머신러닝
(Machine Learning)

LECTURE XIII: 지도 학습 7
(Supervised Learning)

Dai-Gyoung Kim

Department of Applied Mathematics

Hanyang University ERICA

지도 학습
(Supervised Learning)

Contents

- 분류와 회귀
- 일반화, 과대적합, 과소적합
- 지도 학습 알고리즘
  - k-최근접 이웃
  - 선형모델
  - 나이브 베이즈 모델
  - 결정트리
  - 결정트리의 앙상블
  - 커널 서포트 벡터 머신
  - 신경망, 딥러닝
- 분류 예측의 불확실성 추정

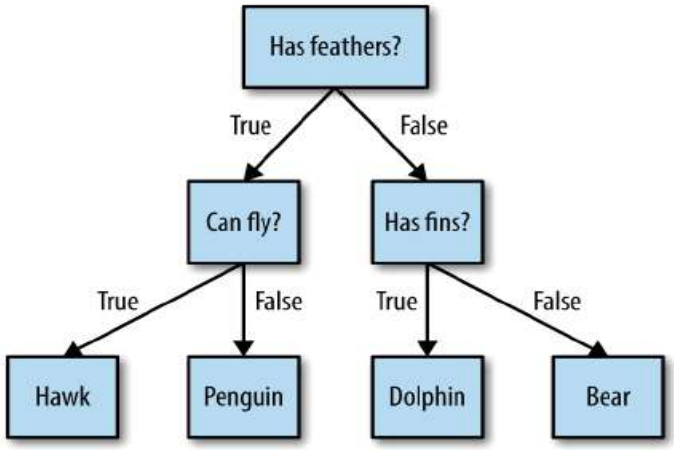
▶ 결정 트리

결정 트리는 SVM 같이 분류와 회귀에 널리 쓰이는 효과적인 모델이며 랜덤 포레스트와 그래디언 부스팅의 앙상블 모델의 기본 요소임.

결정 트리 분류

- 결정 트리(Decision Tree)는 결정에 다다르기 위해 예/아니오 질문을 이어나가면서 학습함.
- 다음은 간단한 예/아니오 질문으로 곰, 매, 펭귄, 돌고래라는 네 가지 동물을 구분하는 모델임 (스무고개 놀이)

```
mglearn.plots.plot_animal_tree()
```

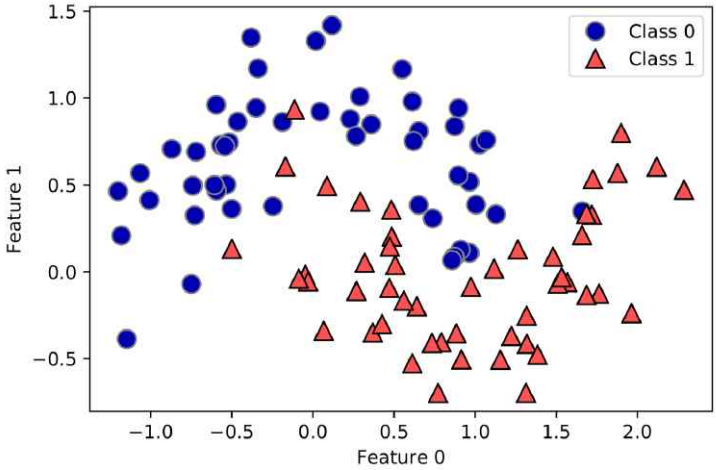


- 노드(node), 에지(edge)
  - ✓ 트리의 **노드**(node)는 질문이나 정답을 담음.
  - ✓ 깊이가 0인 맨 꼭대기 노드를 **루트 노드**(root node)라 하며 다음 분기의 노드를 **자식 노드** (child node)라 함.
  - ✓ 첫 번째 자식 노드의 깊이는 1이고 그 다음 자식 노드의 깊이는 2, 등등.
  - ✓ **리프 노드**(leaf node)는 자식 노드를 가지지 않는 노드임 (마지막 노드).
  - ✓ 타깃 하나로만 이루어진 리프 노드를 **순수 노드**(pure node)라 함.
  - ✓ 에지(edge)는 질문의 답과 다음 질문은 연결함.

- 결정 트리를 학습한다는 것은 정답에 가장 빨리 도달하는 질문 목록을 학습하는 것임.
  - ✓ 머신러닝에서 이런 질문을 테스트라고 함.

❖ 결정 트리 만들기

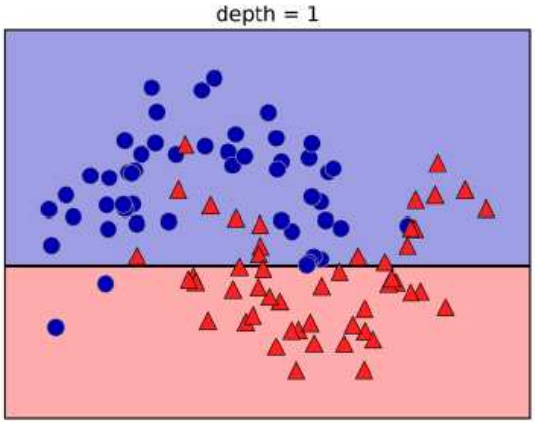
- 다음 그림은 2차원 데이터셋의 산점도이며, 이 데이터셋 two_moons은 각 클래스에 데이터 포인트가 50개씩 있고 반달 두 개가 포개진 모양임.

