2 = 0$$



$$GI = 1 - (0.2^2 + 0.8^2) = 0.32$$



$$GI = 1 - (0.5^2 + 0.5^2) = 0.5$$



$$GI = 1 - (0.25^2 + 0.25^2 + 0.25^2 + 0.25^2)$$

Decision Tree (의사결정 트리)

■ Decision Tree 장단점

장점	단점
<ul style="list-style-type: none">• 만들어진 모델을 쉽게 시각화 할 수 있고, 모델이 어떻게 훈련되었는지 해석하기 쉽다.• 피처 스케일에 거의 구애 받지 않는다.• 데이터의 비선형 관계를 효과적으로 처리할 수 있다.• 범주형 데이터와 연속형 데이터를 모두 처리할 수 있다.• 데이터 분포에 대한 가정이 필요 없다.(비 모수적 모델)• 이상치에 강건하다.• 중요한 특성을 자동으로 선택한다.	<ul style="list-style-type: none">• 트리가 너무 깊어질 경우, 학습 데이터에 과적합되어 일반화 성능이 떨어질 수 있다.• 데이터의 작은 변화에도 트리 구조가 크게 바뀔 수 있다.• 최적의 트리를 찾기 어렵다.• 데이터의 클래스 불균형이 심할 경우 다수 클래스에 편향될 수 있다.• 대규모 데이터셋에서 계산 비용이 높다.

Decision Tree (의사결정 트리)

■ 가지치기를 위한 하이퍼 파라미터(Hyper parameter)

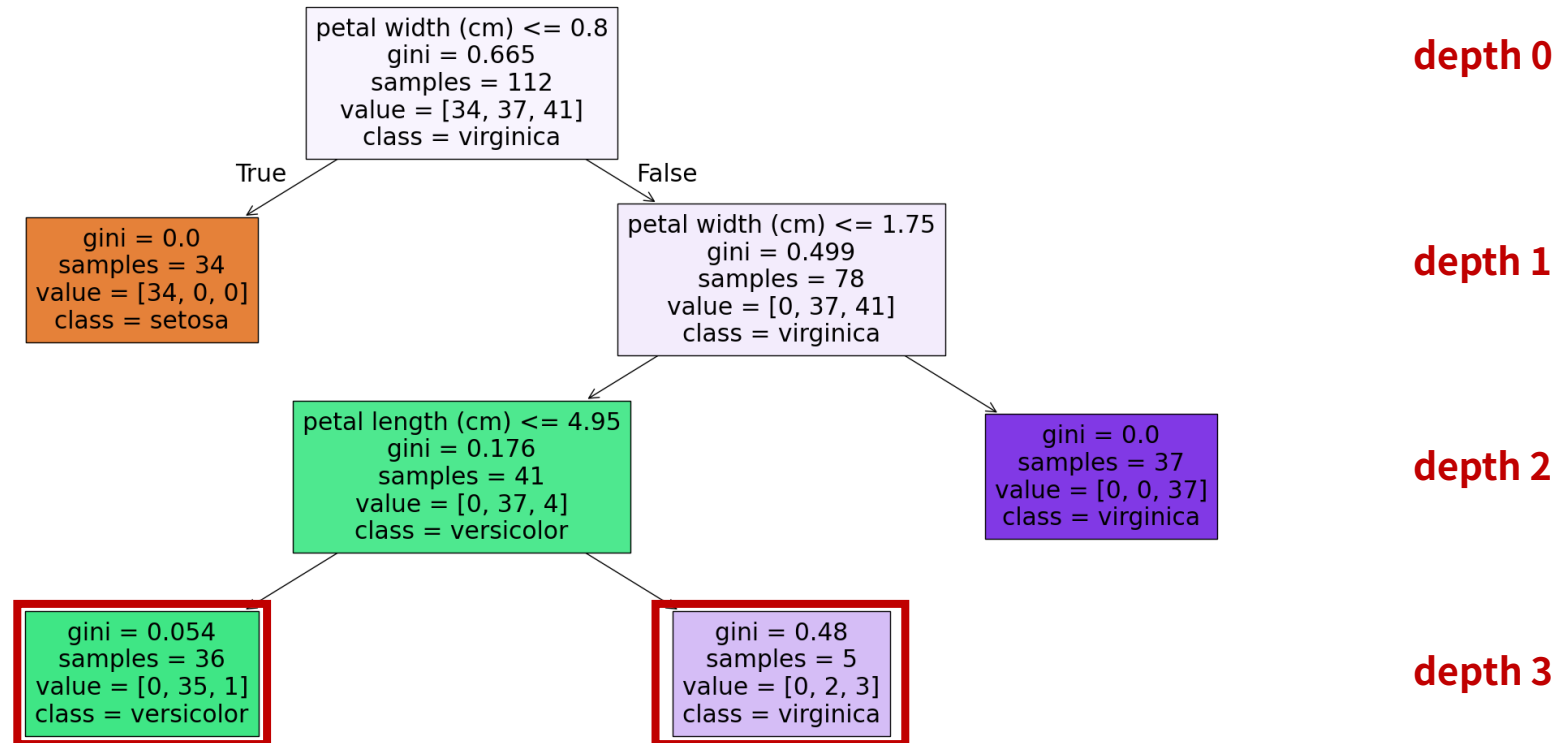
- 목적 : 트리의 복잡성 감소 → 과적합(Overfitting) 방지 → 일반화 성능 향상
- 주요 parameter

max_depth	트리의 최대 깊이 규정. 깊이가 깊어지면 과적합 문제가 발생할 수 있으므로 제어 필요
min_sample_split	노드를 분할하기 위한 최소한의 샘플 데이터 수
min_sample_leaf	leaf(말단 노드)가 되기 위한 최소한의 데이터 수
max_features	최적의 분할을 위해 고려할 최대 피처 개수
max_leaf_nodes	leaf의 최대 개수

- 과도한 가지치기는 underfitting을 발생시킬 수 있으므로 적절한 수준의 가지치기 필요

Decision Tree Classifier

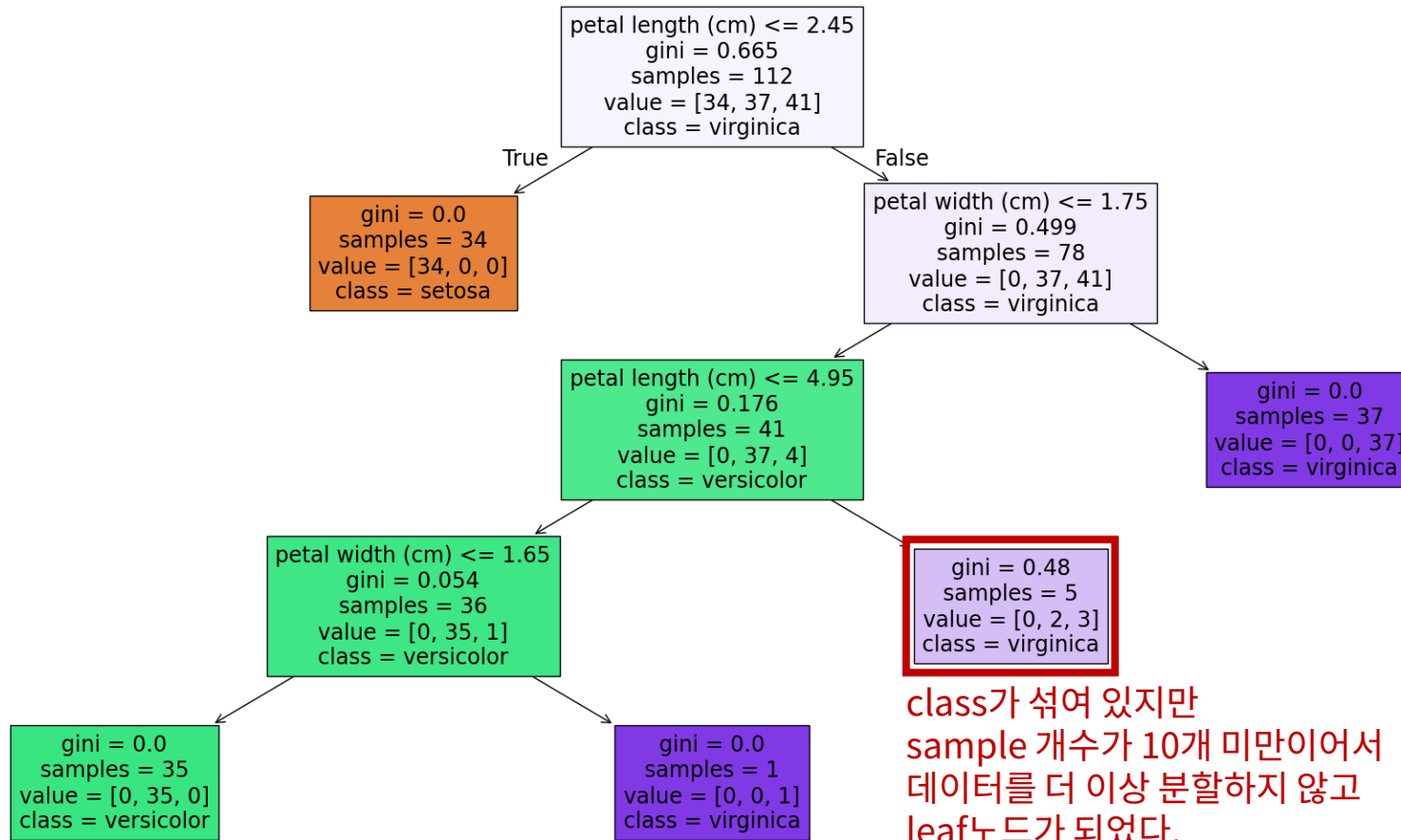
■ 가지치기 ($\text{max_depth}=3$)



class가 섞여 있지만 데이터를 더 이상 분할하지 않고 leaf노드가 되었다.

Decision Tree Classifier

■ 가지치기 (`min_samples_split=10`)



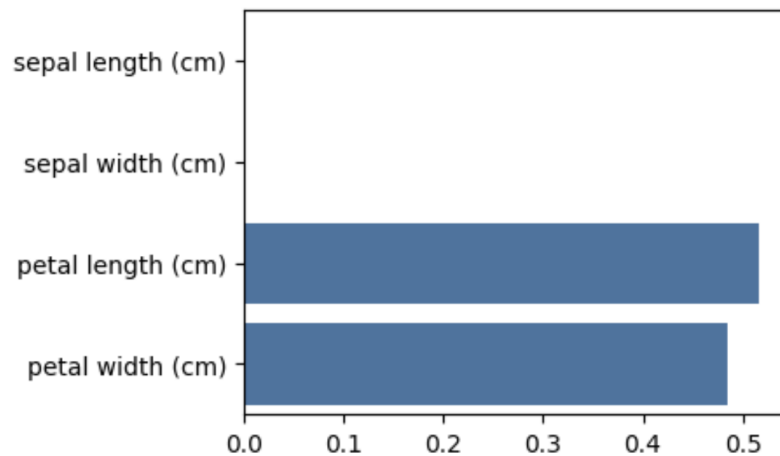
Decision Tree Classifier

■ 특성의 중요도

```
# 특성의 중요도
print(f'특성의 중요도 >>> {dtc.feature_importances_}')

plt.figure(figsize=(4,3))
sns.barplot(y=feature_name, x=dtc.feature_importances_)
plt.show()
```

특성의 중요도 >>> [0. 0. 0.51613034 0.48386966]



앙상블

ensemble

Voting Classifier

앙상블 학습

■ 앙상블 학습 개요

개념	- 여러 개의 모델을 결합하여 더 강력한 예측 모델을 만드는 기법.
특징	- 각 모델은 약한 학습기(weak learner)로, 이들을 결합하여 강한 학습기(long learner)를 만든다. - 서로 다른 알고리즘, 하이퍼파라미터, 또는 학습 데이터 subset을 사용하여 다양한 모델을 생성한다.
장점	- 성능 향상 : 단일 모델보다 일반적으로 더 높은 예측 정확도를 제공한다. - 안정성 : 이상치나 노이즈에 대해 더 안정적인 예측을 할 수 있다. - 일반화 : 개별 모델의 과적합을 줄이고 전체적인 일반화 성능을 향상시킨다. - 편향 감소 : 다양한 모델을 사용함으로써 개별 모델의 편향성을 상쇄할 수 있다. - 상호보완 : 서로 다른 모델의 약점을 보완하고, 각 모델이 놓치는 부분을 다른 모델이 잡아낼 수 있다.
단점	- 계산 비용 : 여러 모델을 학습하고 예측하기 위해 더 많은 시간과 리소스가 필요하다. - 복잡성 : 여러 모델을 관리하고 최적화하는 것이 더 복잡할 수 있다. - 해석의 어려움 : 개별 모델에 비해 최종 예측 결과의 해석이 어려울 수 있다.

앙상블 학습

■ 앙상블 학습의 유형

보팅 Voting

▪ 서로 다른 여러 알고리즘을 사용한 각 분류기에 대해 투표를 통해 최종 결과를 예측한다.

▪ 보팅의 종류

- **하드 보팅**

: 다수결 원칙으로
최종 예측 결정

- **소프트 보팅**

: 각 모델의 예측 확률을
평균하여 최종 예측 결정

보팅의 예시 >>>
- VotingClassifier

배깅 Bagging

▪ 동일한 알고리즘을 사용하지
만 데이터 샘플을 다르게 하여
여러 모델을 학습시킨 후 보팅
을 수행한다.

▪ 샘플 추출 방법

- 부트스트래핑(Bootstrapping)
: 복원 추출

배깅의 예시 >>>
- RandomForest

부스팅 Boosting

▪ 약한 학습기를 순차적으로
학습시키고, 이전 단계에서 오
분류된 데이터에 더 높은 가중
치를 부여하여 학습시킨다.

