

비즈니스를 위한 데이터마이닝

가톨릭대학교 경영학과

이홍주

분류 회귀 트리 (Classification and Regression Trees)

- 트리와 규칙
 - 목표: 입력변수 집합에 따른 결과 분류 또는 예측
 - 결과물: 규칙의 집합
 - 예 - 신용카드 가입 제안 수락 여부
 - 목표: 신용카드 가입 제안 수락 여부 분류
 - 규칙 예: IF (Income ≥ 106) AND (Education < 1.5) AND (Family ≤ 2.5) THEN Class = 0 (Non acceptor)
 - 의사결정 나무 Decision Tree 라고 많이 불림

질서있게 나누다보면, 규칙을 만들 수 있음!

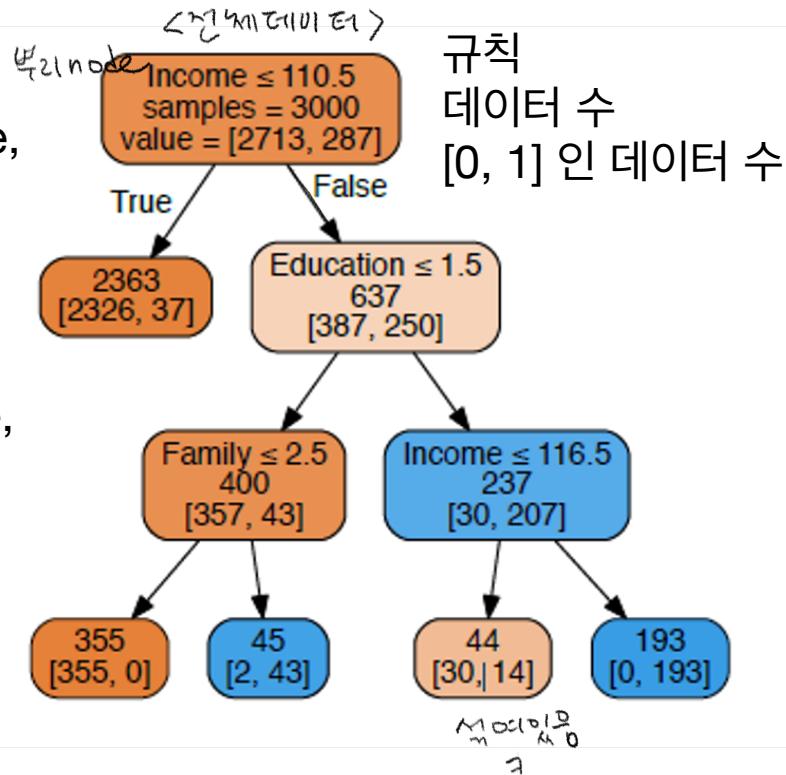
의사결정 나무 예시: 대출 신청 승인 여부

결정 노드

Decision node,
Splitting node

단말 노드

terminal node,
leaf node



의사결정 나무 decision tree

- 재귀적 분할 Recursive partitioning
 - 데이터를 두 부분으로 반복적으로 분할하여 각각의 새로운 부분 내에서 결과의 순수함을 극대화
- 과적합 방지
 - 완전히 성장한 나무는 너무 복잡하여 과적합하게 됨

재귀적 분할

- 입력 변수 중 하나인 x_i 선택
- 학습 데이터를 두 부분으로 나누는 x_i 값(예: s_i)을 선택
 ↑
 임의
- 결과 부분 각각이 얼마나 순수한지의 “순수도(purity)” 또는 동질적인지 측정
- “순수도” = 대부분 한 클래스의 데이터를 포함하는 경우 높아짐
- 알고리즘은 초기 분할에서 순수도를 최대화하기 위해 x_i 와 s_i 의 다른 값을 시도
- “최대 순수도” 분할을 얻은 후, 두 번째 분할(모든 변수에 대해)에 대해 이 프로세스를 반복하는 등의 작업 수행

재귀적 분할 예: 승차식 잔디깎기

- 목표: 24 가구의 승차식 잔디깎기 보유 여부 분류
- 입력변수: Income, Lot Size
대지 대지 면적



표 9-1 24가구의 주택 대지의 크기(Lot size), 소득(Income), 잔디깎이 기계의 소유 여부(Owner)

가구 번호	소득 (1,000달러 단위)	주택 대지의 크기 (1,000제곱피트 단위)	잔디깎이 기계 소유 여부
1	60.0	18.4	Owner
2	85.5	16.8	Owner
3	64.8	21.6	Owner
4	61.5	20.8	Owner
5	87.0	23.6	Owner
6	110.1	19.2	Owner
7	108.0	17.6	Owner
8	82.8	22.4	Owner
9	69.0	20.0	Owner
10	93.0	20.8	Owner
11	51.0	22.0	Owner
12	81.0	20.0	Owner
13	75.0	19.6	Nonowner
14	52.8	20.8	Nonowner
15	64.8	17.2	Nonowner
16	43.2	20.4	Nonowner
17	84.0	17.6	Nonowner
18	49.2	17.6	Nonowner
19	59.4	16.0	Nonowner
20	66.0	18.4	Nonowner
21	47.4	16.4	Nonowner
22	33.0	18.8	Nonowner
23	51.0	14.0	Nonowner
24	63.0	14.8	Nonowner

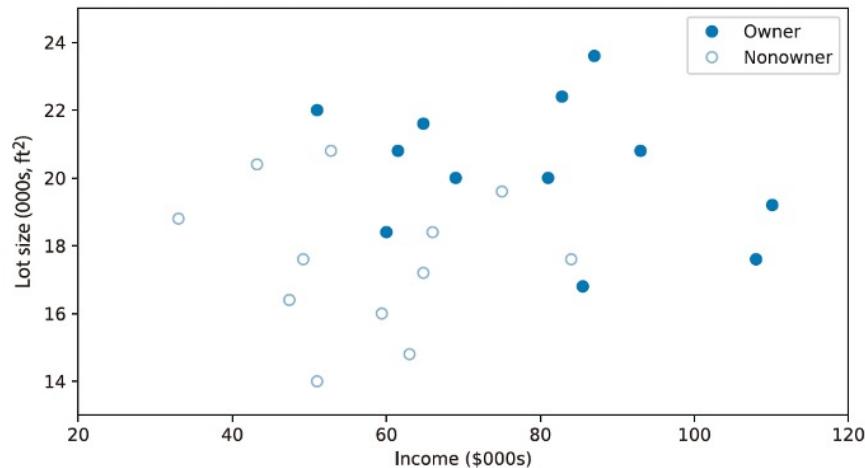


그림 9-2 승차식 기계 소유 및 비소유에 대한 24가구의 주택 대지의 크기 vs. 소득에 대한 산점도

재귀적 분할 예: 승차식 잔디깎기

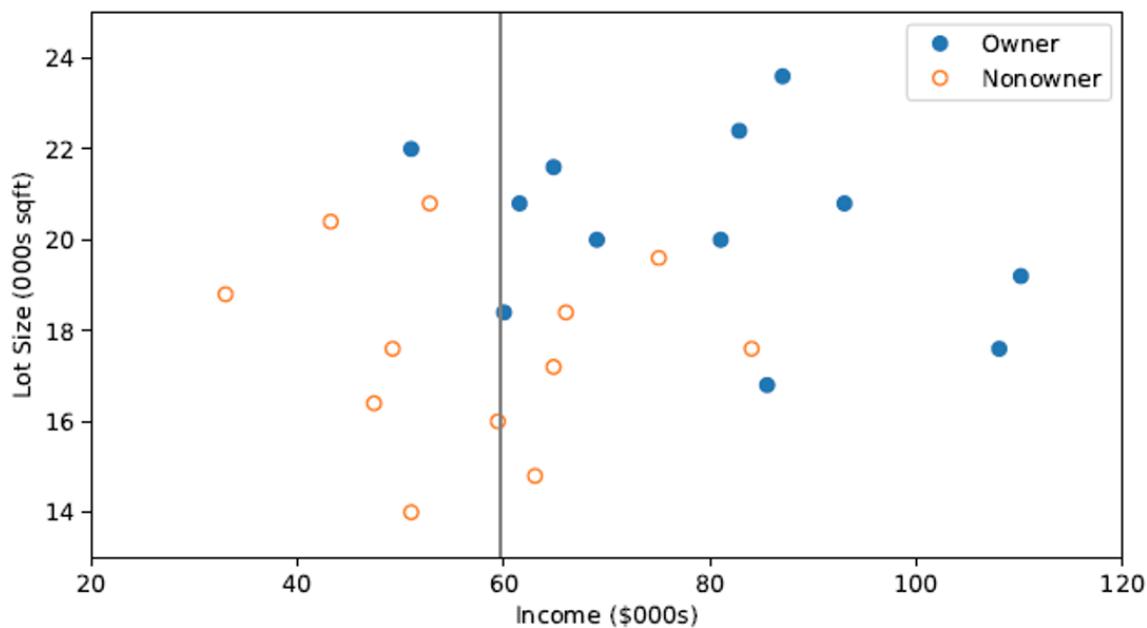
- 분할 방법
 - 하나의 변수(예: Income)에 따라 데이터 정렬
 - 59.7 과 같은 임의의 값을 사용해 레코드를 $\text{Income} \geq 59.7$ 인 데이터 와 < 59.7 인 데이터로 나눔
 - 각 결과 부분에서 클래스의 결과 순수도(동질성)를 측정
 - 다른 모든 분할 값을 시도
 - 다른 변수에 대해 반복
 - 가장 순수도가 높은 변수 및 분할을 하나 선택

