04. 분류 (1-5단원)

16기 분석 이지혜

CONTENTS

분류란?

기존 데이터가 어떤 class에 속하는지 패턴을 알고리즘으로 이해한 후 새롭게 관측되는 데이터에 대한 class를 판단

- Decision Tree
: 데이터 균일도에 따른 규칙 판단
- Ensemble
: 서로 다른/같은 머신러닝 알고리즘의 결합

Decision Tree 배경

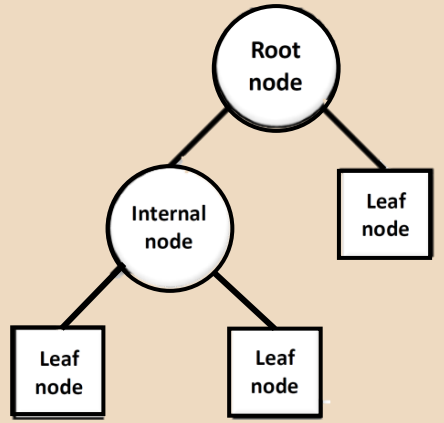
CART

(Classification And Regression Tree)

회귀와 분류 모두에 사용 가능하기 때문

분류 – 범주형 변수 / 회귀 – 연속형 변수

용어



장점

1. 쉽고 유연, 직관적
2. Scaling / Normalization의 영향이 적음

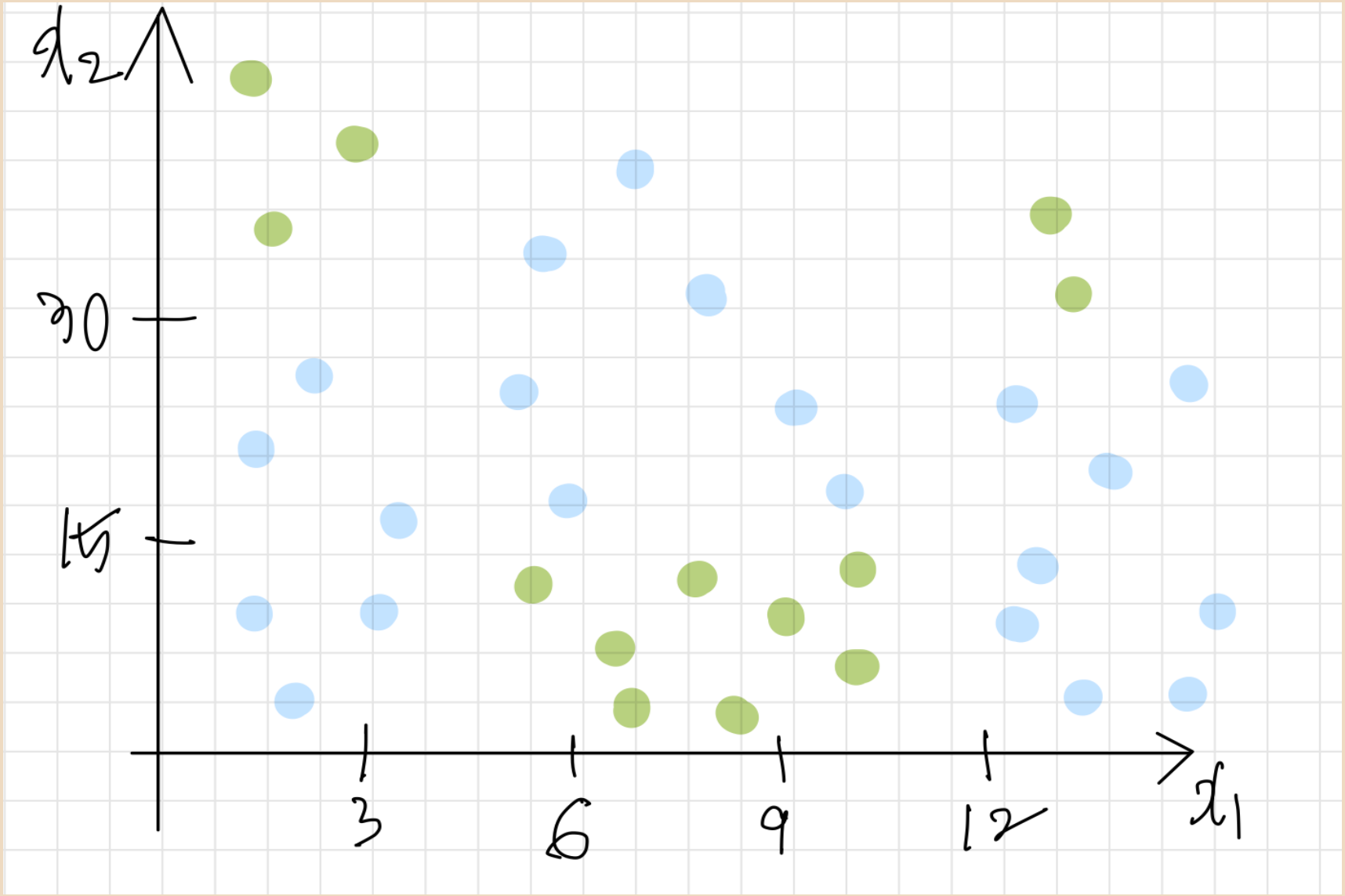
단점

1. Overfitting 발생 위험

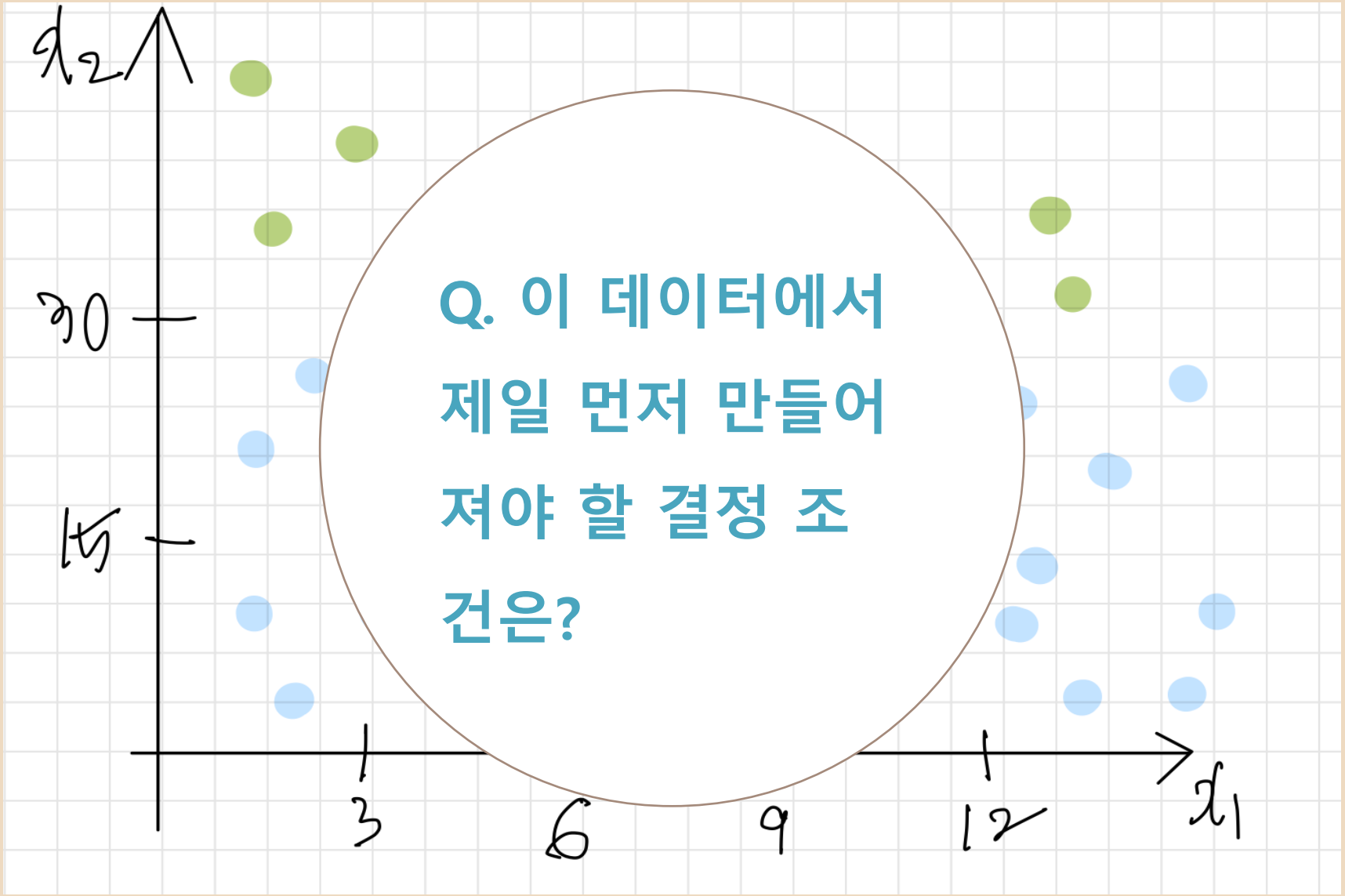
