計算機科学実験及演習 4 エージェント 課題 2

1029-28-2473 二見 颯

2018 年 10 月 19 日

1 プログラム概要

課題 1 で提出したハードマージン SVM に加え、ソフトマージン SVM を新しく実装した。モデルの評価のために、K-fold 交差検証を行えるようにして、特に Gauss カーネル SVM について予測精度が最大となる最適なハイパーパラメータの組を求めた。データの正規化にも対応した。

2 外部仕様

- main.py - プログラムのエントリポイント。svc.py の SoftSVM, HardSVM および utils.py を参照
- svc.py - SVM のクラス HardSVM と SoftSVM を実装
- utils.py - load_data, plot_decision_regions(課題 1 で説明) と cross_val_score を実装
- scaler.py - MinMaxScaler(正規化) のクラスを実装
- extra_data.py - iris, wine のデータセットの読み込み (sklearn.dateset)

プログラムの実行は

./main.py [入力ファイルへのパス] [**-h or -s**] ([**param c**]) [**-n or -p or -g**] ([**param kernel**])

-h or -s でハードマージン SVM を用いるかソフトマージン SVM を用いるか指定できる。また、ソフトマージンの場合、次の [param c] にハイパーパラメータ C の値を指定する。以降はカーネルトリックの種類を指定できる。(課題 1 と同様)

以下は wine のデータセットを読み込んでソフトマージン SVM(C=0.5, 多項式カーネル, p=2.0) で分類する例である。

```
$ ./main.py WINE -s 0.5 -p 2.0
cvxopt.solvers.qp: optimization succeeded
free support vectors:
[8, 19, 20, 24, 56, 67, 76, 86]
theta candidates: [0.7084021246199819, 0.7084772707203442, 0.7139007303715856,
    0.7084021045024342, 0.7084031399344815, 0.7084020933321371,
    0.7084021065508526, 0.7084021134414269]
SoftSVM W = [ 0.08830887 0.14967919 0.30510515 0.14495329 0.05914639 -0.05083011
 -0.44413318 -0.1192389 -0.05448695 0.33204266 -0.31927956 -0.29917772
  0.10649366], θ = 0.7090989604341555
fold 1/5 accuracy: 0.9565217391304348
```

```
cvxopt.solvers.qp: optimization succeeded
free support vectors:
[14, 55, 71, 76, 83, 86, 89]
theta candidates: [0.5551587893610854, 0.555158796964597, 0.5551588093446753,
    0.5551330327872295, 0.5551588059114891, 0.5551587847676993,
    0.5551739666156776]
SoftSVM W = [ 0.21289342 0.1781607 0.17853386 0.08075174 -0.00679296 -0.09959914
 -0.4139207 0.01840754 -0.12596927 0.36167288 -0.33568391 -0.2357181
 -0.0552378 ], θ = 0.5551572836789219
fold 2/5 accuracy: 0.9565217391304348
cvxopt.solvers.qp: optimization succeeded
free support vectors:
[21, 25, 33, 39, 55, 86]
theta candidates: [0.2900072784299388, 0.2900072655921049, 0.2900070578429528,
    0.2900072521962018, 0.290007251049357, 0.2900075710726191]
SoftSVM W = [ 0.17098488 0.14964371 0.21914376 0.05477262 0.05176998 -0.27826677
 -0.41480433 -0.0469668 -0.13237087 0.315096 -0.27987994 -0.34633944
  0.17770832], θ = 0.2900072793638624
fold 3/5 accuracy: 0.9565217391304348
cvxopt.solvers.qp: optimization succeeded
free support vectors:
[14, 21, 25, 43, 83, 84]
theta candidates: [0.4719949477180161, 0.4719949434207287, 0.4719949346704695,
    0.47199493866718045, 0.471994950047387, 0.4719949557813248]
SoftSVM W = [ 0.18554927 0.11934352 0.2010267 0.16951661 -0.01044494 -0.05299243
 -0.38467209 -0.05042564 -0.09414849 0.3313893 -0.32498772 -0.31812269
  0.01811681], θ = 0.4719949450508511
fold 4/5 accuracy: 0.9565217391304348
cvxopt.solvers.qp: optimization succeeded
free support vectors:
[21, 25, 33, 39, 90]
theta candidates: [0.766548728211689, 0.7665727737412085, 0.766548733862761,
    0.7665487398395889, 0.7665487288003439]
SoftSVM W = [ 0.21294178 0.12010485 0.24139871 0.13779676 0.09656673 -0.08730936
 -0.2859088 -0.09577348 -0.3040062 0.38560585 -0.3594145 -0.25748461
  0.0515577 ], θ = 0.7665535408911183
fold 5/5 accuracy: 1.0
5-fold average accuracy: 0.9652173913043478
```

3 内部仕様

課題 1 との差分について報告する

HardSVM クラス (svm.py)

課題 1 の SVClassfier が HardSVM にあたる

SoftSVM クラス (svm.py)

ソフトマージン SVM を定義する

- 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 - kernel function のパラメータ (optional)
- C - ソフトマージン SVM で誤識別に対するペナルティの強さを表すパラメータ
- 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, C 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

3.1 _setLagrange

ソフトマージン利用の場合の最適化問題は、

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

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

となる。例えば α が 2 次元のときに以下のように G, h を定めた。

$$\begin{pmatrix} -1 & 0 \\ 0 & -1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \alpha_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ C \\ C \end{pmatrix} \tag{3}$$

3.2 _setClassifier

ハードマージン SVM ではサポートベクトルを 0 < alpha としたが、ソフトマージン SVM では 0 < alpha < C となる自由サポートベクトルと考える。この自由サポートベクトル点 $\boldsymbol{x_i}$ 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。これらが i によらず一定であることを確認して、その平均を theta とする。

_kernelPolynomial, _kernelGauss

課題 1 と同様

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。

cross_val_score(utils.py)

K-Fold の交差検証を行う
@param[in] X, y, model
@param[in] k: 分割数 (k-Fold)
各分割ごとのデータのクラスラベル等の偏りを無くすため、交差検証の前に X, y をシャッフルする。シャッフル後の X, y について、size を y の要素数として、[0, size/k), [size/k, 2 * size/k), ... とインデックスを k 組に分割して、k-1 組を train data, 1 組を test data とすることを k 回繰り返している。また、モデルを評価する前処理として、train data に対して MinMaxScaler を用いて正規化している。test data は train data と同じパラメータでスケール調整する。

MinMaxScaler クラス (scaler.py)

データの正規化を行う
fit で与えられたデータの最小値、最大値をそれぞれ min, max に代入して、transform(X) で (X - min) / (max - min) に変換する。これにより、各特徴量の値の範囲を 0 から 1 に揃えることができて、精度が向上することが期待される。

4 評価結果

4.1 ソフトマージン SVM の表現力

図 1 のようなデータ点集合 (data/sample_soft.dat) の場合、ハードマージンの線形 SVM では誤識別を許容しないため、最適化問題を解くことができず SVM を構成できない。ここで、C を導入して誤識別を最小化するようなソフトマージン SVM を構成すると、このようなクラスの重なりを含むデータ点集合に対してもある程度の線形分離ができるようになる。図 1 は $C = 0.5$ の場合である。

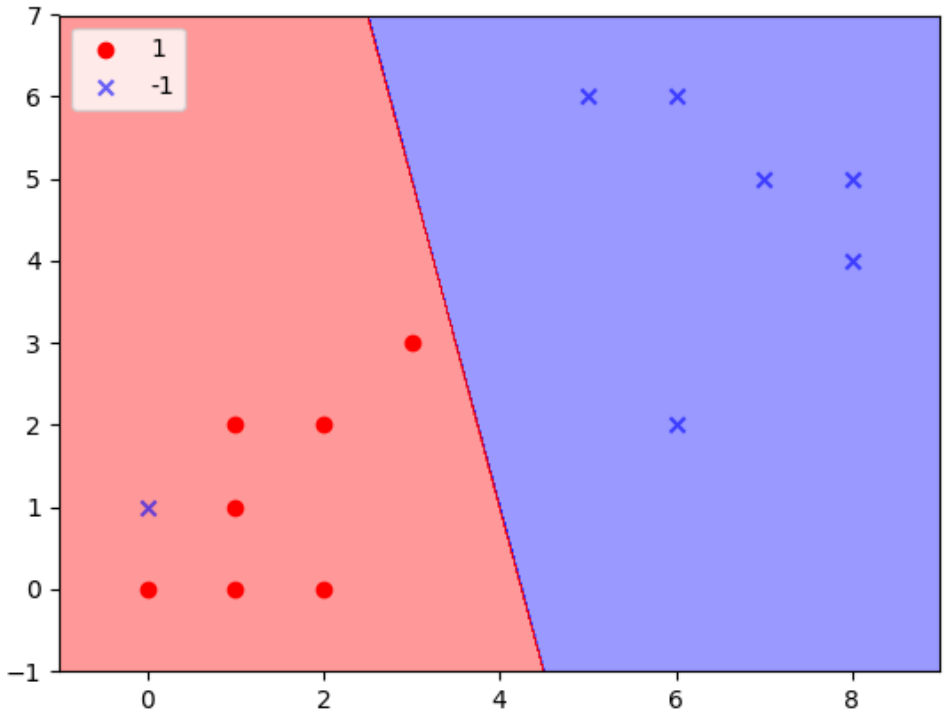

図 1　soft margin example

4.2　ハイパーパラメータ探索 (Grid Search)

sample_circle.dat で Gauss カーネル SVM を評価することを考える。パラメータ p および C の組の値を変更して交差検証を行い accuracy を求めた。p の候補は $(2^(-7), 2^(-6), ..., 2^5)$, C の候補は $(2^(-3), 2^(-2), ..., 2^3)$ として、13*7 通りの Grid Search を行った。
図 2 に結果を示す。この場合最適なパラメータは、$p = 0.25, C = 2.0$ のときで accuracy=0.96 であった。

5　考察

ソフトマージン SVM では、線形識別可能でない場合に対応するためスラック変数 ξ_i を導入する。マージン内で正しく分類できる場合 $\xi_i = 0$, マージン境界を越えるが正しく識別できる場合 $0 < \xi_i <= 1$, 識別境界を越えて誤識別される場合 $\xi_i > 1$ とする。ソフトマージン SVM では、マージンを $2/\|\boldsymbol{w}\|$ とすると、評価関数

$$L = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{w}/2 + C\sum_{i=0}\xi_i \tag{4}$$

を最小化する。パラメータ C を大きくすると、$\|\boldsymbol{w}\|$ の最小化、すなわちマージン最大化よりも誤識別数を最小化する方を優先するように働く。これによって、モデルは訓練データに対して過学習しやすくなる。パラ

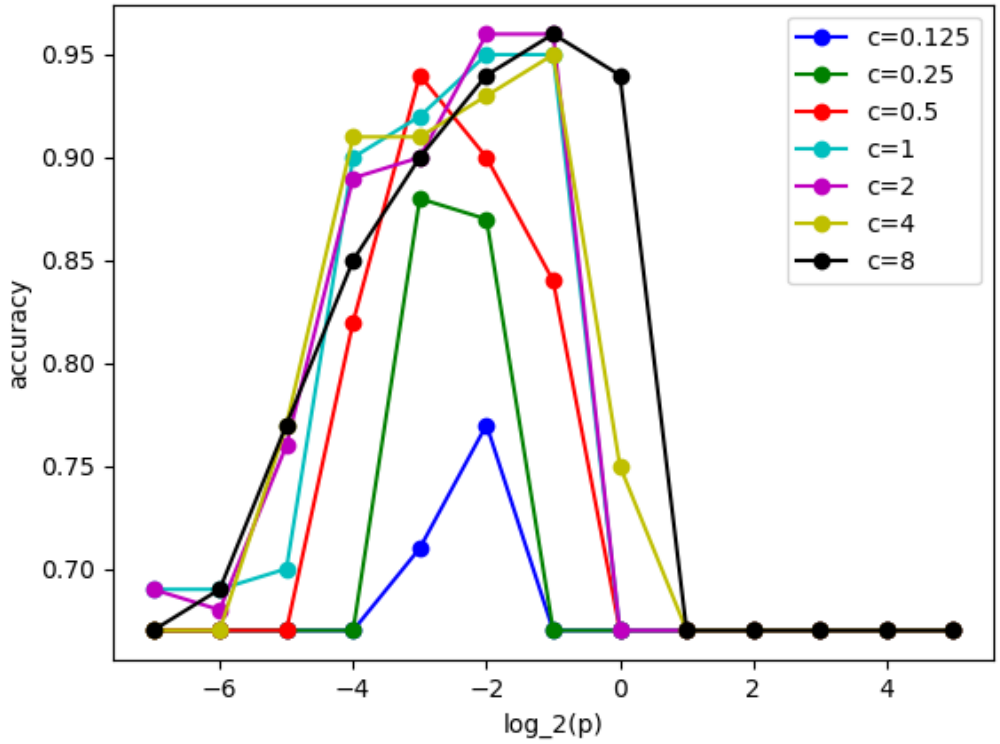

図 2　Grid Search

メータ C を小さくすると、逆にマージン最大化は優先されるが、誤識別は多くなり、訓練データに対して過小適合しやすくなる。パラメータ C はマージン最大化と誤識別数の最小化のトレードオフを表すパラメータといえる。

Gauss カーネルのパラメータ $\sigma(=p)$ を小さくすると、モデルは訓練データに対して過学習して、$\sigma(=p)$ を大きくするとモデルは訓練データに過小適合する傾向が見られる。例として図 3 は、ソフトマージン SVM(C=2.0) で $\sigma = 0.02$ とした場合であり、過学習していることがわかる。SVM の識別関数は $f(\boldsymbol{x}) = sign(\sum_{k=0} \alpha_k y_k \exp(-\|\boldsymbol{x_k} - \boldsymbol{x}\|/2p^2) - \theta)$ となる。σ が小さい場合は入力データ $\boldsymbol{x_k}$ から遠く離れているサポートベクトルが識別に寄与することになるが、σ が大きい場合は入力データ $\boldsymbol{x_k}$ 近傍のサポートベクトルのみが識別に寄与することになる。これにより、過小適合、過学習が説明できる。

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
- scikit-learn と Tensorflow による実践機械学習 Aurelien Geron 著

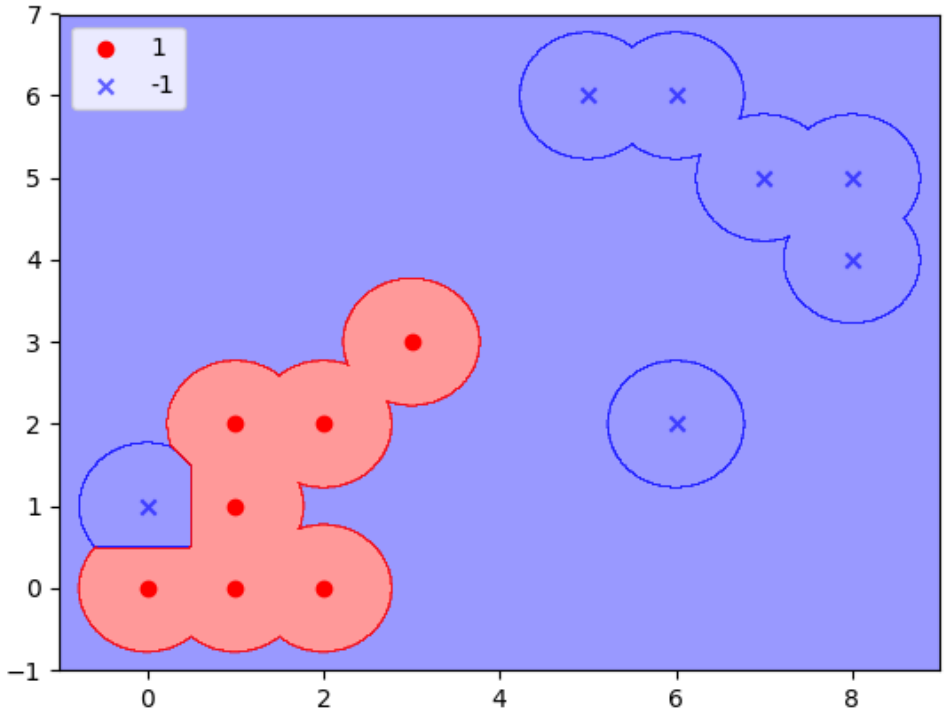

図 3　overfitting