Deep Learning

지도학습

개요

데이터셋

K이웃 :분류, 회귀

선형모델:선형회귀(최소자승법),Ridge회귀,Lasso 회귀,…

Yoon Joong Kim

Department of Computer Engineering, Hanbat National University

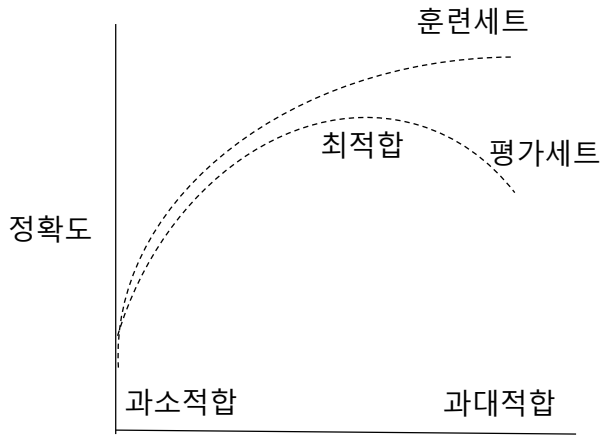
yjkim@hanbat.ac.kr

content

- 지도학습
  1. 지도학습 개요
     1. 분류,회귀,일반화,과대적합,과소적합,복잡도, 데이터셋크기
  2. 예제에서 사용할 데이터셋
     1. X, y = mglearn.datasets.make_forge() #X:(26, 2)y:(26,) int32 [0, 1]
     2. X, y = mglearn.datasets.make_wave(n_samples=40) #X:(40, 1)y:(40,) float64]
     3. cancer = sklearn.datasets.load_breast_cancer() #data:(569, 30) target:(569,) int32 [0, 1]
     4. boston = sklearn.datasets.load_boston() #data:(506, 13) target:(506,) float64
     5. boston_ext= mglearn.datasets.load_extended_boston() #data:(506, 104) target:(506,) float64
     6. X,y =sklearn.datasets.make_blobs(random_state=42) #X:(100, 2) y:[(100,) int32 [0, 1, 2]]
  3. K-최근이웃
     1. K-최근접 이웃 분류
        1. 개요, 2. KNeighbordClassifiers 분석
     2. K-최근접 이웃 회귀
        1. 개요, 2. KNeighborsRegressor 분석
  4. 선형모델
     1. 회귀선형모델
     2. 선형회귀(최소자승법)
     3. Ridge회귀
     4. Lasso 회귀
     5. 분류용 선형 모델
     6. 다중클래스 분류용 선형 모델
     7. 장단점과 매개변수

1. 지도학습개요

- 1 지도학습개요
  - 데이터 샘플과 기대되는 결과(X,y)의 셋에 대하여 알고리즘으로 학습(일반화) 시켜서 임의의 샘플에 대하여 결과를 예측하는 방법
  - 분류와 회귀
    - 분류
      - 미리 정의된 가능성 있는 클래스의 하나를 예측하는 것이다.
      - 3 붓꽃품종 중 하나를 예측
      - 이진분류(binary classification),다중분류(multiclass classification)
    - 회귀
      - 연속적인 수, 실수를 예축하는 것이다
      - 나이, 성적점수
  - 일반화,과대적합,과소적합
    - 일반화
      - 지도학습에서 처음보는 테스트데이터가 훈련데이터와 특성이 같다면 정확하게 예측될 것이다. 이와 같이 처음보는 데이터를 정확하게 예측할 수 있다면 훈련세트가 테스트세트로 일반화 되었다고 한다.
    - 과대적합
      - 너무 많은 정보를 이용하여 너무 복잡한 모델이 만들어 지는 현상을 ‘모델이 과대적합 되었다’고 한다.
      - 훈련세트의 모든 샘플에 너무 가까이 적합(학습)되어 새로운 데이터에 일반화되기 어렵다
    - 과소적합
      - 너무 적은 정보를 이용하여 너무 간단한 모델이 만들어 지는 현상을 ‘모델이 과소적합 되었다’고 한다.
  - 모델의 복잡도와 데이터셋 크기



모델의 복잡도에 따른 훈련 및 평가세트의 정확도변화

1. 지도학습개요(cont.)

- 사례
  - 문제
    - 요트를 구매한 고객과 구매하지 않은 고객의 정보를 이용하여 누가 요트를 살지 예측한다.
  - 데이터 작성
    -
  - 모델 (규칙) 작성
    - 보트를 산 사람을 기준으로 규칙은 많을 수 있다.
      - 나이가 45이상,차량을 보유,주택보유,자녀가2이상인 사람
        - 너무 복잡
      - 나이가 66,52,53,58세 인 사람
      - 새로운 데이터에 대하여 잘 맞을까?
      - 간단한 모델이 유효

표 2-1 고객 샘플 데이터

나이	보유차량수	주택보유	자녀수	혼인상태	애완견	보트구매
66	1	yes	2	사별	no	yes
52	2	yes	3	기혼	no	yes
22	0	no	0	기혼	yes	no
25	1	no	1	미혼	no	no
44	0	no	2	이혼	yes	no
39	1	yes	2	기혼	yes	no
26	1	no	2	미혼	no	no
40	3	yes	1	기혼	yes	no
53	2	yes	2	이혼	no	yes
64	2	yes	3	이혼	no	no
58	2	yes	2	기혼	yes	yes
33	1	no	1	미혼	no	no

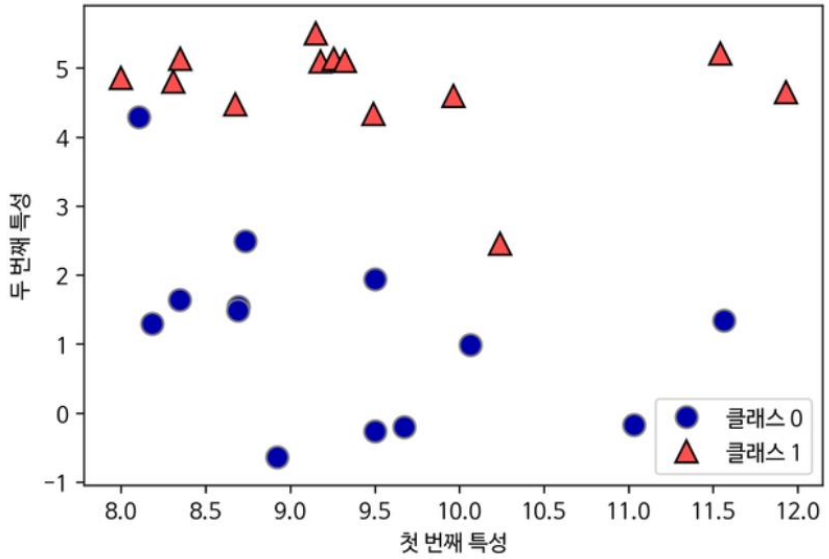
2. 예제에서 사용할 데이터 셋

- 머신러닝 알고리즘의 모델
  - 모델의 사용법
  - 모델의 복잡도 와 성능
  - 모델의 장단점
  - 모델의 매개변수와 옵션
- 2.1 forge 데이터셋
  - 두 개의 특성을 가진 forge 인위적으로 만들어지는 데이터셋 : KNN

Forge

	X[:,,0]	X[:,1]	y
0	9.963466	4.596765	1
1	11.032954	-0.168167	0
…			
24	9.150723	5.498322	1
25	11.563957	1.338940	0

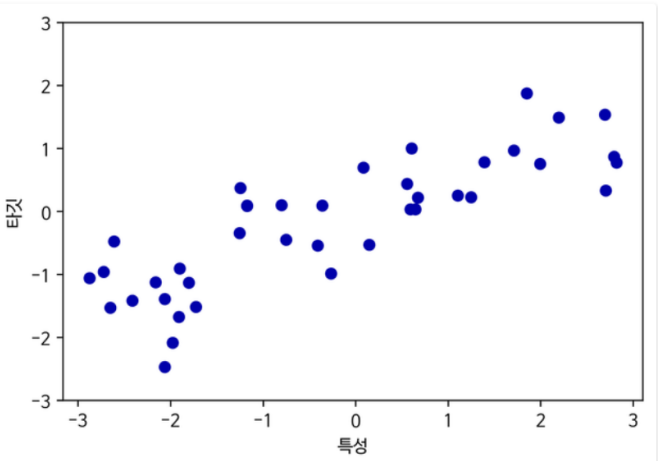
```
# 데이터셋을 만듭니다.
X, y = mglearn.datasets.make_forge()
# 산점도를 그립니다.
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
plt.legend(["클래스 0", "클래스 1"], loc=4)
plt.xlabel("첫 번째 특성")
plt.ylabel("두 번째 특성")
plt.show()
print("X.shape: {}".format(X.shape))  #(26, 2)
```



2. 예제에서 사용할 데이터 셋(cont.)

- 2.2 wave 데이터셋
  - 회귀 알고리즘 설명에는 인위적으로 만든 wave 데이터셋

```
import matplotlib.pyplot as plt
import mglearn
X, y = mglearn.datasets.make_wave(n_samples=40)  #X:(40, 1)   y:[(40,) float64]
print('X:{}   y:[{} {}]'.format(X.shape,y.shape,y.dtype)) #X:(40, 1)   y:[(40,) float64]
plt.plot(X, y, 'o')
plt.ylim(-3, 3)
plt.xlabel("특성")
plt.ylabel("타깃")
plt.show()
```



Wave

	X	y
0	-0.752759	-0.448221
1	2.704286	0.331226
…		
37	-2.413967	-1.415024
38	1.105398	0.254389
39	-0.359085	0.093989

- 2.3 위스콘신 유방암Wisconsin Breast Cancer 데이터셋

```
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()  #data:(569, 30)   target:[(569,) int32 (2,)]
print(cancer.keys())
        #['data', 'target', 'target_names', 'DESCR', 'feature_names', 'filename']
print('data:{}   target:[{} {} {}]'.format(cancer.data.shape, cancer.target.shape,
        cancer.target.dtype,cancer.target_names.shape))
        #data:(569, 30)   target:[(569,) int32 (2,)]
```

2. 예제에서 사용할 데이터 셋(cont.)

- 2.4 보스턴 주택가격Boston Housing 데이터셋
  - 범죄율, 찰스강 인접도, 고속도로 접근성 등의 정보를 이용해 1970년대 보스턴 주변의 주택 평균 가격

```
from sklearn.datasets import load_boston
boston = load_boston()
print("데이터의 형태: {}".format(boston.data.shape))
# 데이터의 형태: (506, 13)
```

- 2.5 보스턴 주택가격Boston Housing 확장 데이터셋
  - 13개의 입력 특성뿐 아니라 특성끼리 곱하여(또는 상호작용이라 부름) 의도적으로 확장
  - 예, 범죄율과 고속도로 접근성의 개별 특성은 물론, 범죄율과 고속도로 접근성의 곱도 특성으로 생각한다는 뜻
  - **특성 공학**feature engineering

```
X, y = mglearn.datasets.load_extended_boston()
print("X.shape: {}".format(X.shape))
#X.shape: (506, 104)
```

- 요약

```
import mglearn
import sklearn
import numpy as np

X, y      = mglearn.datasets.make_forge()                 #X:(26, 2)              y:(26,),    int32, [0, 1], 분류용
X, y      = mglearn.datasets.make_wave(n_samples=40)      #X:(40, 1)              y:(40,),    float64,       회귀용
Cancer = sklearn.datasets.load_breast_cancer()            #data:(569, 30)  target:(569,),  int32, [0, 1]  분류용
Boston = sklearn.datasets.load_boston()                   #data:(506, 13)  target:(506,),  float64        회귀용
boston_ext= mglearn.datasets.load_extended_boston()       #data:(506, 104) target:(506,), float64        회귀용
X,y       =sklearn.datasets.make_blobs(random_state=42)#X:(100, 2)              y:(100,), int32, [0, 1, 2]] 분류용
```

Forge

	X[:,0]	X[:,1]	y
0	9.963466	4.596765	1
1	11.032954	-0.168167	0
…			
24	9.150723	5.498322	1
25	11.563957	1.338940	0

Wave

	X	y
0	-0.752759	-0.448221
1	2.704286	0.331226
…		
38	1.105398	0.254389
39	-0.359085	0.093989

Cnacer

	0	1	...	29	target
0	17.990	10.38	...	0.11890	0
1	20.570	17.77	...	0.08902	0
…					
567	20.600	29.33	...	0.12400	0
568	7.760	24.54	...	0.07039	1

Boston

	0	1	...	29	target
0	17.990	10.38	…	0.11890	0
1	20.570	17.77	…	0.08902	0
…					
567	20.600	29.33	…	0.12400	0
568	7.760	24.54	…	0.07039	1

Blobs

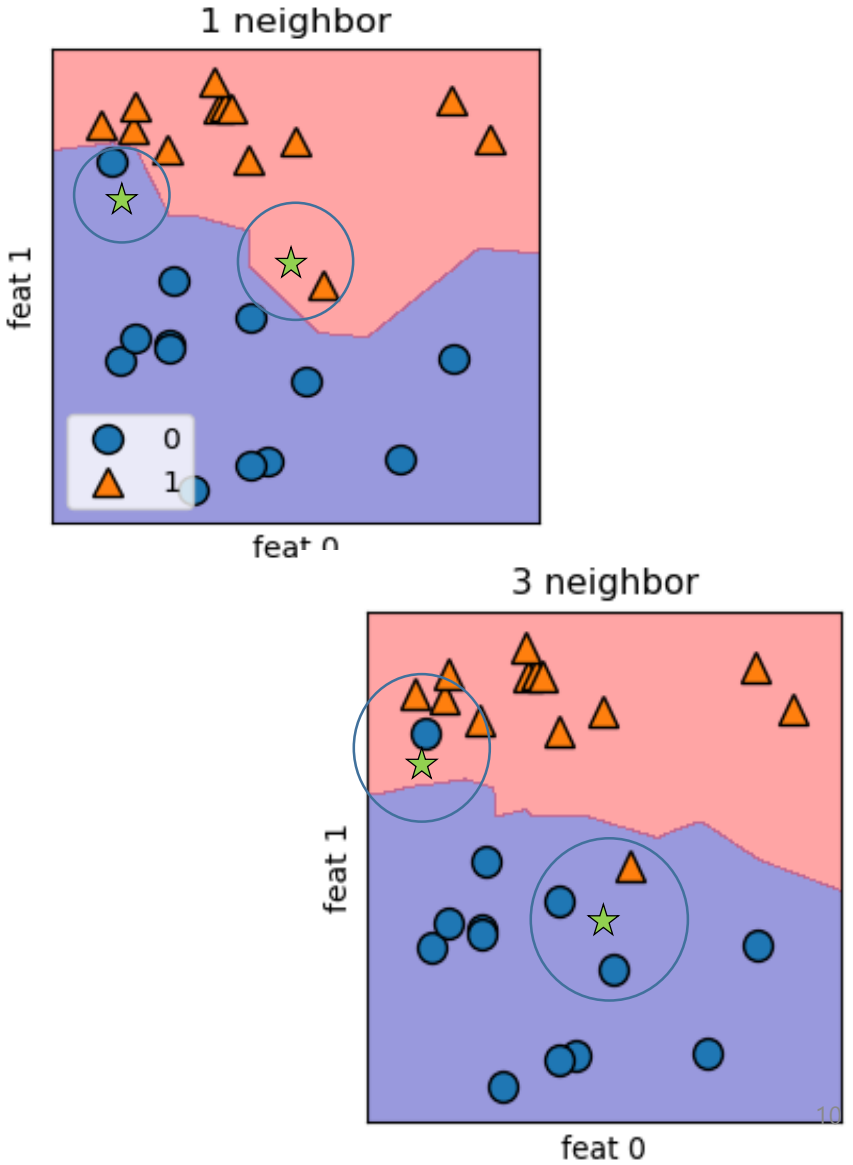
	X[:,0]	X[:,1]	y
0	-7.726421	-8.394957	2
1	5.453396	0.742305	1
…			
98	-5.796576	-5.826308	2
99	-3.348415	8.705074	0

3. k-최근접 이웃

- 3. k-최근접 이웃 분류
  - k-NN(k-Nearest Neighbors) 알고리즘은 가장 간단한 머신러닝 알고리즘입니다. 훈련 데이터셋을 그냥 저장하는 것이 모델을 만드는 과정의 전부입니다. 새로운 데이터 포인트에 대해 예측할 땐 알고리즘이 훈련 데이터셋에서 가장 가까운 데이터 포인트, 즉 '최근접 이웃'을 찾습니다.
- 3.1 k-최근접 이웃 분류
  - 3.1.1 K-최근접 이웃 분류 개요
  - 3.1.2 K-최근접 이웃 분류 알고리즘 KNeighbordClassifiers
- 3.2 K-최근접 이웃 회귀
  - 3.2.1 K-최근접 이웃 회귀 개요
  - 3.2.2 K-최근접 이웃 회귀 알고리즘 KNeighborsRegressor

3.1.1 k-최근접 이웃 개요

- 3.1.1 k-최근접 이웃 분류 개요
  - 모델
    - 모델이 학습데이터를 그대로 보유
  - 분류방법
    - 임의로 주어지는 평가 데이터로부터
      모델데이터와 가장 가까운 k개의 대상데이터셋을 만들고
      대상데이터셋에서 최대 클래스를 분류결과로 판단한다.
  - 1 –Nearest Neighbor 모델의 예
    - 모델 : 13△ ,14● sample로 구성
    - 인식
      - 평가 샘플 ★ 대상클래스셋 ● 결과 ●
      - 평가 샘플 ★ 대상클래스셋 △ 결과 △
  - 3 –Nearest Neighbor의 예
    - 모델 : 13△ ,14● sample로 구성
    - 인식
      - 평가 샘플 ★ 대상클래스셋 △△● 결과 △
      - 평가 샘플 ★ 대상클래스셋 △●● 결과 ●



3.1.2 KNeighborsClassifier 알고리즘

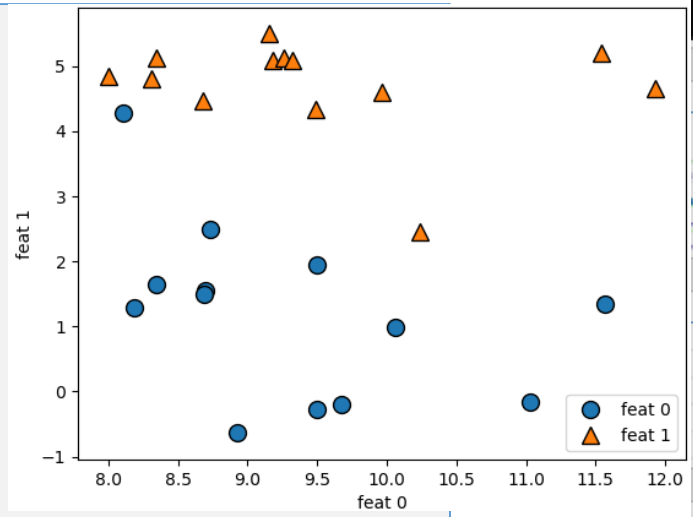
- 3.1.2 K-최근접 이웃 분류 알고리즘 KNeighbordClassifiers

```
X,y=mglearn.datasets.make_forge()       #인위적 데이터셋 적재


X_train,X_test,y_train,y_test=train_test_split(X,y,random_state=0) #학습 및 평가셋으로 분리


#모델 생성 및 평가
knn = KNeighborsClassifier(n_neighbors=3)          #모델 정의(생성)
knn.fit(X_train, y_train)                          #모델학습
print('모델예측  : {}'.format(knn.predict(X_test)))  #예측 [1 0 1 0 1 0 0]
print("모델정확도 : {:2f}".format(knn.score(X_test,y_test))) #모델평가 정확도:86%
```

3.1.2 KNeighborsClassifier 알고리즘(cont.)

- KNeighborsClassifier의 사용법
  - forge 데이터 셋 적재, 산점도, 훈련/평가셋생성, 모델생성,훈련,평가

```
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import mglearn
from sklearn.neighbors import KNeighborsClassifier
#데이터셋 적재
X,y=mglearn.datasets.make_forge()      #인위적으로 데이터셋 생성
print(pd.DataFrame(                    #데이터셋 출력
        [list(xx)+[yy] for xx, yy in zip(X,y)],    #X,y데이터셋 zip
        columns=['feat 0','feat 1','class']))      #특성 및 label

mglearn.discrete_scatter(X[:,0],X[:,1],y)#산점도 분석
plt.xlabel('feat 0')
plt.ylabel('feat 1')
plt.legend(['feat 0','feat 1'],loc=4)
plt.show()

X_train,X_test,y_train,y_test=train_test_split(X,y,random_state=0) #학습 및 평가셋으로 분리
print('데이터 shape\nX_train:{} X_test:{} y_train:{}  y_test:{}      #데이터셋의 shape 출력
        '.format(X_train.shape,X_test.shape,y_train.shape,y_test.shape)

#모델 생성 및 평가
print('\n\nKNeighborsClassifier 모델 생성 및 평가')
knn = KNeighborsClassifier(n_neighbors=1)          #모델 정의(생성)
knn.fit(X_train, y_train)                          #모델학습
print('모델예측  : {}'.format(knn.predict(X_test)))   #모델 샘플의 예측
print('모델타킷  : {}'.format(y_test))
print("모델성능 : {:2f}".format(knn.score(X_test,y_test))) #모델평가
```

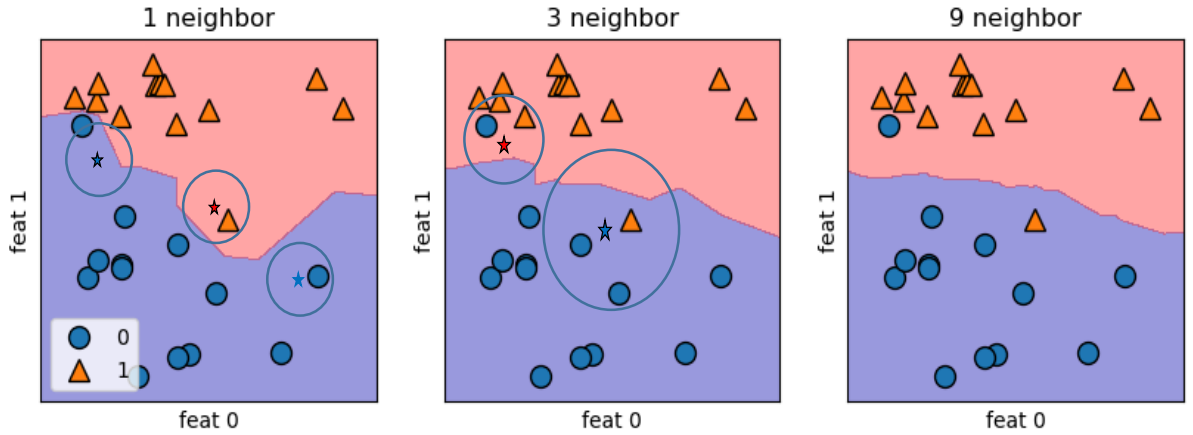



```
      feat 0    feat 1  class
0   9.963466  4.596765      1
1  11.032954 -0.168167      0
2  11.541558  5.211161      1
3   8.692890  1.543220      0
4   8.106227  4.286960      0
5   8.309889  4.806240      1
6  11.930271  4.648663      1
7   9.672847 -0.202832      0
8   8.348103  5.134156      1
9   8.674947  4.475731      1
10  9.177484  5.092832      1
11 10.240289  2.455444      1
12  8.689371  1.487096      0
13  8.922295 -0.639932      0
14  9.491235  4.332248      1
15  9.256942  5.132849      1
16  7.998153  4.852505      1
17  8.183781  1.295642      0
18  8.733709  2.491624      0
19  9.322983  5.098406      1
20 10.063938  0.990781      0
21  9.500490 -0.264303      0
22  8.344688  1.638243      0
23  9.501693  1.938246      0
24  9.150723  5.498322      1
25 11.563957  1.338940      0
데이터 shape
X_train:(19, 2) X_test:(7, 2) y_train:(19,)  y_test:(7,)


KNeighborsClassifier 모델 생성 및 평가
모델예측  : [1 0 1 0 1 0 0]
모델타킷  : [1 0 1 0 1 1 0]
모델정확도 : 0.857143
Press any key to continue . . .
```

3.1.2 KNeighborsClassifier 알고리즘(cont.)

- K값에 따른 KNeighborsClassifier 결정경계(성능)의 분석

```
fig,axes=plt.subplots(1,3, figsize=(10,3))      #크기 (10,3)인치의 캔버스, 3개의 하위그림 축 생성
for k_neighbor, ax in zip([1,3,9],axes) :       #K=[1,3,9]를 축[0,1,2]을 zip
        knn = KNeighborsClassifier(k_neighbors=n_neighbor).fit(X,y) #kNN모델 생성 및 학습
        mglearn.plots.plot_2d_separator(        # ax축에 knn모델의 결정경계를 생성한다.
                knn, X, fill=True, eps=0.5,ax=ax, alpha=.4)
        mglearn.discrete_scatter(          #ax축에 특징의 산점을 배치한다.
                X[:,0], X[:,1], y, ax=ax)          #
        ax.set_title( " {} neighbor " .format(  #축의 title 설정
                k_neighbor))                       #현재의 이웃 수
        ax.set_xlabel('feat 0')                    #ax축의 x축 라벨
        ax.set_ylabel('feat 1')                    #ax축의 y축 라벨
axes[0].legend(loc=3)                              #범례를 ax0의 모서리 3에 배치
plt.show()                                         #화면에 표시
```



- 3 neighbor 가 best ?
- 1 neighbor 모델
  - 가장복잡,복잡한 경계, 과대적합, 훈련데이터 100%정확

3.1.2 KNeighborsClassifier 알고리즘(cont.)

- K(복잡도)에 따른 KNeighborsClassifier 예측 정확도(성능) 분석
  - 유방암 데이터 셋 (569,30)

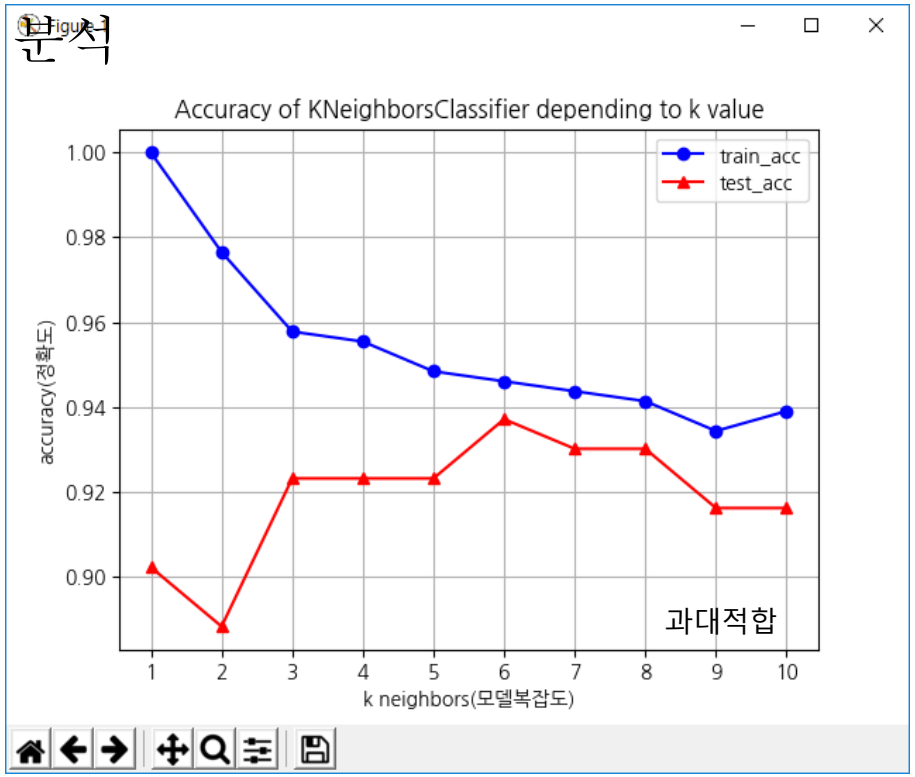
```
from sklearn.model_selection import train_test_split
from sklearn import datasets as skds
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import mglearn
from sklearn.neighbors import KNeighborsClassifier


#유방암 데이터 셋 (569,30)
#K값에 따른 KNeighborsClassifier  예측 정확도(성능) 분석
#데이터셋 적재
ds_cancer=skds.load_breast_cancer()      #유방암 데이터셋


print(pd.DataFrame(                      #데이터셋의 기본정보  출력
     [list(xx)+[yy] for xx, yy in zip(ds_cancer.data,ds_cancer.target)] ))        # 특징,타켓 통합


X_train,X_test,y_train,y_test=train_test_split(                                   #학습 및 평가셋 생성
     ds_cancer.data,ds_cancer.target,stratify=ds_cancer.target,random_state=66)

print('데이터 shape₩nX_train:{} X_test:{} y_train:{}  y_test:{}                  #데이터셋 형태  출력
     '.format(X_train.shape,X_test.shape,y_train.shape,y_test.shape))
```

```
      0      1       2       3       4       5       6   ...      24      25      26      27      28      29  30
0   17.99  10.38  122.80  1001.0  0.11840  0.27760  0.30010  ...  0.16220  0.66560  0.7119  0.2654  0.4601  0.11890   0
1   20.57  17.77  132.90  1326.0  0.08474  0.07864  0.08690  ...  0.12380  0.18660  0.2416  0.1860  0.2750  0.08902   0
2   19.69  21.25  130.00  1203.0  0.10960  0.15990  0.19740  ...  0.14440  0.42450  0.4504  0.2430  0.3613  0.08758   0
3   11.42  20.38   77.58   386.1  0.14250  0.28390  0.24140  ...  0.20980  0.86630  0.6869  0.2575  0.6638  0.17300   0
4   20.29  14.34  135.10  1297.0  0.10030  0.13280  0.19800  ...  0.13740  0.20500  0.4000  0.1625  0.2364  0.07678   0
..    ...    ...     ...     ...      ...      ...      ...  ...      ...      ...     ...     ...     ...  ..
564 21.56  22.39  142.00  1479.0  0.11100  0.11590  0.24390  ...  0.14100  0.21130  0.4107  0.2216  0.2060  0.07115   0
565 20.13  28.25  131.20  1261.0  0.09780  0.10340  0.14400  ...  0.11660  0.19220  0.3215  0.1628  0.2572  0.06637   0
566 16.60  28.08  108.30   858.1  0.08455  0.10230  0.09251  ...  0.11390  0.30940  0.3403  0.1418  0.2218  0.07820   0
567 20.60  29.33  140.10  1265.0  0.11780  0.27700  0.35140  ...  0.16500  0.86810  0.9387  0.2650  0.4087  0.12400   0
568  7.76  24.54   47.92   181.0  0.05263  0.04362  0.00000  ...  0.08996  0.06444  0.0000  0.0000  0.2871  0.07039   1

[569 rows x 31 columns]
데이터 shape
X_train:(426, 30) X_test:(143, 30) y_train:(426,)  y_test:(143,)
```

3.1.2 KNeighborsClassifier 알고리즘(cont.)

- K(복잡도)에 따른 KNeighborsClassifier 예측 정확도(성능) 분석

```
#K값에 따른 KNeighborsClassifier 예측 정확도(성능)

#유방암 데이터셋 로드 및 출력
ds_cancer=skds.load_breast_cancer()
print(pd.DataFrame(
        [list(xx)+[yy] for xx, yy in zip(ds_cancer.data,ds_cancer.target)])) #데이터셋 출력

#학습 및 평가셋 생성 및 shape 출력
X_train,X_test,y_train,y_test=train_test_split(
ds_cancer.data,ds_cancer.target,stratify=ds_cancer.target,random_state=66)
print('데이터 shape\nX_train:{} X_test:{} y_train:{}  y_test:{} '.format(
        X_train.shape,X_test.shape,y_train.shape,y_test.shape))

train_acc=[];test_acc=[]
ks=range(1,11)        #최근 이웃 수 k리스트 생성
for k in ks :
        knn=KNeighborsClassifier(k).fit(X_train,y_train)#knn모델 생성 및 학습
        train_acc.append(knn.score(X_train,y_train))     #모델의 훈련데이터 정확도
        test_acc.append( knn.score(X_test,y_test))       #모델의 평가데이터 정확도
plt.plot(ks,train_acc,'bo-',label='train_acc')        #훈련데이터정확도 그래프
plt.plot(ks,test_acc,'r^-',label='test_acc')          #평가데이터정확도 그래프
plt.xlabel('k neighbors(모델복잡도)')     #x축 라벨
plt.ylabel('accuracy(정확도)')       #y축 라벨
plt.xticks(ks)                       #x축 눈금 설정
plt.legend(loc='best')               #범례 배치
plt.title('Accuracy of KNeighborsClassifier depending to k value')
plt.grid()                           #그리드 배치
plt.show()                           #그림 출력
```

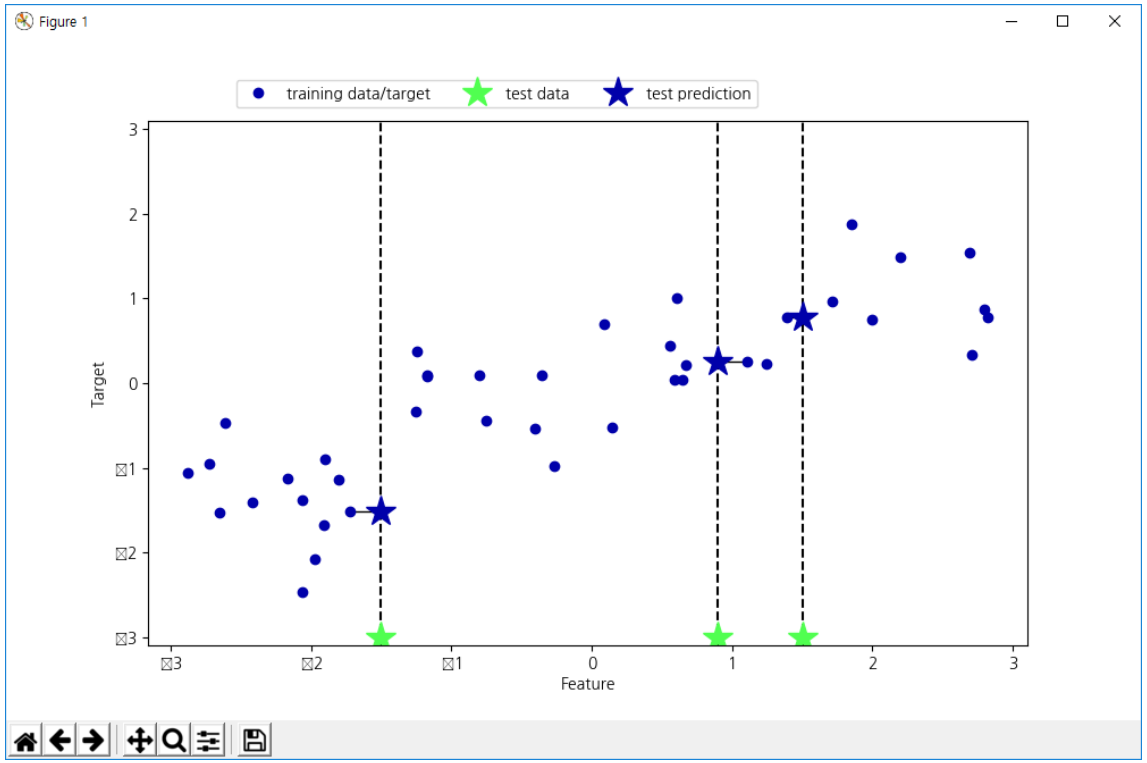


K=1 모델 복잡도, 훈련데이터 100%, 평가데이터 최저
k증가시 점점 단순 정확도 낮아진다.
K=6에서 최적합 (평가데이터 정확도 최대)
k=10 모델 최소단순

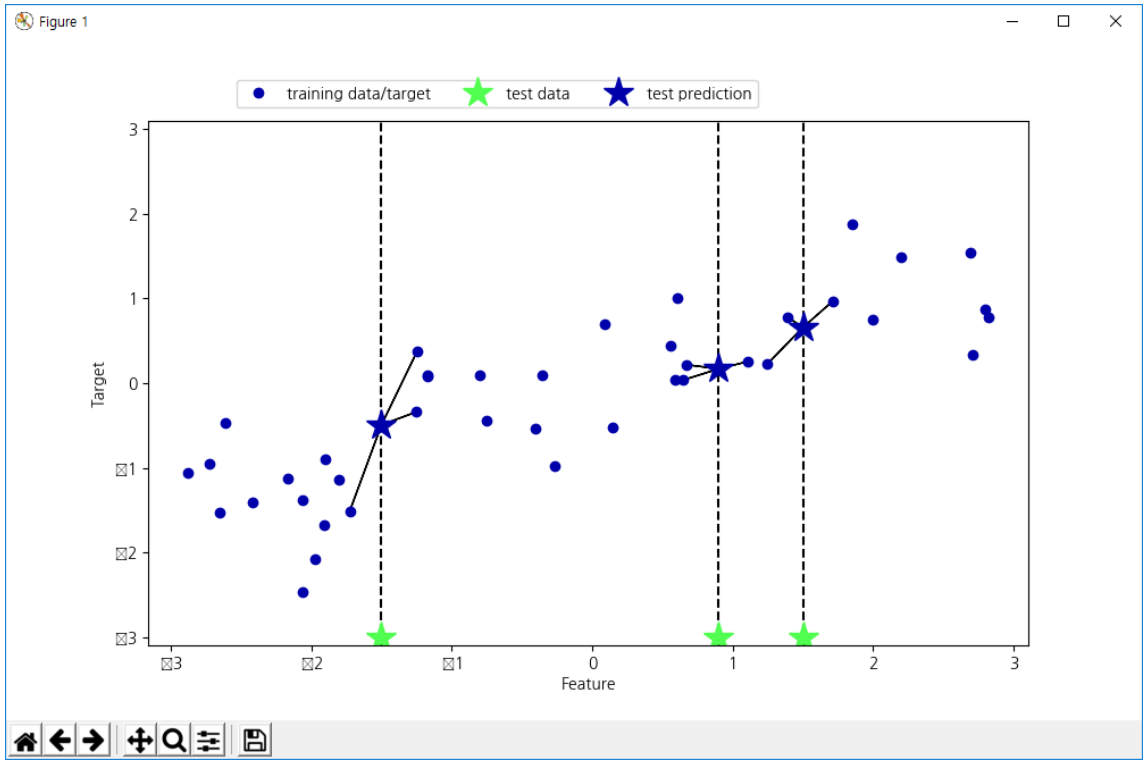
3.2 k 최근접 이웃 회귀

- 3.2.1 k 최근접 이웃 회귀(KNeighborsRegressor)개요
  - 최근접 이웃 분류기를 이용한 회귀–KNeighborsRegressor

```
#wave 데이터셋을 이용한  k-최근접 이웃 회귀, x축에 평가용 1개의 녹색 별에 대한 예측방법
mglearn.plots.plot_knn_regression(n_neighbors=1)
plt.show()
```




```
#wave 데이터셋을 이용한  k-최근접 이웃 회귀, x축에 평가용 3개의 녹색 별에 대한 예측방법
mglearn.plots.plot_knn_regression(n_neighbors=3)
plt.show()
```



3.2.2 k 최근접 이웃 회귀(kNeighborsRegressor)

- 2.3.2 k 최근접 이웃 회귀(KNeighborsRegressor)

```
X, y = mglearn.datasets.make_wave(n_samples=40) #(40,1), (40,)

# wave 데이터셋을 훈련 세트와 테스트 세트로 나눕니다.
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
#(30,1), (10,1), (30,), (10,),

# 이웃의 수를 3으로 하여 모델의 객체를 만듭니다.
reg = KNeighborsRegressor(n_neighbors=3)
# 훈련 데이터와 타깃을 사용하여 모델을 학습시킵니다.
reg.fit(X_train, y_train)

print("테스트 sample     : {}".format([X_test[0]]))
print("테스트 sample 예측: {}".format(reg.predict([X_test[0]])))

print("테스트 세트      :{}".format(X_test))
print("테스트 세트 예측 :{}".format(reg.predict(X_test)))
print("테스트 세트 라벨 :{}".format(y_test))
print("테스트 성능 R^2  : {:.2f}".format(reg.score(X_test, y_test)))
```

```
테스트 sample      :  [array([-1.24713211])]
테스트 sample 예측 : [-0.05396539]

테스트 세트        : [[-1.24713211], [ 0.67111737], [ 1.71105577], [-2.06388816], [-2.87649303],
 [-1.89957294], [ 0.55448741], [ 2.81945911], [-0.40832989], [-2.72129752]]

테스트 세트 예측 : [-0.05396539  0.35686046  1.13671923 -1.89415682 -1.13881398
-1.63113382  0.35686046  0.91241374 -0.44680446 -1.13881398]

테스트 세트 라벨 : [ 0.37299129  0.21778193  0.96695428 -1.38773632 -1.05979555
-0.90496988  0.43655826  0.7789638  -0.54114599 -0.95652133]

테스트 성능 R^2  : 0.83
```

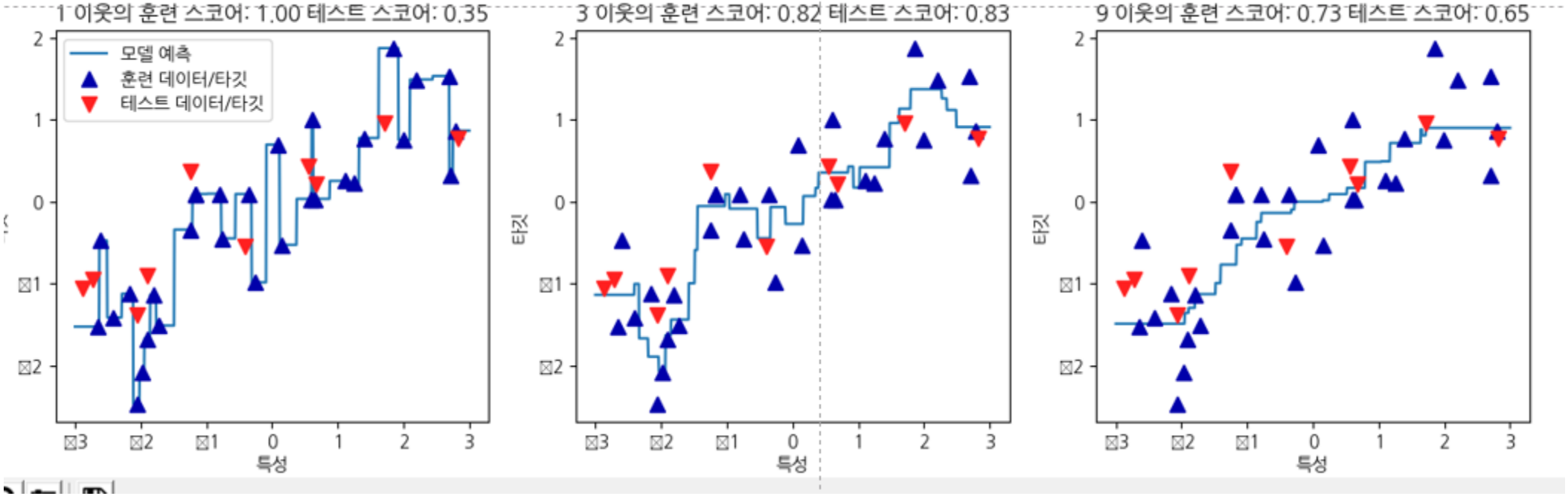
$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}, \hat{y}_i : \text{예측치}, \bar{y}: \text{mean}$$

3.2.2 2 k 최근접 이웃 회귀(KNeig

- 3.3.2 k 최근접 이웃 회귀(KNeighborsRegressor)분석

```
X, y = mglearn.datasets.make_wave(n_samples=40)
# wave 데이터셋을 훈련 세트와 테스트 세트로 나눕니다.
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
fig, axes = plt.subplots(1, 3, figsize=(15, 4))
# -3과 3 사이에 1,000개의 데이터 포인트를 만듭니다.
line = np.linspace(-3, 3, 1000).reshape(-1, 1)
```

```
for n_neighbors, ax in zip([1, 3, 9], axes):
        # 1, 3, 9 이웃을 사용한 예측을 합니다.
        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(
    "{} 이웃의 훈련 스코어: {:.2f} 테스트 스코어: {:.2f}".format(
            n_neighbors, reg.score(X_train, y_train),  reg.score(X_test, y_test)))
        ax.set_xlabel("특성");        ax.set_ylabel("타깃")
axes[0].legend(["모델 예측", "훈련 데이터/타깃", "테스트 데이터/타깃"],
         loc="best")
plt.show()
```

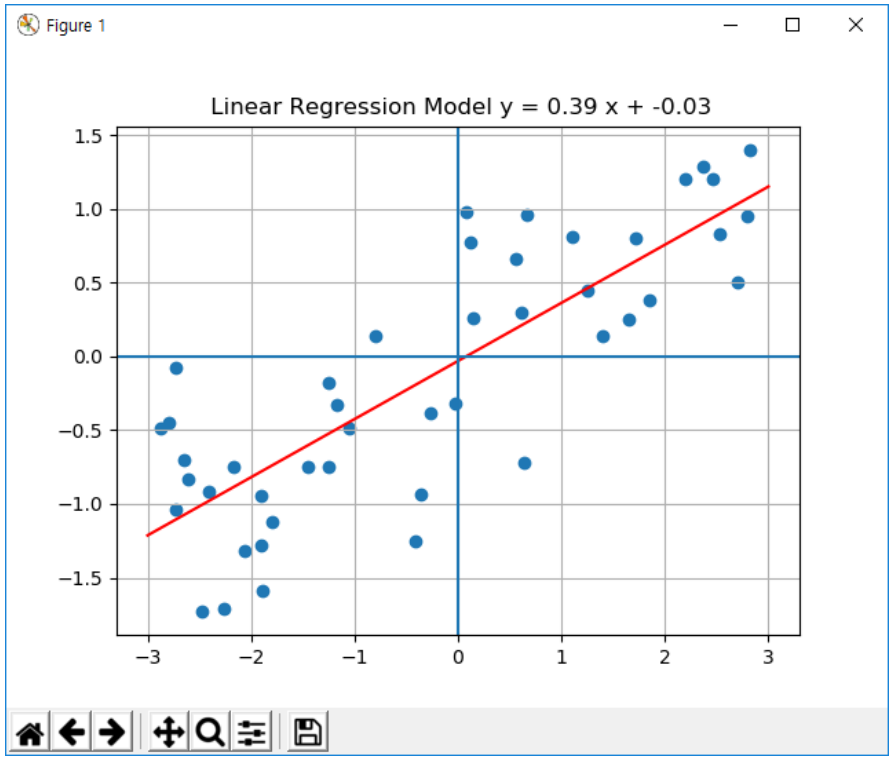


3.2.2 k 최근접 이웃 회귀(KNeighborsRegressor)분석

- 일반적으로 KNeighbors 분류기에 중요한 매개변수는 두 개입니다. 데이터 포인트 사이의 거리를 재는 방법과 이웃의 수입니다. 실제로 이웃의 수는 3개나 5개 정도로 적을 때 잘 작동하지만, 이 매개변수는 잘 조정해야 합니다. 거리 재는 방법을 고르는 문제는 이 책에서 다루지 않습니다만, 기본적으로 여러 환경에서 잘 동작하는 유클리디안 거리 방식을 사용합니다.
- k-NN의 장점은 이해하기 매우 쉬운 모델이라는 점입니다. 그리고 많이 조정하지 않아도 자주 좋은 성능을 발휘합니다. 더 복잡한 알고리즘을 적용해보기 전에 시도해볼 수 있는 좋은 시작점입니다. 보통 최근접 이웃 모델은 매우 빠르게 만들 수 있지만, 훈련 세트가 매우 크면 (특성의 수나 샘플의 수가 클 경우) 예측이 느려집니다. k-NN 알고리즘을 사용할 땐 데이터를 전처리하는 과정이 중요합니다. 그리고 (수백 개 이상의) 많은 특성을 가진 데이터셋에는 잘 동작하지 않으며, 특성 값 대부분이 0인 (즉 희소한) 데이터셋과는 특히 잘 작동하지 않습니다.
- k-최근접 이웃 알고리즘이 이해하긴 쉽지만, 예측이 느리고 많은 특성을 처리하는 능력이 부족해 현업에서는 잘 쓰지 않습니다. 이런 단점이 없는 알고리즘이 다음에 설명할 선형 모델입니다.

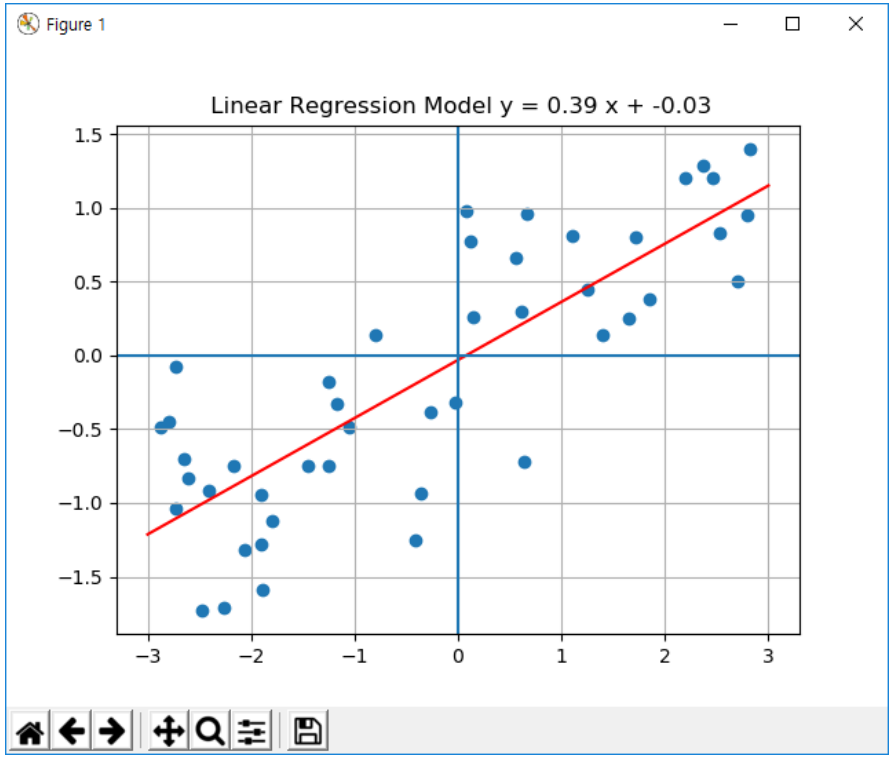
4 선형모델

- 4.1 회귀의 선형모델
  - 선형예측함수를 사용
  - $\hat{y} = w_0 x_0 + w_1 x_1 + \cdots + w_p x_p + b$
    - $\hat{y}$: 모델의 예측 값
    - $\{w_i\}, b$: 학습할 파라메터, 특성,가중치
    - $\{x_i\}, \{y_i\}$: 주어지는 데이터(훈련 또는 평가용)
    - 일종의 선형결합
    - 선형모델의 종류
      - 특성이 하나이면 모델이 직선
        2 이면 평면, 그 이상이면 초평면(hyper plane)이 된다.
  - 예,
    - $\hat{y} = wx + b$



4.2 선형회귀(LinearRegression,최소제곱법)

- 4.2 선형회귀(LinearRegression,최소제곱법)
  - 선형 회귀는 예측과 훈련 세트에 있는 타깃 y 사이의 **평균제곱오차**mean squared error를 최소화하는 파라미터 w와 b를 찾습니다
  - 데이터 $X = \{x_i\}, Y = \{y_i\}$
  - 선형회귀모델
    - $\hat{y} = wx + b$
    - 손실함수(mean square error)
      - $loss(w,b) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i(w,b))^2$
    - 학습
      - $w^* = argmin_w(loss(w,b)))$
  - 평가지수
    - Score, 적합도 0~1
    - $R^2 = 1 - \frac{\sum(y-\hat{y})^2}{\sum(y-\bar{y})^2}$
      - $\hat{y}$: 모델 예측치
      - $\bar{y}$: target y의 평균
      - 모든 예측치가 평균치와 같으면 0
      - 모든 예측치가 target치와 같으면 1 완벽한 모델



4.2 선형회귀(최소제곱법)

- 선형회귀(LinearRegression)
  - 1차원 데이터셋(59,1) 선형 =>0.66 =>과소적합

```
def LR(X,y) :
        #학습(0.75) 및 평가 셋 생성 및 shape 출력
        X_train,X_test,y_train,y_test=train_test_split(X,y,random_state=42)
        print('데이터 shape\nX_train:{} X_test:{} y_train:{}  y_test:{}'.format(
                X_train.shape,X_test.shape,y_train.shape,y_test.shape))

        lr=LinearRegression().fit(X_train,y_train)    #lr 모델생성 및 손실함수 mse로 학습
        print('w : ',lr.coef_)        #w : [0.42073384] _는 scikit-learn에서 유도되는 데이터의 명명규칙
        print('b : ',lr.intercept_)   #b : -0.05983245830361808

        print('훈련세트 성능(점수) : {:.2f}'.format(lr.score(X_train,y_train)))
        print('평가세트 성능(점수) : {:.2f}'.format(lr.score(X_test,y_test)))

#1차원 wave data set생성
X,y=mglearn.datasets.make_wave(n_samples=60)
print('Sample wave 데이터세트₩n',pd.DataFrame(          #기봉 정보 출력
     [list(xx)+[yy] for xx, yy in zip(X,y)],             #X,y데이터셋  zip
     columns=list(range(X.shape[1]))+['target']).tail(3)) #마지막 3sample 출력
LR(X,y) #선형 회귀 알고리즘 호출
```

```
Sample wave 데이터세트 :
           0    target
57 -1.824103 -1.546646
58 -2.728636 -1.037316
59 -1.048018 -0.491317
데이터 shape  X_train:(45, 1) X_test:(15, 1) y_train:(45,)  y_test:(15

w :  [0.39390555]
b :  -0.031804343026759746
훈련세트 성능(점수) : 0.67
평가세트 성능(점수) : 0.66
```

4.2 선형회귀(최소제곱법)

- 선형회귀(LinearRegression)
  - 고차원 데이터셋(506,104) 초평면=>0.94,0.78 => 과대적합

```
#보스톤 주택가격 데이터 셋
#특성 104개, 샘플수 506
#104 차 특성 초평면 모델
X,y=mglearn.datasets.load_extended_boston()
print('보스톤 주택가격 데이터 셋\n',pd.DataFrame(
        [list(xx)+[yy] for xx, yy in zip(X,y)],#X,y데이터셋  zip
        columns=list(range(X.shape[1]))+['target']).tail(3)) #마지막 3sample 출력
LR(X,y)
```

```
보스톤 주택가격 데이터 셋
             0         1         2    3         4         5  ...        99       100       101       102       103  target
503  0.000612  0.0  0.420455  0.0  0.386831  0.654340  ...  0.893617  0.096414  1.000000  0.107892  0.011641    23.9
504  0.001161  0.0  0.420455  0.0  0.386831  0.619467  ...  0.885843  0.117127  0.982677  0.129930  0.017180    22.0
505  0.000462  0.0  0.420455  0.0  0.386831  0.473079  ...  0.893617  0.151649  1.000000  0.169702  0.028799    11.9

[3 rows x 105 columns]
데이터 shape  X_train:(379, 104) X_test:(127, 104) y_train:(379,)  y_test:(127,)

w :  [-5.11126504e+02  4.02559787e+00 -9.45778613e+01  1.34720251e+01
  3.48176257e+01  6.09611391e+01  3.49707471e+01  2.94114542e+00
  3.14525465e+00  8.20792132e+01  1.24254396e+01  3.86676075e+01
 -9.38409521e-01  1.32936334e+01  7.60317098e+02  1.42274855e+03
  2.29220565e+02 -7.79405429e+01  8.79429261e+01  1.39813973e+01
  1.02565346e+02  7.52178879e+02 -1.82071934e+03  5.34143172e+02
 -2.41122305e+01  1.11848898e+02 -4.38177813e+00 -1.23079894e+01
 -3.63360790e+00 -5.64878037e+01  4.60395879e-01  8.18005986e+00
 -2.06294404e+01 -3.49659791e+01  4.31717988e+01 -2.92220843e+00
  1.45250942e+01 -3.24346333e+01  3.66984591e+01 -2.75859278e+00
  6.27805740e+00  4.98379104e+01  6.55060318e+00  3.91047481e+01
 -1.14826290e+01 -8.00990322e-01 -3.68662287e+00  3.36483260e+01
 -1.49103502e+01  1.34720251e+01 -1.80244019e+01 -2.90956806e+01
 -2.78115796e+00 -1.10315060e+01  1.15584830e+00 -8.37313259e-01
 -7.89905136e+00  6.27950290e+00 -1.09538327e+01 -2.48389637e+01
 -1.16316264e+01 -3.00228631e+00  6.83518378e+01 -1.76428626e+01
  6.10371772e+01 -6.12936496e+01 -1.14748321e+01  2.09075528e+01
  3.32421356e+01 -4.11743288e+01 -2.19312422e+01 -2.08881337e+01
 -5.05858326e+01 -2.14714962e+01 -1.11593182e+01 -6.16458839e-01
 -1.12569338e+00 -1.40290786e-01  3.17622544e+01 -2.57159897e+01
  5.51837314e-01 -1.33768644e+01 -3.25170630e+01  5.20806824e+01
  1.08614313e-01 -3.62670514e+01 -2.68217433e+01 -3.42720513e+01
  1.41341012e+01 -6.56371258e+01  8.64151127e+01 -3.08281756e+01
  3.61562583e+01 -2.56736318e+01 -1.69118913e+01  3.35683331e+01
 -7.48792540e+01 -2.02885480e+01  3.35543349e+00  1.07705825e+01
  3.50306579e+00 -5.10021527e+00  2.46929457e+00  2.55749022e+01]
b :  -34.70752210387431
훈련세트 성능(점수) : 0.94
평가세트 성능(점수) : 0.78
```

- 단순모델이 필요
  =>복잡도를 제어할 수 있어야

4.3 릿지회귀

- 4.3 릿지회귀(RidgeRegression)
  - 개요
    - 최소자승 손실함수에 $L2$ 규제를 추가한 선형회귀모델
    - α를 이용하여 계수(w)가 작아지도록 제어한다.
    - 과대적합을 방지하기 위한 목적이다.
  - 데이터 $X = \{x_i\}, Y = \{y_i\}$
  - $L2$ 규제 선형회귀모델
    - $\hat{y} = wx + b$
    - 손실함수(mean square error)
      - $loss(w,b) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i(w,b))^2 + \alpha \sum_i w_i^2$
      - $L2 = \alpha \sum_i w_i^2$, $L2$ 규제(regularization)
      - α값이 크면 w의 크기가 작아진다.
        =>최적합을 탐색
    - 학습
      - $w^* = argmin_w(loss(w,b))$
      - 최소적합법
    - 평가지수
      - $R^2 = 1 - \frac{\sum(y-\hat{y})^2}{\sum(y-\bar{y})^2}$

Alpha에 따른 Ridge 회귀모델의 성능 분석

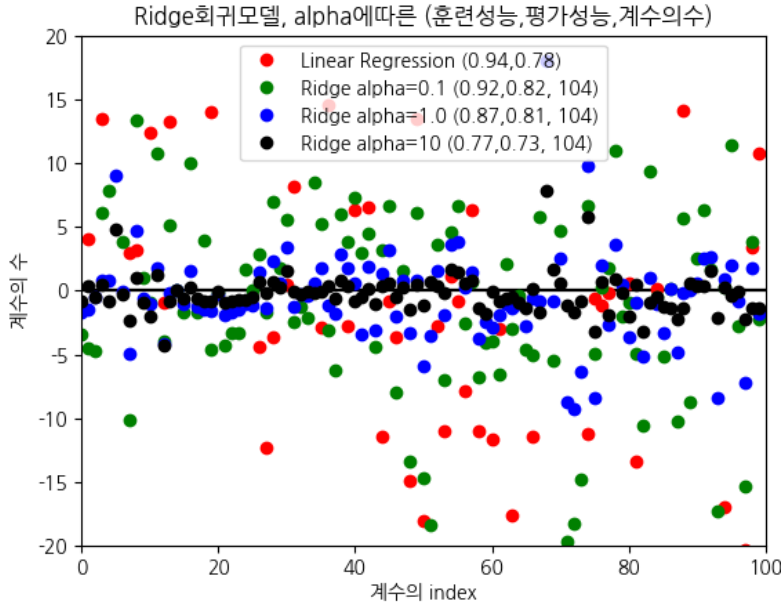
```
#X,y=mglearn.datasets.load_extended_boston()
print('Ridge Regression : ')
ridge=Ridge(alpha=1.0).fit(X_train,y_train) #lr 모델생성 및 손실함수 mse로 학습

print('훈련세트 성능(점수) : {:.2f}'.format(ridge.score(X_train,y_train)))
print('평가세트 성능(점수) : {:.2f}'.format(ridge.score(X_test,y_test)))
```

Linear Regression
훈련세트 성능(점수) : 0.94
평가세트 성능(점수) : 0.78

Ridge Regression, alpha=0.1
훈련세트 성능(점수) : 0.92
평가세트 성능(점수) : 0.82

Ridge Regression, alpha=1.0
훈련세트 성능(점수) : 0.87
평가세트 성능(점수) : 0.81

Ridge Regression, alpha=10
훈련세트 성능(점수) : 0.77
평가세트 성능(점수) : 0.73



LinearRegression보다
alpha=0.1 의 Ridge 회귀가 best
최적합의 alpha탐색이 필요

4.4 Lasso 회귀

- 4.4 Lasso 회귀
  - 개요
    - L1규제 선형회귀모델
    - 릿지 회귀에서와 같이 라쏘lasso도 계수를 0에 가깝게 만들려고 합니다.
    - L1 규제의 결과로 라쏘를 사용할 때 어떤 계수는 정말 0이 됩니다. 이 말은 모델에서 완전히 제외되는 특성이 생긴다는 뜻입니다. 어떻게 보면 특성 선택feature selection이 자동으로 이뤄진다고 볼 수 있습니다.
    - 일부 계수를 0으로 만들면 모델을 이해하기 쉬워지고 이 모델의 가장 중요한 특성이 무엇인지 드러내 줍니다.
  - Lasso회귀의 손실함수
    - $loss(w,b) = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - \hat{y}_i(w,b)\right)^2 + L1$
    - $L1 = \alpha \sum_i |w_i|$
    - W를 작게 만들기 위함, 일부 w는 0이 되어 w의 수가 줄어드는 효과를 본다.
      복잡도 가 작아진다.

Ridge 회귀의 손실함수

$loss(w,b) = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - \hat{y}_i(w,b)\right)^2 + \alpha \sum_i w_i^2$

4.4 Lasso 회귀(cont.)

- 라쏘 회귀 의 성능 분석

```
#lapha=1.0  Lasso 회귀
from sklearn.linear_model import Lasso
X, y = mglearn.datasets.load_extended_boston()  # (506, 104)

lasso = Lasso().fit(X_train, y_train)  #lapha=1.0

print("훈련 세트 점수: {:.2f}".format(lasso.score(X_train, y_train)))
print("테스트 세트 점수: {:.2f}".format(lasso.score(X_test, y_test)))
print("사용한 특성의 수: {}".format(np.sum(lasso.coef_ != 0)))
#훈련 세트 점수    : 0.29  #매우저조
#테스트 세트 점수: 0.21  #매우저조
#사용한 특성의 수: 4       #매우 적다,

#alpha=1.0=>규제가 너무 크게 되었다. 과소적합
```

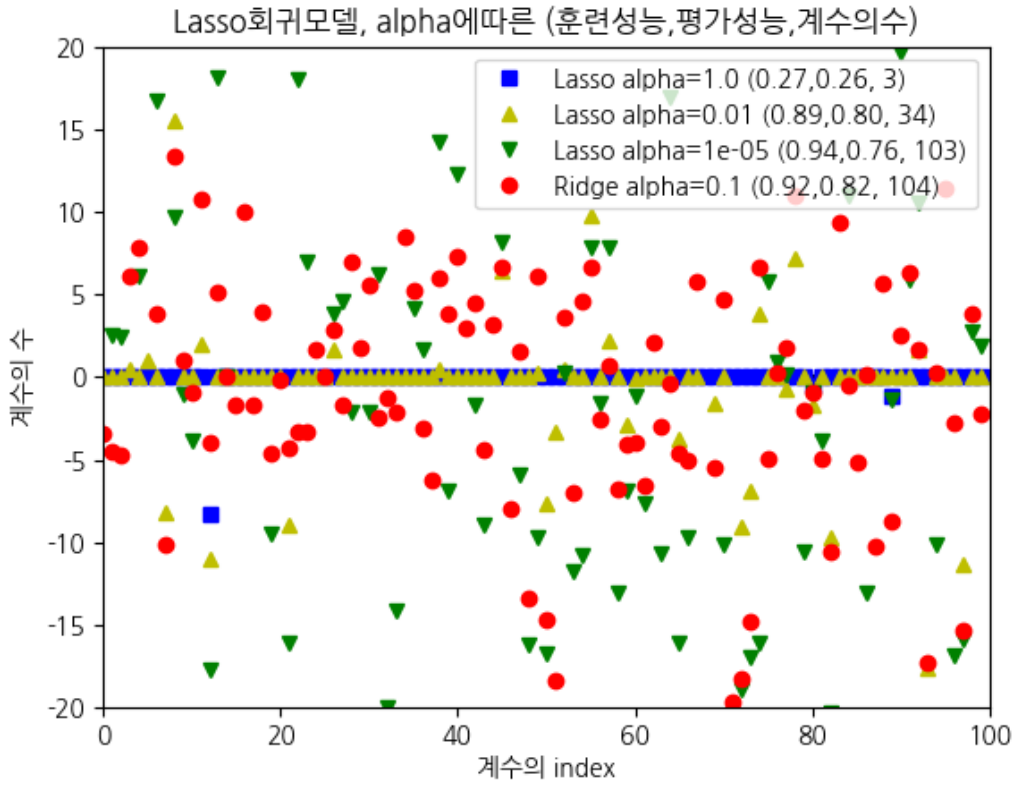
```
#lapha=0.01  Lasso 회귀
# "max_iter" 기본값을 증가시키지 않으면 max_iter 값을 늘리라는 경고가 발생합니다.

lasso001 = Lasso(alpha=0.01, max_iter=100000).fit(X_train, y_train)

print("훈련 세트 점수: {:.2f}".format(lasso001.score(X_train, y_train)))
print("테스트 세트 점수: {:.2f}".format(lasso001.score(X_test, y_test)))
print("사용한 특성의 수: {}".format(np.sum(lasso001.coef_ != 0)))
#훈련 세트 점수    : 0.90  #과대적합
#테스트 세트 점수: 0.77  #
#사용한 특성의 수: 33     #증가했다.

#alpha=0.01 => 규제가 너무 적게 되었다. 과대적합
```

- alpha 값을 낮추면 모델의 복잡도는 증가하여 훈련 세트와 테스트 세트에서의 성능이 좋아집니다. 성능은 Ridge보다 조금 나은데 사용된 특성은 105개 중 33개 뿐이어서, 아마도 모델을 분석하기가 조금 더 쉽습니다.
- alpha 값을 너무 낮추면 규제의 효과가 없어져 과대적합이 되므로 LinearRegression의 결과와 비슷해집니다.

4.4 Lasso 회귀(cont.)

- 라쏘 회귀의 성능 분석

```
def LassoR(X,y,alpha=1.0,color='r') :
        X_train,X_test,y_train,y_test=train_test_split(X,y,random_state=42)
        lm=Lasso(alpha=alpha).fit(X_train,y_train)#lr 모델생성 및 손실함수 mse로 학습
        plt.plot(lm.coef_[:100],color,
                label='Lasso alpha={} ({:.2f},{:.2f}, {})'.format(
                alpha,lm.score(X_train,y_train),lm.score(X_test,y_test),np.sum(lm.coef_ != 0)))
        plt.hlines(0, 0, 100)
        plt.xlim([0,100])
X,y=mglearn.datasets.load_extended_boston()
LassoR(X,y,1.0,'bs')
LassoR(X,y,0.01,'y^')
LassoR(X,y,0.00001,'gv')
RidgeR(X,y,0.1,'ro' )plt.lebends()
```



- 분석
  - alpha=1일 때 (이미 알고 있듯) 계수 대부분이 0일 뿐만 아니라 나머지 계수들도 크기가 작다, 성능저하,과소적합
  - alpha= 0.01로 줄이면 대부분의 특성이 0이 되는 (정삼각형 모양으로 나타낸) 분포
  - alpha=0.0001이 되면 계수 대부분이 0이 아니고 값도 커져 꽤 규제 받지 않은 모델
  - alpha=0.1인 Ridge 모델과
    alpha=0.01인 라쏘 모델은 성능이 비슷하지만 Ridge를 사용하면 어떤 계수도 0이 되지 않습니다.

  - 보통 Ridge 회귀 선호
  - 특성 중 일부만 중요하거나 간단한 모델이 중요하다면 Lasso가 유리
  - scikit-learn은 Lasso와 Ridge의 페널티를 결합한 ElasticNet도 제공합니다. 실제로 이 조합은 최상의 성능을 내지만 L1 규제와 L2 규제를 위한 매개변수 두 개를 조정해야 합니다

4.5 분류용 선형모델

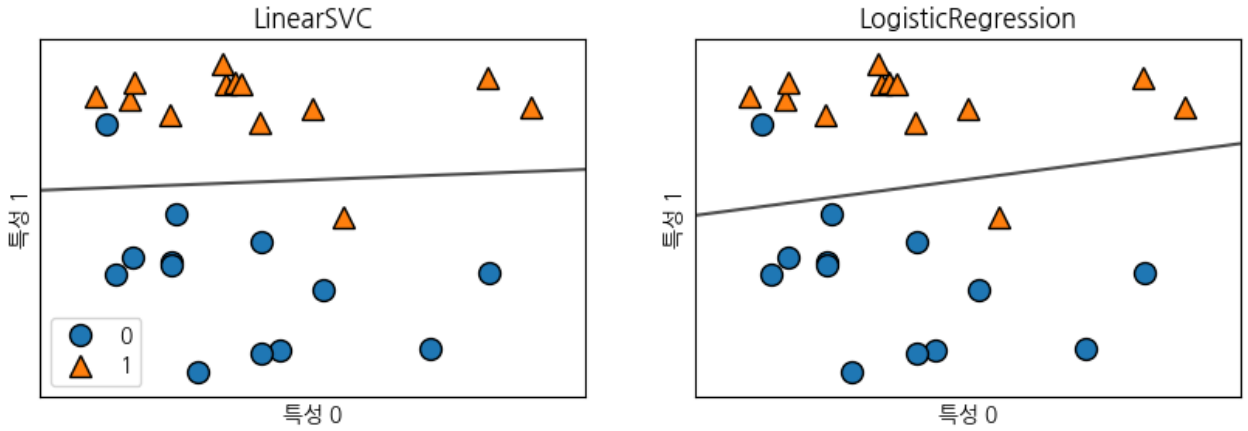
- 개요
  - 선형회귀 모델을 이용하여 분류하기
- 이진분류(binary classification)
  - $\hat{y}_l = w_0 x_0 + w_1 x_1 + \cdots + w_p x_p + b,$

    $$\hat{y} = f(\hat{y}_l) = \begin{cases} +1, \hat{y}_l > 0 \\ -1, \hat{y}_l < -1 \end{cases}$$

    - $\hat{y}$: 모델의 예측 값,
    - $\{w_i\}, b$: 학습할 파라메터, 특성,가중치
    - $\{x_i\}, \{y_i\}$: 주어지는 데이터(훈련 또는 평가용)
    - 회귀용 선형모델 $\hat{y}_l$의 결정경계(직선,평면, 초평면)를 기준으로 두 클래스로 구분한다.
  - 이진분류의 2가지 모델
    - linear_model.LogisticRegression(penalty=l2,C=1.0)
    - svm.LinearSVC(penalty=l2,C=1.0), SVC(support vector classifier)
    - $l2 = C \sum_i w_i^2$

4.5 분류용 선형모델(cont.)

- LinearSVC과 LogisticRegression 의 분석
  - 손실함수에 기본값 l2규제, C=1.0
    - penalty $l2 = C\sum_i w_i^2$
  - C=>0 : w를 0에 가까워진다.

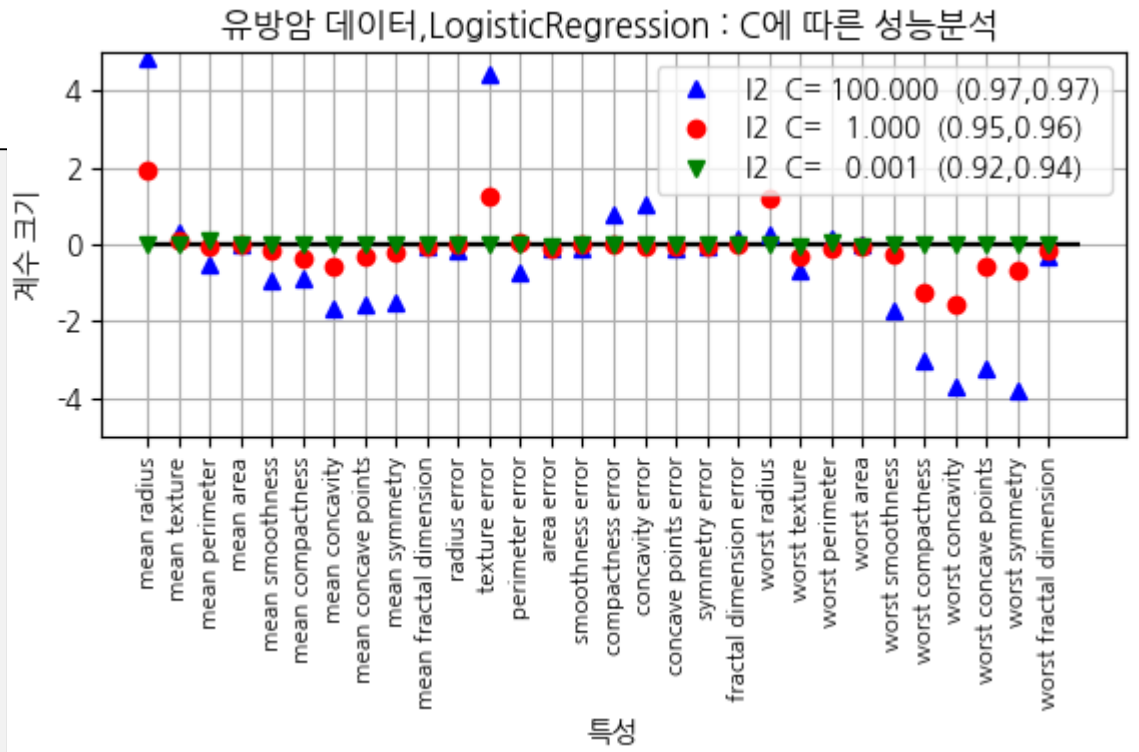



```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
import mglearn
import matplotlib.pyplot as plt

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("특성 0")
    ax.set_ylabel("특성 1")
    break
axes[0].legend()
plt.show()
```

4.5 분류용 선형모델(cont.)

- L2규제의 C값에
  - C,훈련성능,평가성능

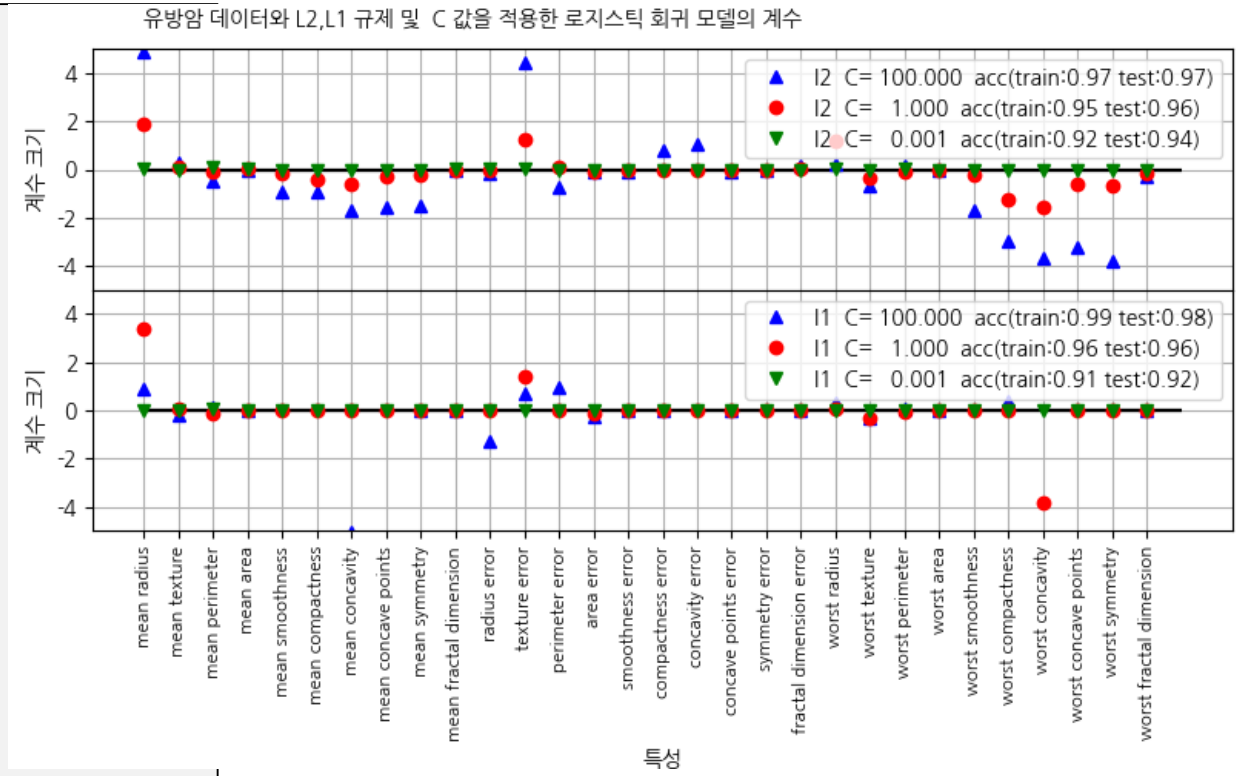
```
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(
cancer.data, cancer.target, stratify=cancer.target, random_state=42)

def LogisticReg(C=1.0,penalty='l2',color='b^',ax=None) :
        logreg = LogisticRegression(penalty=penalty,C=C).fit(X_train, y_train)
        score_train,score_test=logreg.score(X_train, y_train),logreg.score(X_test, y_test)
        ax.plot(logreg.coef_.T,color,
                label='{}  C={:8.3f}  acc(train:{:.2f} test:{:.2f})'.format(
                logreg.penalty,C,score_train,score_test))
        ax.hlines(0, 0, cancer.data.shape[1])
        ax.set_xticks(range(cancer.data.shape[1]))
        ax.set_xticklabels(cancer.feature_names,rotation=90,fontsize='small')
        ax.set_ylim(-5, 5);       ax.set_xlabel("특성"); ax.set_ylabel("계수 크기")
        ax.grid();                ax.legend(loc=1)
fig=plt.figure()
ax=fig.add_subplot()
LogisticReg(C=100,    color='b^',ax=ax)
LogisticReg(C=1,      color='ro',ax=ax)
LogisticReg(C=0.001,  color='gv',ax=ax)
lt.subplots_adjust(left=0.1, bottom=0.4, right=0.9, top=0.8, wspace=0, hspace=0)
plt.title('유방암 데이터,LogisticRegression : C에 따른 성능분석 ')
plt.show()
```



4.5 분류용 선형모델(cont.)

- 유방암 데이터와 L2, L1, C 값에 대하여 적용한 로지스틱 회귀 모델의 계수 및 정확도

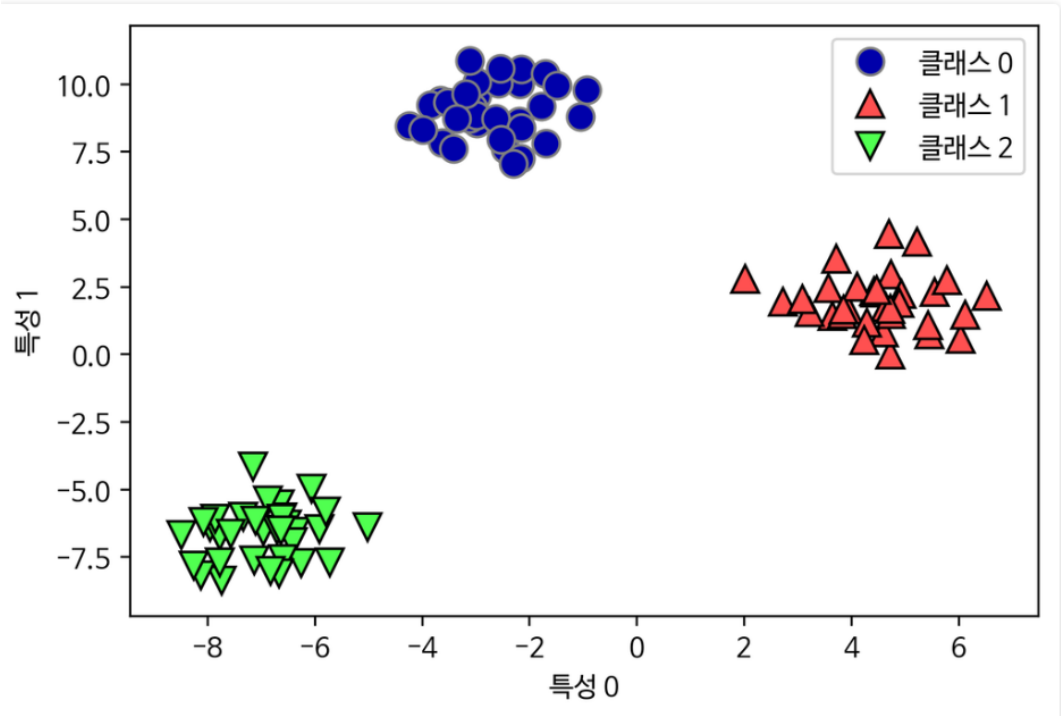
```
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(
cancer.data, cancer.target, stratify=cancer.target, random_state=42)

def LogisticReg(C=1.0,penalty='l2',color='b^',ax=None) :
        logreg = LogisticRegression(penalty=penalty,C=C).fit(X_train, y_train)
        score_train,score_test=logreg.score(X_train, y_train),logreg.score(X_test, y_test)
        ax.plot(logreg.coef_.T,color,
                label='{} C={:8.3f}  acc(train:{:.2f} test:{:.2f})'.format(
                        logreg.penalty, C,score_train,score_test))
        ax.hlines(0, 0, cancer.data.shape[1])
        ax.set_xticks(range(cancer.data.shape[1]))
        ax.set_xticklabels(cancer.feature_names,rotation=90,fontsize='small')
        ax.set_ylim(-5, 5)
        ax.set_xlabel("특성");     ax.set_ylabel("계수 크기")
        ax.grid();                 ax.legend(loc='best')

fig,axes=plt.subplots(2,1,sharex=True,sharey=True)

log=LogisticReg(C=100,color='b^',ax=axes[0])
log=LogisticReg(C=1,color='ro',ax=axes[0])
log=LogisticReg(C=0.001,color='gv',ax=axes[0])

log=LogisticReg(C=100,penalty='l1',color='b^', ax=axes[1])
log=LogisticReg(C=1,penalty='l1',color='ro', ax=axes[1])
log=LogisticReg(C=0.001,penalty='l1',color='gv', ax=axes[1])
plt.subplots_adjust(left=0.1, bottom=0.3, right=0.9, top=0.9, wspace=0, hspace=0)
plt.show()
```



4.6 다중 클래스용 선형모델

- 개요
  - 선형분류모델은 이진분류
  - 선형분류모델(이진분류기)를 확장하여 다중분류기
  - 예,
    - LinearSVC 분류기
  - 데이터 셋
    - 세 개의 클래스를 가진 2차원 데이터셋
      sklearn.datasets import make_blobs

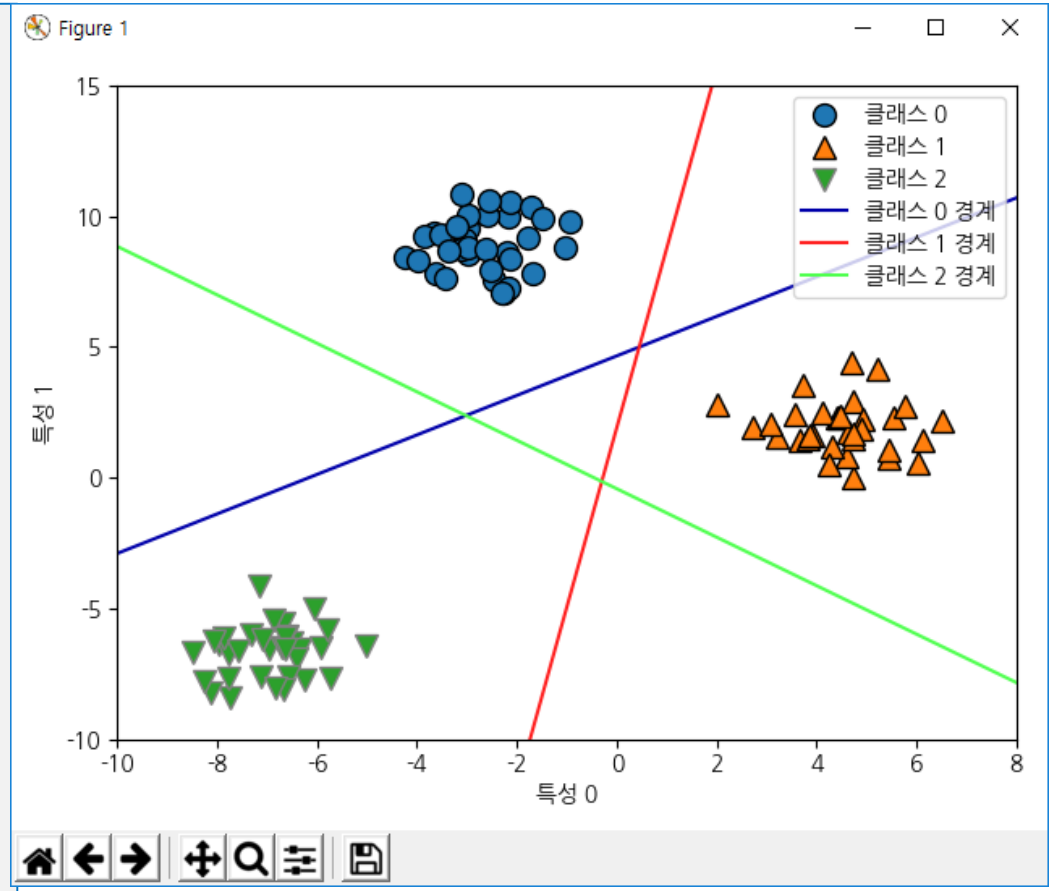
```
from sklearn.datasets import make_blobs
X, y = make_blobs(random_state=42)

mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
plt.xlabel("특성 0")
plt.ylabel("특성 1")
plt.legend(["클래스 0", "클래스 1", "클래스 2"])
plt.show()

linear_svm = LinearSVC().fit(X, y)

print("계수 배열의 크기: ", linear_svm.coef_.shape)      #(3,2)
print("절편 배열의 크기: ", linear_svm.intercept_.shape) #(3,)
```



4.6 다중 클래스용 선형모델(cont.)

- 이진분류기 LinearSVC 를 이용한 3 클래스의 분류

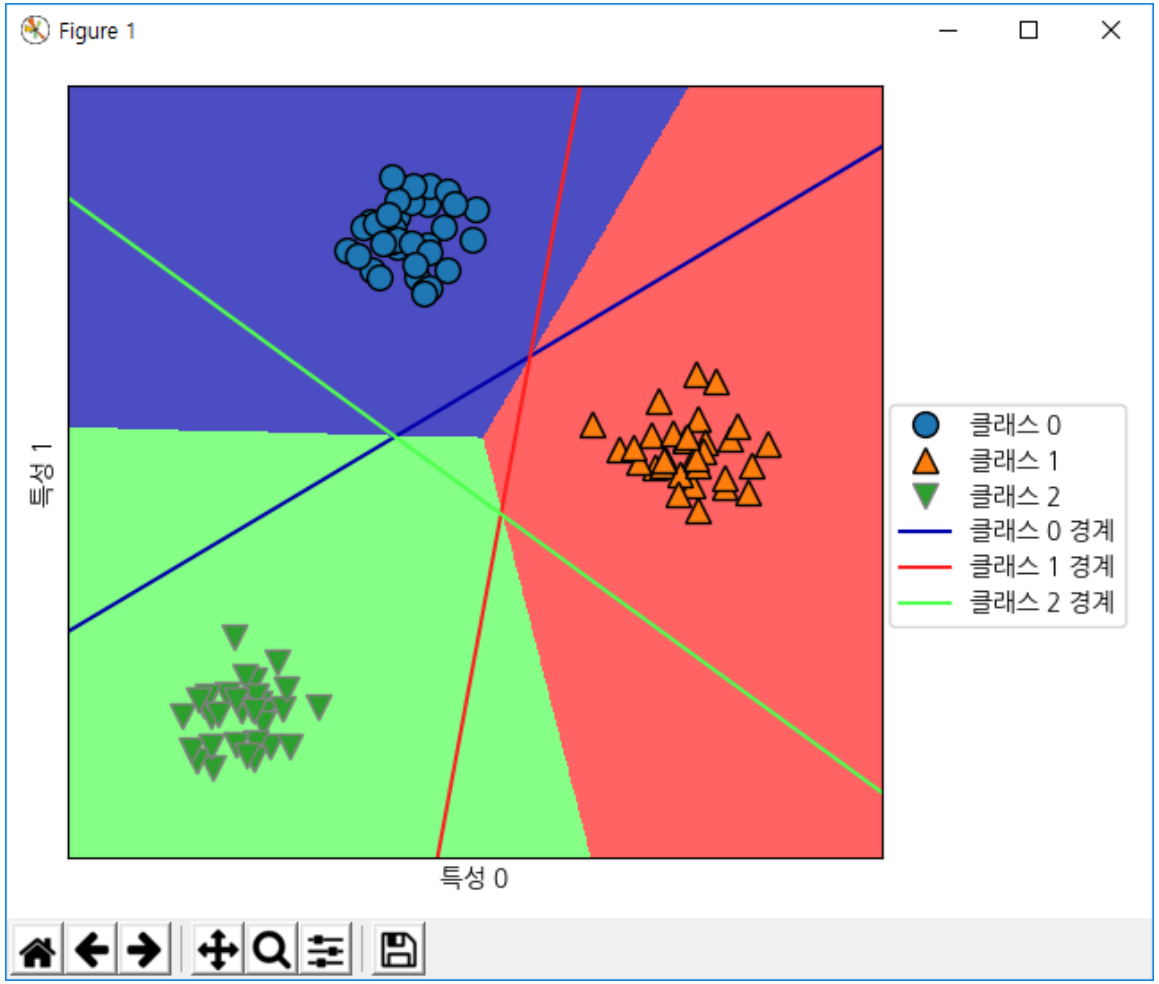
```
from sklearn.model_selection import train_test_split
from sklearn.svm import LinearSVC
import matplotlib.pyplot as plt
import numpy as np
import mglearn
from sklearn.datasets import make_blobs

X, y = make_blobs(random_state=42)

linear_svm = LinearSVC().fit(X, y)
print("계수 배열의 크기: ", linear_svm.coef_.shape)#(3,2)
print("절편 배열의 크기: ", linear_svm.intercept_.shape)#(3,)

mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
line = np.linspace(-15, 15)
for coef, intercept, color in zip(linear_svm.coef_, linear_svm.intercept_,
                                  mglearn.cm3.colors):
    plt.plot(line, -(line * coef[0] + intercept) / coef[1], c=color)
    plt.ylim(-10, 15)
    plt.xlim(-10, 8)
plt.xlabel("특성 0")
plt.ylabel("특성 1")
plt.legend(['클래스 0', '클래스 1', '클래스 2', '클래스 0 경계', '클래스 1 경계','클래스 2 경계'],
loc=(1.01, 0.3))
plt.tight_layout()
plt.show()
```



4.6 다중 클래스용 선형모델(cont.)

- 이진분류기 LinearSVC 를 이용한 3 클래스의 분류
  - 분류경계
  - 삼각형안의 점은 경계와 가장 가까운 선에 포함

```
mglearn.plots.plot_2d_classification(linear_svm, X, fill=True, alpha=.7)
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
line = np.linspace(-15, 15)
for coef, intercept, color in zip(linear_svm.coef_, linear_svm.intercept_,
                                  mglearn.cm3.colors):
    plt.plot(line, -(line * coef[0] + intercept) / coef[1], c=color)
plt.legend(['클래스 0', '클래스 1', '클래스 2', '클래스 0 경계', '클래스 1 경계',
            '클래스 2 경계'], loc=(1.01, 0.3))
plt.xlabel("특성 0")
plt.ylabel("특성 1")
plt.tight_layout()
plt.show()
```



- 요약
  - K-최근접 이웃
    - 분류 : KNeighbordClassifiers
      - knn = KNeighborsClassifier(n_neighbors=1).fit(X,y) #forge X:(26, 2)y:(26,) int32 [0, 1]
    - 회귀 :
      - knn_reg=KNeighborsRegressor(n_neighbors=3).fit(X,y) #wave X:(40, 1)y:(40,) float64]
  - 선형모델
    - 선형회귀(최소자승법)
      - lr=LinearRegression().fit(X_train,y_train) #wave X:(40, 1) y:(40,) float64
    - Ridge 회귀
      - ridge=Ridge(alpha=1.0).fit(X_train,y_train) #boston data:(506, 13) target:(506,) float64
    - Lasso 회귀
      - lasso=Lasso(alpha=1.0).fit(X_train,y_train) #boston data:(506, 13) target:(506,) float64
    - 분류용 선형 모델 (이진분류)
      - log_reg = LogisticRegression(penalty=penalty, C=C).fit(X_train, y_train)
        #forge X:(26, 2) y:(26,) int32 [0, 1]
    - 다중클래스 분류용 선형 모델
      - linear_svm =LinearSVC().fit(X, y) #X:(100, 2) y:(100,) int32 [0, 1, 2]

4.7 장단점과 매개변수

- 장단점과 매개변수
- 매개변수
  - 선형 모델의 주요 매개변수는 회귀 모델에서는 alpha였고 LinearSVC와 LogisticRegression에서는 C입니다.
  - alpha 값이 클수록, C 값이 작을수록 모델이 단순해집니다. 특별히 회귀 모델에서 이 매개변수를 조정하는 일이 매우 중요합니다. 보통 C와 alpha는 로그 스케일(0.01,0.1,1.0,10.0) 등으로 최적치를 정합니다.
  - 그리고 L1 규제를 사용할지 L2 규제를 사용할지를 정해야 합니다. 중요한 특성이 많지 않다고 생각하면 L1 규제를 사용합니다. 그렇지 않으면 기본적으로 L2 규제를 사용해야 합니다. L1 규제는 모델의 해석이 중요한 요소일 때도 사용할 수 있습니다. L1 규제는 몇 가지 특성만 사용하므로 해당 모델에 중요한 특성이 무엇이고 그 효과가 어느 정도인지 설명하기 쉽습니다.
- 데이셋 사이즈
  - 선형 모델은 학습 속도가 빠르고 예측도 빠릅니다. 매우 큰 데이터셋과 희소한 데이터셋에도 잘 작동합니다. 수십만에서 수백만 개의 샘플로 이뤄진 대용량 데이터셋이라면 기본 설정보다 빨리 처리하도록 LogisticRegression과 Ridge에 solver='sag' 옵션을 줍니다. 다른 대안으로는 여기서 설명한 선형 모델의 대용량 처리 버전으로 구현된 SGDClassifier와 SGDRegressor를 사용할 수 있습니다.
    - sgd:Stochastic Average Gradient descent(확률적 평균 경사 하강법)의 약자로서 경사 하강법과 비슷하지만, 반복이 진행될 때 이전에 구한 모든 경사의 평균을 사용하여 계수를 갱신합니다.
- 선형 모델의 또 하나의 장점은 앞서 회귀와 분류에서 본 공식을 사용해 예측이 어떻게 만들어지는지 비교적 쉽게 이해할 수 있다
  - 하지만 계수의 값들이 왜 그런지 명확하지 않을 때가 종종 있습니다. 특히 데이터셋의 특성들이 서로 깊게 연관되어 있을 때 그렇습니다. 그리고 이럴 땐 계수를 분석하기가 매우 어려울 수 있습니다.
- 선형 모델은 샘플에 비해 특성이 많을 때 잘 작동합니다.
  - 다른 모델로 학습하기 어려운 매우 큰 데이터셋에도 선형 모델을 많이 사용합니다. 그러나 저차원의 데이터셋에서는 다른 모델들의 일반화 성능이 더 좋습니다. "커널화 서포트 벡터 머신"에서 선형 모델이 실패할수도 있다.