計算機科学実験及演習 4 エージェント 課題 1

1029-28-2473 二見 颯

2018 年 10 月 12 日

1　プログラム概要

データ点が書かれたファイルを読み込み、SVM の識別器を作成してそのパラメータを出力する。作成した SVM によりクラスの予測を行い、与えられたデータ点と予測された識別境界を平面上にプロットする。

2　外部仕様

- main.py - プログラムのエントリポイント。svc.py の SVClassifier および utils.py の関数を参照している
- svc.py - SVM のクラス SVClassifier を実装
- utils.py - ファイル読み込み (load_data) と結果のプロット (plot_decision_regions) を実装

プログラムの実行は
./main.py [入力ファイルへのパス] [カーネルトリックの種類] [カーネルトリックのパラメータ]
カーネルトリックの種類については、n でカーネルトリックなし、p で多項式カーネル、g でガウスカーネル、s でシグモイドカーネルとする。プログラム引数が正しくない形式の場合は、エラーとしてプログラムが終了する。

```
$ ./main.py data/square.dat n
    pcost dcost gap pres dres
0: -1.2457e-01 -3.5986e-01 2e-01 7e-18 1e+00
1: -1.2497e-01 -1.2743e-01 2e-03 5e-17 3e-02
2: -1.2500e-01 -1.2502e-01 2e-05 2e-17 3e-04
3: -1.2500e-01 -1.2500e-01 2e-07 4e-17 3e-06
4: -1.2500e-01 -1.2500e-01 2e-09 3e-17 3e-08
Optimal solution found.
alpha:
0 [0.0625]
1 [0.0625]
2 [0.0625]
3 [0.0625]
theta candidates: [0.9999999680249498, 1.0, 0.9999999680249498, 1.0]
SVClassifier W = [0.49999999 0. ], θ = 0.9999999840124749
```

```
16 [グラフ出力]
```

必要なライブラリ等は以下

```
1 Python 3.6.1 :: Anaconda custom (64-bit)
2 numpy 1.14.2
3 cvxopt 1.2.1
4 tqdm 4.26.0
5 matplotlib 2.0.2
```

3 内部仕様

SVClassifier クラス (svc.py)

サポートベクタマシンを定義する. private メンバ

- X, y - 訓練データ点とその正解クラス
- n - 訓練データの個数
- kf - kernel trick として用いる関数 (kernel function)
- p, q - kernel function のパラメータ (optional)
- alpha, theta - SVM の内部パラメータであり、それぞれ setLagrange 関数, setClassifier 関数で決定する

コンストラクタにより、kf, p, q を決定する

fit

X, y, n を決定して、alpha, theta を決定する各関数を呼び出す
@param[in] X, y

_setLagrange

以下の 2 次計画問題を cvxopt.solvers.qp を用いて解くことで、alpha を決定する

$$\max\{\sum_{k=0} \alpha_k - \sum_{k=0}\sum_{l=0} \alpha_k y_k \alpha_l y_l K(\boldsymbol{x_k}, \boldsymbol{x_l})/2\} \tag{1}$$

$$(\sum_{k=0} \alpha_k y_k = 0, 0 < \alpha_k) \tag{2}$$

cvxopt.solvers.qp では、以下の 2 次計画問題を解くため、これが以上と等価になるように P, q, G, h, A, b を与える。($\boldsymbol{x}$ を求める alpha とみる)

$$\min\{\boldsymbol{x}^T P \boldsymbol{x}/2 + q^T \boldsymbol{x}\} \tag{3}$$

$$(G\boldsymbol{x} <= h, A\boldsymbol{x} = b) \tag{4}$$

_setClassifier

alpha と x, y から theta を決定して、”W = , θ = ”の形で識別器を出力する
alpha が 0 でないサポートベクタ点 $\boldsymbol{x_i}$ に関して、$\theta_i = \sum_{k=0} \alpha_k y_k K(\boldsymbol{x_k}, \boldsymbol{x_i}) - y_i$ を求める。これらは、i によらず一定であることを表示して確認して、その平均を theta とする。また、$W = \sum_{k=0} \alpha_k y_k \boldsymbol{x_k}$ を theta とともに表示する。

_kernelPolynomial

多項式カーネル $K(\boldsymbol{a}, \boldsymbol{b}) = (1 + (\boldsymbol{a}, \boldsymbol{b}))^p$ を実装する
@param[in] a(batch_size, dim), b(dim) a はバッチ入力に対応している

_kernelGauss

Gauss カーネル $K(\boldsymbol{a}, \boldsymbol{b}) = \exp(-\|\boldsymbol{a} - \boldsymbol{b}\|/2p^2)$ を実装する
@param[in] a(batch_size, dim), b(dim) a はバッチ入力に対応している
numpy.linalg.norm でノルムを計算するときに、行方向か列方向かを指定するため、a がバッチ入力か否かで実装が変わっている。

_kernelSigmoid

シグモイドカーネル $K(\boldsymbol{a}, \boldsymbol{b}) = \tanh(p(\boldsymbol{a}, \boldsymbol{b}) + q)$ を実装する
@param[in] a(batch_size, dim), b(dim) a はバッチ入力に対応している

predict

SVM により tX のクラスの予測をする
@param[in] tX(batch_size * dim) テストデータ
SVM の識別関数は、$f(\boldsymbol{x}) = sign(\sum_{k=0} \alpha_k y_k K(\boldsymbol{x_k}, \boldsymbol{x}) - \theta)$ とする。各 tX[i] について、前計算した ay = $\sum_{k=0} \alpha_k y_k$ と K($\boldsymbol{x}$, tX[i]) の行列積によりこの値を計算する。

load_data(utils.py)

filepath にある sample_linear.dat 同様の形式のデータを X,y に読み込む
@param[in] filepath
@return X, y
入力ファイルのデータ点は、行の最後に正解クラスが指定される以下のような形式で与えられる。

```
1.0, 0.5, 1
0.5, 0.25, -1
```

1 行ごとに”, ”で split して処理するためデータ点が n 次元の特徴で表せる場合にも対応できる。

plot_decision_regions(utils.py)

x を 2 次元平面に plot する. plot された点の色は y によって決まり、領域の色は model の予測結果により決まる
@param[in] x, y, model, resolution resolution によりグリッドの幅を指定する
@return なし
x が 2 次元でない場合、この関数は終了する。
x=(x1, x2) に対する正解クラスを y, SVM による予測クラスを z とする。x1 の最小値-1 から最大値 +1, x2 の最小値-1 から最大値 +1 の範囲でグリッド点を作成して、そのグリッド点に対して model.predict することで z の決定領域が得られる。
その後、各 x の点を y に応じた色 (red, blue) と形 (o, x) で plot する。

4　評価結果

4.1　sample_linear.dat

sample_linear.dat をカーネルなし SVM で学習すると図 1 がプロット結果として得られた。

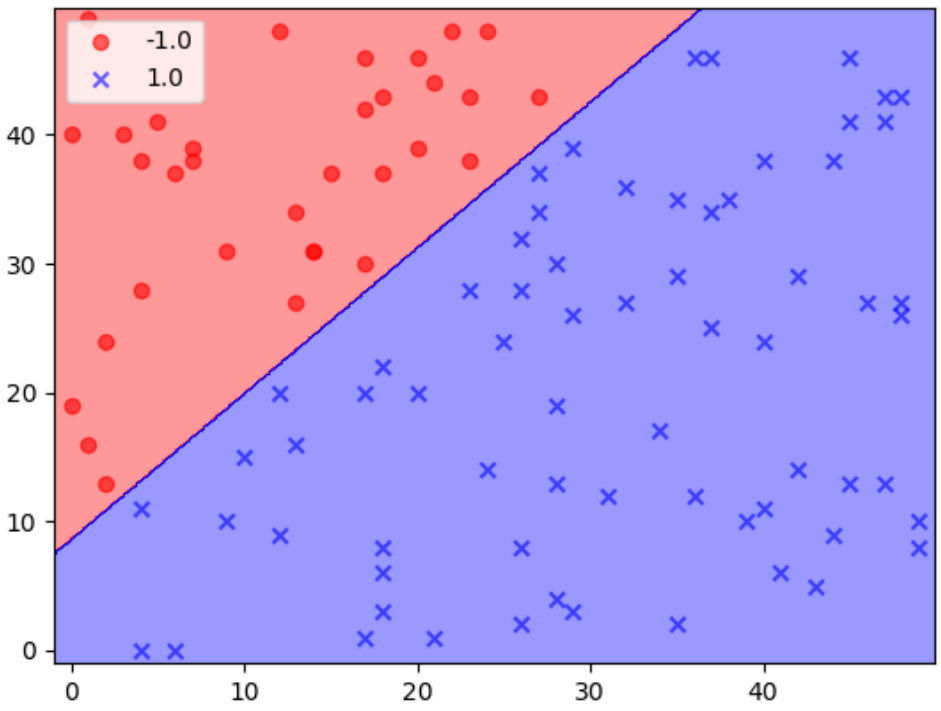

図 1　sample_linear.dat

4.2 sample_circle.dat

sample_circle.dat を Gauss カーネル ($\sigma = 10$)SVM で学習すると図 2 がプロット結果として得られた。

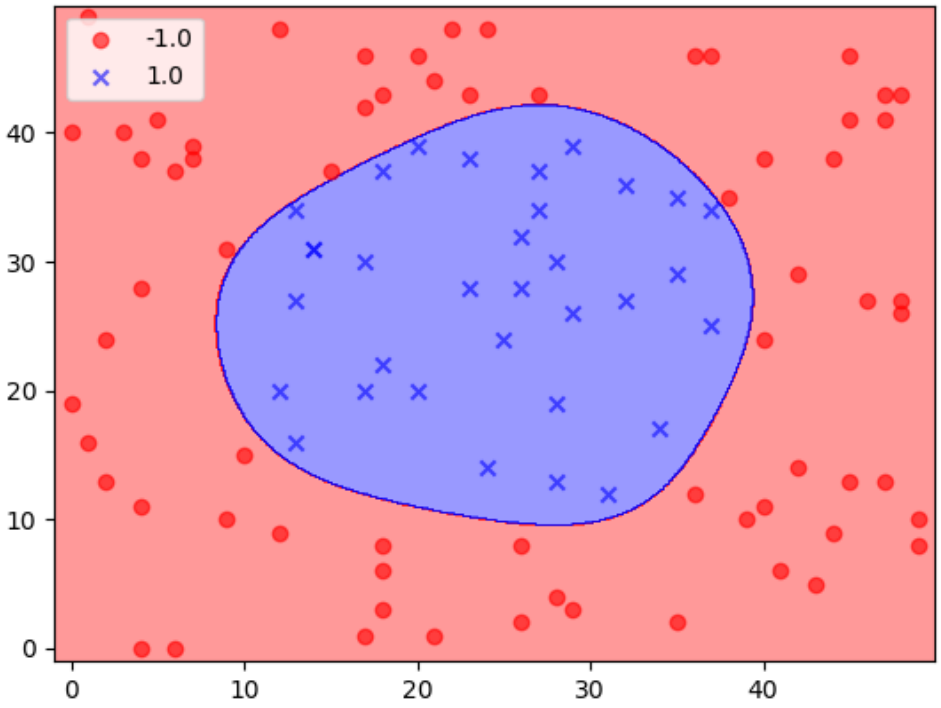

図 2　sample_linear.dat

4.3 poly_five.dat

1 次元データ x=[1, 2, 5, 7, 8], y=[1, 1, -1, -1, 1] を poly_five.dat とする。このとき、α について以下のような結果が得られた。

```
alpha:
0 [2.86101852e-09]
1 [1.01333335]
2 [8.47919771e-08]
3 [4.07999985]
4 [3.06666658]
```

5 考察

sample_circle.dat など線形分離不可能なデータ点に対しては、線形 SVM では 2 次計画問題を解くことができずにプログラムが終了した。一方、カーネル SVM(たとえば Gauss) では、パラメータを適切に設定することでデータ点を高次元空間に射影して、非線形の決定境界でクラスを正確に分類することができた。線形 SVM で 2 次計画問題が解けずに終了するのではなく、ソフトマージンを実装してある程度の精度で分類できるようにしたい。

SVClassifier.predict の実装について、はじめは

```
def predict(self, tX):
    batch_size = tX.shape[0]
    preds = []
    for i in range(batch_size):
        res = 0
        for j in range(self.__n):
            res += self.__alpha[j] * self.__y[j] * self.__kf(self.__X[j], tX[i])
        res -= self.__theta
        preds.append(np.sign(res))
    return np.array(preds)
```

のように実装していたが、このとき"./main.py data/sample_circle.dat g 10.0"(resolution=0.1) を実行するとグラフを表示するまでに自分の環境では 10 分 27 秒必要だった。これは、計算量を概算すると、この実装では batch_size(予測するデータのサイズ) * n(訓練データのサイズ) の計算量であり、x1, x2 がおよそ (0, 50) の範囲で、0.1 の幅でグリッド点を取ることを考えると、500*500*100 と膨大になるためである。kernel 関数をバッチ入力に対応させて、訓練データのサイズ分の for 文を無くした実装に変更すると自分の環境で 0 分 8 秒で実行できるようになった。

6 参考資料

- Python 機械学習プログラミング Sebastian Raschka 著
- はじめてのパターン認識 平井 有三 著