

# AI 기초

2019-2020

강봉주

# 변수 선택

# 변수 선택

## [필요한 패키지]

```
In      # 필요한 패키지
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

import scipy
from scipy import linalg as la
import scipy.stats as ss
import scipy.special

# 한글출력
plt.rcParams['font.family'] = 'Malgun Gothic'
plt.rcParams['axes.unicode_minus'] = False

# 필요한 패키지 2
import statsmodels.formula.api as smf
import statsmodels.api as sm

# 필요한 패키지 3
import sklearn
from matplotlib.ticker import FuncFormatter

from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.model_selection import cross_val_score

sklearn.__version__
Out      '0.22.1'
```

# 변수 선택

## [개요]

- 모형의 적합도(goodness of fit)가 비슷하다면 가급적 변수의 수를 줄이는 이유는
- 차원의 저주, 해석의 용이함, 모델적합 시간 단축 등
- 불필요하거나(redundant) 관련 없는(irrelevant) 변수의 제거
- 잡음(noise)까지 모형화하는 과적합 방지

# 변수 선택

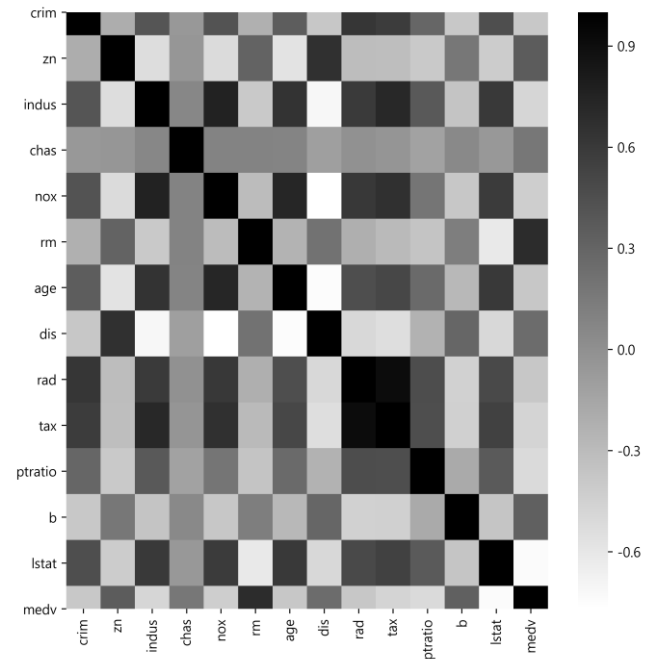
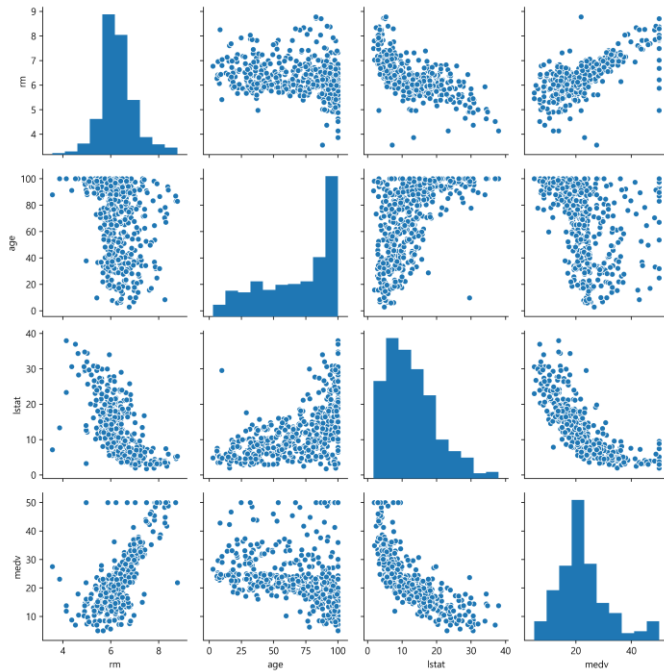
## [개요]

- 완전 탐색 (exhaustive search) : 모든 가능한 변수 조합에 대하여 비용함수를 최소화 하는 변수 선택 방법
- 포장(wrapper) 방법: 블랙박스 모형
- 여과(filter) 방법: 상호 정보(mutual information), 피어슨 상관계수 등
- 내장 방법: 라쏘 또는 능형 회귀

# 변수 선택

## [여과 방법]

- 산점도
- 상관 분석



# 변수 선택

## [포장 방법]

- 여과 방법에서 한 걸음 더
- 분산 분석 기법
- 모델 간의 차이 검증

$$F = \frac{(SSE(reduced) - SSE(full))/d}{SSE(full)/(n - k - 1)} \sim F(d, n - k - 1)$$

- $d$ 는 2개의 모델 간에 적합된 모수의 개수의 차이
- 차이가 유의미하면 완전 모델에만 있는 변수가 유의미함

# 변수 선택

## [포장 방법]

- 교차 검증
- 블랙박스 모형의 결과 값 만을 이용한 변수 선택 또는 모형 선택 방법



# 변수 선택

## [포장 방법]

- 교차 검증
- 단순 교차검증

순서	내용	비고
1	데이터를 무작위로 분할한다. 분할한 데이터를 각각 $S_{\text{train}}, S_{\text{cv}}$ 로 한다.	일반적으로 훈련 데이터를 70%, 검증 데이터를 30%로 함
2	모든 후보 모형에 대하여 훈련 데이터인 $S_{\text{train}}$ 로 학습시킨 후 모형 $\hat{f}_i$ 을 얻는다.	각 모형에서 추정되어야 할 모수 들을 추정
3	모든 모형을 검증 데이터인 $S_{\text{cv}}$ 에 적용하여 가장 작은 오차를 주는 모형을 선택한다.	

# 변수 선택

## [포장 방법]

- 교차 검증
- 단순 교차검증
- 데이터 분할

---

```
In  # 데이터 표준화
    scaled_df = (df - np.mean(df, axis=0))/np.std(df, axis=0)

    # 데이터 분할
    # 훈련 데이터
    np.random.seed(123)
    index = np.random.choice(np.arange(len(scaled_df)), size=np.int(len(scaled_df)*0.7), replace=False)
    train = scaled_df.iloc[index]
    train.shape

Out (354, 14)
```

---

# 변수 선택

## [포장 방법]

- 교차 검증
- 단순 교차검증
- 데이터 분할

---

```
In    # 검증 데이터
      non_index = list(set(np.arange(len(scaled_df)))-set(index))
      valid = scaled_df.iloc[non_index]
      valid.shape
Out    (152, 14)
```

---

# 변수 선택

## [포장 방법]

- 교차 검증
- 단순 교차검증
- 모형 적합

```
In    # 입력 변수 구성
      xvars = list(set(scaled_df.columns) - {'medv'})

      # 각 변수를 하나씩 적합하기
      rss = []
      for j in np.arange(len(xvars)):
          varname = xvars[j]
          fit = smf.ols('medv~'+varname, data=scaled_df).fit()
          pred = fit.predict(sm.add_constant(valid[varname]))
          tmp_rss = np.sum((valid['medv'].values-pred)**2)
          rss.append(tmp_rss)

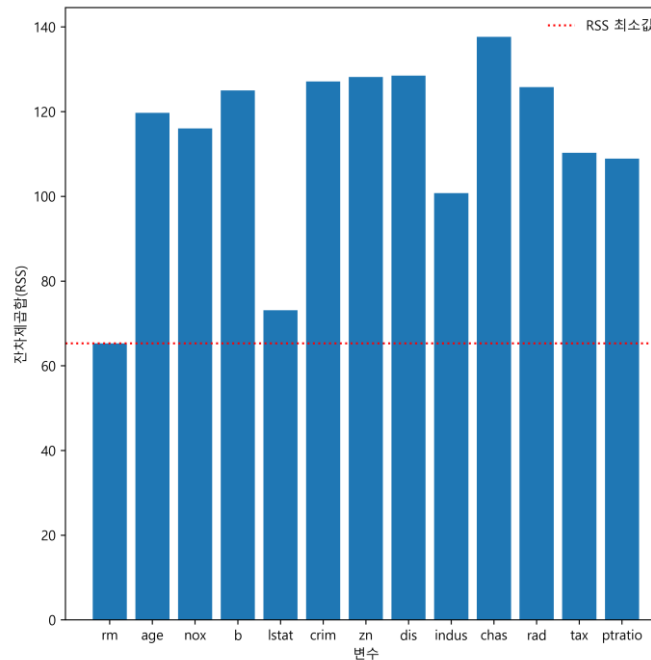
      # 최소값을 주는 변수
      xvars[np.nanargmin(rss)]

Out    'rm'
```

# 변수 선택

## [포장 방법]

- 교차 검증
- 단순 교차검증
- 모형 적합



# 변수 선택

## [포장 방법]

- 교차 검증
- K-곱 교차검증

순서	내용	비고
1	데이터를 무작위로 균등하게 $K$ 개로 분할한다. 단, 서로 데이터가 겹치지 않도록 하여 $S_1, \dots, S_K$ 을 구성한다.	데이터 개수가 100개이고 5등분한다면, 각 표본의 개수는 20개
2	모든 후보 모형에 대하여 $S_j$ 을 제외한 나머지 표본의 합을 이용하여 모형 $\hat{f}_{ij}$ 을 얻고 일반화 오차를 얻는다. 이 과정을 모든 분할된 표본에 적용하고 난 후 각 일반화 오차의 평균을 구한다.	여기서 $i$ 는 후보 모형 색인번호, $j$ 는 검증 데이터 색인번호
3	일반화 오차의 평균 중에 가장 낮은 값을 주는 모형을 선택한다.	

# 변수 선택

## [포장 방법]

- 교차 검증
- K-곱 교차검증

폴더					모형	비용함수
1	2	3	4	5	$i = 1, \dots, p$	$i = 1, \dots, p$
					$\hat{f}_{i1}$	RSS_i1
					$\hat{f}_{i2}$	RSS_i2
					$\hat{f}_{i3}$	RSS_i3
					$\hat{f}_{i4}$	RSS_i4
					$\hat{f}_{i5}$	RSS_i5

# 변수 선택

## [포장 방법]

- 교차 검증
- K-곱 교차검증

```
In    # 필요한 패키지
      from sklearn.model_selection import KFold

      # 폴더 정의
      num_folds = 5
      kf = KFold(n_splits=num_folds, shuffle=True)
      X = scaled_df[xvars]
      y = scaled_df['medv']
      kf.get_n_splits(X, y)
```

```
Out   5
```



# 변수 선택

## [포장 방법]

- 교차 검증
- K-곱 교차검증

```
In      # 결과 배열 정의
        mm = np.zeros(shape=(num_folds, len(xvars)))

        # 각 폴더에 대하여
        for i, (train_index, valid_index) in enumerate(kf.split(X)):
            X_train, X_valid = X.iloc[train_index], X.iloc[valid_index]
            y_train, y_valid = y.iloc[train_index], y.iloc[valid_index]

            # 각 변수에 대하여
            for j in np.arange(len(xvars)):
                varname = xvars[j]
                fit = smf.ols('medv~' + varname, data=pd.concat([X_train, y_train], axis=1)).fit()
                pred = fit.predict(sm.add_constant(X_valid[varname]))
                tmp_rss = np.sum((y_valid.values - pred) ** 2)
                mm[i, j] = tmp_rss

        # 각 변수별로 잔차 제곱합 평균 구하기
        rss_mean = np.mean(mm, axis=0)
        np.round(rss_mean, 3)

Out      array([83.216, 87.343, 77.966, 53.82 , 46.447, 87.398, 95.538, 88.834,
               99.153, 79.35 , 75.676, 86.975, 90.431])
```

# 변수 선택

## [포장 방법]

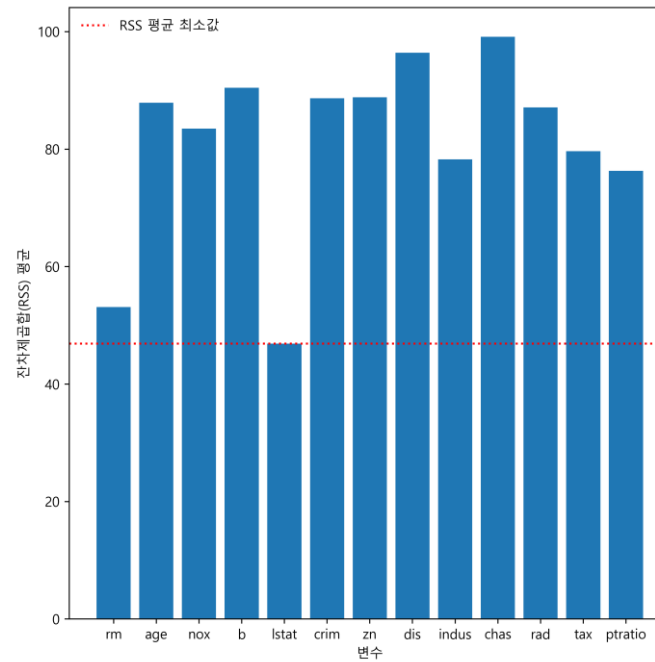
- 교차 검증
- K-곱 교차검증

In	# 최소값을 주는 변수 xvars[np.argmin(rss_mean)]
Out	'lstat'

# 변수 선택

## [포장 방법]

- 교차 검증
- K-곱 교차검증



# 변수 선택

## [후보 모형의 구성]

- 단계적 모형 선택(stepwise selection)
- 전진 단계 선택
- 후진 단계 제거
- 혼성 방법

# 변수 선택

## [후보 모형의 구성]

### ■ 전진 단계 선택

순서	내용	비고
1	초기화를 한다. 즉, $\mathcal{M}_0 = \emptyset$ .	변수를 1개도 사용하지 않은 경우를 일반적으로 널 모형(null model)이라 한다.
2	모든 $k = 0, \dots, p - 1$ 에 대하여 아래 과정을 반복한다. 1) 기존 모형인 $\mathcal{M}_k$ 에 변수를 1개씩 추가하여 $p - k$ 개의 모형을 구성한다. 2) 모형 중에서 훈련 데이터에서 가장 좋은 적합도를 주는 모형을 $\mathcal{M}_{k+1}$ 로 정의한다.	모형을 선택하는 기준(일종의 적합도)이 필요하다. 적합도는 RSS이나 $R^2$ 을 사용한다.
3	$\mathcal{M}_k$ 중에 교차 검증 결과 가장 좋은 적합도를 나타내는 모형을 최적 모형으로 선택하며, 그 모형에 포함된 변수를 유의미한 변수로 선정한다.	교차검증 방법을 사용하지 않은 경우에는 2의 결과 모형에 대하여 AIC, AICC, BIC 등을 적용하여 모형을 선택한다.

# 변수 선택

## [후보 모형의 구성]

- 전진 단계 선택

	$X_1$	$X_2$	$X_3$	$X_4$	$X_5$	$X_6$	$X_7$	$X_8$	$X_9$	$X_{10}$	$X_{11}$	$X_{12}$	$X_{13}$
$M_1$													
$M_2$													
$M_3$													
$M_4$													
$M_5$													
$M_6$													
$M_7$													
$M_8$													
$M_9$													
$M_{10}$													
$M_{11}$													
$M_{12}$													
$M_{13}$													

# 변수 선택

## [후보 모형의 구성]

- 전진 단계 선택

```
In      # 필요한 패키지
import itertools

from sklearn.model_selection import KFold
from sklearn import linear_model
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split

# 데이터 구성
xvars = list(set(df.columns) - {'medv'})

X = df[xvars]
y = df['medv']

# 변수의 개수
k = len(xvars)
k
Out      13
```

# 변수 선택

## [후보 모형의 구성]

- 전진 단계 선택

---

```
In    #
      # 전진 선택 알고리즘: 후보 모형 구성하기
      #

      # 남아있는 변수 리스트 및 선정된 변수 리스트 정의하기
      remaining_features = list(X.columns.values)
      features = []

      # 색인이 0 부터 시작하므로 처음 값을 무한대로 정의
      RSS_list, R_squared_list = [np.inf], [np.inf]

      # 특징 리스트는 사전으로 정의: 키는 변수의 개수
      features_list = dict()
```

---



# 변수 선택

## [후보 모형의 구성]

---

In # 알고리즘

```
# 변수의 개수 별로
for i in range(1, k + 1):
    best_RSS = np.inf

    # 남아 있는 변수 1개씩 추가
    for combi in itertools.combinations(remaining_features, 1):

        # 회귀 모형 적합
        testing_features = list(combi) + features
        fit = linear_model.LinearRegression(fit_intercept=True).fit(X[testing_features], y)
        RSS = mean_squared_error(y, fit.predict(X[testing_features])) * len(y)
        R_squared = fit.score(X[testing_features], y)

        if RSS < best_RSS:
            best_RSS = RSS
            best_R_squared = R_squared
            # 튜플이므로 앞의 1개
            best_feature = combi[0]

    # 결과 갱신
    features.append(best_feature)
    remaining_features.remove(best_feature)

    # 결과 저장
    RSS_list.append(best_RSS)
    R_squared_list.append(best_R_squared)
    features_list[i] = features.copy()
```

# 변수 선택

## [후보 모형의 구성]

- 전진 단계 선택

```
In    # 결과 보기
      [(features_list[i], RSS_list[i].round(1)) for i in np.arange(1, 5)]
Out   [(['lstat'], 19472.4),
      (['lstat', 'rm'], 15439.3),
      (['lstat', 'rm', 'ptratio'], 13728.0),
      (['lstat', 'rm', 'ptratio', 'dis'], 13228.9)]
```

# 변수 선택

## [후보 모형의 구성]

- 전진 단계 선택

In	<pre># 결과 데이터 구성 rdf = pd.concat([pd.DataFrame({'features':features_list}),                   pd.DataFrame({'RSS':RSS_list, 'R_squared': R_squared_list})],                   axis=1, join='inner') rdf['num_features'] = rdf.index print(rdf.head(3))</pre>				
Out	features	RSS	R_squared	num_features	
1	[lstat]	19472.381418	0.544146	1	
2	[lstat, rm]	15439.309201	0.638562	2	
3	[lstat, rm, ptratio]	13727.985314	0.678624	3	

# 변수 선택

## [모형의 선택]

- 전진 단계 선택
- 벌점이 부여된 모형 선택 기준 통계량

모형 선택 기준	계산식
AIC (Akaike information criterion)	$n \log \left( \frac{\text{SSE}}{n} \right) + 2p + n + 2$
BIC, SBC (Bayesian information criterion, Schwarz criterion)	$n \log \left( \frac{\text{SSE}}{n} \right) + p \log(n)$
AdjRSQ	$1 - \frac{(n-1)(1-R^2)}{n-p}$

# 변수 선택

## [모형의 선택]

- 전진 단계 선택
- 벌점이 부여된 모형 선택 기준 통계량

```
In # 벌점화된 통계량에 의한 변수 선택

# 결과 데이터로부터 AIC, BIC and R-square adjusted 계산
p = len(xvars) + 1
n = len(y)

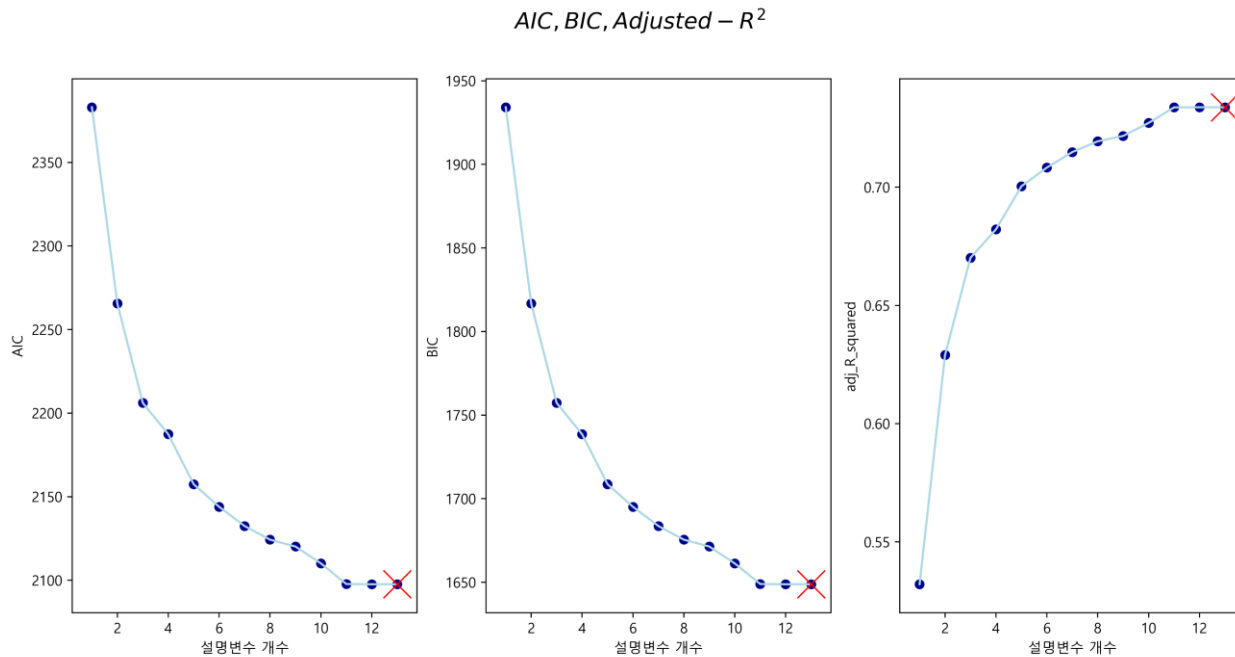
rdf['AIC'] = n * np.log(rdf['RSS']/n) + 2*p + n + 2
rdf['BIC'] = n * np.log(rdf['RSS']/n) + p*np.log(n)
rdf['adj_R_squared'] = 1 - (n-1)*(1-rdf['R_squared'])/(n-p)
print(rdf.head(3))
```

Out	features	RSS	R_squared	num_features	AIC	\
1	[lstat]	19472.381418	0.544146	1	2383.009161	
2	[lstat, rm]	15439.309201	0.638562	2	2265.576519	
3	[lstat, rm, ptratio]	13727.985314	0.678624	3	2206.131472	
	BIC	adj_R_squared				
1	1934.180675	0.532101				
2	1816.748032	0.629011				
3	1757.302985	0.670133				

# 변수 선택

## [모형의 선택]

- 전진 단계 선택
- 벌점이 부여된 모형 선택 기준 통계량



# 변수 선택

## [모형의 선택]

- 전진 단계 선택
- 교차 검증

```
In      # 교차 검증

        # 폴더 정의
        num_folds = 5
        kf = KFold(n_splits=num_folds, shuffle=True)
        X = df[xvars]
        y = df['medv']

        # 폴더 갯수
        kf.get_n_splits(X, y)

Out      5
```

# 변수 선택

## [모형의 선택]

---

```
In      # 교차 검증 알고리즘

        # 결과 행렬 정의
        result_list = []

        # 각 폴더에 대하여
        for i, (train_index, valid_index) in enumerate(kf.split(X)):
            X_train, X_valid = X.iloc[train_index], X.iloc[valid_index]
            y_train, y_valid = y.iloc[train_index], y.iloc[valid_index]
            tmp_list = []
            # 각 변수 그룹에 대하여
            for j in np.arange(len(rdf)):
                vars = rdf.iloc[j]['features']
                fit = linear_model.LinearRegression(fit_intercept=True).fit(X_train[vars], y_train)
                RSS = mean_squared_error(y_valid, fit.predict(X_valid[vars])) * len(y_valid)
                R_squared = fit.score(X_valid[vars], y_valid)
                tmp_list.append(RSS)
            result_list.append(tmp_list)

        result_list = np.array(result_list)

        # 각 변수별로 잔차 제곱합 평균 구하기
        rss_mean = np.nanmean(result_list, axis=0)

        # 최소값을 주는 변수 목록
        min_index = np.nanargmin(rss_mean)
        rdf.iloc[min_index]['features']
```

---



# 변수 선택

## [모형의 선택]

---

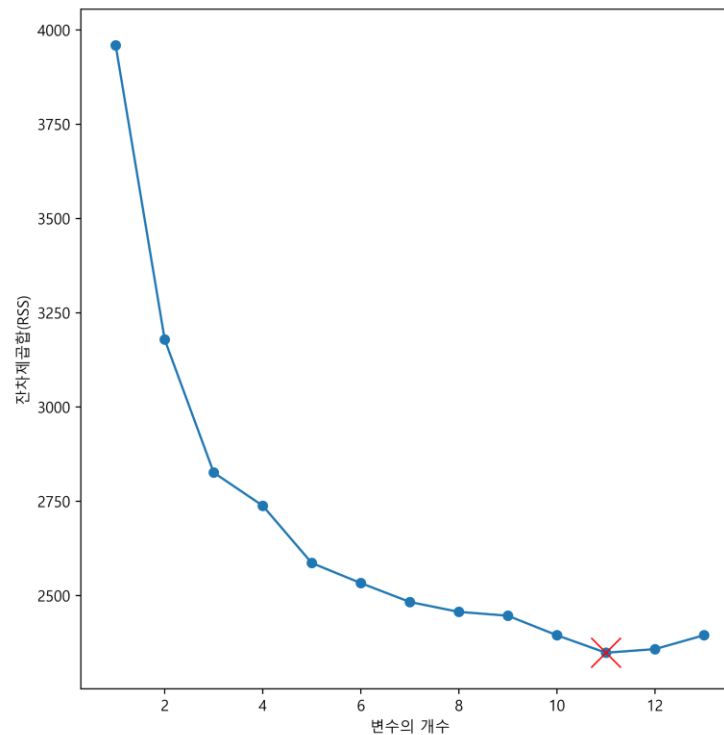
Out	['lstat', 'rm', 'ptratio', 'dis', 'nox', 'chas', 'b', 'zn', 'crim', 'rad', 'tax']
-----	---

---

# 변수 선택

## [모형의 선택]

- 전진 단계 선택
- 교차 검증



# 변수 선택

## [후보 모형의 구성]

### ■ 후진 단계 제거

순서	내용	비고
1	초기화를 한다. 즉, $\mathcal{M}_p$ 을 모든 변수를 이용한 모형으로 정의한다.	$\mathcal{M}_p$ 을 완전 모형(full model) 이라 한다.
2	모든 $k = p, p-1, \dots, 0$ 에 대하여 아래 과정을 반복한다. 1) 기존 모형인 $\mathcal{M}_k$ 에 변수를 1개씩 제거하여 $k$ 개의 모형을 구성한다. 2) 모형 중에서 훈련 데이터에서 가장 좋은 적합도를 주는 모형을 $\mathcal{M}_{k-1}$ 로 정의한다.	모형을 선택하는 기준(일종의 적합도)이 필요하다. 적합도는 RSS이나 $R^2$ 을 사용한다.
3	$\mathcal{M}_k$ 중에 교차 검증 결과 가장 좋은 적합도를 나타내는 모형을 최고의 모형으로 선택하며, 그 모형에 포함된 변수를 유의미한 변수로 선정한다.	교차검증 방법을 사용하지 않은 경우에는 2의 결과 모형에 대하여 AIC, AICC, BIC 등을 적용하여 모형을 선택한다.

# 변수 선택

## [내장 방법]

- 능형 회귀
- 사전에 데이터 표준화가 필수

# 변수 선택

## [내장 방법]

**예제** [HOUSING] 자료에서 모든 변수에 대하여 다음과 같은 표준화를 해보자.

$$z_i = \frac{x_i - \bar{x}}{\sqrt{\frac{\sum_i (x_i - \bar{x})^2}{n - 1}}}$$

# 변수 선택

## [내장 방법]

- 능형 회귀
- 사전에 데이터 표준화가 필수

```
In    # 능형회귀, 라쏘회귀 적합을 위한 데이터 전 처리
      # 변수 리스트 구성하기
      target_var = "medv"
      input_vars = list(df.columns)
      input_vars.remove(target_var)

      # 표준화
      sdf = (df - np.mean(df)) / np.sqrt(np.var(df, ddof=1))

      # X, y 분리
      X = sdf[input_vars].values
      y = sdf[target_var].values
```

Out

# 변수 선택

## [내장 방법]

- 능형 회귀
- 비용 함수

$$J(\beta) = \sum_{i=1}^n \left( y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^p \beta_j^2 = \text{RSS} + \lambda \sum_{j=1}^p \beta_j^2$$

$\lambda \geq 0$ 는 조율모수(tuning parameter)

# 변수 선택

## [내장 방법]

- 능형 회귀
- 람다 값에 따라 후보 모형 생성
- 각 후보 모형에 대하여 교차 검증 방법으로 모형 선택



# 변수 선택

## [내장 방법]

**예제** [HOUSING] 자료에서 주택중위가격(medv)를 종속변수, 나머지 모든 변수를 독립변수로 하여 능형회귀를 적합 시킬 때 람다( $\lambda$ ) 값에 따른 비용함수의 변화를 5-겹 교차검증 방법에 의해 살펴보자.

# 변수 선택

## [내장 방법]

**예제** [HOUSING] 자료에서 주택중위가격(medv)를 종속변수, 나머지 모든 변수를 독립변수로 하여 능형회귀를 적합 시킬 때 람다( $\lambda$ ) 값에 따른 비용함수의 변화를 5-겹 교차검증 방법에 의해 살펴보자.

```
In    # 추정해야 할 모수 (절편 제외)
      nparams = len(input_vars)

      # 람다 값의 범위
      lambdas = np.logspace(-8.5, 6.5, 100, base=np.e)

      # 모수 값을 넣기 위한 행렬 생성: 하나의 열이 동일한 람다 값
      coefficients = np.zeros(shape=(nparams, len(lambdas)))

      # 비용 값을 넣기 위한 벡터 생성
      costs = np.zeros_like(lambdas, float)

      # 교차 검증에 의한 비용 계산
      num_folders = 5
```

# 변수 선택

## [내장 방법]

```
In      for i, l in enumerate(lambdas):
        # 람다 값에 따른 모형의 구성
        # 행별로 데이터 표준화: normalize=True
        ridge = Ridge(alpha=l, normalize=True)
        ridge.fit(X, y)
        coefficients[:, i] = ridge.coef_

        # 각 람다 값에 따른 교차 검증에 의한 비용함수의 계산
        fold_costs = []
        kf = KFold(n_splits=num_folds, shuffle=True, random_state=123)
        for train_index, valid_index in kf.split(X):
            X_train, X_valid = X[train_index], X[valid_index]
            y_train, y_valid = y[train_index], y[valid_index]

            # 각 폴더에 대한 모형 적합
            # 행별로 데이터 표준화: normalize=True
            ridge = Ridge(alpha=l, normalize=True)
            ridge.fit(X_train, y_train)

            # 비용 계산
            pred = ridge.predict(X_valid)
            cost = np.sum((pred - y_valid)**2) + l*np.sum(ridge.coef_**2)
            fold_costs.append(cost)

        # 평균 비용 계산
        costs[i] = np.mean(fold_costs)
```

# 변수 선택

## [내장 방법]

```
In    # 비용을 최소화 하는 인덱스
      min_index = np.argmin(costs)
      min_index
```

```
In    # 람다 값
      lambdas[min_index].round(5)
```

```
Out    0.01917
```

```
In    # 로그 람다 값
      np.log(lambdas[min_index]).round(3)
```

```
Out    -3.955
```

# 변수 선택

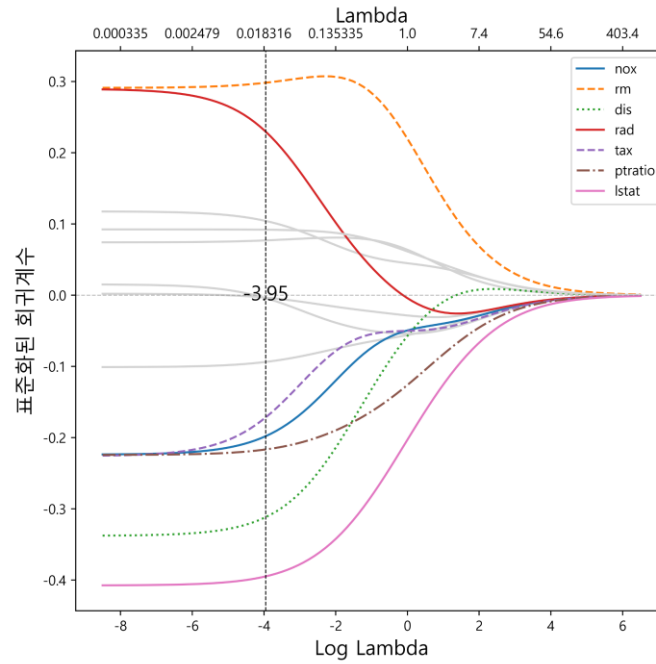
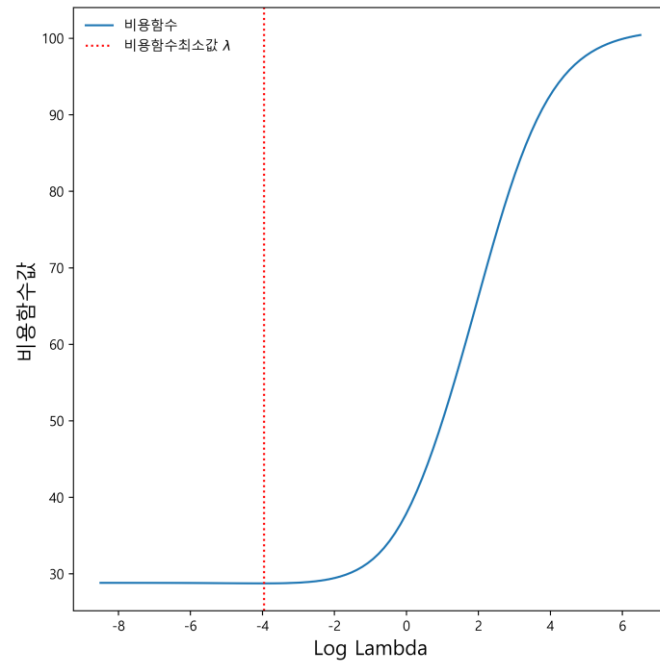
## [내장 방법]

```
In      # 최소값을 줄 때의 표준화된 회귀계수 값들
        res = pd.DataFrame({'Variable':input_vars, 'Coefficient':coefficients[:, min_index]})
        print(res.round(3))
```

Out	Variable	Coefficient
0	crim	-0.094
1	zn	0.104
2	indus	-0.004
3	chas	0.077
4	nox	-0.198
5	rm	0.298
6	age	-0.003
7	dis	-0.312
8	rad	0.230
9	tax	-0.172
10	ptratio	-0.217
11	b	0.092
12	lstat	-0.395

# 변수 선택

## [내장 방법]



# 변수 선택

## [내장 방법]

- 라쏘 회귀
- 비용 함수

$$J(\beta) = \sum_{i=1}^n \left( y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^p |\beta_j| = \text{RSS} + \lambda \sum_{j=1}^p |\beta_j|$$

$\lambda \geq 0$ 는 조율모수(tuning parameter)

# 변수 선택

## [내장 방법]

- 라쏘 회귀
- 람다 값에 따라 후보 모형 생성
- 각 후보 모형에 대하여 교차 검증 방법으로 모형 선택



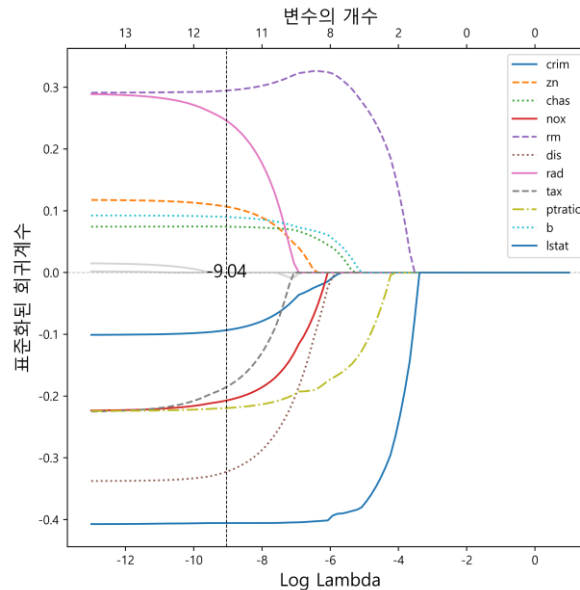
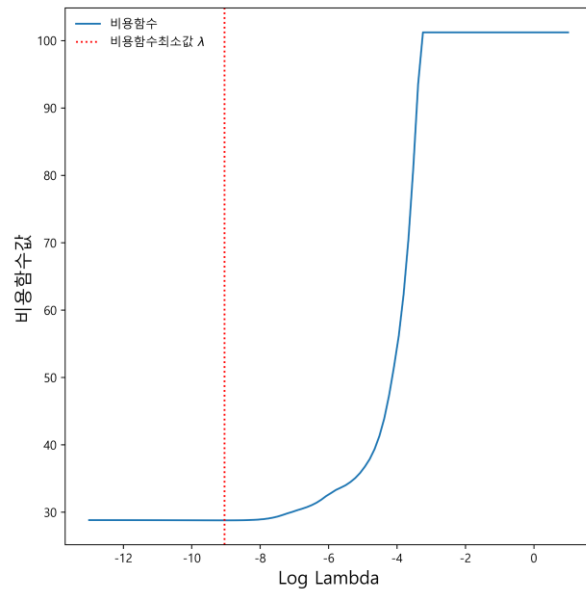
# 변수 선택

## [내장 방법]

```
In      # 람다 값
min_lambda = lambdas[min_index]
min_lambda

# 로그 단위로 변환
min_log_lambda = np.log(min_lambda)

np.round([min_lambda,min_log_lambda], 5)
Out      array([ 1.2000e-04, -9.0404e+00])
```



# 변수 선택

## [내장 방법]

```
In      # 최소값을 줄 때의 표준화된 회귀계수 값들
        res = pd.DataFrame({'Variable':input_vars, 'Coefficient':coefficients[:, min_index]})
        print(res)
```

Out	Variable	Coefficient
0	crim	-0.093380
1	zn	0.106664
2	<b>indus</b>	<b>0.000000</b>
3	chas	0.074225
4	nox	-0.207252
5	rm	0.294648
6	<b>age</b>	<b>-0.000000</b>
7	dis	-0.322610
8	rad	0.245893
9	tax	-0.185416
10	ptratio	-0.219464
11	b	0.089947
12	lstat	-0.405691

# 변수 선택

## [내장 방법]

**과제** [HOUSING] 자료에서 주택중위가격(medv)를 종속변수, 나머지 모든 변수를 독립변수로 하여 라쏘회귀를 적합 시킬 때 람다( $\lambda$ ) 값에 따른 비용함수의 변화를 5-겹 교차검증 방법에 의해 살펴보자.