Tree와 랜덤 포레스트

Tree와 랜덤 포레스트

- 결정 트리 분류기 훈련
  - 의사결정트리 분류기는 일련의 질문에 근거하여 주어진 데이터를 분류해주는 알고리즘입니다.
  - 의사결정트리 학습은 트레이닝 데이터를 이용해 데이터를 최적으로 분류해주는 질문들을 학습하는 머신러닝입니다.
  - 의사결정트리 학습에서 각 노드에서 분기하기 위한 최적의 질문은 정보이득(Information Gain)이라는 값이 최대가 되도록 만들어주는 것이 핵심입니다.
  - 어느 특정 노드에서 m개의 자식 노드로 분기되는 경우 정보이득은 다음의 식으로 정의합니다.
  - $$IG(D_p, f) = I(D_p) - \sum_{j=1}^{m} \frac{N_j}{N_p} I(D_j)$$
  - 데이터 불순도란 데이터가 제대로 분류되지 않고 섞여 있는 정도를 의미합니다.
  - 정보이득 IG는 자식노드의 데이터 불순도가 작으면 작을수록 커지게 됩니다.
  - 계산을 단순화하기 위해 보통 의사결정트리에서 분기되는 자식 노드의 개수는 2개로 하는 이진 의사결정트리(Binary decision tree)라고 부릅니다.

$$IG(D_p, f) = I(D_p) - \frac{N_L}{N_p} I(D_L) - \frac{N_R}{N_p} I(D_R)$$

  - 데이터 불순도를 측정하는 방법 - 지니 인덱스(Gini Index), 엔트로피(entropy), 분류오류(classification error)

Tree와 랜덤 포레스트

- 결정 트리 분류기 훈련
  - 결정 트리 학습기는 노드에서 불순도가 가장 크게 감소하는 결정 규칙을 찾습니다.
  - DecisionTreeClassifier는 기본적으로 지니 불순도를 사용합니다.
  - 불순도를 낮추는 결정 규칙을 찾는 과정은 모든 리프 노드(leaf node)가 순수해지거나(즉, 한 클래스만 남거나) 어떤 임곗값에 도달할 때까지 반복됩니다.

```
from sklearn.tree import DecisionTreeClassifier
from sklearn import datasets

iris = datasets.load_iris()  # 데이터 로드
features = iris.data
target = iris.target

decisiontree = DecisionTreeClassifier(random_state=0)  # 결정 트리 분류기 객체 생성
model = decisiontree.fit(features, target)  # 모델 훈련

observation = [[ 5,  4,  3,  2]]  # New 샘플 데이터
model.predict(observation)  # 샘플 데이터의 클래스 예측
model.predict_proba(observation)  # 세 개의 클래스에 대한 예측 확률을 확인

# 엔트로피를 사용해 결정 트리 분류기를 훈련합니다.
decisiontree_entropy = DecisionTreeClassifier( criterion='entropy', random_state=0)
model_entropy = decisiontree_entropy.fit(features, target)   # 모델 훈련
```

Tree와 랜덤 포레스트

- 결정 트리 분류기 훈련
  - 데이터 불순도를 측정하는 방법 - 지니 인덱스(Gini Index), 엔트로피(entropy), 분류오류(classification error)

지니 인덱스

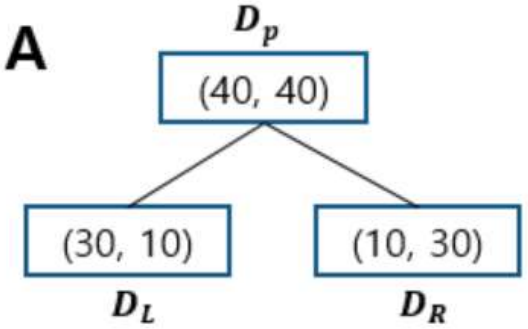
$$I_G(t) = 1 - \sum_{i=1}^{c} p(i\backslash t)^2$$

엔트로피

$$I_H(t) = -\sum_{i=1}^{c} p(i\backslash t)\log_2 p(i\backslash t)$$

분류오류

$$I_E(t) = 1 - \max\{p(i\backslash t)\}$$



의사결정트리 A의 경우 불순도 *I(Dp), I(DL), I(DR)* 및 정보이득 *IG* 계산

$$I_E(D_p) = 1 - 0.5 = 0.5$$

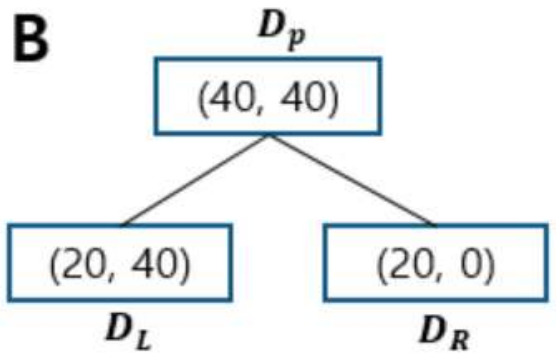
$$I_E(D_L) = 1 - \frac{3}{4} = 0.25$$

$$I_E(D_R) = 1 - \frac{3}{4} = 0.25$$

$$IG_E = 0.5 - \frac{40}{80} \times 0.25 - \frac{40}{80} \times 0.25 = 0.25$$

Tree와 랜덤 포레스트

➢ 결정 트리 분류기 훈련



의사결정트리 B의 경우 불순도 *I(Dp), I(DL), I(DR)* 및 정보이득 *IG* 계산

$$I_E(D_p) = 1 - 0.5 = 0.5$$

$$I_E(D_L) = 1 - \frac{4}{6} = \frac{1}{3}$$

$$I_E(D_R) = 1 - 1 = 0$$

$$IG_E = 0.5 - \frac{60}{80} \times \frac{1}{3} - 0 = 0.25$$

지니 인덱스로 계산한 경우
A에서 정보이득 : 0.125
B에서 정보이득 : 약 0.16

엔트로피로 계산한 경우
A에서 정보이득 : 0.19
B에서 정보이득 : 0.31

- 의사결정트리에서 정보이득이 최대가 되는 것을 채택해야 하므로, 분류오류를 적용하였을 때는 A, B가 모두 같은 값이 나와서 어떤 것을 선택하더라도 무관하지만, 지니 인덱스로 계산하거나 엔트로피로 계산한 경우에는 B가 A보다 정보 이득 값이 크므로 B를 선택하게 될것입니다.

Tree와 랜덤 포레스트

- 결정 트리 분류기 훈련
  - 지니 불순도는 클래스가 균등하게 분포되어 있을 때 최대가 됩니다.
  - 이진 클래스일 경우 클래스 샘플 비율이 0.5일 때 가장 큰 값이 됩니다.
  - 엔트로피도 클래스 샘플 비율이 균등할 때 가장 큰 값이 됩니다.

  - 지니 인덱스, 엔트로피, 분류오류 3가지 불순도 계산 방법 모두 0 또는 1에 가까와질 수록 불순도가 낮아지고, 0.5일 때 분순도가 가장 높게 나옵니다.
  - 데이터 세트에 여러 가지 부류에 속하는 데이터가 섞여 있는 경우, 특정 부류에 속하는 멤버입장에서 고려할 때 그 멤버가 차지하는 비율이 0에 가까와지거나 1에 가까와 질 때 가장 순도가 높고, 0.5일 때 분순도가 가장 높다..고 해석 됩니다.
  - criterion 매개변수는 불순도 계산 방법을 설정합니다. ('entropy' , 'gini')

Tree와 랜덤 포레스트

- 결정 트리 분류기 훈련
  - 트리 기반 학습 알고리즘은 분류와 회귀에서 사용되는 비모수 지도 학습 방법입니다.
  - 트리 기반 학습기의 기본은 일련의 결정 규칙이 연결된 결정 트리입니다.
  - 결정 규칙이 맨 위에 있고 이어지는 결정 규칙이 아래로 퍼져 있습니다.
  - 결정 트리에서 모든 결정 규칙은 결정 노드에서 일어납니다
  - 결정 규칙이 없는 마지막 가지를 리프라고 부릅니다.
  - 트리 기반 모델은 이해하기 쉽습니다.
  - 사이킷런의 DecisionTreeClassifier를 사용합니다.
  - 결정 트리 학습기는 노드에서 불순도가 가장 크게 감소하는 결정 규칙을 찾습니다.
  - DecisionTreeClassifier는 지니 불순도를 기본으로 사용합니다.

$$G(t) = 1 - \sum_{i=1}^{c} p_i^2$$

  - G(t)는 노드 t에서 지니 불순도이고 pi는 노드 t에서 클래스 c의 샘플 비율입니다
  - 불순도를 낮추는 결정 규칙을 찾는 과정은 모든가 순수해지거나 어떤 임계값에 도달할 때까지 반복됩니다.
  - 지니 불순도는 클래스가 균등하게 분포도어 있을 때 최대가 됩니다.
  - 예] 이진 클래스일 경우 클래스 샘플 비율이 0.5일때 가장 큰 값이 됩니다
  - 엔트로피 불순도 공식

$$H(t) = -\sum_{i=1}^{c} p_i \log_2 p_i$$

  - 엔트로피도 클래스 샘플 비율이 균등할 때 가장 큰 값이 됩니다.

  - 주어진 학습 데이터에 따라 생성되는 의사결정트리가 매우 달라져서 일반화하여 사용하기 어렵고 의사 결정 트리를 이용한 학습 결과 역시 성능과 변동의 폭이 크다는 단점을 가지고 있습니다.

Tree와 랜덤 포레스트

➢ 결정 트리 회귀 훈련

- 결정 트리 회귀는 지니 불순도나 엔트로피를 감소하는 대신 기본적으로 얼마나 평균 제곱 오차(MSE)를 감소시키는지에 따라 분할합니다.
- 사이킷런에서는 DecisionTreeRegressor를 사용하여 결정 트리 회귀를 수행할 수 있습니다
- criterion매개변수를 사용하여 분할 품질의 측정 방식을 선택할 수 있습니다.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2$$

```
from sklearn.tree import DecisionTreeRegressor
from sklearn import datasets

boston = datasets.load_boston()   # 데이터 로드
features = boston.data[:,0:2]   #두 개의 특성만 선택
target = boston.target

decisiontree = DecisionTreeRegressor(random_state=0)  # 결정 트리 회귀 모델 객체 생성
model = decisiontree.fit(features, target)   # 모델 훈련

observation = [[0.02, 16]]   #New 샘플 데이터
model.predict(observation)   # 샘플 데이터의 타깃을 예측

# 평균 제곱 오차를 사용한 (평균 절댓값 오차MAE가 감소되는) 결정 트리 회귀 모델 객체 생성
decisiontree_mae = DecisionTreeRegressor(criterion="mae", random_state=0)
model_mae = decisiontree_mae.fit(features, target)   # 모델 훈련
```

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \bar{y}|$$

Tree와 랜덤 포레스트

- 결정 트리 회귀 훈련
  - 사이킷런의 DecisionTreeRegressor를 사용합니다.
  - 트리 기반 학습 알고리즘은 분류와 회귀에서 사용되는 비모수 지도 학습 방법입니다
  - 지니 불순도나 엔트로피를 감소하는 대신 기본적으로 얼마나 평균 제곱 오차(MSE)를 감소시키는지에 따라 분할합니다.
  - criterion 매개변수 - 품질의 측정 방식을 선택할 수 있습니다.

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \bar{y}\right)^2$$

  - 평균 절댓값

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}\left|y_i - \bar{y}\right|$$

  - friendman_mse 방식 - 왼쪽 노드와 오른쪽 노드의 평균을 비교합니다.

$$\mathrm{FriedmanMSE} = \frac{n_{left} \times n_{right}\left(\mu_{left} - \mu_{right}\right)^2}{n_{left} + n_{right}}$$

Tree와 랜덤 포레스트

➢ 결정 트리 모델 시각화

- 결정 트리 분류기의 장점은 훈련된 전체 모델을 시각화할 수 있다는 것이다.
- 훈련된 모델을 DOT 포맷으로 변환한 다음 그래프를 그립니다.

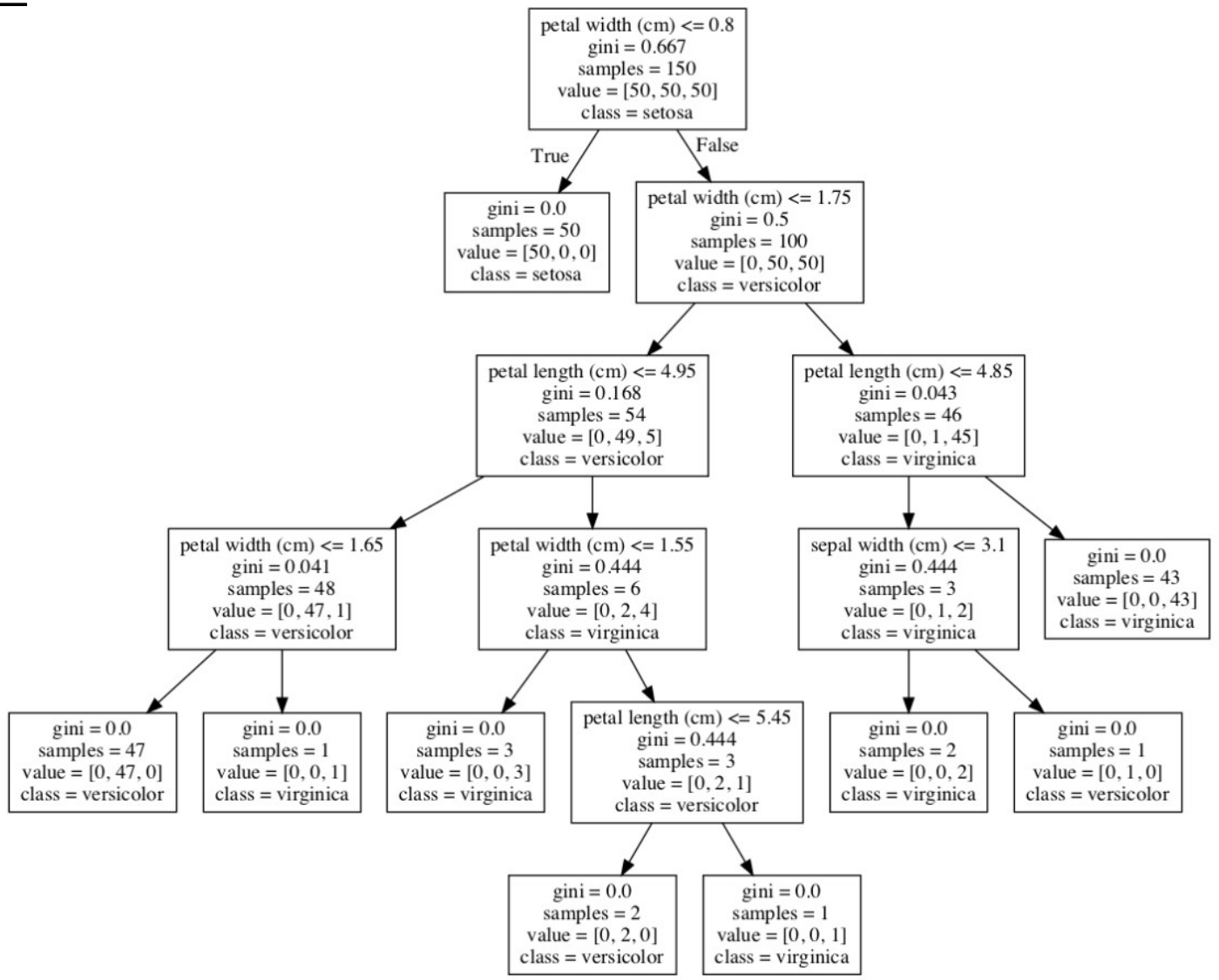
```
import pydotplus
from sklearn.tree import DecisionTreeClassifier
from sklearn import datasets
from IPython.display import Image
from sklearn import tree
iris = datasets.load_iris() # 데이터 로드
features = iris.data
target = iris.target
decisiontree = DecisionTreeClassifier(random_state=0) # 결정 트리 분류기를 만듭니다.

model = decisiontree.fit(features, target) # 모델 훈련
dot_data = tree.export_graphviz(decisiontree,
                                out_file=None,
                                feature_names=iris.feature_names,
                                class_names=iris.target_names) # DOT 데이터를 만듭니다

graph = pydotplus.graph_from_dot_data(dot_data) # 그래프를 그립니다.
Image(graph.create_png()) # 그래프 출력
graph.write_pdf("iris.pdf") # PDF를 만듭니다.
graph.write_png("iris.png") # PNG 파일을 만듭니다
```

Tree와 랜덤 포레스트

- 결정 트리 모델 시각화



Tree와 랜덤 포레스트

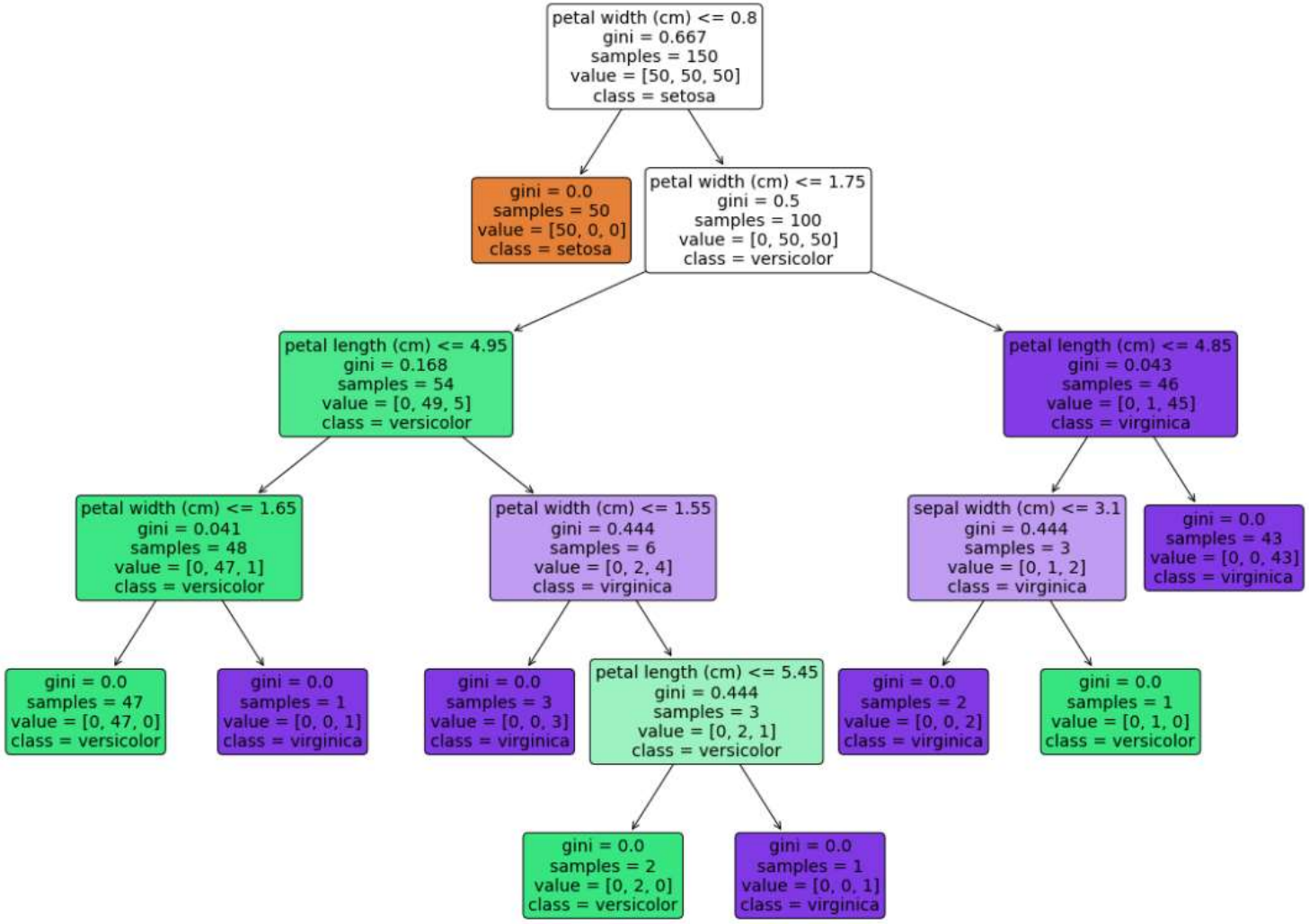
- 결정 트리 모델 시각화
  - export_graphviz()의 filled 매개변수를 True로 지정하면 노드마다 다수의 클래스에 따라 색이 채워집니다.
  - round 매개변수를 True로 지정하면 노드의 모서리를 라운드 처리합니다.
  - matplotlib 기반의 트리 그래프를 그려주는 plot_tree() 는 적절한 그래프 크기를 정의합니다.

```
import matplotlib.pyplot as plt

plt.figure(figsize=(20, 15))
tree.plot_tree(model, filled=True,
               feature_names=iris.feature_names,
               class_names=iris.target_names,
               rounded=True, fontsize=14)
plt.show()
```

Tree와 랜덤 포레스트

- 결정 트리 모델 시각화



Tree와 랜덤 포레스트

➢ 랜덤 포레스트 분류기 훈련

1단계 : 주어진 트레이닝 데이터 세트에서 무작위로 중복을 허용해서 n개 선택합니다.

2단계 : 선택한 n개의 데이터 샘플에서 데이터 특성값을 중복 허용없이 d개 선택합니다.

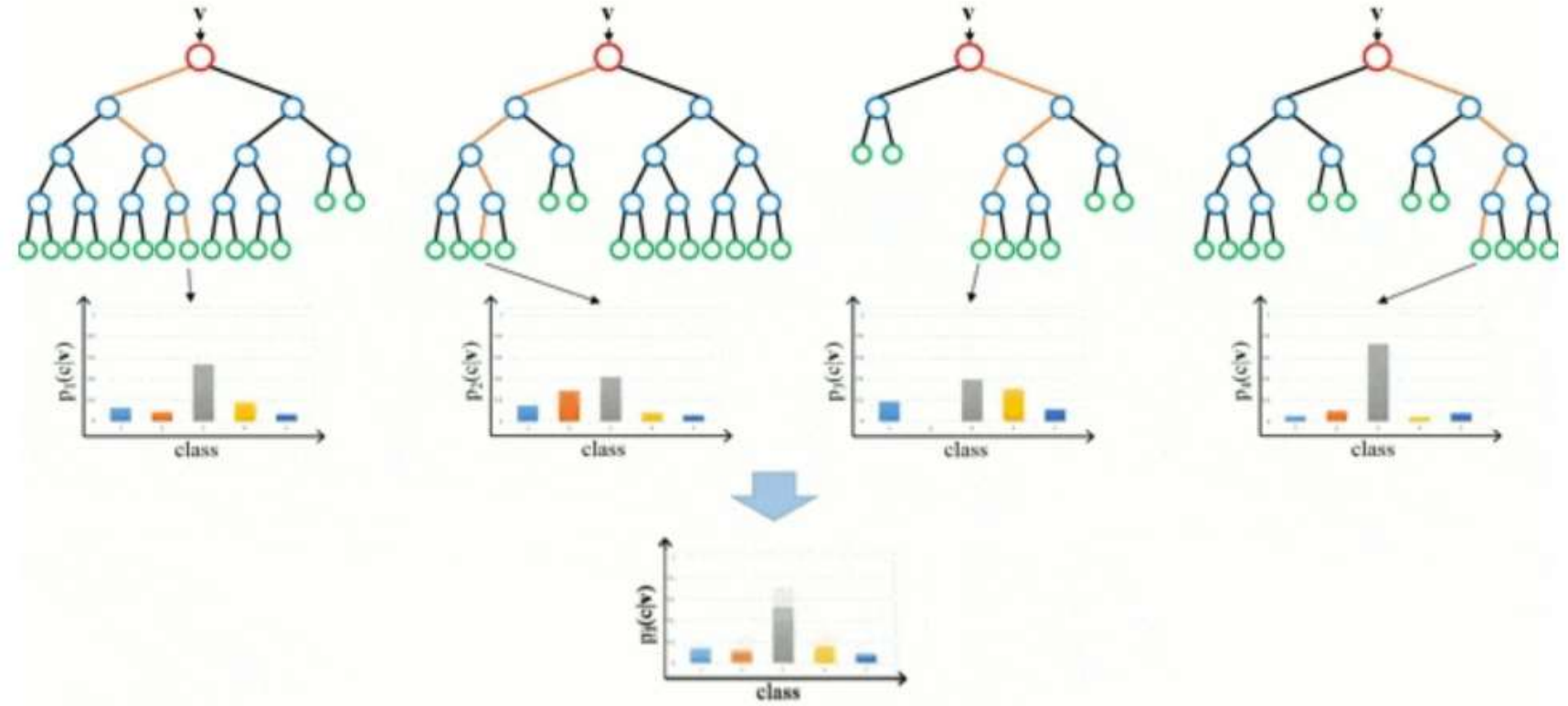
3단계 : 이를 이용해 의사결정트리를 학습하고 생성합니다.

4단계 : 1~3단계를 k번 반복합니다.

- 1~4단계를 통해 생성된 k개의 의사결정트리를 이용해 예측하고, 예측된 결과의 평균이나 가장 많이 등장한 예측 결과를 선택하여 최종 예측값으로 결정합니다.
- 1단계에서 무작위로 중복을 허용해서 선택한 n개의 데이터를 선택하는 과정을 부트스트랩(bootstrap)이라 부르며, 부트스트랩으로 추출된 n개의 데이터를 부트스트랩 샘플이라 부릅니다.
- scikit-learn이 제공하는 랜덤 포레스트 API는 부트스트랩 샘플의 크기 n의 값으로 원래 트레이닝 데이터 전체 개수와 동일한 수를 할당합니다.
- 2 단계에서 d값으로는 보통 주어진 트레이닝 데이터의 전체 특성의 개수의 제곱근으로 주어집니다.
- 여러 개의 의사결정트리로부터 나온 예측 결과들의 평균이나 다수의 예측 결과를 이용하는 방법을 앙상블(ensemble)기법이라고 합니다.
- 다수의 예측 결과를 선택하는 것은 다수결의 원칙과 비슷하다 해서 Majority Voting(다수 투표)이라고 부릅니다.
- 부트스트랩을 이용해 무작위 의사결정트리의 집합인 랜덤 포레스트를 구성하는 것처럼, 부트스트랩으로 다양한 분류기에 대해 앙상블 기법을 활용하여 특징적인 하나의 분류기로 구성하는 것을 bagging이라고 부릅니다.

Tree와 랜덤 포레스트

- 랜덤 포레스트 분류기 훈련
  - 부트스트랩을 이용해 무작위 의사결정트리의 집합인 랜덤 포레스트를 구성하는 것처럼, 부트스트랩으로 다양한 분류기에 대해 앙상블 기법을 활용하여 특징적인 하나의 분류기로 구성하는 것을 bagging이라고 부릅니다.



bagging을 통해 랜덤 포레스트를 구성하고 앙상블 기법을 이용하여 결과값을 예측하는 예

Tree와 랜덤 포레스트

- 랜덤 포레스트 분류기 훈련
  - 사이킷런의 RandomForestClassifier를 사용
  - 결정 트리의 일반적인 문제는 훈련 데이터에 너무 가깝게 맞추려는 경향이 있다는 점입니다.
  - 랜덤 포레스트는 많은 결정 트리를 훈련하지만 각 트리는 부트스트랩 샘플을 사용합니다. (원본 샘플 수와 동일하게 중복을 포함하여 랜덤하게 샘플을 뽑습니다.)
  - 랜덤 포레스트는 다수의 결정 트리들을 학습하는 앙상블 방법 입니다.
  - 랜덤 포레스트는 검출, 분류, 그리고 회귀 등 다양한 문제에 활용되고 있다.
  - 각 노드는 최적의 분할을 결정할 때 특성의 일부만 사용합니다.
  - max_features매개변수는 각 노드에서 사용할 특성의 최대 개수를 결정합니다.
  - max_features매개변수는 정수(특성의 수), 실수(특성 개수 비율), sqrt(특성 개수의 제곱근)를 입력할 수 있습니다.
  - max_features의 기본값은 auto로 sqrt와 동작이 같습니다.
  - bootstrap 매개변수는 트리에 사용할 샘플을 중복을 허용한 샘플링으로 만들지 아닐지를 결정합니다.
  - n_estimators는 랜덤 포레스트에서 만들 결정 트리 개수를 지정합니다. (의사결정트리의 개수 k의 값)
  - n_estimators를 하이퍼파라미터로 취급하여 트리 개수가 증가함에 따라 평가 지표에 미치는 영향을 확인했습니다.
  - n_jobs는 학습을 수행하기 위해 CPU 코어 개수를 병렬적으로 활용하라는 의미입니다. (n_jobs=-1 설정은 모든 코어를 사용합니다.)
  - 의사결정트리를 만드는 횟수 k는 생성되는 의사결정트리의 개수이며, 이 값이 커지면 예측 결과의 품질을 더 좋게 해주지만 컴퓨터의 성능 문제를 일으킬 수 있습니다.

Tree와 랜덤 포레스트

- 랜덤 포레스트 분류기 훈련
  - 사이킷런의 RandomForestClassifier를 사용

```
from sklearn.ensemble import RandomForestClassifier
from sklearn import datasets

iris = datasets.load_iris()  # 데이터 로드
features = iris.data
target = iris.target

# 랜덤 포레스트 분류기 객체를 만듭니다.
randomforest = RandomForestClassifier(random_state=0, n_jobs=-1)
model = randomforest.fit(features, target)  # 모델 훈련

observation = [[ 5,  4,  3,  2]] # 새로운 샘플을 만듭니다.
model.predict(observation)  # 샘플 클래스를 예측합니다.

# 엔트로피를 사용하여 랜덤 포레스트 분류기 객체를 만듭니다.
randomforest_entropy = RandomForestClassifier(criterion="entropy", random_state=0)
model_entropy = randomforest_entropy.fit(features, target)  # 모델 훈련
```

Tree와 랜덤 포레스트

➢ 랜덤 포레스트 회귀 훈련

- 결정 트리 회귀로 랜덤 포레스트 회귀 모델을 만들 수 있습니다.
- 각 트리는 부트스트랩 샘플을 사용하고 각 노드의 결정 규칙은 특성의 일부만 사용합니다.
- max_features 매개변수는 각 노드에서 사용할 최대 특성 개수를 지정합니다. (기본값은 전체 특성 개수입니다.)
- bootstrap 매개변수는 중복을 허용한 샘플링 여부를 지정합니다. (기본값은 True)
- n_estimators 매개변수는 사용할 결정 트리 개수를 지정합니다. (기본값은 10)

```
from sklearn.ensemble import RandomForestRegressor
from sklearn import datasets

boston = datasets.load_boston()     # 데이터 로드
features = boston.data[:,0:2]   #두 개의 특성만 선택
target = boston.target

# 랜덤 포레스트 회귀 객체 생성
randomforest = RandomForestRegressor(random_state=0, n_jobs=-1)
model = randomforest.fit(features, target)  # 모델 훈련
```

Tree와 랜덤 포레스트

- 랜덤 포레스트에서 중요한 특성 구분하기
  - fearure_importances_ 속성에서 특성의 상대적 중요도를 제공합니다 (값이 클수록 더 중요한 특성이며 특성 중요도의 전체 합은 1입니다.)

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn import datasets

iris = datasets.load_iris()  # 데이터 로드
features = iris.data
target = iris.target

randomforest = RandomForestClassifier(random_state=0, n_jobs=-1)  # 랜덤 포레스트 분류기 객체 생성
model = randomforest.fit(features, target)          # 모델 훈련
importances = model.feature_importances_          # 특성 중요도 계산
indices = np.argsort(importances)[::-1]           # 특성 중요도를 내림차순으로 정렬
names = [iris.feature_names[i] for i in indices]  # 정렬된 특성 중요도에 따라 특성의 이름을 나열
plt.figure()                                      # 그래프를 만듭니다.
plt.title("Feature Importance")                   # 그래프 제목 지정
plt.bar(range(features.shape[1]), importances[indices])  # 막대 그래프 추가
plt.xticks(range(features.shape[1]), names, rotation=90)  # x 축 레이블로 특성 이름을 사용
plt.show()  # 그래프 출력
```

Tree와 랜덤 포레스트

➢ 랜덤 포레스트에서 중요한 특성 구분하기

- 사이킷런에서는 순서가 없는 범주형 특성을 여러 개의 이진 특성으로 변환해야 합니다.
- 특성의 중요도 또한 여러 개의 이진 특성으로 나뉘게 됩니다.
- 원본 범주형 특성이 아주 중요하더라도 개별 이진 특성은 중요하지 않게 보일 수 있습니다.
- 두 특성의 상관관계가 크다면 한 특성이 중요하게 나타났을 때 다른 특성은 훨씬 중요하지 않게 보일 것입니다

```
decisiontree = DecisionTreeClassifier(random_state=0)  # 결정 트리 분류기 객체 생성
model = decisiontree.fit(features, target)  # 모델 훈련

