+ コード + テキスト

RAM ディスク

サポートベクターマシン(SVM)

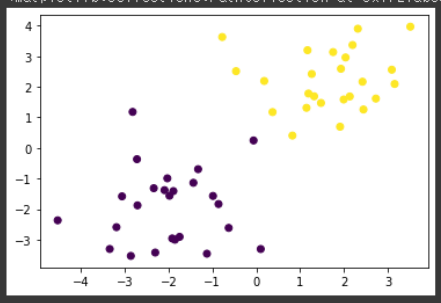
```
[1] 1 %matplotlib inline
    2 import numpy as np
    3 import matplotlib.pyplot as plt
```

訓練データ生成① （線形分離可能）

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

```
[3] 1 X_train, ys_train = gen_data()
    2 plt.scatter(X_train[:, 0], X_train[:, 1], c=ys_train)
```

```
<matplotlib.collections.PathCollection at 0x7f21ab697fd0>
```



学習

特徴空間上で線形なモデル$y(\boldsymbol{x}) = \boldsymbol{w}\phi(\boldsymbol{x}) + b$を用い、その正負によって2値分類を行うことを考える。

サポートベクターマシンではマージンの最大化を行うが、それは結局以下の最適化問題を解くことと同じである。

ただし、訓練データを$\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n]^{\mathrm{T}}, \boldsymbol{t} = [t_1, t_2, \ldots, t_n]^{\mathrm{T}} (t_i = \{-1, +1\})$とする。

$$\min_{\boldsymbol{w}, b} \quad \frac{1}{2}||\boldsymbol{w}||^2$$

$$\text{subject to} \quad t_i(\boldsymbol{w}\phi(\boldsymbol{x}_i) + b) \geq 1 \quad (i = 1, 2, \cdots, n)$$

ラグランジュ乗数法を使うと、上の最適化問題はラグランジュ乗数$\boldsymbol{a}(\geq 0)$を用いて、以下の目的関数を最小化する問題となる。

$$L(\boldsymbol{w}, b, \boldsymbol{a}) = \frac{1}{2}||\boldsymbol{w}||^2 - \sum_{i=1}^{n} a_i t_i (\boldsymbol{w}\phi(\boldsymbol{x}_i) + b - 1) \qquad \cdots (1)$$

目的関数が最小となるのは、$\boldsymbol{w}, b$に関して偏微分した値が0となるときなので、

$$\frac{\partial L}{\partial \boldsymbol{w}} = \boldsymbol{w} - \sum_{i=1}^{n} a_i t_i \phi(\boldsymbol{x}_i) = \boldsymbol{0}$$

$$\frac{\partial L}{\partial b} = \sum_{i=1}^{n} a_i t_i = \boldsymbol{a}^{\mathrm{T}}\boldsymbol{t} = 0$$

これを式(1)　に代入することで、最適化問題は結局以下の目的関数の最大化となる。

+ コード + テキスト

RAM ディスク

$$\tilde{L}(\boldsymbol{a}) = \sum_{i=1}^{n} a_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} a_i a_j t_i t_j \phi(\boldsymbol{x}_i)^{\mathrm{T}}\phi(\boldsymbol{x}_j)$$

$$= \boldsymbol{a}^{\mathrm{T}}\boldsymbol{1} - \frac{1}{2}\boldsymbol{a}^{\mathrm{T}}H\boldsymbol{a}$$

ただし、行列Hのi行j列成分は$H_{ij} = t_i t_j \phi(\boldsymbol{x}_i)^{\mathrm{T}}\phi(\boldsymbol{x}_j) = t_i t_j k(\boldsymbol{x}_i, \boldsymbol{x}_j)$である。また制約条件は、$\boldsymbol{a}^{\mathrm{T}}\boldsymbol{t} = 0 (\frac{1}{2}||\boldsymbol{a}^{\mathrm{T}}\boldsymbol{t}||^2 = 0)$である。

この最適化問題を最急降下法で解く。目的関数と制約条件を$\boldsymbol{a}$で微分すると、

$$\frac{d\tilde{L}}{d\boldsymbol{a}} = \boldsymbol{1} - H\boldsymbol{a}$$

$$\frac{d}{d\boldsymbol{a}}(\frac{1}{2}||\boldsymbol{a}^{\mathrm{T}}\boldsymbol{t}||^2) = (\boldsymbol{a}^{\mathrm{T}}\boldsymbol{t})\boldsymbol{t}$$

なので、$\boldsymbol{a}$を以下の二式で更新する。

$$\boldsymbol{a} \leftarrow \boldsymbol{a} + \eta_1(\boldsymbol{1} - H\boldsymbol{a})$$

$$\boldsymbol{a} \leftarrow \boldsymbol{a} - \eta_2(\boldsymbol{a}^{\mathrm{T}}\boldsymbol{t})\boldsymbol{t}$$

```
[4]  1 t = np.where(ys_train == 1.0, 1.0, -1.0)
     2
     3 n_samples = len(X_train)
     4 # 線形カーネル
     5 K = X_train.dot(X_train.T)
     6
     7 eta1 = 0.01
     8 eta2 = 0.001
     9 n_iter = 500
    10
    11 H = np.outer(t, t) * K
    12
    13 a = np.ones(n_samples)
    14 for _ in range(n_iter):
    15     grad = 1 - H.dot(a)
    16     a += eta1 * grad
    17     a -= eta2 * a.dot(t) * t
    18     a = np.where(a > 0, a, 0)
```

予測

新しいデータ点$\boldsymbol{x}$に対しては、$y(\boldsymbol{x}) = \boldsymbol{w}\phi(\boldsymbol{x}) + b = \sum_{i=1}^{n} a_i t_i k(\boldsymbol{x}, \boldsymbol{x}_i) + b$の正負によって分類する。

ここで、最適化の結果得られた$a_i (i = 1, 2, \ldots, n)$の中で$a_i = 0$に対応するデータ点は予測に影響を与えないので、$a_i > 0$に対応するデータ点（サポートベクトル）のみ保持しておく。bはサポートベクトルのインデックスの集合をSとすると、$b = \frac{1}{S} \sum_{s \in S} \left(t_s - \sum_{i=1}^{n} a_i t_i k(\boldsymbol{x}_i, \boldsymbol{x}_s)\right)$によって求める。

```
[5]  1 index = a > 1e-6
     2 support_vectors = X_train[index]
     3 support_vector_t = t[index]
     4 support_vector_a = a[index]
     5
     6 term2 = K[index][:, index].dot(support_vector_a * support_vector_t)
     7 b = (support_vector_t - term2).mean()
```

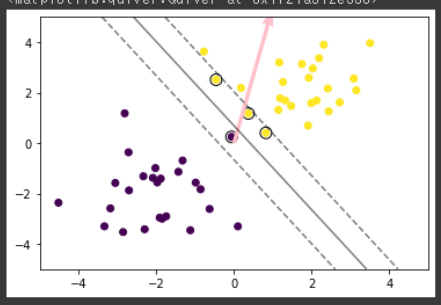
```
[6]  1 xx0, xx1 = np.meshgrid(np.linspace(-5, 5, 100), np.linspace(-5, 5, 100))
     2 xx = np.array([xx0, xx1]).reshape(2, -1).T
```



```
[6]  3
     4 X_test = xx
     5 y_project = np.ones(len(X_test)) * b
     6 for i in range(len(X_test)):
     7     for a, sv_t, sv in zip(support_vector_a, support_vector_t, support_vectors):
     8         y_project[i] += a * sv_t * sv.dot(X_test[i])
     9 y_pred = np.sign(y_project)
```

```
[7]  1 # 訓練データを可視化
     2 plt.scatter(X_train[:, 0], X_train[:, 1], c=ys_train)
     3 # サポートベクトルを可視化
     4 plt.scatter(support_vectors[:, 0], support_vectors[:, 1],
     5                     s=100, facecolors='none', edgecolors='k')
     6 # 領域を可視化
     7 #plt.contourf(xx0, xx1, y_pred.reshape(100, 100), alpha=0.2, levels=np.linspace(0, 1, 3))
     8 # マージンと決定境界を可視化
     9 plt.contour(xx0, xx1, y_project.reshape(100, 100), colors='k',
    10                      levels=[-1, 0, 1], alpha=0.5, linestyles=['--', '-', '--'])
    11
    12
    13 # マージンと決定境界を可視化
    14 plt.quiver(0, 0, 0.1, 0.35, width=0.01, scale=1, color='pink')
```

```
<matplotlib.quiver.Quiver at 0x7f21a312e590>
```



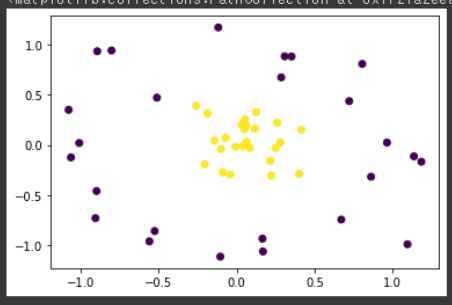
訓練データ生成② （線形分離不可能）

```
[8]  1 factor = .2
     2 n_samples = 50
     3 linspace = np.linspace(0, 2 * np.pi, n_samples // 2 + 1)[:-1]
     4 outer_circ_x = np.cos(linspace)
     5 outer_circ_y = np.sin(linspace)
     6 inner_circ_x = outer_circ_x * factor
     7 inner_circ_y = outer_circ_y * factor
     8
     9 X = np.vstack((np.append(outer_circ_x, inner_circ_x),
    10                np.append(outer_circ_y, inner_circ_y))).T
    11 y = np.hstack([np.zeros(n_samples // 2, dtype=np.intp),
    12                np.ones(n_samples // 2, dtype=np.intp)])
    13 X += np.random.normal(scale=0.15, size=X.shape)
    14 x_train = X
    15 y_train = y
```

+ コード + テキスト

RAM ディスク

```
[9]  <matplotlib.collections.PathCollection at 0x7f21a2eea210>
```



学習

元のデータ空間では線形分離は出来ないが、特徴空間上で線形分離することを考える。

今回はカーネルとしてRBFカーネル（ガウシアンカーネル）を利用する。

```
[10]  1 def rbf(u, v):
      2         sigma = 0.8
      3         return np.exp(-0.5 * ((u - v)**2).sum() / sigma**2)
      4
      5 X_train = x_train
      6 t = np.where(y_train == 1.0, 1.0, -1.0)
      7
      8 n_samples = len(X_train)
      9 # RBFカーネル
     10 K = np.zeros((n_samples, n_samples))
     11 for i in range(n_samples):
     12     for j in range(n_samples):
     13         K[i, j] = rbf(X_train[i], X_train[j])
     14
     15 eta1 = 0.01
     16 eta2 = 0.001
     17 n_iter = 5000
     18
     19 H = np.outer(t, t) * K
     20
     21 a = np.ones(n_samples)
     22 for _ in range(n_iter):
     23     grad = 1 - H.dot(a)
     24     a += eta1 * grad
     25     a -= eta2 * a.dot(t) * t
     26     a = np.where(a > 0, a, 0)
```

予測

```
[11]  1 index = a > 1e-6
      2 support_vectors = X_train[index]
      3 support_vector_t = t[index]
      4 support_vector_a = a[index]
      5
      6 term2 = K[index][:, index].dot(support_vector_a * support_vector_t)
      7 b = (support_vector_t - term2).mean()
```

```
[12]  1 xx0, xx1 = np.meshgrid(np.linspace(-1.5, 1.5, 100), np.linspace(-1.5, 1.5, 100))
```

+ コード + テキスト

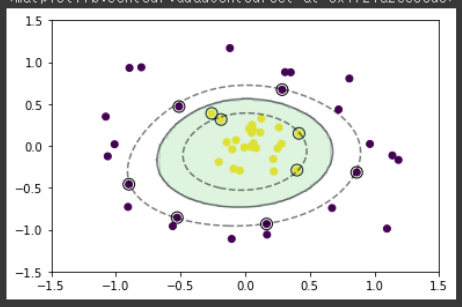
RAM

```
[12]  4 X_test = xx
      5 y_project = np.ones(len(X_test)) * b
      6 for i in range(len(X_test)):
      7     for a, sv_t, sv in zip(support_vector_a, support_vector_t, support_vectors):
      8         y_project[i] += a * sv_t * rbf(X_test[i], sv)
      9 y_pred = np.sign(y_project)
```

＋ コード　＋ テキスト

```
[13]  1 # 訓練データを可視化
      2 plt.scatter(x_train[:, 0], x_train[:, 1], c=y_train)
      3 # サポートベクトルを可視化
      4 plt.scatter(support_vectors[:, 0], support_vectors[:, 1],
      5                    s=100, facecolors='none', edgecolors='k')
      6 # 領域を可視化
      7 plt.contourf(xx0, xx1, y_pred.reshape(100, 100), alpha=0.2, levels=np.linspace(0, 1, 3))
      8 # マージンと決定境界を可視化
      9 plt.contour(xx0, xx1, y_project.reshape(100, 100), colors='k',
     10                     levels=[-1, 0, 1], alpha=0.5, linestyles=['--', '-', '--'])
```

```
<matplotlib.contour.QuadContourSet at 0x7f21a2ec30d0>
```



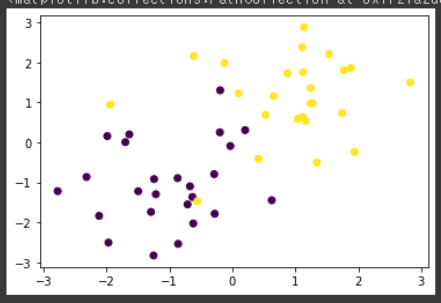
ソフトマージンSVM

訓練データ生成③（重なりあり）

```
[14]  1 x0 = np.random.normal(size=50).reshape(-1, 2) - 1.
      2 x1 = np.random.normal(size=50).reshape(-1, 2) + 1.
      3 x_train = np.concatenate([x0, x1])
      4 y_train = np.concatenate([np.zeros(25), np.ones(25)]).astype(np.int)
```

```
[15]  1 plt.scatter(x_train[:, 0], x_train[:, 1], c=y_train)
```

```
<matplotlib.collections.PathCollection at 0x7f21a2ddb8d0>
```



学習

0 秒　完了時間: 22:17

学習

分離不可能な場合は学習できないが、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する。

スラック変数$\xi_i \geq 0$を導入し、マージン内部に入ったり誤分類された点に対しては、$\xi_i = |1 - t_i y(\boldsymbol{x}_i)|$とし、これらを許容する代わりに対して、ペナルティを与えるように、最適化問題を以下のように修正する。

$$\min_{w,b} \quad \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_{i=1}^{n}\xi_i$$

$$\text{subject to} \quad t_i(\boldsymbol{w}\phi(\boldsymbol{x}_i) + b) \geq 1 - \xi_i \quad (i = 1, 2, \cdots, n)$$

ただし、パラメータCはマージンの大きさと誤差の許容度のトレードオフを決めるパラメータである。この最適化問題をラグランジュ乗数法などを用いると、結局最大化する目的関数はハードマージンSVMと同じとなる。

$$\tilde{L}(\boldsymbol{a}) = \sum_{i=1}^{n} a_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} a_i a_j t_i t_j \phi(\boldsymbol{x}_i)^{\mathrm{T}}\phi(\boldsymbol{x}_j)$$

ただし、制約条件が$a_i \geq 0$の代わりに$0 \leq a_i \leq C (i = 1, 2, \ldots, n)$となる。（ハードマージンSVMと同じ$\sum_{i=1}^{n} a_i t_i = 0$も制約条件）

```
[16] 1 X_train = x_train
     2 t = np.where(y_train == 1.0, 1.0, -1.0)
     3
     4 n_samples = len(X_train)
     5 # 線形カーネル
     6 K = X_train.dot(X_train.T)
     7
     8 C = 1
     9 eta1 = 0.01
    10 eta2 = 0.001
    11 n_iter = 1000
    12
    13 H = np.outer(t, t) * K
    14
    15 a = np.ones(n_samples)
    16 for _ in range(n_iter):
    17     grad = 1 - H.dot(a)
    18     a += eta1 * grad
```



```
[16] 19     a -= eta2 * a.dot(t) * t
     20     a = np.clip(a, 0, C)
```

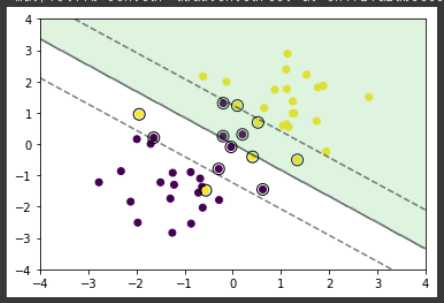
予測


```
[17] 1 index = a > 1e-8
     2 support_vectors = X_train[index]
     3 support_vector_t = t[index]
     4 support_vector_a = a[index]
     5
     6 term2 = K[index][:, index].dot(support_vector_a * support_vector_t)
     7 b = (support_vector_t - term2).mean()
```

```
[18] 1 xx0, xx1 = np.meshgrid(np.linspace(-4, 4, 100), np.linspace(-4, 4, 100))
     2 xx = np.array([xx0, xx1]).reshape(2, -1).T
     3
     4 X_test = xx
     5 y_project = np.ones(len(X_test)) * b
     6 for i in range(len(X_test)):
     7     for a, sv_t, sv in zip(support_vector_a, support_vector_t, support_vectors):
     8         y_project[i] += a * sv_t * sv.dot(X_test[i])
     9 y_pred = np.sign(y_project)
```

```
[19] 1 # 訓練データを可視化
     2 plt.scatter(x_train[:, 0], x_train[:, 1], c=y_train)
     3 # サポートベクトルを可視化
     4 plt.scatter(support_vectors[:, 0], support_vectors[:, 1],
     5                     s=100, facecolors='none', edgecolors='k')
     6 # 領域を可視化
     7 plt.contourf(xx0, xx1, y_pred.reshape(100, 100), alpha=0.2, levels=np.linspace(0, 1, 3))
     8 # マージンと決定境界を可視化
     9 plt.contour(xx0, xx1, y_project.reshape(100, 100), colors='k',
    10                     levels=[-1, 0, 1], alpha=0.5, linestyles=['--', '-', '--'])
```

```
<matplotlib.contour.QuadContourSet at 0x7f21a2db9e90>
```




```
[19] 1
```


0 秒
完了時間: 22:17