빅데이터분석 실습

6주 지도학습 알고리즘 3

데이터 사이언스 전공
담당교수: 곽철완

지난주 내용

- k- 최근접 이웃 회귀분석
- 선형 회귀모델
- 리지 회귀

강의 내용

- 라소 회귀
- 분류용 선형 모델
- 나이브 베이즈 분류기

▪ 기본 라이브러리

◦ 분석을 위해 기본 라이브러리를 import한다

```
%matplotlib inline
from IPython.display import display
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import mglearn
import sklearn
```

1. 라소

■ 목적

- 선형 회귀 규제화에서 리지 대안으로 사용
- 라소 계수를 0으로 만들어 모델의 이해를 쉽게 하고, 모델의 가장 중요한 피처를 드러나게 한다- L1 규제
  - 모델에서 완전히 제외되는 피처가 나타난다
- 결과적으로 피처 선택(feature selection)이 자동으로 이루어 진다

■ 사용 데이터 세트

◦ Boston housing 확장형 사용(피처 수: 104개)

```
from sklearn.datasets import load_boston
boston = load_boston
X, y = mglearn.datasets.load_extended_boston( )
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
```

◦ Lasso 함수 import

```
from sklearn.linear_model import Lasso
```

- 확장된 Boston Housing 데이터 세트에 Lasso 함수 적용하여
- 학습용 데이터 세트와 평가용 데이터 세트의 점수 확인
  - lasso 회귀계수 중 0 이 아닌 계수가 몇 가지인가?

```
lasso = Lasso( ).fit(X_train, y_train)
print("학습용 데이터 세트 점수: {:.2f}".format(lasso.score(X_train, y_train)))
print("평가용 데이터 세트 점수: {:.2f}".format(lasso.score(X_test, y_test)))
print("사용한 피처의 개수: ", np.sum(lasso.coef_ !=0))
```

```
# Lasso import
from sklearn.linear_model import Lasso

lasso = Lasso().fit(X_train, y_train)
print("학습용 데이터 세트 점수: {:.2f}".format(lasso.score(X_train, y_train)))
print("평가용 데이터 세트 점수: {:.2f}".format(lasso.score(X_test, y_test)))
print("사용한 피처의 개수: ", np.sum(lasso.coef_ !=0))
```

```
학습용 데이터 세트 점수: 0.29
평가용 데이터 세트 점수: 0.21
사용한 피처의 개수:  4
```

- Lasso를 이용한 학습용 데이터 세트와 평가용 데이터 세트 점수가 0.29와 0.21이다(리지는 0.89와 0.75)
  - 또한, 사용한 피처의 개수는 4개이다
  - alpha = 1.0 이 기본인데, 과소적합을 줄이기 위해 alpha 값을 줄여 적용한다(이때, 반복 실행횟수 기본값을 늘려야 한다)

```
lasso_001 = Lasso(alpha=0.01, max_iter=100000 ).fit(X_train, y_train)
print("학습용 데이터 세트 점수: {:.2f}".format(lasso_001.score(X_train, y_train)))
print("평가용 데이터 세트 점수: {:.2f}".format(lasso_001.score(X_test, y_test)))
print("사용한 피처의  개수:", np.sum(lasso_001.coef_ !=0))
```

- max_iter 값을 사용하지 않으면 적용하라는 경고가 나타난다

- alpha 값을 낮추어 성능 향상을 가져왔다
  - 사용한 피처는 33개로 증가했다

```
lasso_001 = Lasso(alpha=0.01, max_iter=100000).fit(X_train, y_train)
print("학습용 데이터 세트 점수: {:.2f}".format(lasso_001.score(X_train, y_train)))
print("평가용 데이터 세트 점수: {:.2f}". format(lasso_001.score(X_test, y_test)))
print("사용한 피처의 개수:",np.sum(lasso_001.coef_ != 0))
```

```
학습용 데이터 세트 점수: 0.90
평가용 데이터 세트 점수: 0.77
사용한 피처의 개수: 33
```

◦ alpha 값을 0.0001로 낮춘 결과는?

```
lasso_00001 = Lasso(alpha=0.0001, max_iter=100000 ).fit(X_train, y_train)
print("학습용 데이터 세트 점수: {:.2f}".format(lasso_00001.score(X_train, y_train)))
print("평가용 데이터 세트 점수: {:.2f}".format(lasso_00001.score(X_test, y_test)))
print("사용한 피처의  개수:", np.sum(lasso_00001.coef_ !=0))
```

◦ 학습용 점수는 증가
◦ 하지만, 평가용 점수 감소
◦ 과대적합이 되었다

```
lasso_00001 = Lasso(alpha=0.0001, max_iter=100000).fit(X_train, y_train)
print("학습용 데이터 세트 점수: {:.2f}".format(lasso_00001.score(X_train, y_train)))
print("평가용 데이터 세트 점수: {:.2f}". format(lasso_00001.score(X_test, y_test)))
print("사용한 피처의 개수:",np.sum(lasso_00001.coef_ != 0))
```

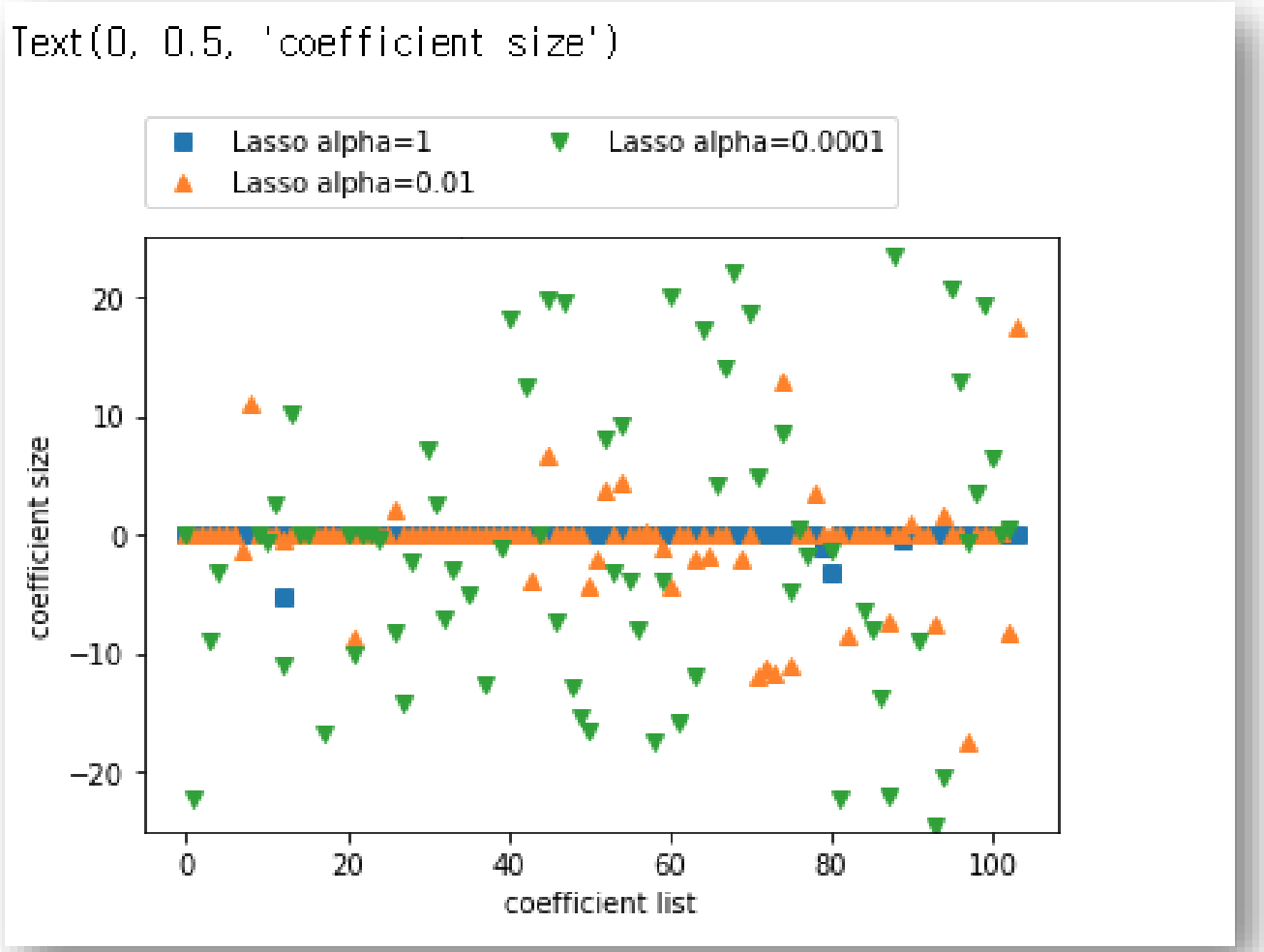
```
학습용 데이터 세트 점수: 0.95
평가용 데이터 세트 점수: 0.64
사용한 피처의 개수: 96
```

◦ alpha 값이 다른 경우 계수의 크기

```
plt.plot(lasso.coef_, ‘s’, label = “Lasso alpha=1”)
plt.plot(lasso_001.coef_, ‘^’, label = “Lasso alpha=0.01”)
plt.plot(lasso_00001.coef_, ‘v’, label = “Lasso alpha=0.0001”)

plt.legend(ncol=2, loc=(0, 1.05))
plt.ylim(-25, 25)
plt.xlabel(“coefficient list”)
plt.ylabel(“coefficient size”)
```

```
plt.plot(lasso.coef_, 's', label = "Lasso alpha=1")
plt.plot(lasso_001.coef_, '^', label = "Lasso alpha=0.01")
plt.plot(lasso_00001.coef_, 'v', label = "Lasso alpha=0.0001")

plt.legend(ncol=2, loc=(0, 1.05))
plt.ylim(-25, 25)
plt.xlabel("coefficient list")
plt.ylabel("coefficient size")
```

- alpha=1 일 때 계수는 0에 가까움
- alpha=0.0001 일 때 계수 대부분이 0이 아니고 값이 커져서 규제를 받지 않는 모델이 됨
- 일반적으로 리지 회귀 모델을 선호하지만, 많은 피처 중에서 일부 피처를 선호한다면, Lasso 모델이 좋은 선택일 수 있다



2. 분류용 선형 모델

- 이진 분류

$$\hat{y} = w[0] \times x[0] + w[1] \times x[1] + \ldots + w[p] \times x[p] + b > 0$$

  - 선형 회귀식과 비슷하지만, 피처들의 가중치 합을 그냥 사용하는 대신, 예측한 값을 0과 비교하여 0보다 작으면 클래스를 -1로 예측하고, 0보다 크면 +1로 예측한다

- 주요 종류
  - 로지스틱 회귀 Losgistic Regression
  - 서포트 벡터 머신 Support Vector Machine

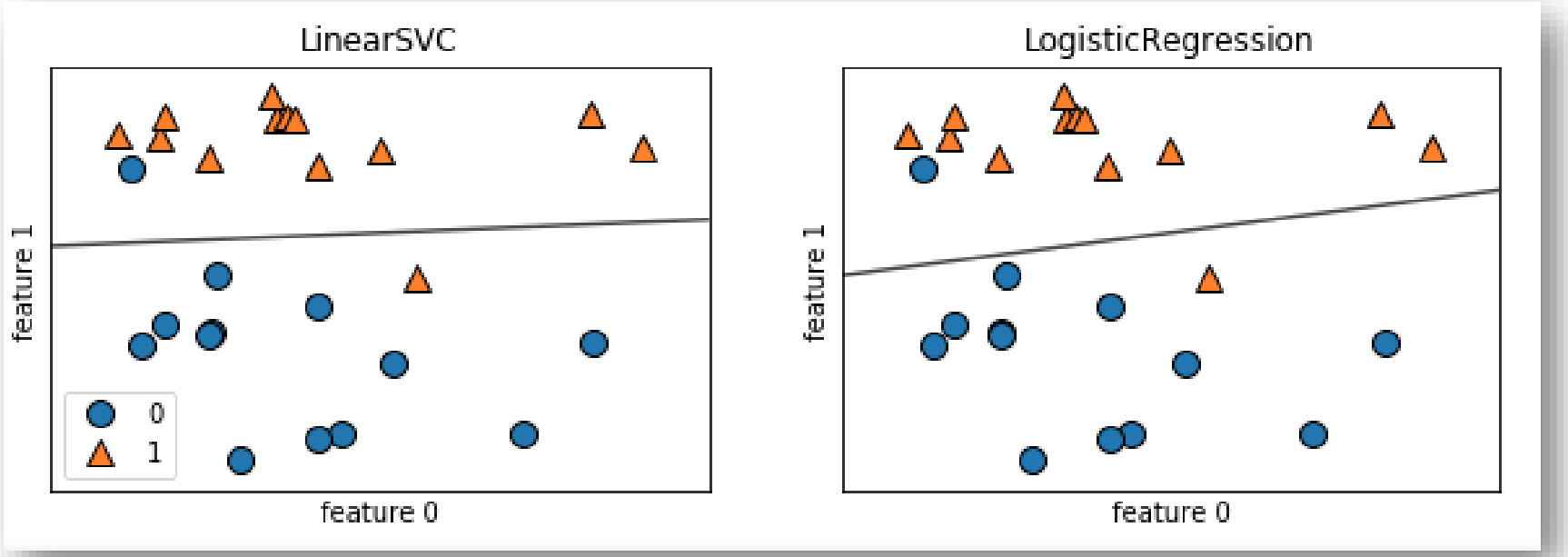
■ 로지스틱 회귀, 서포트 벡터 머신

- forge 데이터 세트는 인위적으로 만든 이진 분류 데이터 세트이다
- 로지스틱 회귀와 서포트 벡터 머신(SVM)이 만든 결정 경계를 그림으로 표시

```
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(clf.__class__.__name__)
    ax.set_xlabel("feature 0")
    ax.set_ylabel("feature 1")
axes[0].legend( )
```

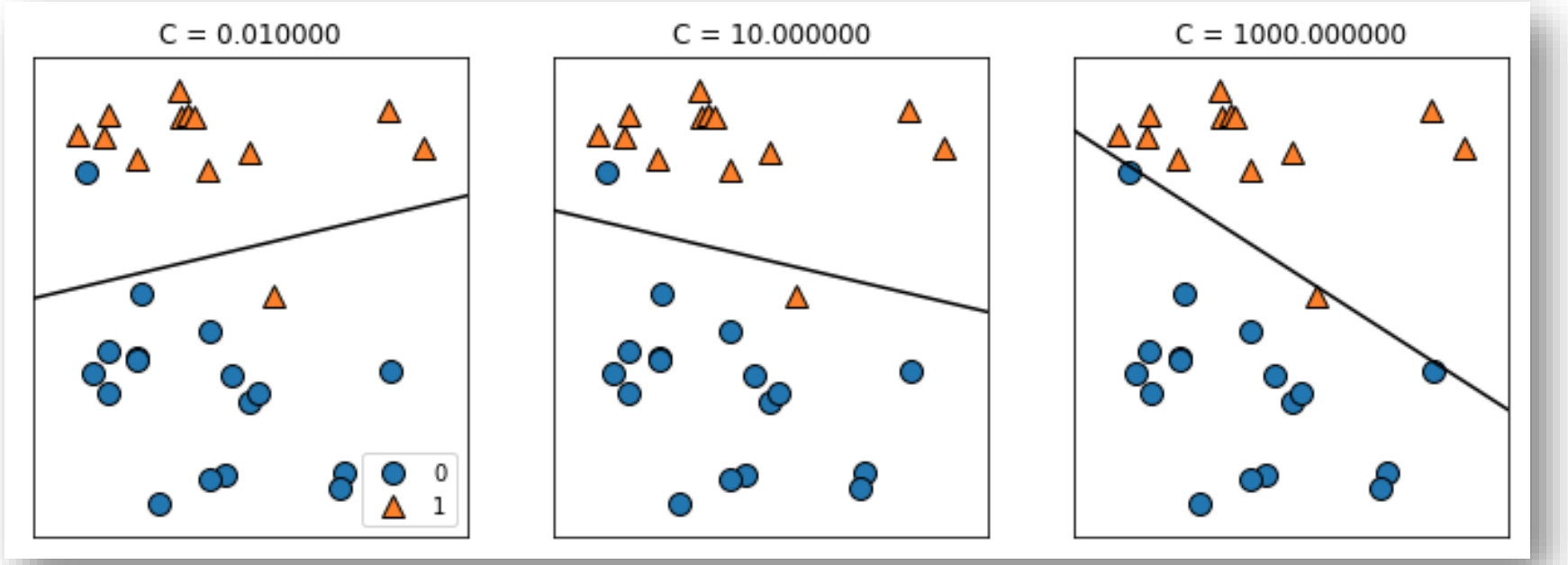
```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.datasets import make_blobs

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(clf.__class__.__name__)
    ax.set_xlabel("feature 0")
    ax.set_ylabel("feature 1")
axes[0].legend()
```



- 위의 두 알고리즘은 기본적으로 L2 규제를 사용한다(Ridge와 동일, 과대적합 감소)
- SVC와 로지스틱 회귀는 규제를 위한 매개변수 C가 있다
- 매개변수 C의 값이 높아지면 규제가 감소한다(학습용 데이터 세트와 유사)
- C의 값을 낮추면 계수 벡터(w)가 0에 가까워진다(피처가 영향을 못 미친다)

◦ C 값에 따른 결정 경계 차이 확인하기

```
mglearn.plots.plot_linear_svc_regularization( )
```



◦ 왼쪽은 규제가 많이 적용, 오른쪽 모델은 과대적합

3. 다중 클래스 분류용 선형 모델

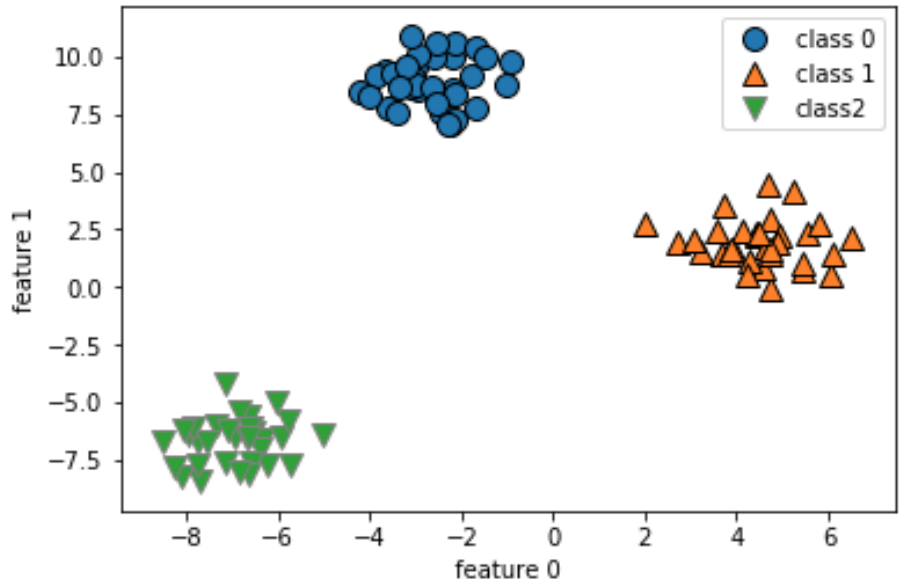
▪ LinearSVC

- 사용 데이터 세트: make_blobs( ): 클러스터링용 가상 데이터 생성 함수
  - 3개 class 그림 그리기

```
from sklearn.datasets import make_blobs

X, y = make_blobs(random_state=42)
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
plt.xlabel(“feature 0”)
plt.ylabel(“feature 1”)
plt.legend([“class 0”, “class 1”, “class 2”])
```

```
from sklearn.datasets import make_blobs

X, y = make_blobs(random_state=42)
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
plt.xlabel("feature 0")
plt.ylabel("feature 1")
plt.legend(["class 0", "class 1", "class2"])
```

```
<matplotlib.legend.Legend at 0x2013c5cbc50>
```



- LinearSVC 분류기를 이용하여 데이터 세트 학습

```
Linear_svm = LinearSVC( ).fit(X, y)
print("회귀 계수 배열의 크기: ", linear_svm.coef_.shape)
print("절편 배열의 크기: ", linear_svm.intercept_.shape)
```

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

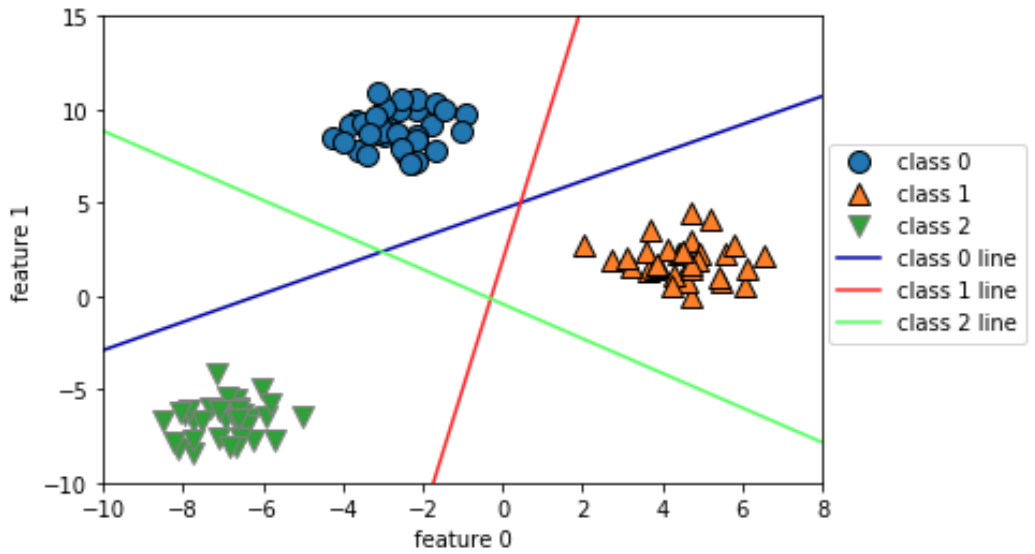
```
회귀 계수 배열의 크기:  (3, 2)
절편 배열의 크기:  (3,)
```

  - 회귀 계수 배열의 크기-> 행: 3가지 클래스, 열: 2가지 피처
  - 절편은 각 클래스의 1차원 벡터

◦ 3개의 이진 분류기가 만든 경계 시각화

```
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("feature 0")
plt.ylabel("feature 1")
plt.legend(['class 0', 'class 1', 'class 2', 'class 0 line', 'class 1 line',
            'class 2 line'], loc=(1.01, 0.3))
```

```
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("feature 0")
plt.ylabel("feature 1")
plt.legend(['class 0', 'class 1', 'class 2', 'class 0 line', 'class 1 line',
        'class 2 line'], loc=(1.01, 0.3))
```

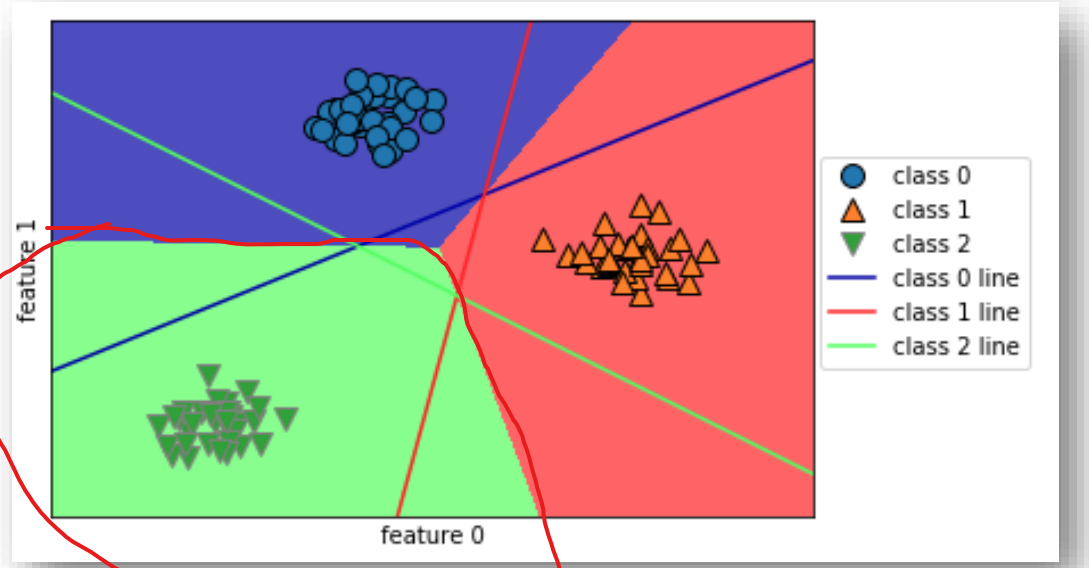


- 클래스 0 은 파랑색 라인, 클래스 1은 주황색 라인, 클래스 2는 녹색 라인으로 구분되었다
- 새로운 데이터가 라인 안쪽에 위치하면 그 클래스에 분류한다
- 단, 가운데 삼각형 부분은 가장 가까운 라인에 분류한다

◦ 평면 분류 예측 결과

```
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.xlabel("feature 0")
plt.ylabel("feature 1")
plt.legend(['class 0', 'class 1', 'class 2', 'class 0 line', 'class 1 line',
            'class 2 line'], loc=(1.01, 0.3))
```

```
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.xlabel("feature 0")
plt.ylabel("feature 1")
plt.legend(['class 0', 'class 1', 'class 2', 'class 0 line', 'class 1 line',
            'class 2 line'], loc=(1.01, 0.3))
```



- 위의 그림은 앞의 그림과 같지만, 2차원 분류 그림을 이용한 결과이다

- 장단점과 매개변수
  - 매개변수
    - 회귀 모델: alpha → 값이 클수록 모델 단순
    - LinearsSVC와 LogisticRegression: C → 값이 작을 수록 모델 단순
  - L1과 L2
    - L1 규제는 중요한 피처가 많지 않을 경우 사용
    - L2 규제는 기본적으로 사용(과대적합을 줄임)

3. 나이브 베이즈 분류기

▪ 특징

- ◦ 머신 러닝 알고리즘으로 주로 분류(classification)의 목적으로 사용
- ◦ 피처들 사이의 독립을 가정하는 베이즈 정리(Bayes theorem)를 이용

▪ 베이즈 정리

- ◦ 두 확률 변수의 사전 확률과 사후 확률 사이의 관계를 나타내는 정리: 사전 확률의 정보를 이용하여 사후 확률을 추정

- ◦ 사전 확률(prior probability): 가지고 있는 정보를 기초로 정한 초기 확률
- ◦ 사후 확률(posterior probability): 결과가 발생했다는 조건에서 어떤 원인이 발생했을 확률

- 우도(likelihood): 원인이 발생했다는 조건에서 결과가 발생했을 확률
- 조건부 확률

$$P(B|A) = \frac{P(A \cap B)}{P(A)}$$

A: 사건의 원인, B: 결과
P(A): 원인이 발생할 사전 확률
P(B): 결과가 발생할 사전 확률

- 확률의 곱셈정리

$$P(A \cap B) = P(A)P(B|A) = P(B)P(A|B)$$

- 조건부 확률식의 확률 곱셈정리로 치환

$$P(B|A) = \frac{P(A)P(B|A)}{P(A)} = \frac{P(B)P(A|B)}{P(A)}$$

  - P(B|A): 원인이 발생했을 때 결과가 발생할 확률
  - P(A|B) 결과가 발생했을 때 원인이 발생할 확률(사후확률)

▪ 데이터 세트

```
from sklearn.naive_bayes import GaussianNB

tennis_data = pd.read_csv('e:\workspace\playtennis.csv')
tennis_data
```

- 나이브 베이즈 분류를 위해 sklearn.naïve_bayes 서브 패키지에서 GaussianNB 모듈을 불러온다
- 데이터 세트는 pd.read_csv( ) 함수를 이용하여 데이터 세트가 저장된 디렉토리에서 데이터 프레임 형태로 저장한다

```
from sklearn.naive_bayes import GaussianNB
```

```
tennis_data = pd.read_csv('e:\workspace\playtennis.csv')
tennis_data
```

- 데이터 전처리가 필요하다
- 문자를 숫자로 변환한다

	outlook	temp	humidity	wind	play
0	Sunny	Hot	High	Weak	No
1	Sunny	Hot	High	Strong	No
2	Overcast	Hot	High	Weak	Yes
3	Rain	Mild	High	Weak	Yes
4	Rain	Cool	Normal	Weak	Yes
5	Rain	Cool	Normal	Strong	No
6	Overcast	Cool	Normal	Strong	Yes
7	Sunny	Mild	High	Weak	No
8	Sunny	Cool	Normal	Weak	Yes
9	Rain	Mild	Normal	Weak	Yes
10	Sunny	Mild	Normal	Strong	Yes
11	Overcast	Mild	High	Strong	Yes
12	Overcast	Hot	Normal	Weak	Yes
13	Rain	Mild	High	Strong	No

```
tennis_data.outlook = tennis_data.outlook.replace('Sunny', 0)
tennis_data.outlook = tennis_data.outlook.replace('Overcast', 1)
tennis_data.outlook = tennis_data.outlook.replace('Rain', 2)

tennis_data.temp = tennis_data.temp.replace('Hot', 3)
tennis_data.temp = tennis_data.temp.replace('Mild', 4)
tennis_data.temp = tennis_data.temp.replace('Cool', 5)

tennis_data.humidity = tennis_data.humidity.replace('High', 6)
tennis_data.humidity = tennis_data.humidity.replace('Normal', 7)

tennis_data.wind = tennis_data.wind.replace('Weak', 8)
tennis_data.wind = tennis_data.wind.replace('Strong', 9)

tennis_data.play = tennis_data.play.replace('No', 10)
tennis_data.wind = tennis_data.play.replace('Yes', 11)
tennis_data
```

```
tennis_data.outlook = tennis_data.outlook.replace('Sunny', 0)
tennis_data.outlook = tennis_data.outlook.replace('Overcast', 1)
tennis_data.outlook = tennis_data.outlook.replace('Rain', 2)

tennis_data.temp = tennis_data.temp.replace('Hot', 3)
tennis_data.temp = tennis_data.temp.replace('Mild', 4)
tennis_data.temp = tennis_data.temp.replace('Cool', 5)

tennis_data.humidity = tennis_data.humidity.replace('High', 6)
tennis_data.humidity = tennis_data.humidity.replace('Normal', 7)

tennis_data.wind = tennis_data.wind.replace('Weak', 8)
tennis_data.wind = tennis_data.wind.replace('Strong', 9)

tennis_data.play = tennis_data.play.replace('No', 10)
tennis_data.play = tennis_data.play.replace('Yes', 11)

tennis_data
```

	outlook	temp	humidity	wind	play
0	0	3	6	8	10
1	0	3	6	9	10
2	1	3	6	8	11
3	2	4	6	8	11
4	2	5	7	8	11
5	2	5	7	9	10
6	1	5	7	9	11
7	0	4	6	8	10
8	0	5	7	8	11
9	2	4	7	8	11
10	0	4	7	9	11
11	1	4	6	9	11
12	1	3	7	8	11
13	2	4	6	9	10

◦ 피처는 X 개체에 저장하고, 종속변수(play)는 y 개체에 저장한 후, 학습용 데이터 세트와 평가용 데이터 세트로 구분한다

```
X = np.array(pd.DataFrame(tennis_data, columns = [‘outlook’, ‘temp’, ‘humidity’, ‘wind’]))
y = np.array(pd.DataFrame(tennis_data, columns = [‘play’]))

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=15)
```

```
X = np.array(pd.DataFrame(tennis_data, columns = ['outlook', 'temp', 'humidity', 'wind']))
y = np.array(pd.DataFrame(tennis_data, columns = ['play']))

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=15)
```

▪ 나이브 베이즈 모델 생성

- Gaussian Naive Bayes 모듈을 gnb_clf 개체로 저장한다
- gnb_clf 의 fit( ) 함수에 학습용 데이터 세트를 입력하여 학습용 모델을 만든다

```
gnb_clf = GaussianNB( )
gnb_clf = gnb_clf.fit(X_train, y_train)
```

```
gnb_clf = GaussianNB()
gnb_clf = gnb_clf.fit(X_train, y_train)
```

◦ predict( ) 함수를 이용하여 평가용 데이터 세트를 예측한 후, 그 결과를 프린트 한다

```
gnb_prediction = gnb_clf.predict(X_test)
print(gnb_prediction)
```

```
gnb_prediction = gnb_clf.predict(X_test)
print(gnb_prediction)
```

```
[10 10 11 11]
```

◦ 예측 결과, 4개에 대한 분류 예측 값이 제시되었다

◦ 첫 번째와 두 번째는 10으로 No로 분류되고, 나머지는 11로 Yes 로 분류되었다

- 분류 성능 측정
  - confusion matrix
  - precision, recall, f-measure
  - f1-score
  - accuracy

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score

print("Confusion Matrix: ")
print(confusion_matrix(y_test, gnb_prediction))
```

  - confusion matrix

```
print("Confusion Matrix: ")
print(confusion_matrix(y_test, gnb_prediction))

Confusion Matrix:
[[1 0]
 [1 2]]
```

행: 평가용 데이터, 열: 예측용 데이터
1행 1열: 정답과 예측 일치가 1개
1행 2열: 정답과 예측 불일치가 1 개
2행 1열: 정답과 예측 불일치가 1개
2행 2열: 정답과 예측 일치가 2개
전체 4개중 3개 일치, 1개 불일치

```
print('Classification Report')
print(classification_report(y_test, gnb_prediction))
```

```
print('Classification Report')
print(classification_report(y_test, gnb_prediction))
```

```
Classification Report
              precision    recall  f1-score   support

          10       0.50      1.00      0.67         1
          11       1.00      0.67      0.80         3

   micro avg       0.75      0.75      0.75         4
   macro avg       0.75      0.83      0.73         4
weighted avg       0.88      0.75      0.77         4
```

```
fmeasure = round(f1_score(y_test, gnb_prediction, average= 'weighted'), 2)
accuracy = round(accuracy_score(y_test, gnb_prediction, normalize = True), 2)

df_nbclf = pd.DataFrame(columns = ['Classifier', 'F-Measure', 'Accuracy'])
df_nbclf.loc[len(df_nbclf)] = ['Naïve Bayes', fmeasure, accuracy]
```

```
fmeasure = round(f1_score(y_test, gnb_prediction, average = 'weighted'), 2)
accuracy = round(accuracy_score(y_test, gnb_prediction, normalize = True ), 2)
```

```
df_nbclf = pd.DataFrame(columns = ['Classifier', 'F-Measure', 'Accuracy'])
df_nbclf.loc[len(df_nbclf)] = ['Naive Bayes', fmeasure, accuracy]
df_nbclf
```

	Classifier	F-Measure	Accuracy
0	Naive Bayes	0.77	0.75

- Precision(정확률, 정밀도)
  - 정확하게 분류한 비율
  - 10으로 2건 분류했지만 정확한 건은 1개
- Recall(재현율)
  - 실제 값 중 정확하게 분류한 비율
  - 11이 3건인데, 2건만 정확하게 분류하여 0.67
- f-1 score
  - 정확률과 재현율을 결합한 평가지표로 어느 한쪽으로 치우치지 않을 경우 높은 지표값을 가진다
- Accuracy(정확도)
  - 예측이 정확한 건수 / 전체 예측 건수 : 3/4 = 0.75

- 실제 데이터를 입력하여 분류 확인
  - outlook: Sunny, 0
  - temp: Cool, 5
  - humidity: High, 6
  - wind: Strong, 9
  - 테니스 경기 여부(No: 10, Yes: 11)→ No

```
A_prediction = gnb_clf.predict([[0, 5, 6, 9]])
print(A_prediction)
```

```
A_prediction = gnb_clf.predict([[0, 5, 6, 9]])
print(A_prediction)
```

```
[10]
```

요약

- 라소 회귀
- 분류용 선형 모델
- 나이브 베이즈 분류기

다음 시간

- 텍스트 분석