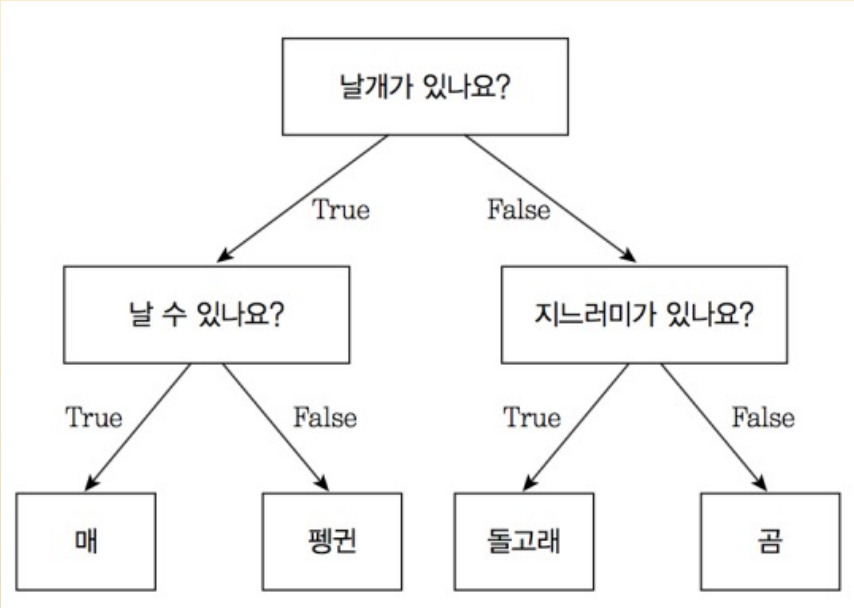
결정트리 앙상블
랜덤 포레스트

Kuggle 겨울방학 정규세션 2주차

백준성 전웅 신다혜

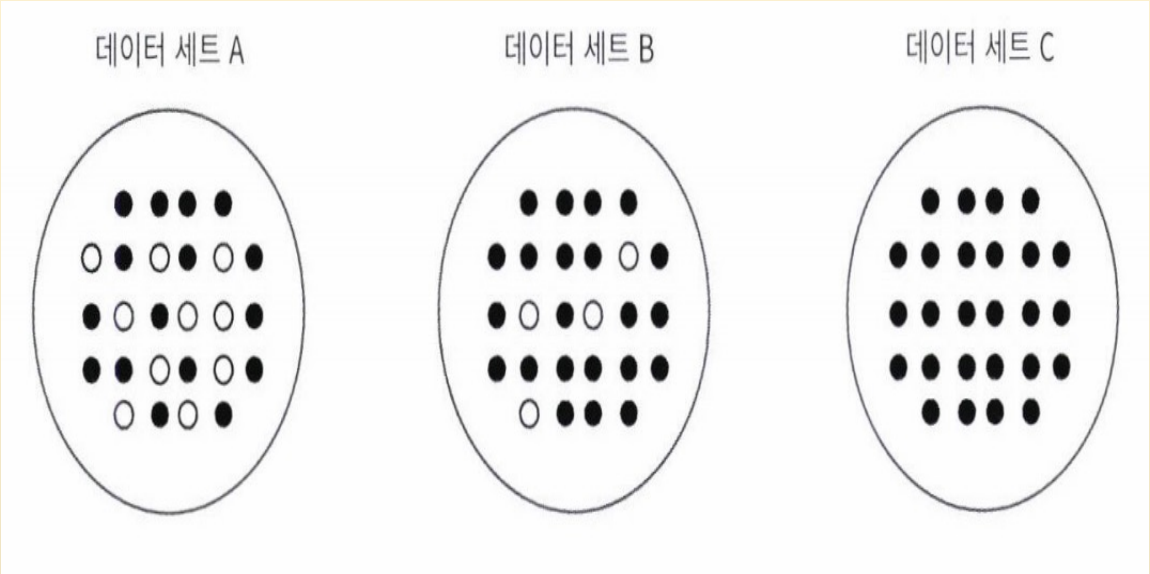
결정트리

- 결정트리(Decision Tree)
- 규칙을 통해 분류해 나가는 알고리즘



결정트리

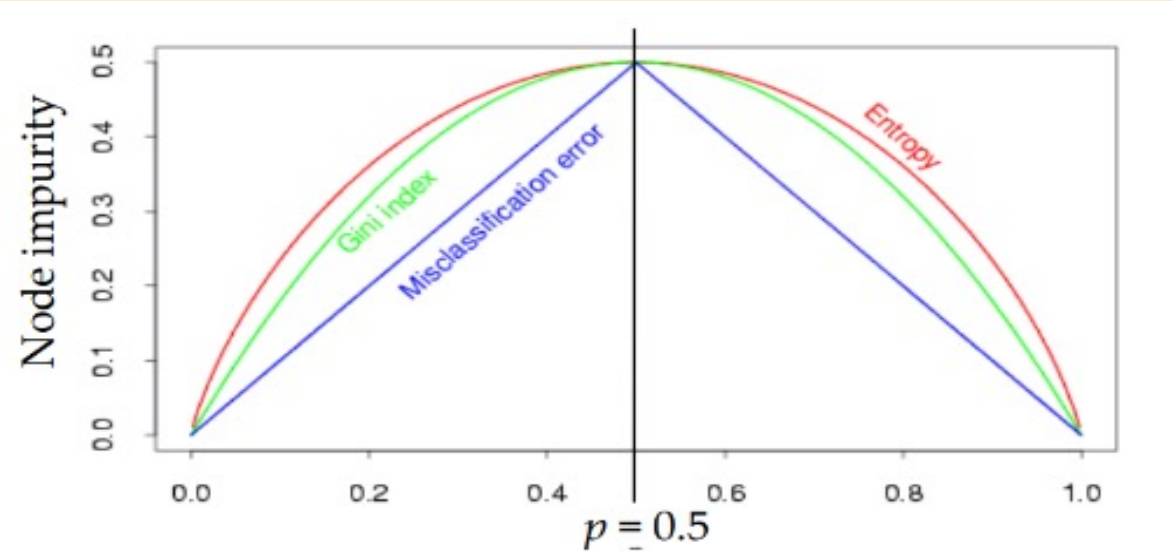
- 균일도



- 균일도 : C > B > A
- 균일도가 낮아지는 방향으로 학습 -> 학습으로 규칙 찾아냄
- 균일도 측정 방법 : 엔트로피, 지니계수

결정트리

- 지니계수 vs 엔트로피



- 엔트로피가 일반적으로 성능 더 좋음
- 하지만 계산량이 많다는 단점

Data Set

```
train_data_m = pd.read_csv("C:/Temp/PlayTennis.csv")
```

```
train_data_m
```

	Outlook	Temperature	Humidity	Wind	Play Tennis
0	Sunny	Hot	High	Weak	No
1	Sunny	Hot	High	Strong	No
2	Overcast	Hot	High	Weak	Yes
3	Rain	Mild	High	Weak	Yes
4	Rain	Cool	Normal	Weak	Yes
5	Rain	Cool	Normal	Strong	No
6	Overcast	Cool	Normal	Strong	Yes
7	Sunny	Mild	High	Weak	No
8	Sunny	Cool	Normal	Weak	Yes
9	Rain	Mild	Normal	Weak	Yes
10	Sunny	Mild	Normal	Strong	Yes
11	Overcast	Mild	High	Strong	Yes
12	Overcast	Hot	Normal	Weak	Yes
13	Rain	Mild	High	Strong	No

Entropy

```
features = train_data_m[["Outlook","Temperature","Humidity","Wind"]]
# 대상 속성(target feature)
target = train_data_m["Play Tennis"]
```

```
# 엔트로피
def entropy(target_col):
    elements, counts = np.unique(target_col, return_counts = True)
    entropy = -np.sum([(counts[i]/np.sum(counts))*np.log2(counts[i]/np.sum(counts)) for i in range(len(elements))])
    return entropy
```

```
print('H(x) = ', round(entropy(target), 5))
```

```
H(x) =  0.94029
```

$$H(X) = -\sum_{k=1}^{K} p(X = k) log p(X = k)$$

InfoGain

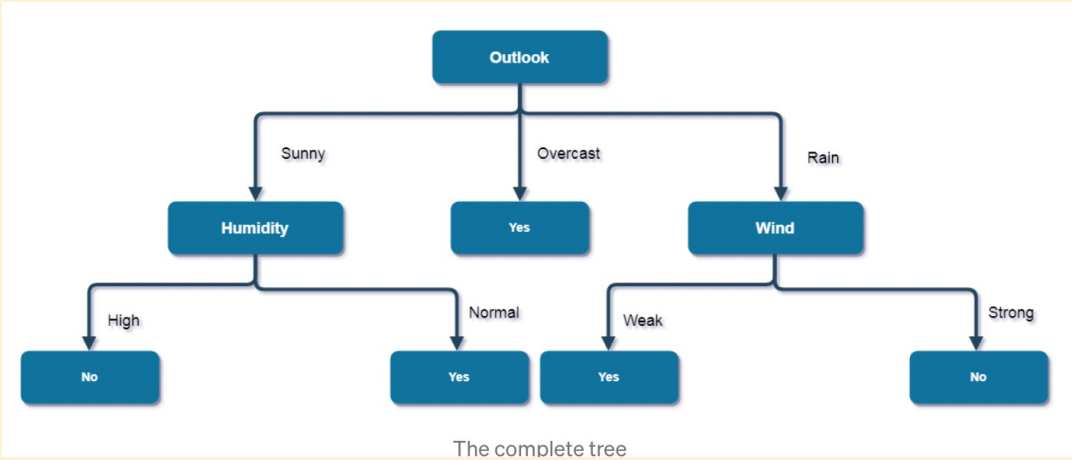
```python
# 정보이득
def InfoGain(data,split_attribute_name,target_name):

    # 전체 엔트로피 계산
    total_entropy = entropy(data[target_name])
    print('Entropy(D) = ', round(total_entropy, 5))

    # 가중 엔트로피 계산
    vals,counts= np.unique(data[split_attribute_name],return_counts=True)
    Weighted_Entropy = np.sum([(counts[i]/np.sum(counts))*
                               entropy(data.where(data[split_attribute_name]==vals[i]).dropna()[target_name])
                               for i in range(len(vals))])
    print('H(', split_attribute_name, ') = ', round(Weighted_Entropy, 5))


    # 정보이득 계산
    Information_Gain = total_entropy - Weighted_Entropy
    return Information_Gain
```

$$InfoGain(feature_d) = Entropy(D) - Entropy(feature_d)$$

$$InfoGain(feature_d, D) = Entropy(D) - \sum_{t \in feature} \left(\frac{|feature_d = t|}{|D|} \times H(feature_d = t)\right)$$

$$= Entropy(D) - \sum_{t \in feature} \left(\frac{|feature_d = t|}{|D|} \times \left(- \sum_{k \in target} \left(P(target = k, feature_d = t) \times \log_2 P(target = k, feature_d = t)\right)\right)\right)$$

InfoGain



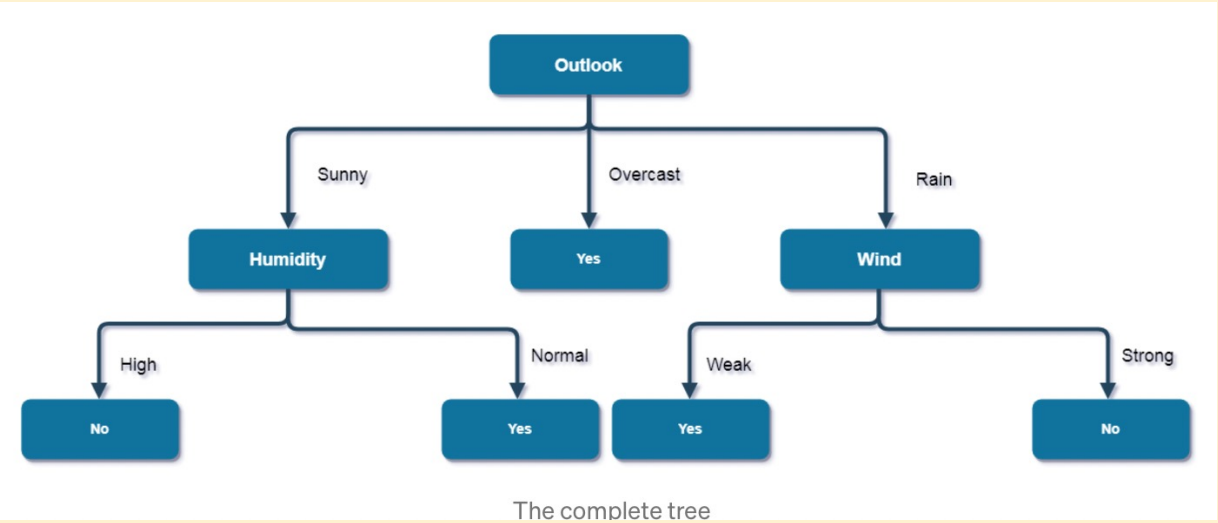
The complete tree

```
Entropy(D) =  0.94029
H( Outlook ) =  0.69354
InfoGain( Outlook ) =  0.24675
Entropy(D) =  0.94029
H( Temperature ) =  0.91106
InfoGain( Temperature ) =  0.02922
Entropy(D) =  0.94029
H( Humidity ) =  0.78845
InfoGain( Humidity ) =  0.15184
Entropy(D) =  0.94029
H( Wind ) =  0.89216
InfoGain( Wind ) =  0.04813
```

Tree 구현 코드

```
def ID3(data,originaldata,features,target_attribute_name,parent_node_class = None):

    # 중지기준 정의

    # 1. 대상 속성이 단일값을 가지면: 해당 대상 속성 반환
    if len(np.unique(data[target_attribute_name])) <= 1:
        return np.unique(data[target_attribute_name])[0]

    # 트리 성장
    else:
        # 부모노드의 대상 속성 정의(예: Good)
        parent_node_class = np.unique(data[target_attribute_name])\
                            [np.argmax(np.unique(data[target_attribute_name], return_counts=True)[1])]

        # 데이터를 분할할 속성 선택
        item_values = [InfoGain(data,feature,target_attribute_name) for feature in features]
        best_feature_index = np.argmax(item_values)
        best_feature = features[best_feature_index]

        # 트리 구조 생성
        tree = {best_feature:{}}

        # 최대 정보이득을 보인 기술 속성 제외
        features = [i for i in features if i != best_feature]

        # 가지 성장
        for value in np.unique(data[best_feature]):
            # 데이터 분할. dropna(): 결측값을 가진 행, 열 제거
            sub_data = data.where(data[best_feature] == value).dropna()

            # ID3 알고리즘
            subtree = ID3(sub_data,data,features,target_attribute_name,parent_node_class)
            tree[best_feature][value] = subtree

        return(tree)
```

Tree 결과값



The complete tree

```
{'Outlook': {'Overcast': 'Yes',
             'Rain': {'Wind': {'Strong': 'No', 'Weak': 'Yes'}},
             'Sunny': {'Humidity': {'High': 'No', 'Normal': 'Yes'}}}}
```

Data Set

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np

# 붓꽃 데이터를 로딩하고, 학습과 페트스 데이터 세트로 분리
iris_data = load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris_data.data, iris_data.target,
                                                    test_size = 0.2, random_state = 11)

# 교재와 똑같이 함
train_df = pd.DataFrame(X_train, columns = iris_data.feature_names)
train_df['target'] = y_train

# feature들 리스트로 모아둠
features = list(train_df.columns)
features.remove('target')
```

Gini

```
# 지니계수 함수
def gini(x):
    return 1-( (x / x.sum() )**2).sum()
```

$$Gini\,(P) = \sum_{i=1}^{n} p_i(1-p_i) = 1 - \sum_{i=1}^{n} (p_i)^2$$

cutting point

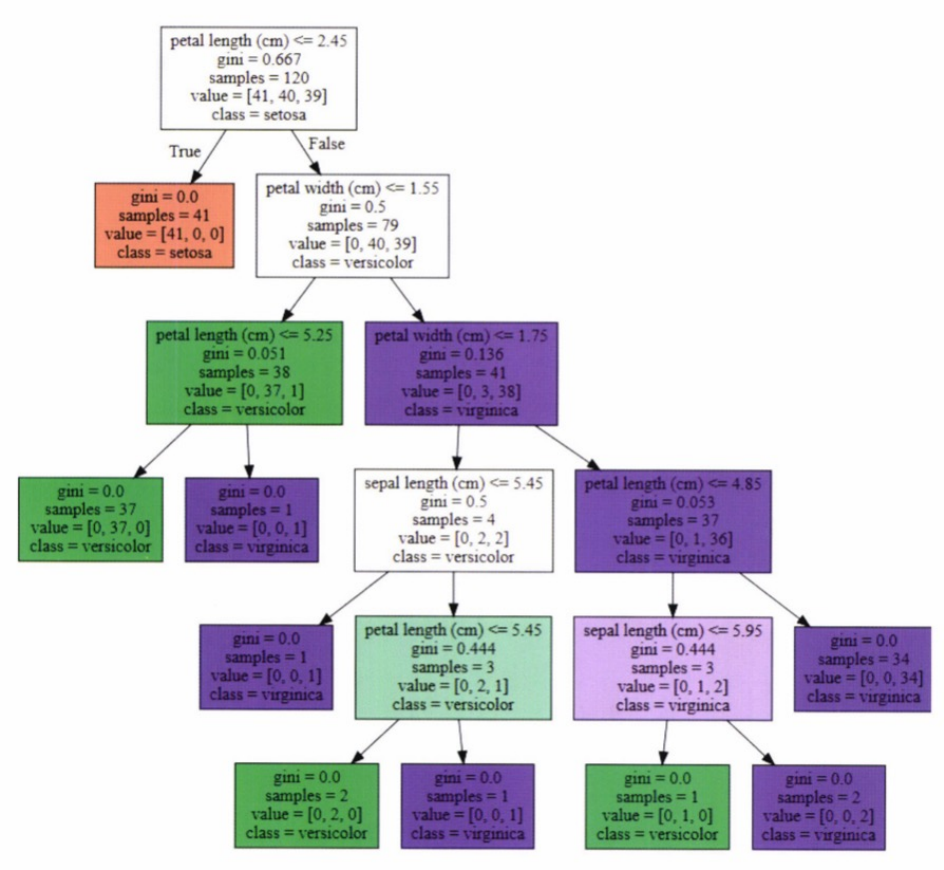
```python
# 지니계수가 최소가 되는 조건을 찾는 알고리즘
def dt_algo(df):
    G_list = []
    G1_list = []
    G2_list = []
    cp_list = []
    num_cp = []
    for feature in features:
        df.sort_values(by = feature, inplace = True)
        df.reset_index(drop = True, inplace = True)
        cut_index = []
        for i in range(len(df)-1):
            if df['target'][i] != df['target'][i+1]:
                cut_index.append(i)
            else:
                pass

        # 지니계수 최소가 되는 cutting point 찾기
        for index in cut_index:
            cp = (df[feature][index] + df[feature][index+1]) / 2 # cutting point
            df_cp1 = df[df[feature] <= cp]
            df_cp2 = df[df[feature] > cp]
            N, N1, N2 = len(df), len(df_cp1), len(df_cp2)
            G1, G2 = gini(df_cp1['target'].value_counts()), gini(df_cp2['target'].value_counts())
            G = G1*(N1/N) + G2*(N2/N)
            G_list.append(G)
            G1_list.append(G1)
            G2_list.append(G2)
            cp_list.append(cp)

        num_cp.append(len(cut_index))

    if np.argmin(G_list) < num_cp[0]:
        return features[0], cp_list[np.argmin(G_list)], min(G1_list)
    elif (num_cp[0] <= np.argmin(G_list) < sum(num_cp[:2])):
        return features[1], cp_list[np.argmin(G_list)], min(G1_list)
    elif (sum(num_cp[:2]) <= np.argmin(G_list) < sum(num_cp[:3])):
        return features[2], cp_list[np.argmin(G_list)], min(G1_list)
    elif (sum(num_cp[:3]) <= np.argmin(G_list) < sum(num_cp[:4])):
        return features[3], cp_list[np.argmin(G_list)], min(G1_list)
```

트리 만들기

```
# dt_algo에 데이터프레임넣으면 gini가 최소가 되는 조건이 나옴. 조건을 따라서 쭉 대입함
print(dt_algo(train_df))
dt1 = train_df.copy()[train_df['petal length (cm)'] <= 2.45]
dt2 = train_df.copy()[train_df['petal length (cm)'] > 2.45]
dt1['target'].value_counts(), dt2['target'].value_counts()

print(dt_algo(dt2))
dt21 = dt2.copy()[dt2['petal width (cm)'] <= 1.55]
dt22 = dt2.copy()[dt2['petal width (cm)'] > 1.55]
dt21['target'].value_counts(), dt22['target'].value_counts()

print(dt_algo(dt21))
dt31 = dt21.copy()[dt21['petal length (cm)'] <= 5.25]
dt32 = dt21.copy()[dt21['petal length (cm)'] > 5.25]
dt31['target'].value_counts(), dt32['target'].value_counts()

print(dt_algo(dt22))
dt33 = dt22.copy()[dt22['petal width (cm)'] <= 1.7]
dt34 = dt22.copy()[dt22['petal width (cm)'] > 1.7]
dt33['target'].value_counts(), dt34['target'].value_counts()

print(dt_algo(dt33))
dt41 = dt33.copy()[dt33['sepal length (cm)'] <= 5.45]
dt42 = dt33.copy()[dt33['sepal length (cm)'] > 5.45]
dt41['target'].value_counts(), dt42['target'].value_counts()

print(dt_algo(dt34))
dt43 = dt34.copy()[dt34['petal length (cm)'] <= 4.8]
dt44 = dt34.copy()[dt34['petal length (cm)'] > 4.8]
dt43['target'].value_counts(), dt44['target'].value_counts()

print(dt_algo(dt42))
dt51 = dt42.copy()[dt42['sepal length (cm)'] <= 6.95]
dt52 = dt42.copy()[dt42['sepal length (cm)'] > 6.95]
dt51['target'].value_counts(), dt52['target'].value_counts()

print(dt_algo(dt43))
dt53 = dt43.copy()[dt43['sepal length (cm)'] <= 5.95]
dt54 = dt43.copy()[dt43['sepal length (cm)'] > 5.95]
dt53['target'].value_counts(), dt54['target'].value_counts()
```

Conclusion

```
('petal length (cm)', 2.45, 0.0)
('petal width (cm)', 1.55, 0.0)
('petal length (cm)', 5.25, 0.0)
('petal width (cm)', 1.7, 0.0)
('sepal length (cm)', 5.45, 0.0)
('petal length (cm)', 4.8, 0.0)
('sepal length (cm)', 6.95, 0.0)
('sepal length (cm)', 5.95, 0.0)

(1    1
 Name: target, dtype: int64,
 2    2
 Name: target, dtype: int64)
```



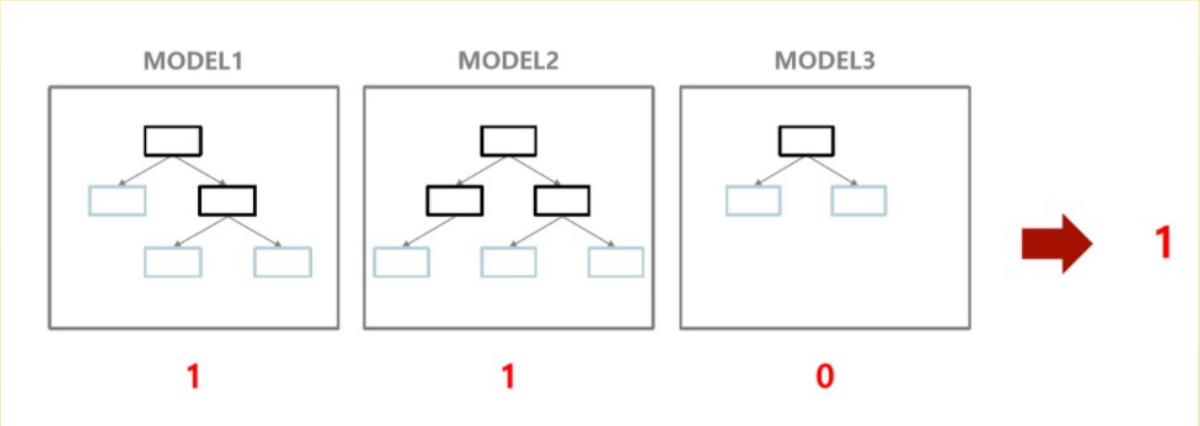
과적합 방지

- Tree의 모든 terminal node를 순도 100%로 만들면 분기가 너무 많아 과적합(overfitting)이 발생 -> 과하게 학습되어 실제 데이터에 대한 오차가 증가한다는 의미
- 과적합 방지 방법은 매우 많은데 그 중 decision tree에 특화된 방법이 **가지치기**와 **앙상블**
- 가지치기
  사전 가지치기 : 트리 최대 depth나 각 노드의 최소 관측값을 지정하여 트리 만드는 도중 stop!
  사후 가지치기 : full tree (순도 100 트리) 만들고 적절 수준에서 terminal node 결합

앙상블

- 앙상블 : 과적합 방지 방법 중 하나, 여러 모델 조합하여 과적합 방지, 예측의 정확도를 높이는 방법
- - **앙상블의 핵심 : 다수결 !**
- 앙상블의 예 : Random Forest – decision tree라는 같은 알고리즘으로 만들어진 모델 여러 개로 조합 (직관적 이해 : 형태가 조금씩 다른 decision tree 모델들이 모여서 숲을 이룬다~)
- BUT, 앙상블이 꼭 같은 알고리즘으로 만들어진 모델들로 조합되는 것은 아님. Random forest가 같은 알고리즘 (decision tree) 으로 조합된 것 뿐, 앙상블이 그렇다는 것은 아니다!
- 앙상블은 decision tree, DNN, Logistic 등 여러 다른 알고리즘으로 모델 만들고 이들 조합도 앙상블이라 한다.

앙상블

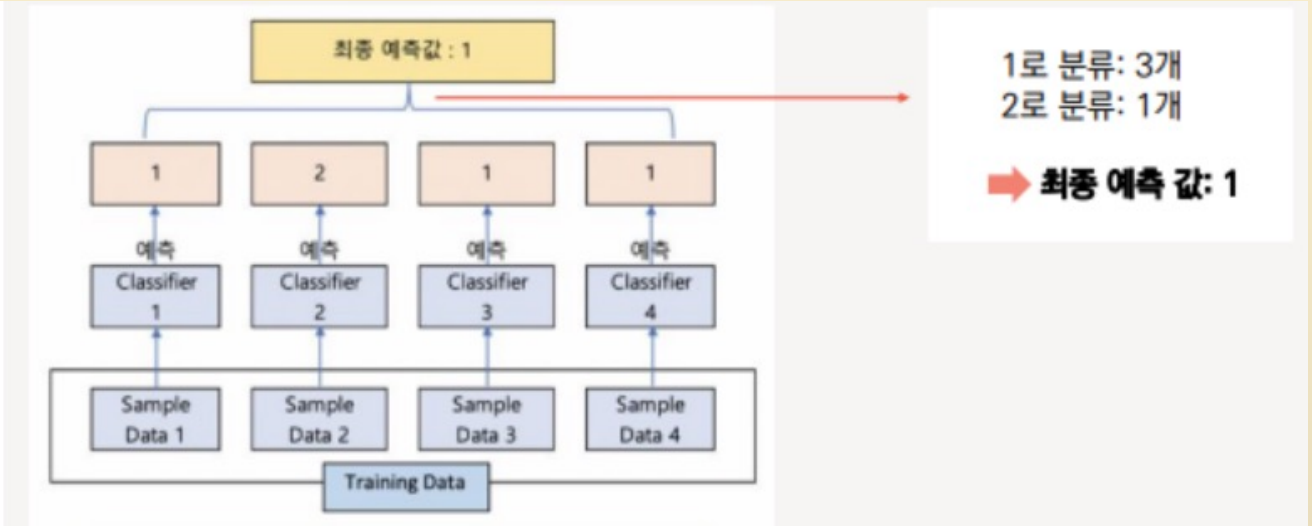


- 0과 1을 분류하는 문제에서 2개의 model은 1이라 대답했고, 단 1개의 model만이 0이라 대답했다면
- 다수결 원칙에 따라 1이라 최종 결정하는 것이 앙상블의 핵심!

앙상블의 키워드

- **1) 보팅**
- 여러 개의 분류기가 투표를 통해 최종 예측 결과를 결정하는 방식 / 같은 data set에 대해 여러 알고리즘으로 학습

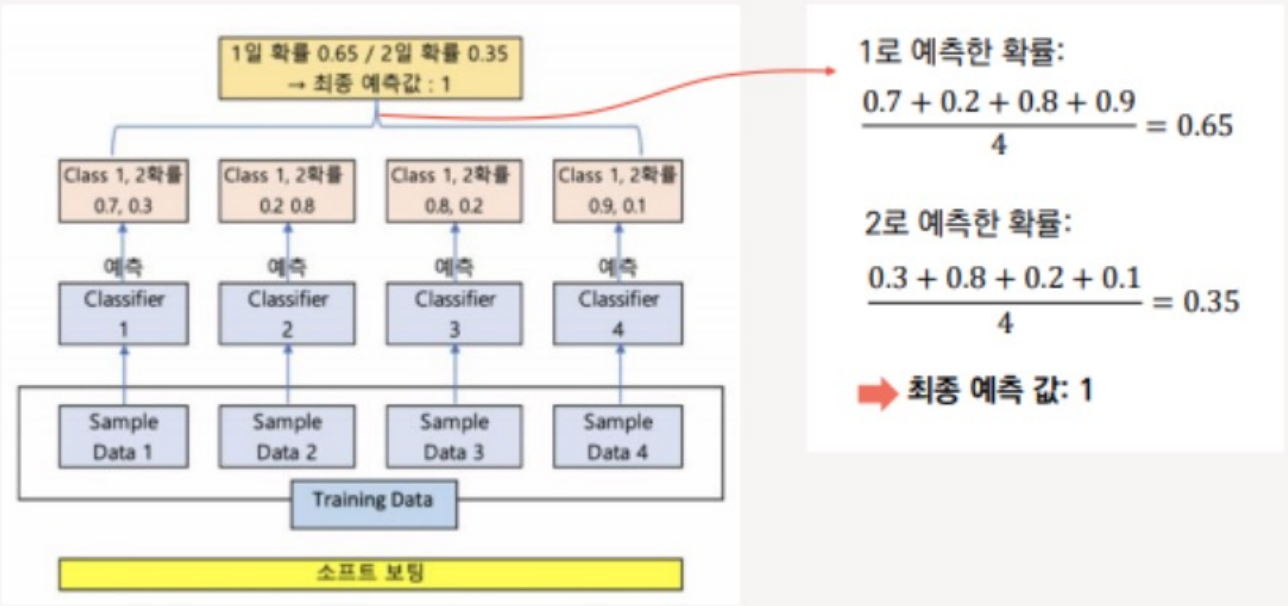
- 하드보팅 : 예측 결과값들 중 다수의 분류기가 결정한 예측값을 최종 결과값으로 선택



같은 data set에서 샘플링하여 각기 다른 분류기 적용(예를 들어 decision tree, logistic svm 등) 하여 학습 시켜 결정한 예측값으로 다수결 한다고 생각하면 됨!

앙상블의 키워드

- **1) 보팅**
- 소프트보팅 : 분류기들의 레이블 값 결정 확률의 평균값 구한 뒤 **확률**이 가장 높은 레이블 값을 최종 결과값으로 결정



같은 data set에서 샘플링하여 각기 다른 분류기 적용하는 것은 하드보팅과 동일하지만 각 분류기에서 0이 나올 확률, 1이 나올 확률(레이블 값)을 모두 고려해준다는 점이 특징! -> 소프트 보팅에서의 예측 성능이 더 좋아서 많이 사용됨.

하드보팅
소프트보팅

```
model_1 = [0.6, 0.4]
model_2 = [0.49, 0.51]
model_3 = [0.49, 0.51]
model_4 = [0.49, 0.51]

models = [model_1, model_2, model_3, model_4]

#Hard Voting
def hard_voting(models):
    class_0 = 0
    class_1 = 0
    for model in models:
        if model[0] < model[1]:
            class_1 += 1
        else:
            class_0 += 1

    if (class_1 > class_0):
        return print("class 1")
    else:
        return print("class 0")
```

```
#Soft Voting
def soft_voting(models):
    class_0 = 0
    class_1 = 0
    for model in models:
        class_0 += model[0]
        class_1 += model[1]
    if (class_1 > class_0):
        return print("class 1")
    else:
        return print("class 0")

hard_voting(models)
soft_voting(models)
```

executed in 8ms, finished 17:35:47 2022-01-02

```
class 1
class 0
```

- 너무 당연한 얘기지만 소프트 보팅은 각 모델의 결정 확률에 0.5가 아닌 다른 값을 곱하면 6:4, 7:3 등 다양한 비율로 가중 평균을 낼 수 있다. 일반적으로 하드 보팅보다는 소프트 보팅 방식이 성능이 더 좋아 자주 사용되고 있다.

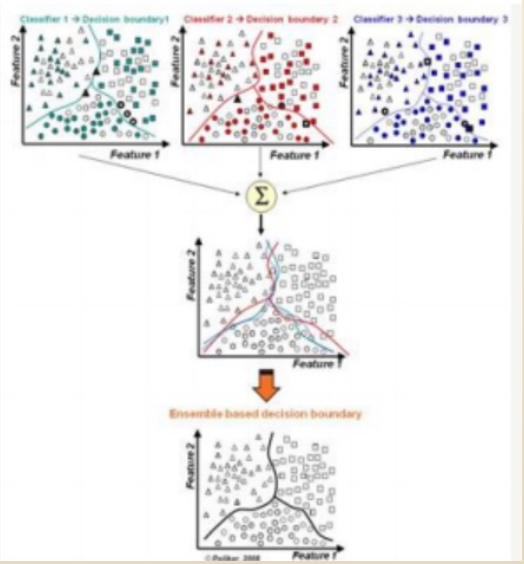
앙상블의 키워드

- **2) 배깅**
- 여러 개 data set을 중첩되게 분리하는 부트스트래핑 기법을 사용하여 예측값을 구하는 방법
- 데이터가 갖는 다양성을 늘려주는 방법이다!
- 원래 데이터 $\boldsymbol{y} = \boldsymbol{F}'(\boldsymbol{x}) + \boldsymbol{\varepsilon}$ 로 반복적으로 학습시키면 오차항이 종속적인 모델로 만들어지기 때문에 같은 모델에서 샘플을 여러 번 뽑아(bootstrap) 각 모델에 학습시켜 결과물을 집계 -> 배깅을 통해 오차항 데이터가 갖는 분포를 왜곡시키기 위해서

✓ 색이 있는 데이터는 1회 이상 샘플링된 데이터

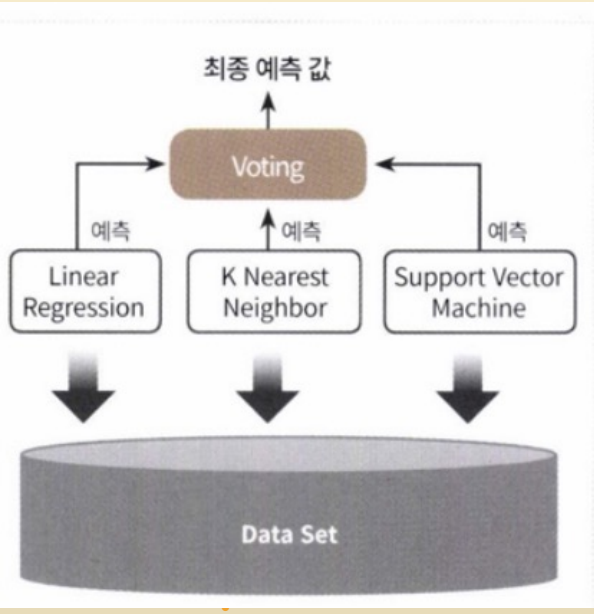
✓ 다른 데이터 분포로 인해 다른 모델이 생성된다.

✓ 각각의 데이터가 만들어낸 모델을 앙상블하여 최종모델을 만들어낸다.

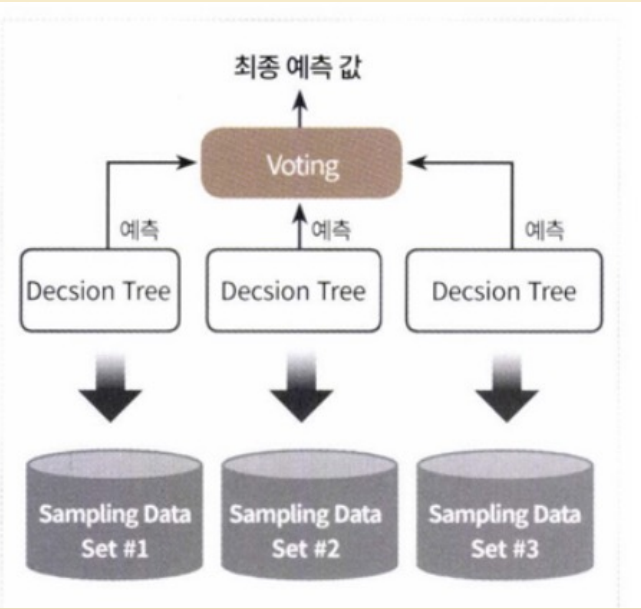


보팅과 배깅

- 보팅 : 일반적으로 서로 다른 알고리즘을 가진 분류기를 결합 ex) svm, decision tree, logistic~
- 배깅 : 모두 같은 유형 알고리즘 기반, 데이터 샘플링을 서로 다르게 (부트스트랩 기법) 학습을 수행 -> 최종적으로는 결국 보팅을 수행
- (즉, 한 알고리즘이 여러 개의 학습데이터를 학습하는 것)



보팅



배깅

랜덤 포레스트

- 앙상블 배깅의 대표적 알고리즘
- 앙상블 모델이 잘 작동하기 위해 Diversity와 Random성 확보 중요 !
- 랜덤 포레스트는 앙상블 알고리즘 중 비교적 빠른 수행 속도와 다양한 영역에서 높은 예측 성능
- 그 이유 두 가지 1) Bagging 2) Random subspace

랜덤 포레스트

1)Bagging

- 앙상블 알고리즘 중 비교적 빠른 수행 속도
- 다양한 영역에서 높은 예측 성능
- **Bagging=Bootstrap Aggregating**
- 부트스트랩 : 데이터셋의 샘플링 방식 / data set 뽑을 때 복원추출 하여 원 데이터 수만큼 데이터 셋을 뽑아야 한다는 점이 부트스트랩의 핵심 기법
- Aggregating : 각 base모델의 결과값을 합쳐서 예측을 하게 되는 방식. Label이 continuous하다면 base모델의 결과값을 평균 내어 예측하고, Label이 class 즉 categorical하다면 보팅을 하게 됨

랜덤 포레스트

2)Random subspace

- 랜덤 포레스트 모델 성능 높이기 위해서는 개별 base모델이 서로 독립적일수록 좋다. 독립적인 모델을 만들 수 있는 방법이 Random subspace
- 랜덤 포레스트의 개별 base모델은 decision tree, 보통 결정트리에서 분기 나눌 때 모든 변수고려 but 랜덤 포레스트에서는 분기점을 원래 변수의 수보다 적은 수의 변수를 random으로 선택해서 임의의 선택된 변수만 가지고 가지치기를 함

변수의 중요도

- 회귀나 로지스틱은 모수적 추정 가능
- Decision tree를 이용하는 random forest는 비모수적모델
  -> **그럼 어떻게 좋은 변수를 선택할거냐?**
- 알려진 확률분포를 가정하지 않기 때문!

: OOB 데이터를 이용한 generalization error를 지표로 삼아 Feature importance를 계산해 볼 수 있음 !