→ 잘못된 예측에 집중하여 성능을
점진적으로 향상시키는 방법

부스팅의 예시 >>>

- AdaBoost
- Gradient Boosting
- XGBoost
- LightGBM
- CatBoost

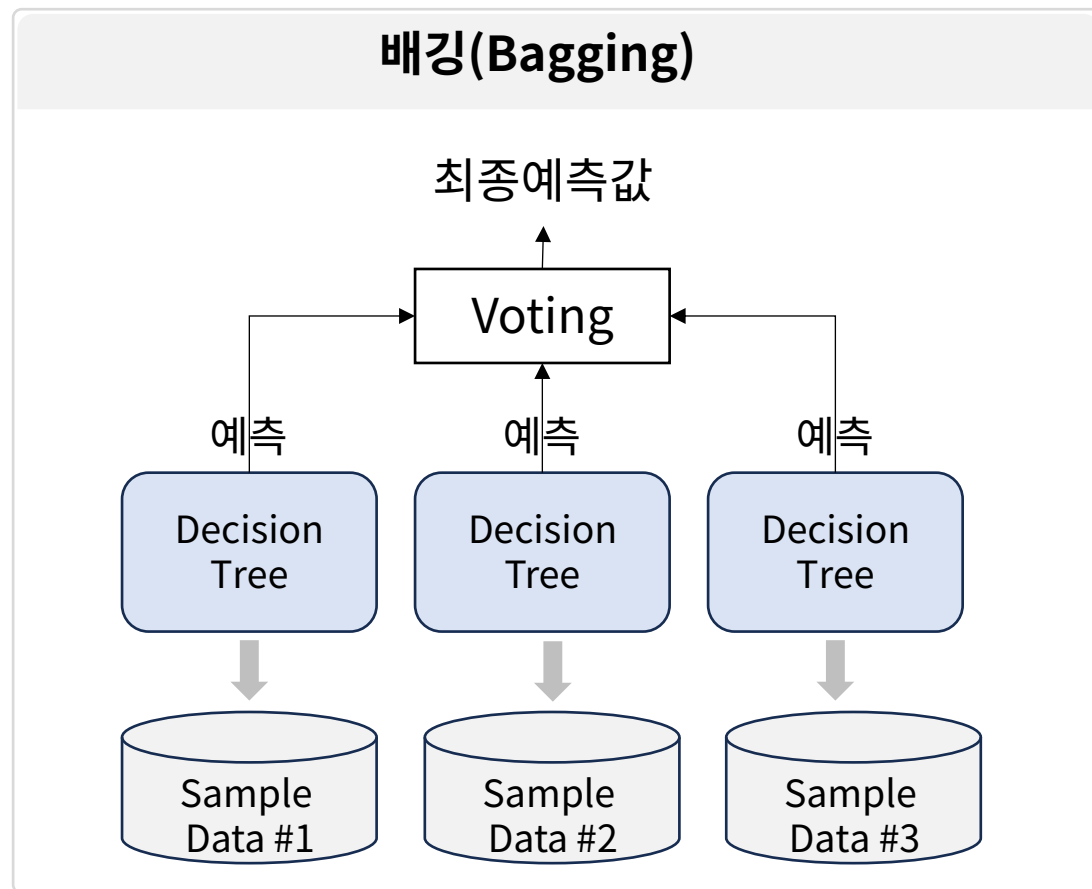
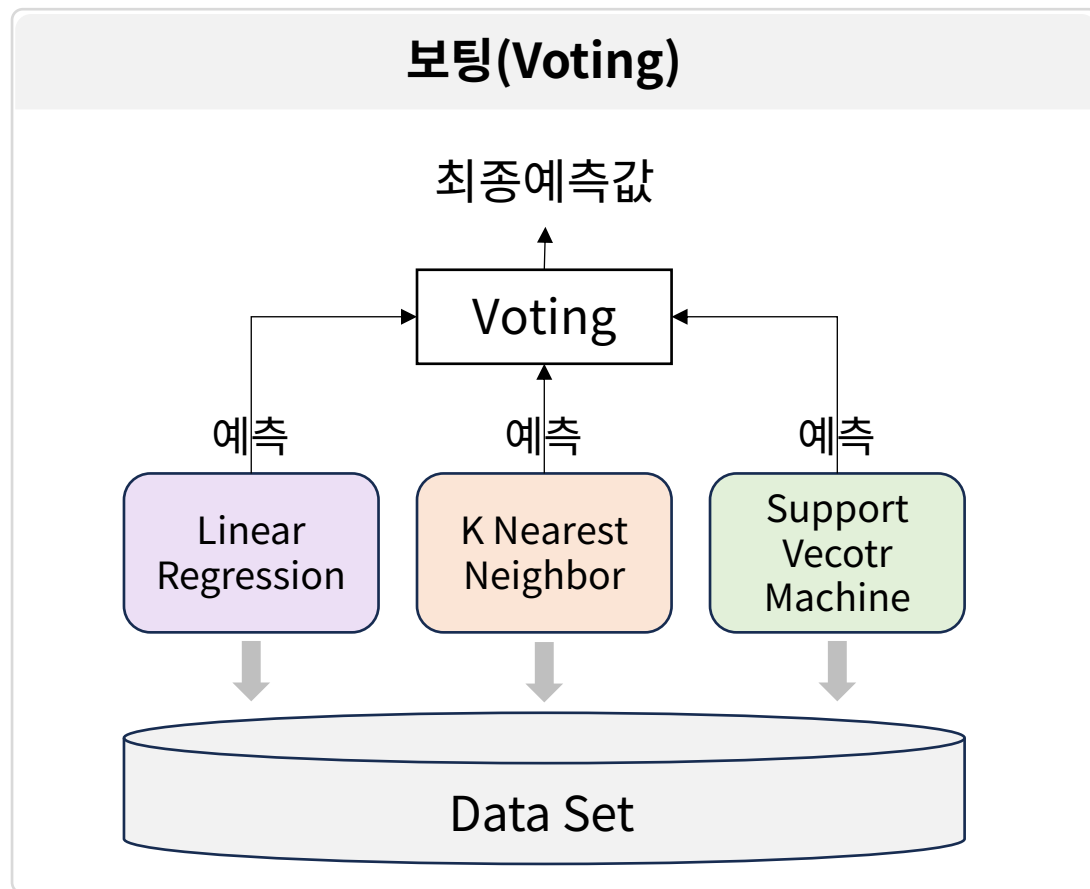
스태킹 Stacking

▪ 여러 개의 서로 다른 모델을
학습시키고, 각 모델의 예측 결
과를 다시 학습데이터로 만들
어 다른 모델(**Meta Model**)로
재 학습시켜 결과를 예측한다.

→ 다양한 모델의 장점을 결합하여
성능을 향상시키는 방법

앙상블 학습

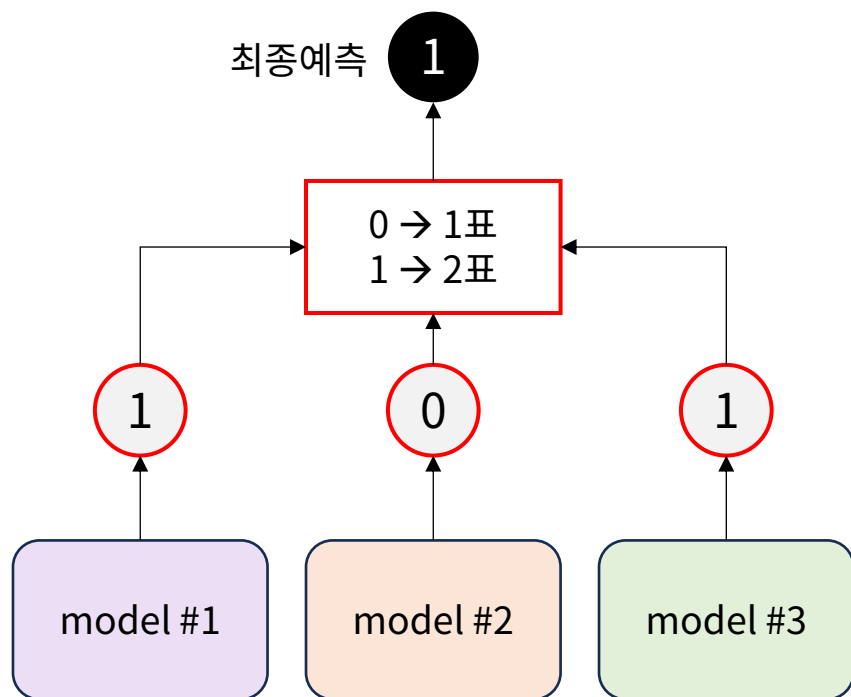
■ 보팅(Voting)과 배깅(Bagging)



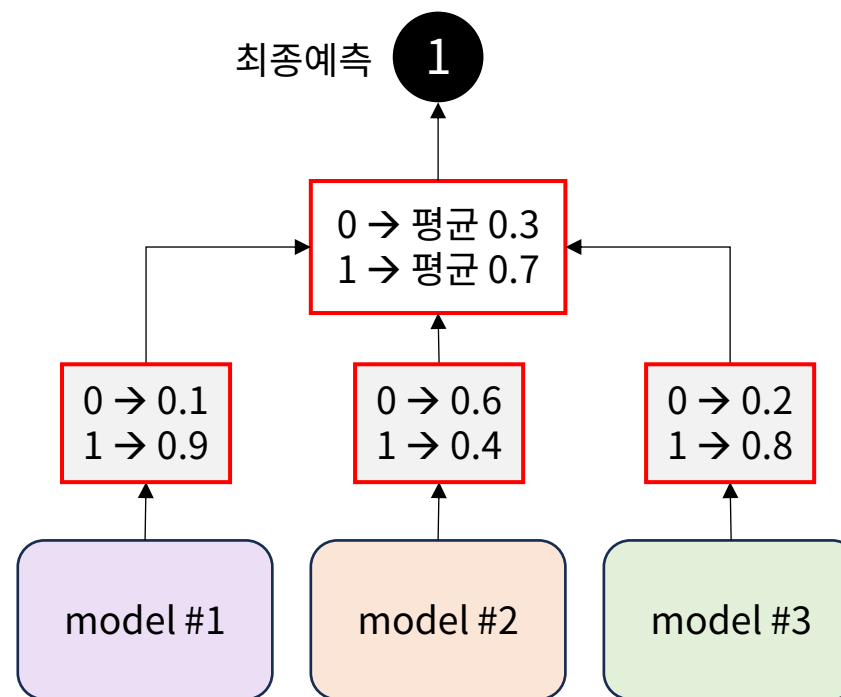
앙상블 학습

■ 하드보팅(Hard Voting)과 소프트보팅(Soft Voting)

하드보팅(Hard Voting)

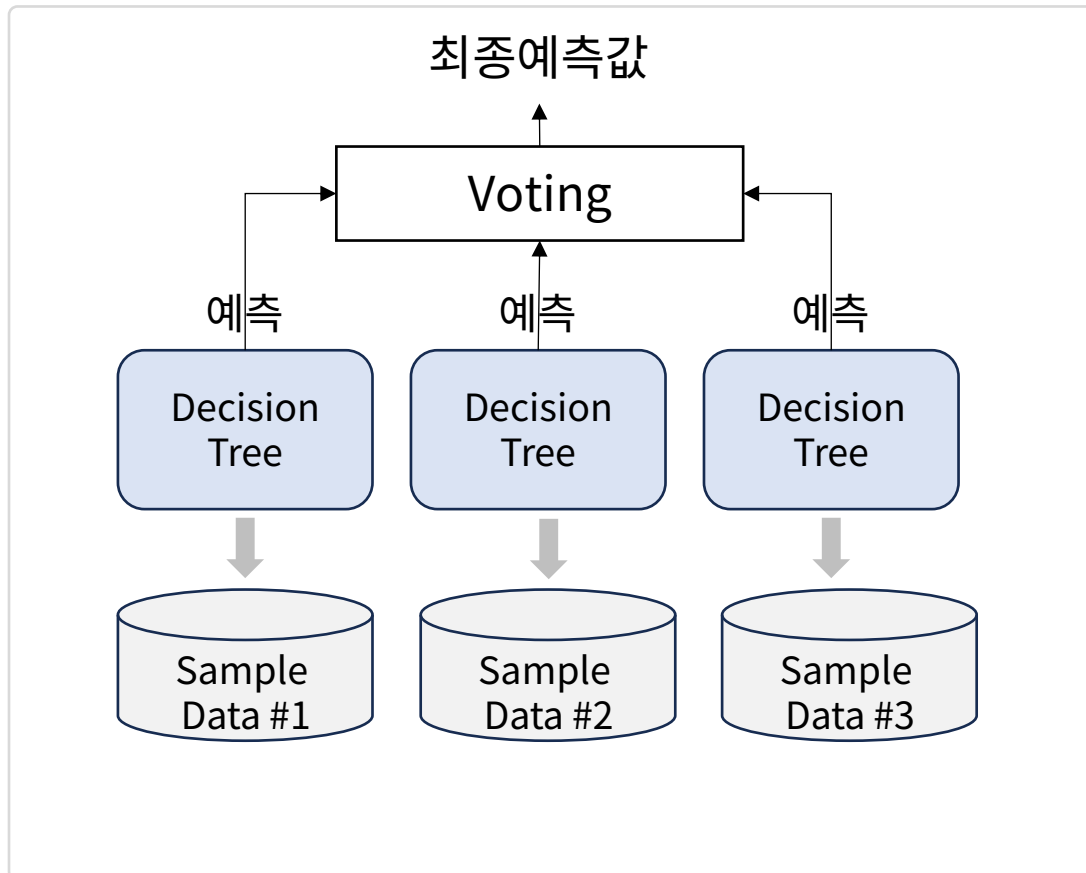


소프트보팅(Soft Voting)



RandomForest

■ 개념 및 특징



- 여러 개의 **의사결정트리(Dicision Tree)**를 결합하여 예측 성능을 향상시키는 기법으로, 분류와 회귀에 모두 적용 가능하다.
- **배깅(Bagging, Bootstrap Aggregating)** 기법을 사용
- **랜덤 피처 선택** : 각 트리의 노드 분할 시 무작위로 선택된 피처의 서브셋만을 사용하여 최적의 분할 기준을 찾는다.
 - 트리 간의 상관성을 줄이고 모델의 다양성을 증가시킨다.
 - 특정 특성에 과도하게 의존하는 것을 막아 과적합을 방지한다.
 - 일부 특성만을 고려하므로 개별 트리의 학습 속도가 빠르다.
- 최종 예측값 결정 방법
 - ✓ 분류 : SoftVoting
 - ✓ 회귀 : 각 트리의 예측 값의 평균

RandomForest

■ 주요 하이퍼파라미터

트리 관련 파라미터

- **n-estimators** : RandomForest를 구성하는 결정트리의 개수. 일반적으로 더 많은 트리를 사용할수록 성능이 향상되지만 학습 시간도 증가하고, 어느 정도 이상으로 늘어나면 성능 향상이 둔화된다.
- **max_depth** : 각 트리의 최대 깊이. 과적합을 방지하기 위해 적절한 값으로 제한하는 것이 좋다.
- **min_sample_split** : 내부 노트를 분할하기 위해 필요한 최소 샘플 수. 과적합 제어를 위한 파라미터.
- **min_sample_leaf** : 리프 노트가 가져야 할 최소 샘플 수. 과적합 제어를 위한 파라미터.

특성 선택 파라미터

- **max_features** : 각 노드에서 분할을 위해 고려할 특성의 최대 수. 낮게 설정하면 트리들 사이의 다양성이 증가하며, 과적합을 방지할 수 있다.

샘플링 관련 파라미터

- **bootstrap** : 부트스트랩을 사용할지 여부
- **max_samples** : 각 트리를 훈련할 때 사용할 샘플의 비율 또는 수

기타 파라미터

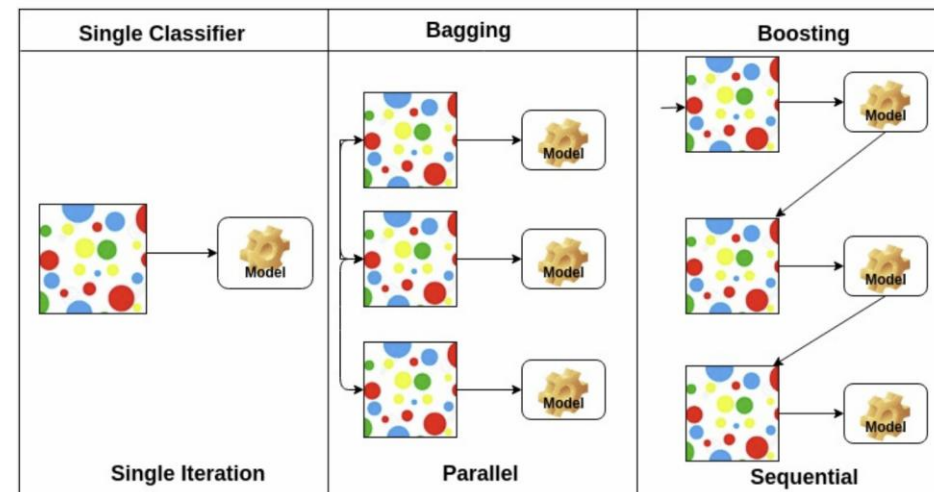
- **criterion** : 분할 품질을 측정하는 기준. (분류:gini/entropy, 회귀:mse/mae)
- **random_state** : 재현성을 위한 난수 시드 값

앙상블 학습

▪ Boosting 알고리즘의 주요 개념

• Boosting 알고리즘의 주요 개념

- ✓ 순차적 학습 : 여러 약한 학습기(Weak learner)를 **순차적으로 학습**시킨다.
- ✓ 가중치 조절 : 모델이 학습하는 동안 **잘못 예측한 데이터에 더 높은 가중치를 부여**하여 이후 모델이 이를 더 정확하게 예측할 수 있도록 유도한다.
- ✓ 최종 모델 : 여러 모델의 예측을 결합하여 최종 모델을 만든다. 이 때 각 학습기의 중요도에 따라 가중치가 부여된다.



