빅데이터분석 실습

SVM

데이터 사이언스 전공

담당교수: 곽철완

강의 내용

- 커널 서포트 벡터 머신
  - 커널 SVM의 필요성
  - 커널 기법(가우시안 커널)
  - SVM 매개변수 튜닝
- 분류 예측의 불확실성 추정
  - predict_proba 함수 사용

1. 커널 서포트 벡터 머신

- **특징**
  - 선형 SVM보다 복잡한 모델을 만들기 위한 알고리즘
  - 직선이나 초평면은 유연하지 못하여 피처 수가 적은 데이터 세트에서는 선형 모델이 제한적임
  - 선형 모델을 유연하게 만드는 방법은 피처들을 서로 곱하거나, 특정 피처를 거듭 제곱하여 피처를 추가하는 방법
- 그래프에서 한글이 깨질 때, 방지하기 위한 방법

```
import matplotlib
from matplotlib import font_manager, rc
font_name = font_manager.FontProperties(fname =
                                        “C:/Windows/Fonts/malgun.ttf”).get_name( )
rc(‘font’, family = font_name)
matplotlib.rcParams[‘axes.Unicode_minus’] = False
```

▪ 연습용 데이터 세트

- make_blobs 데이터 세트를 이용해 데이터 분류
  - 데이터 확인

```
from sklearn.datasets import make_blobs
X, y = make_blobs(centers=4, random_state=8)
y = y % 2

print("데이터 크기: " X.shape)
print("처음 5행:\n", X[:5])
```

```
데이터 크기: (100, 2)
```

```
처음 5행:
 [[-1.72161036 -1.48033142]
 [-3.6573384  -9.5482383 ]
 [ 7.0778163   0.99508772]
 [-1.36579859 -0.3148625 ]
 [-2.66521206 -3.12591651]]
```

◦ X[:, 0], X[:, 1] 내용 확인

```
print(X[:,0], X[:,1])
```

```
[-1.72161036 -3.6573384   7.0778163
 -5.27144331  7.91767139 -1.41284184
```

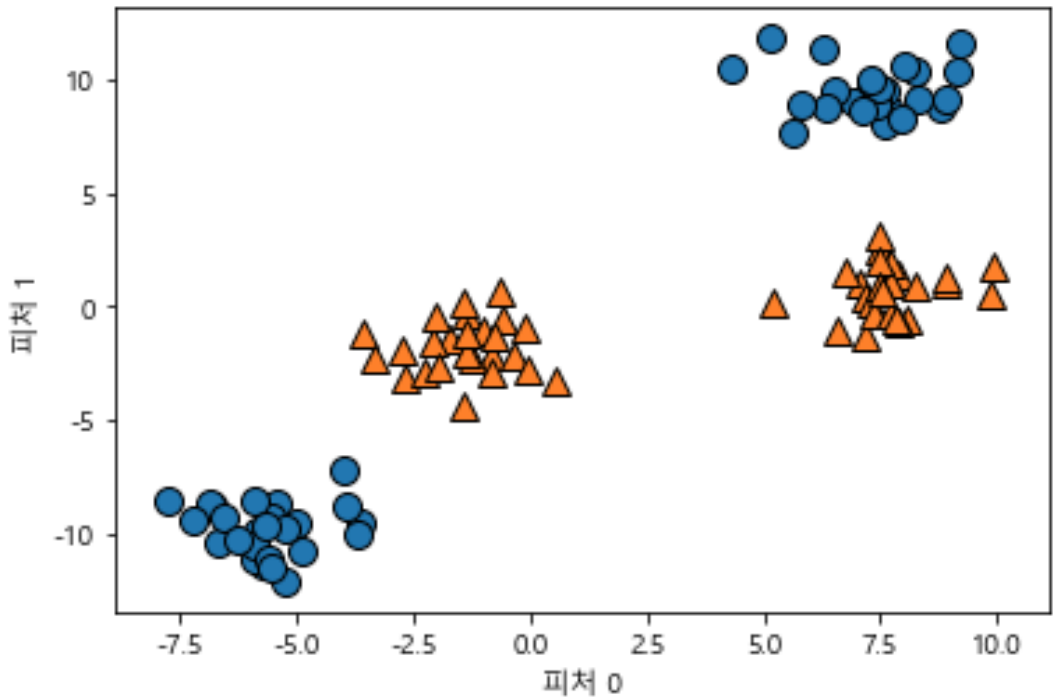
```
-5.52677154  7.49348552
 [ -1.48033142  -9.5482383    0.99508772  -0.314862
6348  -1.03318203
```

◦ make_blobs 데이터 세트를 이용해 데이터 분류

```
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
plt.xlabel("피처 0")
plt.ylabel("피처 1")
```

```
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
plt.xlabel("피처 0")
plt.ylabel("피처 1")
```

```
Text(0, 0.5, '피처 1')
```



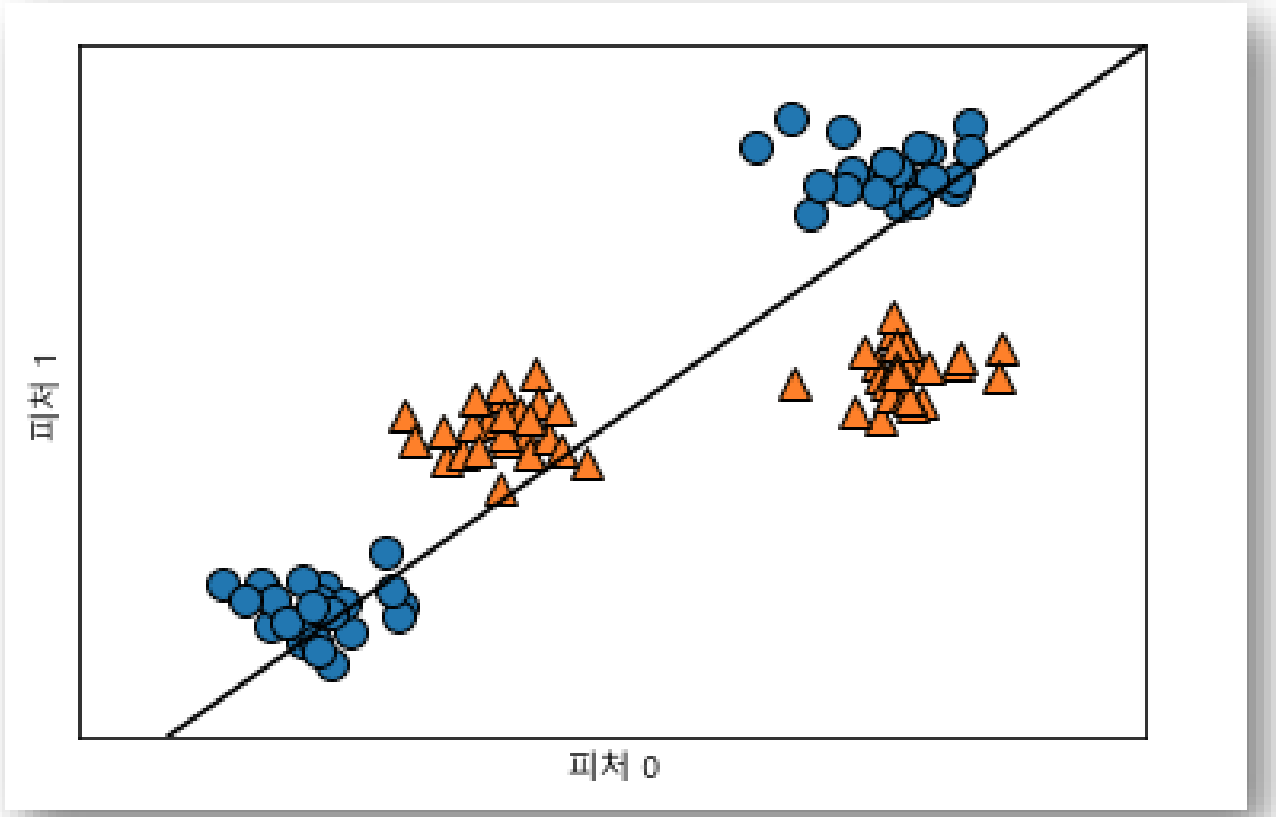
◦ 선형 SVM 적용한 결과는?

```
from sklearn.svm import LinearSVC
linear_svm = LinearSVC( ).fit(X, y)

mglearn.plots.plot_2d_separator(linear_svm, X)
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
plt.xlabel("피처 0")
plt.ylabel("피처 1")
```


```
from sklearn.svm import LinearSVC
linear_svm = LinearSVC( ).fit(X, y)

mglearn.plots.plot_2d_separator(linear_svm, X)
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
plt.xlabel("피처 0")
plt.ylabel("피처 1")
```