재귀적 분할 예: 승차식 잔디깎기

- 범주형 변수 분할 방법
 - 가능한 모든 범주의 조합으로 분할해봄
 - A, B, C 세 개의 범주는 아래와 같이 3개의 조합으로 분할됨
 - $\{A\}, \{B, C\}$
 - $\{B\}, \{A, C\}$
 - $\{C\}, \{A, B\}$

재귀적 분할 예: 승차식 잔디깎기

- 첫 분할: Income = 59.7



재귀적 분할 예: 승차식 잔디깎기

- 두번째 분할: Lot Size = 21.4

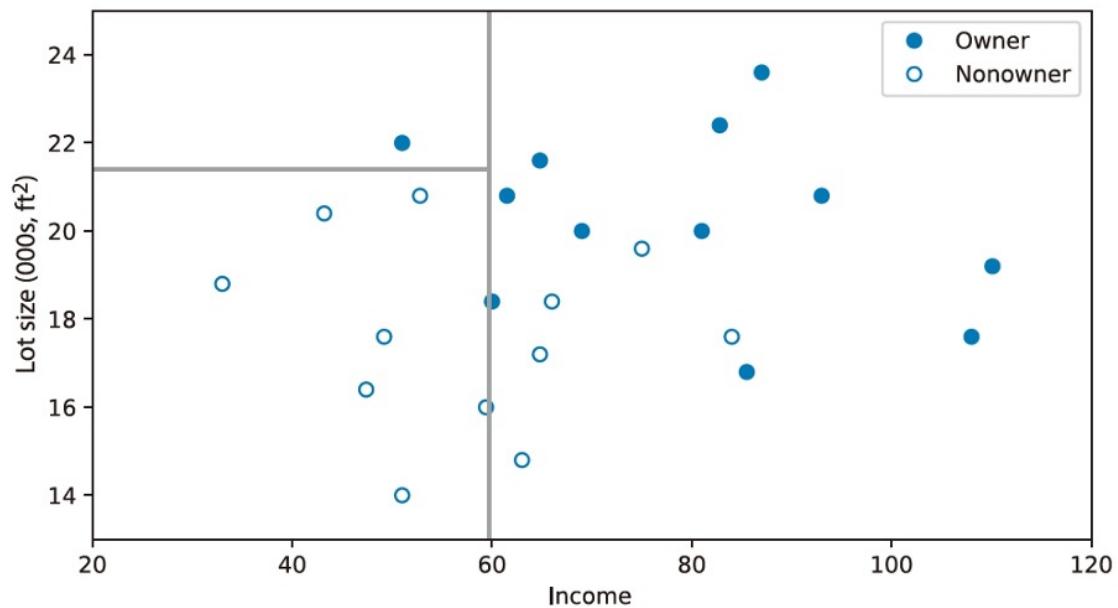


그림 9-5 먼저 소득 59.7로, 그다음에는 대지의 크기 21.4을 기준으로 24개 관측치를 분할한다.

재귀적 분할 예: 승차식 잔디깎기

- 모든 분할 후

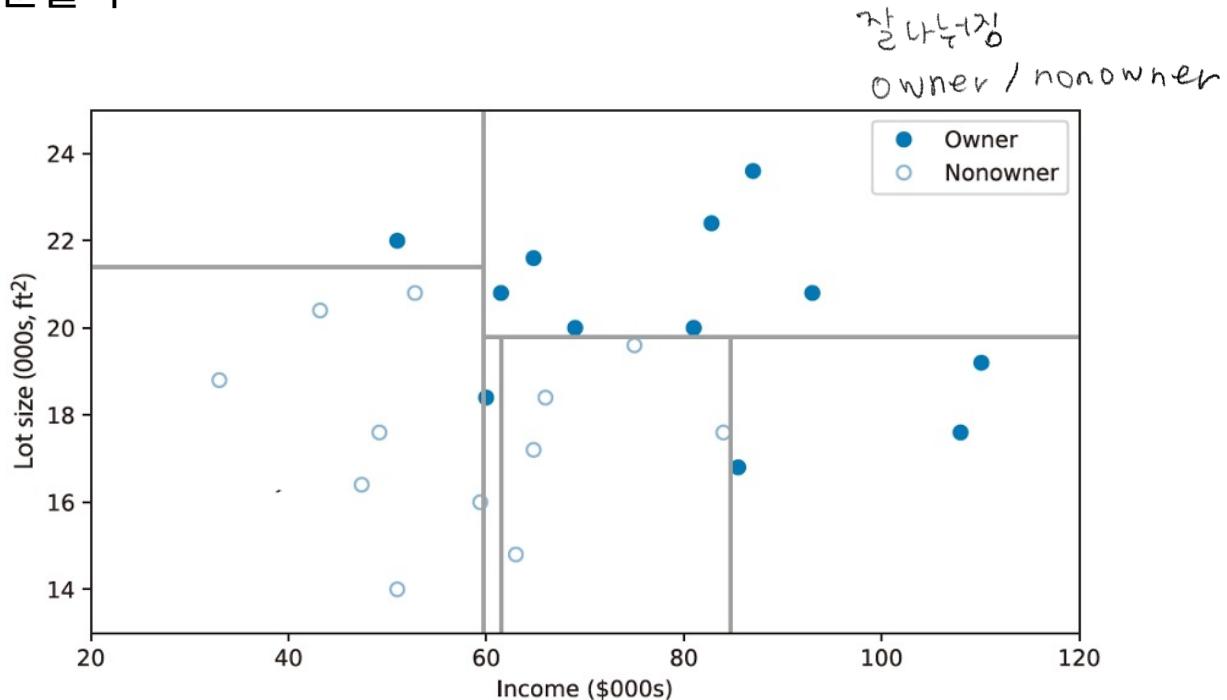


그림 9-6 재귀적 분할의 마지막 단계. 각 직사각형은 하나의 클래스(소유 가구 또는 비소유 가구)로만 구성되어 있다.

