결정트리(decision tree)

학습 내용

- 01 의사결정트리 모델을 생성해 보기
- 02 트리의 특성 중요도 알아보고 시각화 해보기
- 03 의사결정트리의 범주형/연속형 적용해보기
- 04 과적합을 해결해 보기

데이터 셋 다운로드

- https://www.kaggle.com/uciml/pima-indians-diabetes-database (https://www.kaggle.com/uciml/pima-indians-diabetes-database)

- (가) decision tree는 classification(분류)와 regression(회귀) 문제에 널리 사용하는 모델이다.
- (나) 스무고개 놀이의 질문과 비슷하다.

In [ ]:

```
# !pip install mglearn
```

In [ ]:

```
import matplotlib.pyplot as plt
import mglearn
```

In [ ]:

```
plt.figure(figsize=(10,10))
mglearn.plots.plot_animal_tree()
```

의사결정 트리 설명

- (가) 세개의 feature(속성, 변수)가 있다.
  - 'Has feathers?'(날개가 있나요?)
  - 'Can fly?'(날수 있나요?)
  - 'Has fins?'(지느러미가 있나요?)
- (나) 우리가 분류하고자 하는 문제는 네 개의 클래스로 구분하는 모델을 만든다.
  - 네 개의 클래스(매, 펭권, 돌고래, 곰)
- (다) 맨 위의 노드는 **Root Node(루트 노드)**라 한다.
- (라) 맨 마지막 노드를 우리는 **Leaf Node(리프노드)**라 불린다.
- (마) 범주형은 **데이터를 구분하는 질문**을 통해 데이터를 나누고, 연속형 데이터는 **특성 i가 값 a보다 큰가?** 라는 질문을 통해 나눈다.
- (바) target이 하나로만 이루어진 리프 노드를 **순수 노드(pure node)**고 한다.

01 의사결정트리 만들기

In [4]:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_breast_cancer
import seaborn as sns
```

In [5]:

```
cancer = load_breast_cancer()
all_X = cancer.data
all_Y = cancer.target
```

In [6]:

```
cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(all_X,
                                                    all_Y,
                                                    stratify=cancer.target,
                                                    test_size = 30,
                                                    random_state=77)

tree = DecisionTreeClassifier(max_depth=2, 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)))
```

```
훈련 세트 정확도 : 0.952
테스트 세트 정확도 : 0.833
```

02 결정트리 복잡도를 제어해보기 - 일반화

- 결정트리의 깊이를 제한하지 않으면 트리는 무작정 깊어지고 복잡해 질 수 있다.
- 첫번째는 트리가 일정 깊이에 도달하면 트리의 성장을 멈추게 하는 것.
  - max_depth를 이용

In [7]:

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

```
훈련 세트 정확도 : 0.981
테스트 세트 정확도 : 0.867
```

유방암 데이터셋으로 만든 결정 트리 시각화

In [17]:

```
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)
```

In [16]:

```
import graphviz

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

<graphviz.files.Source at 0x7f3e2eb58e10>

03 트리의 특성 중요도

- 특성 중요도 : 이 값은 0과 1사이의 숫자.
  - 0은 전혀 사용되지 않음.
  - 1은 완벽하게 타깃 클래스를 예측했다.
  - 특성 중요도의 전체 합은 1이다.
- 특성의 feature_importance_ 값이 낮다고 해서 특성이 유용하지 않다는 것이 아니다.
- 단지 트리가 그 특성을 선택하지 않았다는 것.

In [8]:

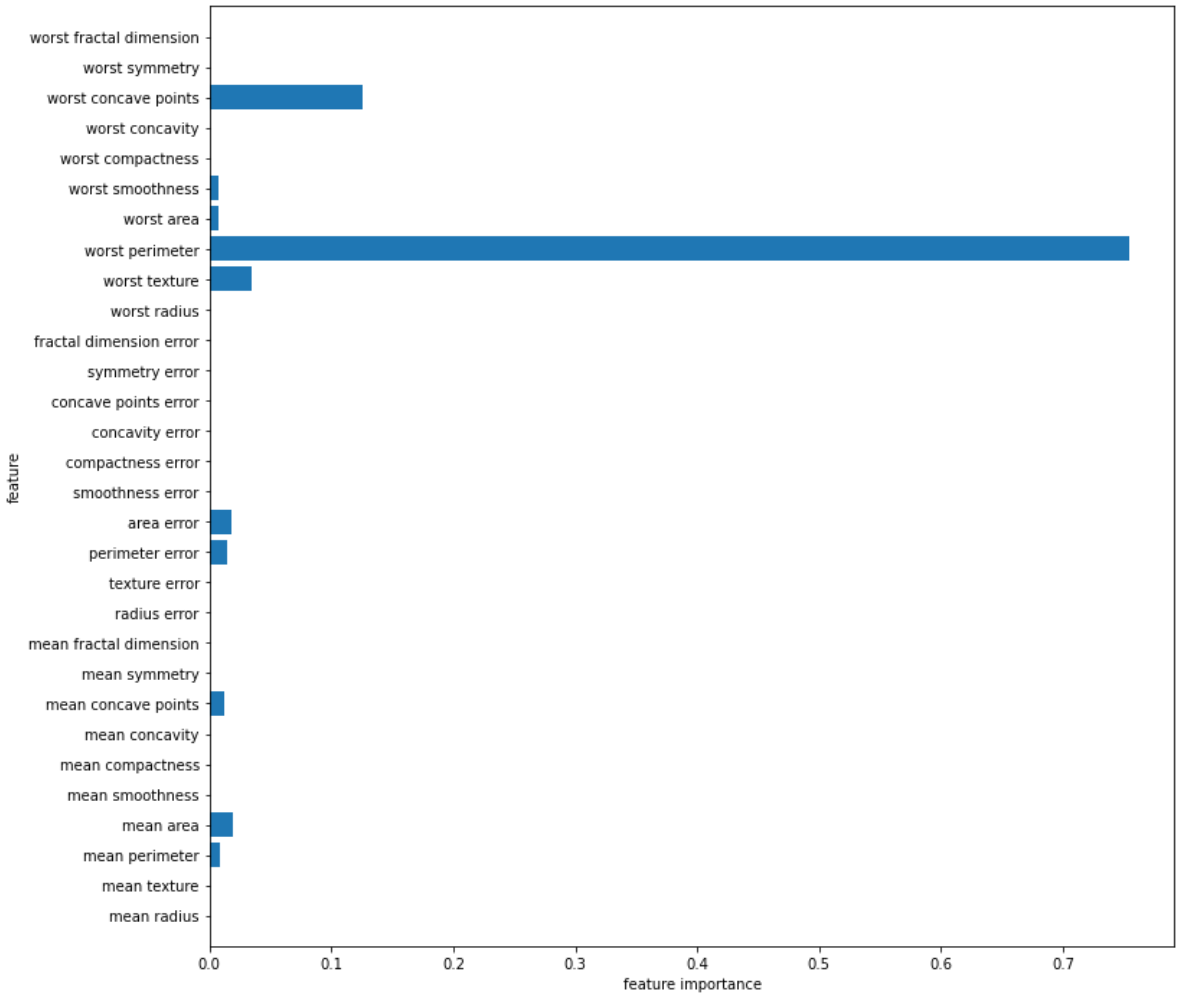
```
import numpy as np
```

In [9]:

```
def plot_feature_imp_cancer(model):
    n_features = cancer.data.shape[1]
    imp = model.feature_importances_
    plt.barh(range(n_features) , imp, align='center')
    plt.yticks(np.arange(n_features), cancer.feature_names)

    plt.xlabel("feature importance")
    plt.ylabel("feature")
    plt.ylim(-1, n_features)
```

In [10]:

```
plt.figure(figsize=(12,12))
plot_feature_imp_cancer(tree)
```



- worst perimeter이 가장 중요한 특성으로 나타난다.
  - 첫번째 노드에서 두 클래스를 꽤 잘 나누고 있다.
- feature_importance_ 값이 낮다고 해서 특성이 유용하지 않다는 뜻이 아님.
-

04 의사결정트리 - 회귀

- 컴퓨터 메모리 가격 동향 데이터 셋 활용해 보기
- x축 : 년 (날짜)

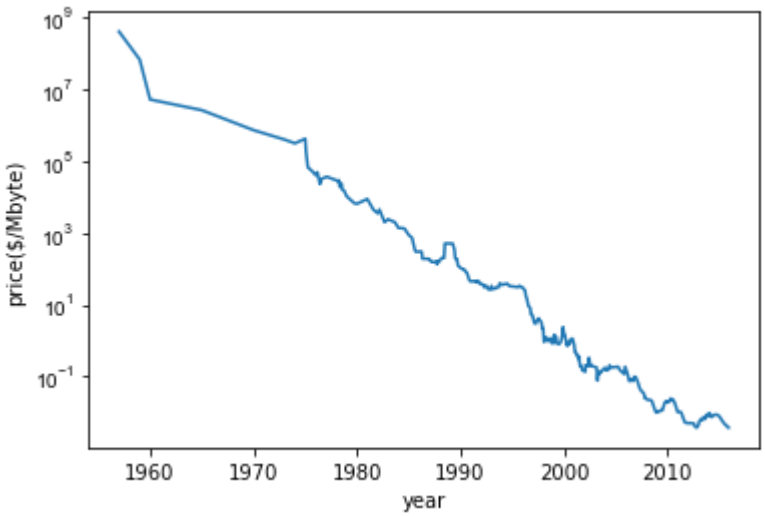
- y축 : 가격 (해당 년도의 램(RAM) 1메가바이트당 가격) - 로그 스케일

In [11]:

```
import os
import pandas as pd
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression

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

# 한글 폰트가 지수에 음수를 표시하지 못하므로 ytick의 폰트를 바꾸어 줍니다.
plt.yticks(fontname = "Arial")

# Make a plot with log scaling on the y axis.(y축 로그 스케일)
plt.semilogy(ram_prices.date, ram_prices.price)
plt.xlabel("year")
plt.ylabel("price($/Mbyte)")
```

Out[11]:

```
Text(0, 0.5, 'price($/Mbyte)')
```



- 가격을 로그 스케일을 바꾸어 비교적 선형적인 관계를 가짐.

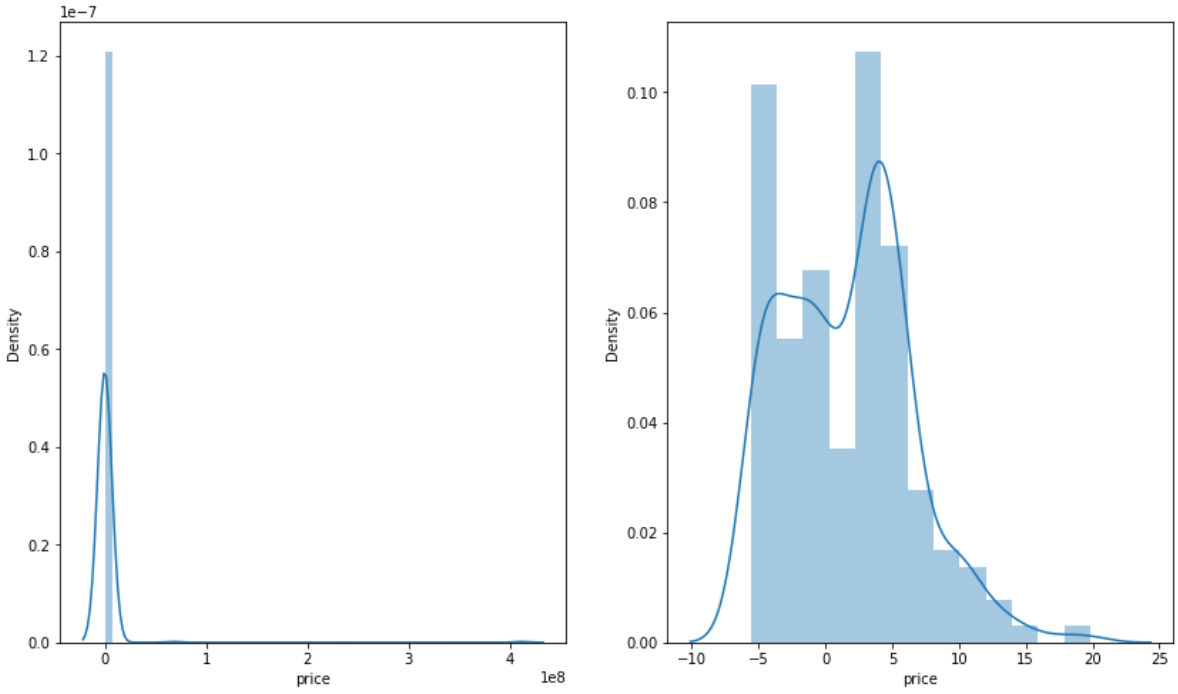
In [12]:

```
plt.figure(figsize=(14,8))

plt.subplot(1,2,1)
sns.distplot(ram_prices.price)

plt.subplot(1,2,2)
log_price = np.log(ram_prices.price)
sns.distplot(log_price)
```

```
C:\Users\toto\anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarni
ng: `distplot` is a deprecated function and will be removed in a future version. Ple
ase adapt your code to use either `displot` (a figure-level function with similar fl
exibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\toto\anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarni
ng: `distplot` is a deprecated function and will be removed in a future version. Ple
ase adapt your code to use either `displot` (a figure-level function with similar fl
exibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[12]:

```
<AxesSubplot:xlabel='price', ylabel='Density'>
```



In [13]:

```
from sklearn.tree import DecisionTreeRegressor

# 2000년 이전을 학습 데이터로,습
# 2000년 이후를 테스트 데이터로 변경
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[:, np.newaxis]

# 데이터와 타깃 사이의 관계를 간단하게 만들기 위해 로그 스케일로 바꿉니다
y_train = np.log(data_train.price)
```

```
<ipython-input-13-6de4b1fb924c>:9: FutureWarning: Support for multi-dimensional inde
xing (e.g. `obj[:, None]`) is deprecated and will be removed in a future version.  C
onvert to a numpy array before indexing instead.
  X_train = data_train.date[:, np.newaxis]
```

In [14]:

```
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[:, 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)
```

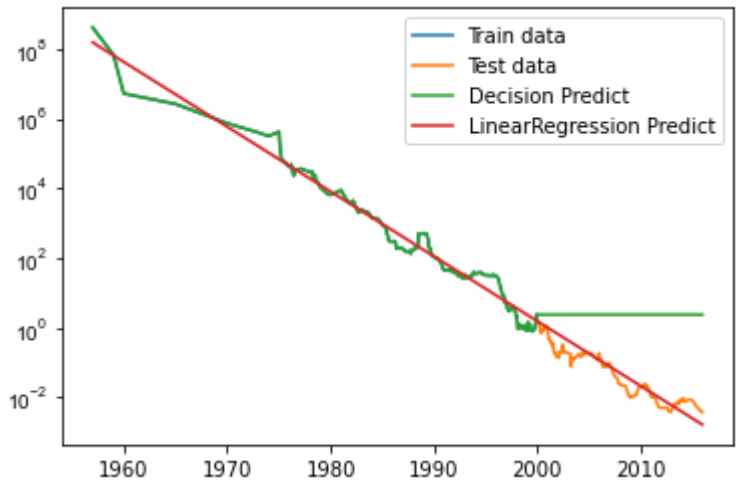
```
<ipython-input-14-95e0b1948a38>:5: FutureWarning: Support for multi-dimensional inde
xing (e.g. `obj[:, None]`) is deprecated and will be removed in a future version.  C
onvert to a numpy array before indexing instead.
  X_all = ram_prices.date[:, np.newaxis]
```

In [15]:

```
plt.yticks(fontname = "Arial") # 한글 폰트가 지수에 음수를 표시하지 못하므로 ytick의 폰트를 바꾸어 줌
plt.semilogy(data_train.date, data_train.price, label="Train data")
plt.semilogy(data_test.date, data_test.price, label="Test data")
plt.semilogy(ram_prices.date, price_tree, label="Decision Predict")
plt.semilogy(ram_prices.date, price_lr, label="LinearRegression Predict")
plt.legend()
```

Out[15]:

<matplotlib.legend.Legend at 0x113e6011e80>



- 두 모델은 확연한 차이를 보인다.
  - (1) 선형모델은 **직선**으로 데이터를 근사한다.
  - (2) 트리모델은 **훈련 데이터를 완벽하게 예측**한다.
  - (3) 트리 모델은 훈련 데이터 밖의 **새로운 데이터를 예측할 능력이 없다.** - 과적합
    - (3)번의 내용이 트리 기반 모델의 **공통된 단점**이다.

결정 트리의 장점

첫째, 만들어진 모델을 쉽게 시각화할 수 있어서, 비전문가도 이해하기 쉽다.
둘째, 데이터의 스케일에 구애받지 않는다.(정규화, 표준화 전처리 과정 필요 없다.)

결정 트리의 단점

사전 가지치기의 사용하지만 Overfitting(과대적합) 되는 경향이 있다.

트리모델의 모델 복잡도 조절 매개 변수

max_depth : 트리의 깊이

- max_depth = 4라면 연속된 질문의 옵션을 최대 4개로 제한
- 트리 깊이를 제한하면 과대적합이 줄어든다.

- 훈련 세트의 정확도는 떨어지지만 테스트 성능은 개선

max_leaf_nodes : 최대 leaf 노드의 수

min_samples_leaf : 노드가 분할하기 위한 잎노드의 최소 데이터 개수

과제

- 2차 캐글 대회의 데이터 셋을 활용하여 선형회귀 모형과 의사결정트리 모델을 구해보자.
- url : https://www.kaggle.com/c/2019-2nd-ml-month-with-kakr의 (https://www.kaggle.com/c/2019-2nd-ml-month-with-kakr%EC%9D%98) 대회

업그레이드

- (1) MSE를 구해보자. 어떤 경우에 가장 좋은 모델이 되는가?
- (2) Knn, 선형회귀, 의사결정트리 어떤 모델이 가장 좋은가? 그때의 MSE를 구해보자

도전 실습

- 나만의 데이터 셋을 활용하여 의사 결정 트리 모델을 적용해 보자.
  - 1-1 의사 결정 트리 모델 적용하여 평가 결과 확인해 보기
  - 1-2 의사 결정 트리 모델의 파라미터를 변경해 보며 결과 확인해 보기

In [ ]: