Deep Learning

지도학습

선형모델
(선형회귀,선형분류)

Yoonjoong Kim

Department of Computer Engineering, Hanbat National University

yjkim@hanbat.ac.kr

content

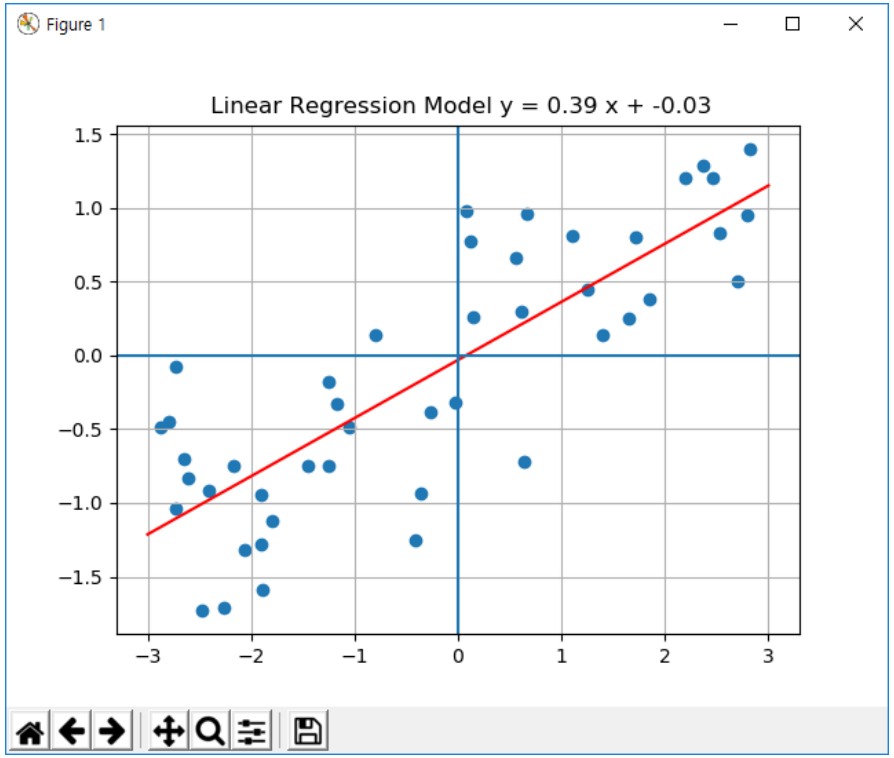
- 지도학습
  1. 지도학습 개요
     - 분류,회귀,일반화,과대적합,과소적합,복잡도
  2. 예제에서 사용할 데이터셋
     - forge, wave, breast_cancer, boston, extended_boston, blobs
  3. K-최근이웃
     1. K-최근접 이웃 분류 KNeighborsClassifiers
     2. K-최근접 이웃 회귀 KNeighborsRegressor
  4. 선형모델
     1. 선형회귀모델의 개념
     2. 선형회귀
     3. Ridge회귀
     4. Lasso 회귀
     5. 이진 선형분류
     6. 다중 선형분류
     7. 장단점과 매개변수

참고:파이썬 라이브러리를 활용한 머신러닝 (Introduction to Machine Learning by Python) /한빛미디어

1. 선형모델

1. 회귀의 선형모델의 개념

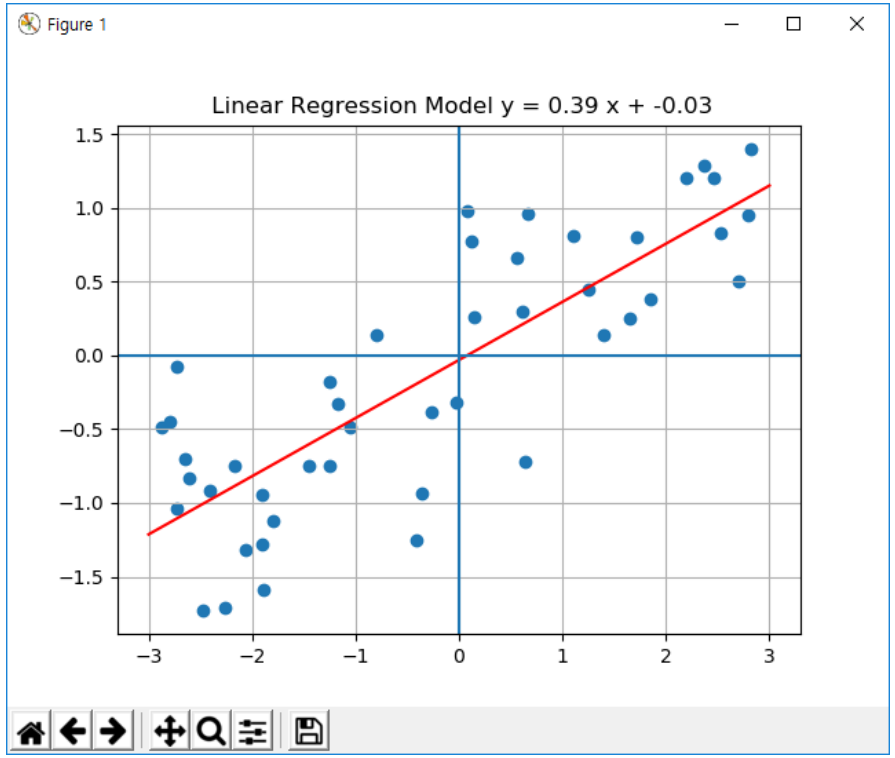
- 선형 회귀는
  - 선형모델을 정의하고 주어지는 데이터셋으로 학습하여 임의의 데이터에 대한 값을 예측하는 것이다.
- 선형회귀모델
  - 선형예측모델(함수)의 일반식
    - $\hat{y} = f(x_0, x_1, \ldots, x_p) = w_0 x_0 + w_1 x_1 + \cdots + w_p x_p + b$
    - $\hat{y}$: $\{x_i\}$에 대한 모델의 예측 값
    - $\{w_i\}, b$: 학습할 파라메터, 웨이트와 바이어스
    - $\{x_{ij}\}, \{y_i\}$: 학습용 데이터
  - 선형회귀모델의 종류
    - 특성이 하나이면 직선 모델
    - 2 이면 평면 모델
    - 그 이상이면 초평면(hyper plane)모델이 된다.
- 직선모델의 예,
  - $\hat{y} = wx + b$
- sklearn 패키지를 이용한 구현



2 선형회귀(LinearRegression)

2. 선형회귀(LinearRegression)

- 선형회귀모델
  - 선형모델 정의 예, $\hat{y} = f(x) = wx + b$
  - 손실함수(mean square error) $loss(w,b) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$
  - 학습 $w^* = argmin_{w,b}(loss(w,b)))$
  - 학습데이터 $\mathrm{X} = \{\mathrm{x_i}\}, Y = \{y_i\}$
  - 예측 $\hat{y} = f(x) = w^*x + b^*$
  - 평가지수 $R^2 = p(\{x_i\}) = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2}$
    - score, 적합도 0~1
    - $\hat{y}$: 모델 예측치 $\bar{y}$: target y의 평균
    - 모든 예측치 $\hat{y}$가 평균치 $\bar{y}$와 같으면 0
    - 모든 예측치 $\hat{y}$가 target치 y와 같으면 1 완벽한 모델



2 선형회귀(cont.)

2.1 선형회귀(LinearRegression) -직선모델

- 1차원 데이터셋(59,1) 직선모델 =>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
```

```
from sklearn.linear_model  import LinearRegression
import mglearn
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import pandas as pd
```

```
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
```

2 선형회귀(cont.)

2.2 선형회귀(LinearRegression) - 초평면모델

- 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)
```

- 단순모델이 필요
  =>복잡도를 제어할 수 있어야

```
def LR(X,y) :
    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)
    print('w : ',lr.coef_)
    print('b : ',lr.intercept_)
    print('훈련세트 성능(점수) : {:.2f}'.format(lr.score(X_train,y_train)))
    print('평가세트 성능(점수) : {:.2f}'.format(lr.score(X_test,y_test)))
```

```
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  6.03611391e+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 -4.11743268e+01 -2.19312422e+01 -2.08881337e+01
 -5.05858326e+01 -2.14714962e+01 -1.11593182e+01 -6.16458839e-01
 -1.12589338e+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 -2.02885460e+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
```

3 릿지회귀

3. 릿지회귀(RidgeRegression)

- 개요
  - 손실함수에 $L2$ 규제를 추가한 선형회귀모델
  - α를 이용하여 계수(w)가 작아지도록 제어한다.
  - 과대적합을 방지하기 위한 목적이다.
- 선형회귀모델
  - 선형모델 $\hat{y} = f(x) = wx + b$
  - 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. =>과적합을 방지하는 효과가 있다.
  - 학습 $w^* = argmin_{w,b}(loss(w,b))$
  - 학습데이터 $X = \{x_i\}, Y = \{y_i\}$
  - 예측 $\hat{y} = f(x) = w^*x + b^*$
  - 평가지수 $R^2 = 1 - \frac{\sum(y-\hat{y})^2}{\sum(y-\bar{y})^2}$

Alpha에 따른 Ridge 회귀모델의 성능 분석

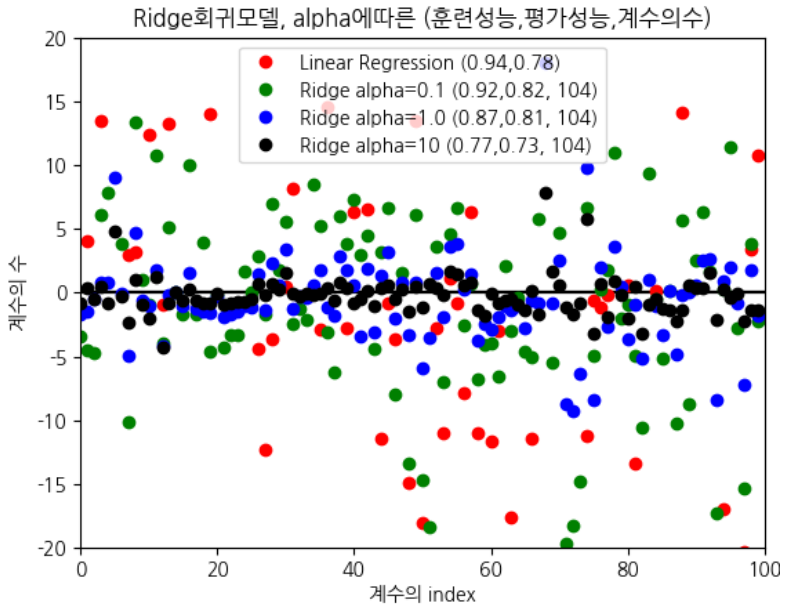
```
#X,y=mglearn.datasets.load_extended_boston()
print('Ridge Regression : ')
ridge=Ridge(alpha=1.0).fit(X_train,y_train) #lr 모델생성 및 손실함수 mse로 학습

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 Lasso 회귀

- 개요
  - 손실함수에 L1규제 선형회귀모델
  - 릿지 회귀와 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.

- Lasso 회귀모델
  - 선형모델 $\hat{y} = f(x) = wx + b$
  - 손실함수(mean square error)

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

    - L1 = $\alpha\sum_i |w_i|$
    - W를 작게 만들기 위함, 일부 w는 0이 되어 w의 수가 줄어드는 효과를 본다. => 복잡도 가 작아진다.
  - 학습 $w^* = argmin_{w,b}(loss(w,b)))$
  - 학습데이터 $\mathrm{X} = \{\mathrm{x_i}\}, Y = \{y_i\}$
  - 예측 $\hat{y} = f(x) = w^*x + b^*$
  - 평가지수 $R^2 = 1 - \frac{\sum(y-\hat{y})^2}{\sum(y-\bar{y})^2}$

4 Lasso 회귀(cont.)

- 라쏘 회귀의 알파 값에 따른 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
```

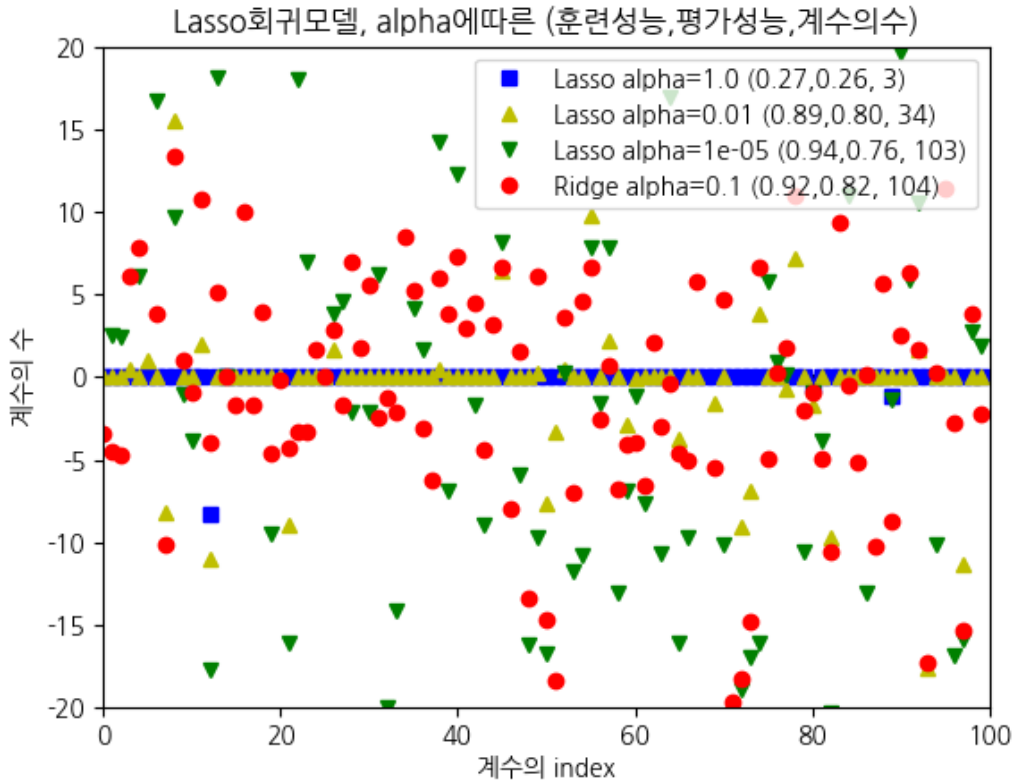
```
#lapha=0.01  Lasso 회귀
# "max_iter" 기본값을 증가시키지 않으면 max_iter 값을 늘리라는 경고가
발생합니다.

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    : 0.89  #과대적합
#테스트 세트 점수: 0.80  #
#사용한 특성의 수: 33     #증가했다.

#alpha=0.01 => 규제가 적절
```

- alpha 값을 낮추면 모델의 복잡도는 증가하여 훈련 세트와 테스트 세트에서의 성능이 좋아집니다. Rasso성능(0.01,0.89,0.80)은 Ridge(0.1, 0.92,0.82)과 비슷한데 사용된 특성은 104개 중 33개 뿐이어서, 아마도 모델을 분석하기가 조금 더 쉽습니다.
- alpha 값을 너무 낮추면 규제의 효과가 없어져 과대적합이 되므로 LinearRegression의 결과와 비슷해집니다.

4 Lasso 회귀(cont.)

- 라쏘 회귀의 알파 값에 따른 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

5 분류용 선형모델

- 개요
  - 선형회귀 모델을 이용하여 분류하기
- 이진분류(binary classification) 선형분류모델
  - 모델정의 $\hat{y} = f(x)$:{+1,-1} ?
    - $\hat{y}_l = f_l(x) = wx + b,$

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

- 이진선형분류의 2 모델
  - LogisticRegression 모델
    - linear_model.LogisticRegression(penalty=l2,C=1.0)
  - LinearSVC 모델
    - Svm.LinearSVC(penalty=l2,C=1.0)
    - $l2 = C\sum_i w_i^2$ SVC(support vector classifier)

- 선형회귀모델
  - 선형모델 $\hat{y} = f(x) = wx + b$
  - 손실함수(mean square error)

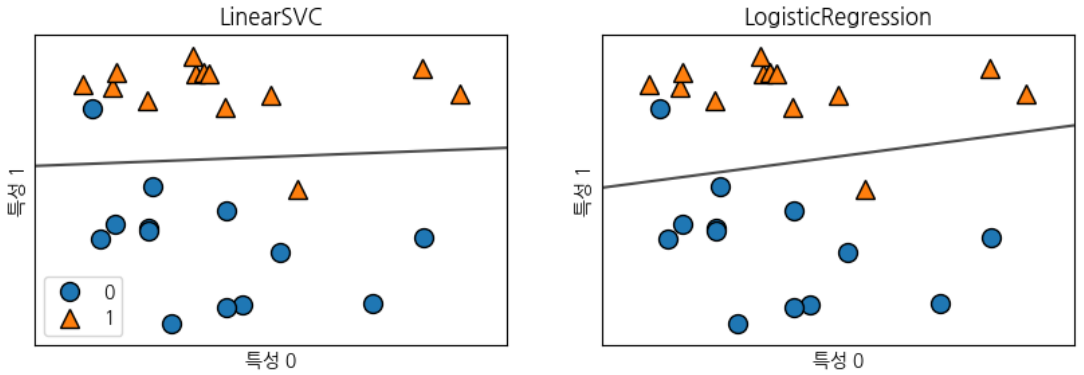
$$loss(w,b) = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - \hat{y}_i(w,b)\right)^2$$

  - 학습 $w^* = argmin_{w,b}(loss(w,b)))$
    - 학습데이터 $X = \{x_i\}, Y = \{y_i\}$
  - 예측 $\hat{y} = f(x) = w^*x + b^*$
  - 평가지수 $R^2 = 1 - \frac{\sum(y-\hat{y})^2}{\sum(y-\bar{y})^2}$

5 분류용 선형모델(cont.)

- 이진 분류 선형모델 LinearSVC과 LogisticRegression 의 비교 분석
  - 손실함수에 기본값 l2규제, C=1.0
    - penalty $l2 = C\sum_i w_i^2$

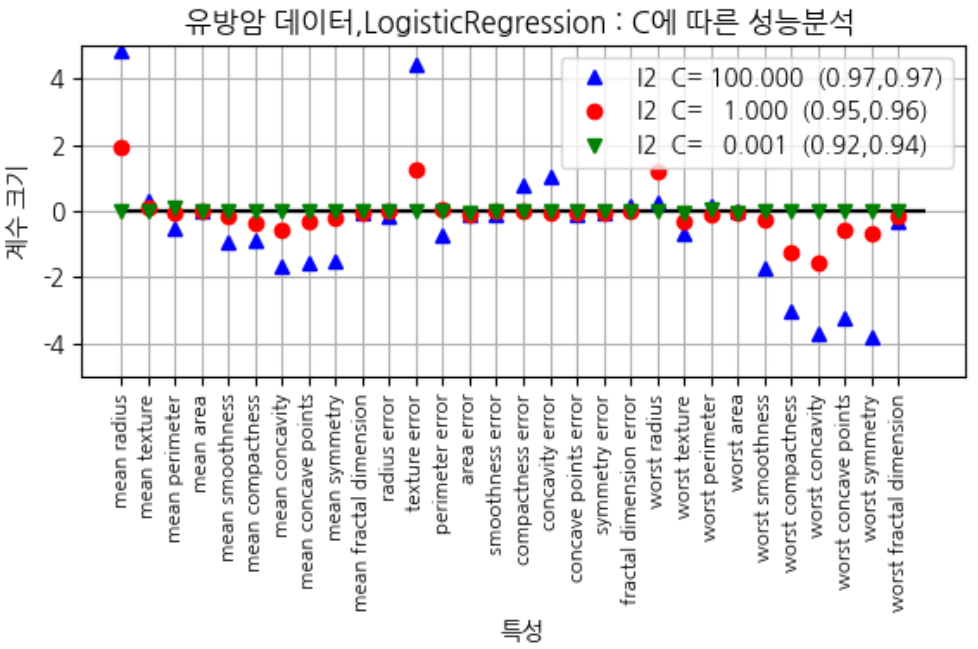
```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
import mglearn
import matplotlib.pyplot as plt

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")

axes[0].legend()
plt.show()
```



5 분류용 선형모델(cont.)

- 이진 분류 선형모델 LogisticRegression 분류모델
  - L2규제의 C값에 따른 훈련성능,평가성능, 유방암 데이터 셋

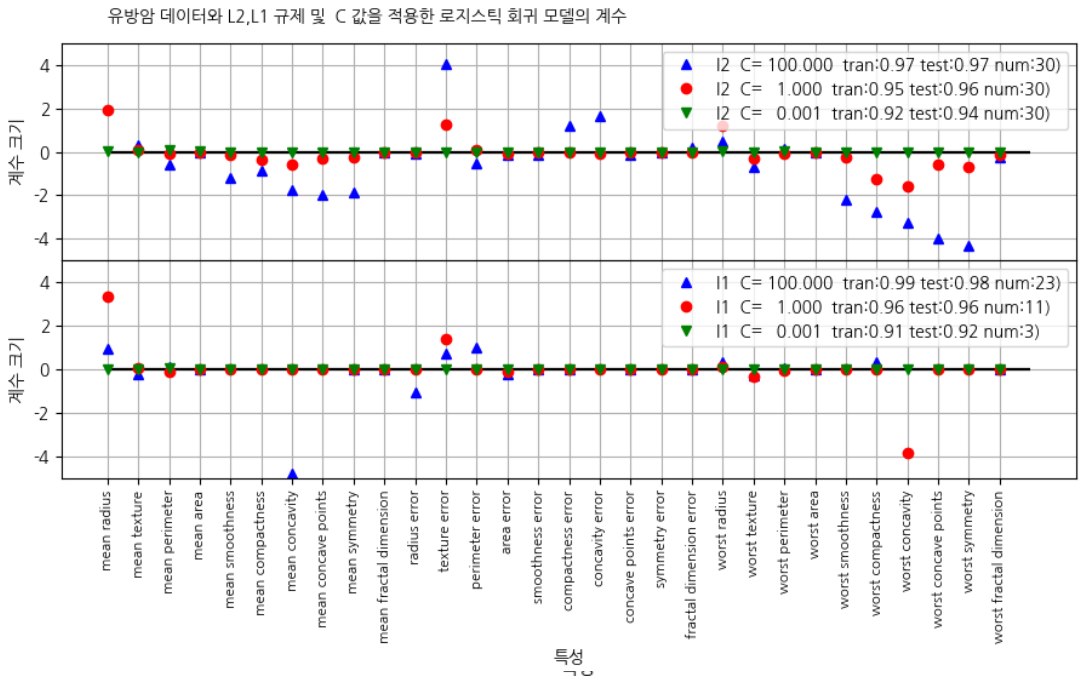
```
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(
cancer.data, cancer.target, stratify=cancer.target, random_state=42)

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()
```



5 분류용 선형모델(cont.)

- 이진 분류 선형모델 LogisticRegression 분류모델
  - L2, L1, C 값에 따른 모델의 계수 및 정확도, 유방암 데이터

```
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}  train:{:.2f} test:{:.2f} num:{})'.format(
                        logreg.penalty, C,score_train,score_test,np.sum(logreg.coef_ != 0)))
        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()
```

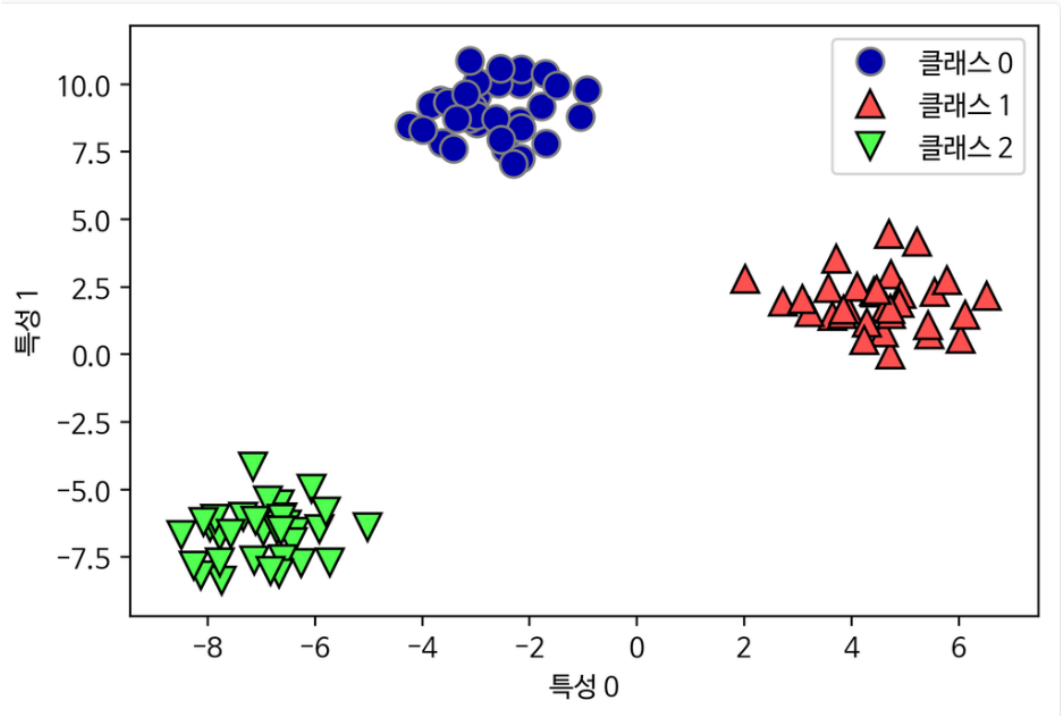


6. 다중 클래스용 분류 선형모델

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

sklearn.datasets import make_blobs

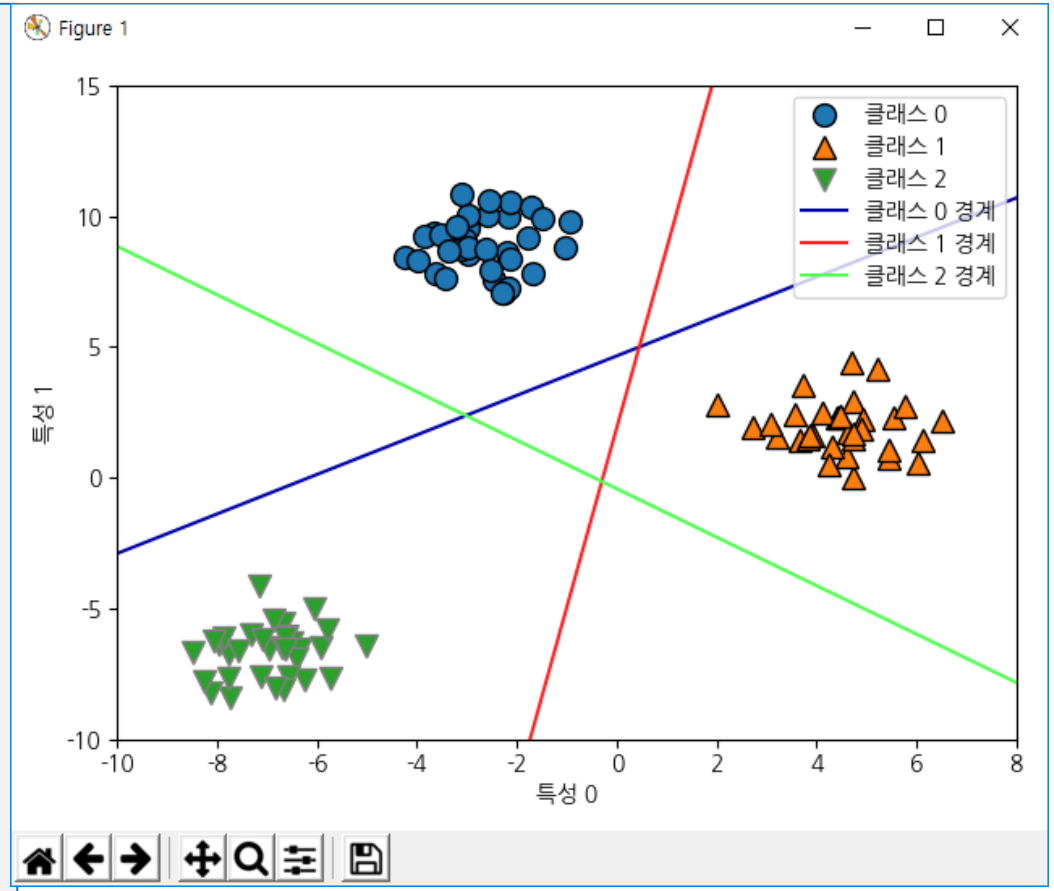
```
from sklearn.datasets import make_blobs
X, y = make_blobs(random_state=42)

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,)
```



6 다중 클래스용 선형모델(cont.)

- 이진분류기 LinearSVC 를 이용한 3 클래스의 분류

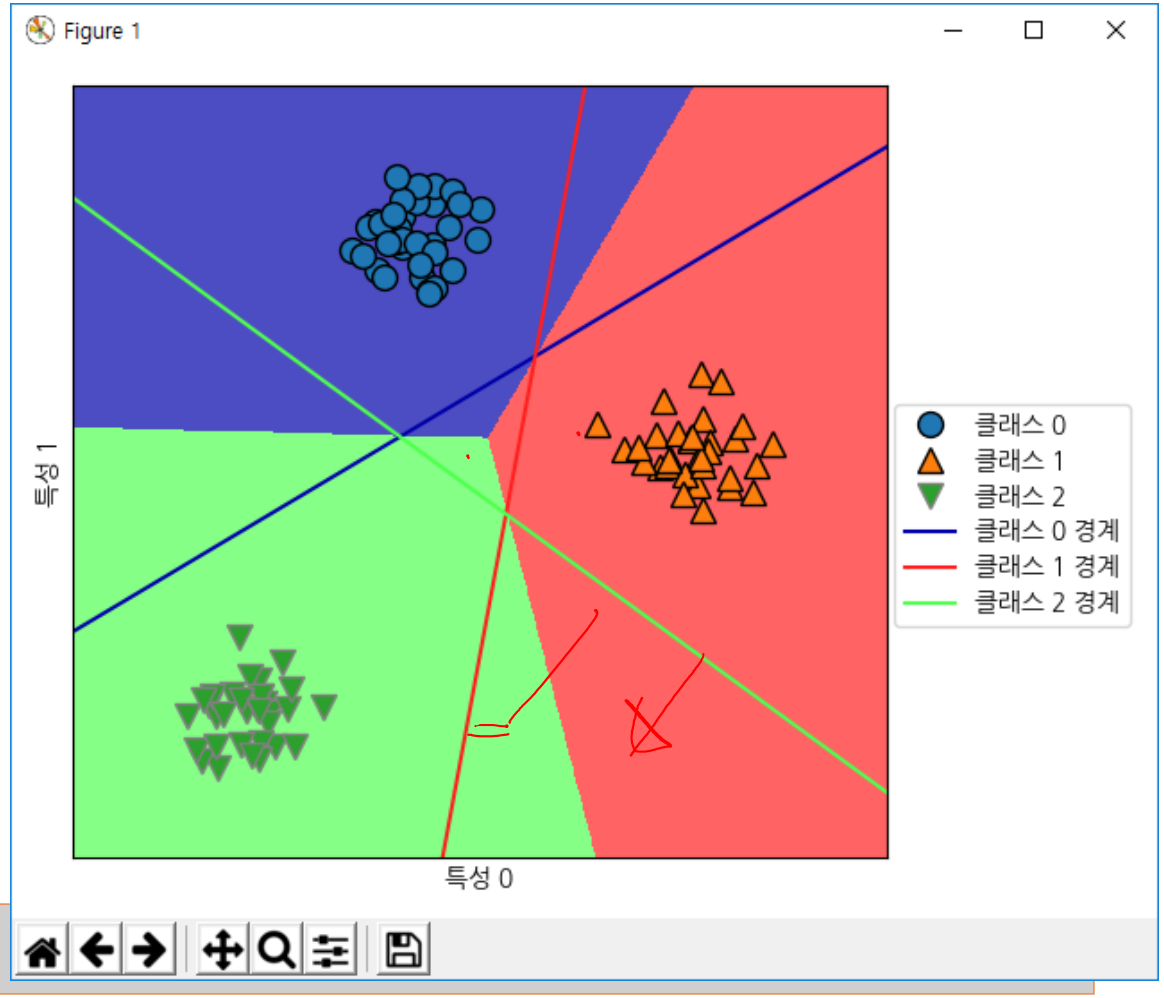
```
from sklearn.model_selection import train_test_split
from sklearn.svm import LinearSVC
import matplotlib.pyplot as plt
import numpy as np
import mglearn
from sklearn.datasets import make_blobs

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()
```



6 다중 클래스용 선형모델(cont.)

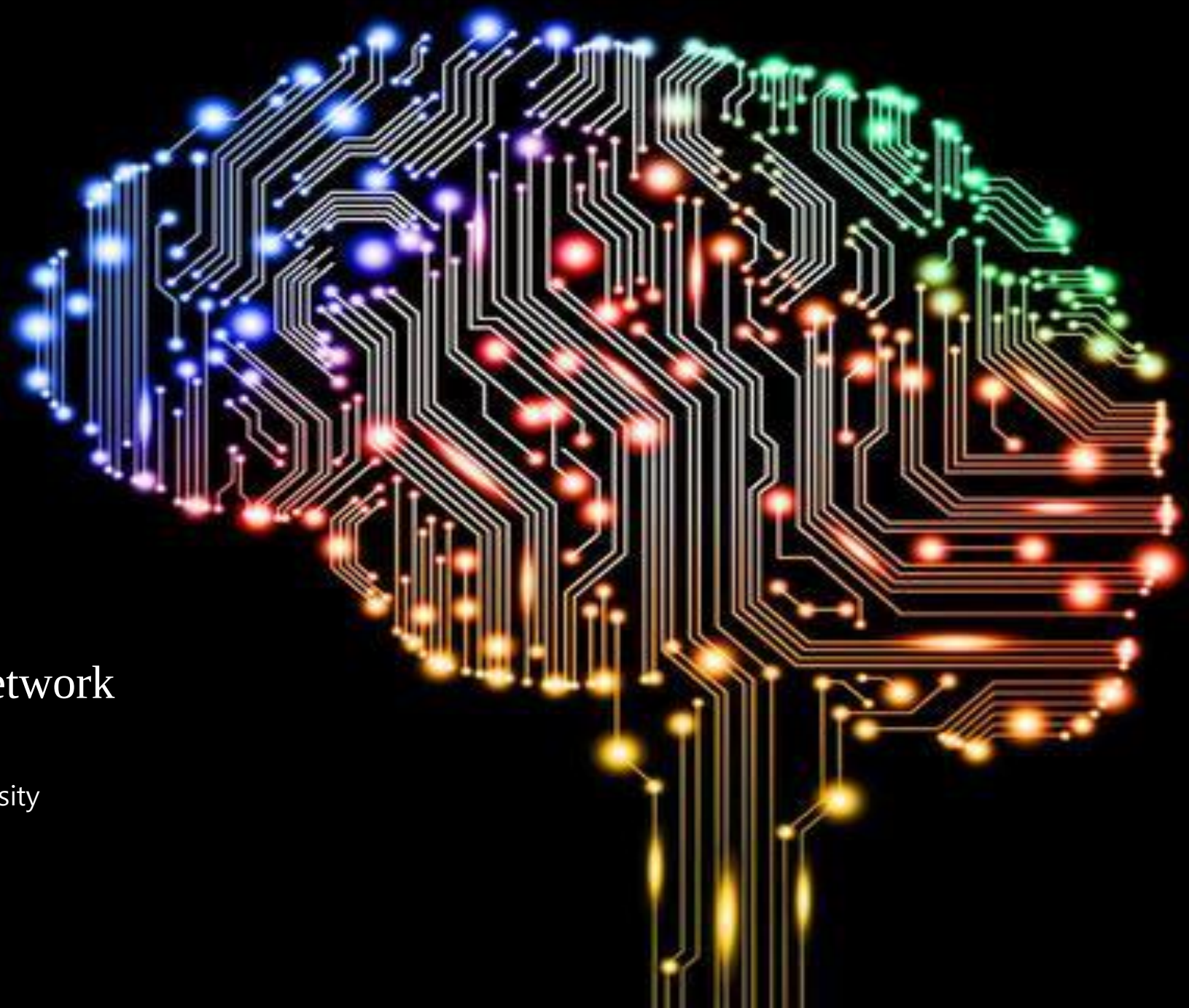
- 이진분류기 LinearSVC 를 이용한 3 클래스의 분류
  - 분류경계
  - 삼각형안의 점은 경계와 가장 가까운 선에 포함

```
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

- 지도학습 요약
  - 지도학습 개요
    - 분류,회귀,일반화,과대적합,과소적합,복잡도
  - K-최근접 이웃
    - 분류 : KNeighborsClassifiers
      - 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) #blobs X:(100, 2) y:(100,) int32 [0, 1, 2]



Deep Learning
Deep Neural Network

Yoon Joong Kim,
Hanbat National University