불순도 impurity 측정

- 지니 불순도 지수

$$\bullet \text{Gini} = \frac{1}{\text{불순도 max}} - \sum_{k=1}^m p_k^2$$

- p_k 는 m 개의 클래스 중에서 k 클래스에 속한 데이터의 비율

- 모든 데이터가 하나의 클래스에 속하면 Gini 값은 0

- 모든 데이터가 균등하게 모든 클래스에 속할 때 최대값

- 이진 분류에서는 최대값은 0.5

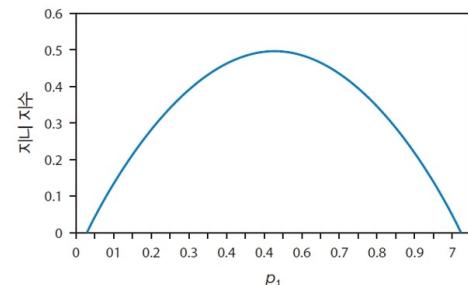


그림 9-4 이진 클래스 문제에 대한 지니 지수의 값: 클래스 1에 속하는 관측치 비율(P_1)에 대한 함수 관계

불순도 impurity 측정

- 엔트로피 Entropy $\stackrel{\text{= 무질서도}}{\text{= 얼마나 무질서?}}$
 $\therefore \text{작은게 좋은것!}$

- $$\text{Entropy}(H) = - \sum_{k=1}^m p_k \log_2 p_k$$

 $\xrightarrow{\text{클래스 비율}}$

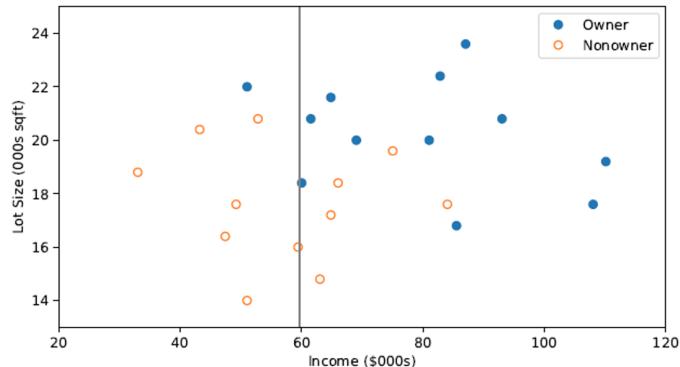
- p_k 는 m 개의 클래스 중에서 k 클래스에 속한 데이터의 비율
- 엔트로피 값의 범위는 가장 순수한 경우가 0이며, 가장 순수하지 않은 경우는 $\log_2(m)$ (m 개의 클래스에 고르게 분포)

- $$p_k = \frac{1}{m} \quad \text{for all } k = 1, 2, \dots, m, \quad H = - \sum_{k=1}^m \frac{1}{m} \log_2 \frac{1}{m}$$

$$\begin{aligned} H &= -m \cdot \frac{1}{m} \log_2 \frac{1}{m} \\ &= -\log_2 \frac{1}{m} \quad \therefore \text{시험 X} \\ &= \log_2 m \end{aligned}$$

불순도 impurity 측정

- 첫번째 분할 불순도 계산
- 왼쪽: 소유자 1, 비소유자 7
- 오른쪽: 소유자 11, 비소유자 5
- 왼쪽, 오른쪽의 불순도 계산 후
가중평균



$$\text{gini_left} = 1 - (7/8)^2 - (1/8)^2 = 0.219$$

$$\text{entropy_left} = -(7/8) \log_2(7/8) - (1/8) \log_2(1/8) = 0.544$$

$$\text{gini_right} = 1 - (11/16)^2 - (5/16)^2 = 0.430$$

$$\text{entropy_right} = -(11/16) \log_2(11/16) - (5/16) \log_2(5/16) = 0.896$$

→ $\text{gini} = (8/24)(0.219) + (16/24)(0.430) = 0.359$

? $\text{entropy} = (8/24)(0.544) + (16/24)(0.896) = 0.779$

낮을 대체로 좋은 것

불순도와 재귀적 분할

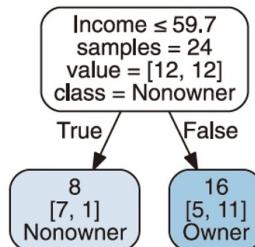
- 전체 불순도 측정값을 얻음
- 연속되는 각 단계에서 모든 변수에서 가능한 모든 분할에 대해 이 측정값을 비교
- 불순도를 가장 많이 줄이는 분할을 선택
인정된 합법적인 날을 끄내까지
- 선택한 분할 지점이 트리의 노드가 됨

예제 1: 승차식 잔디깍기

- 첫번째 분할

분류 트리를 실행하고 시각화하는 파이썬 코드

```
mower_df = pd.read_csv('RidingMowers.csv')
# use max_depth to control tree size (None = full tree)
classTree = DecisionTreeClassifier(random_state=0, max_depth=1)
classTree.fit(mower_df.drop(columns=['Ownership']), mower_df['Ownership'])
print("Classes: {}".format(', '.join(classTree.classes_)))
plotDecisionTree(classTree, feature_names=mower_df.columns[:2],
                  class_names=classTree.classes_)
```



default는 지나!로 불순도 측정

그림 9-7 첫 번째 분할에 대한 트리 모델 표현([그림 9-3]에 대응)

예제 1: 승차식 잔디깍기

- 세번째 분할

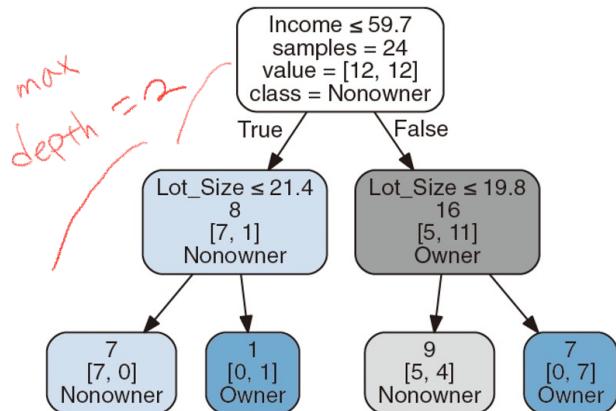


그림 9-8 처음 3개 분할에 대한 트리 모델 표현

예제 1: 승차식 잔디깍기

- 모든 분할 후

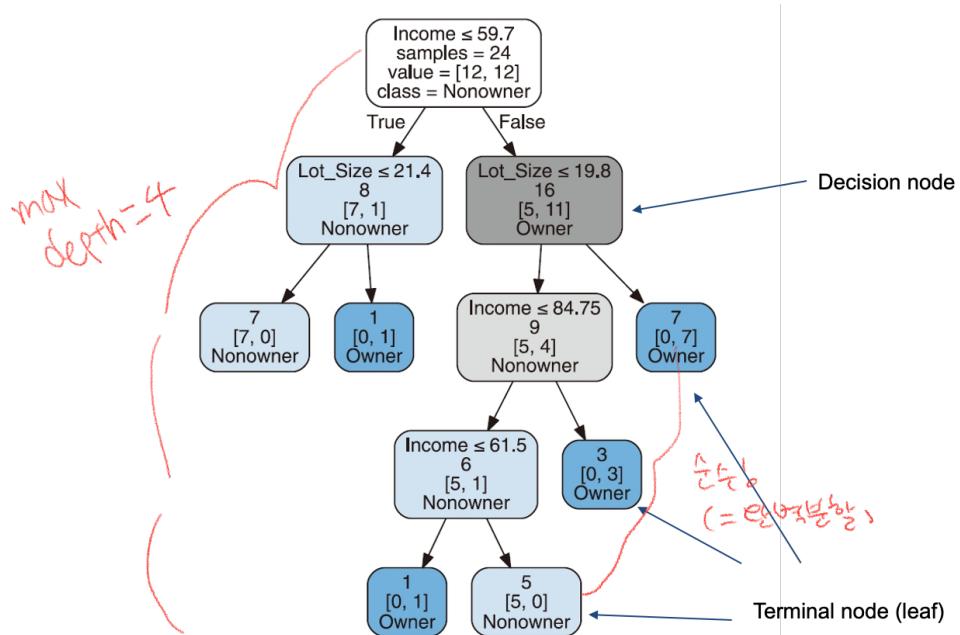


그림 9-9 모든 분할이 끝난 트리 모델 표현([그림 9-6]에 대응). 완전히 성장한 트리 모델

예제 1: 승차식 잔디깍기

- 나무를 읽어 규칙 도출

If Income <= 59.7
Lot Size <= 21.4,
classify as "Nonowner"

