7.1.4 回帰のためのSVM

平成 28 年 9 月 11 日

概 要

PRML の「7.1.4 回帰のための SVM」についての実装と考察

目 次

1　問題設定

回帰問題に SVM を適用する. メリットとしては, 疎な解が得られるということである.

2　アルゴリズム

通常の線形回帰では, 正則化二乗和誤差最小化を用いる.

$$\frac{1}{2}\sum_{n=1}^{N}\{y_n - t_n\}^2 + \frac{\lambda}{2}\|\mathbf{w}\|^2 \quad (7.50)$$

疎な解を得るため, この誤差関数を ϵ 許容誤差関数で置き換える

$$E_\epsilon(y(\mathbf{x}) - t) = \begin{cases} 0 & (|y(\mathbf{x}) - t| < \epsilon \text{のとき}) \\ |y(\mathbf{x}) - t| - \epsilon & (otherwise) \end{cases}$$

この誤差関数を用いて

$$C\sum_{n=1}^{N} E_\epsilon(y(\mathbf{x}_n) - t_n) + \frac{1}{2}\|\mathbf{w}\|^2 \quad (7.51)$$

の最小化を目標にする.
分類問題における SVM と同様, スラック変数 $(\xi_n \geq 0, \hat{\xi}_n \geq 0)$ を導入して表現することができる. $\xi_n > 0$ は $t_n > y(\mathbf{x}_n) + \epsilon$ つまりチューブの上部のデータ点に対応し, $\hat{\xi}_n > 0$ は $t_n < y(\mathbf{x}_n) - \epsilon$ つまりチューブの下部のデータ点に対応している. これより, チューブの外側にデータ点が存在することを許すような制約が書けて,

$$y(\mathbf{x}_n) - \epsilon - \hat{\xi}_n \leq t_n \leq y(\mathbf{x}_n) + \epsilon + \xi_n \quad (7.53)(7.54)$$

スラック変数を用いて SVM 回帰の誤差関数は

$$C\sum_{n=1}^{N}(\xi_n + \hat{\xi}_n) + \frac{1}{2}\|\mathbf{w}\|^2 \quad (7.55)$$

とかけ, これを $\xi_n \geq 0, \hat{\xi}_n \geq 0$ と (7.53)(7.54) の下で最小化する. これには, ラグランジュの未定乗数法を用いる.

$$L = C\sum_{n=1}^{N}(\xi_n + \hat{\xi}_n) + \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{n=1}^{N}(\mu_n\xi_n + \hat{\mu}_n\hat{\xi}_n) - \sum_{n=1}^{N} a_n(\epsilon + \xi_n + y_n - t_n) - \sum_{n=1}^{N}\hat{a}_n(\epsilon + \hat{\xi}_n - y_n + t_n) \quad (7.56)$$

これの $\mathbf{w}, b, \xi_n, \hat{\xi}_n$ による偏微分を 0 として

$$\mathbf{w} = \sum_{n=1}^{N}(a_n - \hat{a}_n)\boldsymbol{\phi}(\mathbf{x}_n),\ \sum_{n=1}^{N}(a_n - \hat{a}_n) = 0,\ a_n + \mu_n = 0,\ \hat{a}_n + \hat{\mu}_n = 0 \quad (7.57)\sim(7.60)$$

この式から双対表現が得られ

$$L(\mathbf{a}, \hat{\mathbf{a}}) = -\frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N}(a_n - \hat{a}_n)(a_m - \hat{a}_m)k(\mathbf{x}_n, \mathbf{x}_m) - \epsilon\sum_{n=1}^{N}(a_n + \hat{a}_n) + \sum_{n=1}^{N}(a_n - \hat{a}_n)t_n \quad (7.61)$$

を $a_n, \hat{a}_n$ について最大化することとなる. ただし, (7.58) と以下の下で

$$0 \leq a_n \leq C,\ 0 \leq \hat{a}_n \leq C \quad (7.62)(7.63)$$

また, 上より $a_n, \hat{a}_n$ は定まるので次に b を考える.
まず, KKT 条件

$$a_n(\epsilon+\xi_n+y_n-t_n)=0,\ \hat{a}_n(\epsilon+\hat{\xi}_n-y_n+t_n)=0,\ (C-a_n)\xi_n=0,\ (C-\hat{a}_n)\hat{\xi}_n=0\ \ (7.65)\sim(7.68)$$

より, $a_n \neq 0$ あるいは$\hat{a}_n \neq 0$ が成り立つサポートベクトル (チューブの外側の点) では $\epsilon+\xi_n+y_n-t_n=0$ あるいは$\epsilon+\hat{\xi}_n-y_n+t_n=0$ が成り立つ.
もしチューブの上側のサポートベクトルなら (7.67) より $\xi_n=0$ で

$$b=t_n-\epsilon-\mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}_n)=t_n-\epsilon-\sum_{m=1}^{N}(a_m-\hat{a}_m)k(\mathbf{x}_n,\mathbf{x}_m)\ \ (7.69)$$

もしチューブの下側のサポートベクトルなら (7.68) より $\hat{\xi}_n=0$ で

$$b=t_n+\epsilon-\mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}_n)=t_n+\epsilon-\sum_{m=1}^{N}(a_m-\hat{a}_m)k(\mathbf{x}_n,\mathbf{x}_m)\ \ (7.69)'$$

これを, 全てのサポートベクトルについて計算し平均をとる.
以上で必要なパラメータは全て求まり, モデルは

$$y(\mathbf{x})=\sum_{n=1}^{N}(a_n-\hat{a}_n)k(\mathbf{x},\mathbf{x}_n)+b\ \ (7.64)$$

で与えることができる.

2.1 パラメータ b の必要性

以下に各コードは b を求めていないが, 求め使ったときよりも誤差が小さい.

$$y(\mathbf{x})=\mathbf{w}^T\boldsymbol{\phi}(\mathbf{x})$$

を用いるのなら, b は必要がないように感じる.
ただ, 多項式カーネルなどでは $x=0$ で $y=0$ を通ってしまうため必要にも思われる.

3 コード

SVM 回帰のコード (SVRpy).

```
K=np.zeros((N,N))
for n in range(N):
        for m in range(N):
                K[n,m]=kernel(x[n],x[m])
q=np.zeros((2*N,2*N))
for n in range(2*N):
        for m in range(2*N):
                if n<N and m<N:
                        q[n,m]=K[n,m]
                if n<N and N<=m:
                        q[n,m]=-K[n,m-N]
                if N<=n and m<N:
                        q[n,m]=-K[n-N,m]
                if N<=n and N<=m:
                        q[n,m]=K[n-N,m-N]
Q = cvxopt.matrix(q)
```

```
#P
p=np.zeros(2*N)
for n in range(2*N):
        if n<N:
                p[n]=eps-t[n]
        else:
                p[n]=eps+t[n-N]
p = cvxopt.matrix(p)
#G
temp1 = np.identity(2*N)*(-1)
temp2 = np.identity(2*N)
G = cvxopt.matrix(np.vstack((temp1, temp2)))
#h
temp1 = np.zeros(2*N)
temp2 = np.ones(2*N)*C
h = cvxopt.matrix(np.hstack((temp1, temp2)))
#A
temp1= np.zeros(2*N)
for n in range(2*N):
        if n<N:
                temp1[n]=1
        else:
                temp1[n]=-1
A = cvxopt.matrix(temp1,(1,2*N))
#0
b = cvxopt.matrix(0.0)

sol = cvxopt.solvers.qp(Q, p, G, h, A, b)
a = np.array(sol['x']).reshape(2*N)
print "a",a,"\n"

"""モデル"""
def model(z):
        res=0
        for n in range(N):
                res+=(a[n]-a[n+N])*kernel(z,x[n])
        return res
```

今現在 cvxopt が ubuntu の python2 でしか使えない.

4 結果

カーネル関数に

$$k(\mathbf{x}_n, \mathbf{x}_m) = exp(-\theta(\mathbf{x}_n - \mathbf{x}_m)^2)$$

を用いた. ただし, データ数は $N = 100$ とし $\theta = 0.1N$ を用いた.

図 1: $\epsilon = 0.1, C = 0.05, 0.5, 5, 50$

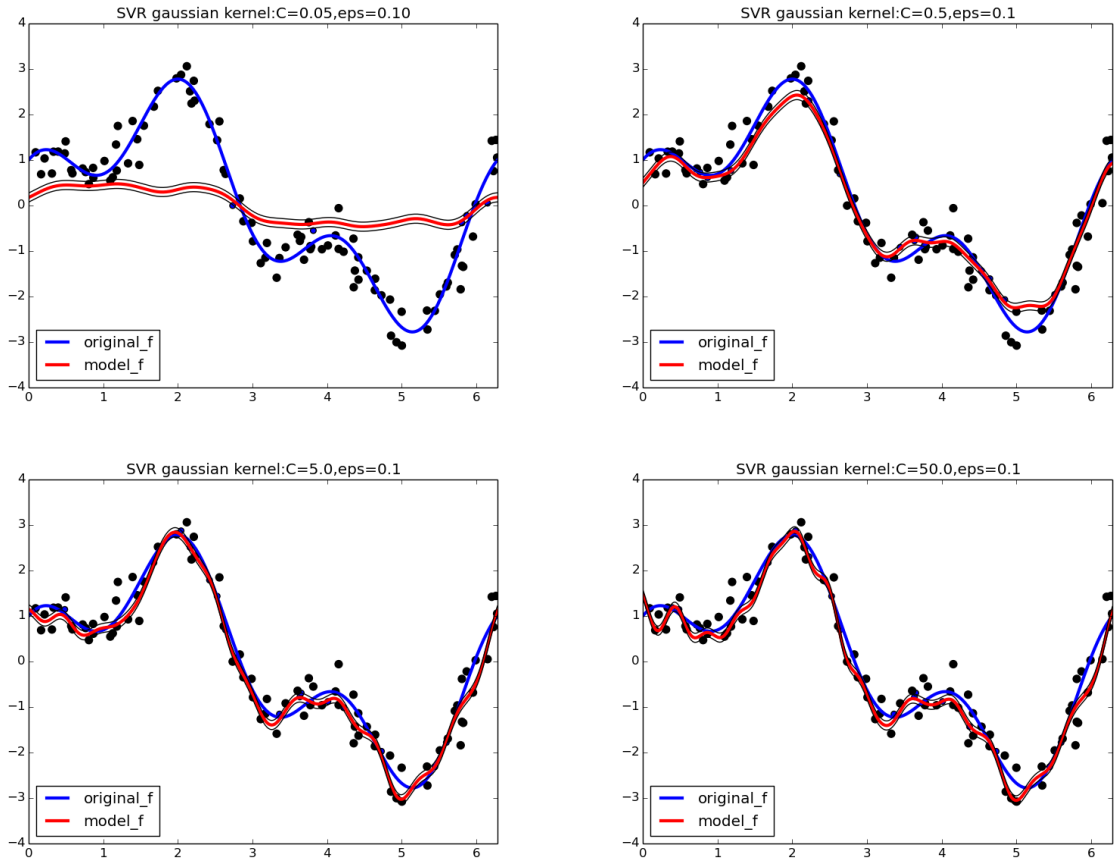


図 2: $\epsilon = 0.3, C = 0, 050.5, 5, 50$

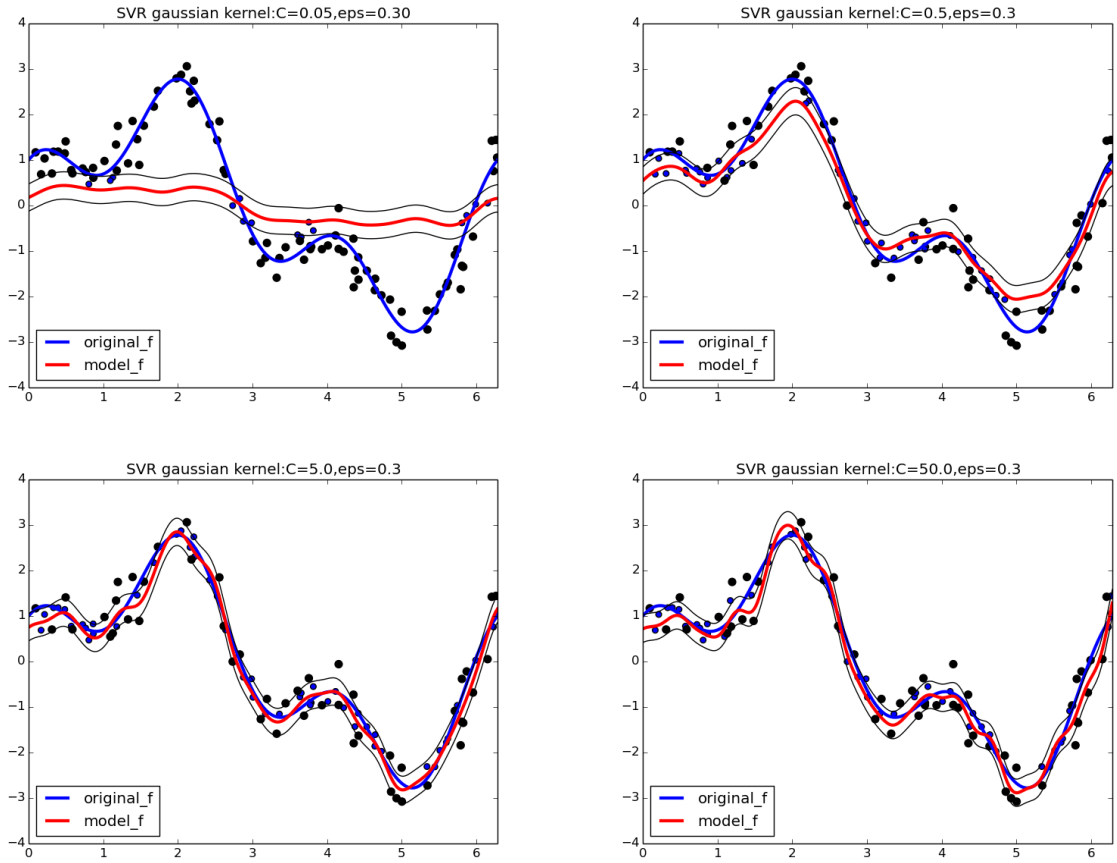


図 3: $\epsilon = 1, C = 0.05, 0.5, 5, 50$

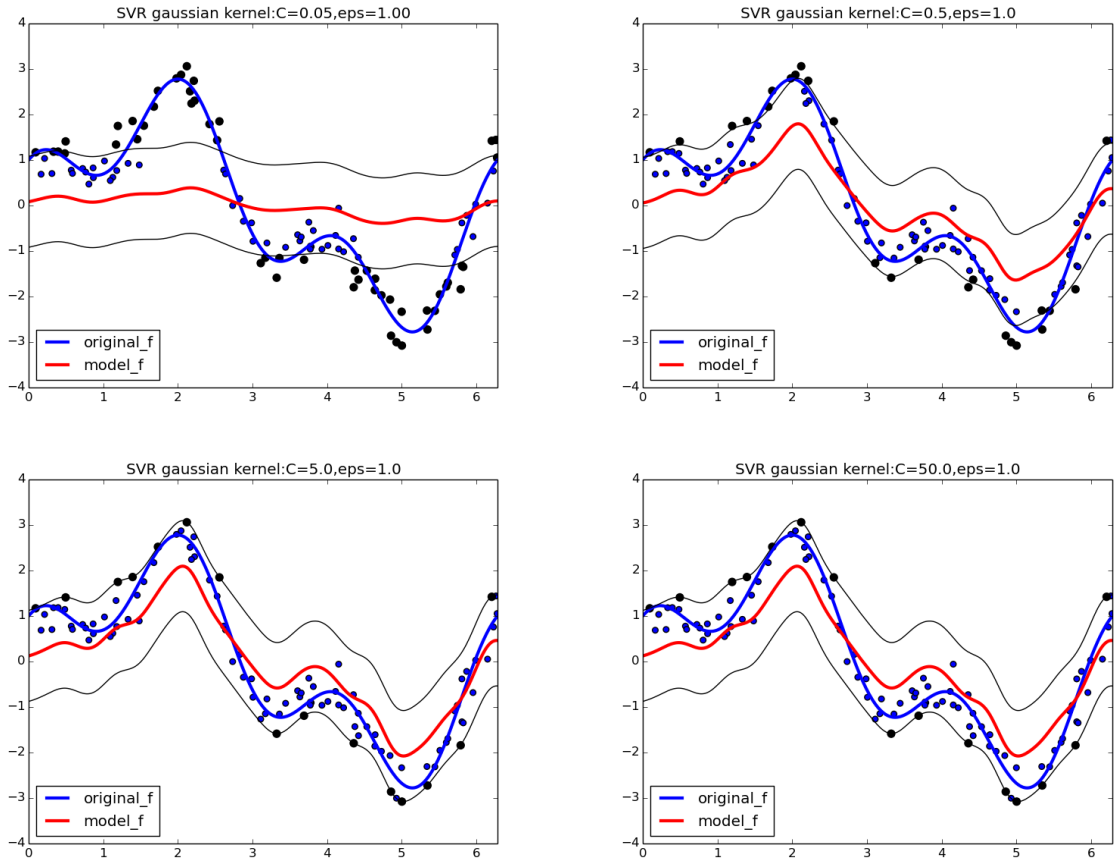
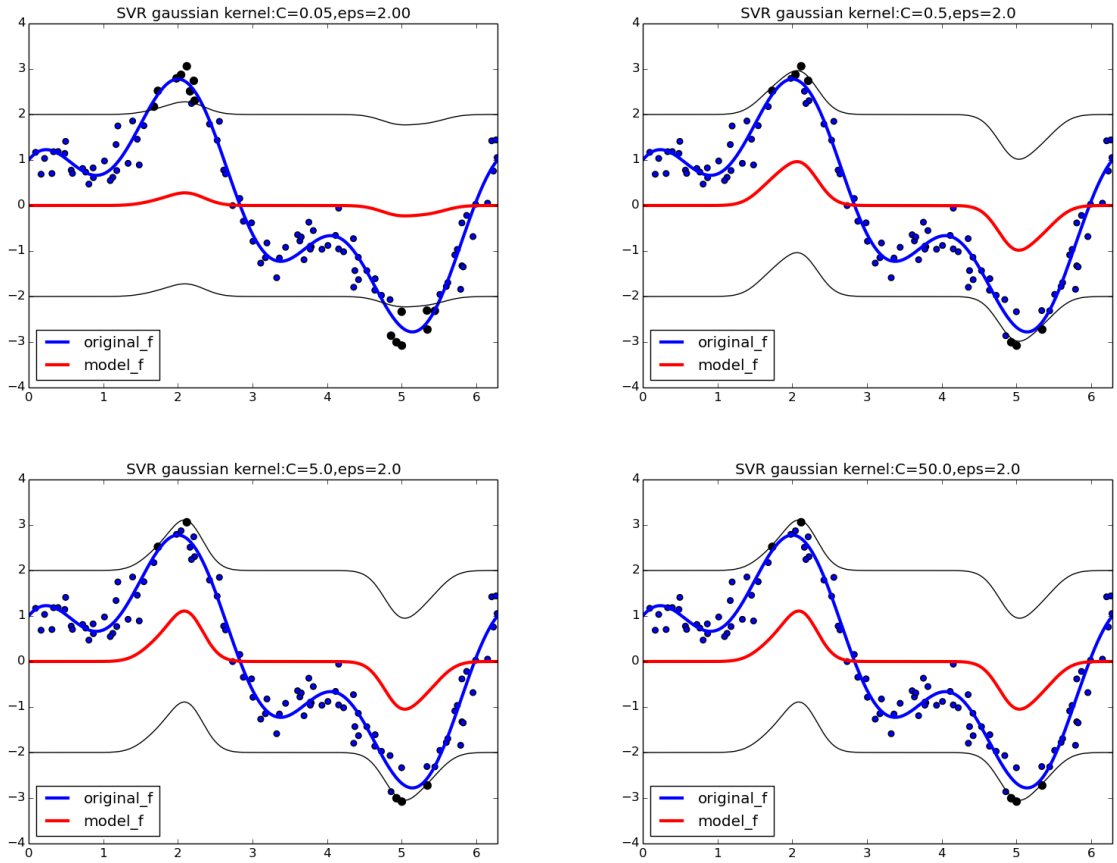


図 4: $\epsilon = 2, C = 0.05, 0.5, 5, 50$



5 まとめ

パラメータ ϵ が小さ過ぎると多くの点がサポートベクトルとなり, SVR のありがたみは消える. 対して, 大き過ぎると, サポートベクトルは少なくなり, 回帰は失敗する.
パラメータ C が小さ過ぎると, ペナルティ項に大きく影響され 0(b) 付近となる. 対して大きすぎると過学習する恐れがある.