supervised-learning_exercise

May 30, 2019

```
In [1]: # %load_ext watermark
        # %watermark -v -p sklearn,numpy,scipy,matplotlib

In [2]: #  - (binary classification) / (multiclass classification)
        #  - /

In [3]: #  -       ,  ,
        #                  .
        #  -   ,
        #  ->      , (generation) -    .

In [1]: import sklearn

In [2]: from sklearn import datasets

In [3]: sklearn.__version__

Out[3]: '0.20.3'

In [4]: from IPython.display import display
        import numpy as np
        import pandas as pd
        import mglearn

        import matplotlib.pyplot as plt
        %matplotlib inline
        # from preamble import *

In [5]: import mglearn

In [6]: import platform

        from matplotlib import font_manager, rc
        plt.rcParams['axes.unicode_minus'] = False

        if platform.system() == 'Darwin':
            rc('font', family='AppleGothic')
        elif platform.system() == 'Windows':
```

```
        path = "c:/Windows/Fonts/malgun.ttf"
        font_name = font_manager.FontProperties(fname=path).get_name()
        rc('font', family=font_name)
    else:
        print('Unknown system... sorry~~~~')
```

0.1

0.1.1

0.1.2

```
In [7]: plt.rcParams['figure.dpi'] = 80
        #   -
        # x   , y
        X, y = mglearn.datasets.make_forge()
        #
        print(X)
        print('-------------------')
        print(y)
```

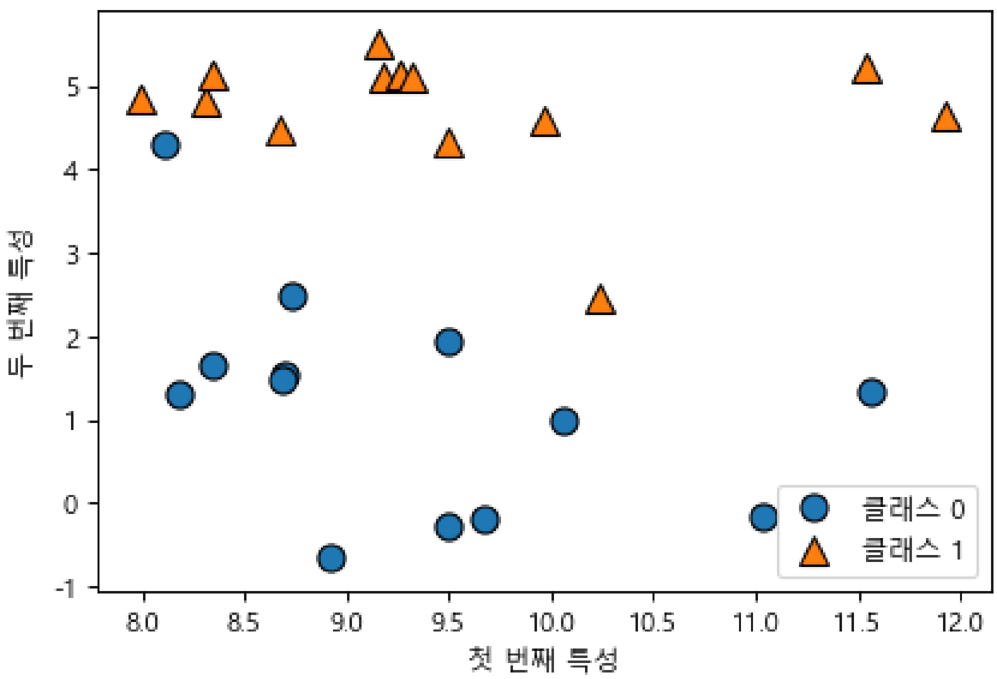
```
[[ 9.96346605  4.59676542]
 [11.0329545  -0.16816717]
 [11.54155807  5.21116083]
 [ 8.69289001  1.54322016]
 [ 8.1062269   4.28695977]
 [ 8.30988863  4.80623966]
 [11.93027136  4.64866327]
 [ 9.67284681 -0.20283165]
 [ 8.34810316  5.13415623]
 [ 8.67494727  4.47573059]
 [ 9.17748385  5.09283177]
 [10.24028948  2.45544401]
 [ 8.68937095  1.48709629]
 [ 8.92229526 -0.63993225]
 [ 9.49123469  4.33224792]
 [ 9.25694192  5.13284858]
 [ 7.99815287  4.8525051 ]
 [ 8.18378052  1.29564214]
 [ 8.7337095   2.49162431]
 [ 9.32298256  5.09840649]
 [10.06393839  0.99078055]
 [ 9.50048972 -0.26430318]
 [ 8.34468785  1.63824349]
 [ 9.50169345  1.93824624]
 [ 9.15072323  5.49832246]
 [11.563957    1.3389402 ]]
-------------------
```

```
[1 0 1 0 0 1 1 0 1 1 1 1 0 0 1 1 1 0 0 1 0 0 0 0 1 0]
```

```
C:\Users\User\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:77: DeprecationWarning:
  warnings.warn(msg, category=DeprecationWarning)
```

```
In [9]: #
        mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
        plt.legend([" 0", " 1"], loc=4)
        plt.xlabel("  ")
        plt.ylabel("  ")
        print("X.shape: {}".format(X.shape))
```

```
X.shape: (26, 2)
```




```
In [10]: #
         X, y = mglearn.datasets.make_wave(n_samples=40)
         print(X)
         print(y)
```

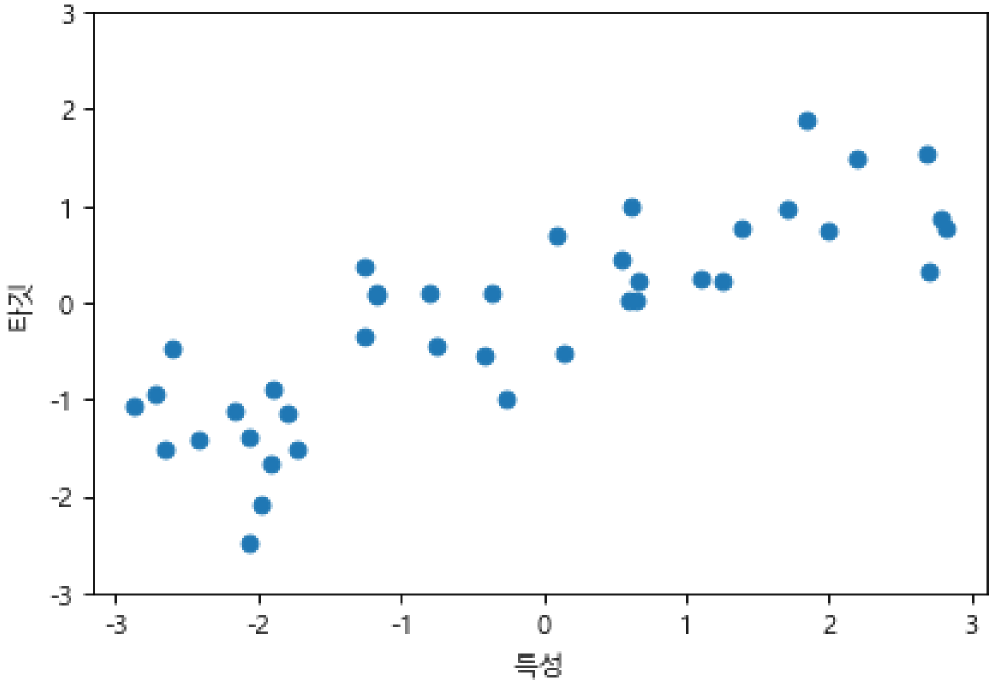
```
[[-0.75275929]
 [ 2.70428584]
 [ 1.39196365]
```

```
 [ 0.59195091]
 [-2.06388816]
 [-2.06403288]
 [-2.65149833]
 [ 2.19705687]
 [ 0.60669007]
 [ 1.24843547]
 [-2.87649303]
 [ 2.81945911]
 [ 1.99465584]
 [-1.72596534]
 [-1.9090502 ]
 [-1.89957294]
 [-1.17454654]
 [ 0.14853859]
 [-0.40832989]
 [-1.25262516]
 [ 0.67111737]
 [-2.16303684]
 [-1.24713211]
 [-0.80182894]
 [-0.26358009]
 [ 1.71105577]
 [-1.80195731]
 [ 0.08540663]
 [ 0.55448741]
 [-2.72129752]
 [ 0.64526911]
 [-1.97685526]
 [-2.60969044]
 [ 2.69331322]
 [ 2.7937922 ]
 [ 1.85038409]
 [-1.17231738]
 [-2.41396732]
 [ 1.10539816]
 [-0.35908504]]
[-0.44822073  0.33122576  0.77932073  0.03497884 -1.38773632 -2.47196233
 -1.52730805  1.49417157  1.00032374  0.22956153 -1.05979555  0.7789638
  0.75418806 -1.51369739 -1.67303415 -0.90496988  0.08448544 -0.52734666
 -0.54114599 -0.3409073   0.21778193 -1.12469096  0.37299129  0.09756349
 -0.98618122  0.96695428 -1.13455014  0.69798591  0.43655826 -0.95652133
  0.03527881 -2.08581717 -0.47411033  1.53708251  0.86893293  1.87664889
  0.0945257  -1.41502356  0.25438895  0.09398858]
```

In [11]:
```
plt.plot(X, y, 'o')
plt.ylim(-3, 3)
```

```
plt.xlabel("")
plt.ylabel("")
```

Out[11]: Text(0, 0.5, '')



In [12]:
```
#     - dictionary

from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
print("cancer.keys(): {}".format(cancer.keys()))
```

```
cancer.keys(): dict_keys(['data', 'target', 'target_names', 'DESCR', 'feature_names', 'filename
```

In [13]:
```
cancer.data
```

Out[13]:
```
array([[1.799e+01, 1.038e+01, 1.228e+02, ..., 2.654e-01, 4.601e-01,
         1.189e-01],
        [2.057e+01, 1.777e+01, 1.329e+02, ..., 1.860e-01, 2.750e-01,
         8.902e-02],
        [1.969e+01, 2.125e+01, 1.300e+02, ..., 2.430e-01, 3.613e-01,
         8.758e-02],
        ...,
        [1.660e+01, 2.808e+01, 1.083e+02, ..., 1.418e-01, 2.218e-01,
         7.820e-02],
```

```
               [2.060e+01, 2.933e+01, 1.401e+02, ..., 2.650e-01, 4.087e-01,
                1.240e-01],
               [7.760e+00, 2.454e+01, 4.792e+01, ..., 0.000e+00, 2.871e-01,
                7.039e-02]])
```

In [14]:
```
cancer.data.shape
```

Out[14]:
```
(569, 30)
```

In [15]:
```
cancer.target
```

Out[15]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,
       1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
       1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
       0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
       1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,
       0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0,
       0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
       0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1,
       1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1,
       1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 0, 0, 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, 0, 0, 0, 0, 0, 0, 1])
```

In [16]:
```
cancer.target.shape
```

Out[16]:
```
(569,)
```

In [17]:
```
cancer.target_names
```

Out[17]:
```
array(['malignant', 'benign'], dtype='<U9')
```

In [18]:
```
cancer.feature_names
```

Out[18]:
```
array(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimension',
       'radius error', 'texture error', 'perimeter error', 'area error',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error',
       'fractal dimension error', 'worst radius', 'worst texture',
       'worst perimeter', 'worst area', 'worst smoothness',
       'worst compactness', 'worst concavity', 'worst concave points',
       'worst symmetry', 'worst fractal dimension'], dtype='<U23')
```

In [19]:
```
print("  : {}".format(cancer.data.shape))
```