   -> Ensemble에서는 장점으로 작용

2. 샘플에 민감

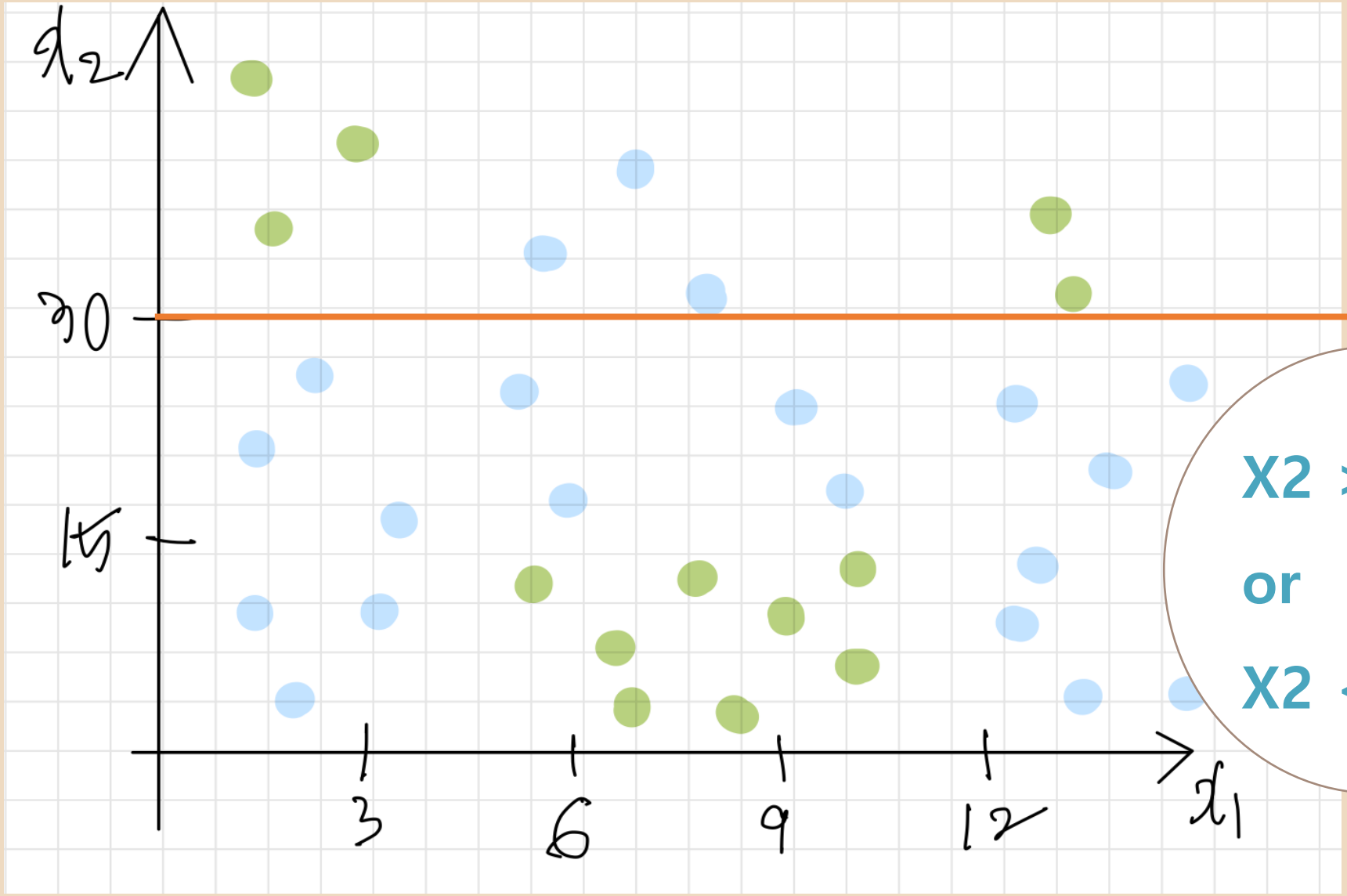
Decision Tree의 결정 조건



Decision Tree의 결정 조건



Decision Tree의 결정 조건

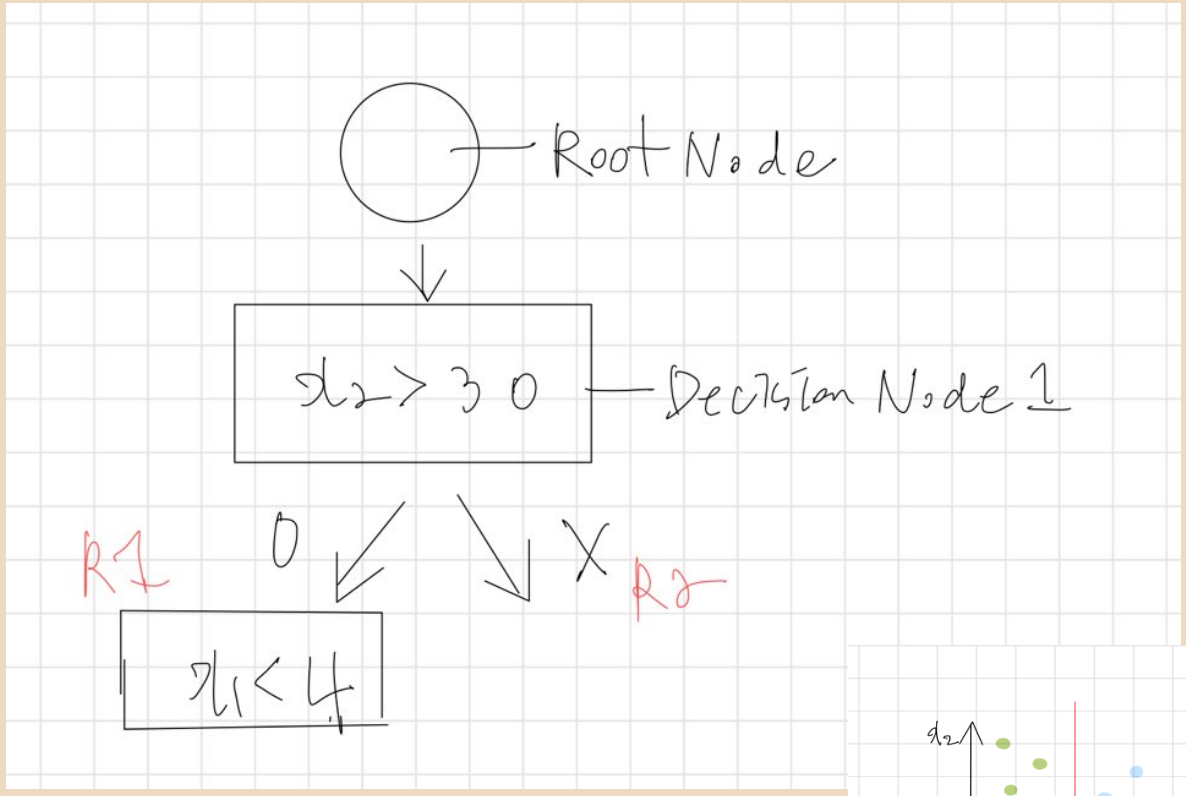


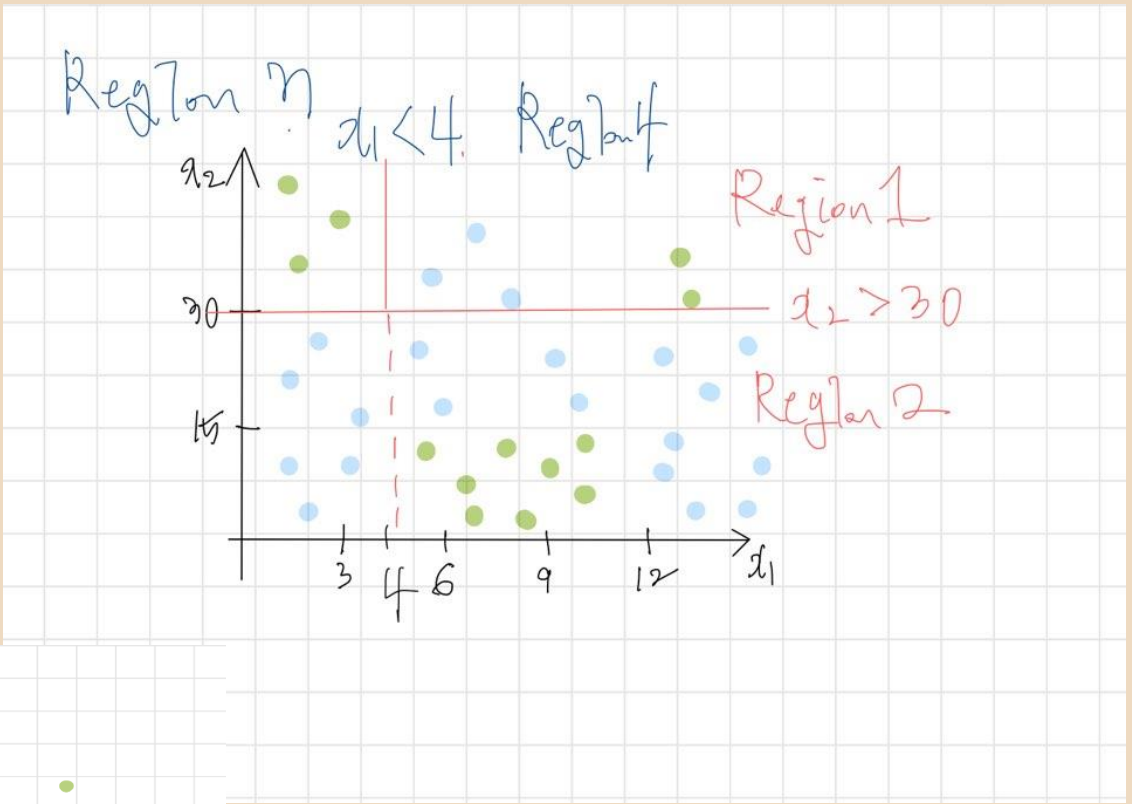
X2 > 30

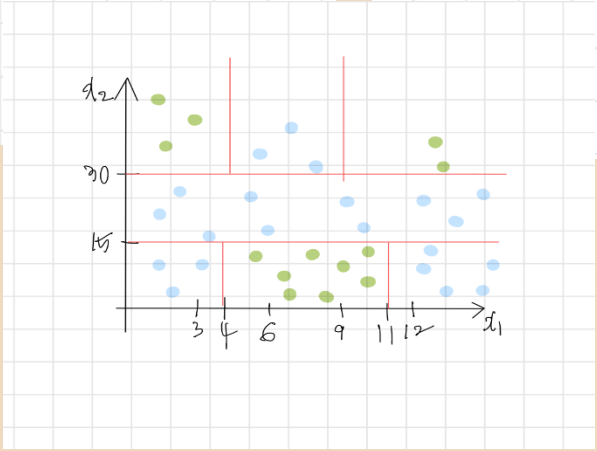
or

X2 <= 30

Decision Tree의 결정 조건







Decision Tree의 결정 조건

- High Depth = Low Accuracy

-> 최대한 많은 데이터셋이 해당 분류에 해당하도록 하는 규칙 설정이 필요

-> 최대한 균일한 데이터셋을 만드는 것이 필요

- Entropy vs Gini Impurity vs Misclassification Loss