# 랜덤 포레스트와 결정 트리의 특성 중요도를 비교합니다.
fig, ax = plt.subplots()
rects1 = ax.bar(np.arange(features.shape[1])-0.25,
                randomforest.feature_importances_, 0.5,
                label='Random Forest')
rects2 = ax.bar(np.arange(features.shape[1])+0.25, model.feature_importances_, 0.5,
                label='Decision Tree')
plt.xticks(range(features.shape[1]), names, rotation=90)
plt.legend()
plt.show()
```

Tree와 랜덤 포레스트

- 랜덤 포레스트에서 중요한 특성 선택하기
  - 모델의 분산을 감소시키거나 가장 중요한 특성만 사용하여 모델을 이해하기 쉽게 만들어야 하는 경우 모델의 특성 개수를 감소시켜야 합니다
  - 사이킷런에서는 두 단계의 워크플로를 사용하여 줄어든 특성으로 모델을 만들 수 있습니다.

1. 모든 특성을 사용해 랜덤 포레스트 모델을 훈련합니다.
2. 중요한 특성만 포함된 새로운 특성 행렬을 만듭니다. (SelectFromModel 클래스를 사용해 threshold 값보다 중요도가 크거나 같은 특성만 포함된 특성 행렬을 만듭니다.)
3. 중요한 특성만을 사용한 새로운 모델을 훈련합니다.

  - 단점1 : 원-핫 인코딩된 순서가 없는 범주형 특성의 중요도는 여러 개의 이진 특성으로 희석됩니다.
  - 단점2 : 상관관계가 높은 특성의 중요도는 양쪽 특성에 고루 분산되는 것이 아니라 한 특성에 집중됩니다.

Tree와 랜덤 포레스트

➢ 랜덤 포레스트에서 중요한 특성 선택하기

```
from sklearn.ensemble import RandomForestClassifier
from sklearn import datasets
from sklearn.feature_selection import SelectFromModel

iris = datasets.load_iris() # 데이터 로드
features = iris.data
target = iris.target
randomforest = RandomForestClassifier(random_state=0, n_jobs=-1) # 랜덤 포레스트 분류기 객체 생성
# 특성 중요도가 임계값보다 크거나 같은 특성으로 객체를 만듭니다.
selector = SelectFromModel(randomforest, threshold=0.3)

# selector를 사용하여 새로운 특성 행렬을 만듭니다.
features_important = selector.fit_transform(features, target)

# 가장 중요한 특성을 사용하여 랜덤 포레스트 모델을 훈련합니다.
model = randomforest.fit(features_important, target)
```

Tree와 랜덤 포레스트

- 불균형한 클래스 다루기
  - 불균형한 클래스를 적절히 처리하지 않으면 모델의 성능을 감소시킬 수 있습니다
  - 사이킷런의 많은 머신러닝 알고리즘은 불균형한 클래스를 바로 잡을 수 있는 방법을 내장하고 있습니다.
  - RandomForestClassifier 클래스의 class_weight 매개변수를 사용하여 불균형한 클래스를 교정할 수 있습니다.
  - 클래스 이름과 원하는 상대적 가중치를 딕셔너리로 만들어 주입하면 그에 따라 RandomForestClassifier가 클래스에 가중치를 부여합니다.
  - 매개변수값 balanced옵션은 데이터에 등장한 비율의 역수로 클래스 가중치를 자동으로 부여합니다.

Tree와 랜덤 포레스트

➢ 불균형한 클래스 다루기

```
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn import datasets

iris = datasets.load_iris()  # 데이터 로드
features = iris.data
target = iris.target
features = features[40:,:]  # 처음 40개의 샘플을 제거, 불균형한 데이터 생성
target = target[40:]  #
# 0인 클래스 이외에는 모두 1인 타깃 벡터를 만듭니다.
target = np.where((target == 0), 0, 1)

randomforest = RandomForestClassifier( random_state=0, n_jobs=-1, class_weight="balanced")
model = randomforest.fit(features, target)            # 모델  훈련
110/(2*10)                                            # 작은 클래스의 가중치를 계산
110/(2*100)                                           # 큰 클래스의 가중치를 계산
```

Tree와 랜덤 포레스트

- 트리 크기 제어
  - 매개변수를 사용하여 결정 트리의 구조와 크기를 수동으로 결정할 수 있습니다
  - 트리 기반 모델에서 과대적합을 막는 방법은 트리의 성장을 제한하는 것입니다.
  - 트리가 모두 성장한 후 노드를 줄이는 방법을 사후 가지치기라고 하고 성장하기 전에 막는 방법을 사전 가지치기라고 부릅니다.
  - 사이킷런은 사전 가지치기만 지원합니다.
  - 트리 모델에서 과대적합을 줄이려면 min_으로 시작하는 매개변수값을 증가시키거나 max_로 시작하는 매개변수값을 줄입니다.
  - max_depth : 트리의 최대 깊이. None이면 모든 리프 노드가 순수해질 때까지 트리가 성장합니다. 정수가 입력되면 트리는 그 깊이까지 성장합니다.
  - min_samples_split : 노드를 분할하기 위한 최소 샘플 개수. 정수가 입력되면 최솟값으로 사용됩니다. 실수가 입력되면 전체 샘플 개수의 비율을 의미합니다.
  - min_samples_leaf : 리프 노드가 되기 위한 최소 샘플 개수. min_samples_split와 동일한 매개변수값을 사용합니다.
  - max_leaf_nodes : 리프 노드의 최대 개수
  - min_impurity_split : 분할하기 위한 불순도 최소 감소량 (min_impurity_decrease)
  - min_weight_fraction_leaf : 가중치가 부여된 전체 샘플 개수에 대한 비율

Tree와 랜덤 포레스트

➢ 트리 크기 제어

```
from sklearn.tree import DecisionTreeClassifier
from sklearn import datasets

iris = datasets.load_iris() # 데이터 로드
features = iris.data
target = iris.target

# 결정 트리 분류기 객체 생성.
decisiontree = DecisionTreeClassifier(random_state=0,
                                      max_depth=None,
                                      min_samples_split=2,
                                      min_samples_leaf=1,
                                      min_weight_fraction_leaf=0,
                                      max_leaf_nodes=None,
                                      min_impurity_decrease=0)

model = decisiontree.fit(features, target) # 모델 훈련
```

Tree와 랜덤 포레스트

- 부스팅을 사용한 성능 향상
  - 결정 트리나 랜덤 포레스트보다 더 높은 성능를 가진 모델
  - AdaBoost 부스팅 형식은 이전 모델이 잘못 예측한 샘플에 높은 우선순위를 부여하는 식으로 약한 모델을 연속적으로 훈련합니다.

1. 모든 샘플 x_i에 초기 가중치 값 $w_i = \frac{1}{n}$을 할당합니다. 여기에서 n은 데이터에 있는 샘플의 전체 개수입니다.
2. 이 데이터에서 약한 모델을 훈련합니다.
3. 각 샘플에 대하여
   a. 모델이 x_i를 올바르게 예측하면 w_i를 낮춥니다.
   b. 모델이 x_i를 잘못 예측하면 w_i를 높입니다.
4. w_i가 큰 샘플에 높은 우선순위를 두는 새로운 약한 모델을 훈련합니다.
5. 데이터가 완벽하게 예측되거나 지정된 개수만큼 모델을 훈련할 때까지 단계 4와 5를 반복합니다.

  - 최종 결과는 예측 측면에서 더 어려운 샘플에 초점을 맞추는 약한 모델들을 모은 앙상블 모델입니다.
  - 사이킷런에는 AdaBoostClassifier와 AdaBoostRegressor 클래스에 에이다 부스트가 구현되어 있습니다.

Tree와 랜덤 포레스트

- 부스팅을 사용한 성능 향상
  - base_estimator : 약한 모델을 훈련하는 데 사용할 학습 알고리즘 (기본값은 결정 트리)
  - n_estimators : 반복적으로 훈련할 모델의 개수
  - learning_rate : 각 모델이 부여하는 가중치 정도 (기본값은 1) 학습률을 감소하면 가중치 감소나 증가량이 줄어들기 때문에 모델의 훈련 속도를 느리게 만듭니다.
  - loss : 가중치를 업데이트할 때 사용하는 손실 함수를 지정 (AdaBoostRegressor에만 해당)

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn import datasets

iris = datasets.load_iris()        # 데이터 로드
features = iris.data
target = iris.target

adaboost = AdaBoostClassifier(random_state=0)       # 에이다부스트 트리 분류기의 객체 생성

model = adaboost.fit(features, target)      ## 모델 훈련
```

Tree와 랜덤 포레스트

- 부스팅을 사용한 성능 향상
  - AdaBoostClassifier는 예측할 때 각 학습기에 부여된 가중치를 더하여 가장 높은 점수의 클래스가 예측 결과가 됩니다.
  - AdaBoostRegressor의 예측은 개별 학습기의 결과를 정렬하여 예측기 가중치의 누적값이 중간 지점에 있는 결과를 사용
  - Graident Boosting은 AdaBoost와 달리 이전 학습기가 만든 잔여 오차에 새로운 트리를 훈련하는 방식으로 앙상블 모델을 구성하여 높은 성능을 냅니다.
  - GradientBoostingClassifier와 GradientBoostingRegressor 모두 깊이가 3이고 criterion이 'friedman_mse'인 DecisionTreeRegressor를 사용합니다

```
from sklearn.ensemble import GradientBoostingClassifier

# 그래디언트 부스팅 분류기의 객체를 만듭니다.
gradientboost = GradientBoostingClassifier(random_state=0)

model = gradientboost.fit(features, target)  # 모델 훈련
```

Tree와 랜덤 포레스트

- 부스팅을 사용한 성능 향상
  - 히스토그램 기반의 그레이디언트 부스팅 : XGBoost, LightGBM 라이브러리에 구현된 HistGradientBoostingClassifier와 HistGradientBoostingRegressor
  - 훈련 데이터를 정수 구간(bin)으로 변환한 후 훈련
  - GradientBoosting 보다 빠름
  - max_bins(구간의 최대 개수 지정)의 기본값은 256

```
from sklearn.experimental import enable_hist_gradient_boosting
from sklearn.ensemble import HistGradientBoostingClassifier

# 히스토그램 기반의 그래디언트 부스팅 분류기의 객체를 만듭니다.
histgradientboost = HistGradientBoostingClassifier(random_state=0)

model = histgradientboost.fit(features, target) # 모델 훈련
```

Tree와 랜덤 포레스트

- **OOB(out-of-bag) 데이터로 랜덤 포레스트 평가**
  - 랜덤 포레스트에서 각 결정 트리는 부트스트랩 샘플을 사용하여 훈련됩니다.
  - 즉 모든 트리는 서로 다른 일부 샘플을 훈련에 사용하지 않습니다. (OOB 샘플)
  - OOB 샘플을 테스트 세트처럼 사용하여 랜덤 포레스트의 성능을 평가할 수 있습니다.
  - 특정 샘플을 사용하여 훈련되지 않은 트리를 통해 학습 알고리즘은 해당 샘플의 정답과 예측값을 비교합니다.
  - 전체 점수를 계산하여 랜덤 포레스트의 성능을 측정합니다.
  - oob_score=True로 지정하면 OOB 점수를 계산할 수 있습니다.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn import datasets

iris = datasets.load_iris()  # 데이터 로드
features = iris.data
target = iris.target
# 랜덤 포레스트 분류기  객체 생성
randomforest = RandomForestClassifier( random_state=0, n_estimators=1000, oob_score=True, n_jobs=-1)
model = randomforest.fit(features, target)   # 모델 훈련
randomforest.oob_score_   # OOB 오차를 확인
```

Tree와 랜덤 포레스트

- **OOB(out-of-bag) 데이터로 랜덤 포레스트 평가**
  -

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier

# 배깅 분류기  객체 생성
bagging = BaggingClassifier(DecisionTreeClassifier(), n_estimators=100,  random_state=0, oob_score=True,
n_jobs=-1)
model = bagging.fit(features, target)   # 모델 훈련
model.oob_score_  # OOB 오차를 확인
```