```
  : (569, 30)
```

In [20]:
```
print("  :\n{}".format(
      {n: v for n, v in zip(cancer.target_names, np.bincount(cancer.target))}))
```

```
  :
{'malignant': 212, 'benign': 357}
```

In [21]:
```
print(" :\n{}".format(cancer.feature_names))
```

```
 :
['mean radius' 'mean texture' 'mean perimeter' 'mean area'
 'mean smoothness' 'mean compactness' 'mean concavity'
 'mean concave points' 'mean symmetry' 'mean fractal dimension'
 'radius error' 'texture error' 'perimeter error' 'area error'
 'smoothness error' 'compactness error' 'concavity error'
 'concave points error' 'symmetry error' 'fractal dimension error'
 'worst radius' 'worst texture' 'worst perimeter' 'worst area'
 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry' 'worst fractal dimension']
```

In [22]:
```
#

from sklearn.datasets import load_boston

boston = load_boston()
print("boston.keys(): {}".format(boston.keys()))
```

```
boston.keys(): dict_keys(['data', 'target', 'feature_names', 'DESCR', 'filename'])
```

In [23]:
```
print("boston  : {}".format(boston.data.shape))
```

```
boston  : (506, 13)
```

```
In [24]: #      -      (feature engineering)

         X, y = mglearn.datasets.load_extended_boston()
         print("X.shape: {}".format(X.shape))
```

```
X.shape: (506, 104)
```

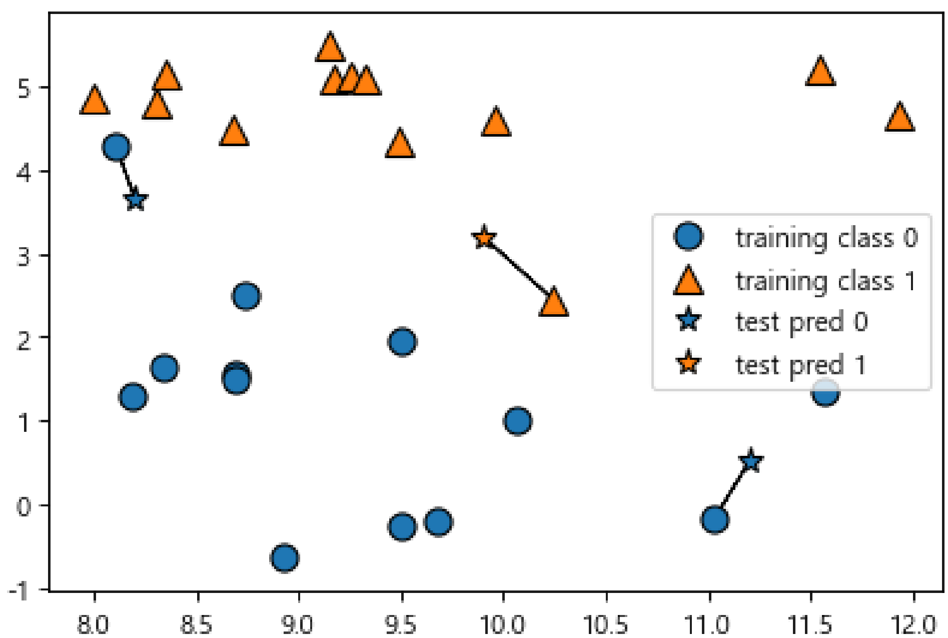
0.1.3 k-

-
- ,

k-

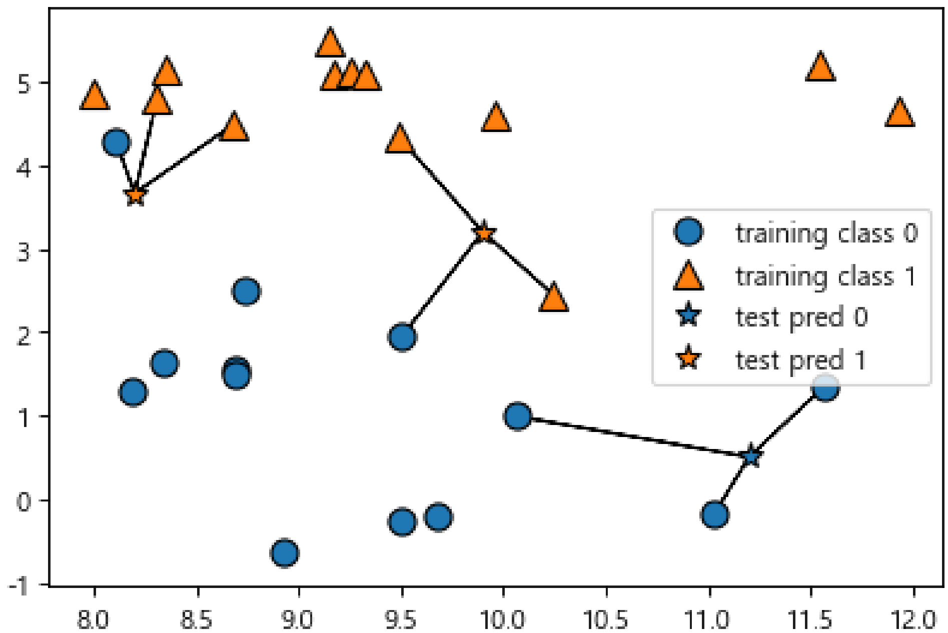
```
In [25]: mglearn.plots.plot_knn_classification(n_neighbors=1)
```

```
C:\Users\User\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:77: DeprecationWarning:
  warnings.warn(msg, category=DeprecationWarning)
```




```
In [26]: mglearn.plots.plot_knn_classification(n_neighbors=3)
```

```
C:\Users\User\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:77: DeprecationWarning:
  warnings.warn(msg, category=DeprecationWarning)
```




```
In [27]: from sklearn.model_selection import train_test_split
         X, y = mglearn.datasets.make_forge()
         print(X)
         print(y)

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

```
[[ 9.96346605  4.59676542]
 [11.0329545  -0.16816717]
 [11.54155807  5.21116083]
 [ 8.69289001  1.54322016]
 [ 8.1062269   4.28695977]
 [ 8.30988863  4.80623966]
 [11.93027136  4.64866327]
 [ 9.67284681 -0.20283165]
 [ 8.34810316  5.13415623]
 [ 8.67494727  4.47573059]
 [ 9.17748385  5.09283177]
 [10.24028948  2.45544401]
 [ 8.68937095  1.48709629]
 [ 8.92229526 -0.63993225]
 [ 9.49123469  4.33224792]
 [ 9.25694192  5.13284858]
 [ 7.99815287  4.8525051 ]
 [ 8.18378052  1.29564214]
 [ 8.7337095   2.49162431]
```

```
 [ 9.32298256  5.09840649]
 [10.06393839  0.99078055]
 [ 9.50048972 -0.26430318]
 [ 8.34468785  1.63824349]
 [ 9.50169345  1.93824624]
 [ 9.15072323  5.49832246]
 [11.563957    1.3389402 ]]
[1 0 1 0 0 1 1 0 1 1 1 1 0 0 1 1 1 0 0 1 0 0 0 0 1 0]
```

```
C:\Users\User\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:77: DeprecationWarning:
  warnings.warn(msg, category=DeprecationWarning)
```

In [28]:
```
# KNeighborsClassifier, k=3
```

In [29]:
```
# train
```

Out[29]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=3, p=2,
                     weights='uniform')
```

In [30]:
```
# test data predict
print("   : {}".format(   ))
```

```
  : [1 0 1 0 1 0 0]
```

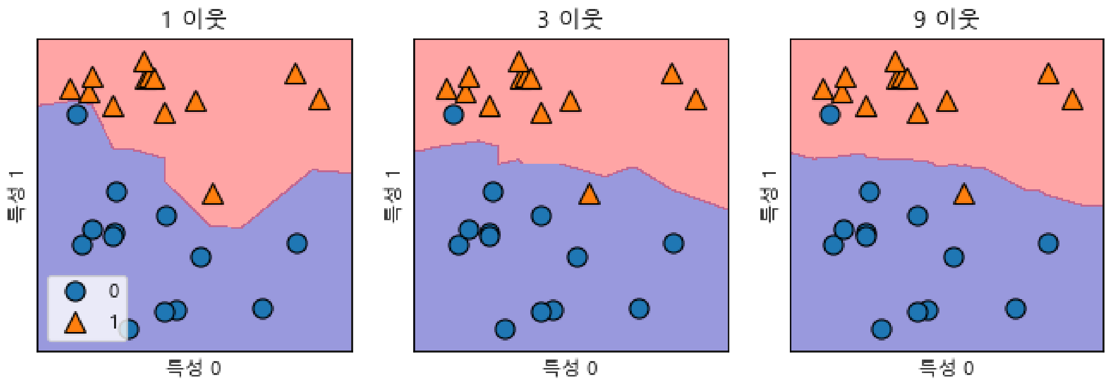
In [32]:
```
# test_data score
print("   : {:.2f}".format(   ))
```

```
  : 0.86
```

KNeighborsClassifier

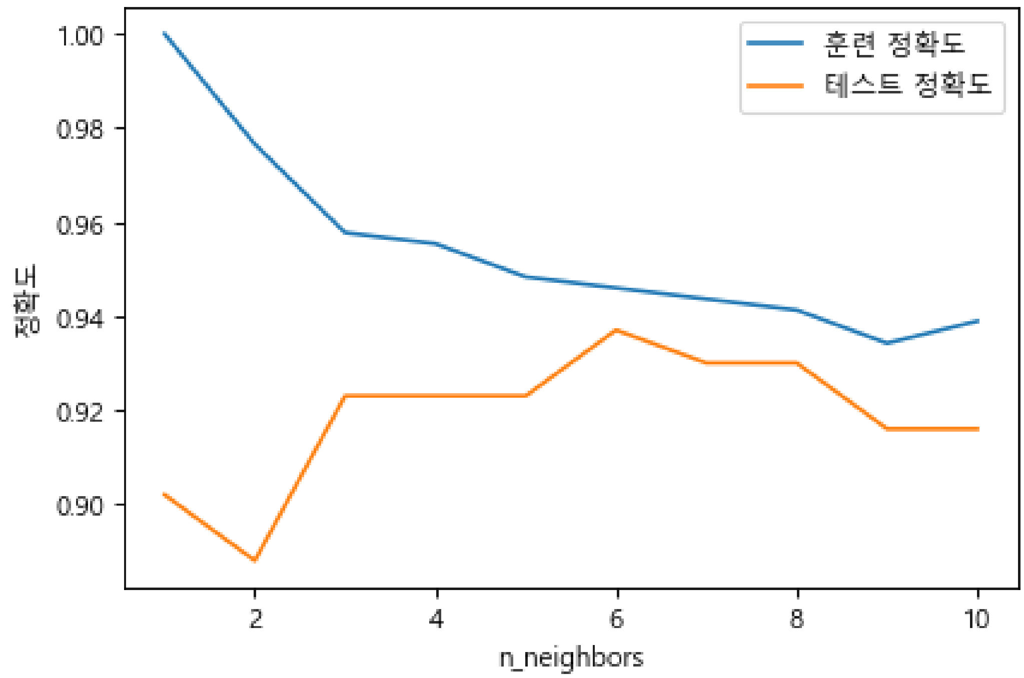
In [33]:
```
# binary classification

fig, axes = plt.subplots(1, 3, figsize=(10, 3))

for n_neighbors, ax in zip([1, 3, 9], axes):
    # fit  self
    #    fit
    clf = KNeighborsClassifier(n_neighbors=n_neighbors).fit(X, y)
    mglearn.plots.plot_2d_separator(clf, X, fill=True, eps=0.5, ax=ax, alpha=.4)
    mglearn.discrete_scatter(X[:, 0], X[:, 1], y, ax=ax)
    ax.set_title("{} ".format(n_neighbors))
    ax.set_xlabel(" 0")
    ax.set_ylabel(" 1")
axes[0].legend(loc=3)
```