Entropy vs Gini Impurity vs Misqualification Loss

	Entropy (엔트로피)	Gini Impurity (지니 불순도)	Misqualification Loss (오분류율)
해당 속성(Feature)으로의 분할 기준	높을 때	낮을 때	낮을 때
의미	- 주어진 데이터 집합의 혼잡도 - 균일도와 반비례하는 개념 - 0 또는 1일 확률이 최소, 0.5인 확률이 최대가 되도록 하는 함수	- 확률분포가 어느 쪽으로 치우쳤는지 측정 - 균일도와 반비례하는 개념	- 해당 클래스를 잘못 예측한 확률
수식	$H(E) = \begin{cases} -\sum_{i \in N} P_{i,E} \log_2 P_{i,E} & P_{i,E} \neq 0 \text{ for all } i \\ 0 & \text{otherwise} \end{cases}$	$\text{Gini Index} = 1 - \sum_{i=1}^{n} (P_i)^2$	$classification\ loss = 1 - \max(p_j)$

Decision Tree를 사용한 분류 예측

- 각 노드는 그 노드를 선택한 데이터 집합을 가짐

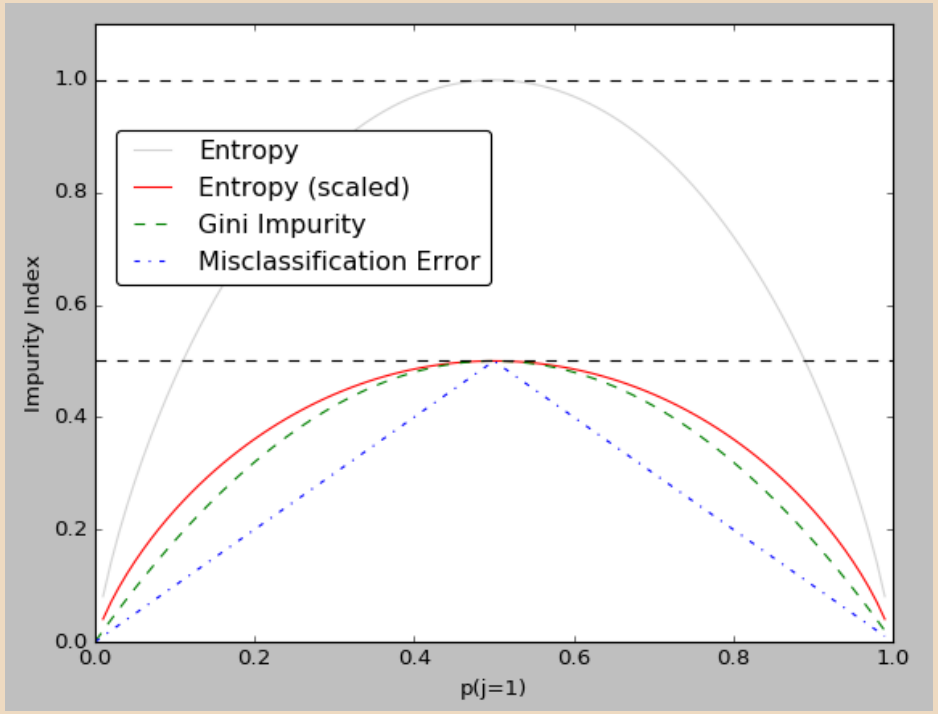
- 마지막 leaf node는 조건부 확률을 이용하여 class를 예측

- 부모 노드와 자식 노드의 불순도를 낮게 만드는 최상의 독립변수와 기준 값을 찾는 것이 목표

- Information Gain

Information Gain

$$IG(D_p) = I(D_p) - \frac{N_{left}}{N_p} I(D_{left}) - \frac{N_{right}}{N_p} I(D_{right})$$



- 너무 깊은 트리에 의한 overfitting의 발생 가능성을 방지하기 위한 목적
- IG를 사용해서 feature vector들 중 제일 쓸모 있는 것이 무엇인지 측정
- 이를 이용해 결정 트리의 노드에 있는 특성들의 순서를 결정한다.

