Decision Tree

Machine Learning Study 2022

Kim Suhyun

Jan 10 2022

Department of Statistics
Chung-Ang University

Contents

Theoretical Part

Classification

- Logistic Regression, Tree, Random Forest, SVM

Regression

- Linear Regression, Tree, Random Forest

Ensemble

- Bagging : Random Forest
- Boosting : Gradient boosting, Xgboost, LightGBM
- 이때 앙상블은 서로 다른 또는 같은 알고리즘을 결합하는 것이다.
  다시말해 매우 많은 여러개의 약한 학습기를 결합해서 확률적인 보완을 수행하고
  오류가 발생하는 부분에 대한 가중치를 update 해나가며 예측 성능을 높이는 것이다.

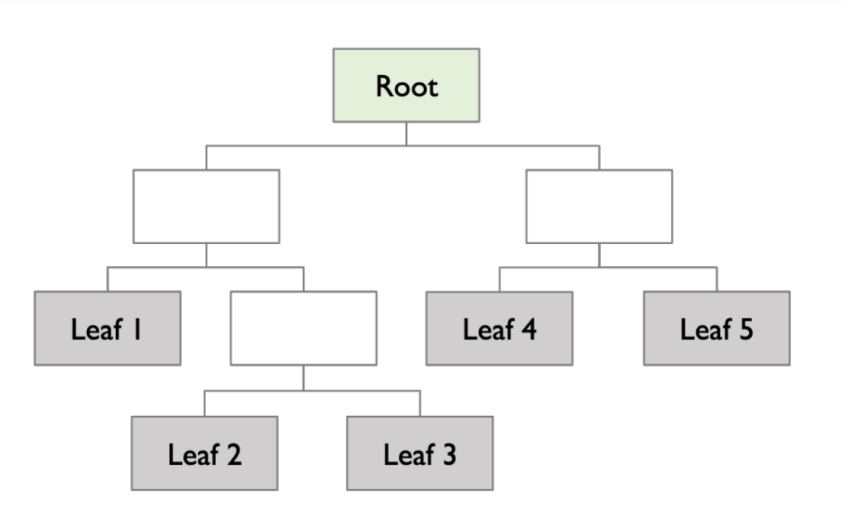

Figure 1: Root & Leaf nodes

- Parent node : node before split
- Child node : node after split
- Split criterion : a certain variable value used for split a node
- Root node : node that only has child nodes but no parent node
- Leaf nodes : nodes that only have a parent node but no child nodes

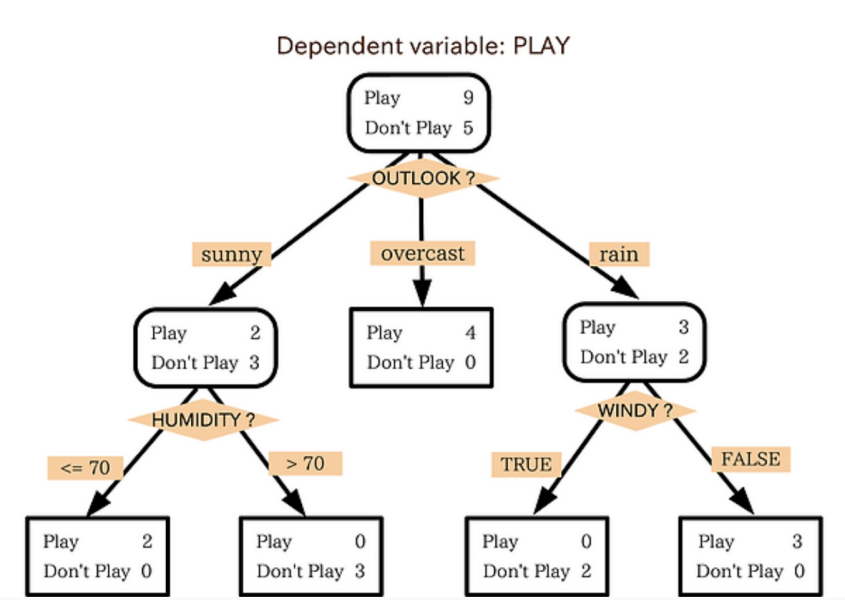

Figure 2: Desision Tree, set of rules

For example, Rain - not Windy $\rightarrow$ Play !

