제2차 빅데이터 분석 교육과정 강의교재

- 가완성본 -

ABRC 인공지능-빅데이터연구센터
AI-BIG DATA RESEARCH CENTER

트리를 이용한 데이터 분석

1. 의사결정 트리를 이용한 데이터 분석

1.1 의사결정 트리(Decision Tree)란?

- 의사결정 트리 또는 의사결정 나무(Decision Tree)는 기계학습(Machine Learning)에서 지도학습(Supervised Learning)의 알고리즘으로 분류(Classification) 또는 회귀(Regression)분석 목적으로 사용
- 의사결정 규칙(Decision Rule)을 나무구조 표현을 통해 분류와 예측을 수행하는 분석방법
- 분류 또는 예측 과정이 나무구조로 표현되어 비교적 쉽게 이해
- 목표변수 유형에 따른 의사결정 트리
  - 범주형 목표변수 : 분류 트리(Classification Tree)
    - 목표변수가 이산형인 경우, 각각의 범주에 속하는 빈도에 기초해 분리 발생 - 분류 트리 구성
  - 연속형 목표변수 : 회귀 트리(Regression Tree)
    - 목표변수가 연속형인 경우, 평균과 표준편차에 기초해 분리 발생 - 회귀 트리 구성

의사결정 트리는 기계학습에서, 지도학습 유형의 알고리즘으로 분류 또는 회귀분석 목적으로 사용됩니다. 의사결정 트리를 이용하면 분류와 예측을 해 볼 수 있습니다. 목표변수가 범주형일 때, 의사결정 트리는 분류 트리가 됩니다. 각각의 범주에 속하는 빈도에 기초해 분리를 진행하여 분류 트리가 구성됩니다. 반면에 목표변수가 연속형일 때, 의사결정 트리는 회귀 트리가 됩니다. 이 때 이 회귀 트리는 평균과 표준편차에 기초하여 만들게 되죠.

1.2 Decision Tree 구성요소

이런 의사결정 트리는 여러 개의 노드와 그 노드들을 잇는 가지들로 구성됩니다. 그림 1은 의사결정 트기 구성요소의 예를 나타냅니다.

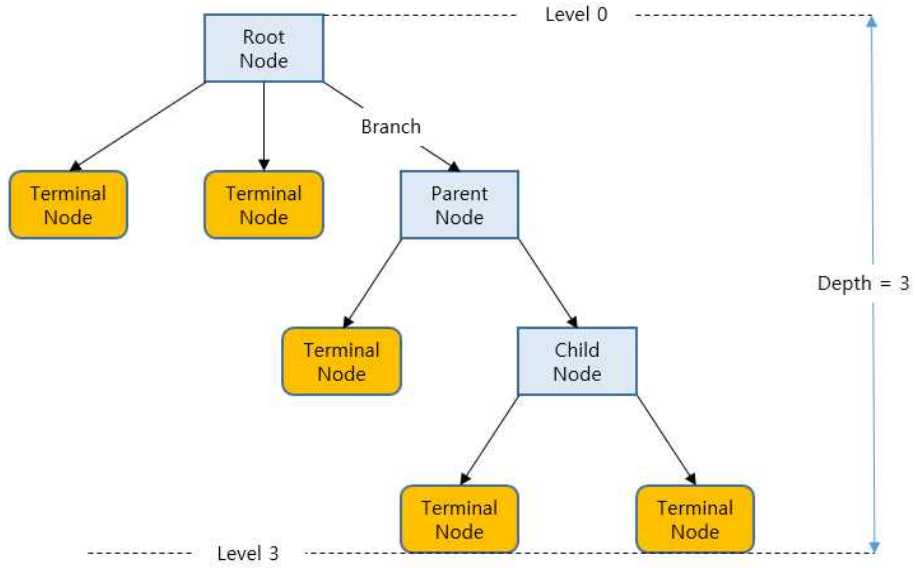


그림 1 트리구성 요소

- 뿌리 마디 (Root Node)
  - 트리구조가 시작되는 마디, 전체 자료로 구성

- 부모 마디 (Parent Node)
  - 자식 마디의 상위 마디

- 자식 마디 (Child Node)
  - 하나의 마디로부터 분리되어 나간 2개 이상의 마디들을 의미

- 끝 마디 (Terminal Node) 또는 잎(Leaf Node)
  - 트리 줄기의 끝에 위치하고 있고 자식 마디가 없는 마디

- 가지 (Branch)
  - 뿌리 마디로부터 끝 마디까지 연결된 마디들

- 깊이 (Depth)
  - 뿌리 마디로부터 끝 마디를 이루는 층의 수

1.3 Decision Tree 분석과정

- 성장 (Growing)
  - 분석의 목적에 따라 각 마디에서 적절한 최적의 분리기준(Split

Criterion)을 찾아 트리를 성장시키는 과정, 적절한 정지규칙(Stopping Rule)을 통한 의사결정 트리 도출

- 가지치기 (Pruning)
  - 오분류를 크게 할 위험(Risk)이 높거나 부적절한 추론규칙(Induction Rule)을 가지는 불필요한 가지(Branch)를 제거

- 타당성 평가
  - 이익 도표(Gain Chart), 위험 도표(Risk Chart) 또는 검증용 자료(Test Data)로 의사결정 트리 평가

- 해석 및 예측
  - 의사결정 트리를 해석하고 예측모형 결정

1.4 예제를 이용한 Decision Tree 동작 과정

이번에는 예제 데이터를 이용해 의사결정 트리가 작동하는 과정에 대해 알아보겠습니다. 우리가 사용할 예제 데이터는 Tennis 데이터입니다. 표 1은 Tennis 예제 데이터입니다.

Day	Outlook	Temperature	Humidity	Play Tennis
D1	Sunny	Hot	High	No
D2	Overcast	Hot	High	No
D3	Overcast	Mild	Normal	Yes
D4	Rain	Mild	Normal	Yes
D5	Sunny	Mild	High	No
D6	Rain	Hot	High	No
D7	Overcast	Hot	Normal	No

표 1 Tennis 예제 데이터

예제 데이터의 속성으로는 Outlook, Temperature, Humidity가 있습니다. Outlook은 흐림, 맑음, 비와 같은 값을 갖습니다. Temperature의 값으로는 Hot, Mild가 있고 Humidity의 값으로는 High, Normal이 있습니다. 예제 데이터의 클래스인 범주를 나타내는 PlayTennis는 No 또는 Yes가 될 수 있습니다. No는 테니스를 치지 못한다는 의미이고, Yes는 테니스를 칠 수 있다는 것을 의미합니다. 즉, Outlook, Temperature, Humidity의 값에 따라 테니스를 할 수 있는지 또는 하지 못하는 날씨인지를 분류하는 것이 목적입니다.

이 의사결정 트리 모델의 뿌리 노드는 Outlook으로 시작됩니다. Outlook에 있는 개체에 의해 왼쪽엔 Sunny, 오른쪽엔 Overcast와 Rain을 배치했습니다. Outlook의 값이 Sunny인 경우 PlayTennis의 값은 'No'만 존재합니다. 따라서 노드 값이 'No'인 말단 노드가 생성되는 거죠. Outlook이 Overcast 또는 Rain인 경우 PlayTennis의 값은 어떻게 될까요?

'No'와 'Yes' 두개 값을 가질 수 있죠? 따라서 다음 하위 노드로 분리가 진행될 수 있는 부모 노드가 만들어집니다. 이렇게 만들어진 부모 노드에서는 Temperature 속성의 값을 테스트해보게 됩니다. Temperature가 가질 수 있는 두 개의 값에 따라 왼쪽엔 Hot, 오른쪽엔 Mild로 분리합니다. Hot과 Mild로 각각 분리된 왼쪽과 오른쪽 노드 위치에, 속성 Humidity인 부모 노드를 만듭니다. 속성 Humidity의 왼쪽 아래는 High, 오른쪽은 Normal로 분리합니다. 그림 2는 예제 데이터에 대한 의사결정 트리 모델입니다.

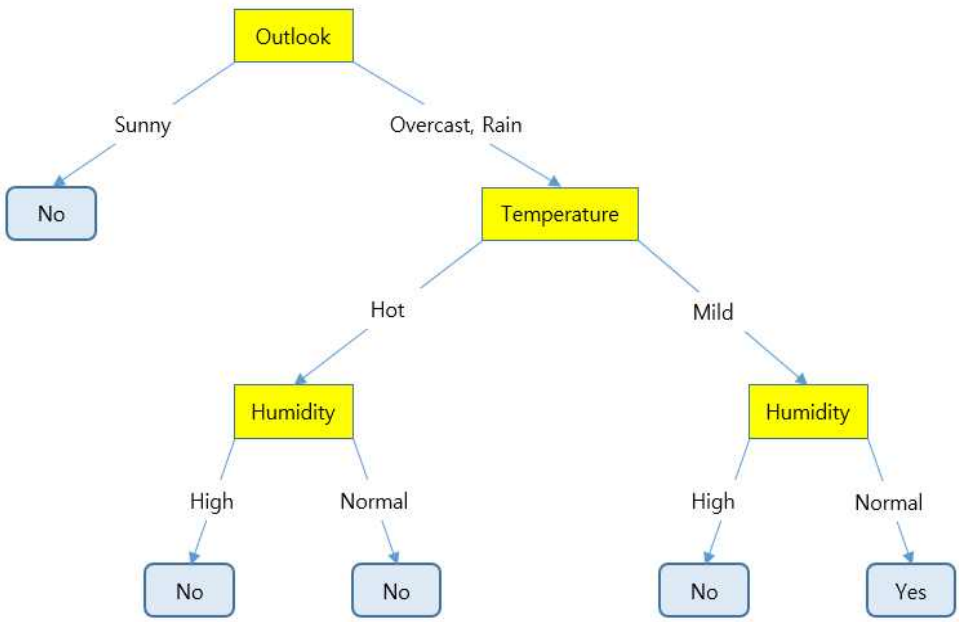
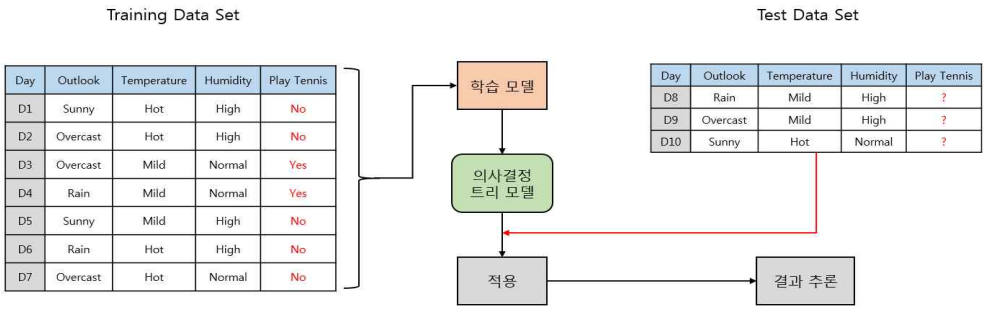


그림 2 예제 데이터에 대한 의사결정 트리 모델

이렇게 훈련 데이터를 통해 만들어진 의사결정 트리 모델에 테스트 데이터의 속성 값을 입력해보겠습니다. 테스트 데이터의 구성은 다음과 같습니다. 속성 Outlook의 개체는 Rain, 속성 Temperature의 개체는 Mild, 속성 Humidity의 개체는 High일 때 PlayTennis의 값은 무엇이 나오는지 알아보겠습

니다. 그림 3은 의사결정 트리 모델에 대한 테스트에 대한 내용입니다.



Day	Outlook	Temperature	Humidity	Play Tennis
D1	Sunny	Hot	High	No
D2	Overcast	Hot	High	No
D3	Overcast	Mild	Normal	Yes
D4	Rain	Mild	Normal	Yes
D5	Sunny	Mild	High	No
D6	Rain	Hot	High	No
D7	Overcast	Hot	Normal	No

Day	Outlook	Temperature	Humidity	Play Tennis
D8	Rain	Mild	High	?
D9	Overcast	Mild	High	?
D10	Sunny	Hot	Normal	?

그림 3 의사결정 트리 모델에 대한 테스트

표 2는 test data에 대한 내용입니다. test data에서 Outlook 속성의 값은 Rain입니다. 이에 따라 Rain이 있는 오른쪽으로 내려갑니다. 그림 4는 test data에 있는 Outlook 속성에 의해 동작되는 의사결정 트리 모델 과정에 대한 내용입니다.

Outlook	Temperature	Humidity	Play Tennis
Rain	Mild	High	?

표 2 Test data

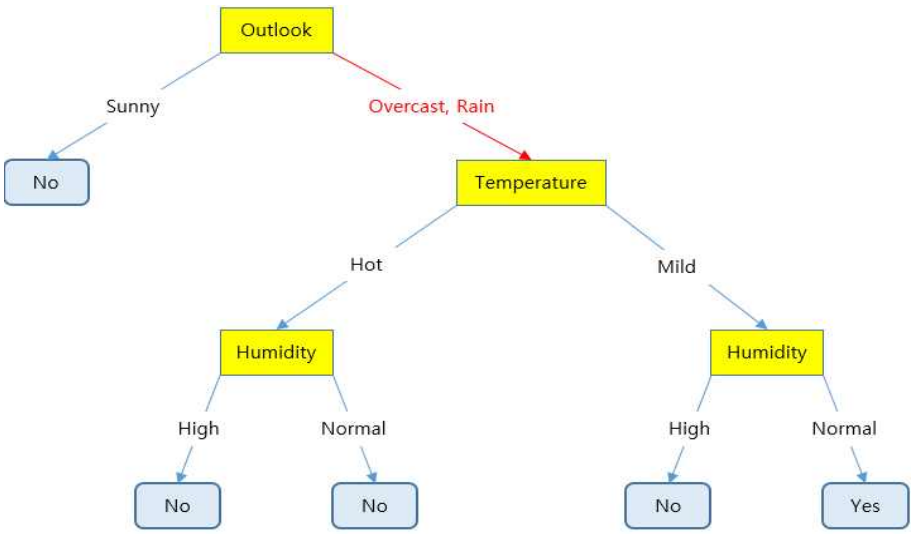


그림 4 의사결정 트리 모델 동작과정 (1)

다음, 속성 Temperature 노드에서는 Mild가 있는 오른쪽으로 분리됩니다. 그림 5는 test data에 있는 Temperature 속성에 의해 동작되는 의사결정 트리 모델 과정에 대한 내용입니다.

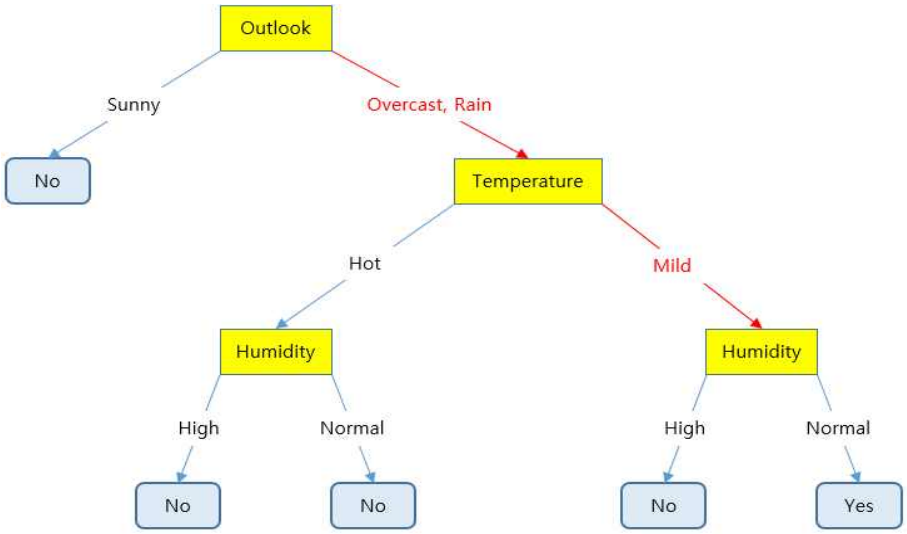


그림 5 의사결정 트리 모델 동작과정 (2)

속성 Humidity 노드에서는 High가 있는 왼쪽으로 분리됩니다. 그림 6은 test data에 있는 Humidity 속성에 의해 동작되는 의사결정 트리 모델 과정에 대한 내용입니다.

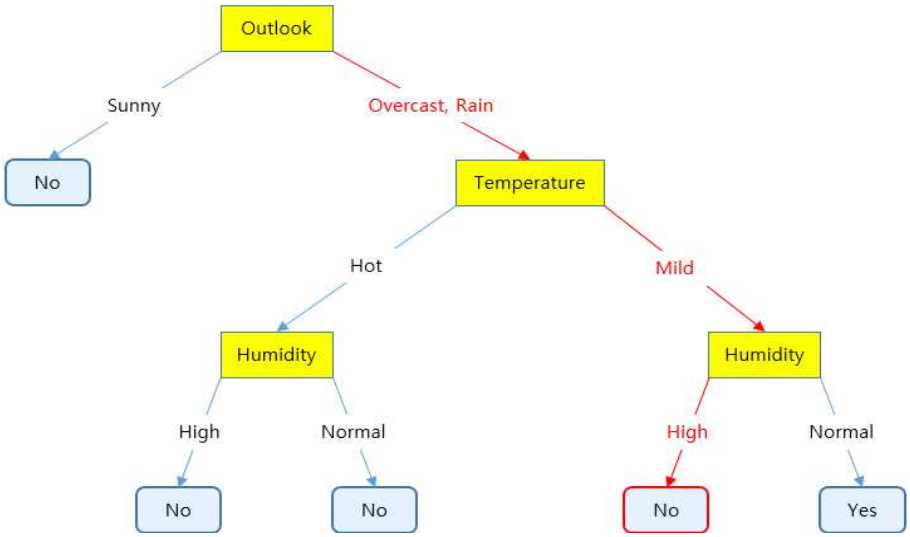


그림 6 의사결정 트리 모델 동작과정 (3)

이렇게, 우리가 만든 의사결정 트리 모델에 테스트 데이터의 모든 속성을 입력해 분류한 결과, PlayTennis 클래스의 값은 'No' 라는 것을 알게 되었습니다.

1.5 Decision Tree 분리기준 (Split Criterion)

- 부모 노드로부터 자식 노드들이 형성될 때, 생성된 자식 노드에 속하는 자료의 순수도(Purity)가 가장 크게 증가하도록 트리를 형성하며 진행
- 입력 변수를 이용해 목표 변수의 분포를 얼마나 잘 구별하는 정도를 파악해 자식 마디가 형성되는데, 목표 변수의 구별 정도를 불순도(Impurity, 다양한 범주들의 개체들이 포함되어 있는 정도)에 의해 측정

분리 기준은 부모 노드로부터 자식 노드들이 형성될 때, 생성된 자식 노드에 속하는 자료의 순수도가 가장 크게 증가하도록 트리를 형성하면서 진행합니다. 또, 입력 변수를 이용하여 목표 변수의 분포를 얼마나 잘 구별하는가하는 정도를 파악하여 자식 마디를 형성 합니다.목표 변수의 구별 정도는 불순도에 의해 측정합니다.

불순도는 다양한 범주들의 개체들이 포함되어 있는 정도를 뜻합니다. 의사결정 트리의 분리 기준은 지니 지수, 엔트로피 지수, 정보 이득 등 여러 가지 분리 기준이 있습니다.

- 지니 지수 (Gini Index)
  - 데이터 집합의 불순도를 측정
  - 지니 지수는 0~1 사이의 값을 가지며, 어떤 데이터 집합에 속한 개체(레코드)들이 같은 범주(클래스)로 구성되어 있으면 지니 지수는 최솟값이 0을 갖고 해당 데이터 집합은 순수하다고 볼 수 있음
  - 즉, 지니 지수가 작을수록 잘 분류된 것으로 볼 수 있음

- 엔트로피 지수 (Entropy Index)
  - 엔트로피는 주어진 데이터 집합의 혼잡도를 의미
  - 주어진 데이터 집합에 서로 다른 범주(클래스)의 개체(레코드)들이 많이 섞여 있으면 엔트로피가 높고, 같은 범주의 개체들이 많이 있으면 엔트로피가 낮음
  - 엔트로피 지수는 0~1 사이의 값을 가지며, 가장 혼잡도가 높은 상태(서로 다른 범주의 개체들이 섞여있는 상태)는 1, 혼잡도가 가장 낮은 상태(하나의 범주의 개체로 구성된 상태)는 0

- 정보 이득 (Information Gain)

- 상위 노드의 엔트로피 지수에서 하위 노드의 가중 평균한 엔트로피 지수를 뺀 것을 의미
- 즉, 원래 상위 노드의 엔트로피를 구하고 어떤 속성을 선택한 후의 x개의 하위 노드로 분리된 것에 대한 가중 평균한 엔트로피를 구한 값의 차를 의미(계산되 나온 값이 클수록 정보 이득이 큰 것을 의미, 선택한 어떤 속성이 분류하기 좋다고 볼 수 있음)

지니 지수는 데이터 집합의 불순도를 측정하는 방법 중 하나입니다. 지니 지수 값의 범위는 0~1 사이인데요, 어떤 데이터 집합에 속한 개체들이 같은 범주로 구성되어 있으면 지니 지수는 최솟값 0을 가지며, 해당 데이터 집합은 순수하다고 볼 수 있습니다. 지니 지수가 작을수록 분류가 잘 된 것으로 볼 수 있습니다.

엔트로피 지수는 주어진 데이터 집합에 대한 혼잡도를 의미합니다. 주어진 데이터 집합에 서로 다른 범주의 개체들이 많이 혼합된 경우 엔트로피 지수가 높고, 같은 범주의 개체들이 많이 있으면 엔트로피 지수가 낮습니다. 엔트로피 지수는 0~1 사이의 값을 가집니다. 서로 다른 범주의 개체들이 혼합되어 있는 상태인 혼잡도가 높은 상태는 엔트로피 지수가 1입니다. 하나의 범주의 개체로 구성된 상태인 혼잡도가 가장 낮은 상태는 엔트로피 지수가 0일 때입니다.

정보 이득은 상위 노드의 엔트로피 지수에서 하위 노드의 가중 평균한 엔트로피 지수를 뺀 것을 의미합니다. 이렇게 의사결정 트리의 분리 기준은 지니 지수, 엔트로피 지수, 정보 이득 등 여러 가지 분리 기준이 있습니다. 여러 분리 기준 중 지니 지수를 이용해 의사결정 트리를 구성하는 방법에 대해서만 살펴보겠습니다. 지니 지수 계산식은 다음과 같습니다.

$$G = 1 - \sum_{j=1}^{c} P(j)^2 = 1 - \sum_{j=1}^{c} \left(\frac{n_j}{n}\right)^2$$

지니 지수 식 $G=1-\sum_{j=1}^{c}P(j)^2$에서 c는 범주의 수를 의미하고, P(j)는 j번째 범주에 분류될 확률을 의미합니다.

같은 결과를 갖는 지니 지수 계산 식 $1-\sum_{j=1}^{c}(\frac{n_j}{n})^2$에서 n은 노드에 속하는 개체 수, n_j는 노드에 속하는 수 중 j번째 범주에 속하는 개체의 수를 의미합니다.

1.6 예제를 이용한 Decision Tree 실습

1.6.1 예제를 이용한 Decision Tree 지니 지수 계산

속성 Outlook, Humidity, Wind와 범주 PlayTennis를 갖는 예제 데이터에서 속성별로 지니 지수를 계산해 보겠습니다. 표 3은 지니 지수 계산에 사용되는 예제 데이터입니다.

Outlook	Humidity	Wind	Class (Play Tennis)
Sunny	High	Weak	N
Sunny	High	Strong	N
Overcast	High	Weak	Y
Rain	High	Weak	Y
Rain	Normal	Strong	Y
Rain	Normal	Strong	N
Overcast	Normal	Weak	Y
Sunny	High	Weak	N
Sunny	Normal	Weak	Y
Rain	High	Weak	N
Sunny	Normal	Strong	Y
Overcast	High	Weak	Y
Overcast	Normal	Weak	N
Rain	High	Strong	N

표 3 예제 데이터

먼저, 속성 Outlook으로 분리하는 경우입니다. 속성 Outlook의 개체 Sunny는 Left, Overcast와 Rain은 Right로 정의합니다. 그리고 범주 PlayTennis에서 첫 번째 범주를 N, 두 번째 범주를 Y로 정의하겠습니다. 속성 Outlook의 총 개체 수는 14입니다.

Outlook 속성으로 분리하는 경우

- Left = {Sunny}, Right = {Overcast, Rain}일 때

	Class = N	Class = Y	
Left	3	2	5
Right	4	5	9
	7	7	14

이 중 Left 노드 즉, Sunny에 속하는 개체 수는 5개입니다. Left 노드에 속하는 수 중 첫 번째 범주 N에 속하는 개체 수는 3개, 그리고 두 번째 범주 Y에 속하는 개체 수는 2개입니다.

Right 노드 즉, Overcast와 Rain에 속하는 개체 수는 9개입니다. Right 노드에 속하는 수 중 첫 번째 범주 N에 속하는 개체 수는 4개, 그리고 두 번째 범주 Y에 속하는 개체 수는 5개입니다.

이제, Left와 Right 각 노드에 대한 지니 지수를 계산해 보겠습니다. 먼저, Left 노드에 대한 지니 지수 계산입니다. Left 노드의 지니 지수 계산식은 다음과 같습니다.

$$1-\left\{\left(\frac{3}{5}\right)^2+\left(\frac{2}{5}\right)^2\right\}$$

계산식에 있는 분수 $\frac{3}{5}$에서 분모 5는 Left 노드에 속하는 개체 수 5개를 의미하고, 분자 3은 Left 노드에 속하는 수 중 첫 번째 범주인 N에 속하는 개체 수 3개를 의미합니다. 분수 $\frac{2}{5}$에서 분모 5는 Left 노드에 속하는 개체 수 5개를 의미하고, 분자 2는 Left 노드에 속하는 수 중 두 번째 범주인 Y에 속하는 개체 수 2개를 의미합니다. 중괄호 안에 있는 분수의 제곱 계산을 먼저 하겠습니다.

Left 노드에 속하는 수 중 첫 번째 범주인 N에 속하는 개체 수 / Left 노드에 속하는 개체 수인 $\frac{3}{5}$을 제곱한 것과, Left 노드에 속하는 수 중 두 번째 범주인 Y에 속하는 개체 수 / Left 노드에 속하는 개체 수인 $\frac{2}{5}$를 제곱한 것

을 더합니다. $\frac{3}{5}$과 $\frac{2}{5}$는 각각 0.6과 0.4로 나타낼 수 있습니다. 0.6을 제곱한 값인 0.36과 0.4를 제곱한 값인 0.16을 더한 값은 0.52가 됩니다.

1에서 중괄호 안에서 계산되어 나온 값인 0.52를 빼주면 Left 노드에 대한 지니 지수가 계산이 됩니다. 계산되어 나온 Left 노드의 지니 지수는 0.48입니다.

다음은, Right 노드에 대한 지니 지수 계산입니다. Right 노드의 지니 지수 계산식은 다음과 같습니다.

$$1-\left\{\left(\frac{4}{9}\right)^2+\left(\frac{5}{9}\right)^2\right\}$$

계산식에 있는 분수 $\frac{4}{9}$에서 분모 9는 Right 노드에 속하는 개체 수 9를 의미하고, 분자 4는 Right 노드에 속하는 수 중 첫 번째 범주인 N에 속하는 개체 수 4를 의미합니다. 분수 $\frac{5}{9}$에서 분모 9는 Right 노드에 속하는 개체 수 9를 의미하고, 분자 5는 Right 노드에 속하는 수 중 두 번째 범주인 Y에 속하는 개체 수 5를 의미합니다.

Left 노드의 지니 지수 계산 순서와 같이 중괄호 안에 있는 분수의 제곱 계산을 먼저 하겠습니다.

Left 노드에 속하는 수 중 첫 번째 범주인 N에 속하는 개체 수 / Left 노드에 속하는 개체 수인 $\frac{4}{9}$를 제곱한 것과 Left 노드에 속하는 수 중 두 번째 범주인 Y에 속하는 개체 수 / Left 노드에 속하는 개체 수인 $\frac{5}{9}$를 제곱한 것을 더합니다.

$\frac{4}{9}$와 $\frac{5}{9}$는 각각 0.44와 0.56으로 나타낼 수 있습니다. 0.44를 제곱한 값인 0.1936과 0.56을 제곱한 값인 0.3136을 더한 값은 0.5072가 됩니다. 중괄호 안에서 계산되어 나온 값인 0.5072를 1에서 빼주면 Right 노드에 대한 지니 지

수가 계산됩니다. 계산되어 나온 Right 노드의 지니 지수는 0.4928입니다.

지니 지수는 작을수록 노드 구성이 잘된 것을 의미합니다. Left 노드의 지니 지수는 0.48, Right 노드의 지니 지수는 0.4928로 Right 노드보다 지니 지수가 더 작은 Left 노드가 더 구성이 잘된 것을 알 수 있습니다. 지금까지 속성 Outlook과 범주 Play Tennis를 이용해 지니 지수 계산에 대해 알아보았습니다.

속성 Humidity와 Wind, 그리고 범주 Play Tennis를 이용한 지니 지수 계산 방법은 앞서 진행된 속성 Outlook에 대한 지니 지수 계산 방법과 동일합니다. 이제부터 예제를 이용한 의사결정 트리 파이썬 코드에 대해 알아보겠습니다.

1.6.2 예제를 이용한 Decision Tree 파이썬 코드 실습

1.6.2.1 패키지 로드

패키지를 불러들이는 부분에 있는 sklearn은 scikit-learn의 약자로 파이썬 프로그래밍 언어용 기계학습 관련 패키지입니다. Sklearn.metrics는 scikit-learn 패키지 중 모형평가에 사용되는 모듈입니다. Sklearn.metrics 패키지의 모듈 중 Classification_report는 주요 분류 측정 항목을 보여주는 보고서 모듈입니다.

Confusion_matrix는 분류의 정확성을 평가하기 위한 오차행렬 계산 모듈입니다. Sklearn.model_selection은 scikit-learn 패키지 중 클래스를 나눌 때, 그리고 함수를 통해 train/test셋을 나눌 때, 아울러 모델 검증에 사용되는 서브 패키지입니다. Sklearn.model_selection 패키지의 모듈 중 Train_test_split은 배열 또는 행렬을 임의의 훈련(train) 및 테스트(test) 셋으로 분할하는 모듈입니다. Sklearn.tree는 scikit-learn 패키지 중 분류 및 회귀를 위한 의사결정 트리 기반 모델이 있는 서브 패키지입니다.

Sklearn.tree 패키지의 모듈 중 DecisionTreeClassifier는 의사결정 트리 분류 모듈입니다. Ipython.display는 Ipython 내에 정보를 보여주는 도구용도의 공용 API입니다. Ipython.display API의 모듈 중 Image는 원시(raw)데이터가 있는 png나 Jpeg등의 이미지 객체를 만드는 모듈입니다.

Pandas는 데이터를 구조화된 형식으로 가공 및 분석할 수 있도록 자료구조를 제공하는 패키지입니다. 여기서 as pd는 pandas를 약칭 pd로 사용한다는 의미입니다.Numpy는 Numerical Python으로 고성능 계산이나 데이터 분석에 유용한 패키지입니다. as np는 numpy를 약칭 np로 사용한다는 의미입니다.

Pydotplus는 그래프를 생성하는 graphviz의 Dot 언어를 파이썬 인터페이스 제공하는 모듈입니다. os는 운영체제(Operating System)와 상호작용하기 위한 기본적인 기능(경로 생성, 변경 등)이 제공되는 모듈입니다. 그림 7은 decision tree 실습에 사용되는 패키지들을 로드하는 내용입니다.

```
1  from sklearn.metrics import classification_report, confusion_matrix
2  from sklearn.model_selection import train_test_split
3  from sklearn.tree import DecisionTreeClassifier
4  from sklearn import tree
5
6  from IPython.display import Image
7
8  import pandas as pd
9  import numpy as np
10 import pydotplus
11 import os
```

그림 7 패키지 로드

1.6.2.2 데이터 로드

1번째 줄의 코드는 예제 데이터인 playtennis의 csv 파일을 pandas의 read_csv 함수를 이용해 로드하고, pandas의 자료구조 중 하나인 dataframe 형식으로 변수 tennis_data에 저장합니다. 2번째 줄의 코드는 변수 tennis_data를 확인하는 코드입니다. 그림 8은 예제 데이터를 dataframe 형식으로 로드하는 내용입니다. 그림 9는 dataframe 형식으로 예제 데이터를 보여줍니다.

```
1  tennis_data = pd.read_csv('playtennis.csv')
2  tennis_data
```

그림 8 dataframe 형식으로 예제 데이터 로드

	Outlook	Temperature	Humidity	Wind	PlayTennis
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

그림 9 dataframe 결과

1.6.2.3 데이터 전처리

1번부터 16번째 줄까지의 코드는 변수 tennis_data의 각 컬럼(Outlook, Temperature, Humidity, Wind, PlayTennis)의 값(Sunny, Overcast, … 등)을 문자열(String) 타입에서 숫자(int) 타입으로 대치(replace)해 변수 tennis_data에 저장합니다. 18번째 줄은 전처리 과정(문자열 타입에서 숫자 타입으로 대치)이 된 변수 tennis_data를 확인하는 코드입니다. 그림 10은 예제 데이터를 전처리 하는 과정이고 그림 11은 전처리 된 예제 데이터를 보여줍니다.

```
1  tennis_data.Outlook = tennis_data.Outlook.replace('Sunny', 0)
2  tennis_data.Outlook = tennis_data.Outlook.replace('Overcast', 1)
3  tennis_data.Outlook = tennis_data.Outlook.replace('Rain', 2)
4
5  tennis_data.Temperature = tennis_data.Temperature.replace('Hot', 3)
6  tennis_data.Temperature = tennis_data.Temperature.replace('Mild', 4)
7  tennis_data.Temperature = tennis_data.Temperature.replace('Cool', 5)
8
9  tennis_data.Humidity = tennis_data.Humidity.replace('High', 6)
10 tennis_data.Humidity = tennis_data.Humidity.replace('Normal', 7)
11
12 tennis_data.Wind = tennis_data.Wind.replace('Weak', 8)
13 tennis_data.Wind = tennis_data.Wind.replace('Strong', 9)
14
15 tennis_data.PlayTennis = tennis_data.PlayTennis.replace('No', 10)
16 tennis_data.PlayTennis = tennis_data.PlayTennis.replace('Yes', 11)
17
18 tennis_data
```

그림 10 예제 데이터 전처리

	Outlook	Temperature	Humidity	Wind	PlayTennis
0	0	3	6	8	10
1	0	3	6	9	10
2	1	3	6	8	11
3	2	4	6	8	11
4	2	5	7	8	11
5	2	5	7	9	10
6	1	5	7	9	11
7	0	4	6	8	10
8	0	5	7	8	11
9	2	4	7	8	11
10	0	4	7	9	11
11	1	4	6	9	11
12	1	3	7	8	11
13	2	4	6	9	10

그림 11 전처리 된 예제 데이터

1.6.2.4 속성과 클래스 분리

1번째 줄 코드는 변수 tennis_data의 컬럼(Outlook, Temperature, Humidity, Wind) 값들을 데이터프레임 형태로 추출하고 np.array 함수를 이용해 추출한 데이터를 배열형태로 변환한 후 변수 X에 저장합니다. 2번째 줄 코드는 변수 tennis_data의 컬럼(PlayTennis) 값을 데이터프레임 형태로 추출하고 Np.array 함수를 이용해 추출한 데이터를 배열형태로 변환한 후 변수 y에 저장합니다. 그림 12는 속성 컬럼인 Outlook, Temperature, Humidity, Wind와 클래스 PlayTennis를 분리해 각각 다른 변수로 저장하는 내용입니다. 그림 13은 데이터의 속성과 클래스를 분리한 결과입니다. 왼쪽 결과가 속성, 오른쪽 결과가 클래스입니다.

```
1 X = np.array(pd.DataFrame(tennis_data, columns = ['Outlook', 'Temperature', 'Humidity', 'Wind']))
2 y = np.array(pd.DataFrame(tennis_data, columns = ['PlayTennis']))
```

그림 12 데이터의 속성과 클래스 분리

```
array([[0, 3, 6, 8],                    array([[10],
       [0, 3, 6, 9],                           [10],
       [1, 3, 6, 8],                           [11],
       [2, 4, 6, 8],                           [11],
       [2, 5, 7, 8],                           [11],
       [2, 5, 7, 9],                           [10],
       [1, 5, 7, 9],                           [11],
       [0, 4, 6, 8],                           [10],
       [0, 5, 7, 8],                           [11],
       [2, 4, 7, 8],                           [11],
       [0, 4, 7, 9],                           [11],
       [1, 4, 6, 9],                           [11],
       [1, 3, 7, 8],                           [11],
       [2, 4, 6, 9]], dtype=int64)             [10]], dtype=int64)
```

그림 13 속성과 클래스 분리한 결과

1번째 줄 코드는 로드(load)된 train_test_split 모듈을 이용해 변수 X에 입력 4개 컬럼의 데이터와 변수 y에 입력한 playtennis 컬럼의 데이터를, train(훈련)과 test(테스트)로 구분해, 임의의 개수로 각각 변수 X_train, X_test, y_train, y_test에 저장합니다. 그림 14는 데이터를 train set과 test set 분리한 내용입니다.

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

그림 14 train set과 test set 분리

1.6.2.5 데이터 학습

1번째 줄 코드는 로드된 의사결정 트리 분류 모듈을 변수 dt_clf에 저장합니다. 2번째 줄 코드는 의사결정 트리 분류 모듈이 저장된 변수 dt_clf의 함수 fit()에 변수 X_train, y_train을 입력해 의사결정 트리 분류 모델을 생성합니다. 그런 다음 생성한 모델을 다시 변수 dt_clf에 저장합니다. 그림 15는 의사결정 트리 모델 생성에 대한 내용입니다.

```
1 dt_clf = DecisionTreeClassifier()
2 dt_clf = dt_clf.fit(X_train, y_train)
```

그림 15 의사결정 트리 모델 생성

1번째 줄 코드는 변수 dt_clf의 함수 predict()에 변수 X_test를 입력합니다. 그리고 입력한 X_test에 대한 클래스 예측 값을 변수 dt_prediction에 저장합니다. 그림 16은 의사결정 트리 모델에 test 값을 넣어 예측 값을 저장하는 내용입니다.

```
1 dt_prediction = dt_clf.predict(X_test)
```

그림 16 test 값에 대한 예측 값 저장

1.6.2.6 성능 평가

1번째 줄 코드는 오차행렬을 계산하는 모듈 confusion_matrix()에 변수 y_test와 dt_prediction을 입력합니다. 입력한 두 변수(y_test, dt_prediction)의 오차행렬을 print 문으로 출력합니다. 그림 17은 예측 값에 대한 confusion matrix 결과입니다.

```
1 print(confusion_matrix(y_test, dt_prediction))
[[1 1]
 [0 2]]
```

그림 17 confusion matrix 결과

1번째 줄 코드는 분류 측정 항목을 보여주는 모듈인 classification_report()에 변수 y_test와 dt_prediction을 입력합니다. 입력한 두 변수(y_test, dt_prediction)에 대한 분류 측정 항목을 print 문으로 출력합니다. 그림 18은 precision, recall, f-measure 값을 나타내는 classification report에 대한 내용입니다.

```
1 print(classification_report(y_test, dt_prediction))
             precision    recall  f1-score   support

         10       1.00      0.50      0.67         2
         11       0.67      1.00      0.80         2

avg / total       0.83      0.75      0.73         4
```

그림 18 classification report 결과

1.6.2.7 의사결정 트리 그래프 표현

자, 이제, Ipython 내에서 그래프를 표현하는 소프트웨어인 graphviz 설치과정에 대해 알아보겠습니다. Window 환경의 최신버전인 graphviz-2.38을 설치하는데요.

다운로드 파일이 있는 사이트 http://www.graphviz.org/Download_window.php에 가면 다운로드가 가능한 2개의 파일이 있습니다. 2개의 파일은 확장자명이 msi와 zip입니다. 2개의 파일 중 확장자가 msi인 graphviz-2.38.msi 파일을 다운로드 합니다. 그림 19는 graphviz-2.38.msi 파일 다운로드에 대한 내용입니다.

그림 19 graphviz-2.38.msi 파일 다운로드

그런 다음 다운로드 한 graphviz-2.38.msi 파일을 실행합니다. 실행과정에서 옵션변경 없이 next 버튼을 선택해 설치를 진행하시면 됩니다. 설치가 완료된 graphviz-2.38 소프트웨어의 기본 경로는 C: \Program Files (x86)가 됩니다.

1번째 줄 코드는 ipython 내에서 그래프를 생성할 수 있는 인터페이스 경로를 추가 설정하는 부분입니다. Graphviz2.38 소프트웨어의 bin 폴더가 있는 경로인 C: \Program Files (x86)/Graphviz2.38/bin/를 os 모듈 중 경로 구분 기호를 반환하는 함수인 os.pathsep을 이용해, 환경변수들을 나타내는 사전함수인 s.environ[‘PATH’]에 동적으로 할당해 저장합니다. 그림 20은 graphviz 동적 할당에 대한 내용입니다.

```
1 os.environ["PATH"] += os.pathsep + 'C:/Program Files (x86)/Graphviz2.38/bin/'
```

그림 20 graphviz 동적 할당

1번째 줄 코드는 트리표현 함수에 입력되는 파라미터 중 하나인 feature_names에 값을 입력하기 위해, 변수 tennis_data의 각 컬럼명을 list형태로 변환한 후 변수 feature_names에 저장합니다. 2번째 줄 코드는 저장된 변수 feature_names를 슬라이싱([0:4])해 Outlook, Tempertature, Humidity, Wind의 컬럼명을 추출한 다음 다시 변수 feature_names에 저장합니다. 그림 21은 컬럼명 추출에 대한 내용입니다.

```
1 feature_names = tennis_data.columns.tolist()
2 feature_names = feature_names[0:4]
```

그림 21 컬럼명 추출

1번째 줄 코드는 트리표현 함수에 입력되는 파라미터 중 하나인 class_names에 값을 입력하기 위해, Target_class 값인 ‘Play No’와 ‘Play Yes’를 배열형태로 변수 target_name에 저장합니다. 그림 22는 target class 값 저장에 대한 내용입니다.

```
target_name = np.array(['Play No', 'Play Yes'])
```

그림 22 target class 값 저장

1번째 줄 코드는 Tree 패키지 중 의사결정 트리를 dot 형식으로 내보내는 함수인 export_graphviz()를 이용해 트리 표현을 변수 dt_dot_data에 저장합니다. dt_clf는 의사결정 트리 분류기, out_file은 의사결정 트리를 파일 또는 문자열로 반환(기본 : tree.dot, None일 경우 문자열로 반환)입니다.

feature_names는 각 features의 이름, class_names는 각 대상 class의 이름을 오름차순으로 정렬, filled는 True일 경우 분류를 위한 다수 클래스, 회귀 값의 극한 또는 다중 출력의 노드 순도를 나타내기 위해 노드를 색칠합니다.

rounded는 True일 경우 둥근 모서리가 있는 노드 상자를 그리고 Times-Roman 대신 Helvetica 글꼴을 사용합니다. 마지막으로 special_characters는 True일 경우 특수 문자 표시를 합니다. 그림 23은 의사결정 트리 표현을 저장하는 내용입니다.

```
dt_dot_data = tree.export_graphviz(dt_clf, out_file = None,
                                   feature_names = feature_names,
                                   class_names = target_name,
                                   filled = True, rounded = True,
                                   special_characters = True)
```

그림 23 의사결정 트리 표현 저장

1번째 줄 코드는 Pydotplus 모듈 중 Dot 형식의 데이터로 정의된 그래프를 로드(load)하는 함수인 Graph_from_dot_data()에 변수 dt_dot_data를 입력한 후 다시 변수 dt_graph에 저장합니다. 그림 24는 그래프를 로드하는 내용입니다.

```
dt_graph = pydotplus.graph_from_dot_data(dt_dot_data)
```

그림 24 그래프 로드

다음의 1번째 줄 코드는 변수 dt_graph에 대한 정보를 png 파일로 생성하는 함수 create_png()를 사용한 후, 이미지 객체를 만드는 Image 모듈을 통해 ipython 내에서 그래프를 표현합니다. 그림 25는 로드한 그래프를 png 파일로 표현에 대한 내용이고 그림 26은 그래프 결과입니다.

```
1 Image(dt_graph.create_png())
```

그림 25 그래프를 png 파일로 표현

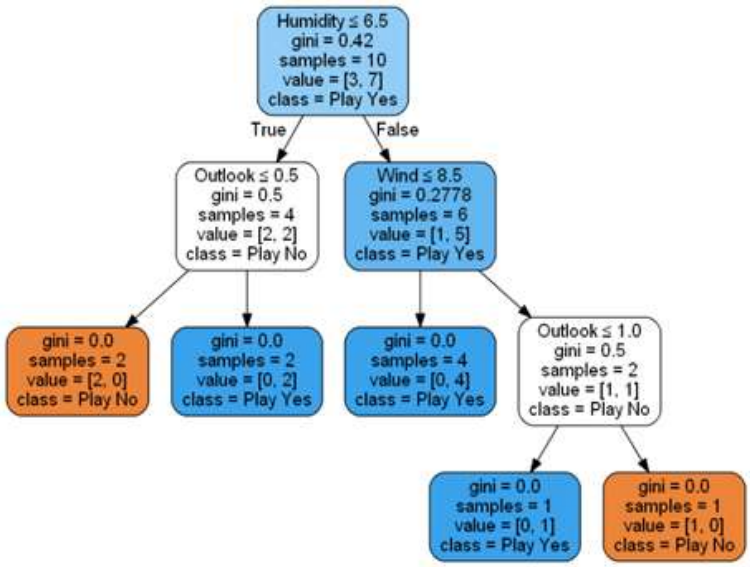


그림 26 그래프 결과

지금까지 의사결정 트리의 개념과 예제 데이터를 이용한 의사결정 트리 구현 방법에 대해 학습했습니다.

2. Random Forest를 이용한 데이터 분석

2.1 Random Forest 소개

- Random forest는 2001년에 Leo Breiman에 의해 처음으로 소개
- Decision Tree의 단점을 개선하기 위한 알고리즘 중 하나
- Decision Tree의 확장
- Random Forest는 훌륭한 데이터 분석 알고리즘 중 하나
  - 데이터 분류(Classification)
  - 데이터 군집(Clustering)
  - Feature의 중요성 확인
  - 데이터 예측

random forest는 의사결정트리의 확장된 알고리즘이라고 생각하시면 편합니다. random forest는 말 그대로 여러 개의 나무가 모여 하나의 숲을 형성한다는 의미인데요. 이러한 random forest는 데이터 분류, 데이터 예측, 데이터에서 특징의 중요성 확인 및 비지도 학습 등 분야에서 많이 사용됩니다.

2.2 Random Forest 이론

본 장에서는 random forest 분류 및 예측 모듈에 대해 설명하고자 합니다. random forest를 이용한 예측은 데이터의 분류와 같은 말입니다. random forest 예측 모듈을 생성하는 과정은 왼쪽에 있는 그림과 같습니다. 그림 27은 random forest 예측 모듈 생성과정에 대한 내용입니다.

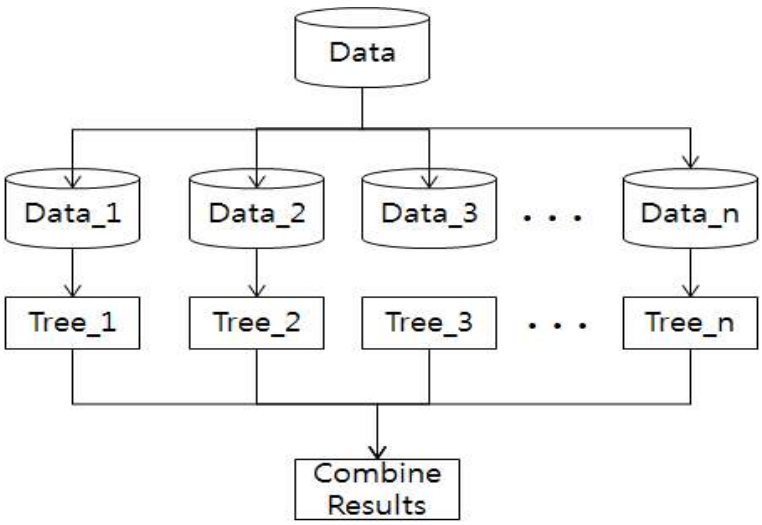


그림 27 random forest 예측 모듈 생성과정

2.2.1 Random Forest 예측 모듈

1) Dataset에서 샘플 데이터를 선택 (bagging, bootstrap aggregation)
2) 샘플 데이터를 이용해 decision tree를 생성
3) 1), 2)를 n번 반복
4) 3)을 통해 생성한 n개의 decision tree를 이용해 예측
5) 예측 결과에서 가장 많이 등장하는 결과를 선택하여 최종 결과로 선택

과정 1은 우선 데이터 셋에서 x개의 데이터를 추출해 새로운 샘플 데이터 셋을 생성합니다. 그런 다음 과정 2에서 새로 생성된 샘플 데이터 셋을 이용해 새로운 의사결정트리를 생성합니다. 위와 같은 과정 1, 2를 n번 반복합니다. 여기서 n은 새로 생성한 샘플 데이터 셋의 개수 및 의사결정트리의 총 개수 입니다. 이렇게 생성된 나무를 하나로 묶으면 random forest의 데이터 예측 모듈이 생성됩니다. 테스트 데이터가 입력되면 각 의사결정트리에 입력해 결과를 구하고 가장 많이 나온 결과를 선택해 최종 결과로 합니다.

2.2.2 Random Forest 예측 예

1) Test Data의 target은 A or B
2) Test Data의 target을 예측
3) 4개의 tree에서 예측 결과가 ‘A’
4) 최종 결과가 Test Data를 A로 예측

목표 값이 A 또는 B인 테스트 데이터를 입력하였을 때 결과를 예측하는 예(example)입니다. 그림에 있는 random forest 예측 모듈은 5개의 나무로 이루어졌습니다. 트리_1부터 트리_5는 각자 동일한 테스트 데이터를 입력 받고 결과를 예측합니다. 예제에서 세 번째 트리를 빼고 예측한 결과는 모두 A입니다. 4개의 트리에서 입력 데이터를 A라고 판단하고 하나만 B라고 판단하였습니다. 예측 결과를 통합한 결과 A=4는 B=1보다 크므로 random forest 예측 모듈은 테스트 데이터를 A로 예측합니다. 그림 28은 random forest 예측의 예입니다.

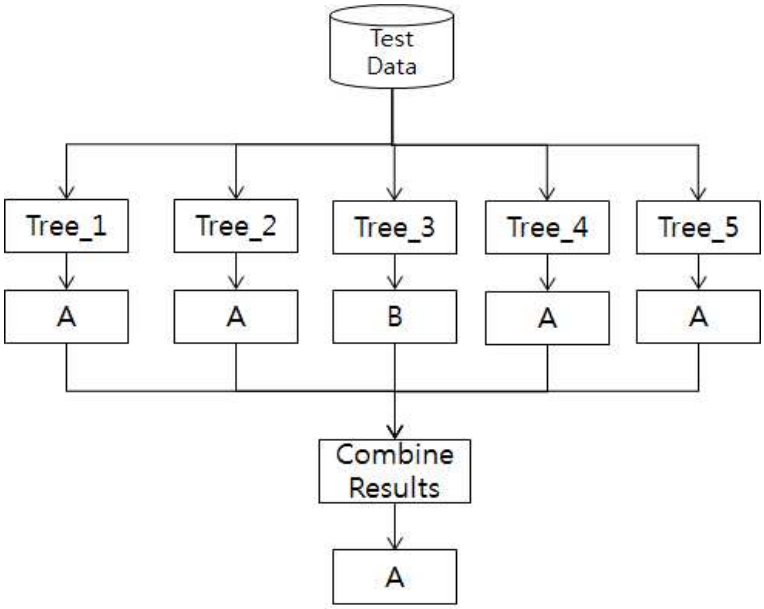


그림 28 random forest 예측 예

2.2.3 Random Forest 특징

1) 여러 개의 decision tree를 결합함으로써 단일 decision tree의 결점을 극복
2) over-fitting 문제가 적음
3) 구현이 간단함
4) 병렬계산이 간단함

위에서 설명한 간단한 예제에서 알 수 있는 바 random forest는 여러 개 의사결정트리들의 결합입니다. 이런 방법을 통해 단일 의사결정트리를 사용하는 데서 나타나는 결점들을 극복합니다. 단일 의사결정트리는 과최적화 문제가 있지만 random forest는 무작위로 많은 의사결정트리를 생성하고 결과를 통합함으로써 과최적화 문제를 해결하였습니다. 그리고 random forest는 구현이 간단하고 병렬 계산으로 전환하기 간편한 장점이 있습니다. Random forest에는 두 rk지의 무작위 처리 방식이 있습니다.

2.2.4 Random Forest에서의 2가지 random

1) Data set에서 샘플 데이터를 random으로 선택
2) 샘플 데이터에서 feature를 random으로 선택해 decision tree를 생성

첫 번째 방법은 샘플 데이터를 무작위로 선택하는 것이고 두 번째 방법은 샘플 데이터로부터 특징 값을 무작위로 선택해 의사결정트리를 생성하는 방

법입니다. 이러한 두 가지 무작위 방식으로 인해 random forest는 의사결정트리의 단점을 극복하는 우수한 알고리즘이 되는 것입니다.

2.2.5 샘플 데이터 선택 예

- random으로 n개의 데이터 선택
- 선택한 n개의 데이터는 중복이 가능
- 선택한 t개의 샘플 데이터 사이 데이터 중복 가능
- 선택한 샘플 데이터로 decision tree 생성

데이터 셋에서 샘플 데이터를 선택하는 과정에 대해 알아보겠습니다. 그림에 있는 훈련 데이터 셋은 전체 데이터 셋에서 학습 데이터와 훈련 데이터를 분리한 후의 훈련 데이터만을 의미합니다. 샘플 데이터를 선택할 때 훈련 데이터 셋에서 무작위로 n개 데이터를 선택하는데 이 n개 데이터의 선택은 박스에서 꺼낸 데이터를 다시 박스 안에 넣는 방법으로 진행합니다.

그러므로 선택한 n개의 데이터는 중복 선택이 가능합니다. 그리고 선택한 t개의 샘플 데이터로 각자 의사결정트리를 생성합니다. 즉 총 t개의 의사결정트리가 생성되는 것입니다. 그림 29는 샘플 데이터 선택에 대한 내용입니다.

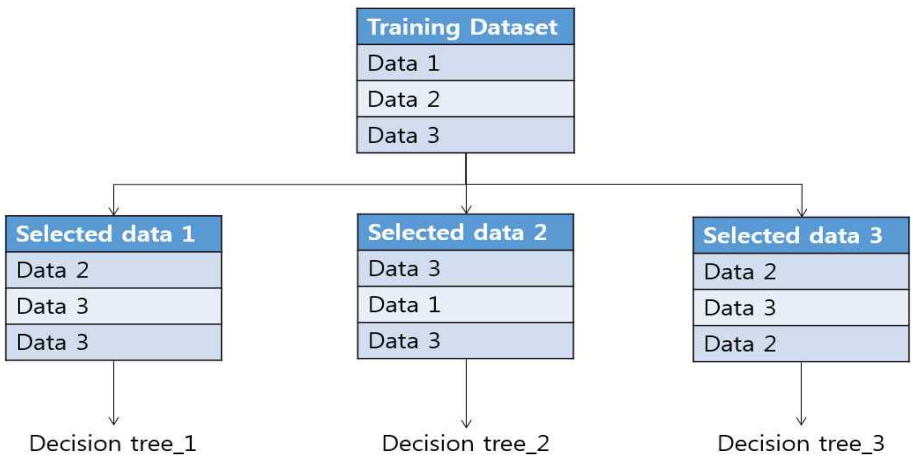


그림 29 샘플 데이터 선택

2.2.6 선택한 샘플 데이터에서 random으로 f개의 feature를 선택

- 선택한 feature의 개수는 sqrt(전체 feature 수), log2(전체 feature 수)등 방법으로 계산

선택한 샘플 데이터로 의사결정트리를 생성하려면 특징 값을 선택하고 특징의 순서를 정해야 합니다. 특징의 선택 및 트리에서 특징의 순서 역시 무작위로 진행됩니다. 이렇게 하는 이유는 우수한 특징 값을 찾고 색다른 트리를 생성해 random forest의 다양성을 확보하기 위해서입니다. 선택하는 특징의 개수는 흔히 제곱근(sqrt), 로그(log2) 방정식 등 방법으로 계산됩니다.

이번에는 random forest의 성능평가 방법에 대해 알아보겠습니다. random forest의 성능평가 방법은 여러 가지가 있는데 여기서는 bagging과 out-of-bag (oob)에 대해 간단히 알아보도록 하겠습니다.

2.2.7 자기 성능 평가

- Bagging
  - 63%의 데이터를 이용해 매개 tree를 생성
  - 나머지 37%의 데이터를 이용해 매개 tree의 성능을 평가
    - 매개 tree에 입력하는 데이터는 다름

- Out-of-Bag (OOB)
  - OOB 데이터를 이용해 tree의 성능을 교정
  - OOB는 성능 통계에서 많이 사용됨

Bagging은 63%의 (샘플)데이터를 이용해 각각의 트리들을 생성하고 나머지 37%의 데이터를 이용해 성능을 평가하는 방법입니다. 여기서 각각의 트리들에 입력하는 데이터는 다를 수도 있습니다. out-of-bag(OOB)은 성능 통계에서 많이 사용되는 방법입니다. OOB 데이터를 이용해 트리의 성능 교정을 진행합니다.

이제부터 python에서 random forest를 구현하는 방법에 대해 알아보겠습니다. 이번에 사용하는 데이터는 기계학습에서 예제로 많이 사용되는 iris 데이터입니다. 우선 iris 데이터에 대해 간단히 알아보겠습니다.

2.3 Iris 데이터를 이용해 간단한 Random Forest 구현

2.3.1 Iris Data 소개

Iris 데이터는 붓꽃의 3가지 종류를 기록한 데이터입니다. Iris 데이터에는 붓꽃 줄기의 길이, 너비 그리고 붓꽃 잎의 길이와 너비 등 4개의 특징이 있

습니다. 그리고 목표 데이터, 즉 붓꽃의 종류인 target은 0, 1, 2로 되어 있는데 이는 각각 setosa, versicolor, virginica를 나타냅니다. 전체 데이터의 크기는 150개(line)입니다. 표 4는 iris 예제 데이터에 대한 내용입니다.

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
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
3	4.6	3.1	1.5	0.2	0
…	…	…	…	…	…
148	6.2	3.4	5.4	2.3	2
149	5.9	3.0	5.1	1.8	2

표 4 iris 예제 데이터

2.3.2 필요한 패키지 및 라이브러리 로드

먼저 이번 강의에서 사용되는 패키지들에 대해 알아보겠습니다. 첫 번째 line에서 Iris 데이터를 사용하기 위해 sklearn.DataSets 패키지에서 load_iris모듈을 import합니다. sklearn.datasets 서브 패키지는 scikit-learn 패키지 중 일반적인 참조 데이터 셋을 load하는 모듈로 많은 샘플 데이터 셋을 무료로 간편하게 다운로드할 수 있게 도와줍니다.

두 번째 line에서 random forest 예측 모듈의 성능을 평가하기 위해 sklearn.metrics 패키지에서 accuracy_score 모듈을 import 합니다. sklearn.metrics는 성능 평가를 제공하는 패키지이고 accuracy_score는 예측 결과의 accuracy를 계산하는 모듈입니다.

세 번째 line에서 numpy 패키지를 import하고 np라고 따로 명명하였습니다. numpy는 파이썬 언어를 위한 행렬, 벡터 등의 수학 계산을 위한 자료구조와 계산 함수를 제공하는 패키지입니다. 네 번째 line에서 Pandas 패키지를 import 하고 pd라고 따로 명명하였습니다. Pandas는 데이터 분석, 가공, 처리 등을 쉽게 하기 위한 자료구조와 처리 함수들을 제공하는 패키지입니다. 그림 30은 필요한 패키지 및 라이브러리 로드에 대한 내용입니다.

```
1 from sklearn.datasets import load_iris
2 from sklearn.metrics import accuracy_score
3 import numpy as np
4 import pandas as pd
```

그림 30 필요한 패키지 및 라이브러리 로드

2.3.3 Training data와 Test data 설정

iris 데이터를 로드하고 훈련 데이터와 테스트 데이터를 분리하는 과정입니다. 두 번째 Line에서 load_iris() 함수를 사용해 iris 데이터를 로드하고 iris라는 변수에 저장합니다. 변수 iris에는 feature와 target이 같이 있는데 예측 모듈을 학습시키려면 이를 구분해야 합니다. iris.data 함수를 사용하면 iris에서 feature 데이터만 호출하게 됩니다. 그리고 iris.target은 iris에서 target 데이터만 호출합니다. 이 두 함수를 사용하면 iris에서 feature와 target을 쉽게 분리할 수 있습니다.

그리고 훈련 데이터와 테스트 데이터를 분리할 때 우리는 전체 데이터에서 80%를 훈련에 사용하고 나머지 20% 데이터로 성능을 테스트 하려고 합니다. 즉 150개 iris 데이터에서 120개 데이터를 훈련 set으로 하고 나머지 30개 데이터를 테스트 set으로 하는 것입니다. 다섯 번째 Line에서 iris.data 함수를 호출해 iris feature 데이터에서 앞쪽 120개 특징 데이터를 x_train이라는 변수에 저장합니다. 여기서 iris.data 함수 뒤 중괄호 안에 ':' -30은 결과 값에서 처음부터 마지막 30번째 값까지만 선택한다는 뜻입니다.

여기서 -30은 데이터를 뒤로부터 count할 때 30번째 위치를 의미합니다. 전체 iris 데이터는 150개이고 여기서 마지막 30개를 제외하면 120개가 됩니다.

여섯 번째 Line에서 iris.target으로 iris target 데이터에서 위와 같이 처음부터 마지막 30번째 데이터까지 선택해 변수 y_train에 저장합니다. X_train, y_train은 훈련 데이터입니다. 여덟 번째 Line에서 iris feature 데이터 중 뒤로부터 30개 데이터를 x_test라는 변수에 저장합니다. 아홉 번째 Line 에서 iris target 데이터에서 마지막 30개 데이터를 y_test는 변수에 저장합니다. 그림 31은 train과 test data 설정에 대한 내용입니다.

```
#loading the iris dataset
iris = load_iris()

#training data 설정
x_train = iris.data[:-30]
y_train = iris.target[:-30]
#test data 설정
x_test = iris.data[-30:] # test feature data
y_test = iris.target[-30:] # test target data
```

그림 31 train, test data 설정

2.3.4 Training data의 target 출력

훈련 데이터와 테스트 데이터의 분리를 완료한 다음 훈련 데이터의 목표 값과 테스트 데이터의 목표 값을 각각 출력해 보았습니다. 그림 32는 train data의 target을 출력하고, 그림 33은 test data의 target을 출력하는 내용입니다.

```
print (y_train)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2]
```

그림 32 train data의 target

2.3.5 Test data의 target 출력

```
print (y_test)
```

```
[2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]
```

그림 33 test data의 target

위에 출력한 훈련 데이터에는 “0, 1, 2” 세 가지 목표 값이 있습니다. 그런데 목표 값 2는 상대적으로 양이 적은 편입니다. 그리고 테스트 데이터의 목표 값을 보면 전부 2입니다. 이는 훈련 데이터와 테스트 데이터의 분리가 합리적이지 않다는 것을 의미합니다. 하지만 우리는 먼저 이 데이터로 간단한 예측 모듈을 생성해 보겠습니다.

2.3.6 Random Forest 분류기 생성

Random forest 예측 모듈을 생성하기 위해 RandomForestClassifier 모듈을 import해야 합니다. RandomForestClassifier는 Sklearn.ensemble 모듈에 속하는데

Sklearn.ensemble 모듈에는 분류, 회귀 및 이상 탐지를 위한 앙상블 기반의 방법을 포함하고 있습니다. RandomForestClassifier는 Random forest 분류기를 포함한 모듈이라고 위에서 설명하였습니다. 그림 34는 random forest classifier 로드한 내용입니다.

```
#RandomForestClassifier 클래스를 import
from sklearn.ensemble import RandomForestClassifier
```

그림 34 random forest classifier 로드

RandomForestClassifier 함수에 n_estimators=10을 주어 나무의 개수가 10인 random forest 분류 모델을 생성하고 변수 rfc에 저장합니다. 생성한 random forest 분류/예측 모듈을 출력해보면 RandomForestClassifier에는 많은 parameter가 있고 default 값도 정해져 있습니다. 여기서 많이 사용되는 몇 개의 파라미터만 설명하고 상세한 내용은 아래 참조 url에서 확인할 수 있습니다. 파라미터 n_estimators는 의사결정트리의 개수를 나타내며 기본 값은 10입니다. 그림 35는 10개의 트리를 가진 random forest를 생성하는 내용입니다.

```
#RandomForestClassifier libary를 import
from sklearn.ensemble import RandomForestClassifier
#tree 의 개수 Random Forest 분류 모듈 생성
rfc = RandomForestClassifier(n_estimators=10)
rfc
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=10, n_jobs=1, oob_score=False, random_state=None,
            verbose=0, warm_start=False)
```

그림 35 10개 tree를 가진 random forest 생성

max_features는 random forest에서 단일 트리를 생성할 때 고려하는 최대 특징의 개수를 의미합니다. 기본 값은 auto로 되어 있습니다. oob_score는 out-of-bag(OOB) 사용 여부를 의미하는데 기본 값은 false입니다.