앙상블 학습

▪ Boosting 알고리즘의 종류

AdaBoost

Adaptive Boosting

- ✓ Boosting 알고리즘의 초기 버전
- ✓ 오분류된 데이터에 더 높은 가중치 부여. 과적합에 비교적 강하다.

Gradient Boosting

- ✓ 경사 하강법을 사용하여 이전 모델의 잔여 오차(residual error)를 학습.
- ✓ 높은 예측 성능을 보이지만 과적합 위험이 있고, 학습시간이 오래 걸림.
- ✓ 분류, 회귀에 모두 적용 가능

XGBoost

eXtremeGradientBoosting

- ✓ GradientBoosting의 문제점 개선
- ✓ 더 빠른 속도와 성능, 더 많은 기능 제공

LightGBM

- ✓ 대규모 데이터에서 효과적으로 학습
- ✓ 적은 데이터셋에 적용할 경우 과적합되기 쉽다.

CatBoost

- ✓ 범주형 변수 처리에 특화된 부스팅 알고리즘

Gradient Boosting의
변형 모델

분류 모델의 성능 평가 지표

Confusion Matrix	혼동행렬
Accuracy	정확도
Precision	정밀도
Recall (or Sensitivity)	재현율 (또는 민감도)
F1 Score	F1 점수
ROC/AUC	ROC곡선과 AUC

분류 모델의 성능 평가 지표

Confusion Matrix

혼동행렬

- 모델의 예측 결과를 실제 라벨과 비교하여 TP, FP, TN, FN의 개수를 집계한 표

		예측값	
		Negative(0)	Positive (1)
실제값	Negative(0)	TN True Negative	FP False Positive
	Positive (1)	FN False Negative	TP True Positive

- TN : Negative로 예측했는데 맞았음
- TP : Positive로 예측했는데 맞았음
- FP : Positive로 예측했는데 틀렸음
- FN : Negative로 틀렸는데 틀렸음

분류 모델의 성능 평가 지표

Accuracy

정확도

- 전체 예측 중 올바르게 예측한 비율

		예측값	
		Negative(0)	Positive (1)
실제값	Negative(0)	TN True Negative	FP False Positive
	Positive (1)	FN False Negative	TP True Positive

- $$\text{Accuracy} = \frac{TN+TP}{TN+TP+FN+FP}$$

분류 모델의 성능 평가 지표

Recall (or Sensitivity)

재현율(또는 민감도)

- 실제 Positive인 사례 중에서 모델이 올바르게 예측한 비율

		예측값	
		Negative(0)	Positive (1)
실제값	Negative(0)	TN True Negative	FP False Positive
	Positive (1)	FN False Negative	TP True Positive

- $\text{Recall(or Sensitivity)} = \frac{TP}{TP+FN}$
- FN를 줄이는 것이 중요한 상황에서 유용
(예) 암진단, 범죄자 식별

분류 모델의 성능 평가 지표

Precision

정밀도

- Positive로 예측한 사례 중 실제 Positive의 비율

		예측값	
		Negative(0)	Positive (1)
실제값	Negative(0)	TN True Negative	FP False Positive
	Positive (1)	FN False Negative	TP True Positive

- $$\text{Precision} = \frac{TP}{TP+FP}$$
- FP를 줄이는 것이 중요한 상황에서 유용
(예) 스팸 필터링, 광고타겟팅

분류 모델의 성능 평가 지표

F1 Score

F1 점수

- 정밀도와 재현율의 조화평균. 균형을 고려한다.

		예측값	
		Negative(0)	Positive (1)
실제값	Negative(0)	TN True Negative	FP False Positive
	Positive (1)	FN False Negative	TP True Positive

- $$F1\ Score = 2 \times \frac{Precision * Recall}{Precision + Recall}$$
- 값이 클수록 모델의 성능이 좋음을 의미

분류 모델의 성능 평가 지표

ROC curve

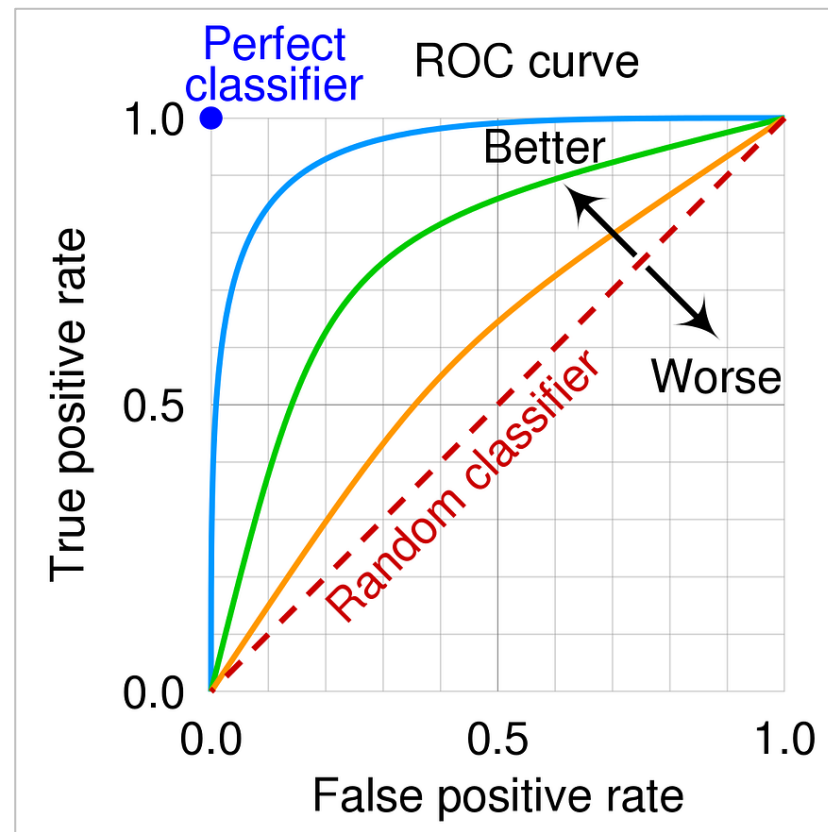
Receiver Operating Characteristic Curve

- 분류 모델의 성능을 다양한 임계값에서 시각적으로 평가하는 그래프
- X축 : FPR (False Positive Rate). 거짓 양성 비율.
 - ✓ 실제 음성인데 양성으로 잘못 예측한 비율
 - ✓ $FPR = \frac{FP}{FP+TN}$
- Y축 : TPR (True Positive Rate). 진짜 양성 비율
 - ✓ 실제 양성인데 모델이 양성으로 잘 예측한 비율
 - ✓ $TPR = \frac{TP}{TP+FN}$

AUC

Area Under the Curve

- ROC 곡선 아래의 면적으로, 모델의 분류 능력을 나타냄
 - ✓ 1: 완벽한 모델
 - ✓ 0.5 : 랜덤 추측과 동일한 모델
 - ✓ <0.5 : 랜덤 추측보다 못한 경우

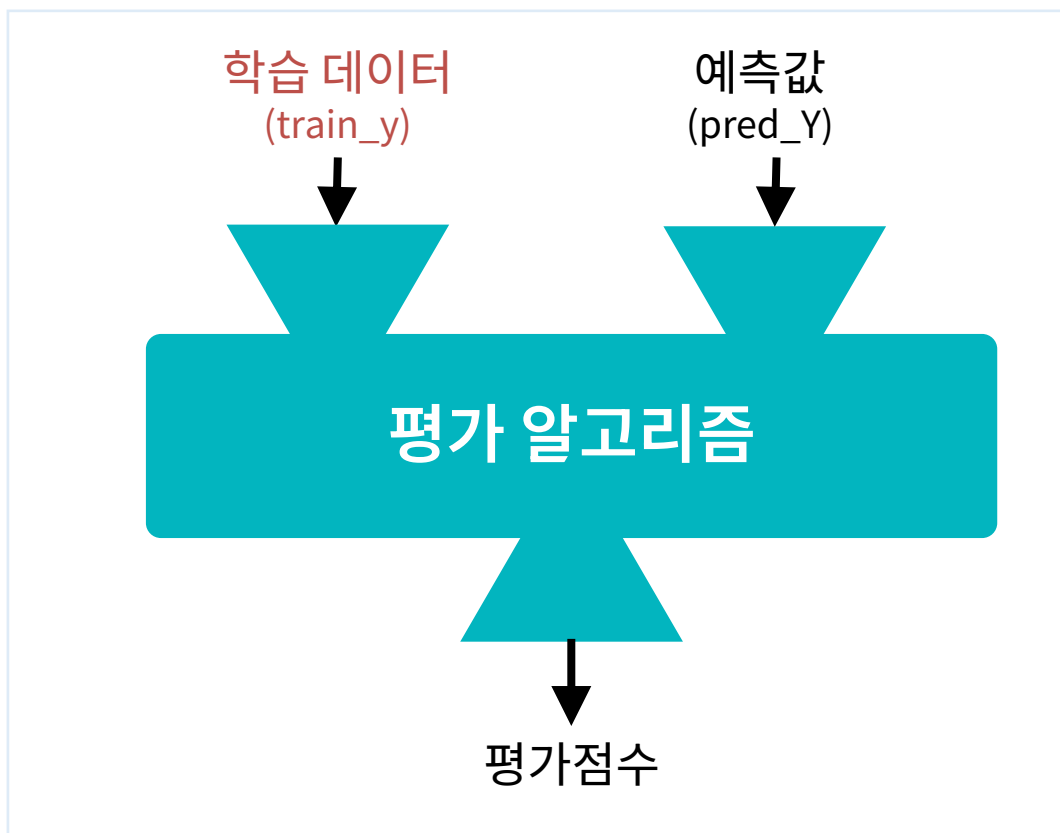


<https://medium.com/@ilyurek/roc-curve-and-auc-evaluating-model-performance-c2178008b02>

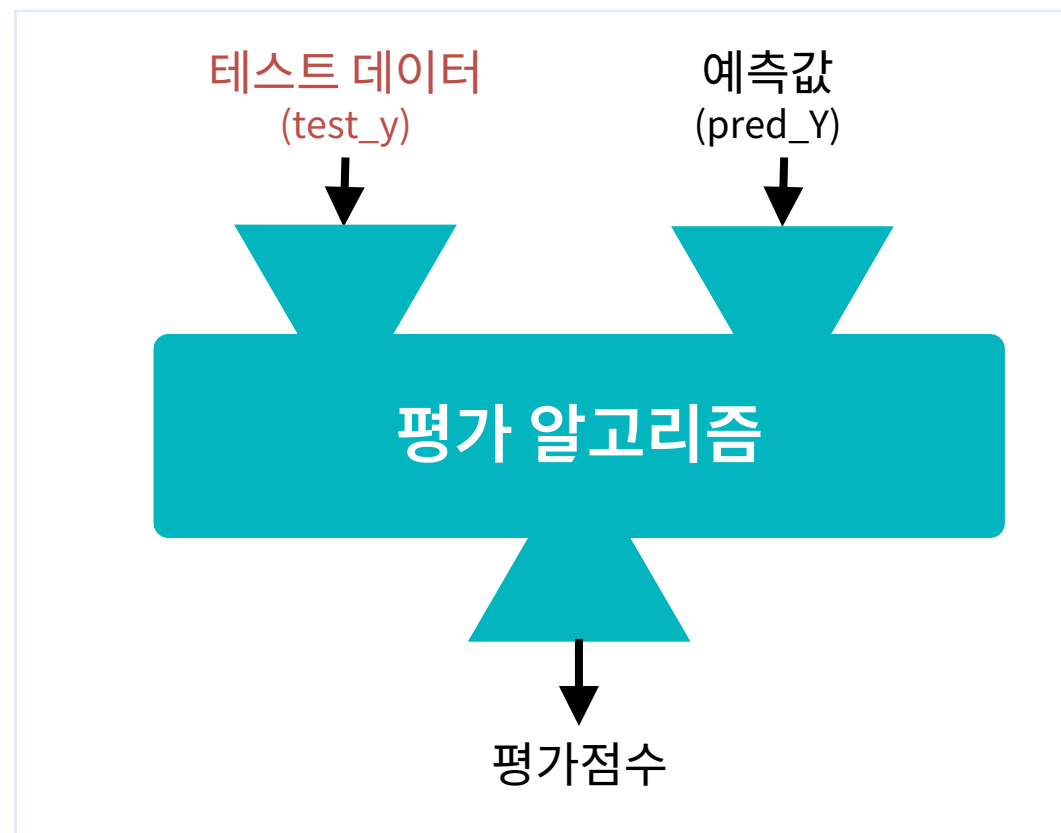
모델 평가

- 모델의 과적합 확인

- 학습데이터의 y 값과 모델의 예측값을 이용하여 평가

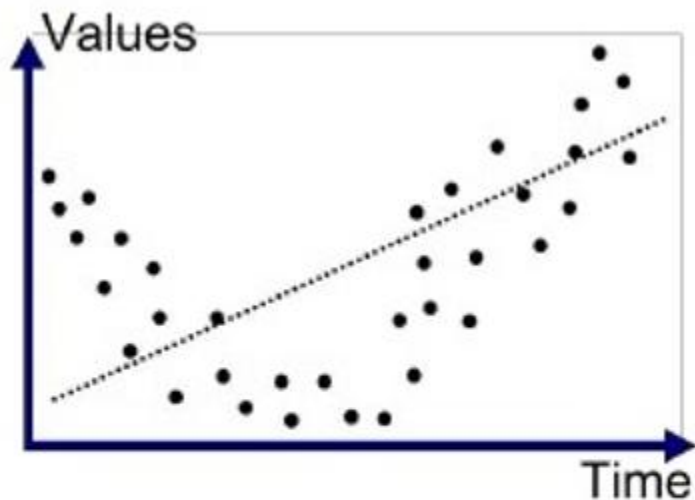


- 테스트데이터의 y 값과 모델의 예측값을 이용하여 평가

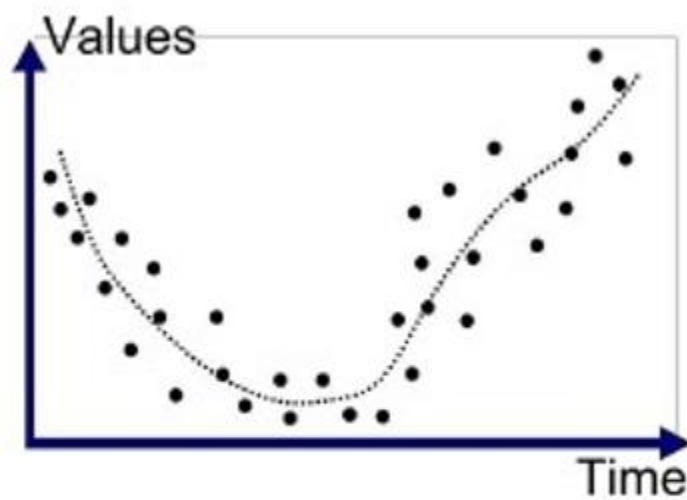


모델 평가

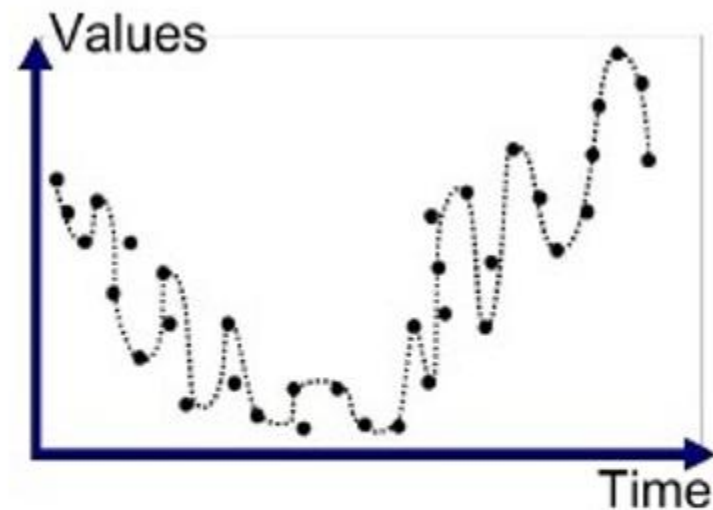
- 모델의 과적합 확인(회귀)
- 학습용/평가용 데이터로 각각 평가하여 과대적합/과소적합 여부를 확인한다.



Underfitted



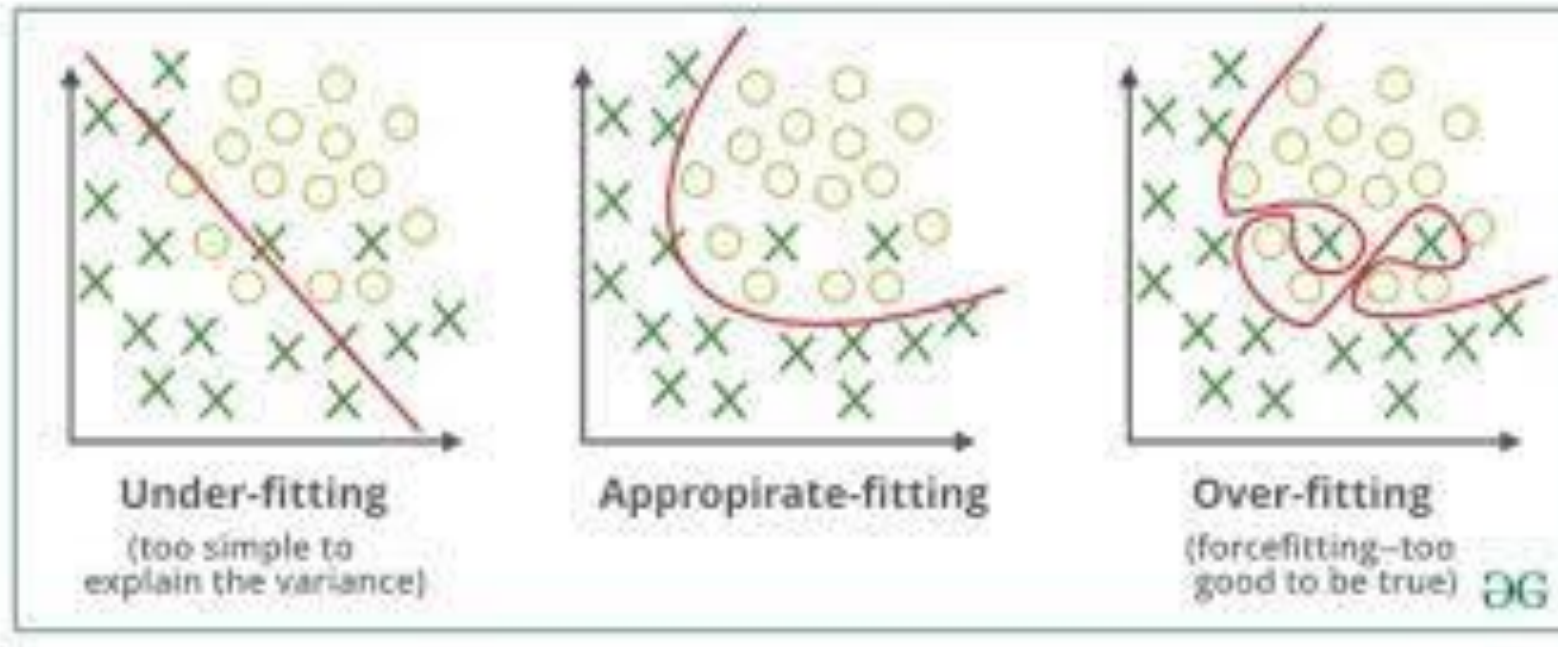
Good Fit/Robust



Overfitted

모델 평가

- 모델의 과적합 확인(분류)
- 학습용/평가용 데이터로 각각 평가하여 과대적합/과소적합 여부를 확인한다.



붓꽃 품종 예측

- 목적 -

붓꽃의 **특성**(꽃잎의 길이와 너비, 꽃받침의 길이와 너비)를 기반으로
붓꽃의 **품종**을 예측한다.



다중분류

■ feature와 target

독립변수

feature



- sepal length
- sepal width
- petal length
- petal width

종속변수

target

setosa



versicolor



verginica



데이터 준비하기

■ 데이터 불러오기

- sklearn에서 제공하는 샘플 데이터셋을 활용한다.

```
from sklearn.datasets import load_iris
```

```
# 붓꽃 데이터셋 로딩
```

```
iris = load_iris()
```

```
# 붓꽃 데이터셋의 구조 확인
```

```
# print(iris)
```

```
print(iris.keys())
```

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


데이터 준비하기

■ 독립변수 살펴보기

- feature : 독립변수

```
# feature 살펴보기
feature_name = iris.feature_names
feature = iris.data
```

```
print(feature_name)
print(feature)
```

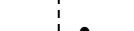
```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
[[5.1 3.5 1.4 0.2]
 [4.9 3. 1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5. 3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
```

-- 생략 --

- target : 종속변수

```
print(target_name)
print(target)
```

[illegible]

- 
- 0: 'setosa'
- 1: 'versicolor'
- 2: 'virginica'

데이터 준비하기

■ 데이터 설명 보기

```
# 데이터 설명 보기  
print(iris.DESCR)
```

```
.. _iris_dataset:
```

```
Iris plants dataset
```

```
-----
```

```
**Data Set Characteristics:**
```

```
:Number of Instances: 150 (50 in each of three classes)
```

```
:Number of Attributes: 4 numeric, predictive attributes and the class
```

```
:Attribute Information:
```

- sepal length in cm
- sepal width in cm
- petal length in cm
- petal width in cm

```
-- 생략 --
```

데이터 탐색하기

■ 데이터프레임 만들기

```
# 데이터프레임 만들기
import pandas as pd
df_iris = pd.DataFrame(feature,
                        columns = feature_name)
df_iris["species"] = target
df_iris.head(3)
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	label
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0

데이터 탐색하기

■ 데이터프레임 정보

```
df_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 (cm)     150 non-null   float64
 1   sepal width (cm)      150 non-null   float64
 2   petal length (cm)     150 non-null   float64
 3   petal width (cm)      150 non-null   float64
 4   species               150 non-null   int64
dtypes: float64(4), int64(1)
memory usage: 6.0 KB
```

null 없음
모든 데이터가 숫자로 되어있음

데이터 탐색하기

■ 종속변수 불균형 여부 확인

```
# species별 데이터 빈도수 확인  
df_iris['species'].value_counts()
```

```
species  
0 50  
1 50  
2 50  
Name: count, dtype: int64
```

클래스의 개수가 균등하다.

데이터 탐색하기

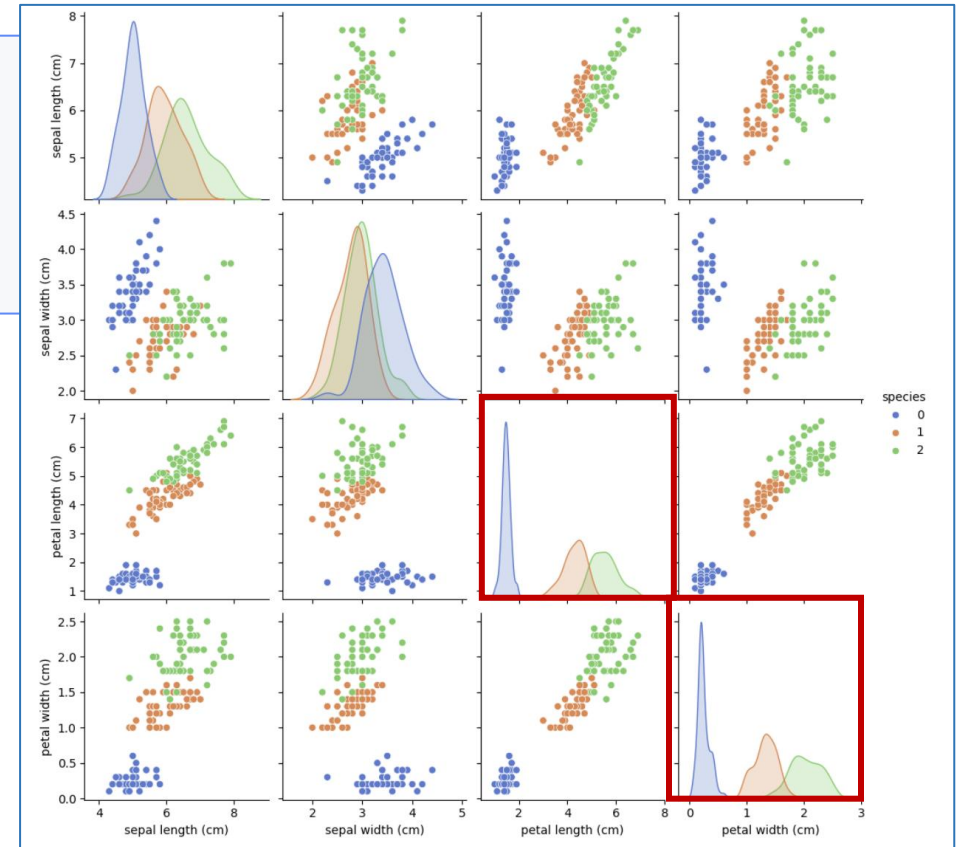
■ 변수간 관계 시각화

시각화

```
import matplotlib.pyplot as plt
import seaborn as sns
sns.pairplot(data=df_iris, hue="species", palette='muted')
plt.show()
```

pariplot

- 데이터셋의 변수 간의 관계와 데이터의 분포를 시각화 하는 도구
 - ✓ 데이터셋의 모든 숫자형 변수 쌍에 대한 산점도를 표시한다.
 - ✓ 대각선 부분에는 각 변수의 히스토그램 또는 커널 밀도 추정(kde) 그래프가 표시되어 해당 변수의 분포를 확인할 수 있다.
 - ✓ hue 파라미터를 사용하여 데이터셋의 특정 카테고리별로 색상을 구분해 시각화 할 수 있다.



데이터 전처리

■ 학습용 데이터와 테스트용 데이터 분할

- 데이터의 **feature**와 **target**을 전달하여 데이터 **분할**

```
from sklearn.model_selection import train_test_split

# 학습용 데이터와 테스트용 데이터 분리
X_train, X_test, y_train, y_test = train_test_split(feature
                                                    , target
                                                    , test_size=0.2
                                                    , random_state=42)

# 데이터의 크기 확인
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)

(120, 4) (30, 4) (120,) (30,)
```


데이터 전처리

■ 학습용 데이터와 테스트용 데이터 분할

■ 분할 결과 확인

```
print(pd.Series(y_train).value_counts())  
print(pd.Series(y_test).value_counts())
```

```
2    43  
0    40  
1    37  
Name: count, dtype: int64  
1    13  
0    10  
2     7  
Name: count, dtype: int64
```

클래스의 개수가 균등하지 않게 나누어짐.

데이터 전처리

■ 학습용 데이터와 테스트용 데이터 분할

- 훈련세트와 테스트세트의 클래스 분포를 원본 클래스의 분포와 동일하게 맞추어 분할

```
from sklearn.model_selection import train_test_split

# 학습용 데이터와 테스트용 데이터 분리
X_train, X_test, y_train, y_test = train_test_split(feature
                                                    , target
                                                    , test_size=0.2
                                                    , random_state=42
                                                    , stratify=target)

# 데이터의 크기 확인
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

(120, 4) (30, 4) (120,) (30,)

- **stratify**: 특정 변수의 비율을 유지하며 데이터를 분할한다.

데이터 전처리

■ 학습용 데이터와 테스트용 데이터 분할

■ 분할 결과 확인

```
print(pd.Series(y_train).value_counts())  
print(pd.Series(y_test).value_counts())
```

```
0    40  
2    40  
1    40  
Name: count, dtype: int64  
2    10  
0    10  
1    10  
Name: count, dtype: int64
```

모델링

■ 모델 생성 및 훈련

■ 훈련세트로 훈련

```
# DecisionTreeClassifier 모델 생성
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier(random_state=42)

# 모델 학습
model.fit(X_train, y_train)
```

▼ DecisionTreeClassifier ⓘ ?
DecisionTreeClassifier(random_state=1)

※ 사이킷런의 API 일관성 및 개발 편의성 ①

: 사이킷런에서 지도 학습을 위한 모든 주요 알고리즘은 **fit()** 메서드를 사용하여 **학습**을 수행한다.

모델 성능평가

■ 예측하기

- 테스트용 데이터의 feature를 이용하여 예측

```
# 테스트용 데이터로 예측
pred = model.predict(X_test)
print("예측label>>",pred)
print("실제label>>",y_test)
```

```
예측label>> [0 2 1 1 0 1 0 0 2 1 2 2 2 1 0 0 0 1 1 2 0 2 1 1 2 2 1 0 2 0]
실제label>> [0 2 1 1 0 1 0 0 2 1 2 2 2 1 0 0 0 1 1 2 0 2 1 2 2 1 1 0 2 0]
```

잘못된 예측

※ 사이킷런의 API일관성 및 개발 편의성②

: 사이킷런에서 지도 학습을 위한 모든 주요 알고리즘은
predict() 메서드를 사용하여 **예측**을 수행한다.

모델 성능평가

■ 모델의 정확도 평가

- 실제 **label**과 모델의 예측한 **label**을 이용하여 모델의 정확도를 평가한다.

```
# 정확도 평가
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, pred)
print(f"예측정확도 >> {accuracy:.2f}")
```

예측정확도 >> 0.93

- **sklearn.metrics**: 모델의 평가와 관련된 도구를 제공하는 사이킷런 모듈
- **accuracy_score**: 분류 모델에서 모델의 정확도를 계산하는 함수
- **accuracy(정확도)**: 전체 데이터 중 모델이 맞춘 데이터의 비율

모델 성능평가

■ 분류 모델의 평가 지표

- 실제 **label**과 모델의 **예측한 label**을 이용하여 모델의 정확도를 평가한다.

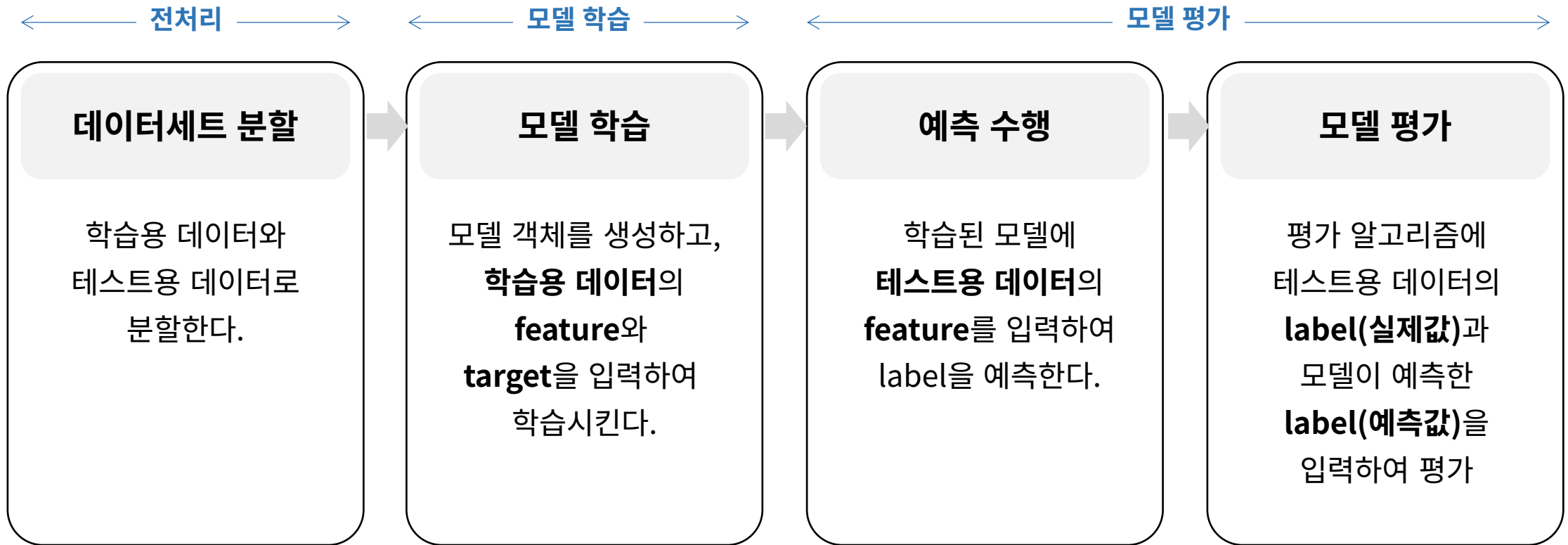
분류 모델의 평가지표

```
from sklearn.metrics import classification_report
print(classification_report(y_test, pred))
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	10
1	0.90	0.90	0.90	10
2	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

모델 성능평가

■ 프로세스 정리

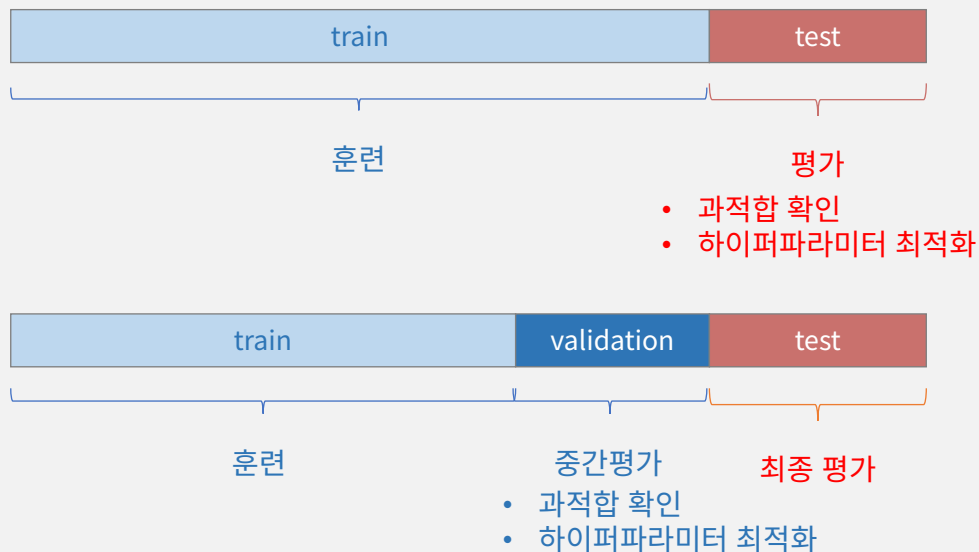


모델의 평가 방법

홀드아웃 검증

Holdout method

전체 데이터를 훈련용/테스트용으로
또는 훈련용/검증용/테스트용으로 분할



k-폴드 교차검증

k-Fold Cross-Validation

학습 데이터를 k개의 동일한 크기의 부분집합으로 나누고,
k-1개의 집단을 학습용으로, 나머지는 검증용으로 설정하여 학습
k번 반복 측정된 결과를 평균 낸 값을 검증 결과값으로 사용



KFold 교차검증

■ KFold 객체 생성

- 데이터를 k개의 폴드로 나누기 위한 설정

```
# KFold객체 생성
from sklearn.model_selection import KFold
kf = KFold(n_splits=5, shuffle=True, random_state=42)
```

- **KFold** : kFold 교차검증을 위한 클래스
- **n_split** : 데이터를 몇 개로 나눌지 설정
- **shuffle** : 데이터를 나누기 전 무작위로 섞을지 여부
- **random_state** : random seed

KFold 교차검증

■ 교차검증 수행

- k개의 fold로 분할하여 k번 반복

```
for tr_idx, val_idx in kf.split(X_train):  
    print(f'tr_idx >> {tr_idx}')  
    print(f'val_idx >> {val_idx}')
```

```
tr_idx >> [ 1 2 3 5 6 7 8 9 12 13 14 15 16 17 19 20 21 22 23 24 25 27 28 29 30 32 33 34 35 37 38 39 41 42 43 46 48 49 50 51 52 53  
54 56 57 58 59 60 61 63 66 67 68 69 71 72 74 75 76 77 78 79 80 81 82 83 84 85 86 87 90 92 93 94 95 96 97 98 99 100 101 102 103 105  
106 108 110 111 112 113 114 115 116 117 118 119]  
val_idx >> [ 0 4 10 11 18 26 31 36 40 44 45 47 55 62 64 65 70 73 88 89 91 104 107 109]  
tr_idx >> [ 0 1 2 3 4 6 7 8 10 11 13 14 16 17 18 19 20 21 23 26 27 29 31 32 34 35 36 37 38 39 40 41 43 44 45 46 47 48 49 50 51 52  
54 55 57 58 59 60 61 62 63 64 65 66 67 68 70 71 72 73 74 75 77 79 80 81 82 83 84 86 87 88 89 91 92 93 94 95 99 100 101 102 103 104  
105 106 107 108 109 111 112 113 115 116 117 119]  
val_idx >> [ 5 9 12 15 22 24 25 28 30 33 42 53 56 69 76 78 85 90 96 97 98 110 114 118]  
-- 생략 --
```

- **kFold.split(features)**
 - 데이터를 k개의 폴더로 분할하는 제너레이터 반환
 - 이 제너레이터는 각 반복 시 훈련세트와 테스트세트에 해당하는 인덱스 반환한다.
 - 반복을 통해 접근할 수 있다.

KFold 교차검증

■ 교차검증 수행

- 학습을 위한 데이터, 검증을 위한 데이터 준비

```
for tr_idx, val_idx in kf.split(X_train):  
    X_tr, y_tr = X_train[tr_idx], y_train[tr_idx] # 학습용  
    X_val, y_val = X_train[val_idx], y_train[val_idx] # 검증용  
    print(X_tr.shape, X_val.shape, y_tr.shape, y_val.shape)
```

```
(120, 4) (30, 4) (120,) (30,)  
(120, 4) (30, 4) (120,) (30,)  
(120, 4) (30, 4) (120,) (30,)  
(120, 4) (30, 4) (120,) (30,)  
(120, 4) (30, 4) (120,) (30,)
```

KFold 교차검증

■ 교차검증 수행

▪ 훈련, 예측, 평가

```
for train_index, test_index in kf.split(feature):  
  
    X_train, y_train = feature[train_index], target[train_index] # 학습용 데이터  
    X_test, y_test = feature[test_index], target[test_index] # 검증용 데이터  
  
    model.fit(X_train, y_train) # 훈련  
    pred = model.predict(X_test) # 예측  
    accuracy = accuracy_score(y_test, pred) # 평가  
    print(f'accuracy >> {accuracy}')
```

```
accuracy:0.9166666666666666  
accuracy:0.8333333333333334  
accuracy:0.9166666666666666  
accuracy:0.9583333333333334  
accuracy:0.875
```

KFold 교차검증

■ 교차검증 수행

▪ 정확도의 평균 계산

```
cv_scores = []
for tr_idx, val_idx in kf.split(X_train):
    X_tr, y_tr = X_train[tr_idx], y_train[tr_idx]    # 학습용
    X_val, y_val = X_train[val_idx], y_train[val_idx] # 검증용

    model.fit(X_tr, y_tr)
    pred = model.predict(X_val)
    accuracy = accuracy_score(y_val, pred)
    cv_scores.append(accuracy)

print(f'cv_scores:{np.round(cv_scores,4)}')
print(f'cv_scores_mean:{np.mean(cv_scores):.4f}')
```

```
cv_scores:[0.9167 0.8333 0.9167 0.9583 0.875 ]
cv_scores_mean:0.9000
```

StratifiedKFold 교차검증

■ KFold vs StratifiedKFold

- Kfold로 교차검증을 수행하는 경우, label데이터의 분포가 불균칙적이다.

```
cv_scores = []
for tr_idx, val_idx in kf.split(X_train):
    X_tr, y_tr = X_train[tr_idx], y_train[tr_idx]    # 학습용
    X_val, y_val = X_train[val_idx], y_train[val_idx] # 검증용
    print(pd.Series(y_val).value_counts())

    model.fit(X_tr, y_tr)
    pred = model.predict(X_val)
    accuracy = accuracy_score(y_val, pred)
    cv_scores.append(accuracy)

print(f'cv_scores:{cv_scores}')
print(f'cv_scores_mean:{np.mean(cv_scores)}')
```

```
1    10
0     8
2     6
Name: count, dtype: int64
0    10
2     8
1     6
Name: count, dtype: int64
0    12
1     6
2     6
Name: count, dtype: int64
1    13
2     9
0     2
Name: count, dtype: int64
2    11
0     8
1     5
Name: count, dtype: int64
```

StratifiedKFold 교차검증

■ KFold vs StratifiedKFold

- StratifiedKFold를 사용하면, 원본 label 데이터의 분포와 동일하게 데이터를 분할한다.

```
# StratifiedKFold 객체 생성
from sklearn.model_selection import StratifiedKFold
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
```

- **StratifiedKFold**
 - 분류 모델에서는 StratifiedKFold, 회귀모델에서는 KFold를 사용
 - 클래스 별 데이터의 수가 불균형한 데이터인 경우 유용하게 사용된다. (예:사기탐지)

StratifiedKFold 교차검증

■ KFold vs StratifiedKFold

▪ StratifiedKFold 교차검증 수행

```
cv_scores = []
for tr_idx, val_idx in skf.split(X_train, y_train):
    X_tr, y_tr = X_train[tr_idx], y_train[tr_idx]    # 학습용
    X_val, y_val = X_train[val_idx], y_train[val_idx] # 검증용
    print(pd.Series(y_val).value_counts())

    model.fit(X_tr, y_tr)
    pred = model.predict(X_val)
    accuracy = accuracy_score(y_val, pred)
    cv_scores.append(accuracy)

print(f'cv_scores:{np.round(cv_scores,4)}')
print(f'cv_scores_mean:{np.mean(cv_scores):.4f}')
```

```
cv_scores:[0.9583 0.9583 0.9583 0.9583 0.9167]
cv_scores_mean:0.9500
```

```
0    8
1    8
2    8
Name: count, dtype: int64
0    8
1    8
2    8
Name: count, dtype: int64
0    8
1    8
2    8
Name: count, dtype: int64
0    8
1    8
2    8
Name: count, dtype: int64
0    8
1    8
2    8
Name: count, dtype: int64
```

cross_val_score()

■ 교차검증을 간편하게 하기 위한 API

```
cross_val_score( estimator,  
                 X,  
                 y=None,  
                 *,  
                 groups=None,  
                 scoring=None,  
                 cv=None,  
                 n_jobs=None,  
                 verbose=0,  
                 fit_params=None,  
                 params=None,  
                 pre_dispatch='2*n_jobs',  
                 error_score=nan)
```

- **cross_val_score** 주요 파라미터
 - ✓ **estimator** : 모델객체
 - ✓ **X** : feature
 - ✓ **y** : label
 - ✓ **scoring** : 평가지표
 - ✓ **cv** : 교차검증방식

cross_val_score()

■ 교차검증 수행

- cross_val_score()을 이용한 교차검증 수행

```
# 교차검증 수행
from sklearn.model_selection import cross_val_score
cv_scores = cross_val_score(model,
                             X = X_train,
                             y = y_train,
                             scoring='accuracy',
                             cv=skf)

print(f'cv_scores : {np.around(cv_scores,4)}')
print(f'cv_scores_mean : {cv_scores.mean():.4f}')

cv_scores : [0.9583 0.9583 0.9583 0.9583 0.9167]
cv_scores_mean : 0.9500
```

cross_val_score()

■ 교차검증 수행

▪ 테스트데이터로 최종 평가

```
# 훈련데이터 전체로 훈련, 테스트데이터로 최종 평가
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'test accuracy : {accuracy:.4f}')
```

```
test accuracy : 0.9333
```

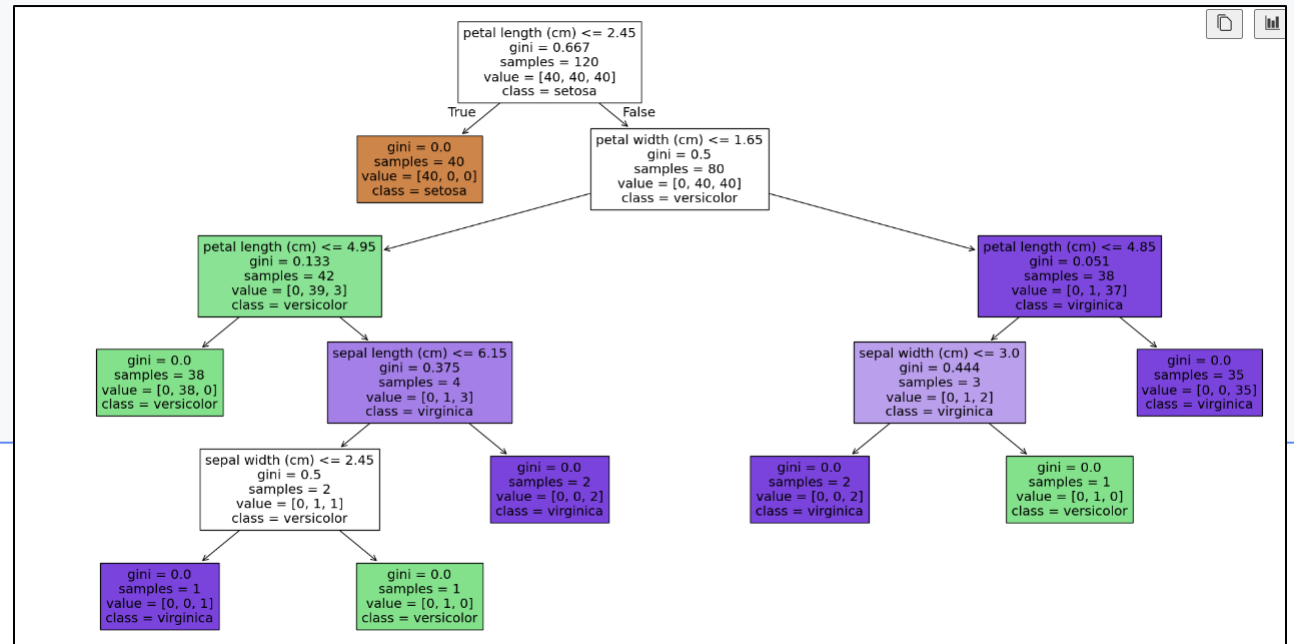
모델의 훈련 결과

■ 의사결정트리 시각화

■ 트리 시각화

트리 시각화

```
from sklearn.tree import plot_tree
plt.figure(figsize=(20,10))
plot_tree(model,
          feature_names=feature_name,
          class_names=target_name,
          filled=True)
plt.tight_layout()
plt.show()
```



모델의 훈련 결과

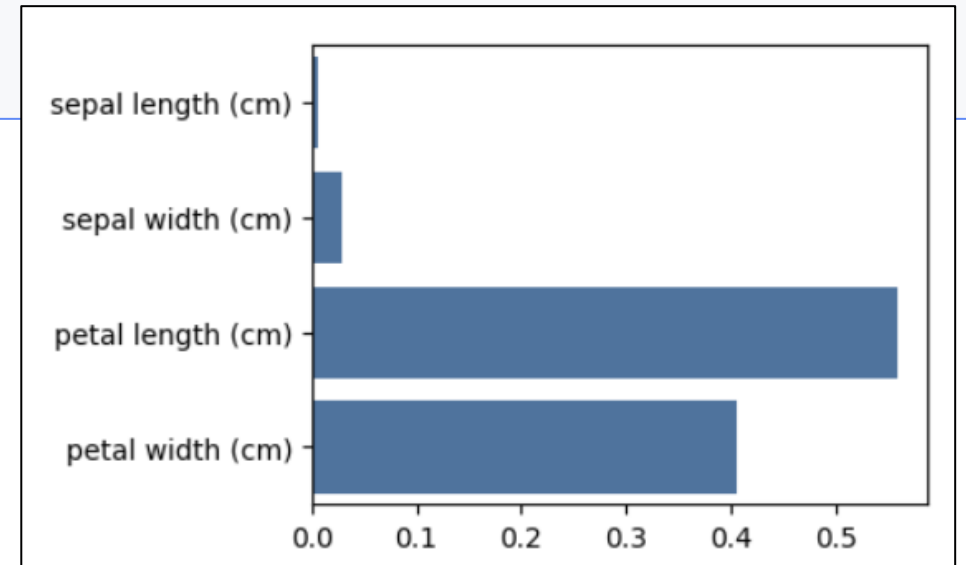
■ 특성의 중요도

▪ 특성의 중요도

```
# 특성의 중요도
print(f'특성의 중요도 >>> {model.feature_importances_}')

plt.figure(figsize=(4,3))
sns.barplot(y=feature_name, x=model.feature_importances_)
plt.show()
```

특성의 중요도 >>> [0.00625 0.02916667 0.5585683 0.40601504]



모델의 훈련 결과

■ 특성의 중요도

- 트리의 최종 깊이

```
print(model.get_depth())
```

5

하이퍼파라미터 튜닝

■ GridSearchCV

- 지정한 하이퍼파라미터를 순차적으로 변경하면서 교차검증을 수행하여 최적의 파라미터 조합을 찾는 방법

```
grid_parameters = {'max_depth':[3,4,5],  
                  'min_sample_split':[2,4]  
}
```

순번	max_depth	min_sample_split
1	3	2
2	3	4
3	4	2
4	4	4
5	5	2
6	5	4



하이퍼파라미터 조합

하이퍼파라미터 튜닝

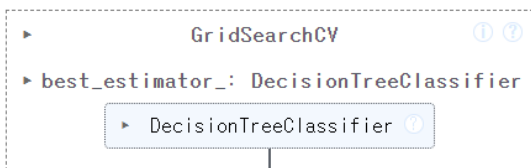
■ GridSearchCV

- GridSeachCV 객체 생성(모델, 하이퍼파라미터그리드, 교차검증횟수)
→ GridSearchCV 객체에 학습용 데이터 전달하여 Grid Search Cross Validation 수행

```
# 하이퍼파라미터 준비
parameters = {'max_depth':[3,4,5],
              'min_samples_split':[2,4]
}

# GridSearchCV 객체 생성
from sklearn.model_selection import GridSearchCV
gscv = GridSearchCV(model, param_grid=parameters, cv=kf, refit=True)

# 하이퍼파라미터를 순차적으로 변경하면서 학습/평가 수행
gscv.fit(X_train, y_train)
```



※ **refit=True**
최적의 하이퍼파라미터를 찾은 후 이를 사용하여 모델을 전체 데이터셋에 대해 재학습시켜 **최종 모델을 만든다.**
 이렇게 재학습된 모델은 GridSearchCV 객체의 **best_estimator_** 속성을 통해 접근할 수 있다.

하이퍼파라미터 튜닝

■ GridSearchCV

■ 그리드서치 결과 확인

```
# 그리드서치 결과 확인
display(pd.DataFrame(gscv.cv_results_))
print(f'최적의 파라미터 >>> {gscv.best_params_}')
print(f'최고 정확도 >>> {gscv.best_score_:.4f}')
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	param_min_samples_split	params	split0_test_score	split1_test_score	split2_test_score	split3_test_score	split4_test_score	mean_test_score	std_test_score	rank_test_score
0	0.000201	0.000402	0.000103	0.000206	3	2	{'max_depth': 3, 'min_samples_split': 2}	0.916667	0.833333	0.958333	1.000000	0.875000	0.916667	0.058926	2
1	0.000000	0.000000	0.000000	0.000000	3	4	{'max_depth': 3, 'min_samples_split': 4}	0.916667	0.833333	0.958333	1.000000	0.875000	0.916667	0.058926	2
2	0.000000	0.000000	0.000000	0.000000	4	2	{'max_depth': 4, 'min_samples_split': 2}	0.916667	0.833333	0.916667	0.958333	0.875000	0.900000	0.042492	6
3	0.002623	0.005245	0.000079	0.000159	4	4	{'max_depth': 4, 'min_samples_split': 4}	0.916667	0.833333	0.958333	1.000000	0.875000	0.916667	0.058926	2
4	0.000000	0.000000	0.000000	0.000000	5	2	{'max_depth': 5, 'min_samples_split': 2}	0.916667	0.833333	0.916667	0.958333	0.916667	0.908333	0.040825	5
5	0.000000	0.000000	0.000000	0.000000	5	4	{'max_depth': 5, 'min_samples_split': 4}	0.916667	0.833333	0.958333	1.000000	0.916667	0.925000	0.055277	1

최적의 파라미터 >>> {'max_depth': 5, 'min_samples_split': 4}

최고 정확도 >>> 0.9250

하이퍼파라미터 튜닝

■ GridSearchCV

- 테스트데이터로 최종 평가

```
# 최종 모델
best_model = gscv.best_estimator_

# 예측
pred = best_model.predict(X_test)

# 평가
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, pred)
print(f'테스트세트 정확도 >> {accuracy:.4f}')
```

테스트세트 정확도 >> 0.9333

로켓발사

- 목적 -

로켓 발사 성공여부 예측 모델 구현

데이터 준비하기

■ 라이브러리 불러오기

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

데이터 준비하기

■ 데이터 불러오기

```
rocket = pd.read_csv('data/RocketLaunchDataCSV.csv')
rocket.head(3)
```

	Name	Date	Time (East Coast)	Location	Crewed or Uncrewed	Launched?	High Temp	Low Temp	Ave Temp	Temp at Launch Time	...	Max Wind Speed	Visibility	Wind Speed at Launch Time	Hist Ave Max Wind Speed	Hist Ave Visibility	Sea Level Pressure	Hist Ave Sea Level Pressure	Day Length	Condition	Notes
0	NaN	04-Dec-58	NaN	Cape Canaveral	NaN	NaN	75.0	68.0	71.00	NaN	...	16.0	15.0	NaN	NaN	NaN	30.22	NaN	10:26	Cloudy	NaN
1	NaN	05-Dec-58	NaN	Cape Canaveral	NaN	NaN	78.0	70.0	73.39	NaN	...	14.0	10.0	NaN	NaN	NaN	30.2	NaN	10:26	Cloudy	NaN
2	Pioneer 3	06-Dec-58	1:45	Cape Canaveral	Uncrewed	Y	73.0	0.0	60.21	62.0	...	15.0	10.0	11.0	NaN	NaN	30.25	NaN	10:25	Cloudy	NaN

데이터 탐색 및 전처리

■ 데이터프레임 정보

```
rocket.info()
```

#	Column	Non-Null Count	Dtype
0	Name	60 non-null	object
1	Date	300 non-null	object
2	Time (East Coast)	59 non-null	object
3	Location	300 non-null	object
4	Crewed or Uncrewed	60 non-null	object
5	Launched?	60 non-null	object
6	High Temp	299 non-null	float64
7	Low Temp	299 non-null	float64
8	Ave Temp	299 non-null	float64
9	Temp at Launch Time	59 non-null	float64
10	Hist High Temp	299 non-null	float64
11	Hist Low Temp	299 non-null	float64
12	Hist Ave Temp	299 non-null	float64
13	Percipitation at Launch Time	299 non-null	float64
14	Hist Ave Percipitation	299 non-null	float64
15	Wind Direction	299 non-null	object
16	Max Wind Speed	299 non-null	float64
17	Visibility	299 non-null	float64
18	Wind Speed at Launch Time	59 non-null	float64
19	Hist Ave Max Wind Speed	0 non-null	float64
20	Hist Ave Visibility	0 non-null	float64
21	Sea Level Pressure	299 non-null	object
22	Hist Ave Sea Level Pressure	0 non-null	float64
23	Day Length	298 non-null	object
24	Condition	298 non-null	object
25	Notes	3 non-null	object

dtypes: float64(15), object(11)
memory usage: 61.1+ KB

데이터 탐색 및 전처리

■ 자료형 변환

- Sea Level Pressure(해수면 기압) → float

```
# 변환할 수 없는 값은 null처리
rocket['Sea Level Pressure'] = pd.to_numeric(rocket['Sea Level Pressure'], errors='coerce')
rocket['Sea Level Pressure']
```

```
0      30.22
1      30.20
2      30.25
3      30.28
4      30.23
...
295    30.08
296    30.05
297    30.03
298    30.01
299    30.08
```

```
Name: Sea Level Pressure, Length: 300, dtype: float64
```


데이터 탐색 및 전처리

■ 자료형 변환

- Sea Level Pressure(해수면 기압) → float

```
def time_to_decimal(time_str):  
    try:  
        # 시간과 분을 분리  
        hours, minutes = map(int, time_str.split(':'))  
        # 24시간 기준으로 소수점 변환 (시간 + 분/60)  
        return hours + minutes/60  
    except:  
        return None
```

컬럼에 적용

```
rocket['Day Length'] = rocket['Day Length'].apply(time_to_decimal)
```

확인

```
print("변환된 값 샘플:", rocket['Day Length'].head())  
print("데이터 타입:", rocket['Day Length'].dtype)
```

```
변환된 값 샘플: 0      10.433333  
1      10.433333  
2      10.416667  
3      10.416667  
4      12.400000  
Name: Day Length, dtype: float64  
데이터 타입: float64
```

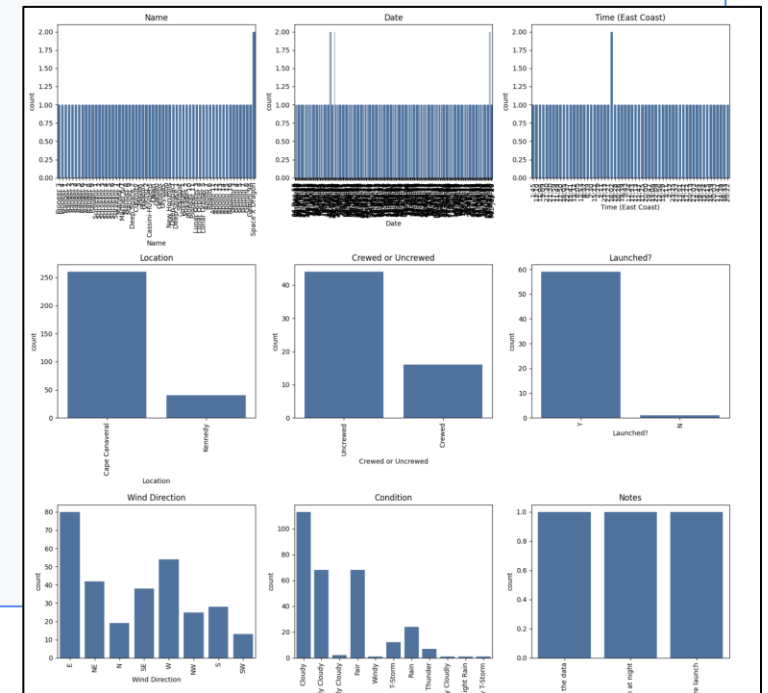
데이터 탐색 및 전처리

■ 데이터 탐색

- object 타입 변수의 데이터 분포 확인

```
# 데이터타입이 object인 컬럼 추출
object_cols = rocket.select_dtypes(include=['object']).columns
```

```
# 데이터 분포 확인
plt.figure(figsize=(15,20))
i = 1
for col in object_cols:
    plt.subplot(3,3,i)
    i+=1
    sns.countplot(data=rocket, x=col)
    plt.xticks(rotation=90)
    plt.title(col)
plt.tight_layout()
```



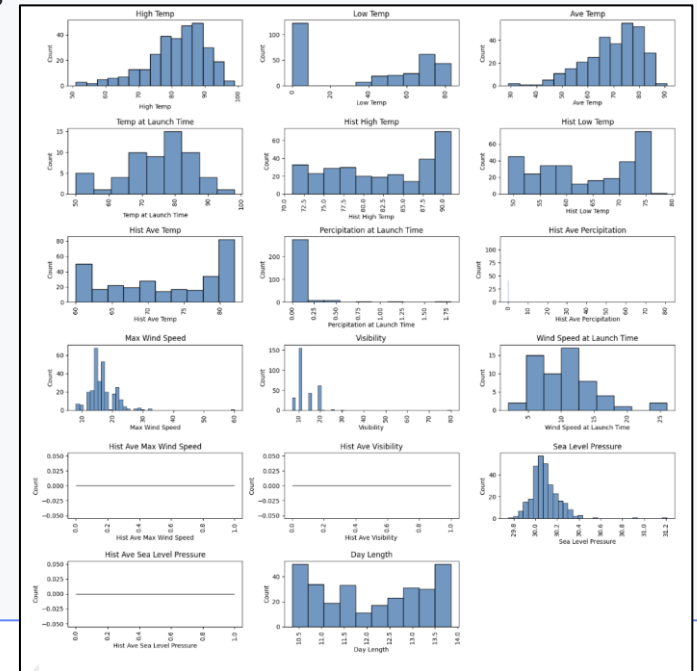
데이터 탐색 및 전처리

■ 데이터 탐색

- 데이터 타입이 숫자인 컬럼의 분포 확인

```
# 데이터타입이 int, float인 컬럼 추출
numeric_cols = rocket.select_dtypes(include=['int','float']).columns
```

```
# 데이터 분포 확인
plt.figure(figsize=(15,15))
i = 1
for col in numeric_cols:
    plt.subplot(6,3,i)
    i+=1
    sns.histplot(data=rocket, x=col)
    plt.xticks(rotation=90)
    plt.title(col)
plt.tight_layout()
```



데이터 탐색 및 전처리

■ 데이터 전처리

▪ 불필요한 컬럼 제거

사용하지 않을 컬럼

```
del_cols = ['Name', 'Date', 'Time (East Coast)', 'Notes', \
            'Percipitation at Launch Time', 'Hist Ave Percipitation', \
            'Hist Ave Max Wind Speed', 'Hist Ave Visibility', 'Hist Ave Sea Level Pressure']
```

```
df = rocket.drop(columns = del_cols)
```

```
df.head()
```

	Location	Crewed or Uncrewed	Launched?	High Temp	Low Temp	Ave Temp	Temp at Launch Time	Hist High Temp	Hist Low Temp	Hist Ave Temp	Wind Direction	Max Wind Speed	Visibility	Wind Speed at Launch Time	Sea Level Pressure	Day Length	Condition
0	Cape Canaveral	NaN	NaN	75.0	68.0	71.00	NaN	75.0	55.0	65.0	E	16.0	15.0	NaN	30.22	10.433333	Cloudy
1	Cape Canaveral	NaN	NaN	78.0	70.0	73.39	NaN	75.0	55.0	65.0	E	14.0	10.0	NaN	30.20	10.433333	Cloudy
2	Cape Canaveral	Uncrewed	Y	73.0	0.0	60.21	62.0	75.0	55.0	65.0	NE	15.0	10.0	11.0	30.25	10.416667	Cloudy
3	Cape Canaveral	NaN	NaN	76.0	57.0	66.04	NaN	75.0	55.0	65.0	N	10.0	10.0	NaN	30.28	10.416667	Partly Cloudy
4	Cape Canaveral	NaN	NaN	79.0	60.0	70.52	NaN	75.0	55.0	65.0	E	12.0	10.0	NaN	30.23	12.400000	Partly Cloudy

데이터 탐색 및 전처리

■ 데이터 전처리

■ 결측치 확인 및 처리

```
df.isnull().sum()
```

Location	0
Crewed or Uncrewed	240
Launched?	240
High Temp	1
Low Temp	1
Ave Temp	1
Temp at Launch Time	241
Hist High Temp	1
Hist Low Temp	1
Hist Ave Temp	1
Wind Direction	1
Max Wind Speed	1
Visibility	1
Wind Speed at Launch Time	241
Sea Level Pressure	2
Day Length	2
Condition	2

dtype: int64

데이터 탐색 및 전처리

■ 데이터 전처리

▪ 결측치 확인 및 처리

```
# Crewed or Uncrewed --> 'Uncrewed'로 채우기  
df['Crewed or Uncrewed'] = df['Crewed or Uncrewed'].fillna('Uncrewed')
```

```
# Launched? --> 'N'으로 채우기  
df['Launched?'] = df['Launched?'].fillna('N')
```

```
# object타입 변수의 결측값 채우기 (최빈값으로 채우기)  
obj_cols = df.select_dtypes(include=['object']).columns  
df[obj_cols] = df[obj_cols].fillna(df[obj_cols].mode().iloc[0])
```

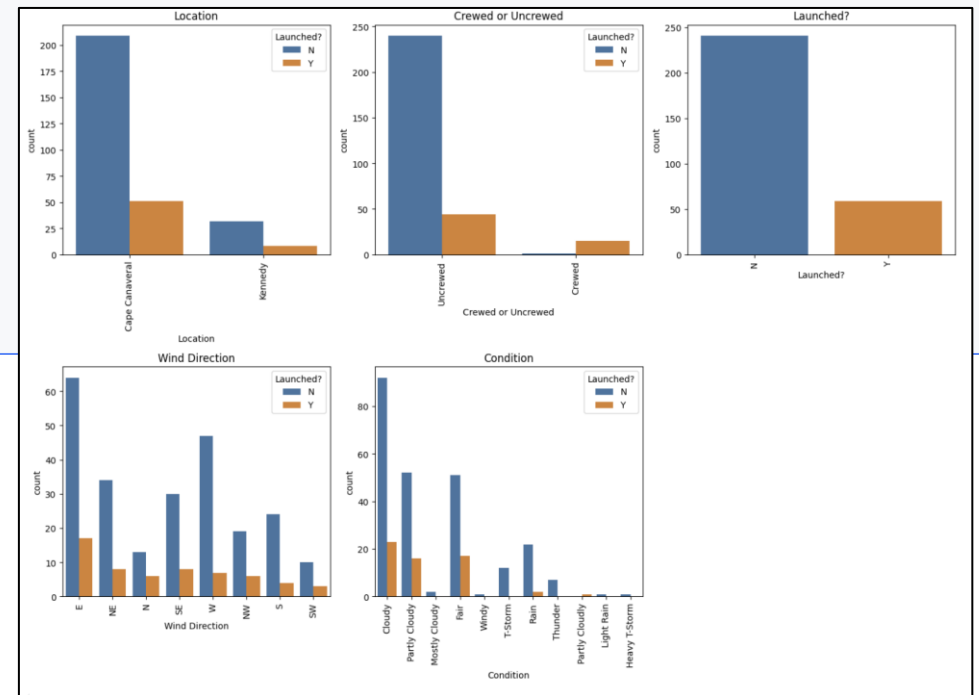
```
# 수치형 타입 변수의 결측값 채우기 (평균값으로 채우기)  
numeric_cols = df.select_dtypes(include=['float64']).columns  
df[numeric_cols] = df[numeric_cols].fillna(df[numeric_cols].mean())
```

데이터 탐색 및 전처리

■ 전처리된 데이터 확인

- 데이터 분포 및 독립변수/종속변수 관계 시각화

```
plt.figure(figsize=(15,15))
i=1
for col in obj_cols:
    plt.subplot(3,3,i)
    i+=1
    sns.countplot(data=df, x=col, hue='Launched?')
    plt.xticks(rotation=90)
    plt.title(col)
plt.tight_layout()
```

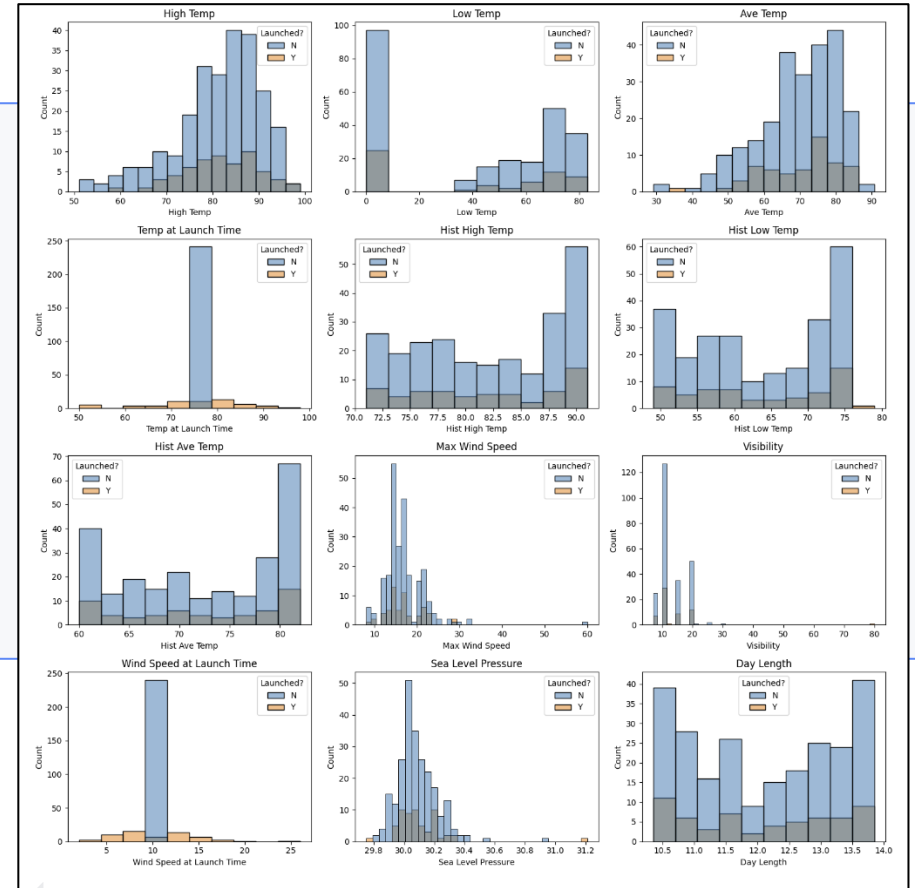


데이터 탐색 및 전처리

■ 전처리된 데이터 확인

■ 데이터 분포 및 독립변수/종속변수 관계 시각화

```
plt.figure(figsize=(15,15))
i=1
for col in numeric_cols:
    plt.subplot(4,3,i)
    i+=1
    sns.histplot(data=df, x=col, hue='Launched?')
    plt.title(col)
plt.tight_layout()
```



데이터 탐색 및 전처리

■ 변수 선택

```
X = df.drop(columns=['Launched?'])  
y = df['Launched?']
```

데이터 탐색 및 전처리

■ 종속변수 인코딩

- $Y \rightarrow 1, N \rightarrow 0$

```
y = y.map({'Y':1, 'N':0})  
y.value_counts()
```

데이터 탐색 및 전처리

■ 독립변수 레이블 인코딩

```
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()

obj_cols = X.select_dtypes(include=['object'])
for col in obj_cols:
    X[col] = encoder.fit_transform(df[col])

X.head(3)
```

	Location	Crewed or Uncrewed	High Temp	Low Temp	Ave Temp	Temp at Launch Time	Hist High Temp	Hist Low Temp	Hist Ave Temp	Wind Direction	Max Wind Speed	Visibility	Wind Speed at Launch Time	Sea Level Pressure	Day Length	Condition
0	0	1	75.0	68.0	71.00	75.101695	75.0	55.0	65.0	0	16.0	15.0	10.59322	30.22	10.433333	0
1	0	1	78.0	70.0	73.39	75.101695	75.0	55.0	65.0	0	14.0	10.0	10.59322	30.20	10.433333	0
2	0	1	73.0	0.0	60.21	62.000000	75.0	55.0	65.0	2	15.0	10.0	11.00000	30.25	10.416667	0

데이터 탐색 및 전처리

■ 훈련세트/테스트세트 분할

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42, stratify=y)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(225, 16) (75, 16) (225,) (75,)
```

모델링

■ 모델 생성 및 훈련

```
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier(random_state=42)
model.fit(X_train, y_train)

print(f'test accuracy score >>> {model.score(X_test, y_test)}')
print(f'train accuracy score >>> {model.score(X_train, y_train)}')
```

```
test accuracy score >>> 0.9866666666666667
train accuracy score >>> 1.0
```

■ 분류 모델의 평가지표

```
pred_test = model.predict(X_test)
pred_train = model.predict(X_train)

from sklearn.metrics import classification_report
print(f'''test score >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
      {classification_report(y_test, pred_test)}''')
print(f'''train score >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
      {classification_report(y_train, pred_train)}''')
```

```
test score >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
              precision    recall  f1-score   support

     0         1.00        0.98        0.99         60
     1         0.94        1.00        0.97         15

 accuracy          0.99         75
 macro avg         0.97        0.99        0.98         75
weighted avg         0.99        0.99        0.99         75

train score >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
              precision    recall  f1-score   support

     0         1.00        1.00        1.00        181
     1         1.00        1.00        1.00         44

 accuracy          1.00        225
 macro avg         1.00        1.00        1.00        225
weighted avg         1.00        1.00        1.00        225
```

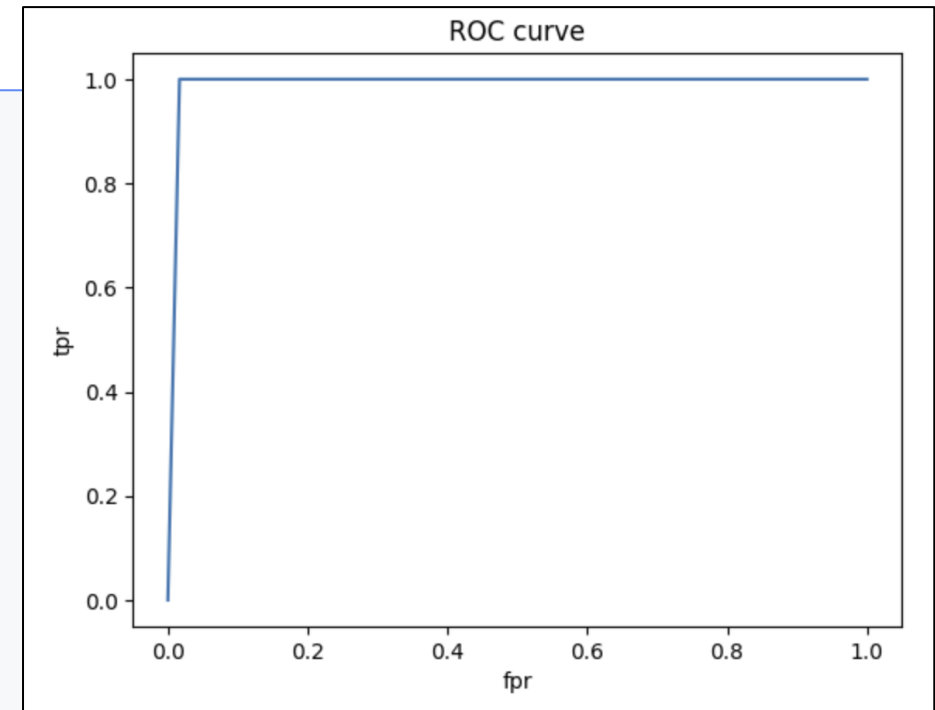
■ 분류 모델의 평가지표

```
#ROC curve와 AUC
from sklearn.metrics import roc_curve, roc_auc_score
pred_proba = model.predict_proba(X_test)

# ROC curve
fpr, tpr, thresholds = roc_curve(y_test, pred_proba[:,1])
plt.plot(fpr, tpr)
plt.xlabel('fpr')
plt.ylabel('tpr')
plt.title('ROC curve')

# AUC
auc = roc_auc_score(y_test, pred_proba[:,1])
print(f'AUC : {auc:.4f}')
```

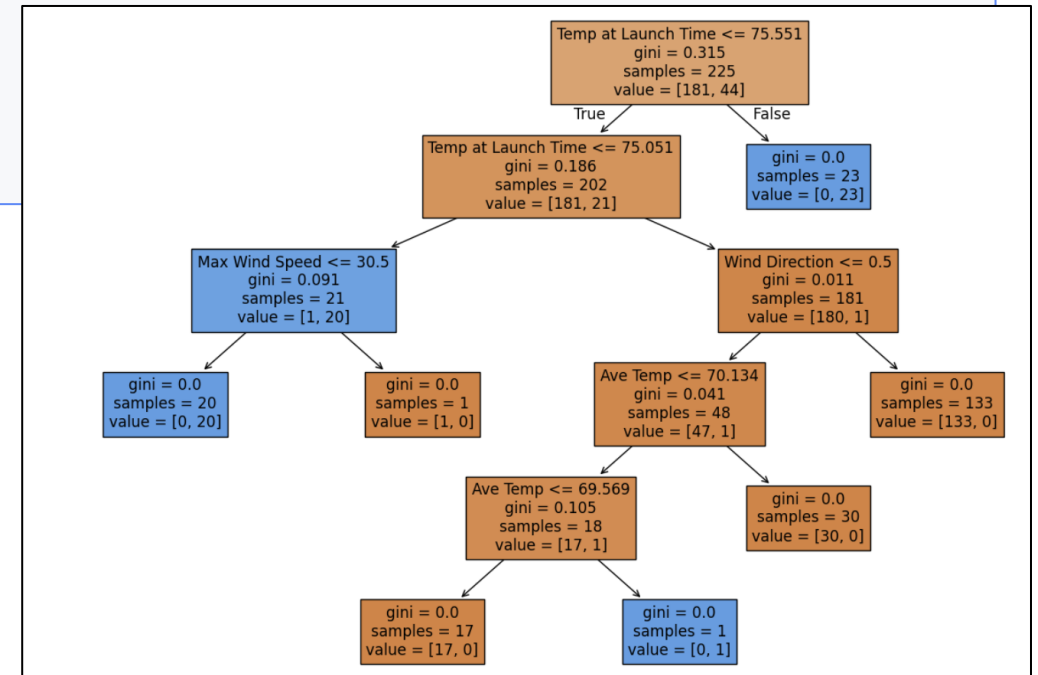
AUC : 0.9917



■ 모델의 학습 결과

시각화

```
from sklearn.tree import plot_tree
plt.figure(figsize=(15,10))
plot_tree(model, feature_names=X.columns, filled=True)
plt.show()
```



■ 모델의 학습 결과

```
# 트리의 깊이  
model.get_depth()
```

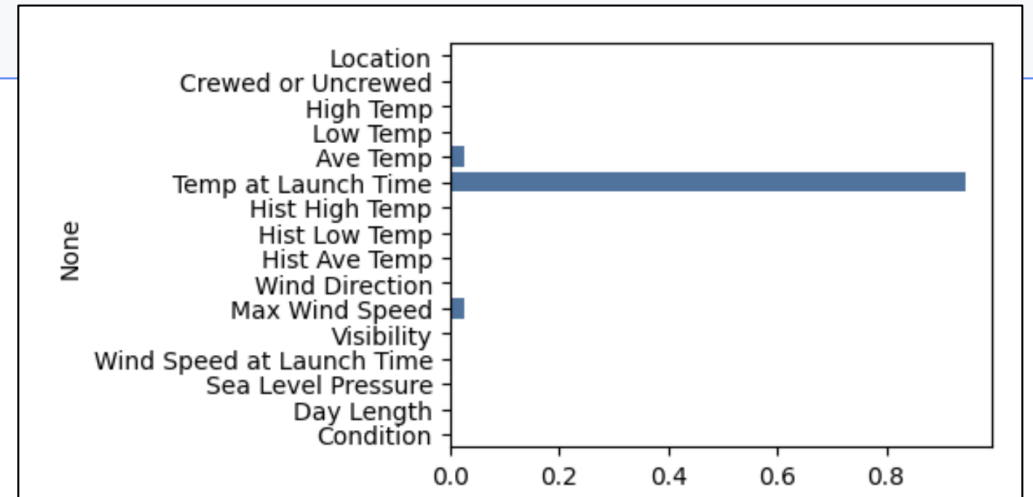
5

모델링

■ 모델의 학습 결과

```
# 특성의 중요도
print(f'특성의 중요도 >>> {model.feature_importances_}')

plt.figure(figsize=(4,3))
sns.barplot(y=X.columns, x=model.feature_importances_)
plt.show()
```



모델링

1. 클래스 불균형 데이터의 경우 accuracy보다는 precision, recall, f1 점수를 보아야 함
2. 과대적합 발생. 해결할 수 있을까?

■ 변수 재 선택하여 다시 학습 및 평가

```
X = X.drop(['Wind Speed at Launch Time', 'Temp at Launch Time', 'Crewed or Uncrewed'], axis=1)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
```

```
model = DecisionTreeClassifier(random_state=42)
model.fit(X_train, y_train)
```

```
pred_train = model.predict(X_train)
pred_test = model.predict(X_test)
```

```
print(f'test>>{classification_report(y_test, pred_test)}')
print(confusion_matrix(y_test, pred_test))
print(f'train>>{classification_report(y_train, pred_train)}')
```

```
test>>
```

	precision	recall	f1-score	support
0	0.78	0.78	0.78	60
1	0.13	0.13	0.13	15
accuracy			0.65	75
macro avg	0.46	0.46	0.46	75
weighted avg	0.65	0.65	0.65	75

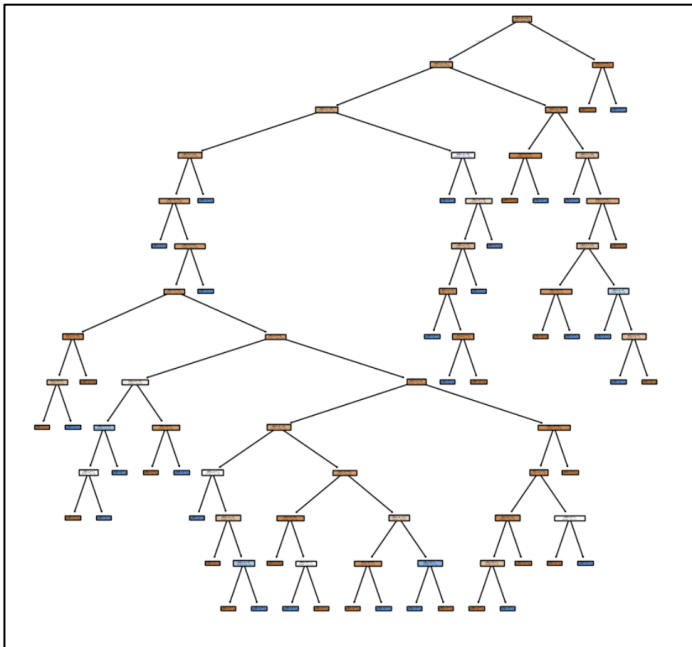
```
[[47 13]
 [13  2]]
train>>
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	181
1	1.00	1.00	1.00	44
accuracy			1.00	225
macro avg	1.00	1.00	1.00	225
weighted avg	1.00	1.00	1.00	225

모델링

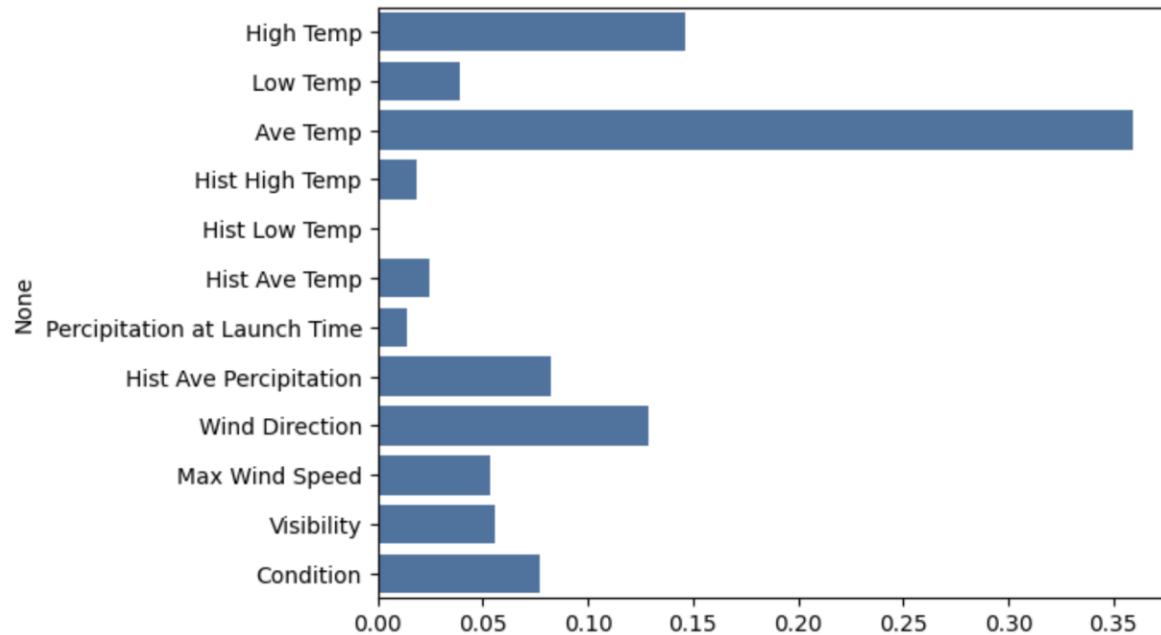
■ 모델의 학습 결과

```
plt.figure(figsize=(5,5))  
plot_tree(model, feature_names=X.columns, filled=True)  
plt.show()
```



■ 모델의 학습 결과

```
sns.barplot(y=X.columns , x=model.feature_importances_)
```



■ 모델의 학습 결과

```
model.get_depth()
```

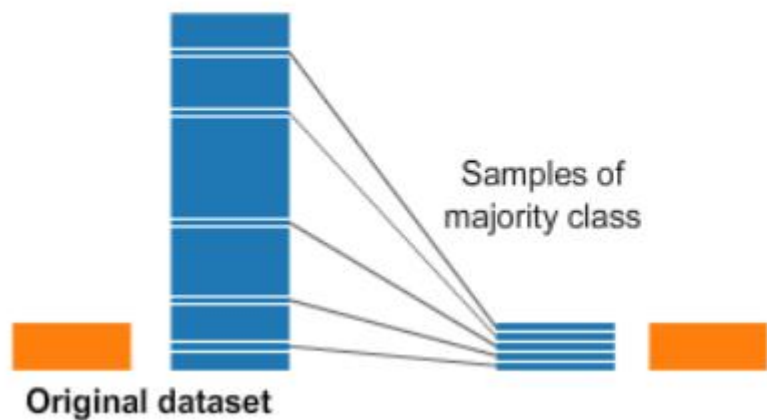
13

하이퍼파라미터 튜닝으로 과적합 해결이 가능할까?

불균형데이터 처리

■ 클래스 불균형 문제 해결 방안

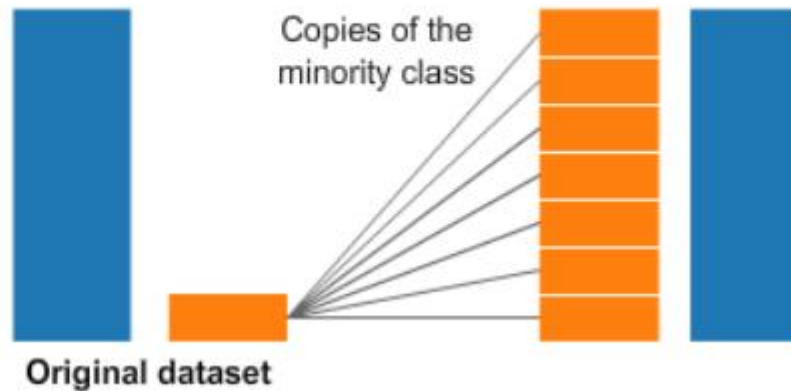
Undersampling



다수 클래스 데이터를 줄인다.

- Tomek Links : 가까운 이웃을 제거
- NearMiss : 소수 근처의 다수 데이터 제거
- Random Undersampling : 무작위 제거

Oversampling



소수 클래스 데이터를 증가시킨다.

- SMOTE : 데이터를 합성하여 증가
- ADASYN : SMOTE의 변형
- Random Oversampling : 무작위 복제

불균형데이터 처리

■ 오버샘플링

▪ 오버샘플링

pip install imblearn

```
# 오버샘플링
from imblearn.over_sampling import RandomOverSampler
# 오버샘플링 적용
ros = RandomOverSampler(random_state=42)
X_train_resampled, y_train_resampled = ros.fit_resample(X_train, y_train)
print(X_train_resampled.shape, y_train_resampled.shape)
print(y_train_resampled.value_counts())
```

(362, 12) (362,)

Launched?

1 181

0 181

Name: count, dtype: int64

불균형데이터 처리

■ 오버샘플링

▪ 재학습 및 평가

```
model = DecisionTreeClassifier(random_state=42, max_depth=4)
model.fit(X_train_resampled, y_train_resampled)
```

```
pred_test = model.predict(X_test)
pred_train = model.predict(X_train)
print(classification_report(y_test, pred_test))
print(classification_report(y_train, pred_train))
```

	precision	recall	f1-score	support
0	0.79	0.38	0.52	60
1	0.20	0.60	0.30	15
accuracy			0.43	75
macro avg	0.49	0.49	0.41	75
weighted avg	0.67	0.43	0.47	75

	precision	recall	f1-score	support
0	1.00	0.46	0.63	181
1	0.31	1.00	0.48	44
accuracy			0.57	225
macro avg	0.66	0.73	0.55	225
weighted avg	0.87	0.57	0.60	225

다른 모델 적용

■ 랜덤포레스트

```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(random_state=42, max_depth=4)
model.fit(X_train_resampled, y_train_resampled)

pred_test = model.predict(X_test)
pred_train = model.predict(X_train)
print(classification_report(y_test, pred_test))
print(classification_report(y_train, pred_train))
```

	precision	recall	f1-score	support
0	0.78	0.63	0.70	60
1	0.15	0.27	0.20	15
accuracy			0.56	75
macro avg	0.46	0.45	0.45	75
weighted avg	0.65	0.56	0.60	75

	precision	recall	f1-score	support
0	0.96	0.75	0.84	181
1	0.46	0.89	0.61	44
accuracy			0.78	225
macro avg	0.71	0.82	0.73	225
weighted avg	0.87	0.78	0.80	225

다른 모델 적용

■ k최근접이웃

```
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier(n_neighbors=20)
model.fit(X_train_resampled, y_train_resampled)

pred_test = model.predict(X_test)
pred_train = model.predict(X_train)
print(classification_report(y_test, pred_test))
print(classification_report(y_train, pred_train))
```

	precision	recall	f1-score	support
0	0.80	0.53	0.64	60
1	0.20	0.47	0.28	15
accuracy			0.52	75
macro avg	0.50	0.50	0.46	75
weighted avg	0.68	0.52	0.57	75

	precision	recall	f1-score	support
0	0.86	0.52	0.65	181
1	0.25	0.64	0.35	44
accuracy			0.55	225
macro avg	0.55	0.58	0.50	225
weighted avg	0.74	0.55	0.59	225