7.2.3 分類のためのRVM

平成 28 年 9 月 11 日

概 要

PRMLの「7.2.3 分類問題に対するRVM」についての実装と考察

目 次

1 問題設定

RVM を分類問題に適用する.

2 アルゴリズム

分類問題ということで, モデルはロジスティックシグモイド関数を用いて

$$y(\boldsymbol{x}, \mathbf{w}) = \sigma(\mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}) \quad (7.108)$$

とする. RVM では, パラメータ W の事前分布の精度パラメータ $\boldsymbol{\alpha}$ を導入する. 分類問題では, パラメータ W について解析的に積分できず, ラプラス近似を用いる.
W の事後分布のモードは次の式を W について最大化する点として与えられる.

$$\begin{aligned}\ln p(\mathbf{w}|\mathbf{t}, \boldsymbol{\alpha}) &= \ln\{p(\mathbf{t}|\mathbf{w})p(\mathbf{w}|\boldsymbol{\alpha})\} - \ln p(\mathbf{t}|\boldsymbol{\alpha}) \\ &= \sum_{n=1}^{N}\{t_n \ln y_n + (1-t_n)\ln(1-y_n)\} - \frac{1}{2}\mathbf{w}^T A\mathbf{w} + const \quad (7.109)\end{aligned}$$

これには, 反復重み付け最小二乗法 (4.3.3 節) を用いる.

$$\mathbf{w}^{(new)} = \mathbf{w}^{(old)} - H^{-1}\nabla E(\mathbf{w}) \quad (4.92)$$

ここで, $H, \nabla E(\mathbf{w})$ は

$$\nabla E(\mathbf{w}) = -\nabla \ln p(\mathbf{w}|\mathbf{t}, \boldsymbol{\alpha}) = A\mathbf{w} - \Phi^T(\mathbf{t} - \mathbf{y}) \quad (7.110)$$

$$H = -\nabla\nabla \ln p(\mathbf{w}|\mathbf{t}, \boldsymbol{\alpha}) = \Phi^T B\Phi + A \quad (7.111)$$

ただし, $B = diag(y_n(1-y_n))$.
得られた近似事後分布のモードは (7.110) より

$$\mathbf{w}^* = A^{-1}\Phi^T(\mathbf{t} - \mathbf{y}) \quad (7.112)$$

$$\Sigma = (\Phi^T B\Phi + A)^{-1} \quad (7.113)$$

得られたラプラス近似を用いて周辺尤度を計算すると

$$\begin{aligned}p(\mathbf{t}|\boldsymbol{\alpha}) &= \int p(\mathbf{t}|\mathbf{w})p(\mathbf{w}|\boldsymbol{\alpha})d\mathbf{w} \\ &\neq p(\mathbf{t}|\mathbf{w}^*)p(\mathbf{w}^*|\boldsymbol{\alpha})(2\pi)^{M/2}|\Sigma|^{1/2}\end{aligned}$$

これを, α_i について最大化すると

$$\alpha_i^{new} = \frac{\gamma_i}{(\mathbf{w}_i^*)^2} \quad (7.116)$$

ただし, $\gamma_i = 1 - \alpha_i\Sigma_{ii}$ とした. これは RVM の回帰のときにも現れた. しかし実際のコーディングでは, γ は不必要.

3 コード

RVM による回帰のコード (RVM.py).

```
#ロジスティックシグモイド関数
def sig(z):
  if z<-10**5:
    return 10**-3
  elif z>10**5:
    return 1-10**-3
  else:
    return 1/(1+np.exp(-z))
#カーネル関数の定義
def kernel(x,z):
  theta=1
  return np.exp(-theta*norm(x-z)**2)
for N in [30,50,100,200]:
  x=data[:N,0:2]
  t=data[:N,2]
  P=np.zeros((N,N))
  for n in range(N):
    for m in range(N):
      P[n,m]=kernel(x[n,:],x[m,:])
  W=np.zeros(N)
  y=[sig(dot(W,P[n,:])) for n in range(N)]
  alpha=np.ones(N)
  A=np.zeros((N,N))
  for n in range(N):
    A[n,n]=alpha[n]
  B=np.zeros((N,N))
  for n in range(N):
    B[n,n]=y[n]*(1-y[n])
  S=np.zeros((N,N))
  """Wの決定"""
  roop=0
  res=1
  while roop<10**5 and res>N*10**-6:
    y=[sig(dot(W,P[n,:])) for n in range(N)]
    E=dot(A,W)-dot(P,t-y)
    H=inv(dot(P,dot(B,P))+A)
    temp=W
    W=temp-dot(H,E)
    res=norm(W-temp)
    S=inv(dot(P,dot(B,P))+A)
    alpha=[1/(W[n]**2+S[n,n]) for n in range(N)]
    for n in range(N):
      A[n,n]=alpha[n]
    for n in range(N):
      B[n,n]=y[n]*(1-y[n])
    roop+=1
    print("roop",roop,"res",res)
  R=[]
  for n in range(N):
    if alpha[n]>10**2:
      R.append(n)
  #求まったパラメータからモデル関数を作り
  def model(z):
    phi=np.zeros(N)
    for n in range(N):
      phi[n]=kernel(z,x[n,:])
    return sig(dot(W,phi))

  """プロット"""
  plt.title("RVM␣discrimination␣:␣N=%d" %N)
  plt.xlim([-10,10])
```

```
plt.ylim([-10,10])
#訓練データの散布図
for n in R:
  plt.scatter(x[n,0],x[n,1],s=80,c="black")
for n in range(N):
  if t[n]==1:
    plt.scatter(x[n,0],x[n,1],s=20,c="red")
  else:
    plt.scatter(x[n,0],x[n,1],s=20,c="blue")
#決定境界のプロット
X1, X2 = meshgrid(linspace(-10,10,200), linspace(-10,10,200))
w, h = X1.shape
X1.resize(X1.size)
X2.resize(X2.size)
Z = array([model(np.array((x1,x2))) for (x1, x2) in zip(X1, X2)])
X1.resize((w, h))
X2.resize((w, h))
Z.resize((w, h))
CS = contour(X1, X2, Z, [0.5], colors='k', linewidths=1, origin='lower')
CS = contour(X1, X2, Z, [0.6], colors='r', linewidths=1, origin='lower')
CS = contour(X1, X2, Z, [0.4], colors='b', linewidths=1, origin='lower')
plt.savefig("gauss_%d_1.png" %N)
plt.show()
```

ここでは, 決定面のほかに $y = 0.4, 0.6$ の面もプロットした.

4 結果

$N = 30, 50,\ C = 0.05, 5, 500$ に対して, $y(\mathbf{x}) = -1, 0, 1$ となる面とサポートベクトルをプロットした.

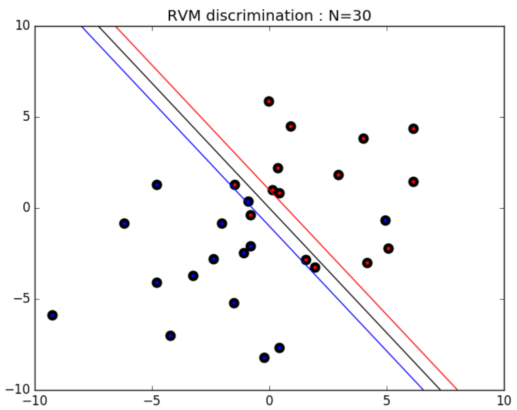
4.1 内積カーネル

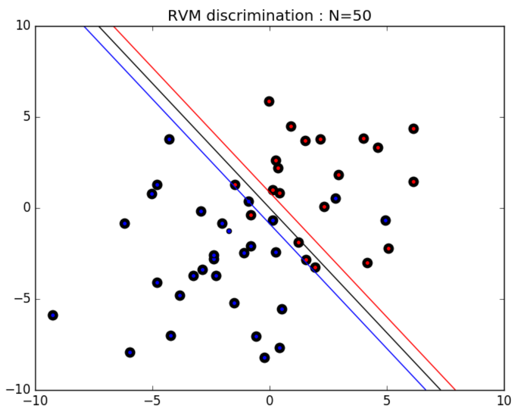
カーネル関数に

$$k(\mathbf{x}_n, \mathbf{x}_m) = \mathbf{x}_n^T \mathbf{x}_m$$

を用いた.

図 1: $N = 30, 50$





4.2　多項式カーネル

カーネル関数に

$$k(\mathbf{x}_n, \mathbf{x}_m) = (\mathbf{x}_n^T \mathbf{x}_m + 1)^\theta$$

を用いた．$\theta = 1, 2, 3$ で試した．

図 2: $N = 30,\ \theta = 1, 2, 3$

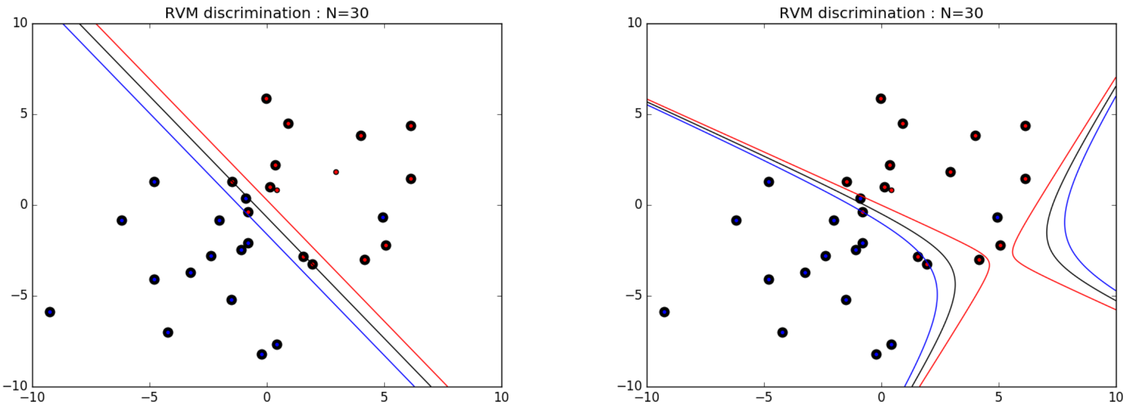
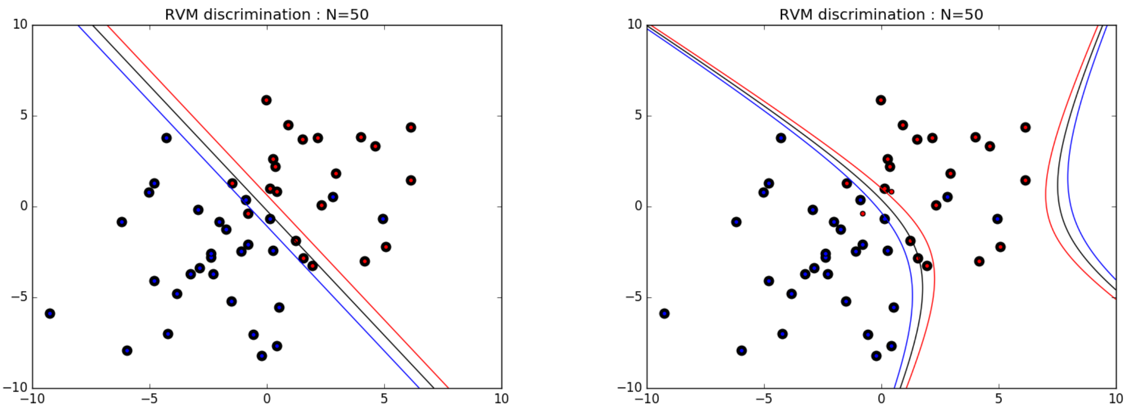


図 3: $N = 50,\ \theta = 1, 2, 3$



ともに $\theta = 3$ では反復重み付け最小二乗法が収束しなかった．

4.3　ガウスカーネル

カーネル関数に以下のガウスカーネルを用いた．$\theta = 0.1, 0.5, 1, 5$ とした．

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

図 4: $N = 30,\ \theta = 0.1, 0.5, 1, 5$

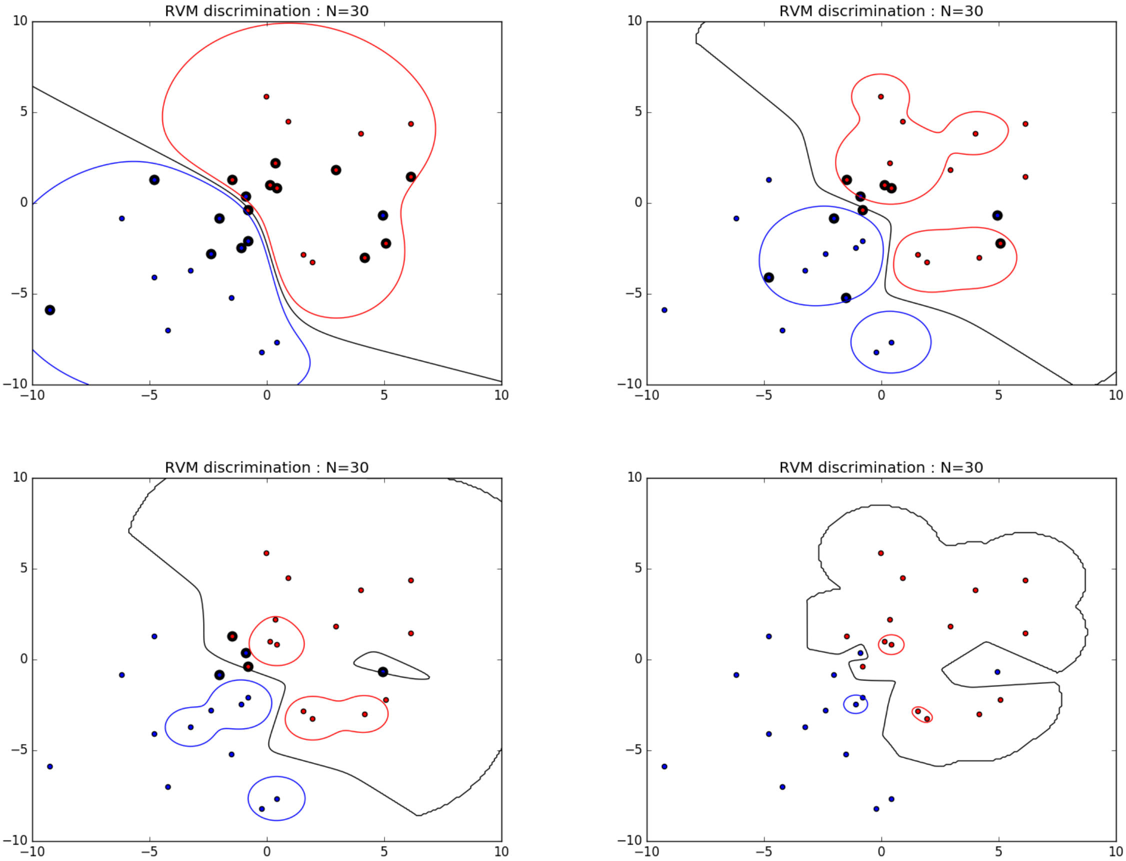


図 5: $N = 50,\ \theta = 0.1, 0.5, 1, 5$

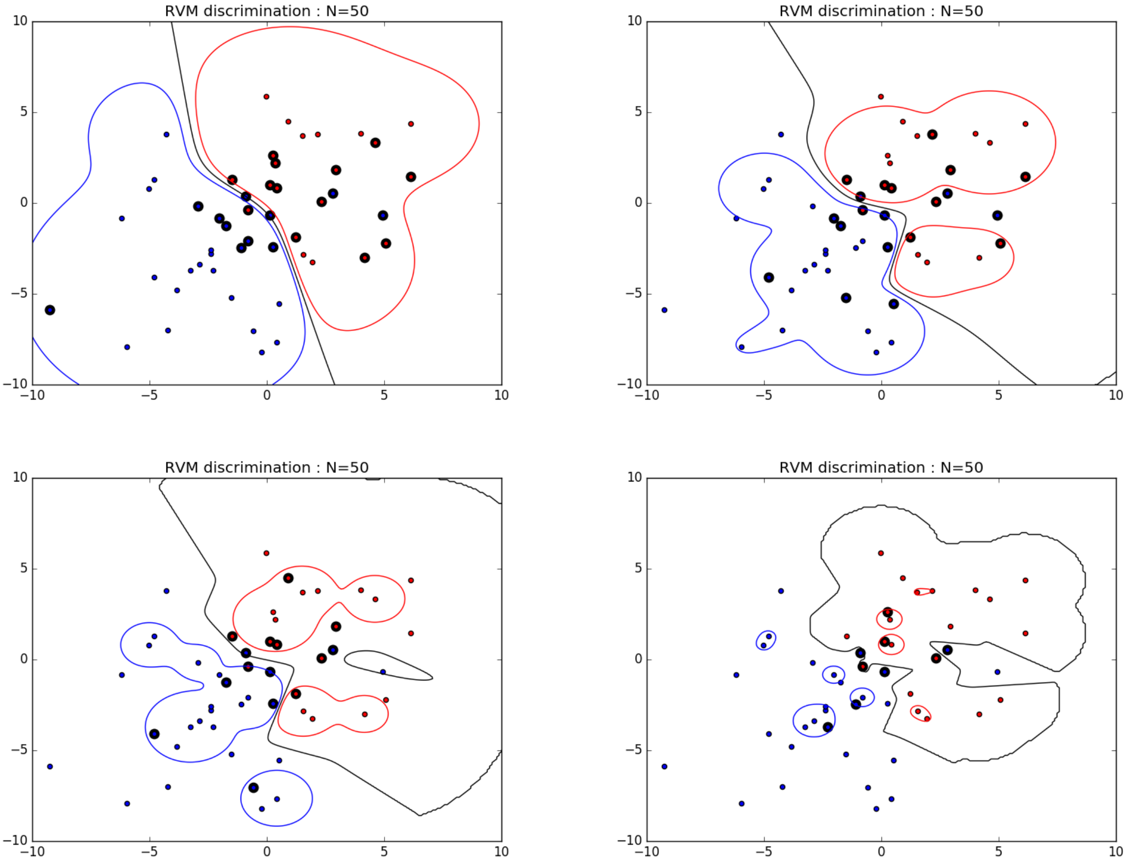


図 6: $N = 100,\ \theta = 0.1, 0.5, 1, 5$

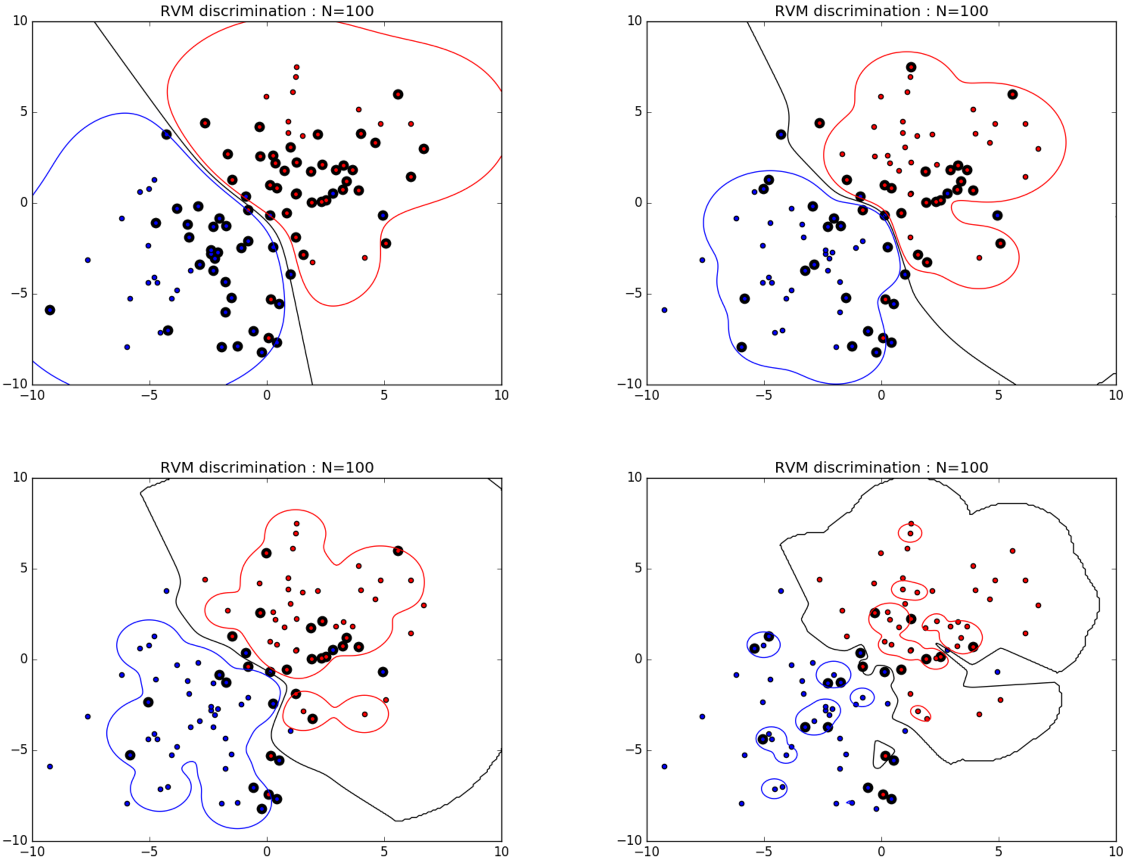
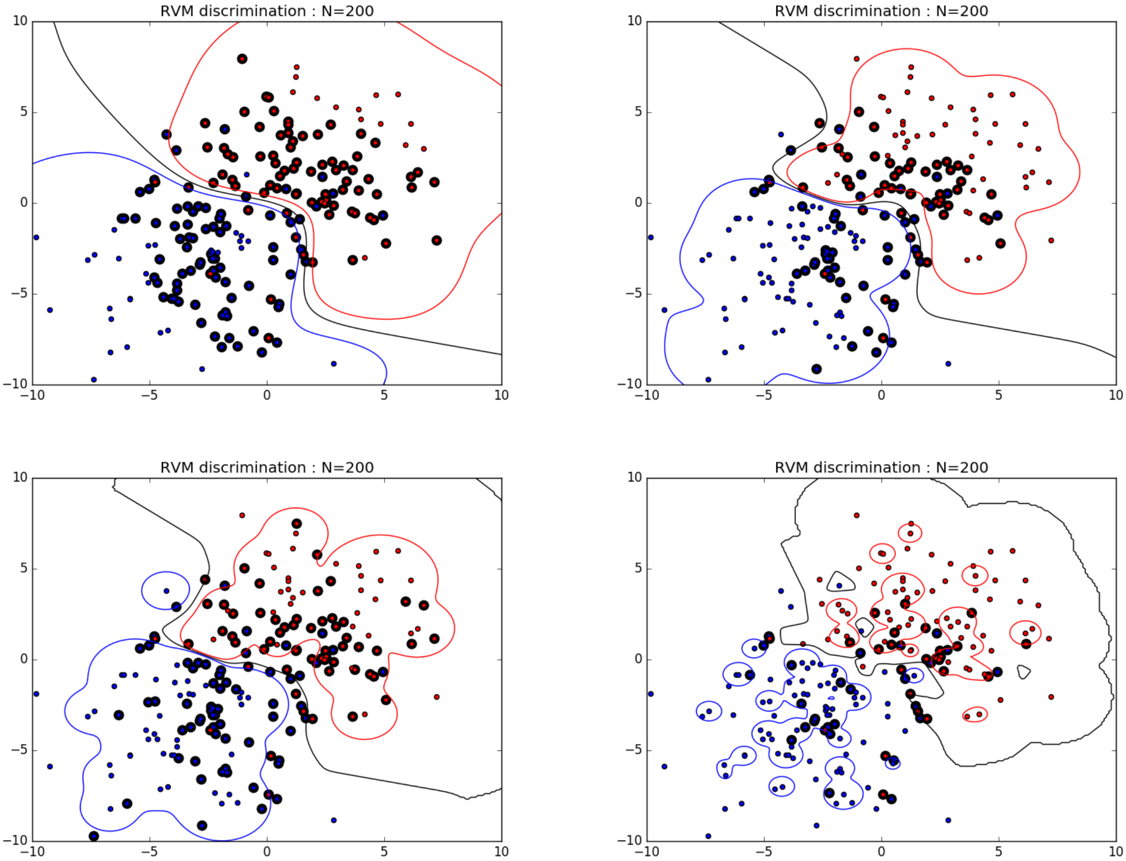


図 7: $N = 200,\ \theta = 0.1, 0.5, 1, 5$



5　まとめ

主に, ガウスカーネルを用いたときの結果から考察する.
ここでは, 関連ベクトルを $\alpha_i > 10^2$ となる添え字の点とした. この結果, 誤分類されたすべての点は関連ベクトルで, 結構バラバラに位置するということになった.
またカーネルのパラメータが妥当なとき決定面は, 結構滑らかなものとなった. パラメータがずれすぎていると, 過学習を引き起こすことがある.