Ensemble

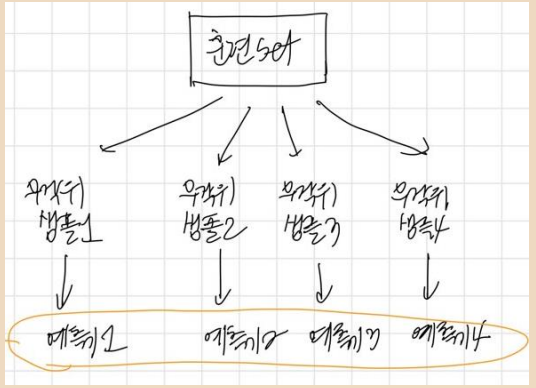
1. Voting

다양한 훈련 알고리즘을 이용
각 분류기의 예측을 모아서 가장 많이 선택된 클래스를 예측
Hard Voting vs Soft Voting

2. Bagging (Bootstrap Aggregating)

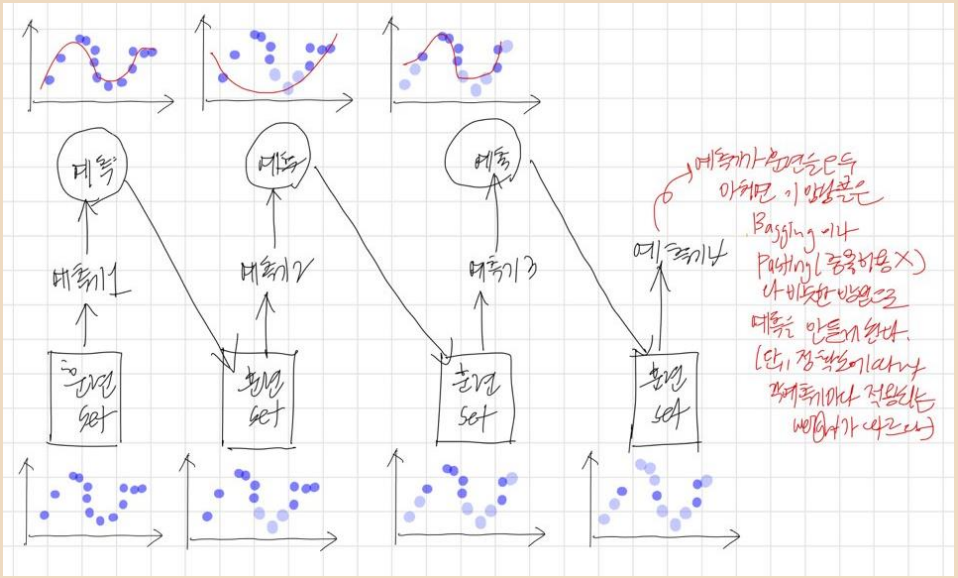
같은 훈련 알고리즘을 사용
훈련 set의 subset을 랜덤하게 구성하여 분류기 별로 각기 다르게 학습