```
Out[33]: <matplotlib.legend.Legend at 0x187300bf358>
```



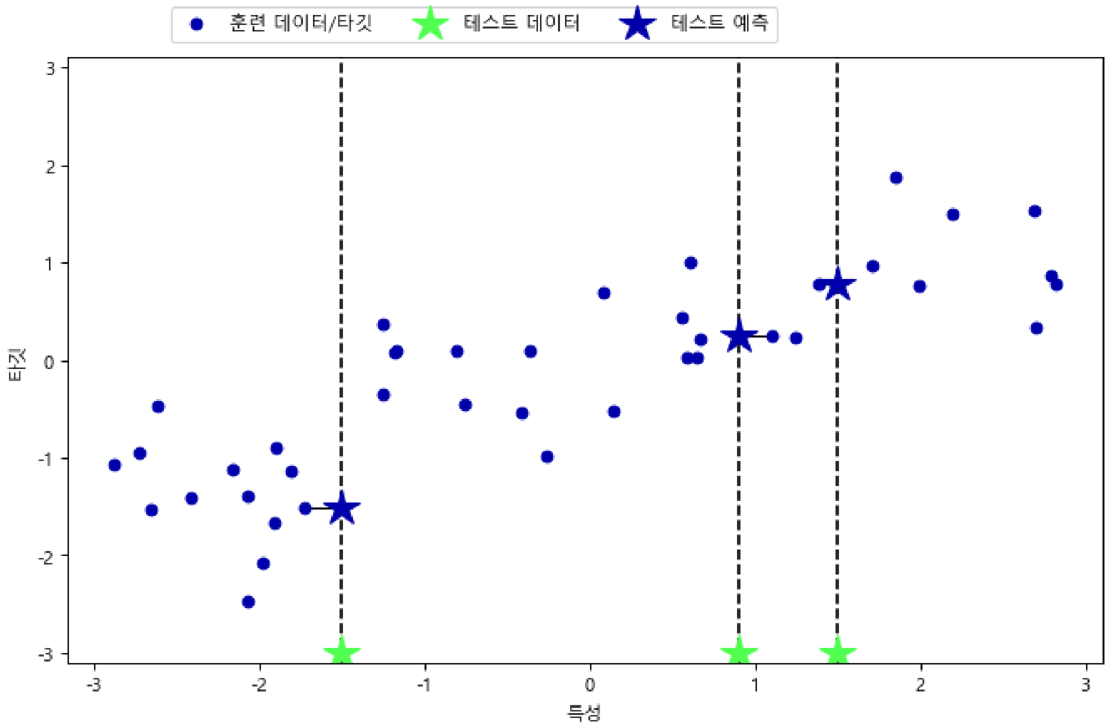

```
In [34]: from sklearn.datasets import load_breast_cancer

         cancer = load_breast_cancer()
         X_train, X_test, y_train, y_test = train_test_split(
             cancer.data, cancer.target, stratify=cancer.target, random_state=66)

         training_accuracy = []
         test_accuracy = []
         # 1  10  n_neighbors
         neighbors_settings = range(1, 11)

         for n_neighbors in neighbors_settings:
             #
             clf = KNeighborsClassifier(n_neighbors=n_neighbors)
             clf.fit(X_train, y_train)
             #
             training_accuracy.append(clf.score(X_train, y_train))
             #
             test_accuracy.append(clf.score(X_test, y_test))

         plt.plot(neighbors_settings, training_accuracy, label="  ")
         plt.plot(neighbors_settings, test_accuracy, label="  ")
         plt.ylabel("")
         plt.xlabel("n_neighbors")
         plt.legend()
```

```
Out[34]: <matplotlib.legend.Legend at 0x18730055c50>
```



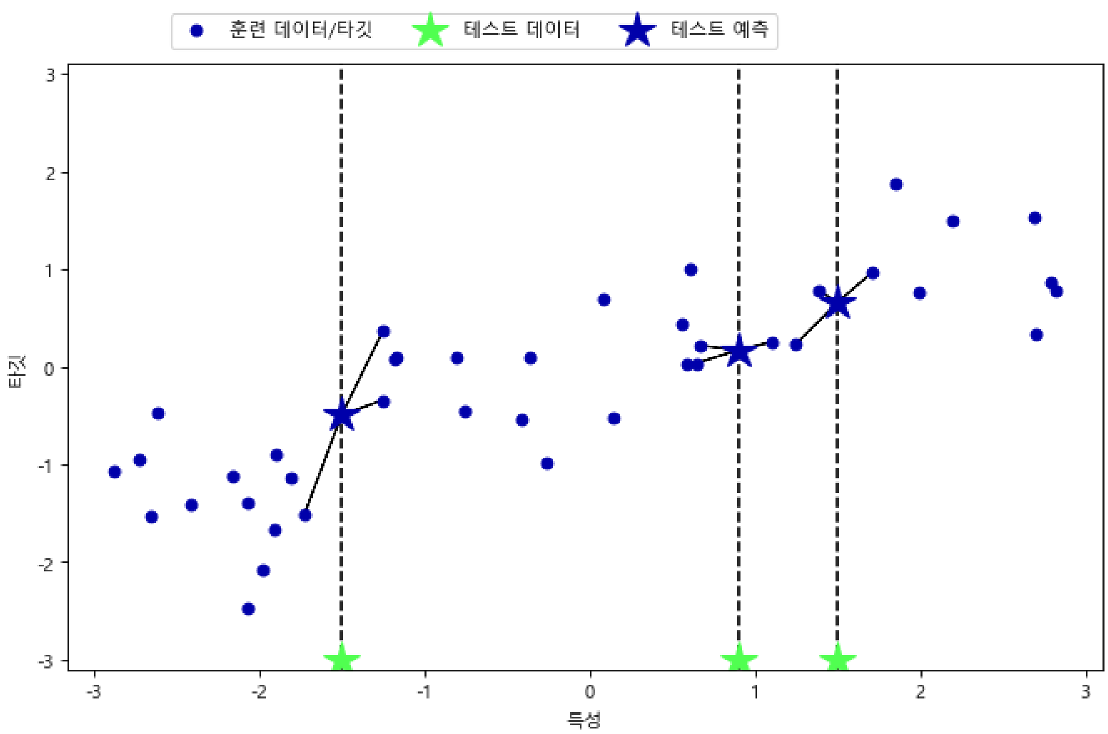
k-Neighbors Regression

```
In [43]: # knn regression (n=1) -

         mglearn.plots.plot_knn_regression(n_neighbors=1)
```



In [44]: # knn regression (n=n) -

```
mglearn.plots.plot_knn_regression(n_neighbors=3)
```



In [35]:
```
from sklearn.neighbors import KNeighborsRegressor

X, y = mglearn.datasets.make_wave(n_samples=40)

# wave
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

#   3
reg = KNeighborsRegressor(n_neighbors=3)
#
reg.fit(X_train, y_train)
```

Out[35]:
```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=3, p=2,
                    weights='uniform')
```

In [36]:
```
print("  :\n{}".format(reg.predict(X_test)))
```

```
  :
[-0.05396539  0.35686046  1.13671923 -1.89415682 -1.13881398 -1.63113382
  0.35686046  0.91241374 -0.44680446 -1.13881398]
```

In [37]:
```
print('  :', y_test)
```

```
  : [ 0.37299129  0.21778193  0.96695428 -1.38773632 -1.05979555 -0.90496988
  0.43655826  0.7789638  -0.54114599 -0.95652133]
```

In [38]:
```
print("  R^2: {:.2f}".format(reg.score(X_test, y_test)))
```

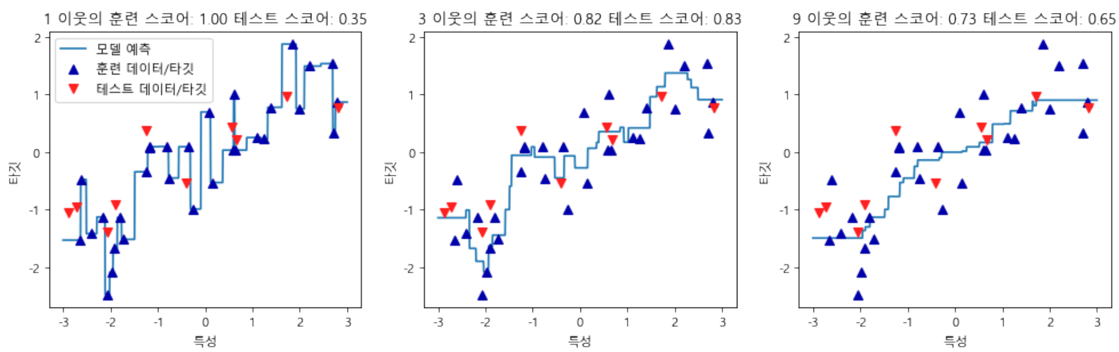
```
  R^2: 0.83
```

KNeighborsRegressor

In [40]:
```
fig, axes = plt.subplots(1, 3, figsize=(15, 4))
# -3  3  1,000
line = np.linspace(-3, 3, 1000).reshape(-1, 1)
for n_neighbors, ax in zip([1, 3, 9], axes):
    # 1, 3, 9
    reg = KNeighborsRegressor(n_neighbors=n_neighbors)
    reg.fit(X_train, y_train)
    ax.plot(line, reg.predict(line))
    ax.plot(X_train, y_train, '^', c=mglearn.cm2(0), markersize=8)
    ax.plot(X_test, y_test, 'v', c=mglearn.cm2(1), markersize=8)
```