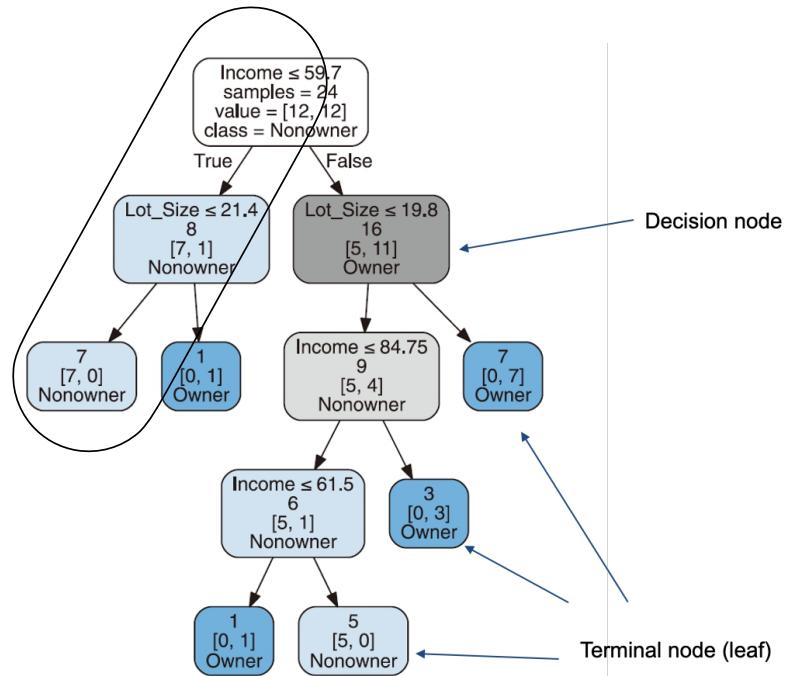


그림 9-9 모든 분할이 끝난 트리 모델 표현([그림 9-6]에 대응). 완전히 성장한 트리 모델

예제 2: 개인 대출 수락

목표변수



표 9-2 유니버설 은행의 20명 고객에 대한 표본 데이터

ID	Age	Professional experience		Family size	CC avg.	Education	Mortgage	Personal loan	Securities account	CD account	Online banking	Credit card
		experience	Income									
1	25	1	49	4	1.60	UG	0	No	Yes	No	No	No
2	45	19	34	3	1.50	UG	0	No	Yes	No	No	No
3	39	15	11	1	1.00	UG	0	No	No	No	No	No
4	35	9	100	1	2.70	Grad	0	No	No	No	No	No
5	35	8	45	4	1.00	Grad	0	No	No	No	No	Yes
6	37	13	29	4	0.40	Grad	155	No	No	No	Yes	No
7	53	27	72	2	1.50	Grad	0	No	No	No	Yes	No
8	50	24	22	1	0.30	Prof	0	No	No	No	No	Yes
9	35	10	81	3	0.60	Grad	104	No	No	No	Yes	No
10	34	9	180	1	8.90	Prof	0	Yes	No	No	No	No
11	65	39	105	4	2.40	Prof	0	No	No	No	No	No
12	29	5	45	3	0.10	Grad	0	No	No	No	Yes	No
13	48	23	114	2	3.80	Prof	0	No	Yes	No	No	No
14	59	32	40	4	2.50	Grad	0	No	No	No	Yes	No
15	67	41	112	1	2.00	UG	0	No	Yes	No	No	No
16	60	30	22	1	1.50	Prof	0	No	No	No	Yes	Yes
17	38	14	130	4	4.70	Prof	134	Yes	No	No	No	No
18	42	18	81	4	2.40	UG	0	No	No	No	No	No
19	46	21	193	2	8.10	Prof	0	Yes	No	No	No	No
20	55	28	21	1	0.50	Grad	0	No	Yes	No	No	Yes

예제 2: 개인 대출 수락

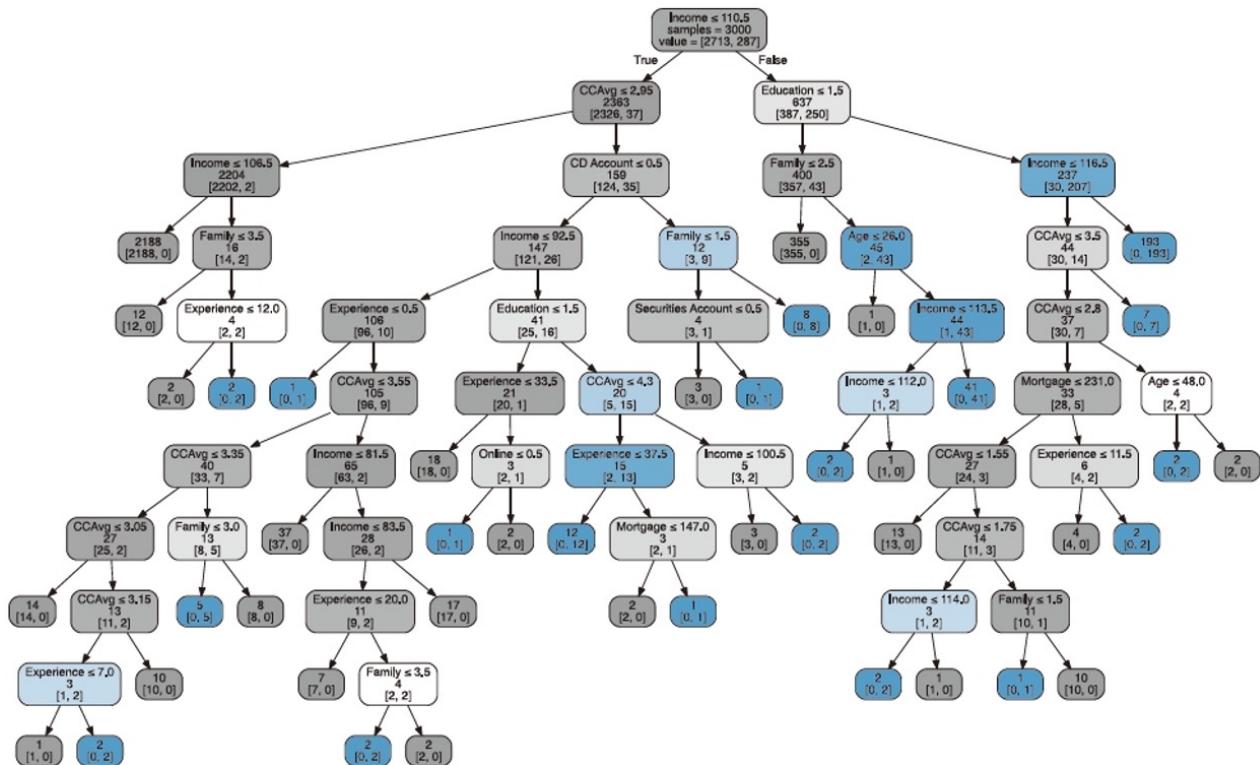


그림 9-10 개인 대출 수락 데이터의 학습 데이터셋(관측치 3,000개)을 이용해서 만든 완전히 성장한 트리 모델

예제 2: 개인 대출 수락



완전히 성장한 분류 트리 모델을 생성하기 위한 파이썬 코드

```
bank_df = pd.read_csv('UniversalBank.csv')
bank_df.drop(columns=['ID', 'ZIP Code'], inplace=True)
X = bank_df.drop(columns=['Personal Loan'])
y = bank_df['Personal Loan']
train_X, valid_X, train_y, valid_y = train_test_split(X, y, test_size=0.4, random_state=1)

fullClassTree = DecisionTreeClassifier(random_state=1)  max_depth 제일 깊은 노드를 가장 높이에 만든다!
fullClassTree.fit(train_X, train_y)

plotDecisionTree(fullClassTree, feature_names=train_X.columns)
```

예제 2: 개인 대출 수락

표 9-3 대출 승인 데이터의 학습·검증 데이터셋에 대한 기본(전체) 분류 트리 모델의 정오 행렬표와 정확도



하나의 트리 모델을 사용해 검증 데이터셋을 분류하고, 학습·검증 데이터셋에 대한 정오 행렬표와 정확도를 계산하기 위한 파이썬 코드

```
classificationSummary(train_y, fullClassTree.predict(train_X))  
classificationSummary(valid_y, fullClassTree.predict(valid_X))
```

실행 결과

```
# full tree: training
```

```
Confusion Matrix (Accuracy 1.0000)      학습 집합 더 높긴 해
```

		Prediction
		0 1
Actual	0	2727 0
	1	0 273

```
# full tree: validation
```

```
Confusion Matrix (Accuracy 0.9790)
```

		Prediction
		0 1
Actual	0	1790 17
	1	25 168



예제 2: 개인 대출 수락

표 9-4 5겹 교차 검증을 이용한 5개의 검증 분할에 대한 기본(전체) 분류 트리 모델의 정확성



완전히 성장한 트리에 5겹 교차 검증을 수행하여 정확도를 계산하는 파이썬 코드

```
treeClassifier = DecisionTreeClassifier(random_state=1)
scores = cross_val_score(treeClassifier, train_X, train_y, cv=5)
print('Accuracy scores of each fold: ', [f'{acc:.3f}' for acc in scores])
```

실행 결과

```
Accuracy scores of each fold: ['0.985', '0.972', '0.992', '0.987', '0.992']
```

최대 나무 full tree

- 개념적으로는 leaf node에 100% 순수 도 달성
- 이는 학습데이터에 과적합된 모형을 만들 수 있음
- 의사결정나무는 불안정할 수 있음
 - 2개 이상의 입력변수 중요도가 비슷한 상황에서 어떤 변수가 먼저 선택되는지 가 의사결정나무 성과에 영향을 미침
 - 학습/테스트 집합으로 나누는 것에 따라 다른 나무가 생성될 수 있음

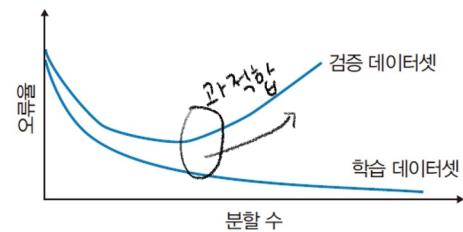


그림 9-11 과적합: 분할 노드 수의 함수로서의 오류율(학습 데이터셋 vs. 검증 데이터셋)

과적합 방지하기

- 의사결정 나무 크기 제한하기

안정적인 나무를 만들고자



- 최대 깊이 max_depth

노드에 있는 데이터를

- 가지칠 최소 데이터 수 min_samples_split
- Leaf node 최소 데이터 수 min_samples_leaf
- 최소 불순도 증가치 min_impurity_decrease
- 최적의 값은?

노드 터너 ~ 불순도↓하는데 그게 너무 미미한 때!

과적합 방지하기



작은 분류 트리 모델을 생성하는 파이썬 코드

```
smallClassTree = DecisionTreeClassifier(max_depth=30, min_samples_split=20,
                                         min_impurity_decrease=0.01, random_state=1)
smallClassTree.fit(train_X, train_y)

plotDecisionTree(smallClassTree, feature_names=train_X.columns)
```

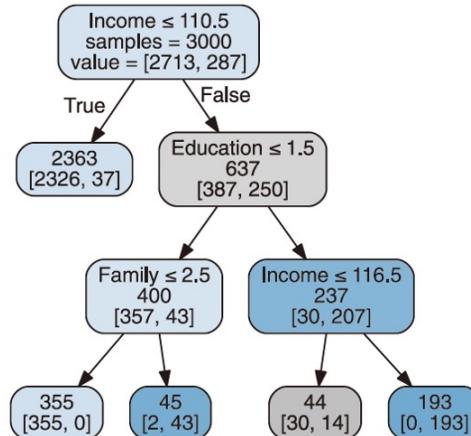


그림 9-12 개인 대출 수락 데이터의 학습 데이터셋(3,000 관측치)을 이용해서 만든 작은 분류 모델

과적합 방지하기

표 9-5 작은 분류 트리 모델: 학습 및 검증 데이터셋에 대한 정오 행렬표와 정확도(개인 대출 예제)



작은 분류 트리 모델을 사용해 검증 데이터셋을 분류하고 학습 및 검증 데이터셋에 대한 정오 행렬표와 정확도를 계산하는 파이썬 코드

```
classificationSummary(train_y, smallClassTree.predict(train_X))
classificationSummary(valid_y, smallClassTree.predict(valid_X))
```

실행 결과

```
# full tree: training
Confusion Matrix (Accuracy 1.0000)
```

Prediction		
Actual	0	1
0	2711	2
1	51	236

```
# full tree: validation
Confusion Matrix (Accuracy 0.9790)
```

Prediction		
Actual	0	1
0	1804	3
1	43	150

GridSearchCV()

파라미터 값을 선택하는 것을 파라미터 튜닝 Parameter tuning이라고 함

```
# Start with an initial guess for parameters
param_grid = {
    'max_depth': [10, 20, 30, 40],           # depth 4x4x5=100
    'min_samples_split': [20, 40, 60, 80, 100],
    'min_impurity_decrease': [0, 0.0005, 0.001, 0.005, 0.01],
}

# Which values are best?
# n_jobs=-1 will utilize all available CPUs
gridSearch =
    GridSearchCV(DecisionTreeClassifier(random_state=1),
                 param_grid, cv=5, n_jobs=-1)
gridSearch.fit(train_X, train_y)
print('Initial score: ', gridSearch.best_score_)
print('Initial parameters: ', gridSearch.best_params_)
```

GridSearchCV() code – refined

```
# Adapt grid based on result from initial grid search
param_grid = {
    'max_depth': list(range(2, 16)), # 14 values
    'min_samples_split': list(range(10, 22)), # 11 values
    'min_impurity_decrease': [0.0009, 0.001, 0.0011], # 3 values
}

gridSearch =
    GridSearchCV(DecisionTreeClassifier(random_state=1),
        param_grid, cv=5, n_jobs=-1)
gridSearch.fit(train_X, train_y)
print('Improved score: ', gridSearch.best_score_)
print('Improved parameters: ', gridSearch.best_params_)
bestClassTree = gridSearch.best_estimator_
```

실행 결과

```
Initial score: 0.988
Initial parameters: {'max_depth': 10, 'min_impurity_decrease': 0.001,
                     'min_samples_split': 20}

Improved score: 0.9883333333333333
Improved parameters: {'max_depth': 5, 'min_impurity_decrease': 0.0011,
                      'min_samples_split': 13}
```

과적합 방지하기



미세 조정한 분류 트리의 성능을 플로팅 및 평가하는 파이썬 코드

성능 평가

```
# fine-tuned tree: training  
> classificationSummary(train_y, bestClassTree.predict(train_X))  
Confusion Matrix (Accuracy 0.9900)
```

Prediction		
Actual	0	1
0	2703	10
1	20	267

```
# fine-tuned tree: validation  
> classificationSummary(valid_y, bestClassTree.predict(valid_X))  
Confusion Matrix (Accuracy 0.9825)
```

Prediction		
Actual	0	1
0	1793	14
1	21	172

트리 플로팅

```
plotDecisionTree(bestClassTree, feature_names=train_X.columns)
```

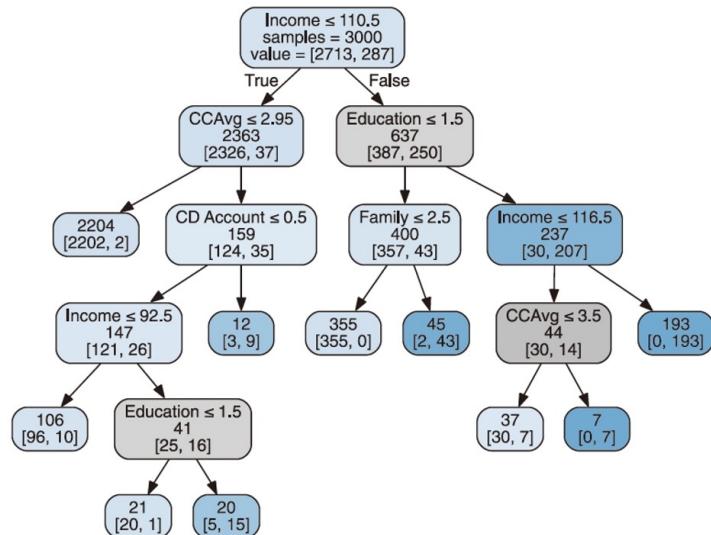


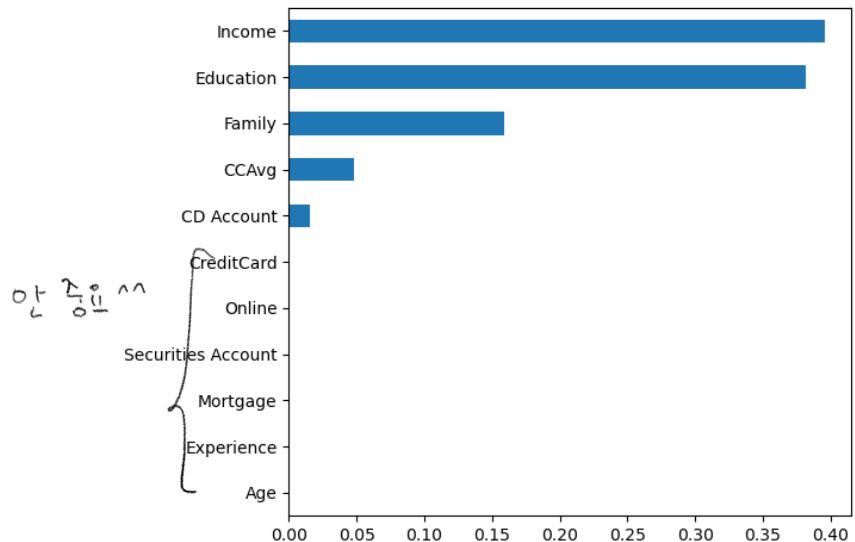
그림 9-13 학습 데이터셋을 사용한 대출 수락 데이터에 대해 최적으로 가지치기된 트리 모델(관측치 3,000개)

변수 중요도

```
importances = bestClassTree.feature_importances_
df = pd.DataFrame({'feature': train_X.columns, 'importance': importances})
df = df.sort_values('importance')
print(df)

ax = df.plot(kind='barh', x='feature', legend=False)
ax.set_ylabel('')

plt.tight_layout()
plt.show()
```



RandomizedCV()

- 가능한 파라미터 조합 중에서 무작위로 선택해서 수행
- GridSearch()가 너무 많은 조합을 탐색해야하는 경우 선택할 수 있음
- GridSearch()처럼 선택할 수 있는 값을 입력하는 대신 범위를 입력할 수 있음



회귀 나무 모델 Regression Trees

- 의사결정나무를 수치 예측 문제에 적용하는 것
- 수치 예측은 leaf 노드에 속한 데이터들의 평균값
- 불순도는 노드의 평균에서 편차 제곱의 합으로 계산함
- 성과지표는 수치예측에 활용하는 지표 사용 (RMSE 등)

회귀 나무 모델 Regression Trees

표 9-7 최적으로 가치지기된 회귀 트리 모델의 학습과 평가

 회귀 트리를 생성하는 파이썬 코드

```
from sklearn.tree import DecisionTreeRegressor
toyotaCorolla_df = pd.read_csv('ToyotaCorolla.csv').iloc[:1000,:]
toyotaCorolla_df = toyotaCorolla_df.rename(columns={'Age_08_04': 'Age', 'Quarterly_Tax': 'Tax'})

predictors = ['Age', 'KM', 'Fuel_Type', 'HP', 'Met_Color', 'Automatic', 'CC',
              'Doors', 'Tax', 'Weight']
outcome = 'Price'

X = pd.get_dummies(toyotaCorolla_df[predictors], drop_first=True)
y = toyotaCorolla_df[outcome]

train_X, valid_X, train_y, valid_y = train_test_split(X, y, test_size=0.4, random_state=1)

# user grid search to find optimized tree
param_grid = {
    'max_depth': [5, 10, 15, 20, 25],
    'min_impurity_decrease': [0, 0.001, 0.005, 0.01],
    'min_samples_split': [10, 20, 30, 40, 50],
}
gridSearch = GridSearchCV(DecisionTreeRegressor(), param_grid, cv=5, n_jobs=-1)
gridSearch.fit(train_X, train_y)
print('Initial parameters: ', gridSearch.best_params_)
```

회귀 나무 모델 Regression Trees

```
param_grid = {
    'max_depth': [3, 4, 5, 6, 7, 8, 9, 10, 11, 12],
    'min_impurity_decrease': [0, 0.001, 0.002, 0.003, 0.005, 0.006, 0.007, 0.008],
    'min_samples_split': [14, 15, 16, 18, 20, ],
}

gridSearch = GridSearchCV(DecisionTreeRegressor(), param_grid, cv=5, n_jobs=-1)
gridSearch.fit(train_X, train_y)
print('Improved parameters: ', gridSearch.best_params_)

regTree = gridSearch.best_estimator_

regressionSummary(train_y, regTree.predict(train_X))
regressionSummary(valid_y, regTree.predict(valid_X))
```

실행 결과

```
Initial parameters: {'max_depth': 10,'min_impurity_decrease': 0.001,'min_samples_split': 20}
Improved parameters: {'max_depth': 6,'min_impurity_decrease': 0.01,'min_samples_split': 12}
```

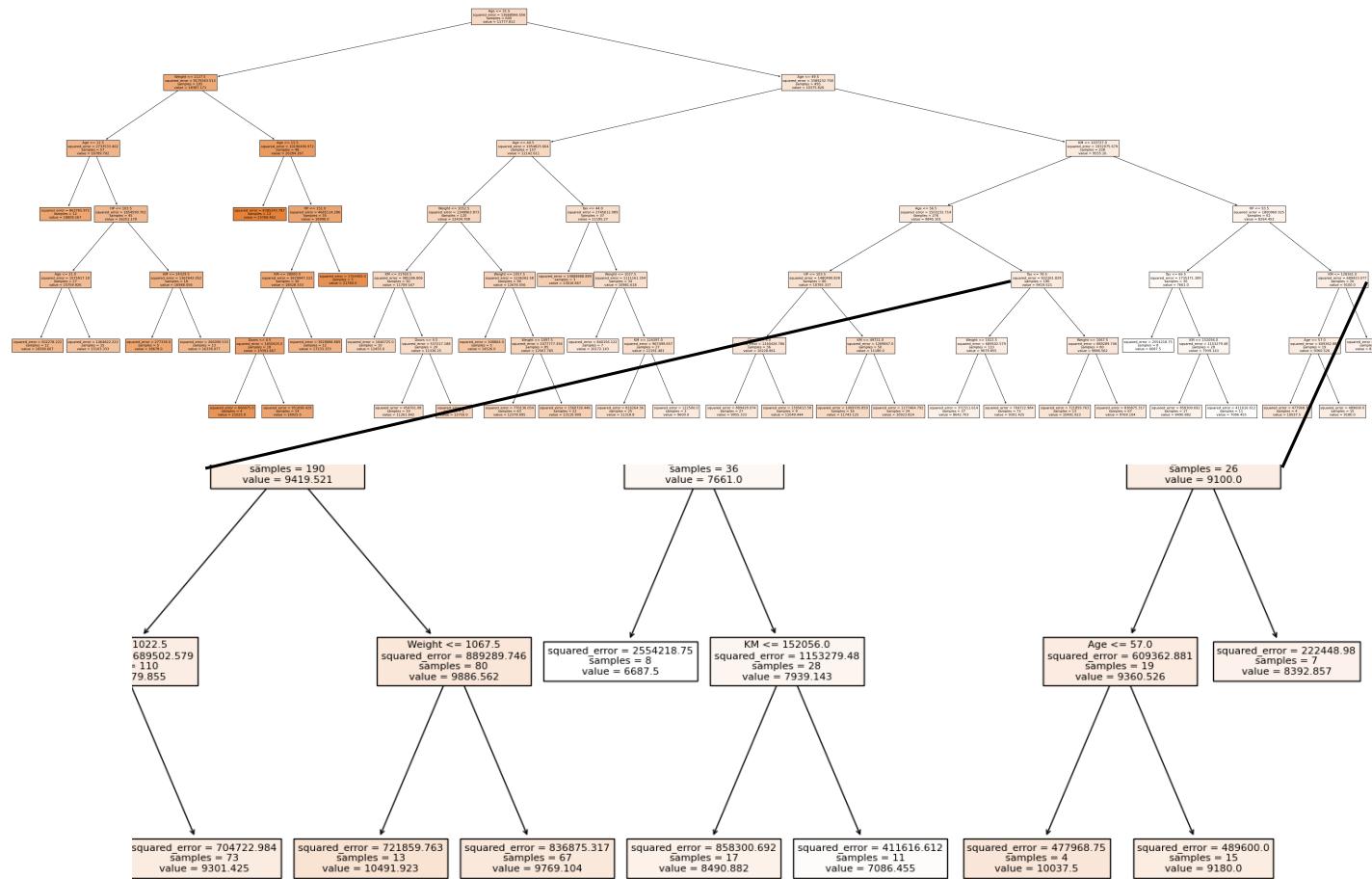
Regression statistics

```
Mean Error (ME) : 0.0000
Root Mean Squared Error (RMSE) : 1058.8202
Mean Absolute Error (MAE) : 767.7203
Mean Percentage Error (MPE) : -0.8074
Mean Absolute Percentage Error (MAPE) : 6.8325
```

