의사결정트리 (2)

개요

과거에 수집된 데이터들을 분석하여 이들 사이에 존재하는 패턴(범주별 특성)을 속성의 조합으로 나타나내느 분류 모형

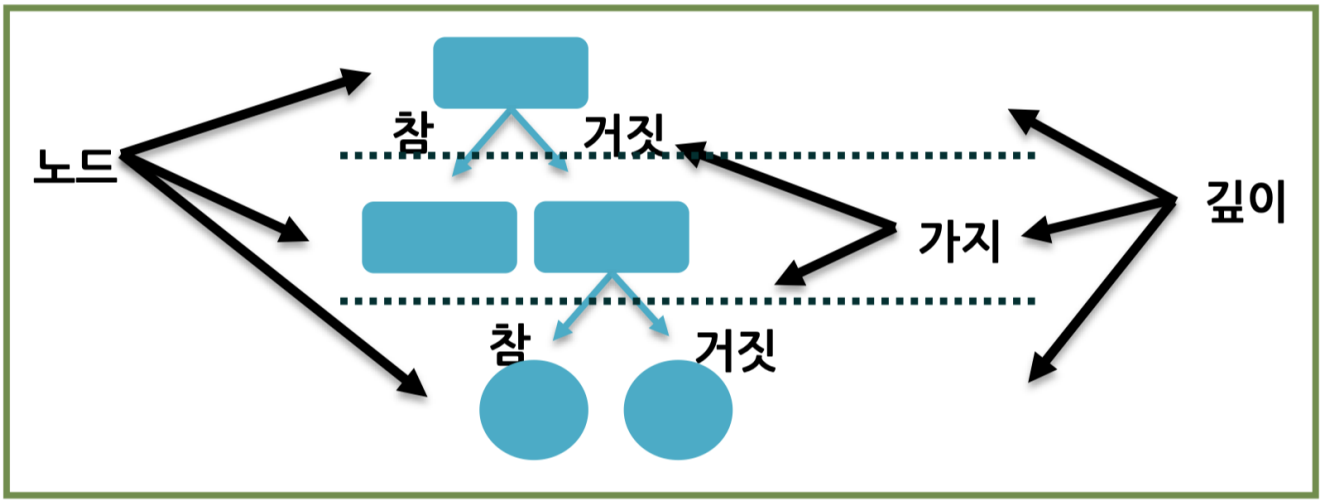
- 새로운 데이터에 대한 분류
- 해당 범주의 값을 예측
- 데이터로부터 트리 구조의 일반화된 지식을 추출

의사결정트리 유형

1. 범주형: 분류트리
2. 연속형: 회귀트리

의사결정트리 구성

대표적으로 노드(Node), 가지(Branch), 깊이(Depth)로 구성



- Root Node : 시작점
- Child Node : 하나 이상의 노드로부터 분리되어 나간 2개 이상의 노드들
- Parent Node : 특정 노드의 상위 노드
- Terminal Node : 더이상 자식을 갖지 않는 노드
- Internal Node : 부모와 자식을 모두 갖는 노드

의사결정트리 특징

장점

- 이해하기 쉬운 규칙이 생성된다. (if~else)
- 분류예측에 유용하지만 회귀예측도 가능 (범주형, 연속형 모두 가능)
- 어느 변수가 상대적으로 더 중요한지 확인 가능
- 비교적 빠른 의사결정이 가능

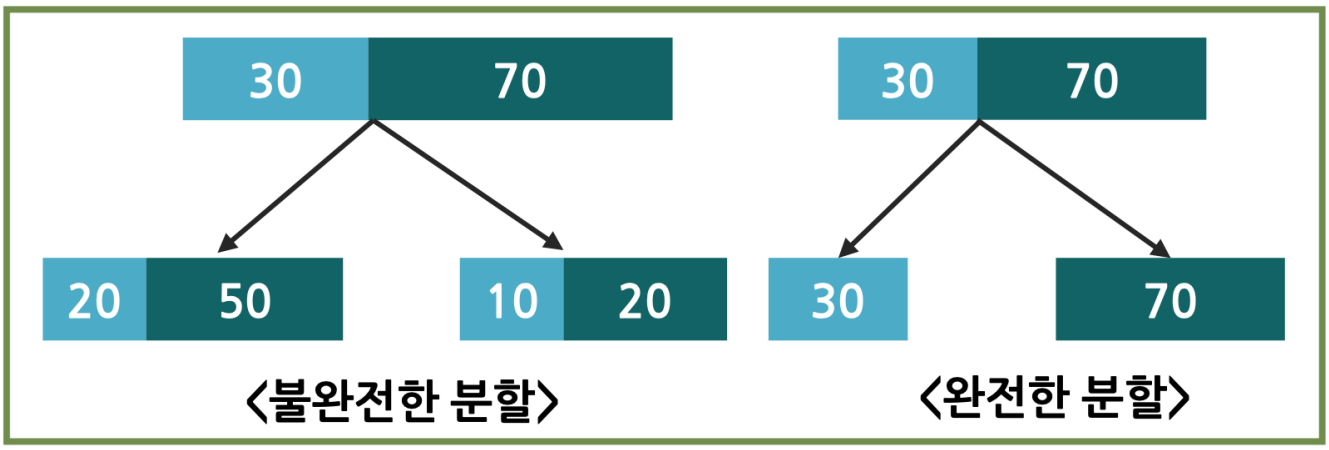
단점

- 연속형 변수값을 예측(회귀)할 때 예측력이 떨어짐(적당하지 않다.)
- 트리가 복잡할 수록 예측력 저하, 해석이 어려움, 상황에 따라 계산량이 많아서 처리속도 느림
- 안정성이 떨어짐(데이터에 약간의 변형이 있는 경우 결과가 나빠질 수 있음)

의사결정트리 진행 절차

의사결정트리 분리

- 훈련용 데이터를 이용하여 독립변수의 차원 공간을 반복적으로 분할
- 평가용 데이터를 이용하여 가지치기를 수행
- 분할기준: 부모마디마다 자식마디의 **순수도**가 증가하도록 분류를 형성
  - **순수도** : 특정 범주의 개체들이 포함되어 있는 정도
- 순수한 데이터 비율이 높을수록 좋은 트리가 됨



반복적 분리 과정

- 위의 과정을 최종 노드에 포함된 변수가 모두 동일한 집단에 속하도록 하는 것

#01. 패키지 참조

```
from sklearn.datasets import load_breast_cancer
from pandas import DataFrame
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.model_selection import cross_val_score, cross_validate

from sklearn.metrics import accuracy_score, recall_score
from sklearn.metrics import precision_score, f1_score
from sklearn.metrics import confusion_matrix
```

#02. 데이터 가져오기

유방암 진단을 위한 데이터셋.

30개의 독립변수를 통해 유방암 진단을 결정한다.

```
dataset = load_breast_cancer()
```

```
origin = DataFrame(data=dataset.data, columns=dataset.feature_names)
origin['target'] = dataset.target

print(origin.info())

origin.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   mean radius              569 non-null    float64
 1   mean texture             569 non-null    float64
 2   mean perimeter           569 non-null    float64
 3   mean area                569 non-null    float64
 4   mean smoothness          569 non-null    float64
 5   mean compactness         569 non-null    float64
 6   mean concavity           569 non-null    float64
 7   mean concave points      569 non-null    float64
 8   mean symmetry            569 non-null    float64
 9   mean fractal dimension   569 non-null    float64
 10  radius error             569 non-null    float64
 11  texture error            569 non-null    float64
 12  perimeter error          569 non-null    float64
 13  area error               569 non-null    float64
 14  smoothness error         569 non-null    float64
 15  compactness error        569 non-null    float64
 16  concavity error          569 non-null    float64
 17  concave points error     569 non-null    float64
 18  symmetry error           569 non-null    float64
 19  fractal dimension error  569 non-null    float64
 20  worst radius             569 non-null    float64
 21  worst texture            569 non-null    float64
 22  worst perimeter          569 non-null    float64
 23  worst area               569 non-null    float64
 24  worst smoothness         569 non-null    float64
 25  worst compactness        569 non-null    float64
 26  worst concavity          569 non-null    float64
 27  worst concave points     569 non-null    float64
 28  worst symmetry           569 non-null    float64
 29  worst fractal dimension  569 non-null    float64
 30  target                   569 non-null    int32
dtypes: float64(30), int32(1)
memory usage: 135.7 KB
None
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	m symme
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	m symme
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809

5 rows × 31 columns

#03. 데이터 전처리

훈련 데이터 검증 데이터 분리

```
x = origin.drop('target', axis=1)
y = origin['target']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3,
                                                    random_state=777)

x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

```
((398, 30), (171, 30), (398,), (171,))
```

#04. 의사결정트리 학습

학습모델 구축

```
dtree = DecisionTreeClassifier()
dtree.fit(x_train, y_train)
y_pred = dtree.predict(x_test)
y_pred[:5]
```

```
array([1, 1, 0, 1, 0])
```

훈련 정확도

accuracy_score와 동일한 값

```
dtree.score(x_train, y_train), dtree.score(x_test, y_test)
```

```
(1.0, 0.9122807017543859)
```

성능평가

```
acc = accuracy_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
matrix = confusion_matrix(y_test, y_pred)

print('Accuracy: ', acc)
print('Recall: ', recall)
print('Precision: ', precision)
print('F1_score: ', f1)
print('Confusion Matrix:', matrix)
```

```
Accuracy:  0.9122807017543859
Recall:  0.9122807017543859
Precision:  0.9541284403669725
F1_score:  0.9327354260089685
Confusion Matrix: [[ 52   5]
 [ 10 104]]
```

#05. 성능 향상 대작전~!!

K-Fold

정확도는 다소 떨질 수 있으나 fold로 설정한 수 만큼 학습을 진행한 후 평균 정확도를 내는 방식이므로 결과의 안정성이 더 높아진다고 할 수 있다.

```
# 분할되지 않은 원본 데이터를 전달해야 한다.
# -> 그 데이터를 5쌍으로 분할하겠다는 의미
scores = cross_val_score(dtree, x, y, cv=5, n_jobs=-1)
print(scores)
print("교차검증 평균: ", scores.mean())
```

```
[0.9122807  0.9122807  0.9122807  0.93859649 0.90265487]
교차검증 평균:  0.915618692749573
```

```
cv = cross_validate(dtree, x, y, cv=5, n_jobs=-1)
cvdf = DataFrame(cv)
cvdf
```

	fit_time	score_time	test_score
0	0.016971	0.002000	0.912281
1	0.019997	0.002001	0.938596
2	0.021998	0.003000	0.912281
3	0.021003	0.003002	0.947368
4	0.023004	0.002002	0.902655

하이퍼파라미터 튜닝

여러 개의 설정값을 확인하고 싶은 파라미터를 집어넣어 교차검증을 수행할 수 있게 한다.

```
dtree = DecisionTreeClassifier(random_state=777)

params = {
    'max_depth': [3, 5, 7, 9],
    'min_samples_split': [2, 3, 4],      # 노드를 분할하는데 필요한 최소 샘플 수
    'splitter': ['best', 'random']       # 각 노드에서 분할을 선택하는데 사용되는 전략
}

grid_dt = GridSearchCV(dtree, param_grid=params, cv=5, n_jobs=-1)

grid_dt.fit(x, y)

result = DataFrame(grid_dt.cv_results_['params'])
result['mean_test_score'] = grid_dt.cv_results_['mean_test_score']
result.sort_values(by='mean_test_score', ascending=False)
```

	max_depth	min_samples_split	splitter	mean_test_score
9	5	3	random	0.938472
11	5	4	random	0.938472
7	5	2	random	0.936718
23	9	4	random	0.929685
19	9	2	random	0.926176
3	3	3	random	0.924406
5	3	4	random	0.924406
15	7	3	random	0.924375
1	3	2	random	0.922652
17	7	4	random	0.920882
21	9	3	random	0.917404
13	7	2	random	0.917358
0	3	2	best	0.917358
4	3	4	best	0.917358
2	3	3	best	0.917358
14	7	3	best	0.915619
12	7	2	best	0.915619
20	9	3	best	0.915588
16	7	4	best	0.913864
18	9	2	best	0.913833
22	9	4	best	0.913833

	max_depth	min_samples_split	splitter	mean_test_score
10	5	4	best	0.912110
8	5	3	best	0.912110
6	5	2	best	0.912110