CART

- binary split
- Gini Impurity(Classification) & SSE(Regression)

C4.5 & C5.0

- Multi split(classification) & Binary split(Regression)
- Entropy Inpurity

CHAID

- Multi split
- χ^2 (classification) & ANOVA F (regression)

CART, Classification And Regression Tree

Generate a set of rules by recursively partitioning the entire datasets to increase the purity of the partitioned area (Breiman, 1984)

- **Recursive Partitioning**
- **Pruning**

Goodness of Split : Impurity of a node

→ **Impurity Measure**

- Impurity of Classification Tree is measured by **Gini index** & **Deviance**

$$I(A) = \sum_{i=1}^{d}(R_i(1 - \sum_{k=1}^{m} p_{ik}^2))$$

, where R_i = proportion of cases in ractangle R_i among the training data.

$$D_i = -2\sum_{k} n_{ik} log(p_{ik})$$

, where p_{ik} = probability of class k in node I, for i : node index, k : class index

- Impurity of Regression: (Variance) **SSE**

$$SSE = \sum_{i=1}^{n}(y_i - \hat{y})^2$$

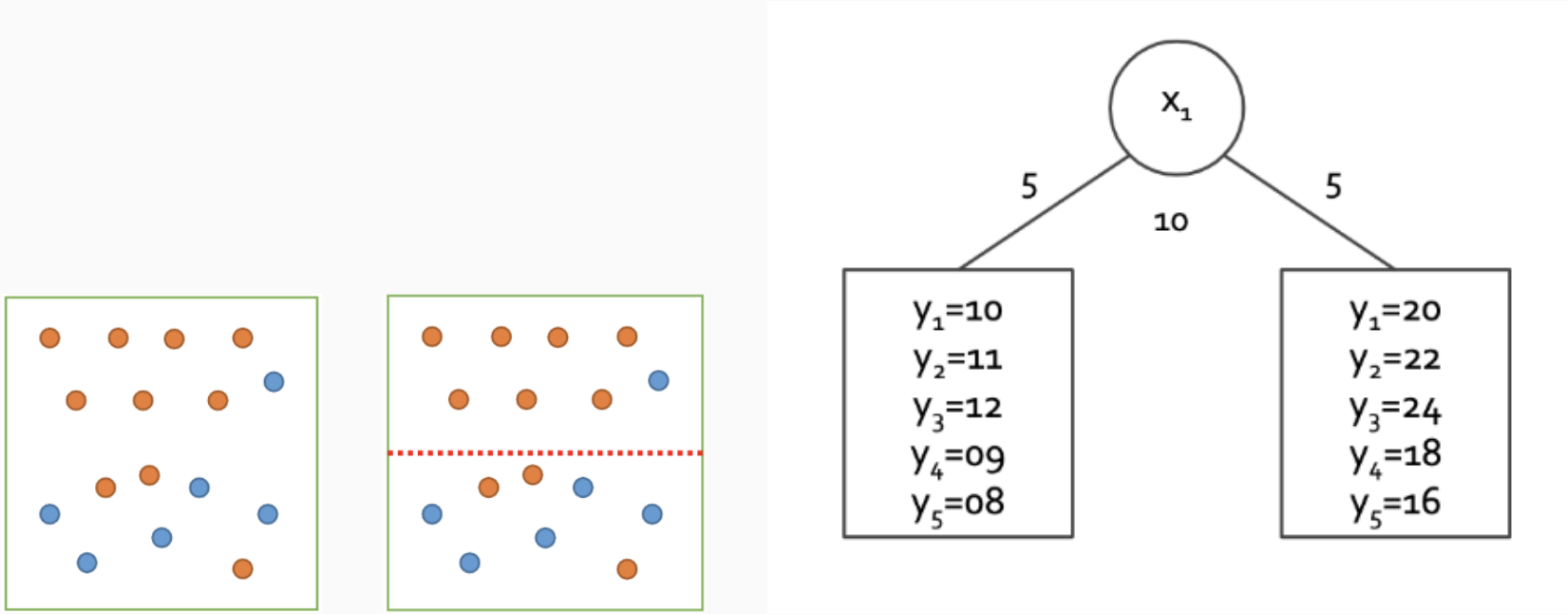

Figure 3: Let's calculate the information gain

The Best split is maximize the information gain

Then repeat the splitting for each node

Income	Lot size	Ownership
33.0	18.8	Non-owner
47.4	16.4	Non-owner
49.2	17.6	Non-owner
51.0	14.0	Non-owner
59.4	16.0	Non-owner
60.0	18.4	Owner
63.0	14.8	Non-owner
64.8	17.2	Non-owner
66.0	18.4	Non-owner
84.0	17.6	Non-owner
85.5	16.8	Owner
108.0	17.6	Owner

Figure 4: Example : Riding Mowers

Order records according to one variable

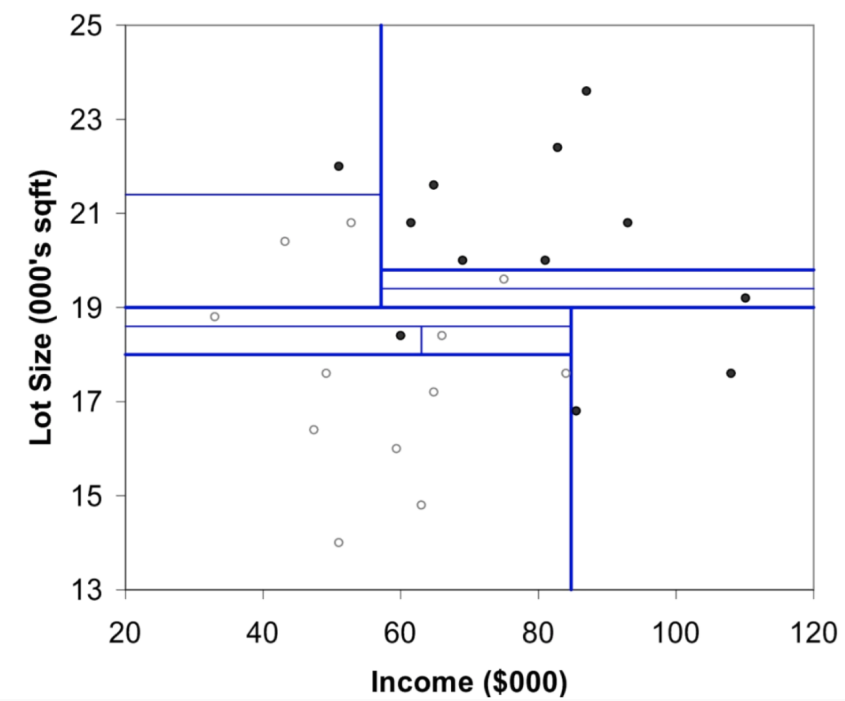

Figure 5: Now we get Full Tree

Recursive partitioning is completed when every leaf node has 100% purity (pure node)

