Support Vector Machine

Part 1. Linear SVM Classifiers

1. Create Datasets

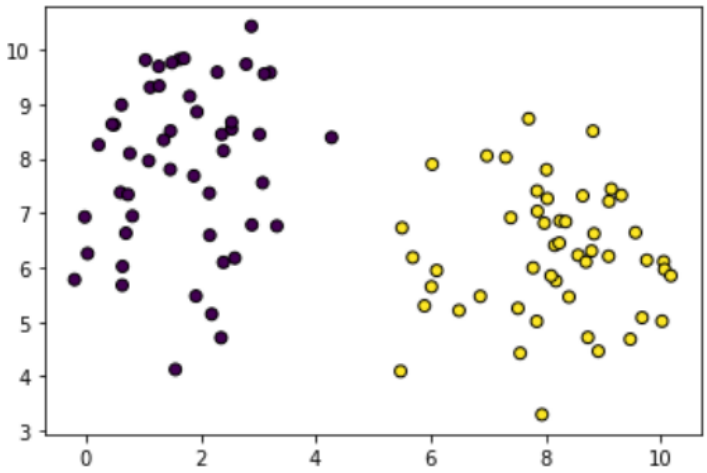
[Code]

```
from sklearn.datasets import make_blobs
X, y = make_blobs(n_samples = 100, centers = 2, cluster_std = 1.2, random_state = 40)

plt.scatter(X[:, 0], X[:, 1], c = y, edgecolor = 'k')
plt.show()
```

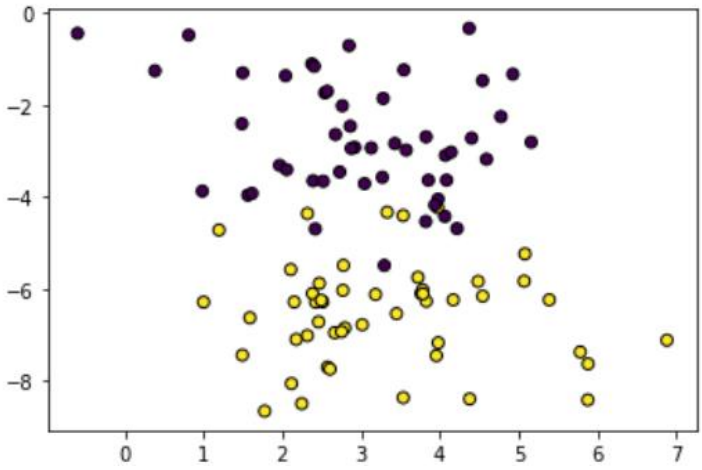
- 2개의 그룹으로 나누기 위해서 centers = 2를 이용. 나머지는 주어진 조건을 활용해서 make_blobs를 수행

- plt.scatter

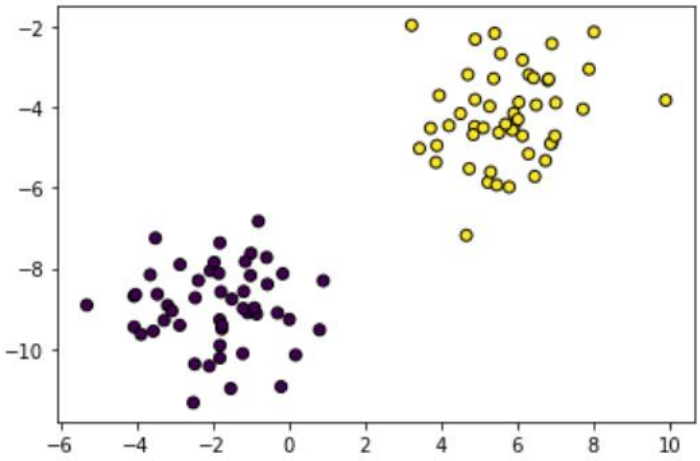
[random_state = 20]



[random_state = 30]



[random_state = 40]