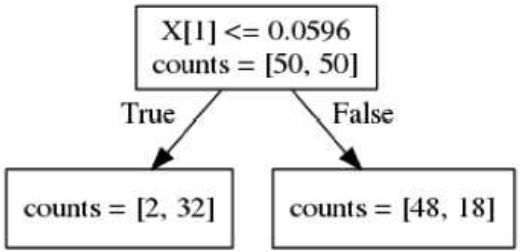


- 위의 two_moons 데이터셋은 연속된 특성으로 구성되어있음.
- 연속적인 데이터에 적용되는 테스트는 특성값의 대소 관계로 설정함.
  - ✓ 범주형 데이터에 적용되는 테스트는 예/아니오의 형태의 특성으로 구성됨.

[트리 만들기]

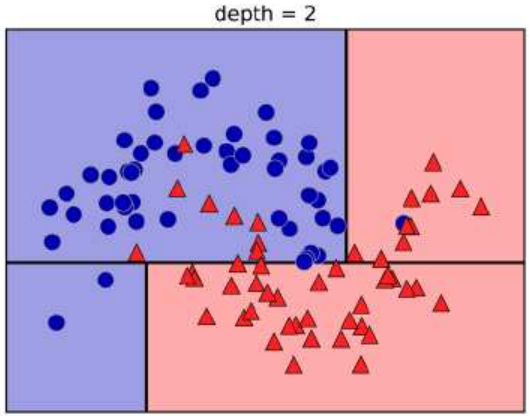
- 트리를 만들 때, 알고리즘은 가능한 모든 테스트에서 타깃값에 대해 가장 많은 정보를 가진 것을 선택함.
- 다음 그림은 첫 번째로 선택된 테스트를 보여줌.

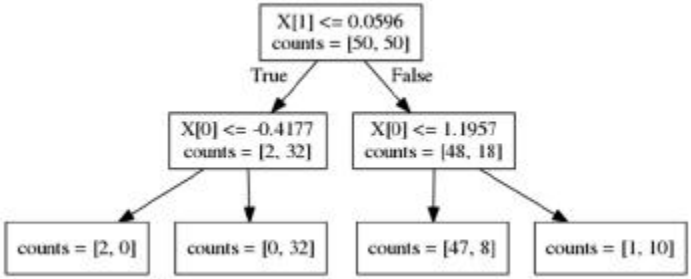




✓ x[1] 값을 기준으로 한 수평축으로 클래스 0에 속한 포인트와 클래스 1에 속한 포인트를 나누고 있음. 위의 첫 번째 분류기는 두 클래스를 완벽하게 분류하고 있지 않음.

- 다음 그림은 다음 단계로 가장 많은 정보를 올바로 담을 수 있도록 x[0] 값을 기준으로 테스트한 것을 보여줌.

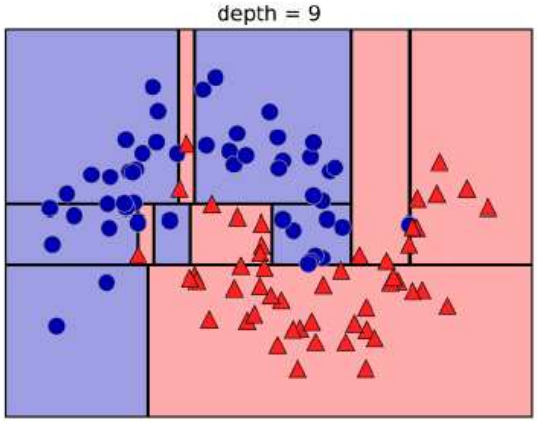


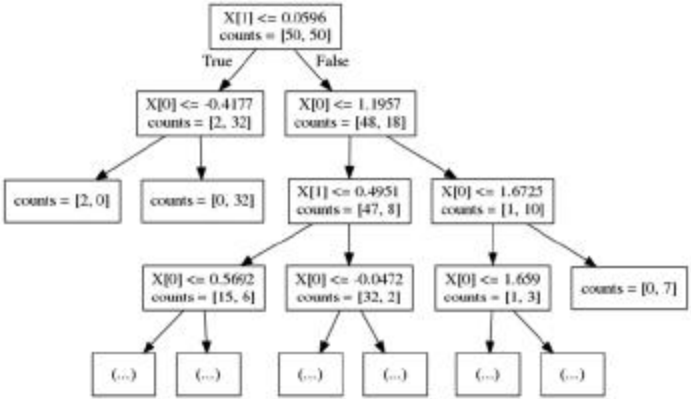


✓ 각 분기에서 x[0] 값을 기준으로 한 수직축으로 클래스 0에 속한 포인트와 클래스 1에 속한 포인트를 나누고 있음.

- 반복된 프로세스는 각 노드가 테스트 하나씩을 가진 이진 결정트리를 만듦 (계층적 구조 알고리즘).

- 데이터의 분할은 각 분할된 영역이 (결정 트리의 리프) 한 개의 타깃값 (순수 노드)을 가질 때까지 반복됨.
- 다음은 깊이 9인 결정 트리의 일부와 이 트리가 만든 결정 경계를 보여줌.





[결정 트리의 예측과 특성]

- 새로운 데이터 포인트의 예측은 주어진 데이터 포인트가 특성을 분할한 영역의 타깃값 중 다수(순수노드일 경우 하나)인 것을 예측결과로 채택함.
- 결정 트리의 여러 장점 중에 하나는 데이터 전처리가 거의 필요하지 않다는 것임. 특히 특성의 스케일을 맞출 필요가 없다는 것임.
- 결정 트리는 매우 직관적이고 결정 방식을 이해하기 쉬움. 이런 모델을 **화이트박스** (white box) 모델이라고 함.
- 랜덤 포레스트나 신경망은 **블랙박스**(black box) 모델임이며, 이런 종류의 모델 예측은 결과를 규명하기 어려움.

결정 트리 회귀

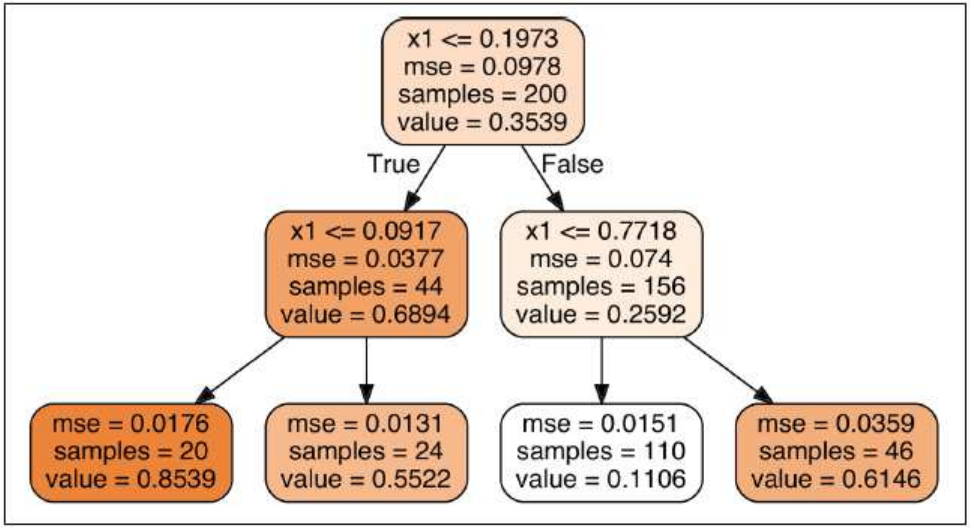
- 결정 트리는 회귀 문제에도 적용될 수 있음.
- 분류 트리와 매우 비슷하지만, 주요한 차이는 각 노드에서 클래스를 예측하는 대신 찾은 리프노드의 훈련 데이터 평균값을 예측값으로 함.