2.3.7 Random Forest 분류 학습

첫 번째 Line은 random forest 분류 모듈에 있는 내부 함수 rfc.fit()에 훈련 데이터를 입력해 random forest 모듈을 학습시킵니다. 여기서 x_train은 훈련 데이터의 특징 값이고 y_train은 훈련 데이터 목표 값입니다. rfc.fit()는 입력 데이터를 이용해 분류기를 학습시키는 함수입니다. 세 번째 Line은 학습시킨

random forest 분류 모델에 테스트 데이터를 입력해 목표 값을 예측하고 예측 결과를 변수 prediction에 저장합니다.

rfc.predict() 함수는 입력 데이터의 분류 결과를 예측하는 함수입니다. 다섯 번째 Line은 분류 결과를 예측한 목표 값 prediction과 실제 목표 값 y_테스트를 비교해 값이 같으면 true 다르면 false로 출력한 결과입니다. 결과를 보면 30개의 테스트 데이터에서 7개를 잘못 예측하였습니다. 그림 36은 예측한 target 값과 실제 target 값을 비교한 내용입니다.

```
1 rfc.fit(x_train, y_train)
2 #Test data를 입력해 target data를 예측
3 prediction = rfc.predict(x_test)
4 #예측 결과 precision과 실제 test data의 target 을 비교
5 print (prediction==y_test)
```

```
[ True  True  True False  True  True False False  True False  True  True
  True False False  True  True  True False  True  True  True  True  True
  True  True  True  True  True  True]
```

그림 36 예측 target 값과 실제 target 값 비교

2.3.8 Random Forest 분류 예측 결과

rfc.score()는 RandomForestClassifier 클래스 안에 있는 분류 결과의 정확도(Accuracy)를 계산하는 함수입니다. 입력 데이터는 test data의 feature와 tareget 값입니다. 테스트 데이터에 대한 예측 결과의 정확도는 0.7667입니다. 그림 37은 예측 결과에 대한 내용입니다.

```
1 #Random forest 정확도 측정
2 rfc.score(x_test, y_test)
```

```
0.76666666666666672
```

그림 37 예측 결과

2.3.9 Random Forest 분류 성능 평가

분류 예측 모듈의 평가에는 accuracy뿐만 아니라 precision, recall도 많이 사용됩니다. sklearn.metrics 모듈은 scikit-learn 패키지 중 모델의 성능 평가 방법을 모아 놓은 모듈입니다. 이 모듈에서 accuracy_score와 classification_report를 import 해 모델의 성능을 다시 평가 해보려고 합니다. 함수 accuracy_score()는 분류 결과의 accuracy를 계산하고, 함수 classification_report()는 분류 결과의

precision, recall을 계산합니다. 다섯 번째 Line은 accuracy를 계산하는 코드이고 일곱 번째 Line은 precision과 recall을 계산하는 코드입니다.

여기서 입력 데이터는 예측 결과 값인 prediction과 실제 테스트 데이터의 목표 값인 y_test입니다. 결과를 살펴보면 precision과 recall은 낮은 것을 알 수 있습니다. 그림 38은 성능평가에 대한 내용입니다.

```
1 from sklearn.metrics import accuracy_score
2 from sklearn.metrics import classification_report
3
4
5 print ("Accuracy is : ",accuracy_score(prediction, y_test))
6 print ("=============================================================")
7 print (classification_report(prediction, y_test))
```

```
Accuracy is :  0.766666666667
=============================================================
             precision    recall  f1-score   support

          1       0.00      0.00      0.00         7
          2       0.77      1.00      0.87        23

avg / total       0.59      0.77      0.67        30
```

그림 38 성능 평가

이것은 우리가 위에서 훈련 데이터와 테스트 데이터를 잘 분리하지 못한 이유로 분류 성능이 낮게 나온 것입니다. 데이터 셋에서 훈련 데이터 셋과 테스트 데이터 셋을 분리하는 방법에는 여러 가지가 있습니다.

2.3.10 Training, Test data 재생성

함수 Train_test_split은 데이터를 무작위로 혼합한 후 training data set과 test data set을 일정한 비율에 따라 분리해줍니다. 네 번째 Line은 데이터를 무작위로 혼합한 후 x의 80%를 X_train, x의 20%를 X_test, y의 80%를 Y_train, y의 20%를 Y_test에 저장합니다.

여기서 x는 iris feature data, y는 iris target data입니다. 네 번째 Line에서 마지막 파라미터 test_size = 0.2는 훈련 셋을 전체 데이터의 80%, 테스트 셋을 전체 데이터의 20%로 선택하겠다는 뜻입니다.

다섯 번째 Line은 위에서 마지막 30개 데이터를 선택한 테스트 데이터의 목표 값을 출력하고 여섯 번째 Line은 데이터를 무작위로 혼합한 후 선택한 150개 데이터의 20%인 30개 테스트 데이터의 목표 값을 출력합니다.

출력 결과를 보시면 데이터를 혼합한 결과 Y_test에는 목표 데이터의 종류가 이전과는 다르게 동일하지 않다는 것을 알 수 있습니다. 그림 39는 train_test_split 함수를 사용해 training data와 test data를 생성하는 내용입니다.

```
1 from sklearn.model_selection import train_test_split
2 x = iris.data
3 y = iris.target
4 X_train, X_test, Y_train, Y_test = train_test_split(x, y, test_size=0.2)
5 print (y_test)
6 print (Y_test)
```

```
[2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]
[1 2 1 2 2 2 2 1 0 1 1 1 2 2 0 1 2 1 0 1 2 1 2 1 2 2 2 1 0 1]
```

그림 39 train_test_split 함수를 사용한 training, test data set

2.3.11 Random Forest 분류 성능 평가

새로 생성한 훈련 셋과 테스트 셋을 이용해 새 random forest 분류 예측 모델을 생성하고 성능을 다시 측정해 보겠습니다. 첫 번째 Line에서 tree의 개수가 10인 random forest를 생성하고 clf에 저장합니다. 두 번째 Line에서 데이터를 혼합하고 선택한 훈련 데이터 X_train과 Y_train을 입력해 모듈을 학습 시킵니다. 세 번째 Line은 predict 함수를 이용해 X_test 의 목표를 분류 예측하고 결과를 prediction_1에 저장합니다.

그런 다음 accuracy_score 함수와 classification_report 함수로 accuracy, precision, recall을 계산하고 출력합니다. 출력 결과를 보시면 분류기의 성능인 accuracy는 0.9667로 높아지고 precision, recall의 값도 0.97로 모두 향상 되었습니다. 그림 40은 train_test_split 함수를 이용해 생성한 training data와 test data로 실험한 성능에 대한 내용입니다.

```
clf = RandomForestClassifier(n_estimators=10)
clf.fit(X_train, Y_train)
prediction_1 = rfc.predict(X_test)
#print (prediction_1 == Y_test)
print ("Accuracy is : ",accuracy_score(prediction_1, Y_test))
print ("============================================================")
print (classification_report(prediction_1, Y_test))
```

```
Accuracy is :  0.966666666667
============================================================
             precision    recall  f1-score   support

          0       1.00      1.00      1.00        12
          1       1.00      0.89      0.94         9
          2       0.90      1.00      0.95         9

avg / total       0.97      0.97      0.97        30
```

그림 40 성능평가

이처럼 분류 예측 모듈을 만들 때 훈련 데이터를 제대로 선택해 모듈을 학습시키고 테스트 데이터를 합리적으로 선택하는 것은 매우 중요하다는 것을 알 수 있겠죠?

2.3.12 Random Forest 분류 성능 높이는 방법

Random forest 분류 예측 모듈의 성능을 높이는 방법은 아주 많습니다. 그 중 제일 간단한 방법은 random forest를 생성할 때 트리의 개수를 적당히 확장하는 것입니다. 간단하지만 실전에서 많이 사용되고 효율적인 방법 중 하나입니다. 하지만 주의할 것은 트리의 개수가 많을수록 random forest의 성능이 높아지는 것은 아니라는 것입니다.

트리의 개수가 일정 개수를 넘어가면 오히려 성능이 낮아질 수도 있습니다. 그리고 또 하나의 간단한 방법은 max_features의 값을 수정하는 것입니다. 하지만 여기서는 iris 데이터 셋의 사이즈가 작고 데이터가 간단하기 때문에 성능 차이가 크지 않습니다. 코드에서 두 번째 Line에서 tree가 200개인 random forest를 생성하였습니다. max_features의 값은 4로 설정하고 obb_score의 값은 true로 선택해 out-of-bag을 사용한다고 설정하였습니다. 그리고 위에서와 같이 모듈을 학습시키고 테스트 데이터의 목표를 예측한 결과를 보면 성능의 차이가 보이지 않습니다. 그림 41은 tree가 200개인 random forest의 성능평가입니다.

```
# Initialize the model
clf_2 = RandomForestClassifier(n_estimators=200, # Number of trees
                               max_features=4,    # Num features considered
                                  oob_score=True)    # Use OOB scoring*
clf_2.fit(X_train, Y_train)
prediction_2 = clf_2.predict(X_test)
print (prediction_2 == Y_test)
print ("Accuracy is : ",accuracy_score(prediction_2, Y_test))
print ("=============================================================")
print (classification_report(prediction_2, Y_test))
```

```
[ True  True  True  True  True  True  True  True  True  True False  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True]
Accuracy is :  0.966666666667
=============================================================
             precision    recall  f1-score   support

          0       1.00      1.00      1.00        12
          1       1.00      0.89      0.94         9
          2       0.90      1.00      0.95         9

avg / total       0.97      0.97      0.97        30
```

그림 41 tree가 200개인 random forest의 성능평가

이것은 iris 데이터가 간단하고 size가 작기 때문입니다. 여기서 obb_score의 값을 true로 선택하였는데 이렇게 하면 특징들의 중요도를 볼 수 있습니다.

2.3.13 각 feature의 중요도 확인

코드에서 첫 번째 Line은 분류 모듈에서 특징의 이름에 따라 중요도를 가져와 두 번째 Line에서 출력합니다. 결과를 보면 feature (벚꽃 잎의 너비) petal width의 중요도가 가장 높습니다. 그림 42는 feature 중요도에 대한 내용입니다.

```
for feature, imp in zip(iris.feature_names, clf_2.feature_importances_):
    print(feature, imp)
```

```
sepal length (cm) 0.0142678005207
sepal width (cm) 0.0179653187533
petal length (cm) 0.386870282196
petal width (cm) 0.58089659853
```

그림 42 각 feature 중요도

Random Forest는 여러 개의 의사결정트리를 결합해 하나의 모형을 생성합니다. Random Forest에서 트리의 개수가 많을수록 좋은 것은 아닙니다. 분류 예측 모듈에서 훈련 데이터와 테스트 데이터의 선택은 매우 중요합니다.