◦ 직선으로 2개 집단을 분류하기가 불가능하다

- 2번째 열의 데이터를 제곱하여 새로운 열로 추가한다
  - 회귀모델에서 피처 수를 늘려서 모델 만드는 법을 생각
  - numpy의 hstack( ) 함수를 이용하여 열(피처)을 하나 만들어 추가

```
X_new = np.hstack([X, X[:, 1:]**2])
```

- 3차원 그래프 그리기

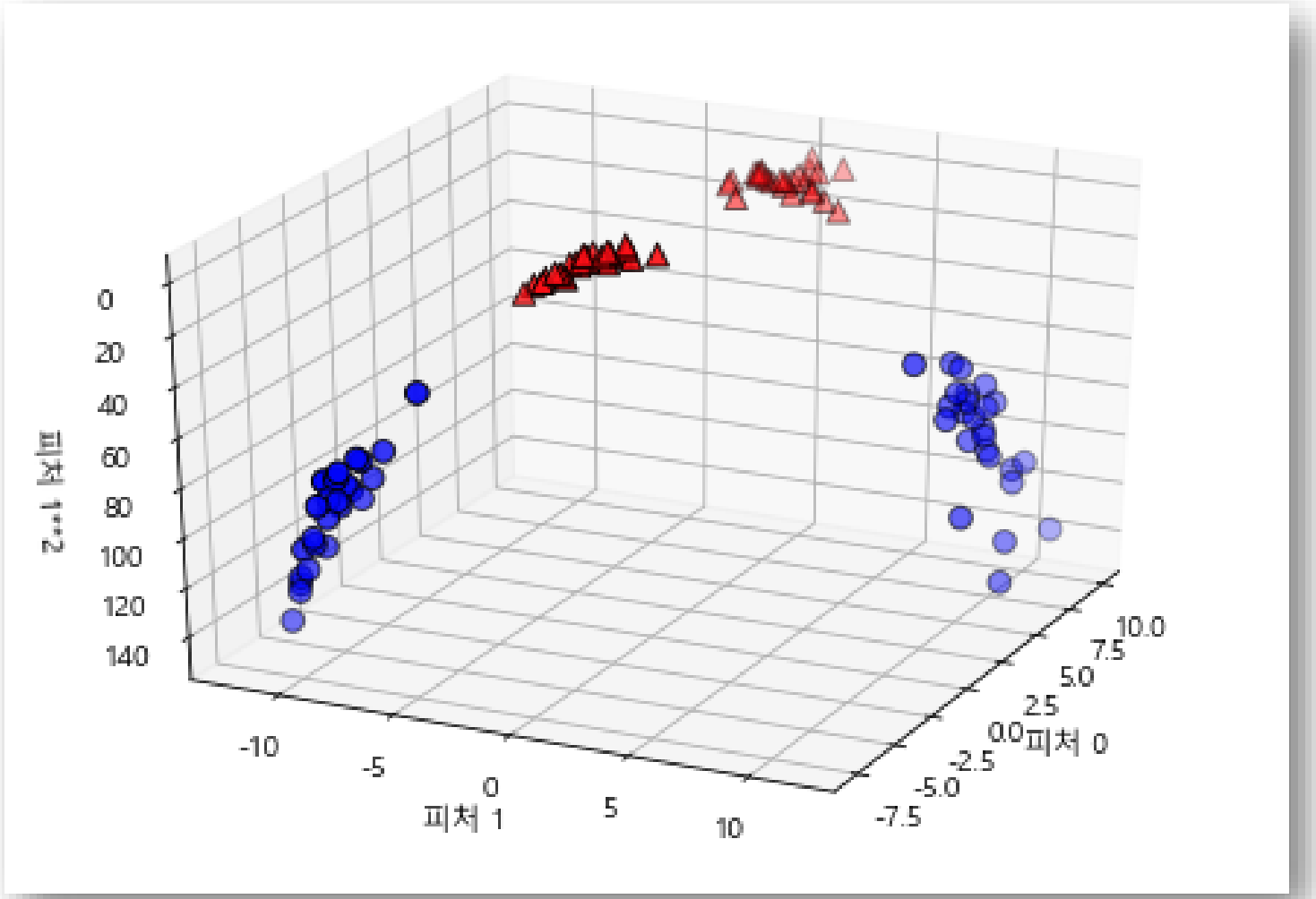
```
from mpl_toolkits.mplot3d import Axes3D, axes3d
figure = plt.figure( )
ax = Axes3D(figure, elev = -152, azim = -26)
mask = y == 0
ax.scatter(X_new[mask, 0], X_new[mask, 1], X_new[mask, 2],
      c='b', cmap =mglearn.cm2, s=60, edgecolor='k')
```

```
ax.scatter(X_new[~mask, 0], X_new[~mask, 1], X_new[~mask, 2],
      c='r', marker = '^', cmap=mglearn.cm2, s=60, edgecolor='k')
ax.set_xlabel("피처 0")
ax.set_ylabel("피처 1")
ax.set_zlabel("피처 1**2")
```

```
X_new = np.hstack([X, X[:, 1:]**2])
from mpl_toolkits.mplot3d import Axes3D, axes3d
figure = plt.figure( )

ax = Axes3D(figure, elev = -152, azim = -26)
mask = y == 0
ax.scatter(X_new[mask, 0], X_new[mask, 1], X_new[mask, 2], c='b', cmap =mglearn.cm2, s=60, edgecolor='k')
ax.scatter(X_new[~mask, 0], X_new[~mask, 1], X_new[~mask, 2], c='r', marker = '^', cmap=mglearn.cm2, s=60, edgecolor='k')
ax.set_xlabel("피처 0")
ax.set_ylabel("피처 1")
ax.set_zlabel("피처 1**2")
```

- elev: 입면도 디폴트 30, azim: 방위각, 디폴트 -60
- c: 색깔, cmap: 컬러 맵

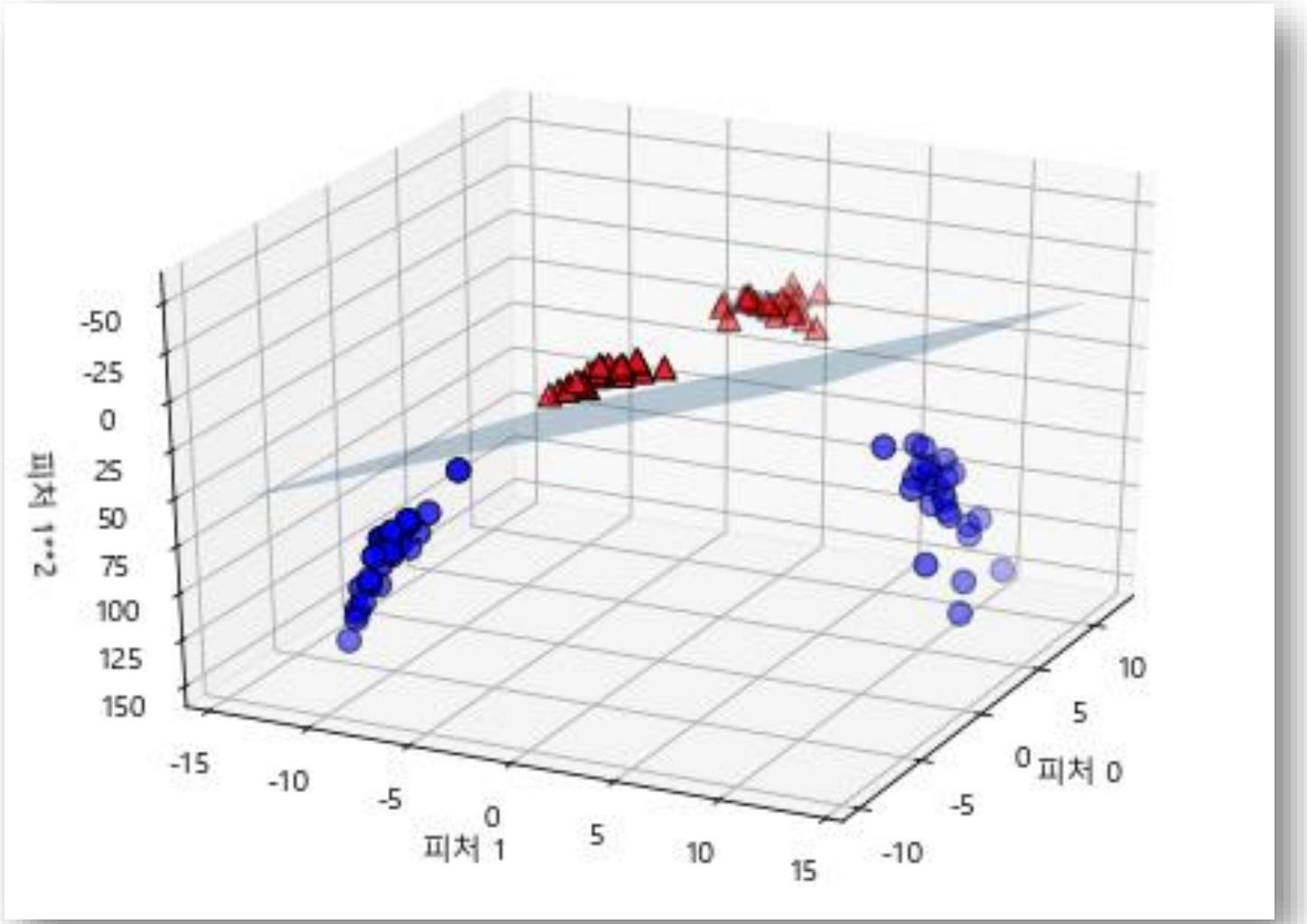

0
20
40
60
80
100
120
140
피처 2
-10
-5
0
5
10
피처 1
10.0
7.5
5.0
2.5
0.0
-2.5
-5.0
-7.5
피처 0

◦ 3차원 공간을 사용하면 선형 모델을 이용해서도 분류가 가능

```
linear_svm_3d = LinearSVC( ).fit(X_new, y)
coef, intercept = linear_svm_3d.coef_.ravel( ), linear_svm_3d.intercept_
figure = plt.figure( )
ax = Axes3D(figure, elev = -152, azim = -26)
xx = np.linspace(X_new[:, 0].min( ) -2, X_new[:, 0].max( ) + 2, 50)
yy = np.linspace(X_new[:, 1].min( ) -2, X_new[:, 1].max( ) + 2, 50)
```


```
linear_svm_3d = LinearSVC( ).fit(X_new, y)
coef, intercept = linear_svm_3d.coef_.ravel( ), linear_svm_3d.intercept_
figure = plt.figure( )
ax = Axes3D(figure, elev = -152, azim = -26)
xx = np.linspace(X_new[:, 0].min( ) -2, X_new[:, 0].max( ) + 2, 50)
yy = np.linspace(X_new[:, 1].min( ) -2, X_new[:, 1].max( ) + 2, 50)
```

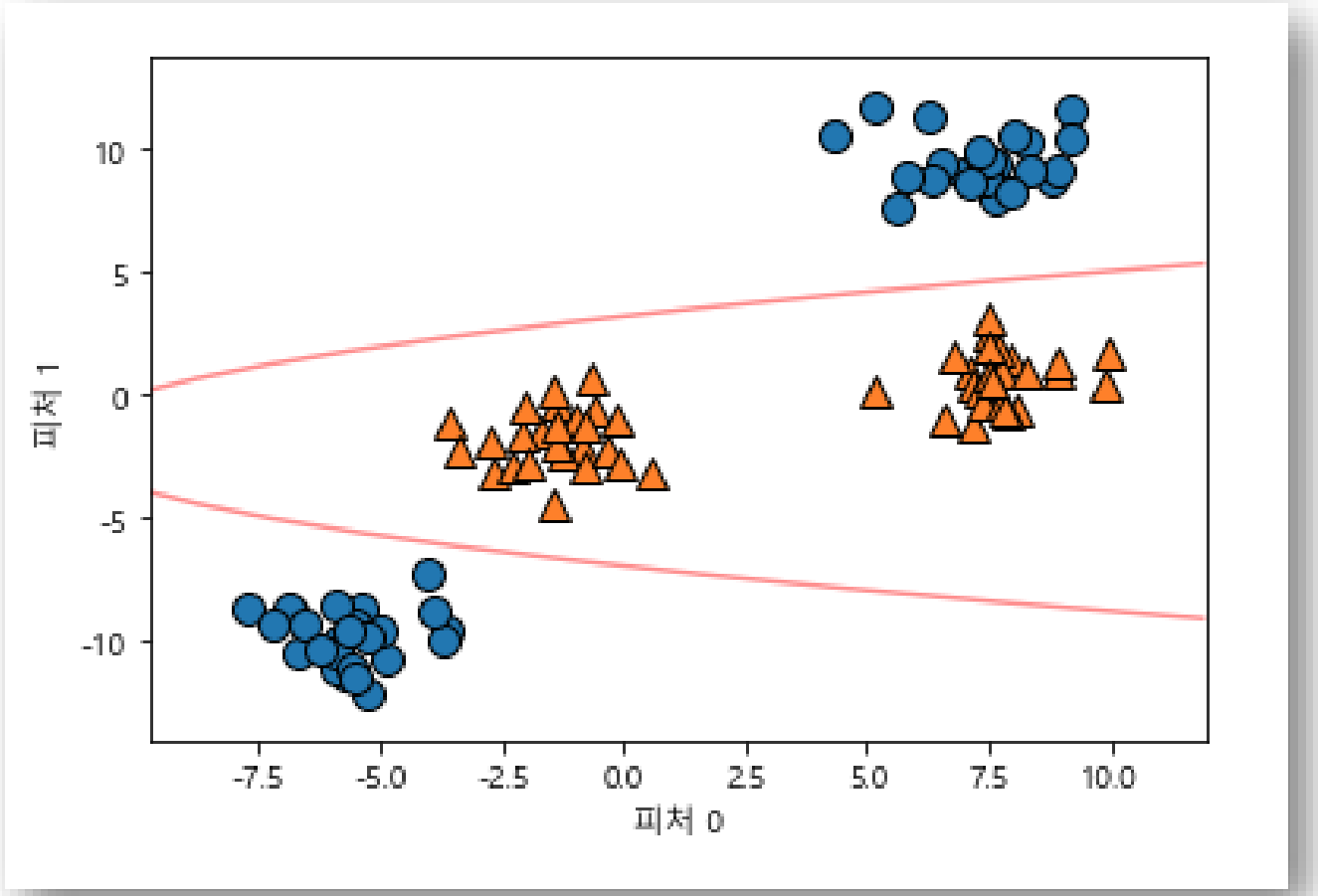
```
XX, YY = np.meshgrid(xx, yy)
ZZ = (coef[0] * XX + coef[1] * YY + intercept) / -coef[2]
ax.plot_surface(XX, YY, ZZ, rstride=8, cstride=8, alpha=0.3)
ax.scatter(X_new[mask, 0], X_new[mask, 1], X_new[mask, 2],
      c='b', cmap =mglearn.cm2, s=60, edgecolor='k')
ax.scatter(X_new[~mask, 0], X_new[~mask, 1], X_new[~mask, 2],
       c='r', marker = '^', cmap=mglearn.cm2, s=60, edgecolor='k')
ax.set_xlabel("피처 0")
ax.set_ylabel("피처 1")
ax.set_zlabel("피처 1**2")
```


-50
-25
0
25
50
75
100
125
150
피처 1**2
-15
-10
-5
0
5
10
15
피처 1
10
5
0
-5
-10
피처 0

◦ 처음의 2가지 피처를 기준으로 이 선형 SVM 모델을 보면 직선에 가까운 타원형을 보여준다

```
ZZ = YY ** 2
dec = linear_svm_3d.decision_function(np.c_[XX.ravel( ), YY.ravel( ), ZZ.ravel( )])
plt.contour(XX, YY, dec.reshape(XX.shape), levels=[dec.min( ), 0, dec.max( )],
                    cmap=mglearn.cm2, alpha=0.5)
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
plt.xlabel(“피처 0”)
plt.ylabel(“피처 1”)
```

```
ZZ = YY ** 2
dec = linear_svm_3d.decision_function(np.c_[XX.ravel( ), YY.ravel( ), ZZ.ravel( )])
plt.contour(XX, YY, dec.reshape(XX.shape), levels=[dec.min( ), 0, dec.max( )], cmap=mglearn.cm2, alpha=0.5)
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
plt.xlabel("피처 0")
plt.ylabel("피처 1")
```