- 사이킷런(scikit-learn)의 'DecisionTreeRegressor'를 사용하여 노이즈가 섞인 2차 함수 형태의 데이터셋에서 'max_depth=2' 설정으로 회귀 트리를 만들 수 있음.

```
# Quadratic training set + noise
np.random.seed(42)
m = 200
X = np.random.rand(m, 1)
y = 4 * (X - 0.5) ** 2
y = y + np.random.randn(m, 1)/10
```

```
from sklearn.tree import DecisionTreeRegressor

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

- 위의 코드로 만들어진 트리는 다음과 같음.

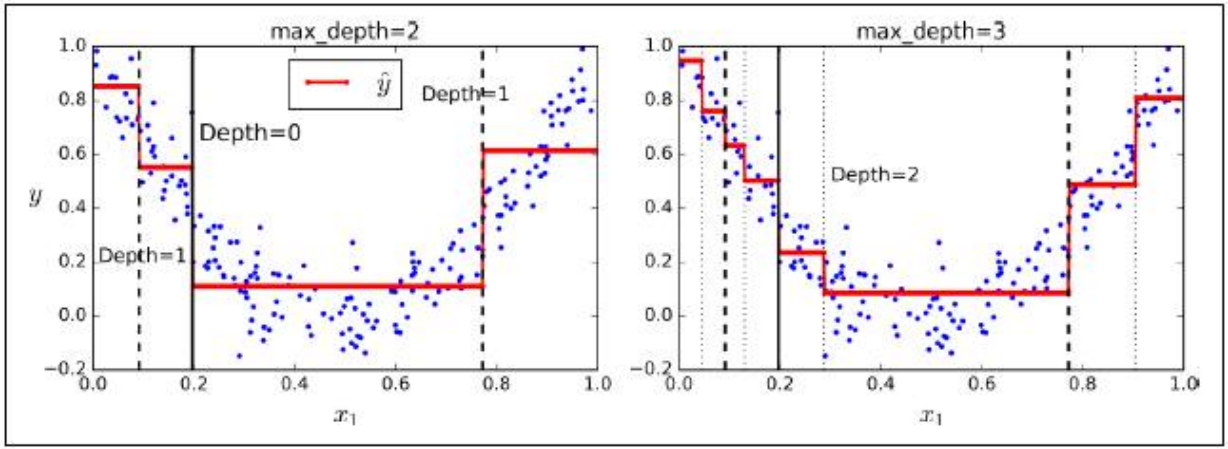


✓ 예를 들어 $x_1 = 0.6$인 샘플의 클래스를 예측한다고 하면, 루트 노드부터 시작하여 트리를 순회하면 결국 value=0.1106인 리프노드에 도달하게 됨.

✓ 이 리프 노드에 있는 110개 훈련 샘플의 평균 타깃값(0.1106)이 예측값이 됨

✓ 이 예측값과 해당 노드의 110개 샘플에 대한 타깃값의 평균제곱오차는 0.0151.

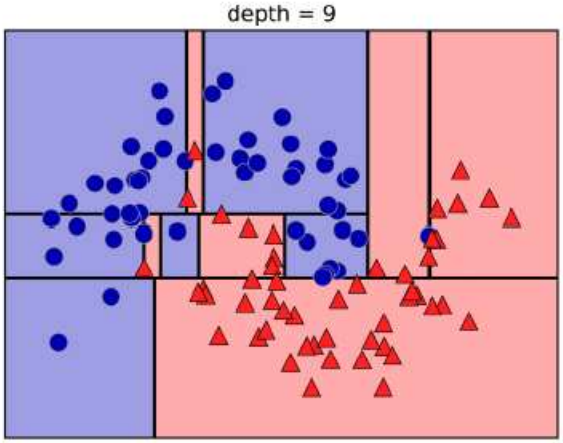
- 위 모델의 예측은 다음 그림과 같이 됨.



✓ 이 알고리즘은 예측값과 가능한 한 많은 샘플이 가까이 있도록 영역을 분할함.

❖ 결정 트리의 복잡도 제어하기

- 결정 트리는 훈련 데이터에 대한 제약사항은 거의 없으나, 제한을 두지 않으면 트리가 훈련 데이터에 아주 가깝게 맞추려고 해서 대부분 과대적합되기 쉬움.
- 결정 트리의 모델은 훈련되기 전에 파라미터 수가 결정되지 않기 때문에 **비모수 모델**(nonparametric model)이라 함. 따라서 결정 트리 모델의 구조는 데이터에 맞춰져서 미리 고정되지 않고 자유로움.
- 특히 결정 트리 모델의 과대적합으로 인한 결정 경계는 각 데이터 포인트에 맞추어져 이상치 하나에도에 너무 민감하게 반응하여 복잡한 모양을 띠고 있음.



- 훈련 데이터에 대한 과대적합을 줄이기 위해 학습할 때 결정 트리의 자유도를 제한해야 함.
- 과대적합을 막는 전략은 크게 두 가지임:

  1) **사전 가지치기**(pre-pruning): 트리 생성을 일찍 중단하는 전략.
     - ✓ 트리의 최대 깊이 제한하기.
     - ✓ 리프의 최대 개수 제한하기.
     - ✓ 노드가 분할하기 위한 포인트의 최소 개수 지정하기.
  2) **사후 가지치기**(post-pruning): 트리를 만든 후 데이터 포인트가 적은 노드를 삭제하거나 병합하는 전략.

- 사이킷런(scikit-learn)에서 결정 트리는 'DecisionTreeRegressor'와 'DecisionTreeClassifier'로 구현함.
  - ✓ 사이킷런은 사전 가지치기만 지원함.

- 사이킷런에서 'max_depth' 매개변수로 최대 깊이를 제어함.
- 'DecisionTreeClassifier'에서 결정 트리의 형태를 제어하는 매개 변수는 다음과 같음.
  1) **min_samples_split**: 분할되기 위해 노드가 가져야 하는 최소 샘플 수
  2) **min_samples_leaf**: 리프 노드가 가지고 있어야 할 최소 샘플 수
  3) **min_weight_fraction_leaf**: min_samples_leaf와 같지만 가중치가 부여된 전체 샘플 수에서의 비율
  4) **max_leaf_nodes**: 리프 노드의 최대 수
  5) **max_features**: 각 노드에서 분할에 사용할 특성의 최대 수

- 다음은 유방암 데이터셋에 이용하여 결정 트리 모델을 적용하는 코드임.

```
from sklearn.tree import DecisionTreeClassifier


cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, stratify=cancer.target, random_state=42)
```

```
tree = DecisionTreeClassifier(random_state=0)
tree.fit(X_train, y_train)

print("Accuracy on training set: {:.3f}".format(tree.score(X_train, y_train)))
print("Accuracy on test set: {:.3f}".format(tree.score(X_test, y_test)))
```

```
Accuracy on training set: 1.000
Accuracy on test set: 0.937
```

✓ 'DecisionTreeClassifier'의 기본값 설정은 완전한 트리 모델을 만듦.

✓ 모든 리프 노드가 순수 노드이므로 훈련 세트의 정확도는 100%임.

✓ 테스트 세트의 정확도는 93.7%임 (이전의 로지스틱 회귀는 95%임).

- 사전 가지치기를 위해 "max_depth=4" 옵션을 가하여 과대적합을 줄일 수 있음 (이 경우 연속된 질문을 최대 4개로 제한함).

```
tree = DecisionTreeClassifier(max_depth=4, random_state=0)
tree.fit(X_train, y_train))
```

```
print("Accuracy on training set: {:.3f}".format(tree.score(X_train, y_train)))
print("Accuracy on test set: {:.3f}".format(tree.score(X_test, y_test)))
```

```
Accuracy on training set: 0.988
Accuracy on test set: 0.951
```

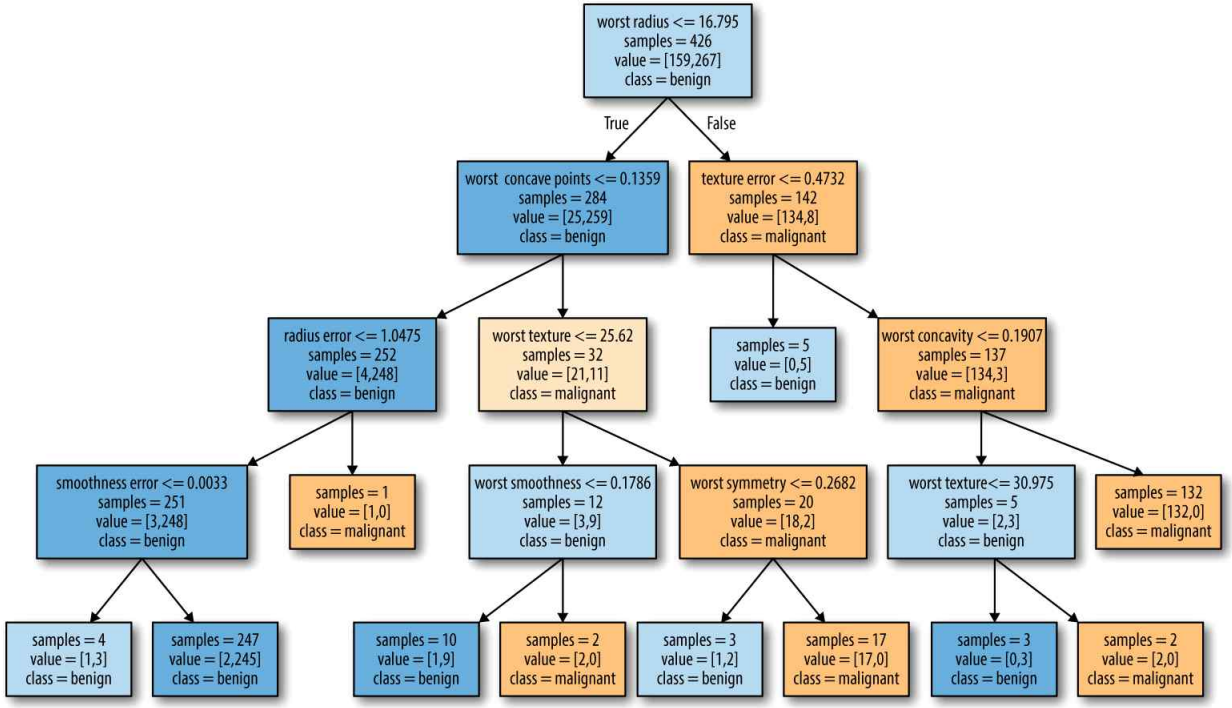
✓ 이 경우 훈련 세트의 정확도는 떨어지지만, 테스트 세트의 성능은 향상됨.

❖ 결정 트리 분석

- 트리 모듈의 'export_graphviz()' 함수를 사용해 트리를 시각화할 수 있음.
- 이 함수는 그래프 저장용 텍스트 파일 포맷인 .dot 파일로 만들고, 이 파일을 불러들여 graphviz 모듈을 사용해 결정 트리를 시각화함.

```
from sklearn.tree import export_graphviz

export_graphviz(tree, out_file="tree.dot", class_names=["malignant", "benign"],
                feature_names=cancer.feature_names, impurity=False, filled=True)

import graphviz
```

```
with open("tree.dot") as f:
    dot_graph = f.read()
display(graphviz.Source(dot_graph))
```



- ✓ 위 트리는 유방암 데이터셋으로 만든 것으로 깊이가 4인 결정 트리임.
- ✓ 각 노드에 적힌 'samples'는 각 노드에 있는 샘플의 수를 나타내며 'value'는 클래스당 샘플 수를 제공함.
- ✓ 루트 노드의 오른쪽 가지를 따라가면(worst radius>16.795) 악성 샘플이 134개, 양성 샘플이 8개인 노드를 만듦.
- ✓ 이 방향의 나머지 트리는 이 8개의 양성 샘플을 더 세부적으로 분리함.
- ✓ 첫 노드에서 오른쪽으로 분리된 142개 샘플 중 거의 대부분(132개)이 가장 오른쪽 노드로 분류됨.
- ✓ 루트 노드에서 왼쪽으로 간 데이터(worst radius<=16.795)는 악성 샘플이 25개이고 양성 샘플이 259개임. 대부분 양성 샘플은 왼쪽에서 두 번째 노드에 할당되고 나머지 리프 노드 대부분은 매우 적은 양의 샘플만 가지고 있음.

❖ 트리 특성의 중요도

- 깊이가 깊을 때 전체 트리를 살펴보는 것은 사실상 어려움.
- 전체 트리를 살펴보는 대신 트리가 어떻게 작동하는지 요약하는 속성들을 사용할 수 있음. 가장 널리 사용되는 속성은 트리를 만드는 결정에 각 특성이 얼마나 중요한지를 평가하는 **특성 중요도**(feature importance)임.
  - ✓ 특성 중요도는 0과 1사이의 값으로 표현되며, 각 특성에 대해 0은 전혀 사용되지 않았다는 뜻이며, 1은 완벽하게 타깃 클래스를 예측했다는 뜻임. 특성 중요도의 전체 합은 1임.

```
print("Feature importances:\n{}".format(tree.feature_importances_))
```

```
Feature importances:
[ 0.      0.      0.      0.      0.      0.      0.      0.      0.      0.      0.01
  0.048   0.      0.      0.002   0.      0.      0.      0.      0.      0.727   0.046
  0.      0.      0.014   0.      0.018   0.122   0.012   0. ]
```

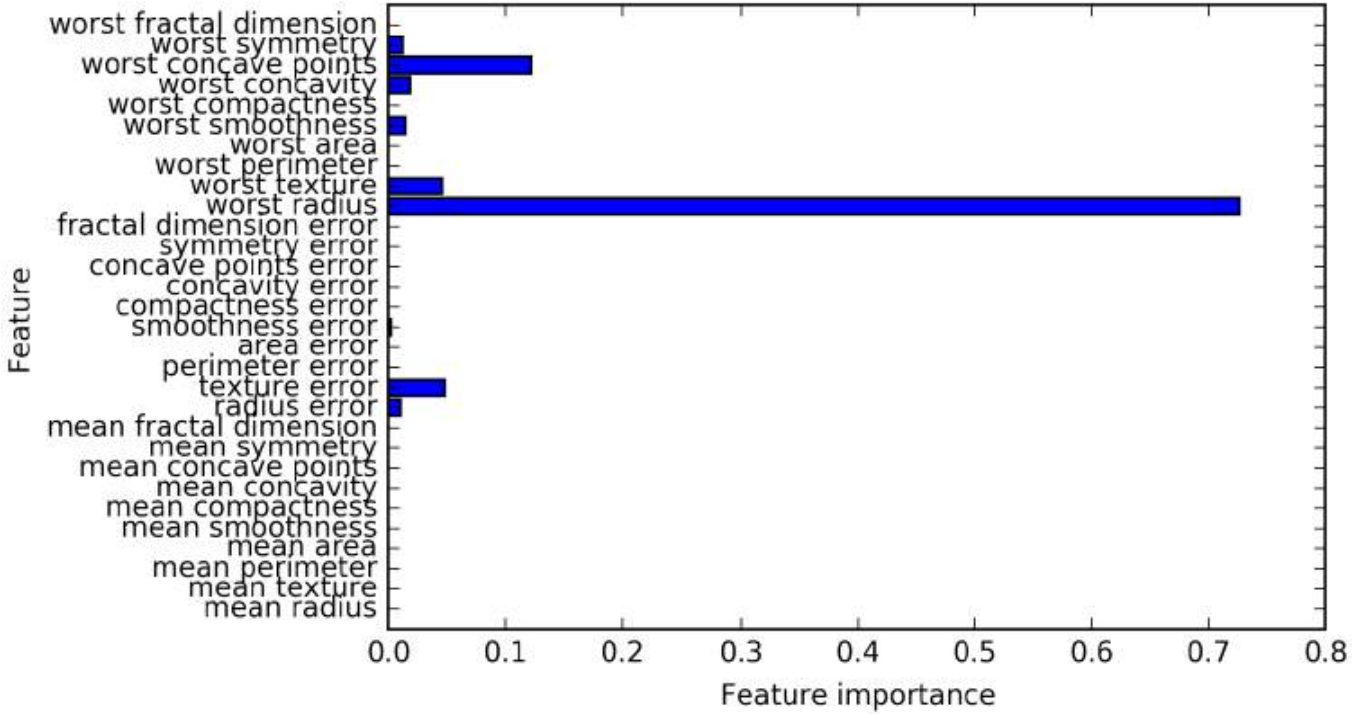
- 다음은 유방암 데이터로 학습시킨 결정 트리의 특성 중요도를 시각화하는 코드임.

```
def plot_feature_importances_cancer(model):
    n_features = cancer.data.shape[1]
    plt.barh(np.arange(n_features), model.feature_importances_, align='center')
    plt.yticks(np.arange(n_features), cancer.feature_names)
    plt.xlabel("Feature importance")
    plt.ylabel("Feature")
    plt.ylim(-1, n_features)
plot_feature_importances_cancer(tree)
```

  - ✓ 첫 번째 노드의 특성 “worst radius”가 가장 중요한 특성으로 나타남.
  - ✓ 어떤 특성의 ‘feature_importance_’ 값이 낮다고 해서 그 특성이 유용하지 않다는 뜻이 아님. 단지 이 트리가 그 특성을 선택하지 않았을 뿐이며 다른 특성이 동일한 정보를 지니고 있을 가능성이 많음.

- 선형 모델과 달리, 결정 트리의 특성의 중요도는 항상 양수이며 특성이 어떤 클래스를 지지하는지 알 수 없음.

✓ 특성 중요도의 값은 “worst radius”가 가장 중요하다고 알려주지만 큰 반지름이 양성을 의미하는지 악성을 의미하는지는 알 수 없음.

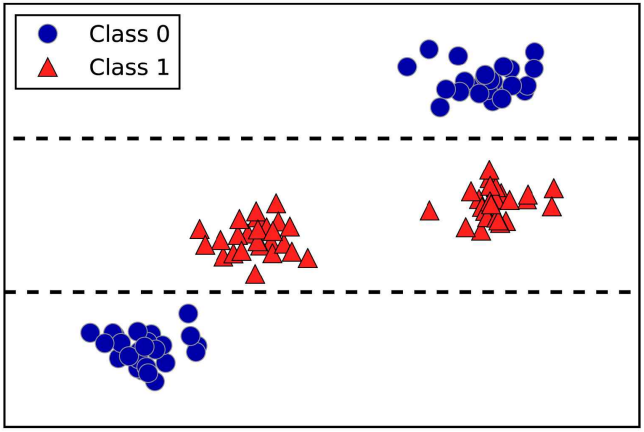


[특성과 클래스 사이의 복잡한 관계가 있는 경우]

- 특성과 클래스 레이블사이에 비례관계가 성립하지 않은 복합적인 관계가 있을 수 있음.

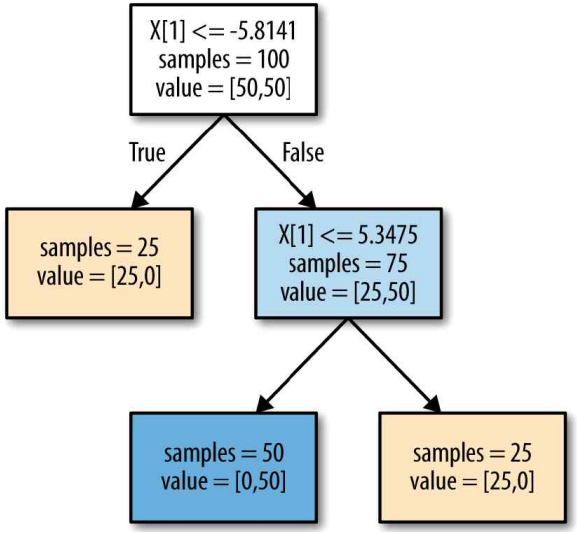
```
tree = mglearn.plots.plot_tree_not_monotone()
display(tree)
```

```
Feature importances: [ 0.  1.]
```



- ✓ 위 예에서는 두 개의 특성과 두 개의 클래스를 가진 데이터셋을 보여줌.
- ✓ 결정 트리에서 x[1]에 있는 정보만 사용되었고, 특성 x[1]과 출력 클래스는 단순하게 비례 또는 반비례의 관계가 없음.

- 다음은 위 데이터셋으로 학습한 결정 트리를 보여줌.

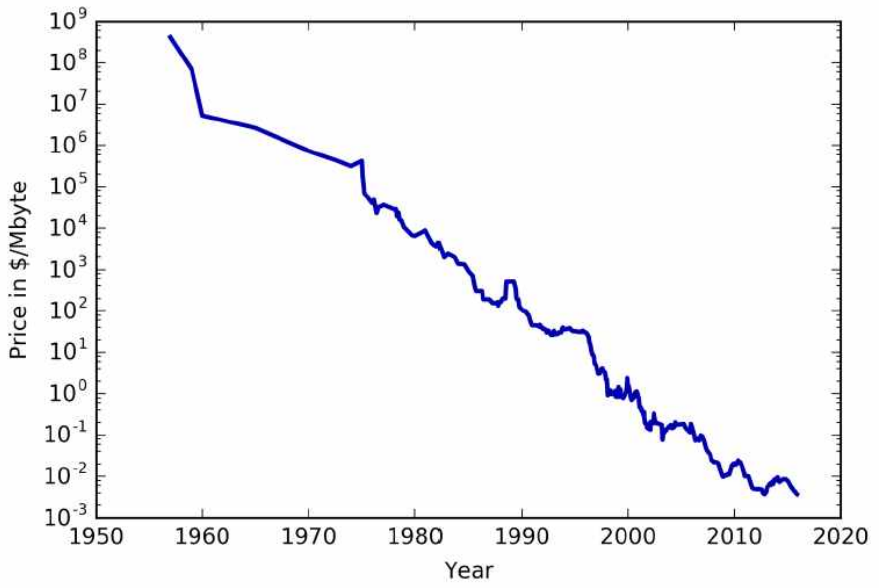


[트리 기반의 회귀]

- 회귀를 위한 트리 기반 모델(DecisionRegressor)의 사용법과 분석도 분류 트리와 매우 비슷함.
- 회귀를 위한 회귀 트리 기반 모델은 일반 회귀 모델과 달리 외삽(extrapolation) 기능이 없음 (즉, 훈련 데이터 밖의 포인트에 대해 예측을 할 수 없음).

- 다음은 트리 기반의 회귀의 특성을 살펴보기 위한 데이터셋으로서 컴퓨터 메모리 가격 동향의 데이터임.

```
import os
ram_prices = pd.read_csv(os.path.join(mglearn.datasets.DATA_PATH, "ram_price.csv"))


plt.semilogy(ram_prices.date, ram_prices.price)
plt.xlabel("Year")
plt.ylabel("Price in $/Mbyte")
```



✓ y 축은 해당 년도의 램(RAM) 1메가바이트당 가격임 (y축은 로그 스케일).

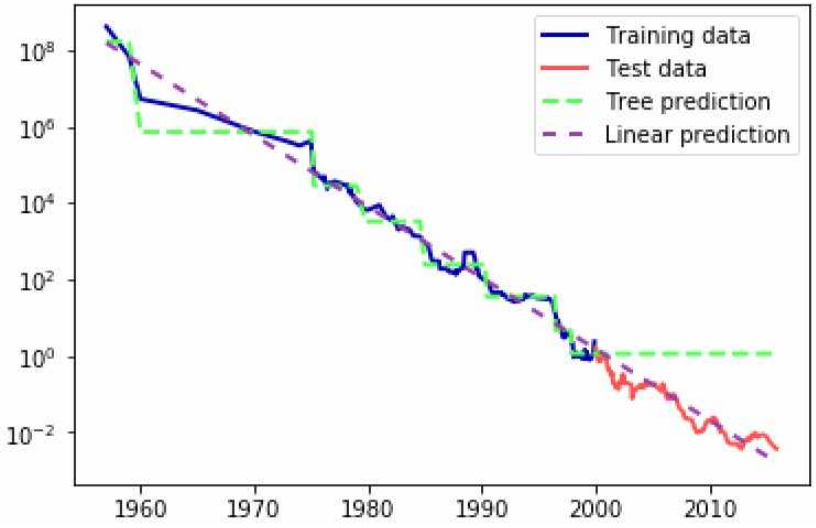
- 날짜 특성 하나만으로 2000년 전까지의 데이터로부터 2000년 후의 가격을 예측해 보고자 함.

- 간단한 두 모델 'DecisionRegressor'와 'LinearRegression'을 비교하고자 함.
- 모델을 훈련시키고 예측을 수행한 다음, 로그 스케일을 되돌려야 함.

```
from sklearn.tree import DecisionTreeRegressor
# use historical data to forecast prices after the year 2000
data_train = ram_prices[ram_prices.date < 2000]
data_test = ram_prices[ram_prices.date >= 2000]

# predict prices based on date
X_train = data_train.date[:, np.newaxis]
# we use a log-transform to get a simpler relationship of data to target
y_train = np.log(data_train.price)

tree = DecisionTreeRegressor(max_depth=3).fit(X_train, y_train)
linear_reg = LinearRegression().fit(X_train, y_train)
```

```
# predict on all data
X_all = ram_prices.date[:, np.newaxis]

pred_tree = tree.predict(X_all)
pred_lr = linear_reg.predict(X_all)

# undo log-transform
price_tree = np.exp(pred_tree)
price_lr = np.exp(pred_lr)
```

```
plt.semilogy(data_train.date, data_train.price, label="Training data")
plt.semilogy(data_test.date, data_test.price, label="Test data")
plt.semilogy(ram_prices.date, price_tree, label="Tree prediction")
plt.semilogy(ram_prices.date, price_lr, label="Linear prediction")
plt.legend()
```



✓ 선형 모델은 직선으로 훈련 데이터뿐만아니라 테스트 데이터를 효과적으로 근사함. 반면 트리 모델은 훈련 데이터를 잘 근사하지만 새로운 테스트 데이터를 예측할 능력이 없음 (단순히 마지막 포인트를 이용하여 예측함).

- 모든 트리 기반의 모델은 새로운 데이터에 대해 좋은 예측을 할 수 없음. 특히 시계열 같은 데이터에는 잘 맞지 않음.

❖ 장단점과 매개변수

- 과대적합을 막는 데 사용되는 규제 매개변수는 사전 가지치기임.
  - ✓ max_depth: 트리의 최대 깊이 지정
  - ✓ max_leaf_nodes: 리프 노드의 최대 개수 지정
  - ✓ min_samples_leaf: 리프 노드가 가지고 있어야 할 최소 샘플 개수 지정
  - ✓ min_samples_split: 노드가 분기할 수 있는 최소 샘플 개수 지정

[결정 트리의 장점]

1) 만들어진 모델을 쉽게 시각화할 수 있어 비전문가도 이해하기 쉬움.

2) 데이터 특성간의 스케일 차이에 영향을 받지 않음.
   - ✓ 각 특성이 개별적으로 처리되어 데이터를 분할하는데 데이터 스케일에 영향을 받지 않으므로 결정 트리에서는 특성의 정규화나 표준화 같은 전처리 과정이 필요 없음.
   - ✓ 특히 특성의 스케일이 서로 다르거나 이진 특성과 연속적인 특성이 혼합되어 있을 때도 잘 작동함.

[결정 트리의 단점]

- 결정 트리의 주요 단점은 사전 가지치기를 사용함에도 불구하고 과대적합되는 경향이 있어 일반화 성능이 좋지 않을 수 있음.
- 이를 극복하기 위해 단일 결정 트리대신에 앙상블 방법을 도입함.