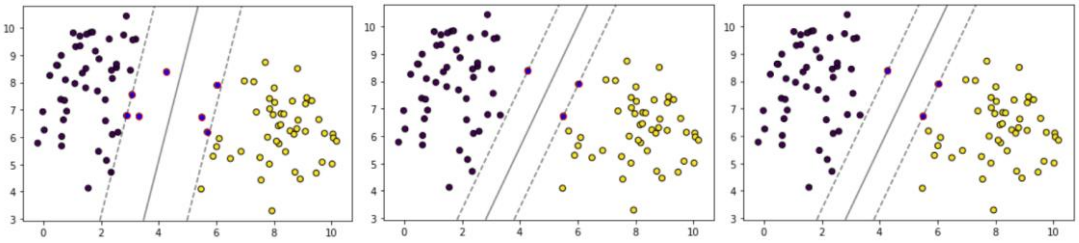


2. Train SVM

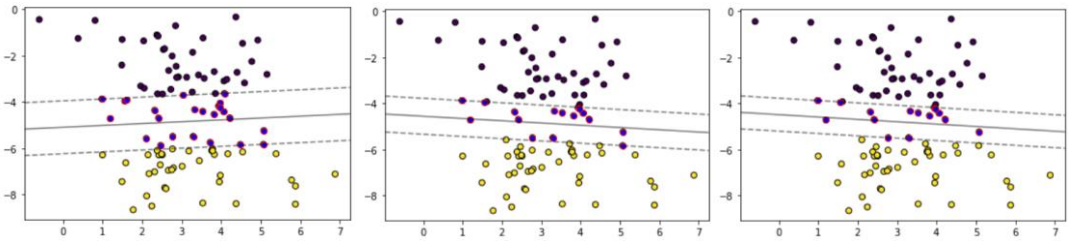
[Code]

```
# %%
# 2번
from sklearn.model_selection import train_test_split

# Train
model = SVC(kernel = 'linear', C = 10)
model.fit(X, y)

def SVC_decision_plot(model, ax = None, plot_support = True):
    if ax is None:
        ax = plt.gca()
    xlim = ax.get_xlim()
    ylim = ax.get_ylim()

    # grid 표시
    x = np.linspace(xlim[0], xlim[1], 30)
    y = np.linspace(ylim[0], ylim[1], 30)
    Y, X = np.meshgrid(y, x)
    xy = np.vstack([X.ravel(), Y.ravel()]).T
    Z = model.decision_function(xy).reshape(X.shape)

    # margin에 따른 decision boundary 표시
    ax.contour(X, Y, Z, colors='k',
               levels = [-1, 0, 1], alpha = 0.5,
               linestyles = ['--', '-', '--'])

    ax.set_xlim(xlim)
    ax.set_ylim(ylim)

plt.scatter(X[:, 0], X[:, 1], c = y, edgecolor = 'k')
# 해당하는 것들 표시
plt.scatter(model.support_vectors_[:,0], model.support_vectors_[:,1],
            edgecolor = 'r', facecolors = 'b')

SVC_decision_plot(model)
```

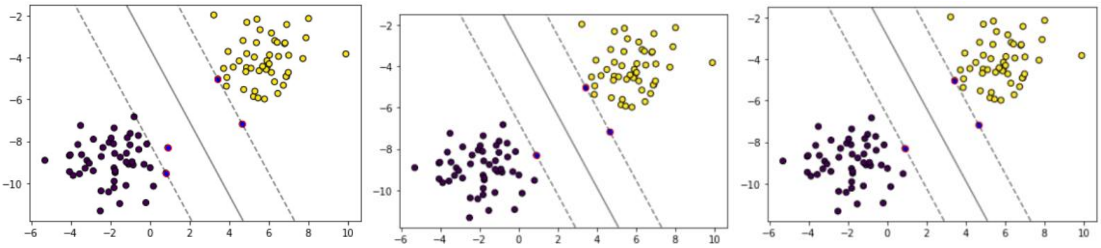
[random_state = 20, C = 0.1, 1, 10 순]



[random_state = 30, C = 0.1, 1, 10 순]



[random_state = 40, C = 0.1, 1, 10 순]



- C값이 커지면 커질수록 hard margin 분류에 가까워지는 것을 알 수 있다.

- 얼핏 보면 분류가 잘 되는 것으로 보이기 때문에 hard margin 분류의 단점은 데이터가 선형적으로 구분되어야 하고, 이상치에 민감하다는 점이다. 이런 단점을 해결하기 위해서 도로의 폭을 가능한 넓게 유지, 마진 오류 사이의 적절한 균형을 잡는 소프트 마진 분류를 하기 위해 C값을 적절히 조정해야 한다.

- 선형적으로 완벽하게 hard margin 하기 위해서는 이상치를 없애야 한다.

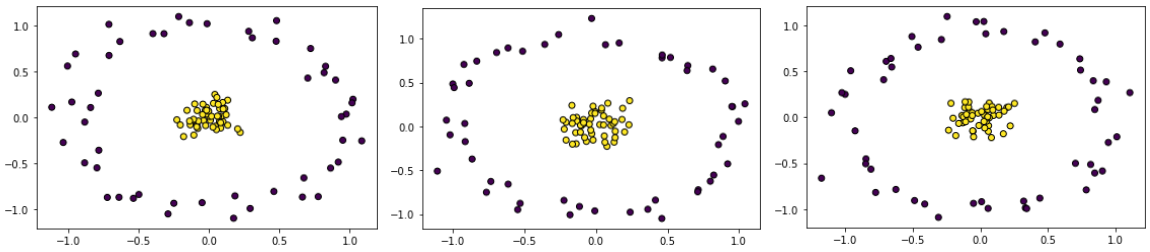
Part 2. Nonlinear SVM

1. Create Datasets

[Code]

```
# %%
# Part 2
# 1번
from sklearn.datasets import make_circles
# make_circle, noise, factor 인자 전달.
X, y = make_circles(n_samples = 100, noise = 0.1, factor = 0.1, random_state = 40)

plt.scatter(X[:, 0], X[:, 1], c = y, edgecolor = 'k')
plt.show()
```

[n_samples = 100, noise = 0.1, factor = 0.1, random_state = 20, 30, 40 순]

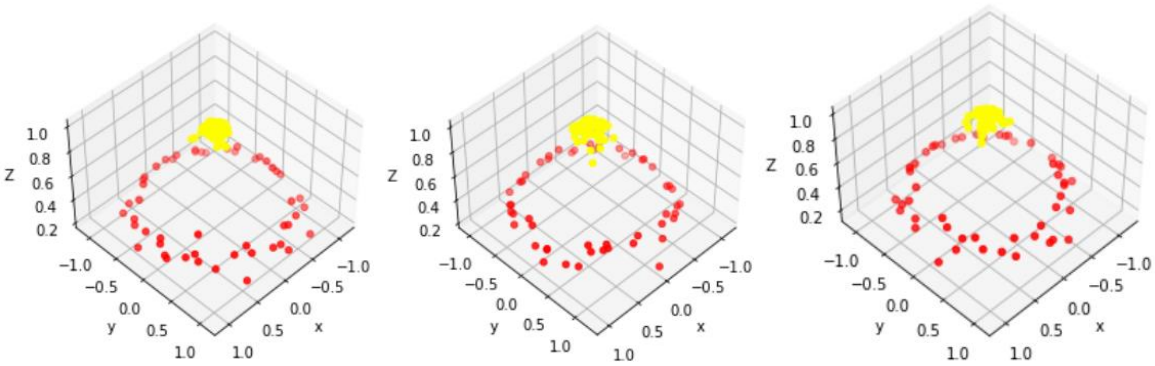


2. Kernel Function

[Code]

```
# %%
# 2번
from mpl_toolkits import mplot3d
from ipywidgets import interact, fixed

# RBF 식 표현
Z = np.exp(-(abs(X)**2).sum(1))

# grid, 시각화를 통해 각도 표현
def plot3D(elev = 45, azim = 45, X = X, y = y):
    ax = plt.subplot(projection = '3d')
    ax.scatter3D(X[:, 0], X[:, 1], Z, c = y, cmap = 'autumn')
    ax.view_init(elev = elev, azim = azim)
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_zlabel('Z')

# interact, plot3D 함수 이용한 시각화
interact(plot3D, elev = [45, 45], X = fixed(X), y = fixed(y));
```

[n_samples = 100, noise = 0.1, factor = 0.1, random_state = 20, 30, 40 순]

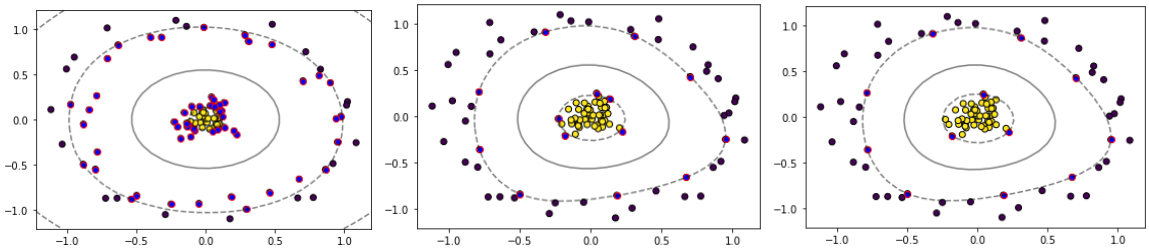


3. Train SVM

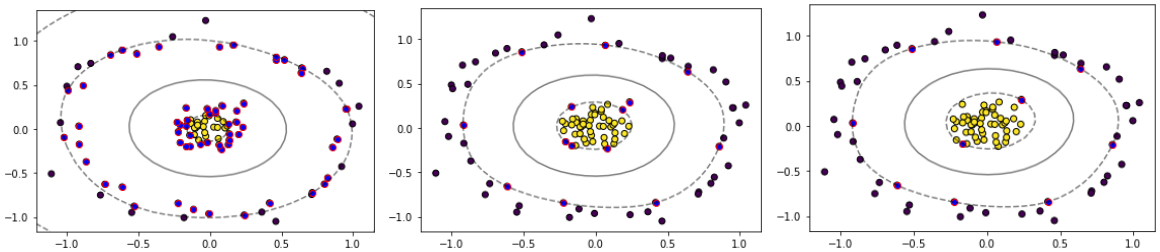
[Code]

```
# %%
# 3번

model = SVC(kernel = 'rbf', C = 10)
model.fit(X, y)

def SVC_decision_plot(model, ax = None, plot_support = True):
    if ax is None:
        ax = plt.gca()
    xlim = ax.get_xlim()
    ylim = ax.get_ylim()

    # grid 표시
    x = np.linspace(xlim[0], xlim[1], 30)
    y = np.linspace(ylim[0], ylim[1], 30)
    Y, X = np.meshgrid(y, x)
    xy = np.vstack([X.ravel(), Y.ravel()]).T
    Z = model.decision_function(xy).reshape(X.shape)

    # margin에 따른 decision boundary 표시
    ax.contour(X, Y, Z, colors='k',
               levels = [-1, 0, 1], alpha = 0.5,
               linestyles = ['--', '-', '--'])

    ax.set_xlim(xlim)
    ax.set_ylim(ylim)

plt.scatter(X[:, 0], X[:, 1], c = y, edgecolor = 'k')
# 해당하는 것들 표시
plt.scatter(model.support_vectors_[:, 0], model.support_vectors_[:, 1],
            edgecolor = 'r', facecolors = 'b')

SVC_decision_plot(model)
```

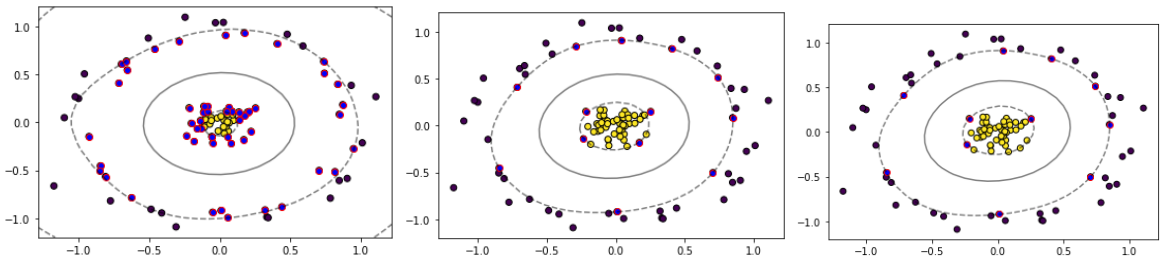
[random_state = 20, C = 0.1, 1, 10 순]



[random_state = 30, C = 0.1, 1, 10 순]



[random_state = 40, C = 0.1, 1, 10 순]



- circle도 앞에서의 blob과 마찬가지로 C값이 커지면 커질수록 hard margin 분류에 가까워지는 것을 알 수 있다.(더 이상치 없이 잘 분류된다.

- 얼핏 보면 분류가 잘 되는 것으로 보이기 때문에 hard margin 분류의 단점은 데이터가 선형적으로 구분되어야 하고, 이상치에 민감하다는 점이다. 이런 단점을 해결하기 위해서 도로의 폭을 가능한 넓게 유지, 마진 오류 사이의 적절한 균형을 잡는 소프트 마진 분류를 하기 위해 C값을 적절히 조정해야 한다.

- 선형적으로 완벽하게 hard margin 하기 위해서는 이상치를 없애야 한다.