10
5
0
-5
-10
피처 1
-7.5
-5.0
-2.5
0.0
2.5
5.0
7.5
10.0
피처 0

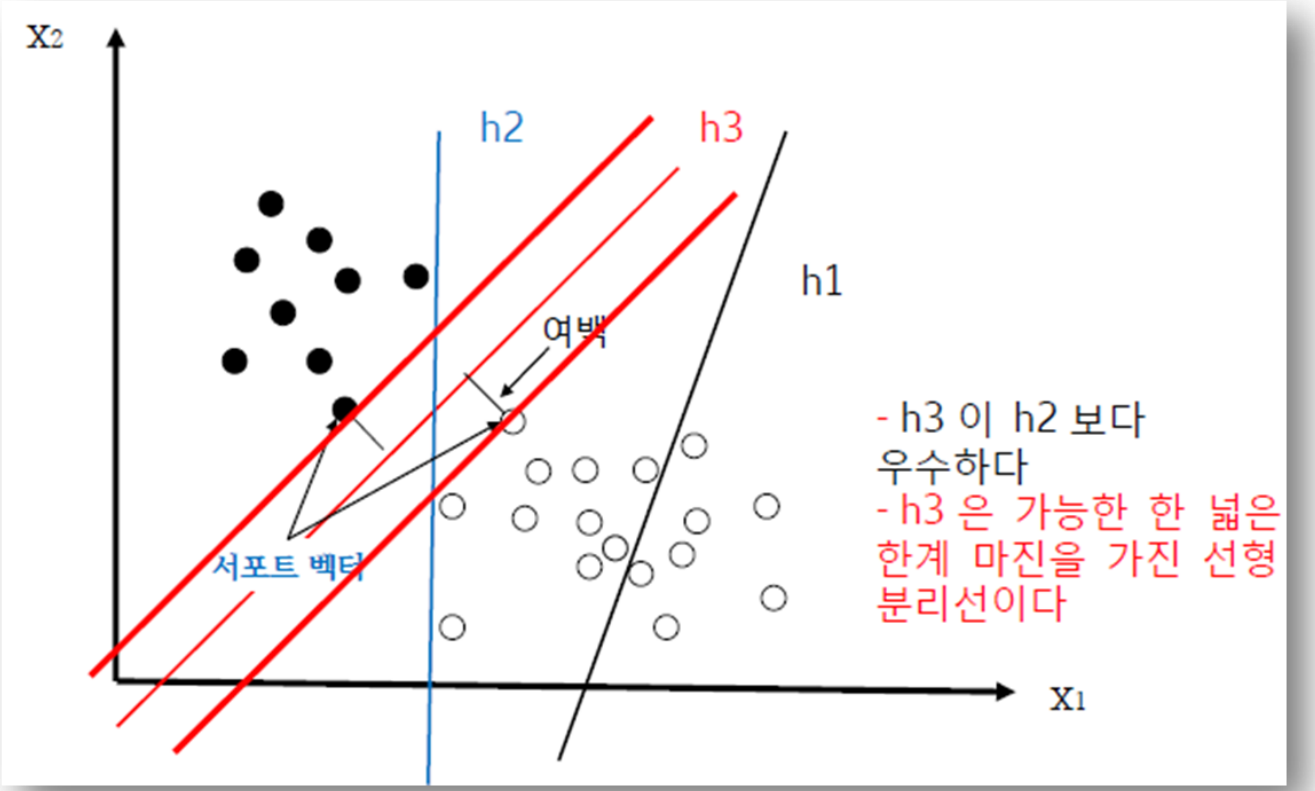
▪ 커널 기법

- 선형 SVM으로 분류가 어려운 데이터 세트에 새로운 피처(기존 피처를 제곱하여 만듦)를 추가하여 분류가 가능하게 하였다
- 하지만 어떤 피처를 제곱하여 피처로 추가할 지 모른다(모든 경우의 수를 다 만든다면 연산 비용이 커진다)
- 수학적인 기교를 이용하여 고차원에서 분류기를 학습시킬 수 있다 → 커널 기법 kernel trick → 실제 데이터를 확장하지 않고, 확장된 피처에 대한 데이터 포인트 들의 거리를 계산
- 방법
  - 다항식 커널: 피처의 가능한 조합을 지정된 차수까지 모두 계산
  - 가우시안 커널(radial basis function 커널): 차원이 무한한 피처 공간에 매핑

▪ SVM 이해

- 선형 SVM으로 분류가 어려운 데이터 세트에 새로운 피처(기존 피처를 제곱하여 만듦)를 추가하여 분류가 가능하게 하였다
- 하지만 어떤 피처를 제곱하여 피처로 추가할 지 모른다(모든 경우의 수를 다 만든다면 연산 비용이 커진다)
- 수학적인 기교를 이용하여 고차원에서 분류기를 학습시킬 수 있다 → 커널 기법 kernel trick → 실제 데이터를 확장하지 않고, 확장된 피처에 대한 데이터 포인트 들의 거리를 계산
- 방법
  - 다항식 커널: 피처의 가능한 조합을 지정된 차수까지 모두 계산
  - 가우시안 커널(radial basis function 커널): 차원이 무한한 피처 공간에 매핑

- SVM 기초
  - 분류된 두 범주 사이에서 해당 범주에 속하는 데이터의 분류 오차를 줄이면서 여백(margin)을 최대화하는 결정 경계(decision boundary)를 찾는 방법
  - 두 범주 사이에 여러 경계가 존재할 수 있지만, 여백을 최대화하는 경계를 최대 마진 분류기라 함
  - h3 : 결정 경계



◦ 가우시안 커널은 결정 경계를 가우시안 커널에 의해 계산한다

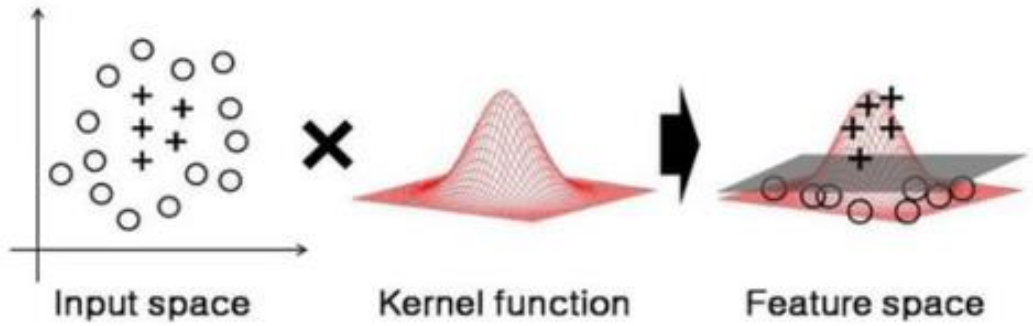
$$k(x_1, x_2) = \exp\left(-\gamma ||x_1 - x_2||^2\right)$$

데이터 포인트

유클리디안 거리

매개변수

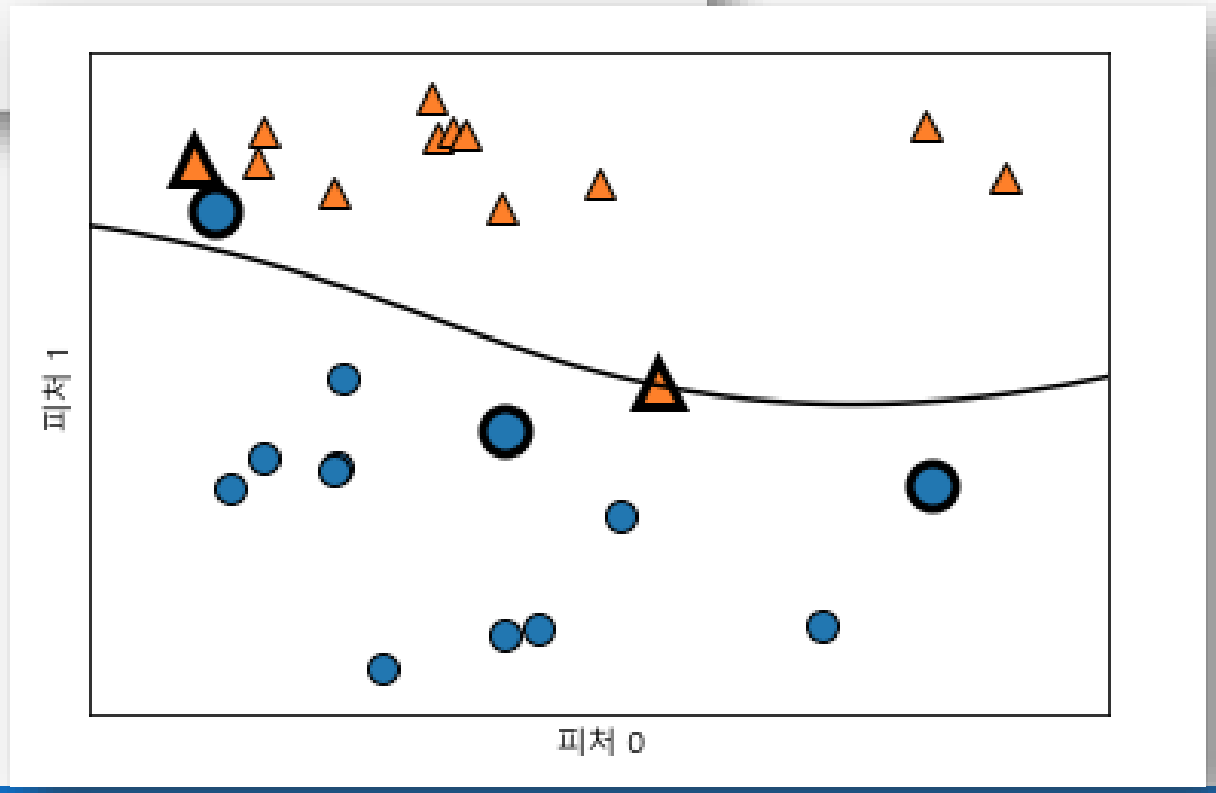
◦ 매개변수는 가우시안 커널의 폭을 제어한다



- forge 데이터 세트에 SVM 학습
  - 4: 결정 경계 그리기　　5: 데이터 포인트 그리기
  - 6: 서포트 벡터 표시하기(굵은 표시)

```
1  from sklearn.svm import SVC
2  X, y = mglearn.tools.make_handcrafted_dataset( )
3  svm = SVC(kernel = 'rbf', C=10, gamma=0.1).fit(X, y)
4  mglearn.plots.plot_2d_separator(svm, X, eps=.5)
5  mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
6  sv = svm.support_vectors_
7  sv_labels = svm.dual_coef_.ravel( ) > 0
8  mglearn.discrete_scatter(sv[:, 0], sv[:, 1], sv_labels , s=15, markeredgewidth=3)
9  plt.xlabel("피처 0")
10 plt.ylabel("피처 1")
```

```
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_
sv_labels = svm.dual_coef_.ravel( ) > 0
mglearn.discrete_scatter(sv[:, 0], sv[:, 1], sv_labels , s=15, markeredgewidth=3)
plt.xlabel("피처 0")
plt.ylabel("피처 1")
```



▪ SVM 매개변수 튜닝

- gamma 매개변수
  - 학습용 데이터 세트의 표준편차와 관련이 있는데
  - gamma 값이 클수록 작은 표준편차를 가지며 영향력이 미치는 범위가 적어진다
  - gamma 값은 결정 경계 곡선의 모양을 결정한다
- C 매개변수
  - C 값이 커지면 학습용 데이터 세트에 맞추려고 한다(과대적합)
  - C 값이 작아지면 계수가 0에 가까워져 피처가 분류에 영향을 미치지 못한다(피처의 중요도를 감소시킨다)