Regression statistics

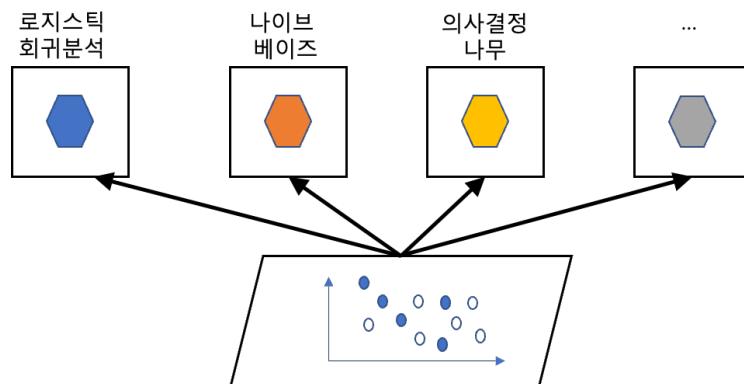
```
Mean Error (ME) : 60.5241
Root Mean Squared Error (RMSE) : 1554.9146
Mean Absolute Error (MAE) : 1026.3487
Mean Percentage Error (MPE) : -1.3082
Mean Absolute Percentage Error (MAPE) : 9.2311
```

회귀 나무 모델 Regression Trees



앙상블 Ensembles

- 여러 지도 학습 기반의 모델들을 하나의 ‘슈퍼모델’로 결합한 방법
- 예측 성능을 향상하기 위해 가중치를 사용해 여러 모델을 결합
- 여러 방안의 앙상블이 가끔 더 정확하게 예측함
- 다양한 알고리즘을 활용하여 모형을 학습하는지 또는 학습데이터를 나누어서 각각 학습하는지에 따라 다양한 앙상블 방법을 구성할 수 있음



대중의 지혜 The Wisdom of Crowds

- The Ox Contest 황소 콘테스트
- Francis Galton은 유명한 통계학자로, 한 지역 축제에서 황소의 무게를 맞히는 대회가 열리는 것을 봄
- 참가자들의 개별 추측치는 매우 다양했지만, 모든 추측의 평균값은 실제 황소의 무게와 1% 이내의 오차만을 보였음



앙상블 방안

- 앙상블(ensemble) 접근법에서는 여러 가지 방법을 처음에 함께 사용하고, 그 결과로 얻어진 예측값이나 분류 결과를 종합함
- 숫자 값을 예측할 때: 여러 방법이 예측한 값들의 평균을 계산
- 클래스를 예측할 때: 여러 방법이 예측한 클래스들 중 다수결로 결정(투표)
- 확률(또는 성향, propensity)을 예측할 때: 각 방법의 예측 확률의 평균을 계산

왜 양상블이 예측력을 높일 수 있는가

- 핵심은 위험 부담(=예측 오차) 줄이기
- 각 모델이 갖는 예측 오차의 평균 0으로 가정 $E(e_{1,i}) = E(e_{2,i}) = 0$
- 예측치의 평균값 $\bar{y}_i = \frac{\hat{y}_{1,i} + \hat{y}_{2,i}}{2}$
- 예측 오차의 기댓값

$$\begin{aligned} E(y_i - \bar{y}_i) &= E\left(y_i - \frac{\hat{y}_{1,i} + \hat{y}_{2,i}}{2}\right) \\ &= E\left(\frac{y_i - \hat{y}_{1,i}}{2} + \frac{y_i - \hat{y}_{2,i}}{2}\right) = E\left(\frac{e_{1,i} + e_{2,i}}{2}\right) = 0 \end{aligned} \tag{13.1}$$

- 양상블의 예측 오차들의 분산

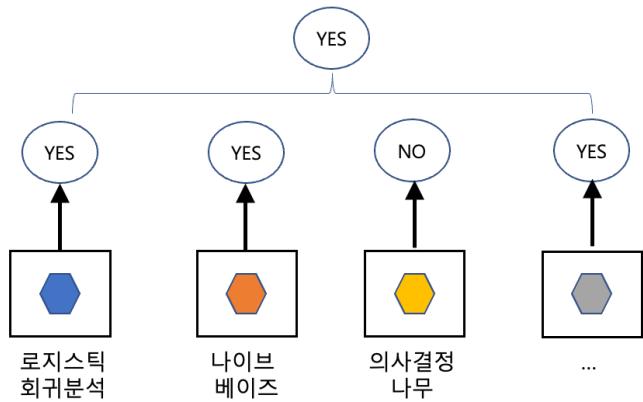
$$\text{Var}\left(\frac{e_{1,i} + e_{2,i}}{2}\right) = \frac{1}{4} \left[(\text{Var}(e_{1,i}) + \text{Var}(e_{2,i})) \right] + \frac{1}{4} \times 2\text{Cov}(e_{1,i}, e_{2,i}) \tag{13.2}$$

$$\text{Var}(aX) = a^2 \text{Var}(X)$$

$$\text{Var}(X + Y) = \text{Var}(X) + \text{Var}(Y) + 2 \text{Cov}(X, Y)$$

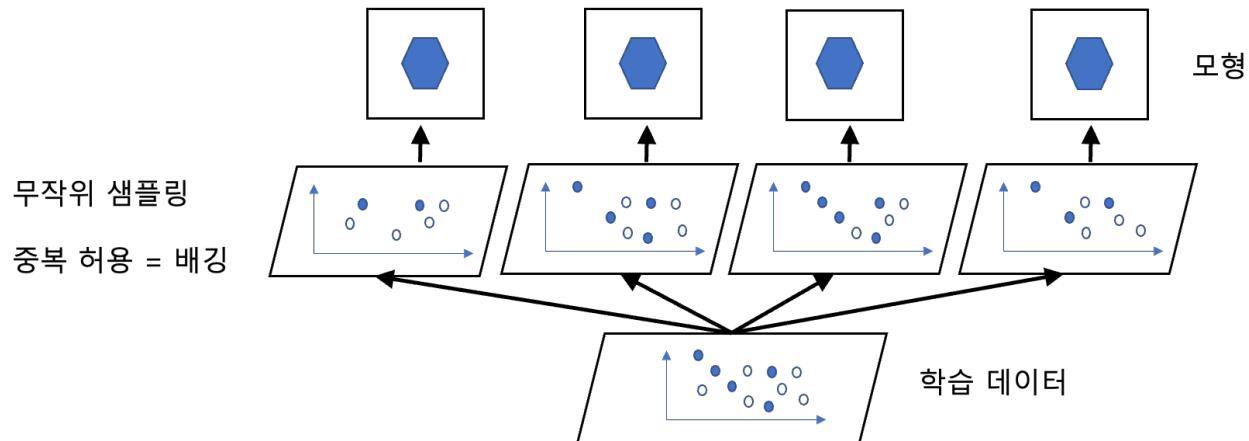
투표 기반 양상들

- 학습 데이터를 가지고 다양한 알고리즘을 통해 분류 모형을 만듦
- 테스트 집합을 각각 학습한 모형에 적용하여 예측 분류 결과를 파악함
- 각 분류기의 예측을 모아서 가장 많이 선택된 분류로 할당함
- 여러 모형의 결과를 모아 다수결로 결정하는 방안을 직접 투표(hard voting)이라고 함
- 간접 투표 (soft voting) : 분류 결과를 Yes/No, 1/0 이 아니라 Yes 혹은 No 일 확률로 구하여 각 분류모형으로부터 나온 확률값의 평균을 취해 최종 결정을 하는 방안



배깅 Bagging (= bootstrap aggregating)

- 투표 기반 양상들은 같은 데이터를 여러 개의 모형이 학습한 후에 나온 결과를 취합하여 결정하는 방식임
- 다른 방안으로는 학습데이터에서 무작위로 데이터를 선택해서 여러 개의 부분집합을 만들고 이를 같은 알고리즘을 사용하여 여러 개의 모형을 학습하여 결과를 취합하는 방안



배깅 Bagging

```
# single tree
defaultTree = DecisionTreeClassifier(random_state=1)
defaultTree.fit(X_train, y_train)
classes = defaultTree.classes_
classificationSummary(y_valid, defaultTree.predict(X_valid),
    class_names=classes)

# bagging
bagging = BaggingClassifier(DecisionTreeClassifier(random_state=1),
n_estimators=100, random_state=1)
bagging.fit(X_train, y_train)
classificationSummary(y_valid, bagging.predict(X_valid),
    class_names=classes)
```

랜덤 포레스트 Random Forest

- 배깅의 예를 들면서 의사결정나무를 활용하였으나 다른 분류 모형을 사용하여도 됨
- 실제로는 배깅에 의사결정나무 기반 기법을 많이 활용함
- 이에 따라 의사결정나무를 배깅하는 방안을 사용해도 되지만 의사결정나무에 맞게 다양한 요소가 고려된 랜덤 포레스트(Random Forrest) 방안 활용

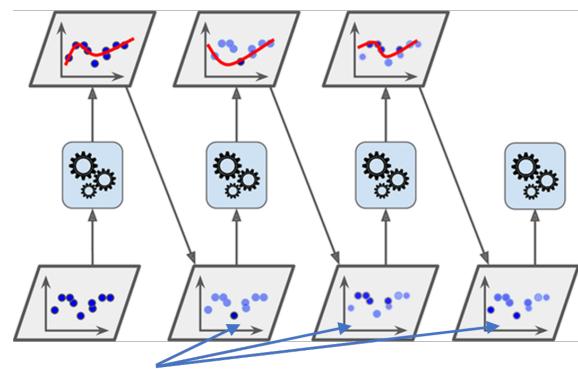
```
from sklearn.ensemble import RandomForestClassifier  
  
rnd_clf = RandomForestClassifier(  
    n_estimators = 500,  
    max_leaf_nodes = 16,  
    max_features = 'auto',  
    max_samples=0.5,  
    bootstrap = True,  
    n_jobs=-1  
)
```



```
bag_clf = BaggingClassifier(  
    DecisionTreeClassifier(max_leaf_nodes = 16),  
    n_estimators = 500,  
    max_features = 'auto',  
    max_samples=0.5,  
    bootstrap = True,  
    n_jobs = -1  
)
```

부스팅

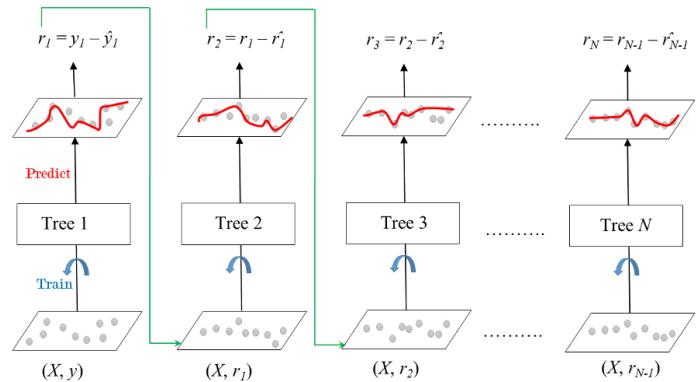
- Boosting은 여러 개의 학습모형을 연결하여 앞의 모형을 보완해 가면서 뒤로 갈수록 더욱 강한 모형을 만드는 양상을 방안임
- 여러 가지 Boosting 방안이 있지만 여기서는 기본적인 에이다 부스트(Adaptive Boosting, AdaBoost)와 그래디언트 부스팅(Gradient Boosting)에 대해서 알아보자.
- 에이다 부스트(AdaBoost)
 - 이전 모형을 보완하여 새로운 모형을 만드는 방법의 기본은 이전 모형이 잘 맞추지 못한 데이터에 대해서 가중치를 더 높이는 것임
 - 새로운 모형은 가중치가 더 높은 데이터를 잘 구분하려고 할 것이기에 어려운 데이터를 점점 더 잘 분류할 수 있게 되며 이것이 AdaBoost의 기본적인 개념



(출처: 핸즈온 머신러닝, 2판, 7장, 그림 7-7)

그래디언트 부스팅 (Gradient Boosting, GBM)

- 그래디언트 부스팅은 에이다 부스트(AdaBoost)처럼 반복마다 데이터의 가중치를 수정하는 대신 이전 모형이 만든 잔여 오차(residual error)에 새로운 모형을 학습시킴
- 옆그림처럼 첫 모형의 오차를 가지고 다음 모형이 학습하여 예측하고, 여기서 발생한 오차를 다음 모형이 학습하게 됨
- 이렇게 지속적으로 다음 모형이 앞 모형의 오차를 학습하여 예측함
- 분류 문제와 수치 예측 문제에 모두 적용 가능



그래디언트 부스팅

- 회귀 의사결정나무로 수치를 예측 하는 경우를 가지고 그래디언트 부스트의 개념을 설명해 보자

```
tree_reg1 = DecisionTreeRegressor(max_depth = 2)  
tree_reg1.fit(X, y)
```

```
y2 = y - tree_reg1.predict(X)  
tree_reg2 = DecisionTreeRegressor(max_depth = 2)  
tree_reg2.fit(X, y2)
```

```
y3 = y2 - tree_reg2.predict(X)  
tree_reg3 = DecisionTreeRegressor(max_depth = 2)  
tree_reg3.fit(X, y3)
```

```
y_pred = sum(tree.predict(X) for tree in (tree_reg1, tree_reg2, tree_reg3))
```

```
rmse = math.sqrt(mean_squared_error(y_pred, y))  
rmse
```

부스팅

```
# single tree
defaultTree = DecisionTreeClassifier(random_state=1)
defaultTree.fit(X_train, y_train)
classes = defaultTree.classes_
classificationSummary(y_valid, defaultTree.predict(X_valid),
    class_names=classes)

# AdaBoost
boost = AdaBoostClassifier(DecisionTreeClassifier(random_state=1),
    n_estimators=100,
    random_state=1)
boost.fit(X_train, y_train)
classificationSummary(y_valid, boost.predict(X_valid), class_names=classes)

# GradientBoosting
boost = GradientBoostingClassifier(max_depth=5,
    n_estimators = 20,
    learning_rate = 1.0)
boost.fit(train_X, train_y)
classificationSummary(valid_y, boost.predict(valid_X))
```

의사결정 나무의 장점

- 쉬운 사용과 이해: 단일 트리는 해석 및 구현이 쉬운 규칙을 생성
- 변수 선택 및 축소가 자동으로 이루어짐
- 통계적 모델의 가정이 필요하지 않음
- 누락된 데이터를 광범위하게 처리하지 않고도 작업 가능
- 단점

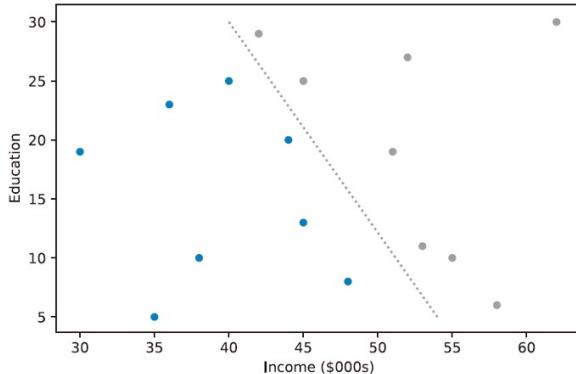


그림 9-16 2개의 예측 변수로 묘사한 이진 클래스 산점도: 가장 좋은 분할은 대각선을 이용해 얻을 수 있는데, 이는 분류 트리로 구현할 수 없다.