

⑥サポートベクターマシン_実装演習

In [3]:

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
```

=====

1. サポートベクターマシン(SVM) → 線形分離可能な場合

- ・ 訓練データの生成① (線形分離可能)

In [4]:

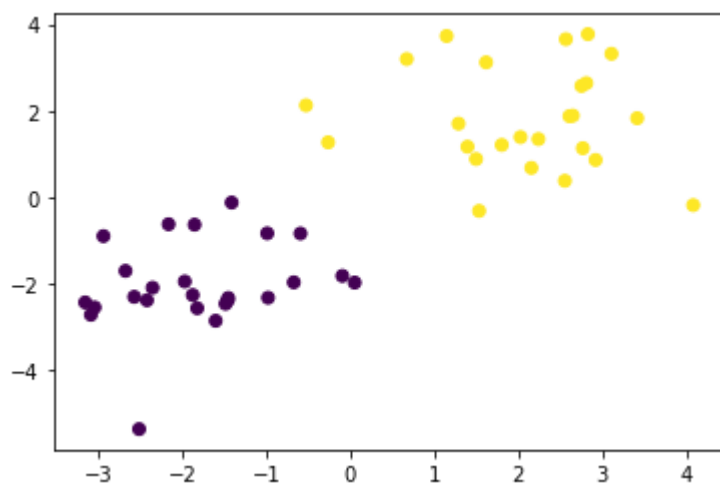
```
def gen_data():
    x0 = np.random.normal(size=50).reshape(-1, 2) - 2.
    x1 = np.random.normal(size=50).reshape(-1, 2) + 2.
    X_train = np.concatenate([x0, x1])
    ys_train = np.concatenate([np.zeros(25), np.ones(25)]).astype(np.int)
    return X_train, ys_train
```

In [5]:

```
X_train, ys_train = gen_data()
plt.scatter(X_train[:, 0], X_train[:, 1], c=ys_train)
```

Out[5]:

<matplotlib.collections.PathCollection at 0x21ee4056508>



- ・ 学習

In [6]:

```

t = np.where(ys_train == 1.0, 1.0, -1.0)

n_samples = len(X_train)
# 線形カーネル
K = X_train.dot(X_train.T)

eta1 = 0.01
eta2 = 0.001
n_iter = 500

H = np.outer(t, t) * K

a = np.ones(n_samples)
for _ in range(n_iter):
    grad = 1 - H.dot(a)
    a += eta1 * grad
    a -= eta2 * a.dot(t) * t
    a = np.where(a > 0, a, 0)

```

• 予測

In [7]:

```

index = a > 1e-6
support_vectors = X_train[index]
support_vector_t = t[index]
support_vector_a = a[index]

term2 = K[index][:, index].dot(support_vector_a * support_vector_t)
b = (support_vector_t - term2).mean()

```

In [8]:

```

xx0, xx1 = np.meshgrid(np.linspace(-5, 5, 100), np.linspace(-5, 5, 100))
xx = np.array([xx0, xx1]).reshape(2, -1).T

X_test = xx
y_project = np.ones(len(X_test)) * b
for i in range(len(X_test)):
    for a, sv_t, sv in zip(support_vector_a, support_vector_t, support_vectors):
        y_project[i] += a * sv_t * sv.dot(X_test[i])
y_pred = np.sign(y_project)

```

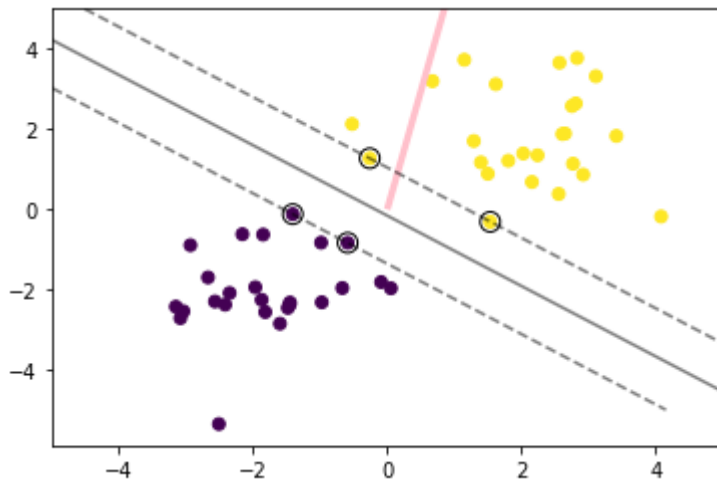
In [9]:

```
# 訓練データを可視化
plt.scatter(X_train[:, 0], X_train[:, 1], c=ys_train)
# サポートベクトルを可視化
plt.scatter(support_vectors[:, 0], support_vectors[:, 1],
            s=100, facecolors='none', edgecolors='k')
# 領域を可視化
# plt.contourf(xx0, xx1, y_pred.reshape(100, 100), alpha=0.2, levels=np.linspace(0, 1, 3))
# マージンと決定境界を可視化
plt.contour(xx0, xx1, y_project.reshape(100, 100), colors='k',
            levels=[-1, 0, 1], alpha=0.5, linestyles=['--', '-', '--'])

# マージンと決定境界を可視化
plt.quiver(0, 0, 0.1, 0.35, width=0.01, scale=1, color='pink')
```

Out[9]:

<matplotlib.quiver.Quiver at 0x21ee4815ec8>



=====

2. サポートベクターマシン(SVM) → 線形分離不可能な場合

- ・ 訓練データの生成② (線形分離不可能)

In [10]:

```
factor = .2
n_samples = 50
linspace = np.linspace(0, 2 * np.pi, n_samples // 2 + 1)[:~1]
outer_circ_x = np.cos(linspace)
outer_circ_y = np.sin(linspace)
inner_circ_x = outer_circ_x * factor
inner_circ_y = outer_circ_y * factor

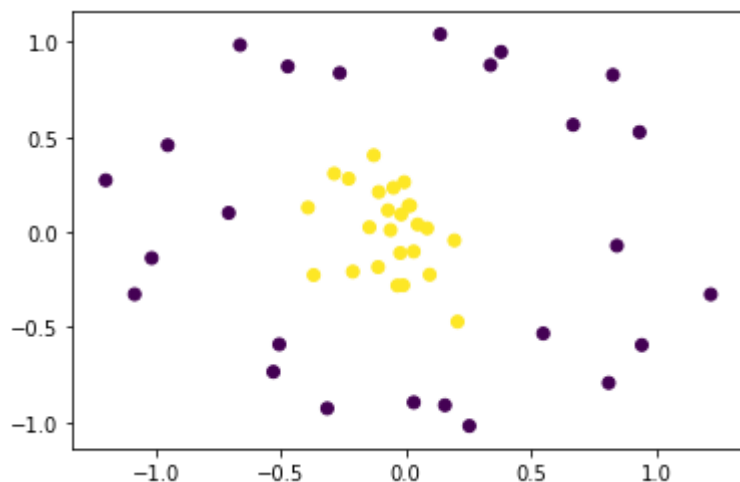
X = np.vstack((np.append(outer_circ_x, inner_circ_x),
                    np.append(outer_circ_y, inner_circ_y))).T
y = np.hstack([np.zeros(n_samples // 2, dtype=np.intp),
               np.ones(n_samples // 2, dtype=np.intp)])
X += np.random.normal(scale=0.15, size=X.shape)
x_train = X
y_train = y
```

In [11]:

```
plt.scatter(x_train[:,0], x_train[:,1], c=y_train)
```

Out[11]:

<matplotlib.collections.PathCollection at 0x21ee4893ac8>



・学習

In [13]:

```
def rbf(u, v):
    sigma = 0.8
    return np.exp(-0.5 * ((u - v)**2).sum() / sigma**2)
```

```
X_train = x_train
t = np.where(y_train == 1.0, 1.0, -1.0)
```

```
n_samples = len(X_train)
# RBFカーネル
K = np.zeros((n_samples, n_samples))
for i in range(n_samples):
    for j in range(n_samples):
        K[i, j] = rbf(X_train[i], X_train[j])
```

```
eta1 = 0.01
eta2 = 0.001
n_iter = 5000
```

```
H = np.outer(t, t) * K
```

```
a = np.ones(n_samples)
for _ in range(n_iter):
    grad = 1 - H.dot(a)
    a += eta1 * grad
    a -= eta2 * a.dot(t) * t
    a = np.where(a > 0, a, 0)
```

・予測

In [14]:

```

index = a > 1e-6
support_vectors = X_train[index]
support_vector_t = t[index]
support_vector_a = a[index]

term2 = K[index][:, index].dot(support_vector_a * support_vector_t)
b = (support_vector_t - term2).mean()

```

In [15]:

```

xx0, xx1 = np.meshgrid(np.linspace(-1.5, 1.5, 100), np.linspace(-1.5, 1.5, 100))
xx = np.array([xx0, xx1]).reshape(2, -1).T

X_test = xx
y_project = np.ones(len(X_test)) * b
for i in range(len(X_test)):
    for a, sv_t, sv in zip(support_vector_a, support_vector_t, support_vectors):
        y_project[i] += a * sv_t * rbf(X_test[i], sv)
y_pred = np.sign(y_project)

```

In [16]:

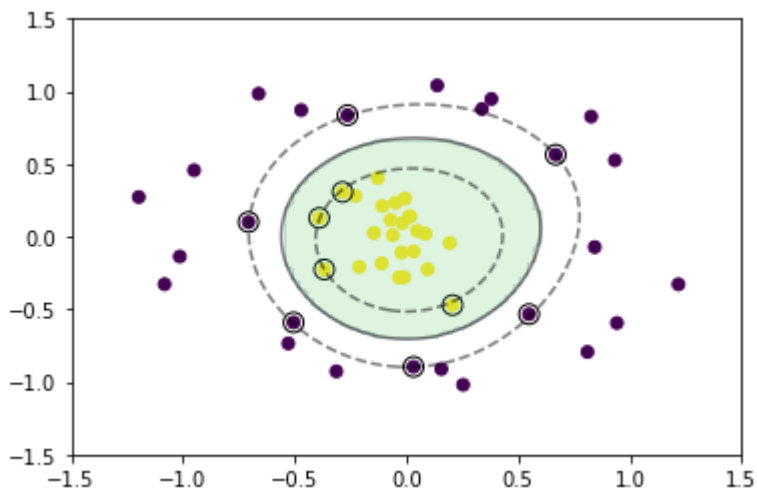
```

# 訓練データを可視化
plt.scatter(x_train[:, 0], x_train[:, 1], c=y_train)
# サポートベクトルを可視化
plt.scatter(support_vectors[:, 0], support_vectors[:, 1],
            s=100, facecolors='none', edgecolors='k')
# 領域を可視化
plt.contourf(xx0, xx1, y_pred.reshape(100, 100), alpha=0.2, levels=np.linspace(0, 1, 3))
# マージンと決定境界を可視化
plt.contour(xx0, xx1, y_project.reshape(100, 100), colors='k',
            levels=[-1, 0, 1], alpha=0.5, linestyles=['--', '-', '--'])

```

Out[16]:

<matplotlib.contour.QuadContourSet at 0x21ee490df88>



=====

3. ソフトマージンSVM

・訓練データの生成③（重なりあり）

In [17]:

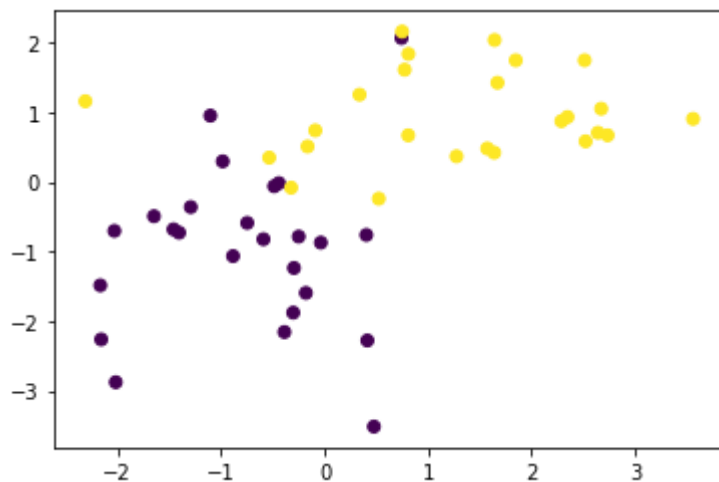
```
x0 = np.random.normal(size=50).reshape(-1, 2) - 1.
x1 = np.random.normal(size=50).reshape(-1, 2) + 1.
x_train = np.concatenate([x0, x1])
y_train = np.concatenate([np.zeros(25), np.ones(25)]).astype(np.int)
```

In [18]:

```
plt.scatter(x_train[:, 0], x_train[:, 1], c=y_train)
```

Out[18]:

<matplotlib.collections.PathCollection at 0x21ee496e788>



・学習

In [19]:

```
X_train = x_train
t = np.where(y_train == 1.0, 1.0, -1.0)

n_samples = len(X_train)
# 線形カーネル
K = X_train.dot(X_train.T)

C = 1
eta1 = 0.01
eta2 = 0.001
n_iter = 1000

H = np.outer(t, t) * K

a = np.ones(n_samples)
for _ in range(n_iter):
    grad = 1 - H.dot(a)
    a += eta1 * grad
    a -= eta2 * a.dot(t) * t
    a = np.clip(a, 0, C)
```

・予測

In [20]:

```

index = a > 1e-8
support_vectors = X_train[index]
support_vector_t = t[index]
support_vector_a = a[index]

term2 = K[index][:, index].dot(support_vector_a * support_vector_t)
b = (support_vector_t - term2).mean()

```

In [21]:

```

xx0, xx1 = np.meshgrid(np.linspace(-4, 4, 100), np.linspace(-4, 4, 100))
xx = np.array([xx0, xx1]).reshape(2, -1).T

X_test = xx
y_project = np.ones(len(X_test)) * b
for i in range(len(X_test)):
    for a, sv_t, sv in zip(support_vector_a, support_vector_t, support_vectors):
        y_project[i] += a * sv_t * sv.dot(X_test[i])
y_pred = np.sign(y_project)

```

In [22]:

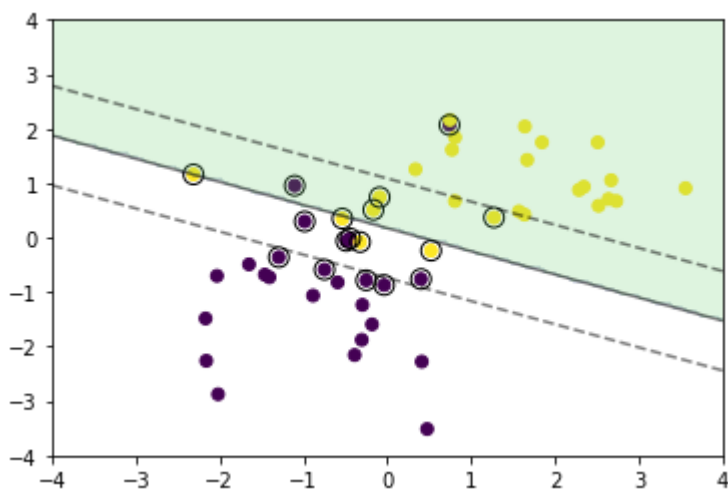
```

# 訓練データを可視化
plt.scatter(x_train[:, 0], x_train[:, 1], c=y_train)
# サポートベクトルを可視化
plt.scatter(support_vectors[:, 0], support_vectors[:, 1],
            s=100, facecolors='none', edgecolors='k')
# 領域を可視化
plt.contourf(xx0, xx1, y_pred.reshape(100, 100), alpha=0.2, levels=np.linspace(0, 1, 3))
# マージンと決定境界を可視化
plt.contour(xx0, xx1, y_project.reshape(100, 100), colors='k',
            levels=[-1, 0, 1], alpha=0.5, linestyles=['--', '-', '--'])

```

Out[22]:

<matplotlib.contour.QuadContourSet at 0x21ee49def08>



=====

・サポートベクターマシーン →教師あり学習で、分類と回帰を扱うことができるが、主に分類のタスクで使われる。

- ・データを分割する直線に最も近いデータ点(サポートベクター)を定め、分割線(決定境界)を導く。 決定境界のどちら側にデータが存在するかより、どのクラスに属しているかの予測ができる。
- ・線形分離可能な場合(①)、線形分離不可能な場合(②)、データに重なりがある場合(③)の3パターンの2値分類を実施。
- ・①は、特徴空間上で線形なモデルを用い、決定境界を導く。
- ・②は、元のデータ空間上では線形分離できないが、特徴空間上で線形分離を実施。
- ・③の場合、分離不可能な場合のため学習できないが、データ点がマージン内部に入ることや誤分類を許容することでその問題を回避。

In []: