# 중요한 특징(feature)를 선택하는 방법

- 통계가 좋은 특징 - 일변량 통계(univariate statistics)
- 하나의 모델을 통해 선택 - 모델 기반 선택(model-based selection)
- 모델을 통해 반복적 선택 - 반복적 선택(iterative selection)

## 학습 목표

- boston 데이터 셋을 활용한 여러가지 방법을 활용한 변수 선택

## 목차



## 1-1-1 일변량 통계

목차로 이동하기

- 개개의 특성과 타깃(목표변수) 사이에 중요한 통계적 관계가 있는지 계산
- 분류에서는 분산분석(ANOVA)라고 한다.
- 각 특성(feature)이 독립적으로 평가.
- 계산이 매우 빠르고 평가를 위한 모델을 만들 필요가 없음.
- SelectPercentile에서 특성을 선택하는 기준은 **F-값**
  - 값이 클수록 클래스 평균의 분산이 비교적 크다.

## 평가 지표

- 분류는 f_classif를 사용하고 회귀는 f_regression를 사용한다.

In [1]:

```
import warnings
warnings.filterwarnings(action='ignore')
# warnings.filterwarnings(action='default')
```

In [2]:

```
from sklearn.feature_selection import SelectPercentile
from sklearn.feature_selection import f_regression, f_classif

from sklearn.model_selection import train_test_split

from sklearn.datasets import load_boston
from sklearn.preprocessing import MinMaxScaler, PolynomialFeatures
import numpy as np
import pandas as pd

from sklearn.linear_model import LinearRegression
```

In [3]:

```
boston = load_boston()

df_boston = pd.DataFrame(boston.data, columns=boston.feature_names)
df_boston['target'] = pd.Series(boston.target)
df_boston.head()
```

Out[3]:

| | CRIM | ZN | INDUS | CHAS | NOX | RM | AGE | DIS | RAD | TAX | PTRATIO | B | LS |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.00632 | 18.0 | 2.31 | 0.0 | 0.538 | 6.575 | 65.2 | 4.0900 | 1.0 | 296.0 | 15.3 | 396.90 | |
| **1** | 0.02731 | 0.0 | 7.07 | 0.0 | 0.469 | 6.421 | 78.9 | 4.9671 | 2.0 | 242.0 | 17.8 | 396.90 | |
| **2** | 0.02729 | 0.0 | 7.07 | 0.0 | 0.469 | 7.185 | 61.1 | 4.9671 | 2.0 | 242.0 | 17.8 | 392.83 | |
| **3** | 0.03237 | 0.0 | 2.18 | 0.0 | 0.458 | 6.998 | 45.8 | 6.0622 | 3.0 | 222.0 | 18.7 | 394.63 | |
| **4** | 0.06905 | 0.0 | 2.18 | 0.0 | 0.458 | 7.147 | 54.2 | 6.0622 | 3.0 | 222.0 | 18.7 | 396.90 | |

In [4]:

```
# 입력 데이터와 출력데이터를 지정해준다.
X = df_boston.loc[ : , 'CRIM':'LSTAT']
y = boston.target
print("정규화, 확장 전 데이터 셋 : ", X.shape, y.shape)
```

정규화, 확장 전 데이터 셋 : (506, 13) (506,)

In [5]:

```
## 값을 전체적으로 0~1로 사이로 만들기
nor_X = MinMaxScaler().fit_transform(X)
nor_X.min(), nor_X.max()
```

Out[5]:

(0.0, 1.0)

## 변수 생성

In [6]:

```
ex_X = PolynomialFeatures(degree=2, include_bias=False).fit_transform(nor_X)
print( ex_X.shape, type(ex_X) )
```

(506, 104) <class 'numpy.ndarray'>

## 정규화와 특징 생성 후, 데이터 나누기

In [7]:

```
X = ex_X          # 입력
y = boston.target # 출력

X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    random_state=0,
                                                    test_size=0.5)

# 50%를 뽑는 것을 학습
select = SelectPercentile(score_func=f_regression, percentile=50)
select.fit(X_train, y_train)
```

Out[7]:

```
SelectPercentile(percentile=50,
                 score_func=<function f_regression at 0x000001F9335B2820>)
```

In [8]:

```
## 학습 세트에 적용
X_tr_selected = select.transform(X_train)

print( "X_train.shape:", X_train.shape)
print( "X_train_selected.shape", X_tr_selected.shape)
```

X_train.shape: (253, 104)
X_train_selected.shape (253, 52)

- 결과를 통해 우리는 특징 개수가 104개에서 52개로 줄어든 것을 확인할 수 있음.

## 어떤 특징이 선택 되었는지 시각화를 통해 확인

In [9]:

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

In [10]:

```
### 어떤 특성이 선택되었는지 확인
mask = select.get_support()
print(mask)
plt.matshow(mask.reshape(1, -1), cmap='gray_r')
```

```
[ True  True  True False  True  True False False False  True  True False
  True False False  True False  True False  True  True False  True  True
 False  True False False False  True  True  True False False False False
  True False  True False  True False  True False  True  True  True False
  True False False False False False False False False False False  True
 False  True False  True  True  True False  True  True False  True False
 False False  True  True  True False  True  True  True False  True False
 False False False False False  True  True  True False  True  True  True
 False  True  True False  True False  True  True]
```

Out[10]:

```
<matplotlib.image.AxesImage at 0x1f9361f1880>
```


## 전체 특성을 사용한 점수, 선택된 특성을 사용한 점수

In [11]:

```
lr = LinearRegression()
lr.fit(X_train, y_train)

print("전체 특성 사용 score(학습) : {:.3f}".format(lr.score(X_train, y_train)))
print("전체 특성 사용 score(테스트): {:.3f}".format(lr.score(X_test, y_test)))
```

```
전체 특성 사용 score(학습) : 0.969
전체 특성 사용 score(테스트): 0.664
```

In [12]:

```
select = SelectPercentile(score_func=f_regression, percentile=50)
select.fit(X_train, y_train)
X_tr_selected = select.transform(X_train)
mask = select.get_support()
```

In [13]:

```
lr1 = LinearRegression()
lr1.fit(X_tr_selected, y_train)

X_test_selected = X_test[:, mask]
print(X_tr_selected.shape, X_test_selected.shape )

print("선택된 일부 특성 사용(학습용) : {:.3f}".format(lr1.score(X_tr_selected, y_train)))
print("선택된 일부 특성 사용(테스트용) : {:.3f}".format(lr1.score(X_test_selected, y_test)))
```

```
(253, 52) (253, 52)
선택된 일부 특성 사용(학습용) : 0.930
선택된 일부 특성 사용(테스트용) : 0.758
```

- 0.664에서 선택된 일부 특성(50%)를 선택 후, 성능이 0.758로 향상되었습니다.

## 1-1-2 모델 기반 특성 선택

목차로 이동하기

- 지도 학습 머신러닝 모델을 사용하여 **특성의 중요도를 평가해서 가장 중요한 특성**들만 선택
- 특성 선택에 사용하는 지도 학습 모델은 최종적으로 사용할 지도학습 모델과 같을 필요는 없음.
- 결정 트리와 유사한 모델은 feature_importance_ 속성을 제공함.
- 선형 모델의 절대값으로 특성의 중요도를 재는데 사용
- 모델 기반의 특성 선택은 SelectFromModel에 구현되어 있음.

In [14]:

```
from sklearn.feature_selection import SelectFromModel
from sklearn.ensemble import RandomForestRegressor
```

In [15]:

```
select = SelectFromModel(RandomForestRegressor(n_estimators=100,
                                               random_state=42),
                                               threshold="median")  # 1.25*mean, 0.75*mean
```

- SelectFromModel은 지도학습 모델(랜덤포레스트)로 계산된 중요도가 기준
- 임계치보다 큰 모든 특성을 선택
- 절반 가량의 특성이 선택될 수 있도록 중간값(threshold='median')을 임계치로 사용.
- 트리 100개로 만든 랜덤 포레스트 분류기를 사용.

In [16]:

```
select.fit(X_train, y_train)
X_train_l1 = select.transform(X_train)
print("X_train.shape :" , X_train.shape)
print("X_train_l1.shape :", X_train_l1.shape)
```

```
X_train.shape : (253, 104)
X_train_l1.shape : (253, 52)
```

In [17]:

```
### 어떤 특성이 선택되었는지 확인
mask = select.get_support()
print(mask)
plt.matshow(mask.reshape(1, -1), cmap='gray_r')
plt.xlabel("특성 번호")
```

```
[False False False False False  True False False False False False  True
  True False  True  True False  True  True  True False False  True  True
 False  True False  True False False False False False False False False
 False False False False  True False  True  True  True False  True  True
  True False False False False False False False False False False False
 False False  True False  True  True False  True  True  True  True  True
  True  True  True  True False  True  True  True  True  True  True False
  True  True  True  True  True False False  True  True  True False  True
  True  True False  True  True False  True  True]
```

Out[17]:

```
Text(0.5, 0, '특성 번호')
```


In [18]:

```
# 학습용, 테스트 데이터 변환
X_train_l1 = select.transform(X_train)
mask = select.get_support()

X_test_l1 = X_test[:, mask]

lr1 = LinearRegression()
lr1.fit(X_train_l1, y_train)

print("일부 특성 사용(SelectFromModel-학습) : {:.3f}".format(lr1.score(X_train_l1, y_train)))
print("일부 특성 사용(SelectFromModel-테스트) : {:.3f}".format(lr1.score(X_test_l1, y_test)))
```

```
일부 특성 사용(SelectFromModel-학습) : 0.940
일부 특성 사용(SelectFromModel-테스트) : 0.770
```

- 0.664에서 선택된 일부 특성(50%)를 선택 후, 성능이 0.758로 향상되었습니다.(SelectPercentile)
- 0.664에서 선택된 일부 특성(50%)를 선택 후, 성능이 0.770로 향상되었습니다.(SelectFromModel)
- 위의 결과와 비교해 보면 SelectPercentile으로 선택한 것보다 성능이 좋음.

## 1-1-3 반복적 특성 선택

목차로 이동하기

- A. 일변량 모델은 모델을 사용하지 않음.(F값)
- B. 모델 기반 선택은 **하나의 모델**을 사용
- C. 반복적 특성 선택(iterative Feature Selection)에서는 특성 선택 시, 각각 다른 모델을 사용.
  - 하나, 특성을 하나도 선택하지 않은 상태로 시작해서 어떤 종료 조건까지 하나씩 추가
  - 둘, 모든 특성을 가지고 시작하여 어떤 종료 조건이 될때까지 특성을 하나씩 제거.
- D. 많은 모델 사용하므로 앞서 소개한 방법들보다 계산 비용이 훨씬 많이 든다.

- 재귀적 특성 제거(RFE:recursive feature elimination)가 하나의 방법

In [19]:

```
from sklearn.feature_selection import RFE
```

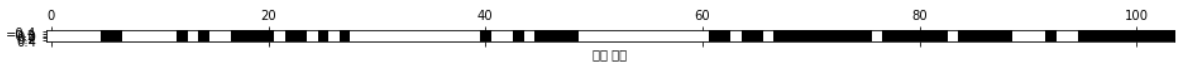
In [20]:

```
%%time

# RFE 반복적인 변수의 제거를 통해 좋은 피처만 남긴다.
select = RFE(RandomForestRegressor(n_estimators=100, random_state=42),
             n_features_to_select=52)

select.fit(X_train, y_train)

# 선택된 특성을 표시합니다.
mask = select.get_support()
plt.matshow(mask.reshape(1,-1), cmap='gray_r')
plt.xlabel("특성 번호")
```

```
Wall time: 32.7 s
```

Out[20]:

```
Text(0.5, 0, '특성 번호')
```




- 일변량 분석이나 모델 기반 특성보다 특성 선택이 나아짐.
- 랜덤 포레스트 모델은 특성이 누락될때마다 다시 학습하므로 52번 실행.
- 이 코드를 실행하면 모델 기반 선택보다 훨씬 오래 걸림.

In [21]:

```
X_tr_rfe = select.transform(X_train)
mask = select.get_support()

X_test_rfe = X_test[:, mask]
```

In [22]:

```
model = LinearRegression().fit(X_tr_rfe, y_train)

print("일부 특성 사용(RFE-학습) : {:.3f}".format(model.score(X_tr_rfe, y_train)))
print("일부 특성 사용(RFE-테스트) : {:.3f}".format(model.score(X_test_rfe, y_test)))
```

```
일부 특성 사용(RFE-학습) : 0.941
일부 특성 사용(RFE-테스트) : 0.799
```

- 0.664에서 선택된 일부 특성(50%)를 선택 후, 성능이 0.758로 향상되었습니다.(SelectPercentile)
- 0.664에서 선택된 일부 특성(50%)를 선택 후, 성능이 0.770로 향상되었습니다.(SelectFromModel)
- 0.664에서 선택된 일부 특성(50%)를 선택 후, 성능이 0.799로 향상되었습니다.(RFE)

## 최종 확인 결과

- F통계량, 랜덤포레스트 특성 중요도 이용, RFE 이용하여 확인한 결과
  - RFE > RandomForestRegressor 특성 이용 > 통계량 이용 순으로 성능이 차이가 있었다.

In [23]:

```
### RFE에서 사용된 모델로 예측
model = LinearRegression().fit(X_tr_rfe, y_train)
print("최종 모델 테스트 점수 : {:.3f}".format(model.score(X_test_rfe, y_test)))
```

```
최종 모델 테스트 점수 : 0.799
```