데이터 전처리,
Feature Selection (모델 특성 선택)
및 Cross Validation (교차 검증)

Kyungsik Han

본 영상에서 배울 내용

CODING

- 데이터 스케일링이란(data scaling)
- 특성 선택(feature selection)

왜 데이터 스케일링?

CODING ✖

- 일부 알고리즘은 데이터 스케일에 민감함
  - Distance, Similarity-based algorithms
    - k-NN, SVM
- 일부 알고리즘은 괜찮지만, scale 해주는 것이 일반적으로 좋음
  - Decision trees, Random Forest

스케일링 방법

CODING

- StandardScaler
  - 평균 0, 분산 1로 변경
  - 모든 특성이 같은 크기를 가짐
- MinMaxScaler
  - 모든 특성이 0과 1 사이에 위치하도록 변경
- Normalizer
  - 행(데이터포인트)마다 각각 정규화

스케일링 방법

CODING

In [2]:
```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
cancer = load_breast_cancer()

X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target, random_state=1)
print(X_train.shape)
print(X_test.shape)
```

```
(426, 30)
(143, 30)
```

In [3]:
```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
```

In [4]:
```
scaler.fit(X_train)
```

Out[4]:
```
MinMaxScaler(copy=True, feature_range=(0, 1))
```

In [5]:
```
# 데이터 변환
X_train_scaled = scaler.transform(X_train)
```

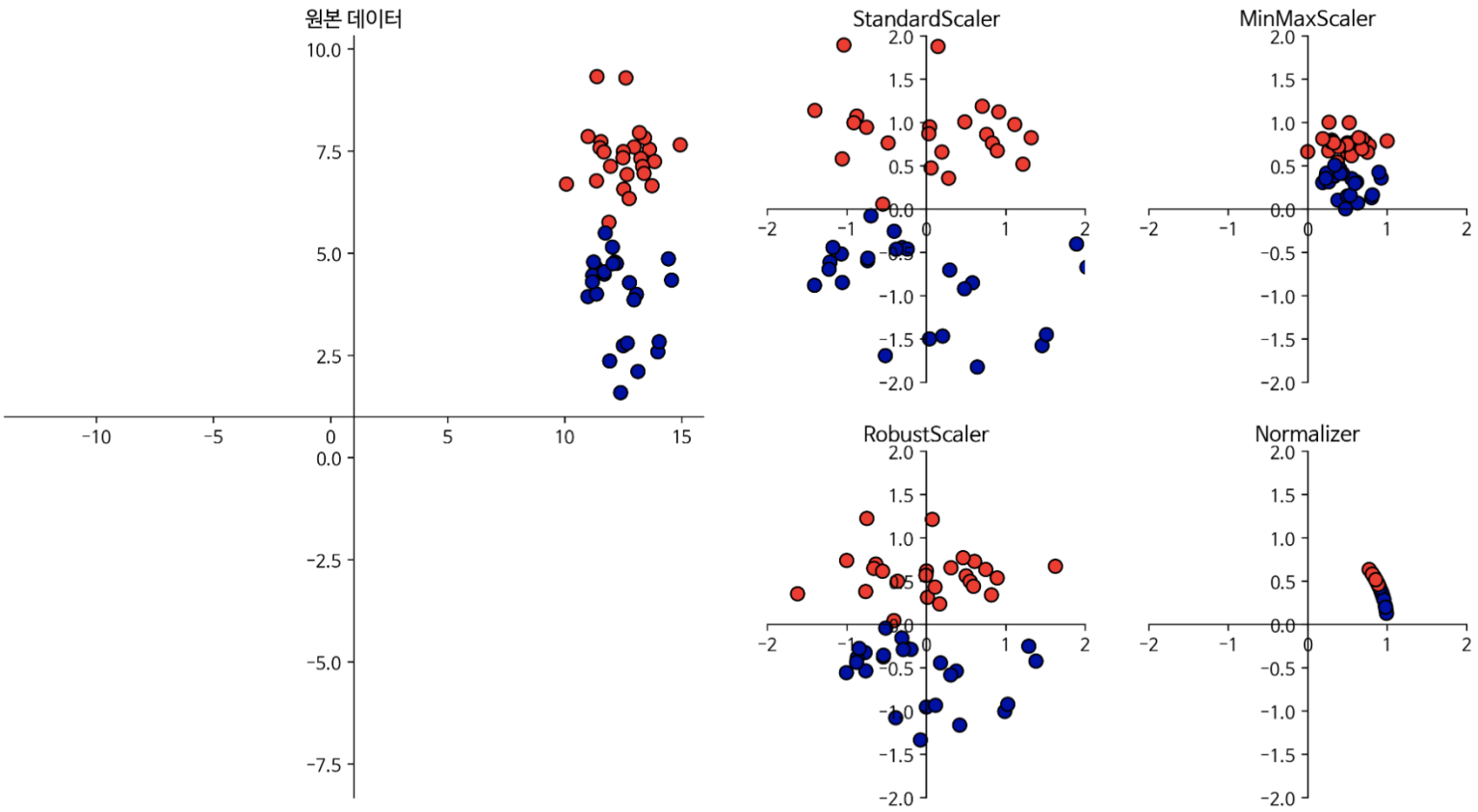
스케일링 방법

CODING

In [6]:
```
# 스케일이 조정된 후 데이터셋 속성 출력
# 스케일이 조정된 후 데이터셋의 속성을 출력합니다
print("변환된 후 크기: {}".format(X_train_scaled.shape))
print("스케일 조정 전 특성별 최소값:\n {}".format(X_train.min(axis=0)))
print("스케일 조정 전 특성별 최대값:\n {}".format(X_train.max(axis=0)))
print("스케일 조정 후 특성별 최소값:\n {}".format(X_train_scaled.min(axis=0)))
print("스케일 조정 후 특성별 최대값:\n {}".format(X_train_scaled.max(axis=0)))
```

```
변환된 후 크기: (426, 30)
스케일 조정 전 특성별 최소값:
 [  6.98100000e+00   9.71000000e+00   4.37900000e+01   1.43500000e+02
   5.26300000e-02   1.93800000e-02   0.00000000e+00   0.00000000e+00
   1.06000000e-01   5.02400000e-02   1.15300000e-01   3.60200000e-01
   7.57000000e-01   6.80200000e+00   1.71300000e-03   2.25200000e-03
   0.00000000e+00   0.00000000e+00   9.53900000e-03   8.94800000e-04
   7.93000000e+00   1.20200000e+01   5.04100000e+01   1.85200000e+02
   7.11700000e-02   2.72900000e-02   0.00000000e+00   0.00000000e+00
   1.56600000e-01   5.52100000e-02]
스케일 조정 전 특성별 최대값:
 [  2.81100000e+01   3.92800000e+01   1.88500000e+02   2.50100000e+03
   1.63400000e-01   2.86700000e-01   4.26800000e-01   2.01200000e-01
   3.04000000e-01   9.57500000e-02   2.87300000e+00   4.88500000e+00
   2.19800000e+01   5.42200000e+02   3.11300000e-02   1.35400000e-01
   3.96000000e-01   5.27900000e-02   6.14600000e-02   2.98400000e-02
   3.60400000e+01   4.95400000e+01   2.51200000e+02   4.25400000e+03
   2.22600000e-01   9.37900000e-01   1.17000000e+00   2.91000000e-01
   5.77400000e-01   1.48600000e-01]
스케일 조정 후 특성별 최소값:
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
스케일 조정 후 특성별 최대값:
 [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.
  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
```

스케일링 예제


원본 데이터
StandardScaler
MinMaxScaler
RobustScaler
Normalizer

스케일링 효과

CODING

In [8]:
```
from sklearn.svm import SVC

X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target,
                                                    random_state=0)

svm = SVC(C=100)
svm.fit(X_train, y_train)
print("테스트 세트 정확도: {:.2f}".format(svm.score(X_test, y_test)))
```

```
테스트 세트 정확도: 0.63
```

In [9]:
```
# 0~1 사이로 스케일 조정
scaler = MinMaxScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 조정된 데이터로 SVM 학습
svm.fit(X_train_scaled, y_train)

# 스케일 조정된 테스트 세트의 정확도
print("스케일 조정된 테스트 세트의 정확도: {:.2f}".format(svm.score(X_test_scaled, y_test)))
```

```
스케일 조정된 테스트 세트의 정확도: 0.97
```

스케일링 효과

CODING

In [10]:
```
# 평균 0, 분산 1을 갖도록 스케일 조정
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 조정된 데이터로 SVM 학습
svm.fit(X_train_scaled, y_train)

# 스케일 조정된 테스트 세트의 정확도
print("SVM test accuracy: {:.2f}".format(svm.score(X_test_scaled, y_test)))
```

```
SVM test accuracy: 0.96
```

특성 선택

CODING

- 새로운 특성(feature)이 만들어지는 다양한 방법으로 인해 특성 수가 증가할 수 있음

- 특성이 추가되면 모델은 더욱 복잡해지고 과대적합(overfitting)될 가능성이 높아짐

- 가장 유용한 특성만 고려해서 모델 구축에 사용되면 모델이 간단해지고 일반화 성능이 올라가는 효과를 볼 수 있음

특성 선택

CODING ✖

- 네 가지 방법에 대해서 학습
  - 일변량 통계 (univariate statistics)
  - 모델 기반 선택 (model-based selection)
  - 반복적 선택 (iterative selection)
  - 차원 축소 (dimensionality reduction)

일변량 통계

CODING

- 각 특성과 target 사이에 중요한 통계적 관계가 있는지 계산
- 깊게 관련되어 있는 특성만 선택
- 핵심요소: 각 특성이 독립적으로 평가됨

일변량 통계

CODING

특성 자동 선택

단변량 통계

In [12]:
```
%matplotlib inline
import matplotlib.pyplot as plt
```

In [13]:
```
from sklearn.datasets import load_breast_cancer
from sklearn.feature_selection import SelectPercentile, f_classif
from sklearn.model_selection import train_test_split
import numpy as np

cancer = load_breast_cancer()

# 고정된 난수를 발생시킵니다
rng = np.random.RandomState(42)
noise = rng.normal(size=(len(cancer.data), 50))
# 데이터에 노이즈 특성을 추가합니다
# 처음 30개는 원본 특성이고 다음 50개는 노이즈입니다
X_w_noise = np.hstack([cancer.data, noise])

X_train, X_test, y_train, y_test = train_test_split(
    X_w_noise, cancer.target, random_state=0, test_size=.5)
# f_classif(기본값)와 SelectPercentile을 사용하여 특성의 50%를 선택합니다
select = SelectPercentile(score_func=f_classif, percentile=50)
select.fit(X_train, y_train)
# 훈련 세트에 적용합니다
X_train_selected = select.transform(X_train)

print("X_train.shape: {}".format(X_train.shape))
print("X_train_selected.shape: {}".format(X_train_selected.shape))
```

```
X_train.shape: (284, 80)
X_train_selected.shape: (284, 40)
```

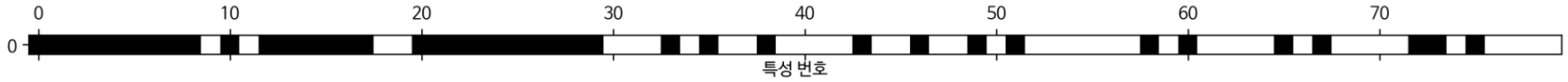
일변량 통계

CODING

In [41]:
```
mask = select.get_support()
print(mask)
# True는 검은색, False는 흰색으로 마스킹합니다
plt.matshow(mask.reshape(1, -1), cmap='gray_r')
plt.xlabel("특성 번호")
plt.yticks([0])
```

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

Out[41]:
```
([<matplotlib.axis.YTick at 0x1c2938e2b0>],
 <a list of 1 Text yticklabel objects>)
```



일변량 통계

CODING ✖

In [42]:
```
from sklearn.linear_model import LogisticRegression

# 테스트 데이터 변환
X_test_selected = select.transform(X_test)

lr = LogisticRegression()
lr.fit(X_train, y_train)
print("전체 특성을 사용한 점수: {:.3f}".format(lr.score(X_test, y_test)))
lr.fit(X_train_selected, y_train)
print("선택된 일부 특성을 사용한 점수: {:.3f}".format(
        lr.score(X_test_selected, y_test)))
```

```
전체 특성을 사용한 점수: 0.930
선택된 일부 특성을 사용한 점수: 0.940
```

모델 기반 특성 선택

- 지도학습 머신러닝 모델을 사용하여 특성의 중요도를 평가
- 가장 중요한 특성들만 선택
- 특성 선택을 위한 모델은 각 특성의 중요도를 측정하여 순서를 매김
- 결정 트리 사용: 결정 트리와 이를 기반으로 한 모델은 각 특성의 중요도가 담겨있는 feature_importance_ 속성을 제공
- 모델 기반의 특성 선택은 SelectFromModel에 구현되어 있음

모델 기반 특성 선택

CODING

모델 기반 특성 선택

In [43]:
```
from sklearn.feature_selection import SelectFromModel
from sklearn.ensemble import RandomForestClassifier
select = SelectFromModel(
    RandomForestClassifier(n_estimators=100, random_state=42),
    threshold="median")
```

In [44]:
```
select.fit(X_train, y_train)
X_train_l1 = select.transform(X_train)
print("X_train.shape: {}".format(X_train.shape))
print("X_train_l1.shape: {}".format(X_train_l1.shape))
```

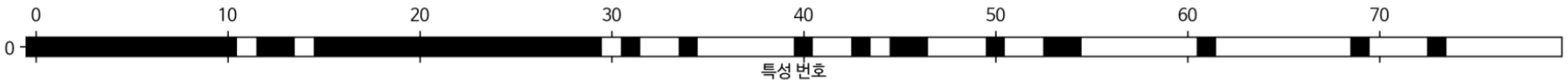
```
X_train.shape: (284, 80)
X_train_l1.shape: (284, 40)
```

모델 기반 특성 선택

CODING

In [45]:
```
mask = select.get_support()
# True는 검은색, False는 흰색으로 마스킹합니다
plt.matshow(mask.reshape(1, -1), cmap='gray_r')
plt.xlabel("특성 번호")
plt.yticks([0])
```

Out[45]:
```
([<matplotlib.axis.YTick at 0x1c29881240>],
 <a list of 1 Text yticklabel objects>)
```



모델 기반 특성 선택

CODING

In [46]:
```
X_test_l1 = select.transform(X_test)
score = LogisticRegression().fit(X_train_l1, y_train).score(X_test_l1, y_test)
print("Test score: {:.3f}".format(score))
```

```
Test score: 0.951
```

반복적 특성 선택

- 특성의 수가 각기 다른 일련의 모델이 생성됨
- 기본적으로 두 가지 방법
  - 1) 특성을 하나도 선택하지 않은 상태에서 시작하여 어떤 종료 조건에 도달할 때까지 진행
  - 2) 모든 특성을 가지고 시작해서 어떤 종료 조건이 될 때까지 특성을 하나씩 제거해가는 방법: 재귀적 특성 제거 (RFE, Recursive Feature Elimination)
- 단점: 계산 비용이 많이 들 수 있음

반복적 특성 선택

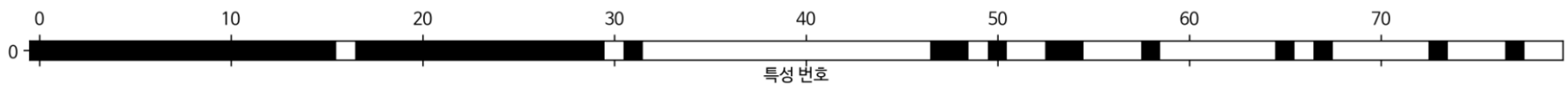
CODING

반복적 특성 선택

In [47]:
```
from sklearn.feature_selection import RFE
select = RFE(RandomForestClassifier(n_estimators=100, random_state=42),
             n_features_to_select=40)

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")
plt.yticks([0])
```

Out[47]:
```
([<matplotlib.axis.YTick at 0x1c298afb00>],
 <a list of 1 Text yticklabel objects>)
```



반복적 특성 선택

CODING

In [48]:
```
X_train_rfe = select.transform(X_train)
X_test_rfe = select.transform(X_test)

score = LogisticRegression().fit(X_train_rfe, y_train).score(X_test_rfe, y_test)
print("테스트 점수: {:.3f}".format(score))
```

```
테스트 점수: 0.951
```

In [49]:
```
print("테스트 점수: {:.3f}".format(select.score(X_test, y_test)))
```

```
테스트 점수: 0.951
```

차원 축소

CODING

- 주어진 데이터의 잠재 공간을 파악하고, 데이터 간의 연관 관계를 최대한 반영하면서 차원을 축소
- 단순히 데이터 압축 및 잡음을 제거하는 것이 아님을 기억
- 차원의 원본 feature 공간을 저차원(low dimension)의 새로운 feature 공간으로 투영시킴
- 새롭게 구성된 feature 공간은 보통은 원본 feature 공간의 선형 또는 비선형 결합

차원 축소

CODING

- 예로는
  - Principle Component Analysis (PCA)
  - Linear Discriminant Analysis (LDA)
  - Canonical Correlation Analysis (CCA)
- 등이 있음
- PCA 의 장점은 직접 dimension (차원) 수를 정할 수 있음

차원 축소

CODING ✖

차원 축소

In [23]:
```
from sklearn.decomposition import PCA
```

In [45]:
```
pca = PCA(n_components=40)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.fit_transform(X_test)
```

In [46]:
```
score = LogisticRegression().fit(X_train_pca, y_train).score(X_test_pca, y_test)
print("테스트 점수: {:.3f}".format(score))
```

```
테스트 점수: 0.793
```

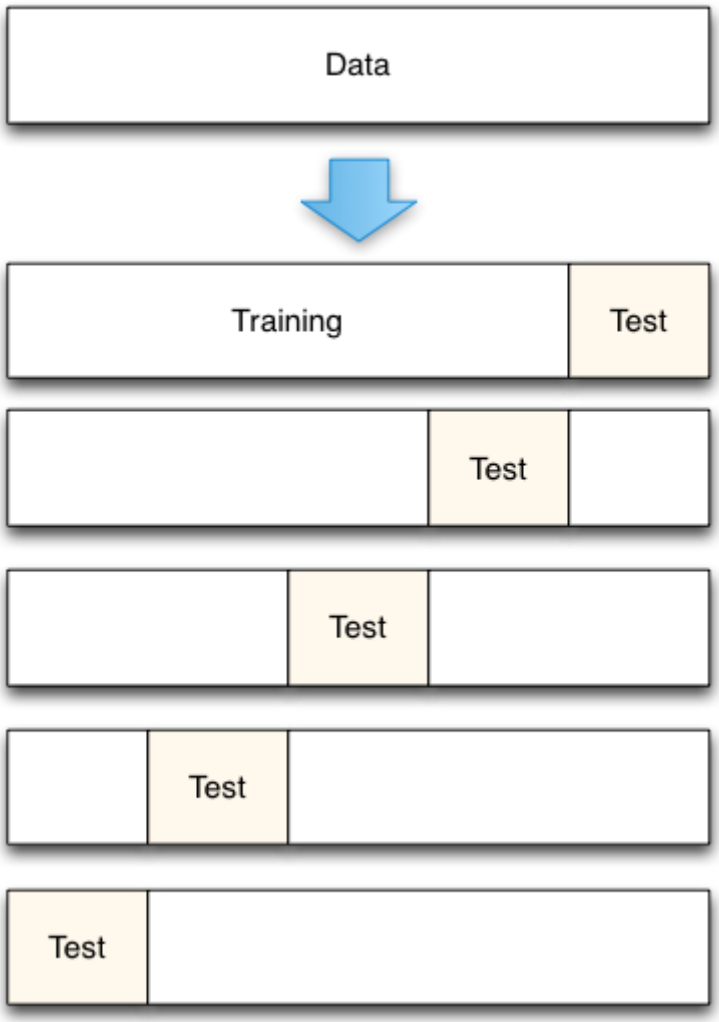
Cross Validation (교차 검증)

교차 검증

CODING

- 일반화 성능 측정을 위해 훈련 셋과 테스트 셋을 여러 번 반복해서 나누어 여러 모델을 학습
- 각 학습된 모델을 평균 내어 평가
- k-Fold Cross Validation

교차 검증



5-Fold Cross Validation

교차 검증

CODING

Cross Validation in sklearn

In [7]:
```
from sklearn.model_selection import cross_val_score
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
```

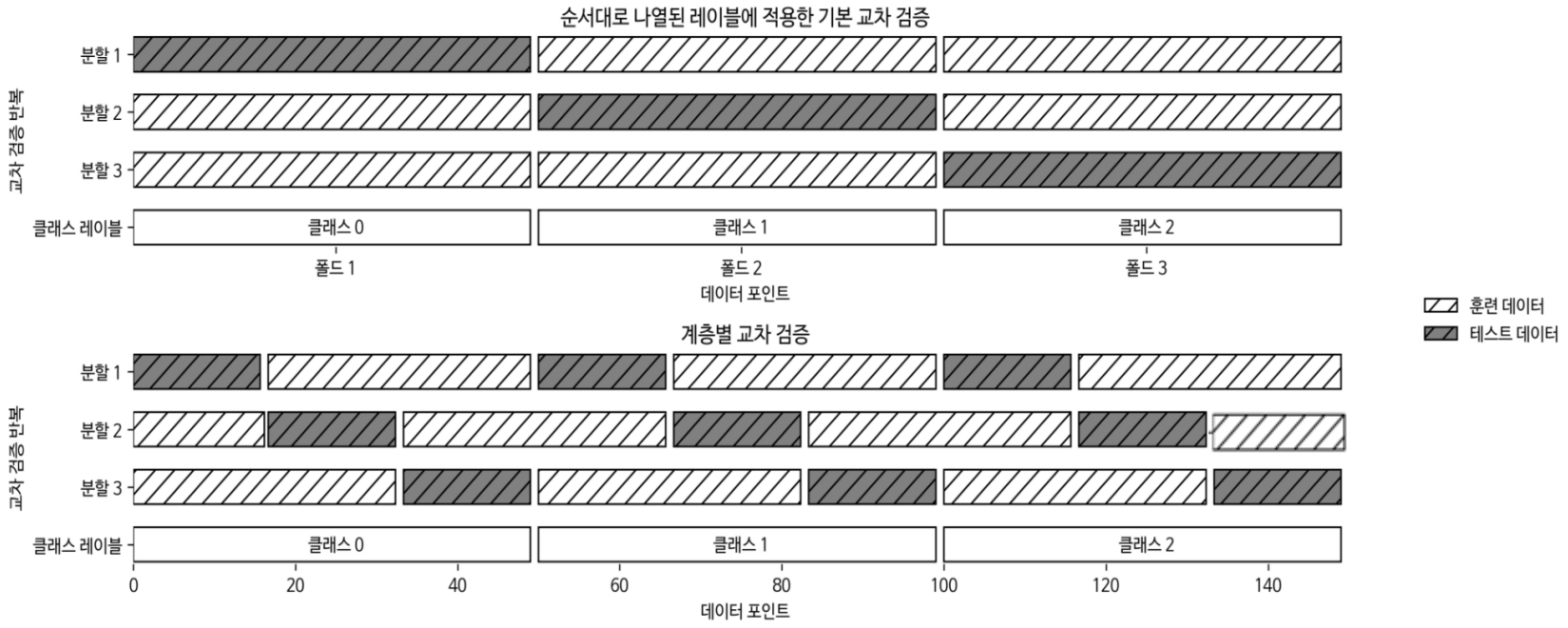
In [8]:
```
iris = load_iris()
logreg = LogisticRegression()
```

In [11]:
```
scores = cross_val_score(logreg, iris.data, iris.target, cv=10)
print("교차 검증 점수: {}".format(scores))
```

```
교차 검증 점수: [ 1.          1.          1.          0.93333333  0.93333333  0.93333333
  0.8         0.93333333  1.          1.        ]
```

교차 검증

- Stratified k-fold cross validation
  - 단순 k-fold cross validation 은 문제가 있을 수 있음
  - Fold 안의 클래스 비율이 전체 데이터셋의 클래스 비율과 같도록 데이터를 나눔



다음 영상에서 배울 내용

CODING X

- 데이터 스케일링, 특성 선택 실습

수고하셨습니다