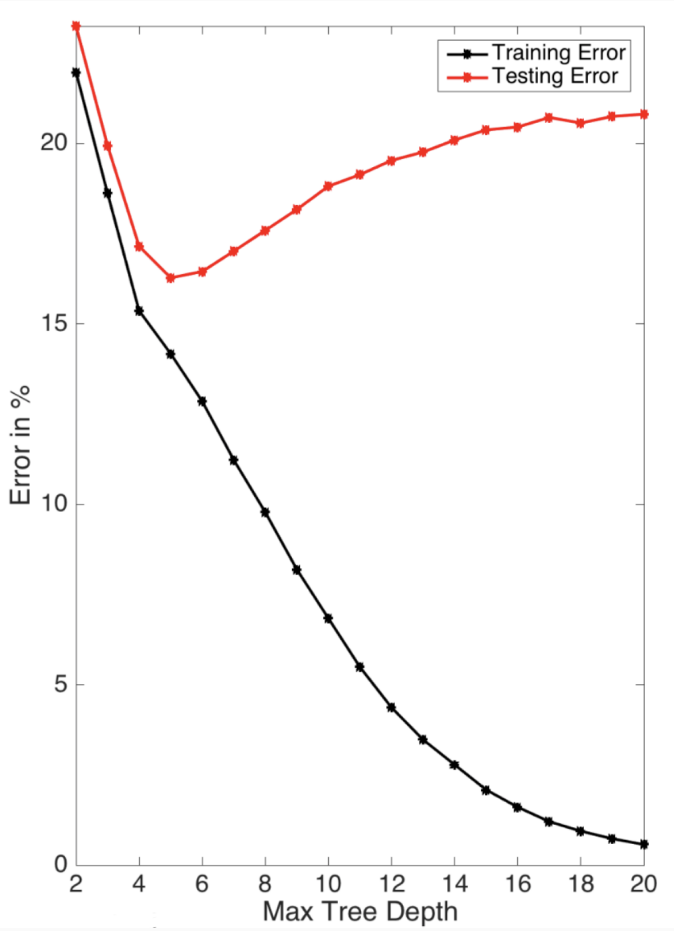

Figure 6: Error rate versus Max tree depth

Why we do Pruning?

- Full tree has risk of overfitting & poor generalization ability

- pre-pruning
- post-pruning

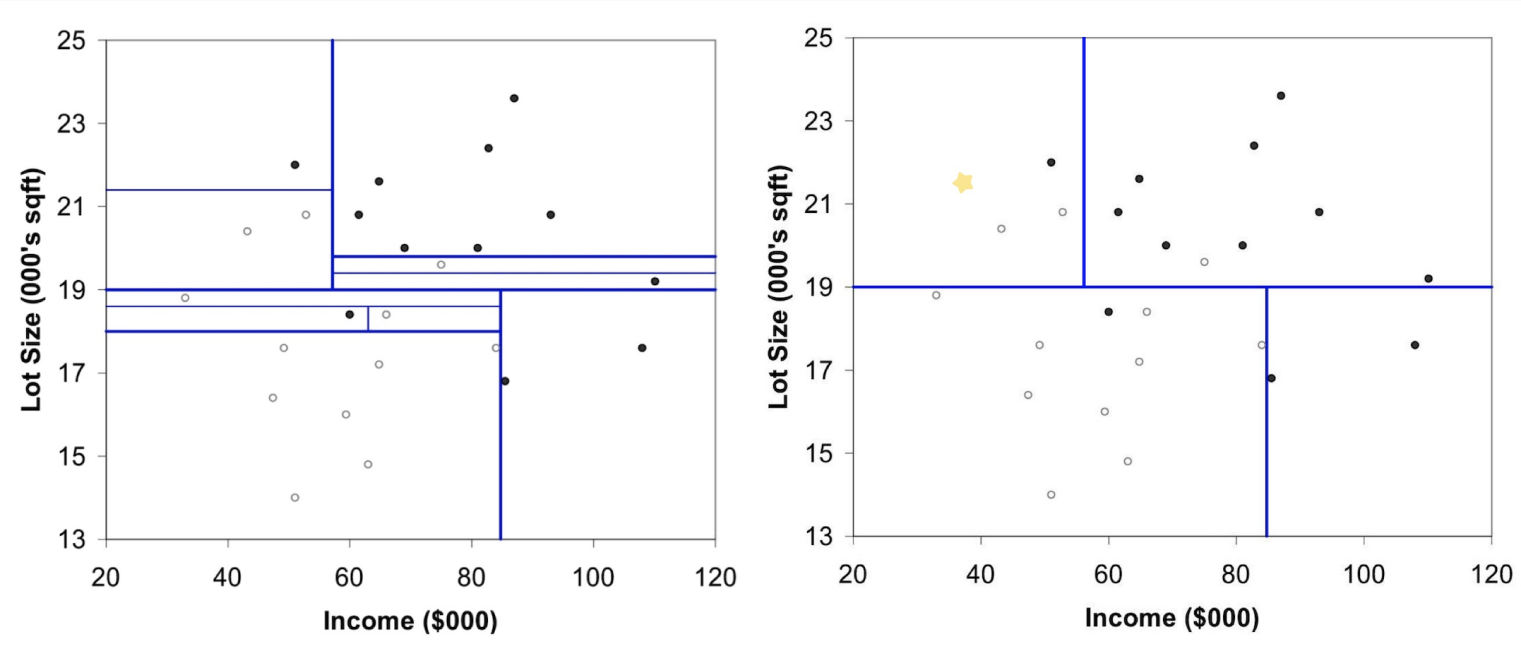

Figure 7: Full Tree versus Pruned Tree

Stopping Rule

- When every node is pure node(purity 100%)
- When the impurity is minimum (could not have lower value)
- By the Parameter setting, prepruning

Cost complexity

$$CC(T) = Err(T) + \alpha(T)$$

,

- $CC(T)$: cost complexity of a tree
- $ERR(T)$: proportion of the misclassified records in the validation data
- α : penalty factor attached to the tree size
- $L(T)$: the number of Leaf node

Pros

- Decision tree covers both Classification & Regression.
- Simple to interpret & understand 직관적
- 이상치, 노이즈에 큰 영향 없음
- 균일도 에만 초점 가능(전처리-스케일링, 정규화등 크게 불필요)
- 모형에 가정이 필요없는 비모수적 모형이다. 즉 정규 가정이 필요 없음

Cons

- 일반화가 어려움 : 불안정성 : 학습데이터에 따른 차이 큼 : 모델 variance 큼
- 오버피팅 가능성 높음
- 변수간 상호작용 불가능

Python code

```
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, stratify=cancer.target, random_state=42)

from sklearn.tree import DecisionTreeClassifier

tree = DecisionTreeClassifier(random_state=0)
tree.fit(X_train, y_train)
print("훈련 세트 정확도: {:.3f}".format(tree.score(X_train, y_train)))
print("테스트 세트 정확도: {:.3f}".format(tree.score(X_test, y_test)))
```

2. Control the complexity of Decision Tree, parameters

DecisionTreeClassifier can control the complexity of Decision Tree by parameters options, **prepruning**

```
tree = DecisionTreeClassifier(random_state=0)
```

```
tree = DecisionTreeClassifier(random_state=0, max_depth=4)
```

- max_depth : 최대 깊이 설정
- min_samples_split : 분할되기 위해 노드가 가지고 있어야하는 최소 샘플 수
- min_samples_leaf : 분할되기 위해 노드가 가져야하는 최소 샘플 수
- max_leaf_nodes : 리프 노드가 가지고 있어야하는 최소 샘플 수
- max_features : 리프 노드의 최대 수

2. Control the complexity of Decision Tree, ccp_alphas

DecisionTreeClassifier.cost_complexity_pruning_path provides another option to control the size of a tree, **post pruning**

- This pruning technique is parameterized by the cost complexity parameter, **ccp_alpha**.
- It returns the effective alphas and the corresponding total leaf impurities at each step of the pruning process.
- As alpha increases, more of the tree is pruned, which increases the total impurity of its leaves.

2. Control the complexity of Decision Tree, ccp_alphas

```
X, y = load_breast_cancer(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

clf = DecisionTreeClassifier(random_state=0)
path = clf.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas, impurities = path.ccp_alphas, path.impurities

clfs = []
for ccp_alpha in ccp_alphas:
    clf = DecisionTreeClassifier(random_state=0, ccp_alpha=ccp_alpha)
    clf.fit(X_train, y_train)
    clfs.append(clf)
print(
    "Number of nodes in the last tree is: {} with ccp_alpha: {}".format(
        clfs[-1].tree_.node_count, ccp_alphas[-1]
    )
)
```

Number of nodes in the last tree is: 1 with ccp_alpha: 0.3272984419327777

2. Control the complexity of Decision Tree, ccp_alphas

```
clfs = clfs[:-1]
ccp_alphas = ccp_alphas[:-1]

train_scores = [clf.score(X_train, y_train) for clf in clfs]
test_scores = [clf.score(X_test, y_test) for clf in clfs]

fig, ax = plt.subplots()
ax.set_xlabel("alpha")
ax.set_ylabel("accuracy")
ax.set_title("Accuracy vs alpha for training and testing sets")
ax.plot(ccp_alphas, train_scores, marker="o", label="train", drawstyle="steps-p
ax.plot(ccp_alphas, test_scores, marker="o", label="test", drawstyle="steps-pos
ax.legend()
plt.show()
```

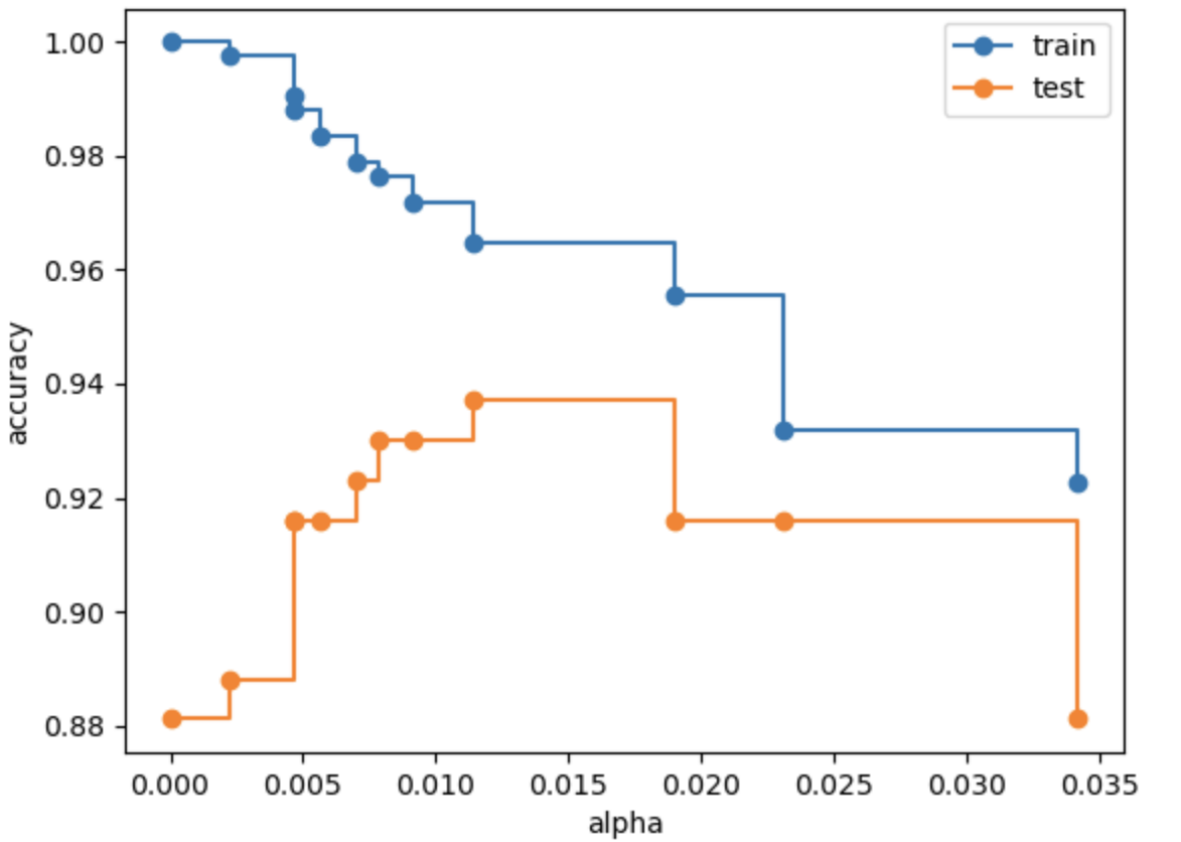

Figure 8: Accuracy versus alpha for training and test sets

```
from sklearn.tree import export_graphviz
export_graphviz(tree, out_file="tree.dot", class_names=["악성", "양성"],
                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))
```

```
from sklearn.tree import plot_tree

plot_tree(tree, class_names=["악성", "양성"], feature_names=cancer.feature_names
          impurity=False, filled=True, rounded=True, fontsize=4)

plt.show()
```

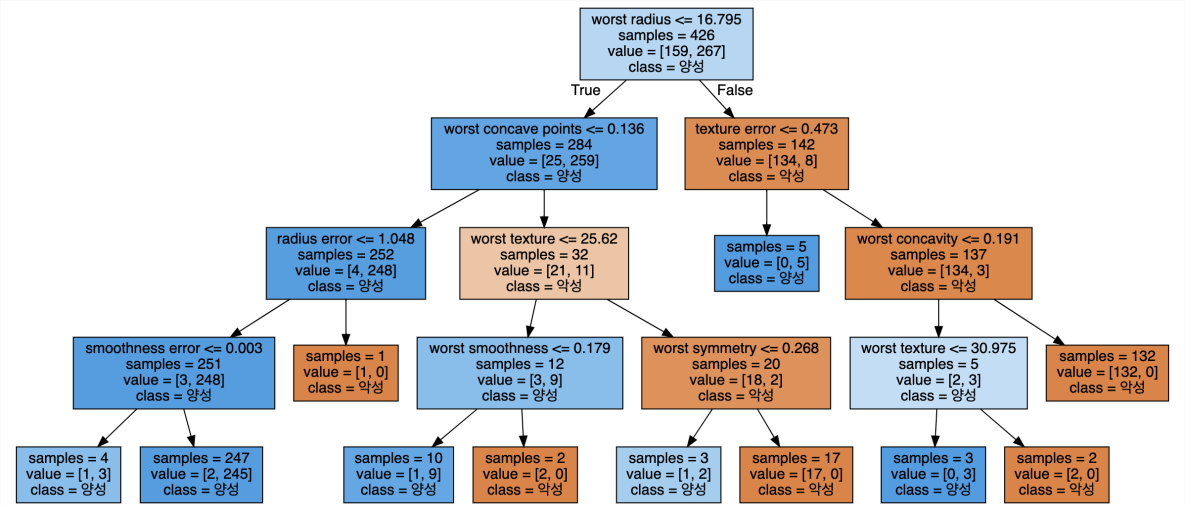

Figure 9: Decision tree of caner data

```
feature_importances_ : between 0 and 1

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("특성 중요도")
    plt.ylabel("특성")
    plt.ylim(-1, n_features)

plot_feature_importances_cancer(tree)
```

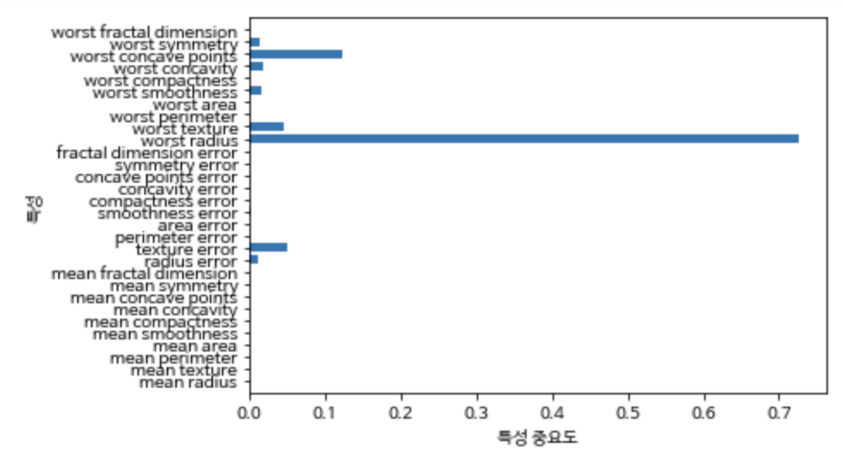

Figure 10: Feature importance

Attributes:

classes_ : *ndarray of shape (n_classes,) or list of ndarray*
The classes labels (single output problem), or a list of arrays of class labels (multi-output problem).

`feature_importances_` : *ndarray of shape (n_features,)*
Return the feature importances.

max_features_ : *int*
The inferred value of max_features.

n_classes_ : *int or list of int*
The number of classes (for single output problems), or a list containing the number of classes for each output (for multi-output problems).

`n_features_` : *int*
DEPRECATED: The attribute `n_features_` is deprecated in 1.0 and will be removed in 1.2.

n_features_in_ : *int*
Number of features seen during fit.

New in version 0.24.

feature_names_in_ : *ndarray of shape (`n_features_in_`,)*
Names of features seen during fit. Defined only when `X` has feature names that are all strings.

New in version 1.0.

n_outputs_ : *int*
The number of outputs when `fit` is performed.

tree_ : *Tree instance*
The underlying Tree object. Please refer to `help(sklearn.tree._tree.Tree)` for attributes of Tree object and Understanding the decision tree structure for basic usage of these attributes.

Figure 11: DecisionTreeClassifier, Attributes

5. DecisionTreeRegressor : Ram Price

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

from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression

# 2000년 이전을 훈련 데이터로, 2000년 이후를 테스트 데이터로 만듭니다
data_train = ram_prices[ram_prices.date < 2000]
data_test = ram_prices[ram_prices.date >= 2000]

# 가격 예측을 위해 날짜 특성만을 이용합니다
X_train = data_train.date.to_numpy()[:, np.newaxis]
# 데이터와 타깃 사이의 관계를 간단하게 만들기 위해 로그 스케일로 바꿉니다
y_train = np.log(data_train.price)

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

# 예측은 전체 기간에 대해서 수행합니다
X_all = ram_prices.date.to_numpy()[:, np.newaxis]

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

# 예측한 값의 로그 스케일을 되돌립니다
price_tree = np.exp(pred_tree)
price_lr = np.exp(pred_lr)
```

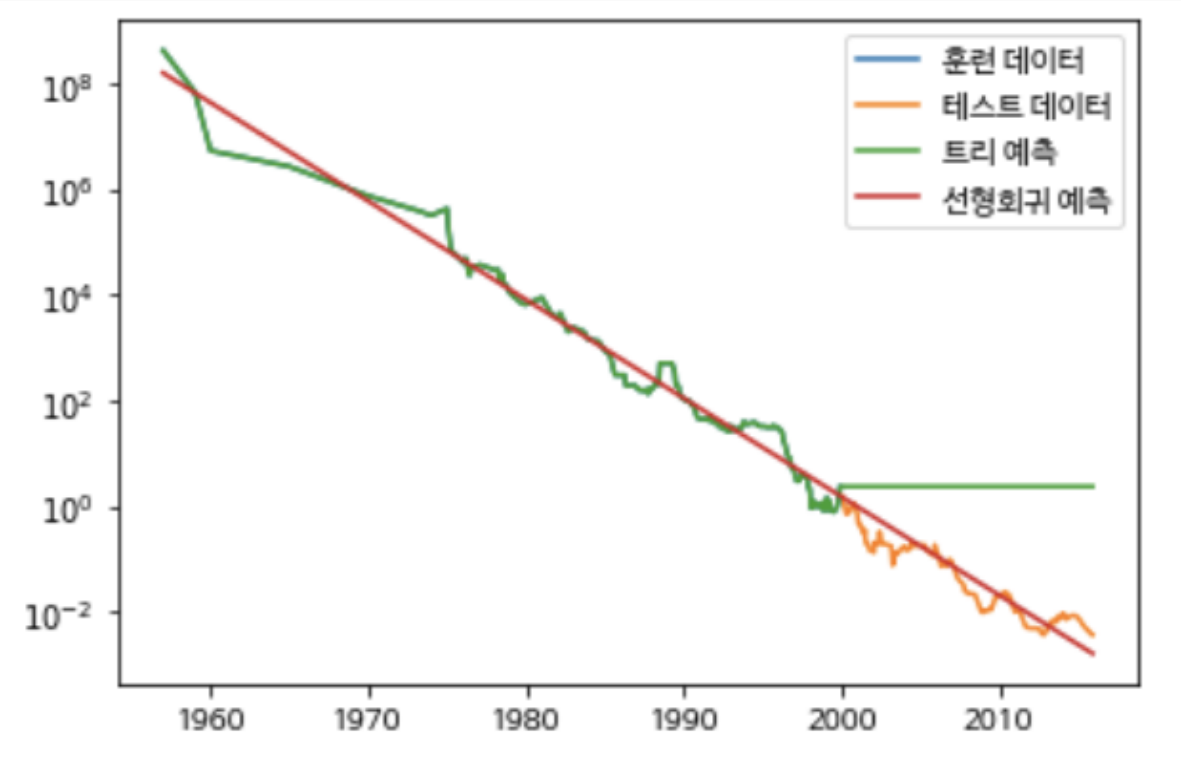

Figure 12: Predictors of ram price data

DecisionTreeRegressor can not predict the range outside of the training set

Thanks you.