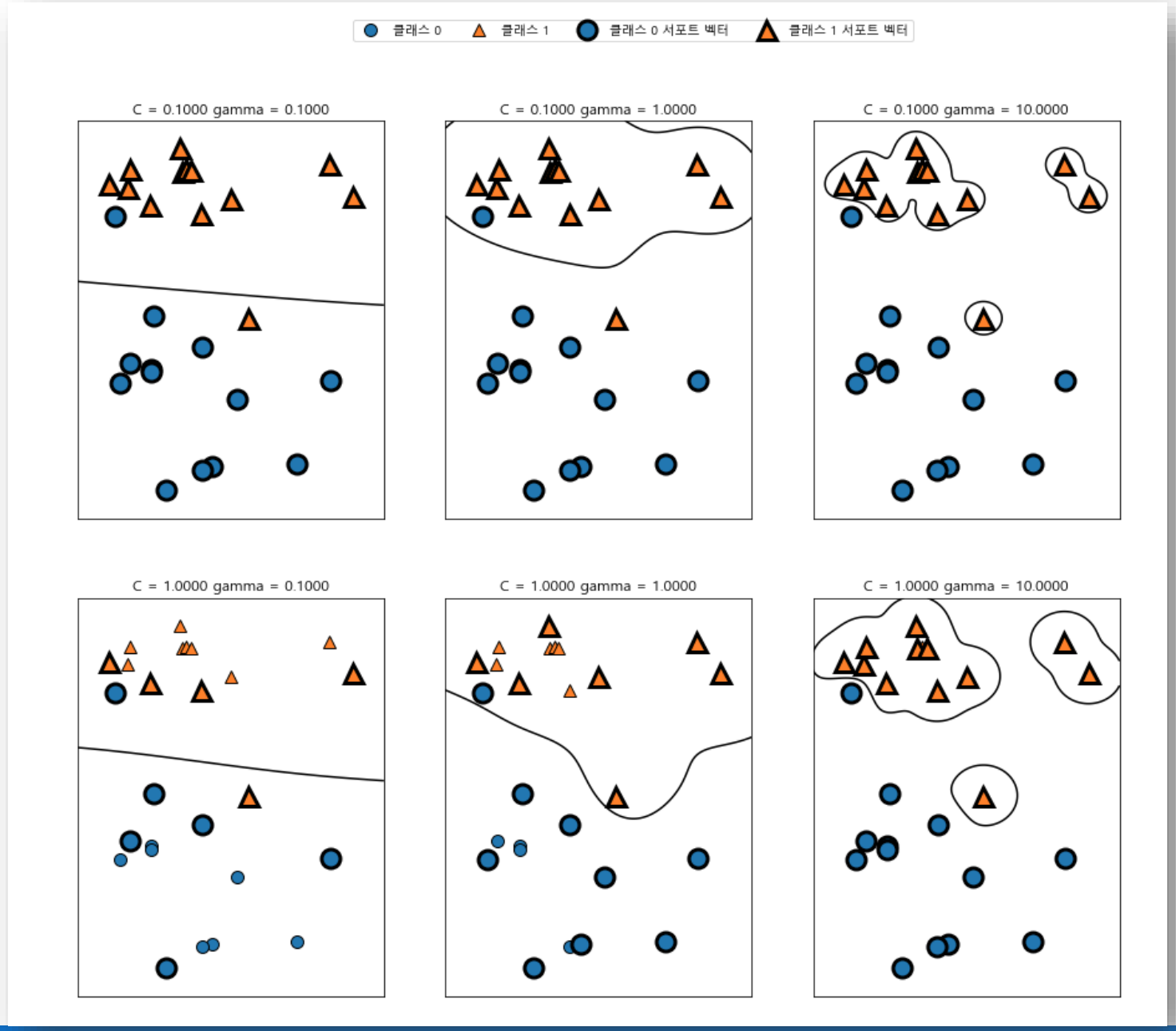
◦ 그림으로 비교

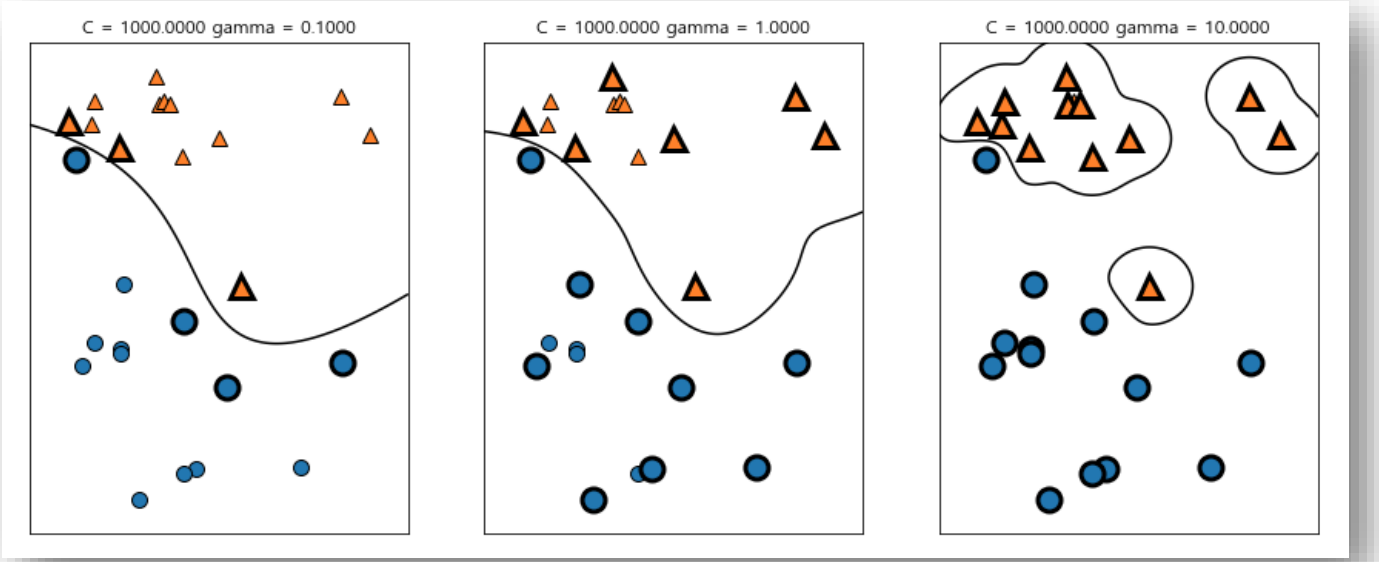
```
fig, axes = plt.subplots(3, 3, figsize=(15, 20))

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, ax=a)
axes[0, 0].legend([“클래스 0”, “클래스 1”, “클래스 0 서포트 벡터”,
                “클래스 1 서포트 벡터”], ncol=4, loc=(.9, 1.2))
```

```
fig, axes = plt.subplots(3, 3, figsize=(15, 20))

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(["클래스 0", "클래스 1", "클래스 0 서포트 벡터", "클래스 1 서포트 벡터"],
                  ncol=4, loc=(.9, 1.2))
```


클래스 0
클래스 1
클래스 0 서포트 벡터
클래스 1 서포트 벡터
C = 0.1000 gamma = 0.1000
C = 0.1000 gamma = 1.0000
C = 0.1000 gamma = 10.0000
C = 1.0000 gamma = 0.1000
C = 1.0000 gamma = 1.0000
C = 1.0000 gamma = 10.0000



- 오른쪽으로 이동하면서 gamma 매개변수를 0.1에서 10으로 증가 → 결정 경계가 포인트에 민감해진다. 큰 gamma 값은 더 복잡한 모델을 만든다
- 아래로 이동하면서 C 매개변수를 0.1에서 1000으로 증가 → 모델에 영향을 주어 결정 경계를 곡선으로 만든다

▪ SVM 장단점과 매개변수

- SVM은 강력한 모델이며, 다양한 데이터 세트에서도 잘 작동된다
- 피처가 적은 경우에도 복잡한 결정 경계를 만들 수 있다
- 하지만, 데이터 세트가 많을 때는 잘 작동하지 않는다
- SVM을 적용할 경우에는 매개변수를 잘 이용해야 한다

2. 분류 예측의 불확실성 추정

- 목적
  - 특정 알고리즘을 사용하여 분류 예측을 하였는데, 그 결과가 얼마나 정확한지 파악할 수 있다
- 종류
  - decision_function 함수
  - predict_proba 함수

▪ GradientBoostingClassifier 분류기

◦ 인위적으로 만든 2차원 데이터 세트를 이용해 비교한다

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.datasets import make_circles
from sklearn.model_selection import train_test_split
X, y = make_circles(noise = 0.25, factor = 0.5, random_state =1)
y_named = np.array(["blue", "red"])[y]
X_train, X_test, y_train-named, y_test_named, y_train,y_test =
          train_test_split(X, y_named, y, random_state=0)
gbrt = GradientBoostingClassifier(random_state=0)
gbrt.fit(X_train, y_train_named)
```

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.datasets import make_circles
from sklearn.model_selection import train_test_split
X, y = make_circles(noise = 0.25, factor = 0.5, random_state = 1)

y_named = np.array(["blue", "red"])[y]

X_train, X_test, y_train_named, y_test_named, y_train, y_test = train_test_split(X, y_named, y, random_state=0)

gbrt = GradientBoostingClassifier(random_state=0)
gbrt.fit(X_train, y_train_named)
```

```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='deviance', max_depth=3,
              max_features=None, 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=100,
              n_iter_no_change=None, presort='auto', random_state=0,
              subsample=1.0, tol=0.0001, validation_fraction=0.1,
              verbose=0, warm_start=False)
```

- predict_proba(예측 확률)
  - predict_proba의 출력은 각 클래스에 대한 확률이다
  - 이진 분류에서 값의 크기는 (데이터 크기, 2) 이다

```
print(“X_test.shape:”, X_test.shape)
print(“predict_proba 값의 형태 :”, gbrt.predict_proba(X_test).shape)
```

```
print("X_test.shape:", X_test.shape)
print("predict_proba 결과 형태:", gbrt.predict_proba(X_test).shape)
```

```
X_test.shape: (25, 2)
predict_proba 결과 형태: (25, 2)
```

- 각 행의 첫 번째 원소는 첫 번째 클래스의 predict_proba이다
- 두 번째 원소는 두 번째 클래스의 predict_proba이다
- 확률이기 때문에 값은 항상 0 ~ 1 이며, 두 클래스에 대한 확률의 합은 항상 1 이다

```
print("predict_proba : \n", gbrt.predict_proba(X_test[:5]))
```

```
print("predict_proba : \n", gbrt.predict_proba(X_test[:5]))

predict_proba :
 [[0.01573626 0.98426374]
 [0.84575649 0.15424351]
 [0.98112869 0.01887131]
 [0.97406775 0.02593225]
 [0.01352142 0.98647858]]
```

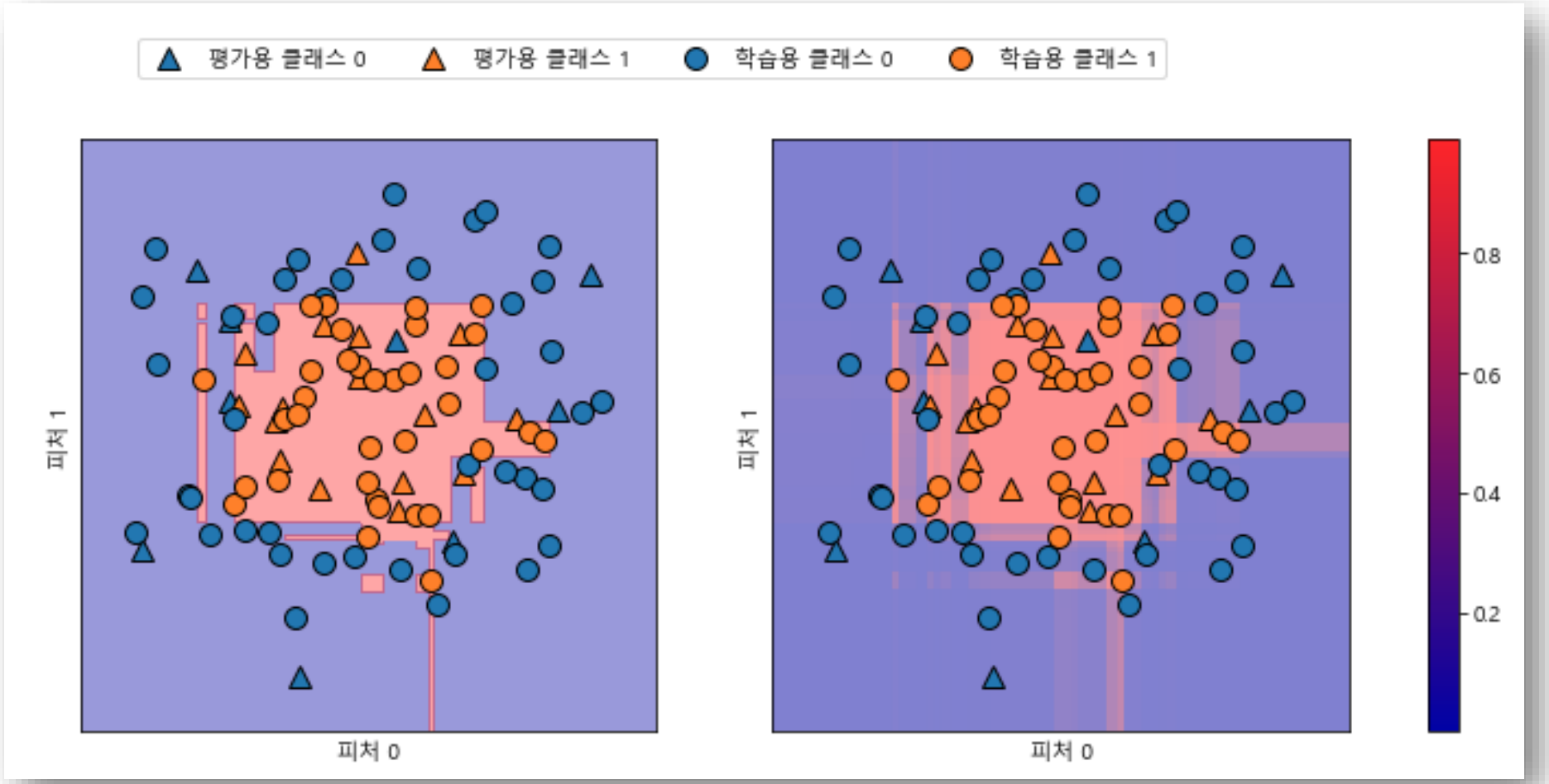
- predict_proba 값의 반영 정도는 모델과 매개변수 설정에 따라 결정된다
- 과대적합(overfitting) 모델은 잘못된 예측이라도 예측의 확신이 강한다
- 반대로, 과소적합(복잡도가 낮은) 모델은 예측에 불확실성이 크다
- 그러므로 불확실성과 모델의 정확도가 동등하면, 이 모델이 보정 calibration 되었다고 한다
  - 보정된 모델에서 70%의 확신을 가진 예측은 70%의 정확도를 나타낸다

◦ 결정 경계와 클래스 1의 확률 그리기

```
fig, axes = plt.subplots(1, 2, figsize = (13, 5))
mglearn.tools.plot_2d_separator(
      gbrt, X, ax=axes[0], alpha=.4, fill=True, cm=mglearn.cm2)
scores_image = mglearn.tools.plot_2d_scores(
      gbrt, X, ax=axes[1], alpha=.5, cm=mglearn.ReBl, function='predict_proba')
for ax in axes:
      mglearn.discrete_scatter(X_test[:, 0], X_test[:, 1], y_test,
                                    markers='^', ax=ax)
       mglearn.discrete_scatter(X_train[:, 0], X_train[:, 1], y_train,
                                    markers='o', ax=ax)
       ax.set_xlabel("피처 0")
       ax.set_ylabel("피처 1")
```

```
cbar = plt.colorbar(scores_image, ax=axes.tolist( ))
cbar.set_alpha(1)
cbar.draw_all( )
axes[0].legend(["평가용 클래스 0", "평가용 클래스 1", "학습용 클래스 0",
                "학습용 클래스 1"], ncol=4, loc=(.1, 1.1))
```

```
fig, axes = plt.subplots(1, 2, figsize = (13, 5))
mglearn.tools.plot_2d_separator(
      gbrt, X, ax=axes[0], alpha=.4, fill=True, cm=mglearn.cm2)
scores_image = mglearn.tools.plot_2d_scores(
   gbrt, X, ax=axes[1], alpha=.5, cm=mglearn.ReBl, function='predict_proba')
for ax in axes:
   mglearn.discrete_scatter(X_test[:, 0], X_test[:, 1], y_test, markers='^', ax=ax)
   mglearn.discrete_scatter(X_train[:, 0], X_train[:, 1], y_train, markers='o', ax=ax)
   ax.set_xlabel("피처 0")
   ax.set_ylabel("피처 1")
cbar = plt.colorbar(scores_image, ax=axes.tolist( ))
cbar.set_alpha(1)
cbar.draw_all( )
axes[0].legend(["평가용 클래스 0", "평가용 클래스 1", "학습용 클래스 0",
           "학습용 클래스 1"], ncol=4, loc=(.1, 1.1))
```



◦ 왼쪽 그림이 결정 경계이며, 오른쪽 그림이 클래스 1의 확률이다

요약

- 커널 서포트 벡터 머신
  - 커널 SVM의 필요성
  - 커널 기법(가우시안 커널)
  - SVM 매개변수 튜닝
- 분류 예측의 불확실성 추정
  - predict_proba 함수 사용