先端データ解析論
第六回　レポート

37-196360　森田涼介

2019 年 5 月 21 日

宿題 1

サポートベクトルマシンの双対最適化問題の相補性条件は次のようになる。

$$\alpha_i \left(y_i \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_i - 1 + \xi_i \right) = 0 \tag{1}$$
$$\beta_i \xi_i = 0 \tag{2}$$

ここで，ソフトマージンを採用すると次式が成り立つ。

$$\begin{aligned} & y_i \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_i \geq 1 - \xi_i \\ & \Leftrightarrow y_i \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_i - 1 + \xi_i \geq 0 \end{aligned} \tag{3}$$
$$\xi_i \geq 0 \tag{4}$$

また，サポートベクトルマシンのラグランジュ関数の ξ_i による微分が 0 となることから，

$$\alpha_i + \beta_i = C \tag{5}$$

である。

以下では，α_i と $y_i \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_i$ の値それぞれについて場合分けをし，各場合でのもう一方の値（の範囲）を求める。

1. $\alpha_i = 0$ のとき

式 (5) より，

$$\beta_i = C$$

式 (2) より，

$$\xi_i = 0$$

式 (3) より，

$$y_i \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_i \geq 1 \tag{6}$$

2. $0 < \alpha_i < C$ のとき

式 (5) より，

$$\beta_i \neq 0$$

式 (2) より，

$$\xi_i = 0$$

式 (1) より，$\alpha_i \neq 0$ に注意して，

$$y_i \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_i = 1 \tag{7}$$

3. $\alpha_i = C$ のとき

式 (5) より，

$$\beta_i = 0$$

これと式 (2) より，$\xi_i = 0$ である必要はないことが確認される。式 (1) 及び式 (4) より，$\alpha_i \neq 0$ に注意して，

$$y_i \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_i = 1 - \xi_i \leq 1 \tag{8}$$

4. $y_i \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_i > 1$ のとき

式 (4) より，

$$y_i \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_i > 1 \geq 1 - \xi_i$$
$$y_i \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_i - 1 + \xi_i > 0$$

式 (1) より，

$$\alpha_i = 0 \tag{9}$$

5. $y_i \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_i < 1$ のとき

式 (3) より，

$$\xi_i \geq 1 - y_i \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_i > 0$$
$$\xi_i > 0$$

式 (2) より，

$$\beta_i = 0$$

式 (5) より，

$$\alpha_i = C \tag{10}$$

宿題 2

線形モデル

$$f_{\boldsymbol{w},b}(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b \tag{11}$$

に対するサポートベクトルマシンの劣勾配アルゴリズムを実装する。

理論

ソフトマージンを用いることとすれば，損失関数は次のようになる。

$$L = ||\boldsymbol{w}||^2 + C\sum_{i=1}^{n} \max(0,\ 1 - y_i f_{\boldsymbol{w},b}(\boldsymbol{x}_i)) \tag{12}$$

この損失関数の，パラメータ $\boldsymbol{w}$, b による偏微分を考える。第一項については，

$$\frac{\partial}{\partial \boldsymbol{w}}\left(||\boldsymbol{w}||^2\right) = 2\boldsymbol{w} \tag{13}$$

$$\frac{\partial}{\partial b}\left(||\boldsymbol{w}||^2\right) = 0 \tag{14}$$

となることが容易に分かる。第二項について考える。いま，

$$z_i = y_i f_{\boldsymbol{w},b}(\boldsymbol{x}_i) = y_i\left(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b\right) \tag{15}$$

とおくと，

$$\frac{\partial z_i}{\partial \boldsymbol{w}} = y_i \boldsymbol{x}_i \tag{16}$$

$$\frac{\partial z_i}{\partial b} = y_i \tag{17}$$

である。いま，微分したい関数 $\max(0,\ 1 - z_i)$ は $z_i = 1$ で微分不可能なので，劣微分を考える。J の θ による劣微分を $\partial_\theta J$ と表すこととすると，

$$\partial_{w_j} \max(0,\ 1 - z_i) = \begin{cases} -\partial z_i \big/ \partial w_j = -y_i x_{i,j} & (z < 1) \\ \left[-\partial z_i \big/ \partial w_j\ ,\ 0\right] = [-y_i x_{i,j},\ 0] & (z = 1) \\ 0 & (z > 1) \end{cases} \tag{18}$$

$$\partial_b \max(0,\ 1 - z_i) = \begin{cases} -\partial z_i \big/ \partial b = -y_i & (z < 1) \\ \left[-\partial z_i \big/ \partial b\ ,\ 0\right] = [-y_i,\ 0] & (z = 1) \\ 0 & (z > 1) \end{cases} \tag{19}$$

となる。以上より，損失関数の偏微分は次のようになる。

$$\frac{\partial L}{\partial w_j} = 2w_j + C\partial_{w_j} \max(0,\ 1 - z_i) \tag{20}$$

$$\frac{\partial L}{\partial w_j} = C\partial_b \max(0,\ 1 - z_i) \tag{21}$$

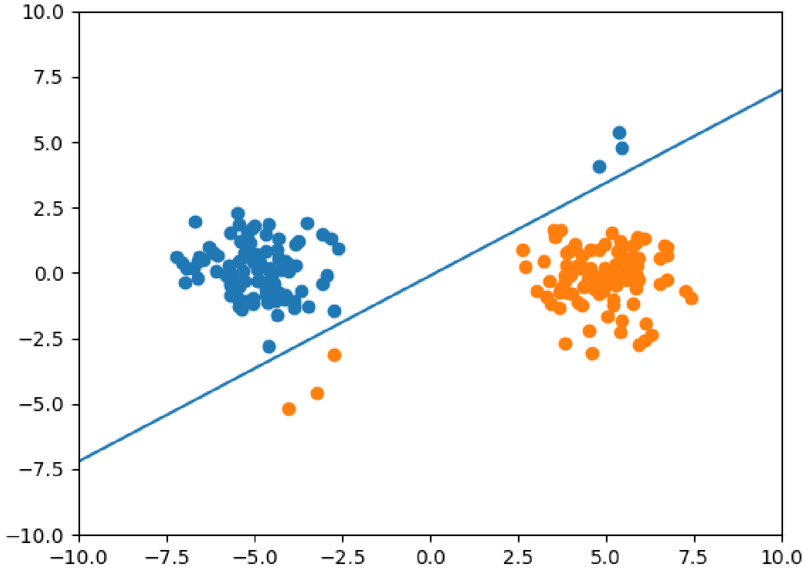

図 1: 境界及び点群

結果

サンプル数 200 の二次元点群について，式 (11) の線形モデルに対するサポートベクトルマシンを用いて境界を求めた。パラメータの更新には劣勾配アルゴリズムを用いた。ハイパーパラメータは，$C = 0.1$，学習率 0.05 とした。終了条件は，パラメータの前の更新との差分のノルムが 1×10^{-4} 以下，または 10,000 回更新を行ったときとした。なお，プログラムは 5 ページの Listing 1 に示した。

学習の結果，ハイパーパラメータは次のようになった。

$$\boldsymbol{w} = \begin{bmatrix} -0.3960 \\ 0.5339 \end{bmatrix} \tag{22}$$

$$b = 0.02218 \tag{23}$$

また，このとき求まった境界及び点群を図 1 に示した。図をみると，他のカテゴリの方に近くなってしまっている小数のサンプルについてもうまく分類できていることがわかる。

プログラム

実行環境と用いた言語・ライブラリを以下の表 1 に示す。

表 1: プログラムの実行環境

OS	:	Microsoft Windows 10 Pro (64bit)
CPU	:	Intel(R) Core(TM) i5-4300U
RAM	:	4.00 GB
使用言語	:	Python3.6
可視化	:	matplotlib ライブラリ

Listings 1: `assignment2.py`

```
# -*- coding: utf-8 -*-


import numpy as np
import matplotlib
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt


def generate_data(sample_size):
    """Generate training data.

    Since
    f(x) = w^{T}x + b
    can be written as
    f(x) = (w^{T}, b)(x^{T}, 1)^{T},
    for the sake of simpler implementation of SVM,
    we return (x^{T}, 1)^{T} instead of x

    :param sample_size: number of data points in the sample
    :return: a tuple of data point and label
    """

    x = np.random.normal(size=(sample_size, 3))
    x[:, 2] = 1.
    x[:sample_size // 2, 0] -= 5.
    x[sample_size // 2:, 0] += 5.
    y = np.concatenate([np.ones(sample_size // 2, dtype=np.int64),
                        -np.ones(sample_size // 2, dtype=np.int64)])
    x[:3, 1] -= 5.
    y[:3] = -1
    x[-3:, 1] += 5.
```

```
    y[-3:] = 1
    return x, y


def calc_subgrad(x, y, w):
    f = x.dot(w)
    z = y * f
    yx = y[:, np.newaxis] * x

    indices_over_1 = (z > 1)
    indices_equals_1 = (z == 1)
    indices_under_1 = (z < 1)

    subgrads = np.zeros_like(x)
    subgrads[indices_over_1] = 0
    subgrads[indices_under_1] = - yx[indices_under_1]
    subgrads[indices_equals_1] = 0

    subgrad = subgrads.sum(axis=0)
    return subgrad


def calc_grad(x, y, w, c):
    subgrad = calc_subgrad(x, y, w)
    grad_w = 2*w
    grad_w[2] = 0
    grad = grad_w + c*subgrad
    return grad


def update(x, y, w, c, lr):
    grad = calc_grad(x, y, w, c)
    w_new = w - lr * grad
    return w_new


def svm(x, y, c, lr, max_iter=1e4, eps=1e-3):
    """Linear SVM implementation using gradient descent algorithm.

    f_w(x) = w^{T} (x^{T}, 1)^{T}

    :param x: data points
    :param y: label
    :param l: regularization parameter
    :param lr: learning rate
    :return: three-dimensional vector w
    """
```

```
    d = x.shape[1]
    w = np.zeros(d)
    prev_w = w.copy()
    for i in range(int(max_iter)):
        w = update(x, y, w, c, lr)

        # convergence condition
        if np.linalg.norm(w - prev_w) < eps:
            break
        prev_w = w.copy()
    n_iter = i + 1
    return w, n_iter


def visualize(x, y, w, path=None):
    plt.clf()
    plt.xlim(-10, 10)
    plt.ylim(-10, 10)
    plt.scatter(x[y == 1, 0], x[y == 1, 1])
    plt.scatter(x[y == -1, 0], x[y == -1, 1])
    plt.plot([-10, 10], -(w[2] + np.array([-10, 10]) * w[0]) / w[1])
    if path:
        plt.savefig(path)
    plt.show()


def main():
    # settings
    n_sample = 200
    fig_path = '../figures/assignment2_result.png'
    np.random.seed(0)

    # load data
    x, y = generate_data(n_sample)

    # train
    w, n_iter = svm(x, y, c=.1, lr=0.05, max_iter=1e4, eps=1e-4)

    # result
    print(f'#Sample: {n_sample}')
    print(f'#Iter: {n_iter}')
    print(f'w: {w}')
    visualize(x, y, w, fig_path)


if __name__ == '__main__':
    main()
```