결정 트리의 High Variance + Low Bias라는 특성을 적절히 활용

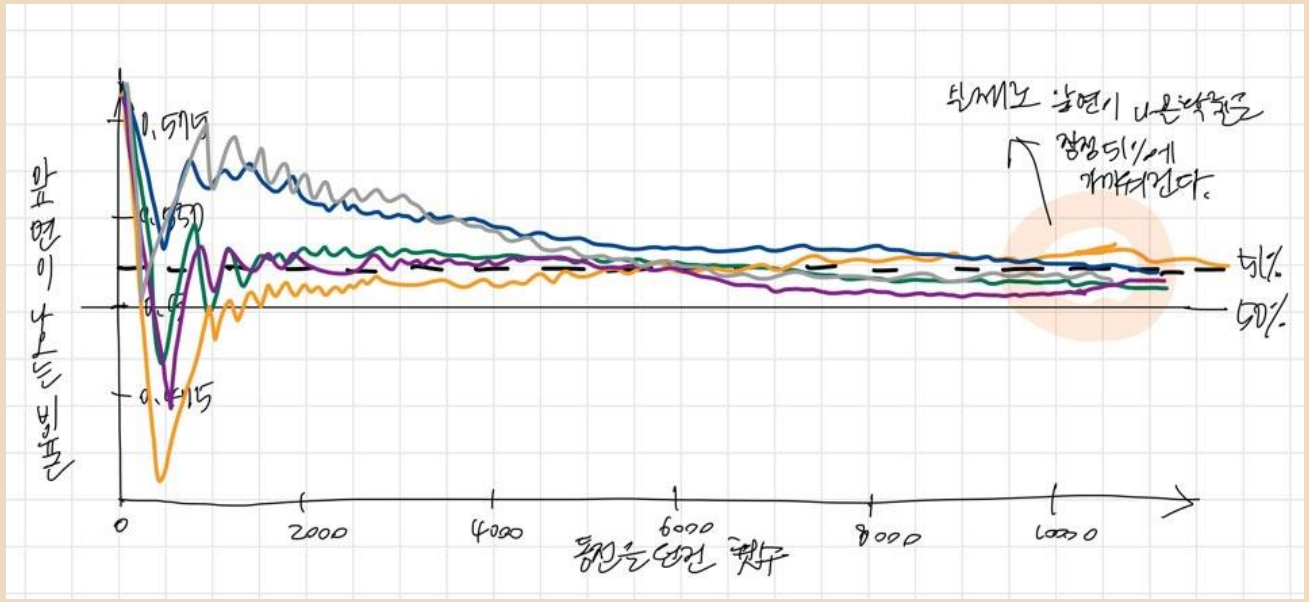


3. Boosting

여러 개의 학습기를 순차적으로 학습 – 예측
잘못 예측한 데이터에 가중치(weight) 부여
Weak learner의 결합으로 최종 예측



Voting



큰 수의 법칙

Voting Classifier

분류만 가능

Hard Voting

직접 투표

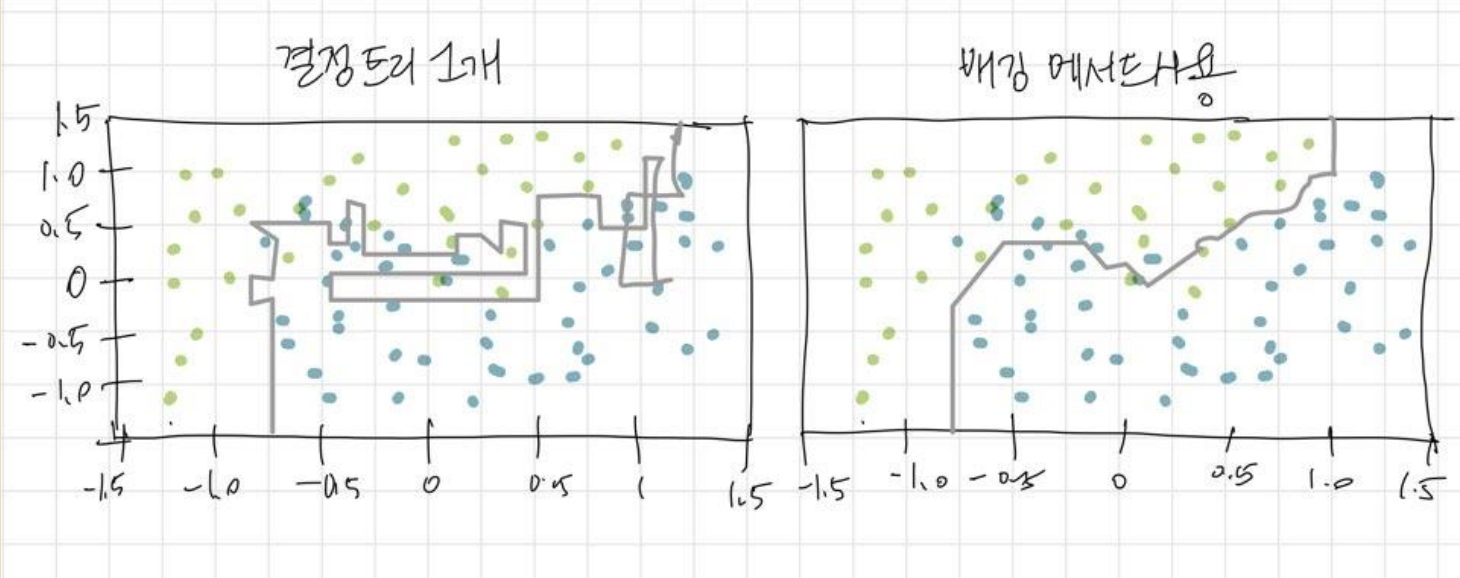
각 분류기가 전체 데이터를 바탕으로 예측한 클래스에 다수결의 원칙 적용

Soft Voting

간접 투표

모든 분류기가 각 클래스별 확률 값을 예측해서 평균을 내 최대 확률의 클래스로 결정

Bagging



Random Forest Regressor &

Random Forest Classifier

Decision Tree vs Random Forest

같은 훈련 알고리즘을 사용

훈련 set의 subset을 랜덤하게 구성하여 분류기 별로 각기 다르게 학습

1. 일반화가 잘 됨
2. 비슷한 편향 = 오차의 수가 거의 비슷
3. 작은 분산 = 덜 불규칙적인 결정 경계
랜덤으로 샘플을 선택해서 각 분류기마다 다르게 입력하는 randomization에 의해 분산 감소 + 편향 조금 증가

Random Forest Algorithm

결정 트리로만 구성된 앙상블

무작위성 부여

편향 손해 but 전체적 예측 정확도 증가

Boosting

Gradient Boosting

AdaBoost와 달리 반복마다 이전 예측기가 만든 잔여 오차에 새로운 예측기를 학습

$Loss = \frac{1}{2}(y_i - f(x_i))^2$ -> $Residual = (y_i - f(x_i)$

Residual ≔ negative gradient

Residual 값을 줄이는 방향으로 weak learner 결합

기존 예측값 + (잔차 * **learning_rate)**로 새롭게 예측값 update

1. Initialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$.
2. For $m = 1$ to M:
   (a) For $i = 1, 2, \ldots, N$ compute
   $$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}.$$
   (b) Fit a regression tree to the targets r_{im} giving terminal regions R_{jm}, $j = 1, 2, \ldots, J_m$.
   (c) For $j = 1, 2, \ldots, J_m$ compute
   $$\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L\left(y_i, f_{m-1}(x_i) + \gamma\right).$$
   (d) Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$.
3. Output $\hat{f}(x) = f_M(x)$.

AdaBoost

이전 예측기를 보완해 새 예측기를 학습시키기 위해 이전 예측기가 under fit했던 훈련 샘플의 가중치를 올림

약한 분류기로 depth가 2인, leaf node가 2개인 한번의 학습만 가능한 것을 순차적으로 사용

Gini 계수가 가장 낮은 feature로 시작

1. Initialize the observation weights $w_i = 1/N$, $i = 1, 2, \ldots, N$.
2. For $m = 1$ to M:
   (a) Fit a classifier $G_m(x)$ to the training data using weights w_i.
   (b) Compute
   $$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}.$$
   (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$.
   (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))]$, $i = 1, 2, \ldots, N$.
3. Output $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

Dataset (Fashion MNIST)

```
In [2]:
(X_train, Y_train), (X_test, Y_test) = tf.keras.datasets.fashion_mnist.load_data()
```

```
In [ ]:
X_train.shape
Out[ ]:
(60000, 28, 28)

In [7]:
img_train = X_train.reshape(X_train.shape[0], X_train.shape[1]*X_train.shape[2])
img_test = X_test.reshape(X_test.shape[0], X_test.shape[1]*X_test.shape[2])
```

```
In [8]:
from sklearn.preprocessing import LabelBinarizer
LabelBin = LabelBinarizer()
Ytrain_Hot = LabelBin.fit_transform(Y_train)
Ytest_Hot = LabelBin.fit_transform(Y_test)
```

Decision Tree Classifier

criterion

- 현재 기준 노드의 feature의 적정성을 계산해주는 방법 설정
- Gini계수를 사용하는 것이 기본 설정이나, entropy loss로 측정하고 싶다면 'entropy'로 변경

class_weight

- {class label : weight})의 사전형형태로 저장된 각 class에 부여하는 가중치
- 'balanced'로 설정하면 **n_samples / (n_classes * np.bincount(y))** 의 형태로 계산

```
class sklearn.tree.DecisionTreeClassifier(
    *, criterion='gini', splitter='best', max_depth=None, min_samples_split=2,
    min_samples_leaf=1) min_weight_fraction_leaf=0.0,
    max_features=None, random_state=None, max_leaf_nodes=None,
    min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None, ccp_alp
ha=0.0
)
```

min_samples_split & min_samples_leaf & max_leaf_nodes &min_impurity_split

- 트리의 노드가 나뉠 때 너무 트리의 깊이가 깊어지지 않도록 하기 위해서 트리의 성장에 규제를 가해주는 대표적인 parameter

max_features

- 제일 적당한 region spit을 위해서 고려해 주는 feature의 개수의 최대값을 정하는 방법 parameter
- int, float, auto, sqrt, log2, None

Decision Tree Classifier

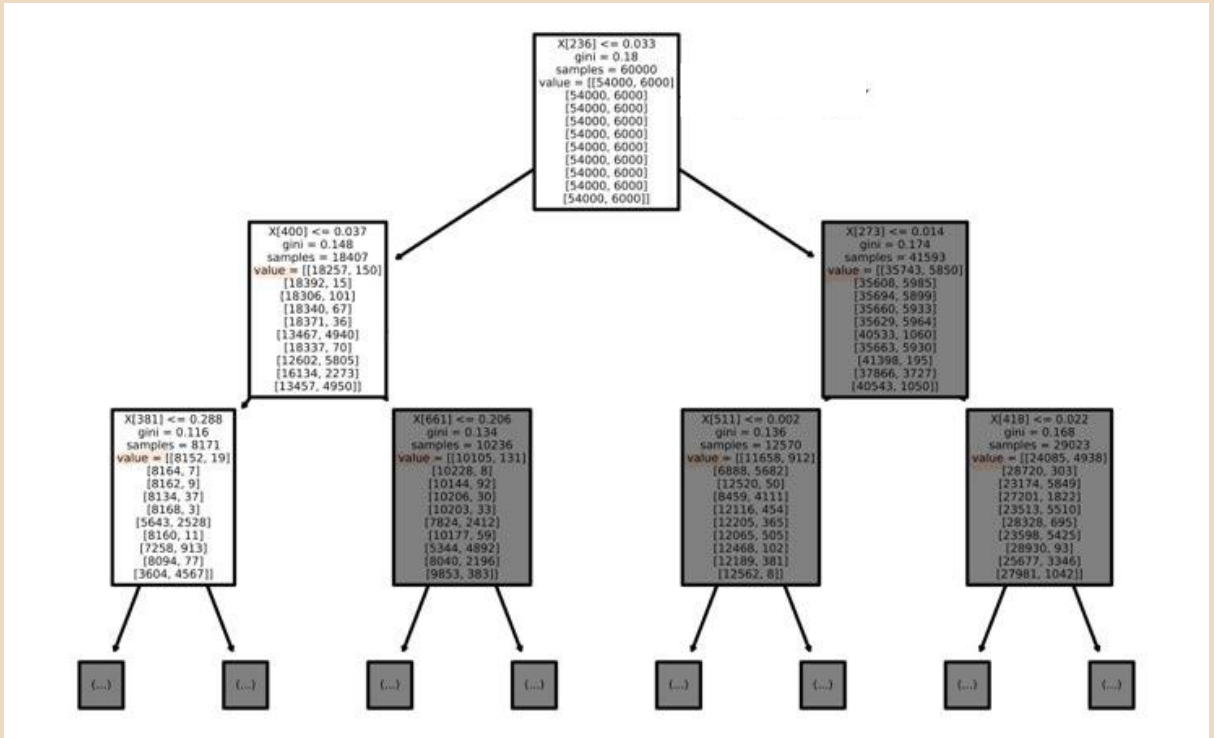
```
from sklearn.tree import DecisionTreeClassifier

TreeCLF = DecisionTreeClassifier(max_depth = 30, max_features = "log2", max_leaf_nodes = 1000, random_state = 42)
```

```
TreeCLF.fit(img_train/255, Ytrain_Hot)
TreeAccuracy = TreeCLF.score(img_test/255, Ytest_Hot)
```

```
print('TreeAccuracy : {}'.format(TreeAccuracy * 100))
```

```
TreeAccuracy : 74.91
```



Voting Classifier

estimators

- 이 parameter은 필수로 입력해 줘야 하는 값으로,
사용하고자 하는 다양한 회귀/분류 모델을 입력해 넣어야 한다.

voting

- hard, soft

```
class sklearn.ensemble.VotingClassifier(
    estimators, *, voting='hard', weights=None,
    n_jobs=None, flatten_transform=True, verbose=False
)
```

Voting Classifier

```
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier

clf1 = LogisticRegression(random_state = 42, solver = 'sag')
clf2 = GaussianNB()
clf3 = RandomForestClassifier(n_estimators = 50, random_state = 42)

VoteCLF = VotingClassifier(
    estimators = [
        ('LogReg', clf1), ('Gauss', clf2), ('Forest', clf3)
    ], voting = 'hard'
)

VoteCLF.fit(img_train/255, Y_train)
VoteAccuracy = VoteCLF.score(img_test/255, Y_test)
print('Voting Clf Accuracy : {}'.format(VoteAccuracy*100))
```

Estimator 결정 시 주의

- Hard Voting은 각 분류기가 예측하는 class의 값을 바탕으로,
- Soft Voting은 예측하는 class별 확률 값을 바탕으로 최종을 하기 때문에

estimator로 입력하는 분류기들 또한 이에 맞춘 출력 값을 반환할 수 있도록 해야 한다.

Random Forest Classifier

ccp_alpha

최적의 나무 크기를 결정하기 위한 변수

$$C_\alpha(T) = |T|\sum m = \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha|T|$$

ccp_alpha보다 작지만 최대의 cost complexity를 가진 서브 트리가 선택된다.

```
class sklearn.ensemble.RandomForestClassifier(
    n_estimators=100, *, criterion='gini', max_depth=None, min_samples_split=2, mi
n_samples_leaf=1,
    min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_im
purity_decrease=0.0,
    min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=None, random_
state=None,
    verbose=0, warm_start=False, class_weight=None, ccp_alpha=0.0, max_samples=Non
e
)
```

bootstrap

- True / False의 값으로 선택
- False선택 시에 새로운 트리를 만드는데 있어서 전체 데이터가 사용

oob_score

- Out Of Bag score
- Bootstrapping을 적용한 이후 선택되지 못한 샘플을 이용해 검증한 점수

Random Forest Classifier

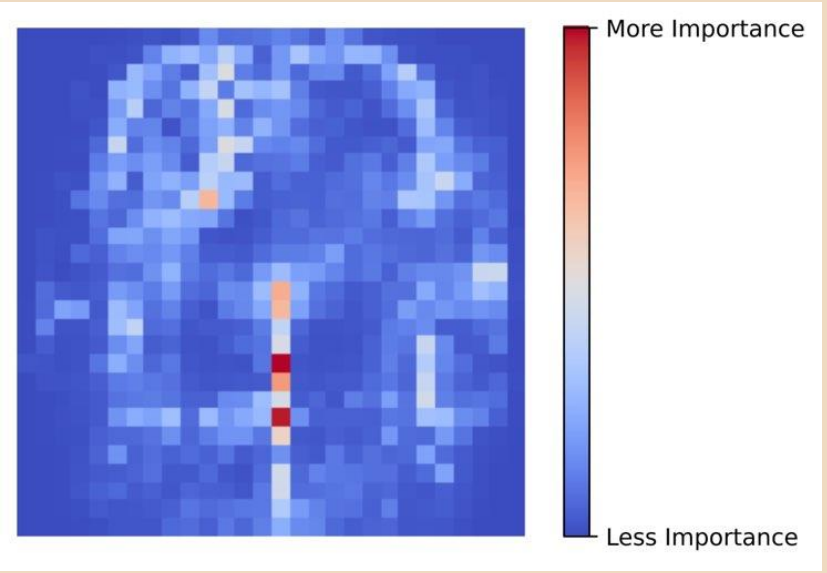
```
from sklearn.ensemble import RandomForestClassifier
RandForCLF = RandomForestClassifier(class_weight = "balanced_subsample", bootstrap = Tru
e, random_state = 42, max_leaf_nodes = 1000)
RandForCLF.fit(img_train, Ytrain_Hot)
RFAccuracy = RandForCLF.score(img_test, Ytest_Hot)
print('Random Forest Accuracy : {}'.format(RFAccuracy*100))
```

```
Random Forest Accuracy : 80.03
```

```
import matplotlib.pyplot as plt
import matplotlib
from matplotlib.colors import Colormap as cm
def plot_digit(data):
    image = data.reshape(28, 28) # 이미지를 하나씩 한수에 입력해서 출력해 본다.
    # ID array로, img_data의 shape가 (data number, 784)이기 때문에 이를 reshape해야 이미지로
인식이 된다.
    plt.imshow(image, cmap = matplotlib.cm.coolwarm, interpolation = "nearest")
    plt.axis("off")

plot_digit(RandForCLF.feature_importances_)

cbar = plt.colorbar(ticks = [RandForCLF.feature_importances_.min(), RandForCLF.feature_im
portances_.max()])
cbar.ax.set_yticklabels(['Less Importance', "More Importance"])
plt.show()
```



feature_importances_

학습시킨 데이터의 shape = (n_samples, 28x28)이기 때문에 각 이미지당 random forest 안의 decision tree가 고려해야 하는 feature의 개수는 684개

각각에 대한 분류기준 설정시의 중요도 확인 가능

Gradient Boosting Classifier

loss

- Deviance : 분류를 각 클래스의 확률 값으로 출력할 때에 적용
- Exponential : AdaBoost에서 사용한 손실 함수를 적용

learning_rate

작을 수록 over fitting을 막기 위해 더 많은 개수의 악한 예측기가 필요
Gradient descent를 적용할 때의 'step size'를 조절
기본 값은 0.1

```
sklearn.ensemble.GradientBoostingClassifier(
    *, loss='deviance', learning_rate=0.1, n_estimators=100, subsample=1.0, criter
ion='friedman_mse',
    min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_dep
th=3,
    min_impurity_decrease=0.0, min_impurity_split=None, init=None, random_state=No
ne,
    max_features=None, verbose=0, max_leaf_nodes=None, warm_start=False, validatio
n_fraction=0.1,
    n_iter_no_change=None, tol=0.0001, ccp_alpha=0.0)
```

tol

Early Stopping에 관여하는 parameter
Loss값이 tol로 설정한 값(기본값 1e-4)보다 내려간다면 학습을 멈춤

validation_fraction & n_iter_no_change

전체 훈련 데이터에서 검증 데이터로 선택할 비율 지정
이 값을 설정해 주어야만 n_iter_no_change를 사용 가능
Validation Score이 증가하지 않으면 **Early Stopping**이 가능

Gradient Boosting Classifier

```
from sklearn.ensemble import GradientBoostingClassifier

GradBoostCLF = GradientBoostingClassifier(max_depth = 2, n_estimators = 20, learning_rat
e = 0.1, random_state=42,
                                          validation_fraction = 0.2, n_iter_no_change
= 10, verbose = 1)
GradBoostCLF.fit(img_train, Y_train)
GradBoostAcc = GradBoostCLF.score(img_test, Y_test)

print('Gradient Boosting Accuracy : {}'.format(GradBoostAcc * 100))
```

```
      Iter       Train Loss   Remaining Time
         1       92794.0112           10.99m
         2       82622.5193           10.41m
         3       75077.1204            9.83m
         4       69087.9825            9.25m
         5       64016.1541            8.68m
         6       59893.1888            8.10m
         7       56268.0564            7.52m
         9       50366.1584            6.37m
        10       48049.3693            5.79m
        20       33733.6854            0.00s
Gradient Boosting Accuracy : 78.94
```

Early Stopping 적용

```
from sklearn.metrics import mean_squared_error

errors = [mean_squared_error(Y_test, y_pred)
          for y_pred in GradBoostCLF.staged_predict(img_test)]
best_n_estimators = np.argmin(errors) + 1

GradBoost_best = GradientBoostingClassifier(max_depth = 2, n_estimators = best_n_estimat
ors, learning_rate = 0.1, random_state=None,
                                            validation_fraction = 0.2, n_iter_no_change
= 10, verbose = 1)
GradBoost_best.fit(img_train, Y_train)
BestGradAcc = GradBoost_best.score(img_test, Y_test)

print('Best Gradient Boosting Accuracy : {}'.format(BestGradAcc * 100))
```

```
      Iter       Train Loss   Remaining Time
         1       92821.7450           11.01m
         2       82627.2347           10.43m
         3       74983.2848            9.84m
         4       69032.1478            9.26m
         5       64088.5143            8.68m
         6       59893.7227            8.10m
         7       56270.2774            7.52m
         8       53151.5085            6.94m
         9       50507.8447            6.36m
        10       48130.8722            5.78m
        20       33795.0259            0.00s
Best Gradient Boosting Accuracy : 79.10000000000001
```

감사합니다