```
        ax.set_title(
            "{}    : {:.2f}   : {:.2f}".format(
                n_neighbors, reg.score(X_train, y_train), reg.score(X_test, y_test)))
        ax.set_xlabel("")
        ax.set_ylabel("")
    axes[0].legend([" ", " /", " /"], loc="best")
```

Out[40]: <matplotlib.legend.Legend at 0x18730264278>



In [49]:
```
import sklearn

from sklearn import datasets
from sklearn.model_selection import train_test_split

from IPython.display import display
import numpy as np
import pandas as pd
# import mglearn

import matplotlib.pyplot as plt
%matplotlib inline
# from preamble import *

import mglearn
```

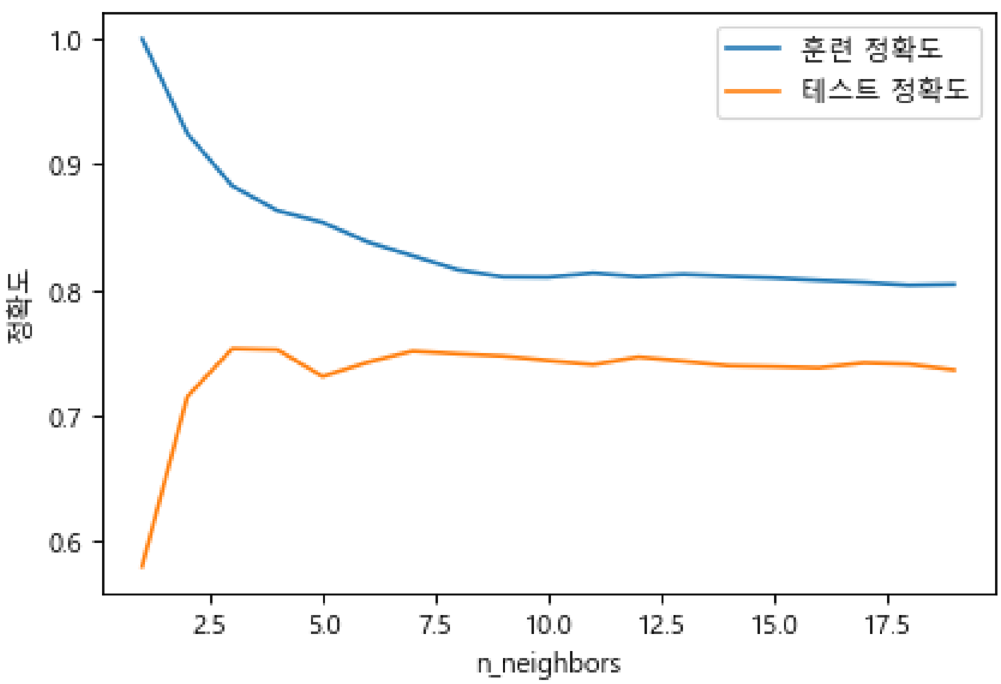
In [52]:
```
# Knn

from sklearn.datasets import load_breast_cancer
from sklearn.neighbors import KNeighborsRegressor

cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, stratify=cancer.target, random_state=66)
```

```python
training_accuracy = []
test_accuracy = []
# 1 ~ 10 , n_neighbors -> train score, test score
```

```python
print(training_accuracy)
print('Training_Accuracy_Best')
print(test_accuracy)
plt.plot(neighbors_settings, training_accuracy, label=" ")
plt.plot(neighbors_settings, test_accuracy, label=" ")
plt.ylabel("")
plt.xlabel("n_neighbors")
plt.legend()
```

```
[1.0, 0.9247403010387959, 0.8829293571714603, 0.8632782135538125, 0.8538958377499823, 0.8383331
Training_Accuracy_Best
[0.5802935010482181, 0.7151991614255766, 0.7535057069648264, 0.7526729559748428, 0.7313878406
```

```
Out[52]: <matplotlib.legend.Legend at 0x18732b9f550>
```



```
In [51]: # cross_validation

         from sklearn.model_selection import cross_val_score
```

```
In [54]: knn_reg = KNeighborsRegressor(n_neighbors=6)

         # cross_val_scores(, , , cv=?)
         scores = cross_val_score(knn_reg, cancer.data, cancer.target, cv=5)
         print(' : {}'.format(scores))
         print(' : {}'.format(scores.mean()))
         print(' : {}'.format(scores.std())) #   - cv
```

```
 : [0.51001705 0.73652538 0.87697072 0.75591616 0.66141075]
 : 0.7081680115173086
 : 0.12085398276285679
```

```
In [55]: knn_reg = KNeighborsRegressor(n_neighbors=6)

         # cross_val_scores(, , , cv=?)
         scores = cross_val_score(knn_reg, cancer.data, cancer.target, scoring='neg_mean_square
         print(' : {}'.format(scores))
         print(' : {}'.format(scores.mean()))
         print(' : {}'.format(scores.std())) #   - cv
```

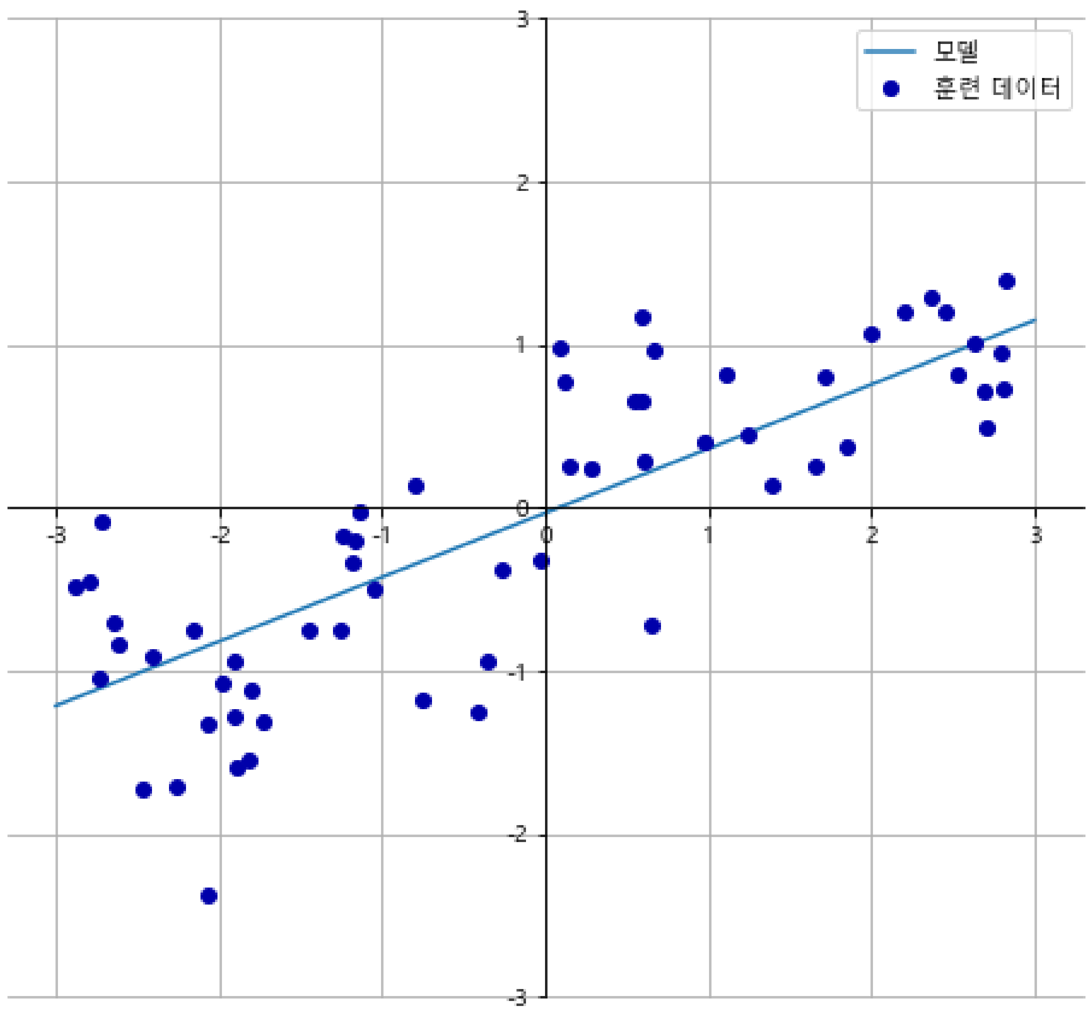
```
 : [-0.11793372 -0.06457115 -0.02802144 -0.0462963  -0.05998033]
 : -0.0633605892804775
 : 0.030103700493914677
```

```
In [ ]:
```

0.1.4

```
In [56]: mglearn.plots.plot_linear_regression_wave()
```

```
w[0]: 0.393906  b: -0.031804
```



()

```
In [53]: from sklearn.linear_model import LinearRegression
         X, y = mglearn.datasets.make_wave(n_samples=60)
         X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

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

```
In [54]: print("lr.coef_: {}".format(lr.coef_))
         print("lr.intercept_: {}".format(lr.intercept_))
```

```
lr.coef_: [0.39390555]
lr.intercept_: -0.031804343026759746
```

```
In [55]: print("  : {:.2f}".format(lr.score(X_train, y_train)))
         print("  : {:.2f}".format(lr.score(X_test, y_test)))
```

: 0.67
: 0.66

```
In [57]: from sklearn.model_selection import train_test_split
         from sklearn.linear_model import LinearRegression

         X, y = mglearn.datasets.load_extended_boston()

         X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
         lr = LinearRegression().fit(X_train, y_train)
```

```
In [58]: print("  : {:.2f}".format(lr.score(X_train, y_train)))
         print("  : {:.2f}".format(lr.score(X_test, y_test)))
```

: 0.95
: 0.61

- (w) , w 0
- ( )
- -
- L2
- a*((w)) . a

```
In [62]: #  -  ,
         from sklearn.linear_model import Ridge
         ridge01 = Ridge(alpha=0).fit(X_train, y_train)
         print("  : {:.2f}".format(ridge01.score(X_train, y_train)))
         print("  : {:.2f}".format(ridge01.score(X_test, y_test)))
```

: 0.95
: 0.61

```
C:\Users\User\Anaconda3\lib\site-packages\sklearn\linear_model\ridge.py:125: LinAlgWarning: Il
  overwrite_a=True).T
```

```
In [63]: from sklearn.linear_model import Ridge

         ridge = Ridge().fit(X_train, y_train)
         print("  : {:.2f}".format(ridge.score(X_train, y_train)))
         print("  : {:.2f}".format(ridge.score(X_test, y_test)))
```

: 0.89
: 0.75

```
In [66]: # ridge01  /alpha = 0.01
```

```
: 0.94
: 0.70
```

```
In [67]: # ridge01 alpha =0.1
```

```
: 0.93
: 0.77
```

```
In [76]: # ridge01 / alpha=1
```

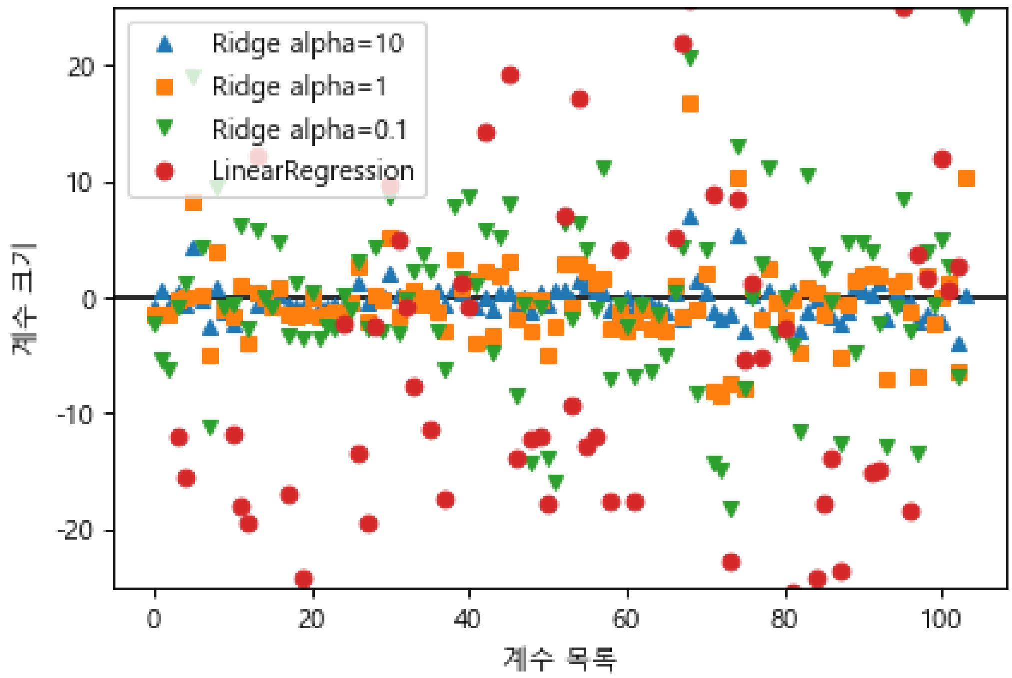
```
: 0.89
: 0.75
```

```
In [69]: # ridge10 / alpha=10
```

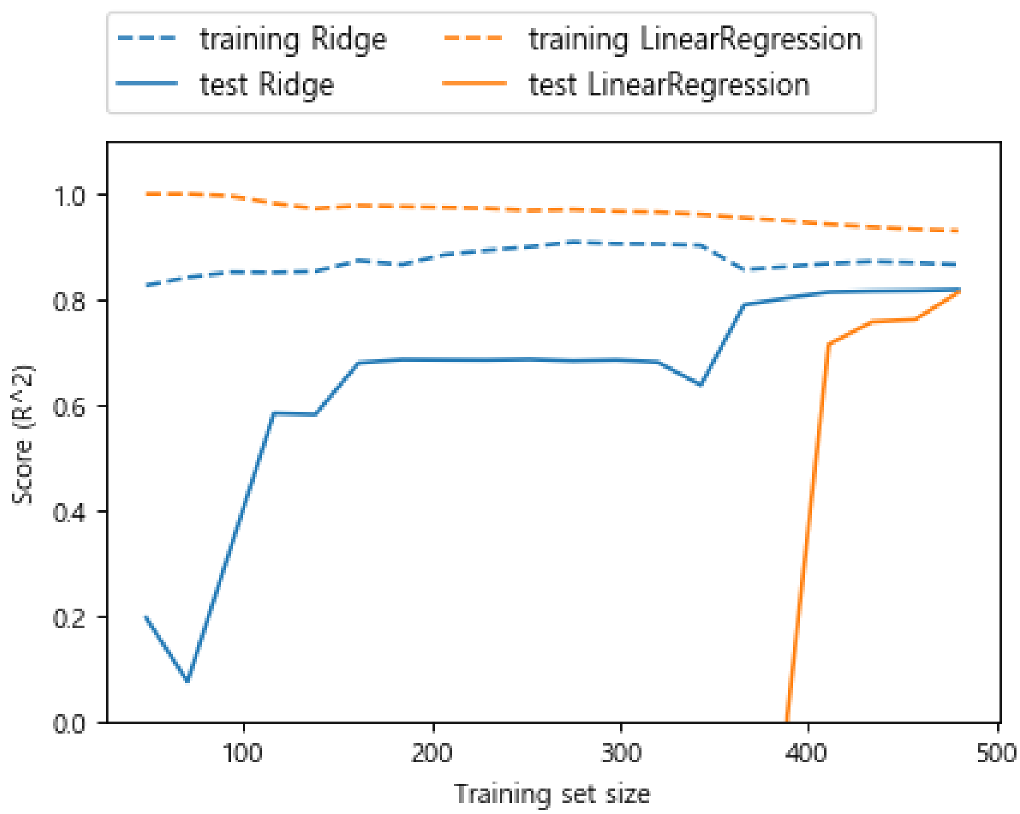
```
: 0.79
: 0.64
```

```
In [70]: plt.plot(ridge10.coef_, '^', label="Ridge alpha=10")
         plt.plot(ridge.coef_, 's', label="Ridge alpha=1")
         plt.plot(ridge01.coef_, 'v', label="Ridge alpha=0.1")

         plt.plot(lr.coef_, 'o', label="LinearRegression")
         plt.xlabel(" ")
         plt.ylabel(" ")
         xlims = plt.xlim()
         plt.hlines(0, xlims[0], xlims[1])
         plt.xlim(xlims)
         plt.ylim(-25, 25)
         plt.legend()
```

```
Out[70]: <matplotlib.legend.Legend at 0x18733ab7a20>
```



In [71]: mglearn.plots.plot_ridge_n_samples()



Lasso

- (w) , w 0
- ( )
- 0 . .
- -
- L2
- a*((w)) . a

In [73]:
```
from sklearn.linear_model import Lasso # , 105  4

lasso = Lasso().fit(X_train, y_train)
print("  : {:.2f}".format(lasso.score(X_train, y_train)))
print("  : {:.2f}".format(lasso.score(X_test, y_test)))
print("  : {}".format(np.sum(lasso.coef_ != 0)))
```

```
 : 0.29
 : 0.21
 : 4
```

In [74]:
```
lasso001 = Lasso(alpha=0.01, max_iter=100000).fit(X_train, y_train)


print("  : {}".format(np.sum(lasso001.coef_ != 0)))
```

```
 : 0.90
 : 0.77
 : 33
```

In [75]:
```
#, alpha      .
lasso00001 = Lasso(alpha=0.0001, max_iter=100000).fit(X_train, y_train)


print("  : {}".format(np.sum(lasso00001.coef_ != 0)))
```

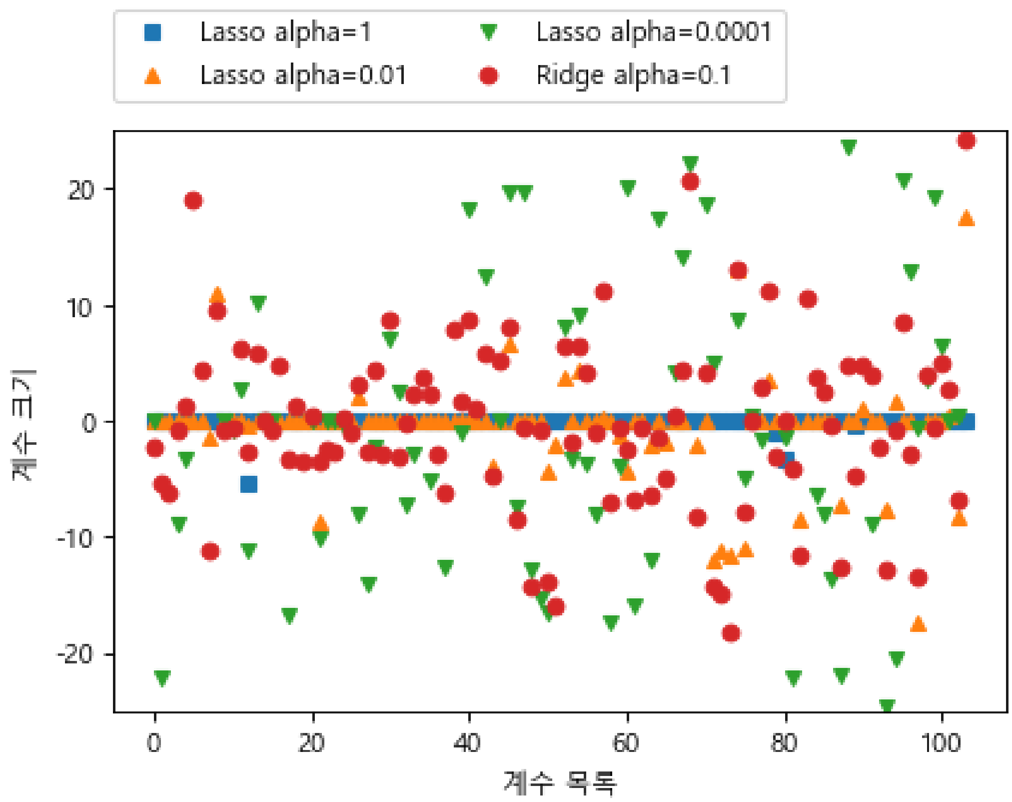
```
 : 0.95
 : 0.64
 : 96
```

In [17]:
```
#
lasso00001 = Lasso(alpha=1, max_iter=100000).fit(X_train, y_train)


print("  : {}".format(np.sum(lasso00001.coef_ != 0)))
```

```
 : 0.29
 : 0.21
 : 4
```

In [76]:
```
plt.plot(lasso.coef_, 's', label="Lasso alpha=1")
plt.plot(lasso001.coef_, '^', label="Lasso alpha=0.01")
plt.plot(lasso00001.coef_, 'v', label="Lasso alpha=0.0001")

plt.plot(ridge01.coef_, 'o', label="Ridge alpha=0.1")
plt.legend(ncol=2, loc=(0, 1.05))
plt.ylim(-25, 25)
plt.xlabel(" ")
plt.ylabel(" ")
```

Out[76]:
```
Text(0, 0.5, ' ')
```

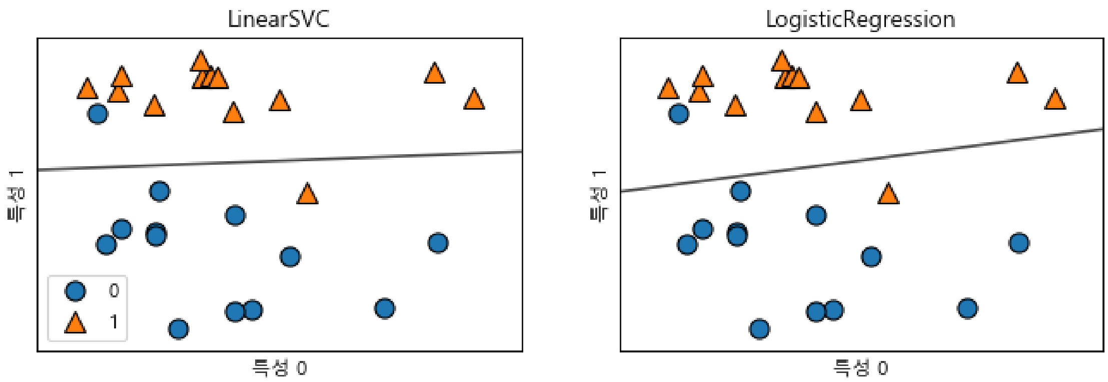


0.2

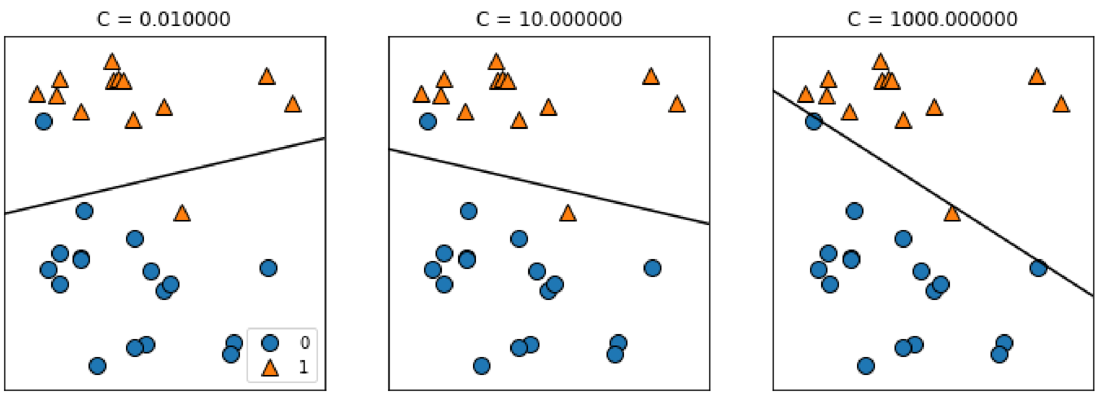
```
In [78]: import mglearn
         import matplotlib.pyplot as plt
         from sklearn.linear_model import LogisticRegression
         from sklearn.svm import LinearSVC

         X, y = mglearn.datasets.make_forge()

         fig, axes = plt.subplots(1, 2, figsize=(10, 3))

         for model, ax in zip([LinearSVC(), LogisticRegression()], axes):
             clf = model.fit(X, y)
             mglearn.plots.plot_2d_separator(clf, X, fill=False, eps=0.5,
                                             ax=ax, alpha=.7)
             mglearn.discrete_scatter(X[:, 0], X[:, 1], y, ax=ax)
             ax.set_title("{}".format(clf.__class__.__name__))
             ax.set_xlabel(" 0")
             ax.set_ylabel(" 1")
         axes[0].legend()
```

```
C:\Users\User\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:77: DeprecationWarning:
  warnings.warn(msg, category=DeprecationWarning)
C:\Users\User\Anaconda3\lib\site-packages\sklearn\svm\base.py:931: ConvergenceWarning: Libline
  "the number of iterations.", ConvergenceWarning)
C:\Users\User\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:433: FutureWarning:
  FutureWarning)
```

```
Out[78]: <matplotlib.legend.Legend at 0x187322f2278>
```




```
In [7]: # C    ,     ,
        # C    , (w)   0 ,

        mglearn.plots.plot_linear_svc_regularization()
```




```
In [79]: 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, stratify=cancer.target, random_state=42)


         logreg = LogisticRegression().fit(X_train, y_train)
         print("  : {:.3f}".format(logreg.score(X_train, y_train)))
         print("  : {:.3f}".format(logreg.score(X_test, y_test)))
```

```
  : 0.955
  : 0.958


C:\Users\User\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:433: FutureWarning:
  FutureWarning)
```

```
In [80]: from sklearn.datasets import load_breast_cancer
         cancer = load_breast_cancer()
         X_train, X_test, y_train, y_test = train_test_split(
             cancer.data, cancer.target, stratify=cancer.target, random_state=42)
         logreg = LogisticRegression(C=1).fit(X_train, y_train)
         print("  : {:.3f}".format(logreg.score(X_train, y_train)))
         print("  : {:.3f}".format(logreg.score(X_test, y_test)))
```

```
  : 0.955
  : 0.958


C:\Users\User\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:433: FutureWarning:
  FutureWarning)
```

In [81]: # . - .

```
logreg100 = LogisticRegression(C=100).fit(X_train, y_train)
print("  : {:.3f}".format(logreg100.score(X_train, y_train)))
print("  : {:.3f}".format(logreg100.score(X_test, y_test)))
```

```
 : 0.972
 : 0.965
```

```
C:\Users\User\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:433: FutureWarning:
  FutureWarning)
```

In [82]: #   -

```
logreg001 = LogisticRegression(C=0.01).fit(X_train, y_train)
print("  : {:.3f}".format(logreg001.score(X_train, y_train)))
print("  : {:.3f}".format(logreg001.score(X_test, y_test)))
```

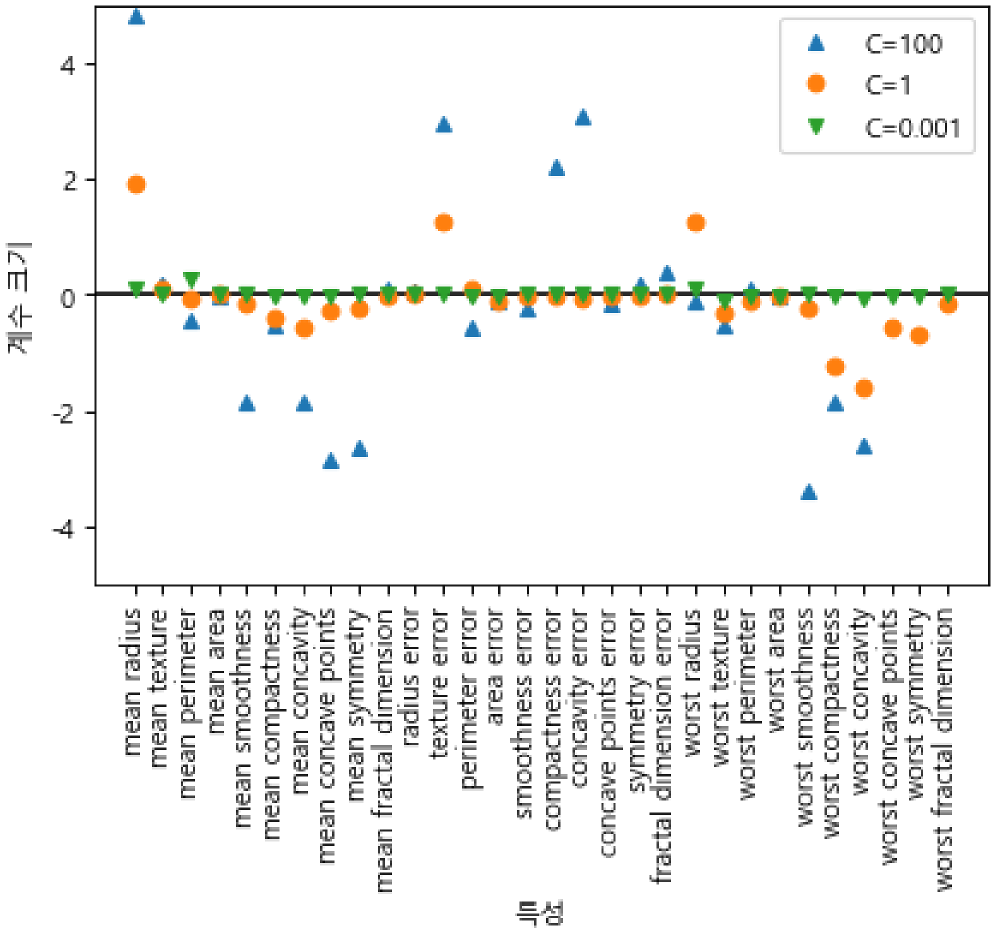
```
 : 0.934
 : 0.930
```

```
C:\Users\User\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:433: FutureWarning:
  FutureWarning)
```

In [83]:

```
plt.plot(logreg100.coef_.T, '^', label="C=100")
plt.plot(logreg.coef_.T, 'o', label="C=1")
plt.plot(logreg001.coef_.T, 'v', label="C=0.001")
plt.xticks(range(cancer.data.shape[1]), cancer.feature_names, rotation=90)
xlims = plt.xlim()
plt.hlines(0, xlims[0], xlims[1])
plt.xlim(xlims)
plt.ylim(-5, 5)
plt.xlabel("")
plt.ylabel(" ")
plt.legend()
```

Out[83]: <matplotlib.legend.Legend at 0x187323b2dd8>



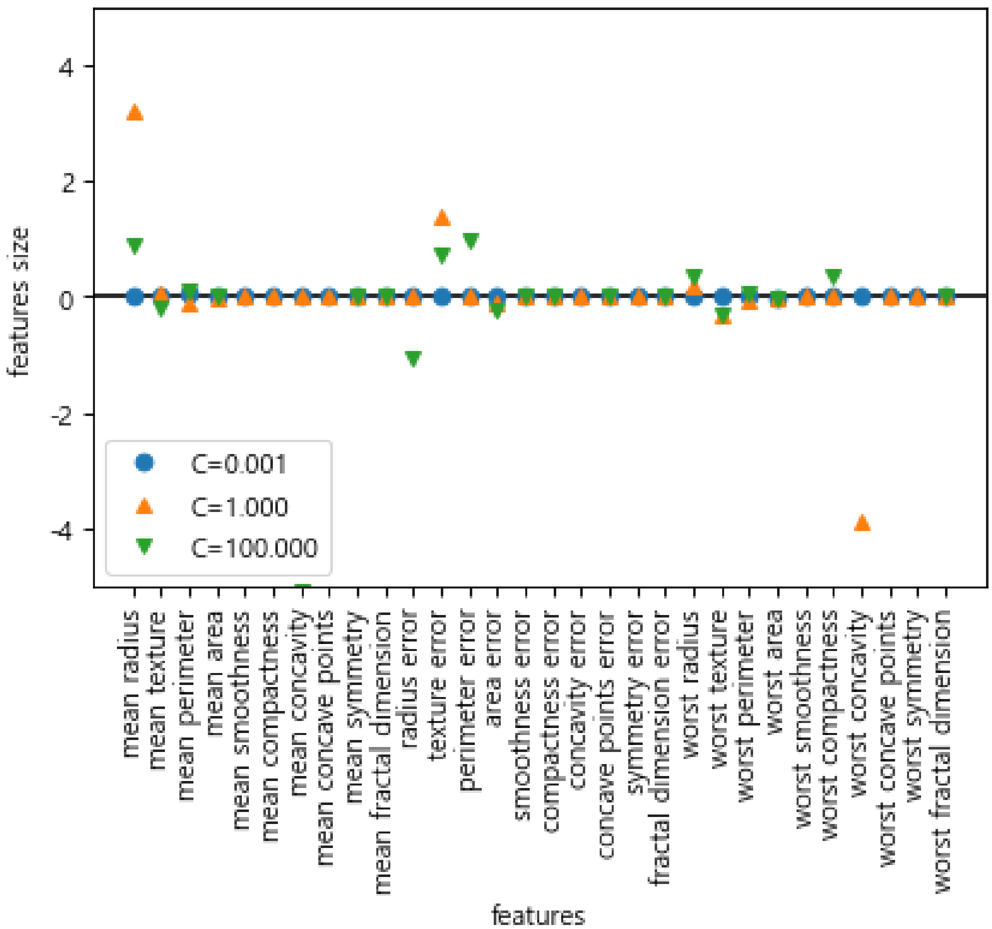

```
In [84]: for C, marker in zip([0.001, 1, 100], ['o', '^', 'v']):
             lr_l1 = LogisticRegression(C=C, penalty="l1").fit(X_train, y_train)
             print("C={:.3f}  l1    : {:.2f}".format(
                   C, lr_l1.score(X_train, y_train)))
             print("C={:.3f}  l1    : {:.2f}".format(
                   C, lr_l1.score(X_test, y_test)))
             plt.plot(lr_l1.coef_.T, marker, label="C={:.3f}".format(C))

         plt.xticks(range(cancer.data.shape[1]), cancer.feature_names, rotation=90)
         xlims = plt.xlim()
         plt.hlines(0, xlims[0], xlims[1])
         plt.xlim(xlims)
         plt.xlabel("features")
         plt.ylabel("features size")

         plt.ylim(-5, 5)
         plt.legend(loc=3)
```

```
        # penalty -
        # Penalty - l1 :   L1
        # Penalty - l2 :    L2
```

```
C:\Users\User\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:433: FutureWarning:
  FutureWarning)
C:\Users\User\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:433: FutureWarning:
  FutureWarning)
C:\Users\User\Anaconda3\lib\site-packages\sklearn\svm\base.py:931: ConvergenceWarning: Liblinea
  "the number of iterations.", ConvergenceWarning)
C:\Users\User\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:433: FutureWarning:
  FutureWarning)
```

```
C=0.001  l1    : 0.91
C=0.001  l1    : 0.92
C=1.000  l1    : 0.96
C=1.000  l1    : 0.96
C=100.000  l1    : 0.99
C=100.000  l1    : 0.98
```

```
Out[84]: <matplotlib.legend.Legend at 0x1873244a518>
```



0.2.1

- - 1. 2. 3.
- - (sklearn)
- (feature_importance_) - 1, -

```
In [87]: from sklearn.tree import DecisionTreeClassifier

         cancer = load_breast_cancer()
         X_train, X_test, y_train, y_test = train_test_split(
             cancer.data, cancer.target, stratify=cancer.target, random_state=42)


         # tree model
```

```
: 1.000
: 0.937
```

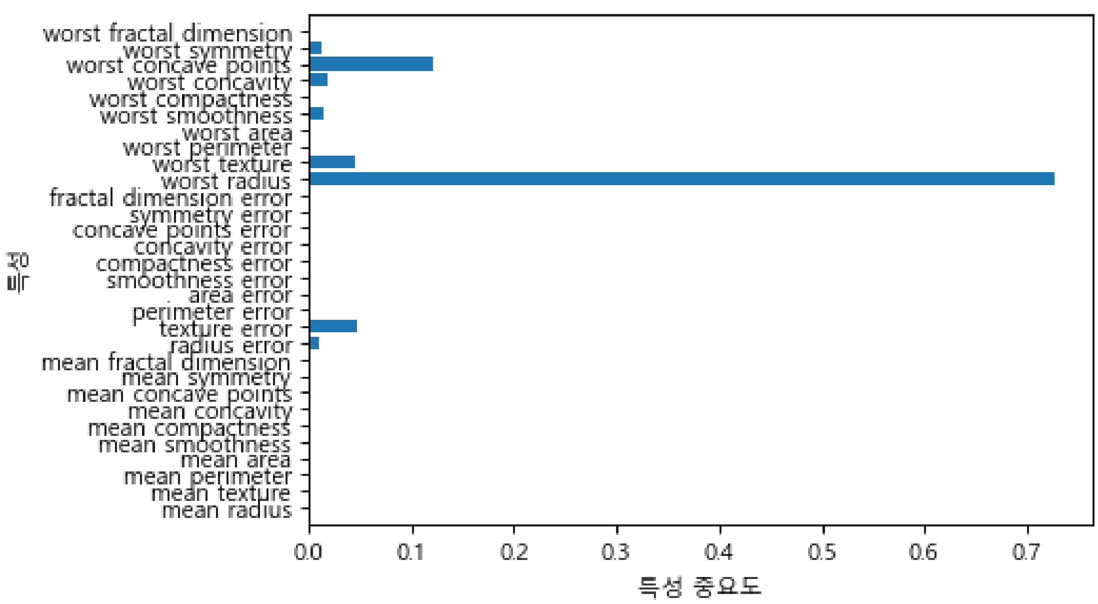
```
In [88]: # tree model maxdepth = 4
```

```
: 0.988
: 0.951
```

```
In [91]: print(" :\n{}".format(                ))
```

```
 :
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.01019737 0.04839825
 0.         0.         0.0024156  0.         0.         0.
 0.         0.         0.72682851 0.0458159  0.         0.
 0.0141577  0.         0.018188   0.1221132  0.01188548 0.        ]
```

```
In [105]: def plot_feature_importances_cancer(model):
              n_features = cancer.data.shape[1]
              plt.barh(range(n_features), model.feature_importances_, align='center')
              plt.yticks(np.arange(n_features), cancer.feature_names)
              plt.xlabel(" ")
              plt.ylabel("")
              plt.ylim(-1, n_features)

          plot_feature_importances_cancer(tree)
```



0.2.2

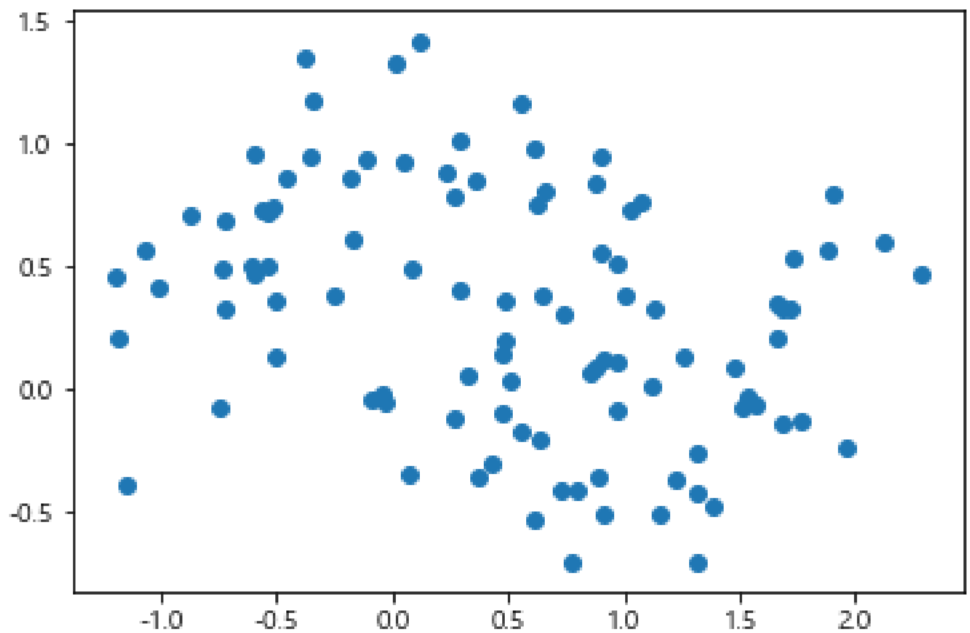
-
-
- ,
- - , ##### #####

```
In [96]: from sklearn.ensemble import RandomForestClassifier
         from sklearn.datasets import make_moons
         from sklearn.model_selection import train_test_split
         import matplotlib.pyplot as plt
         import pandas as pd
         import numpy as np
         %matplotlib inline

         X, y = make_moons(n_samples=100, noise=0.25, random_state=3)
         X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, random_state=42)


         forest = RandomForestClassifier(n_estimators=5, random_state=2)
         forest.fit(X_train, y_train)
```

```
Out[96]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                     max_depth=None, max_features='auto', max_leaf_nodes=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, n_estimators=5, n_jobs=None,
                     oob_score=False, random_state=2, verbose=0, warm_start=False)
```

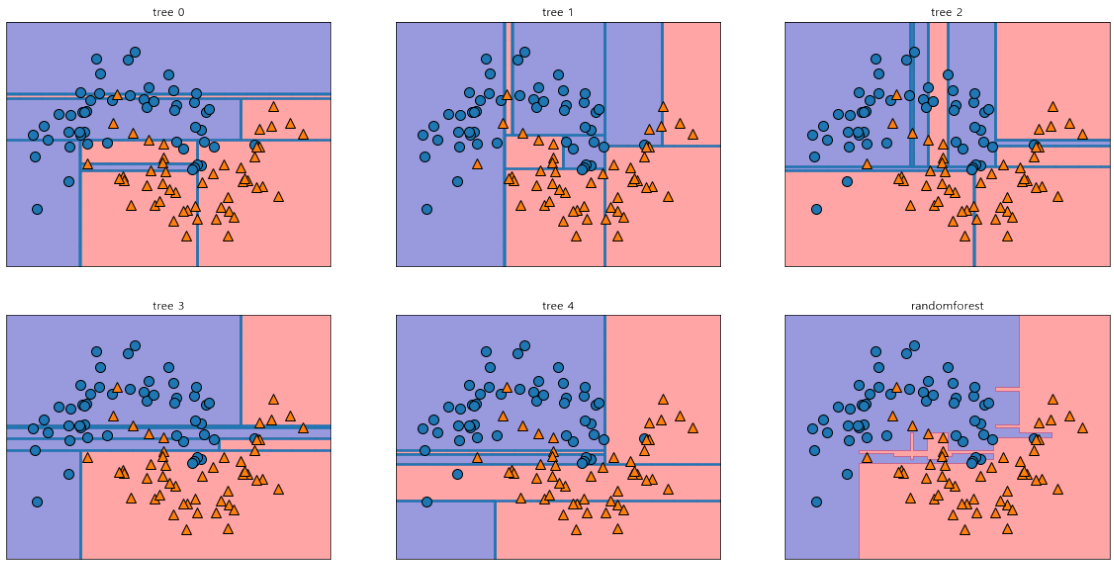
```
In [97]: plt.scatter(X[:,0], X[:,1])
```

```
Out[97]: <matplotlib.collections.PathCollection at 0x18736ab39e8>
```



In [98]:
```
import mglearn
fig, axes = plt.subplots(2, 3, figsize=(20, 10))
for i, (ax, tree) in enumerate(zip(axes.ravel(), forest.estimators_)):
    ax.set_title("tree {}".format(i))
    mglearn.plots.plot_tree_partition(X, y, tree, ax=ax)

mglearn.plots.plot_2d_separator(forest, X, fill=True, ax=axes[-1, -1], alpha=.4)
axes[-1, -1].set_title("randomforest")
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
```

Out[98]:
```
[<matplotlib.lines.Line2D at 0x18736bc0cf8>,
 <matplotlib.lines.Line2D at 0x18736c12d68>]
```

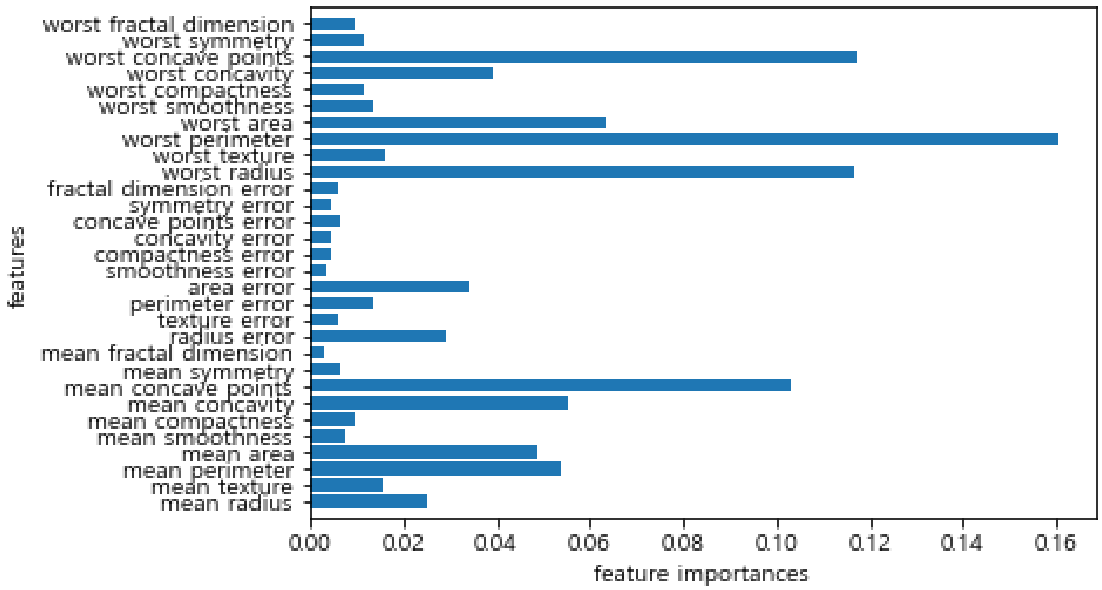



```
In [99]: from sklearn.datasets import load_breast_cancer
         cancer = load_breast_cancer()

         X_train, X_test, y_train, y_test = train_test_split(
             cancer.data, cancer.target, random_state=0)
         forest = RandomForestClassifier(n_estimators=100, random_state=0)
         forest.fit(X_train, y_train)

         print("   : {:.3f}".format(forest.score(X_train, y_train)))
         print("   : {:.3f}".format(forest.score(X_test, y_test)))
```

```
  : 1.000
  : 0.972
```

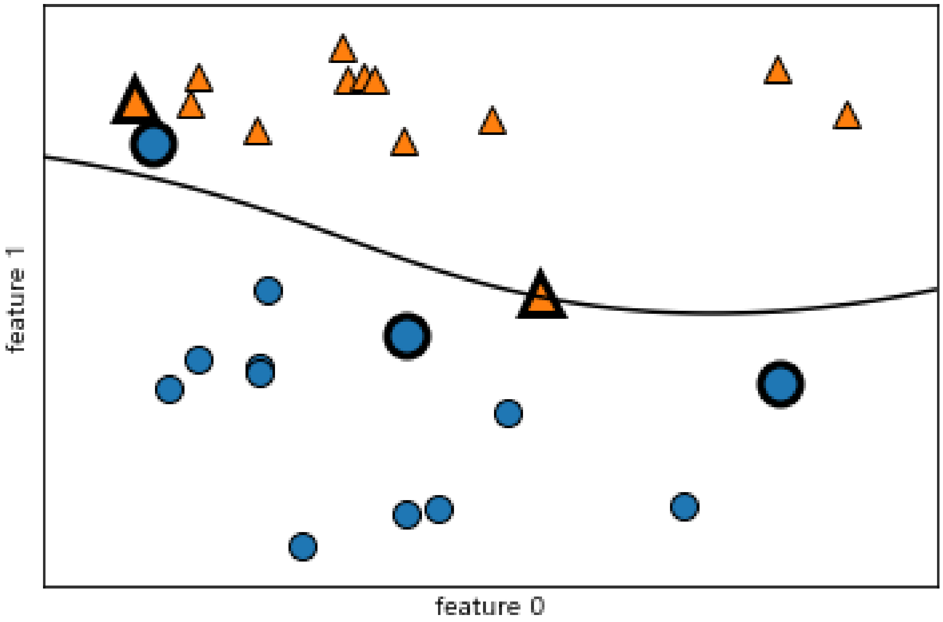
```
In [101]: def plot_feature_importances_cancer(model):
              n_features = cancer.data.shape[1]
              plt.barh(range(n_features), model.feature_importances_, align='center')
              plt.yticks(np.arange(n_features), cancer.feature_names)
              plt.xlabel("feature importances")
              plt.ylabel("features")
              plt.ylim(-1, n_features)
```

```
In [102]: plot_feature_importances_cancer(forest)
```



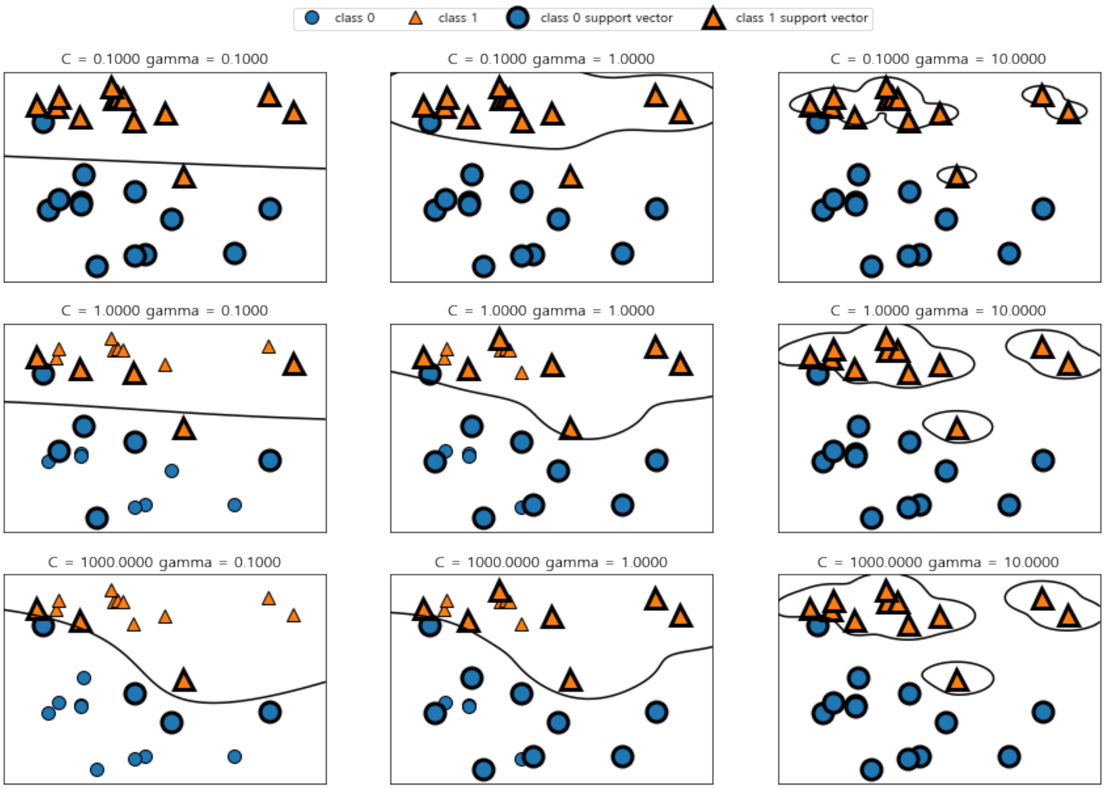
0.2.3 SVM

```
In [103]: from sklearn.svm import SVC

          X, y = mglearn.tools.make_handcrafted_dataset()

          svm = SVC(kernel='rbf', C=10, gamma=0.1).fit(X, y)
          mglearn.plots.plot_2d_separator(svm, X, eps=.5)
          #
          mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
          #
          sv = svm.support_vectors_
          # dual_coef_
          sv_labels = svm.dual_coef_.ravel() > 0
          mglearn.discrete_scatter(sv[:, 0], sv[:, 1], sv_labels, s=15, markeredgewidth=3)
          plt.xlabel("feature 0")
          plt.ylabel("feature 1")

Out[103]: Text(0, 0.5, 'feature 1')
```



SVM

```
In [106]: fig, axes = plt.subplots(3, 3, figsize=(15, 10))

          for ax, C in zip(axes, [-1, 0, 3]):
              for a, gamma in zip(ax, range(-1, 2)):
                  mglearn.plots.plot_svm(log_C=C, log_gamma=gamma, ax=a)

          axes[0, 0].legend(["class 0", "class 1", "class 0 support vector", "class 1 support v
                            ncol=4, loc=(.9, 1.2))

          # gamma
          #
          #     .
          # C
          #   ,
          #

Out[106]: <matplotlib.legend.Legend at 0x18737ee8f28>
```




```
In [136]: X_train, X_test, y_train, y_test = train_test_split(
              cancer.data, cancer.target, random_state=77)

          svc = SVC(C=0.1, gamma=100)
          svc.fit(X_train, y_train)

          print("   : {:.2f}".format(svc.score(X_train, y_train)))
          print("   : {:.2f}".format(svc.score(X_test, y_test)))
```

```
  : 0.63
  : 0.62
```

```
In [137]: score = 0

          # use for, select best score and best parameters set

          gamma  : [0.1, 1, 10, 100, 1000, 10000]:
              C  : [0.1, 0.5, 1, 2, 4, 8, 20, 100]:
```

```
        print(best_score)
        print(best_params)
```

```
0.6223776223776224
{'C': 100, 'cache_size': 200, 'class_weight': None, 'coef0': 0.0, 'decision_function_shape': '
```

In [138]: # train score and test score of best model

Out[138]: 0.6223776223776224

In [139]: # use gridsearchCV

```
C:\Users\User\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:2053: FutureWarnin
  warnings.warn(CV_WARNING, FutureWarning)
```

```
Out[139]: GridSearchCV(cv='warn', error_score='raise-deprecating',
                 estimator=SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
            decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
            kernel='rbf', max_iter=-1, probability=False, random_state=None,
            shrinking=True, tol=0.001, verbose=False),
                 fit_params=None, iid='warn', n_jobs=None,
                 param_grid={'C': [0.1, 0.5, 1, 2, 4, 8, 20, 100], 'gamma': [0.1, 1, 10, 100,
                 pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
                 scoring=None, verbose=0)
```

In [140]: # best score

Out[140]: 0.6291079812206573

In [141]: # best parameters set

Out[141]: {'C': 0.1, 'gamma': 0.1}

In [142]: # train score and test score

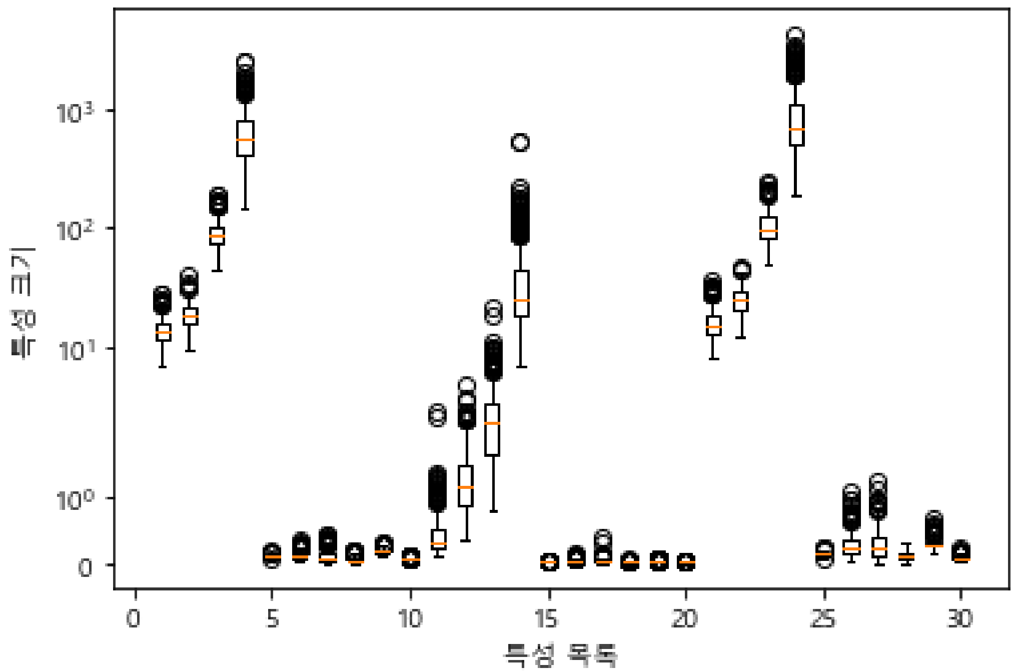
```
0.6291079812206573
0.6223776223776224
```

In [ ]:

In [ ]:

```
In [143]: plt.boxplot(X_train, manage_xticks=False)
          plt.yscale("symlog")
          plt.xlabel(" ")
          plt.ylabel(" ")
```

Out[143]: Text(0, 0.5, ' ')



SVM

```
In [144]: # data scaling
```

```
[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.]

 [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.]
```

```
In [157]: X_test_scaled = (X_test - min_on_training) / range_on_training
```

```
In [158]: svc = SVC(C=10000, gamma=10000)
          svc.fit(X_train_scaled, y_train)

          print("  : {:.3f}".format(svc.score(X_train_scaled, y_train)))
          print("  : {:.3f}".format(svc.score(X_test_scaled, y_test)))
```

```
  : 1.000
  : 0.622
```

```
In [159]: # model optimization through scaling data
```

```
  : 1.000
  : 0.944
```

```
In [ ]:
```