Support Matrix Machine Implementation
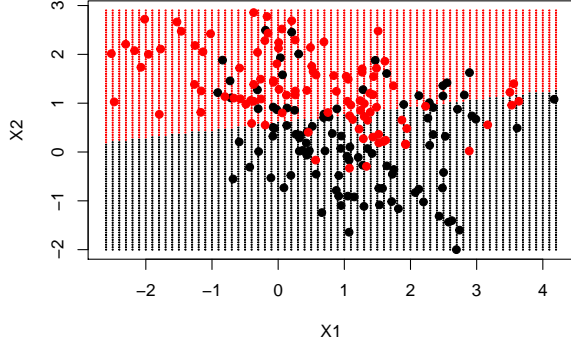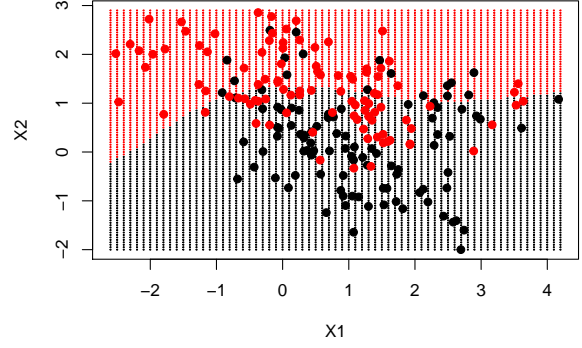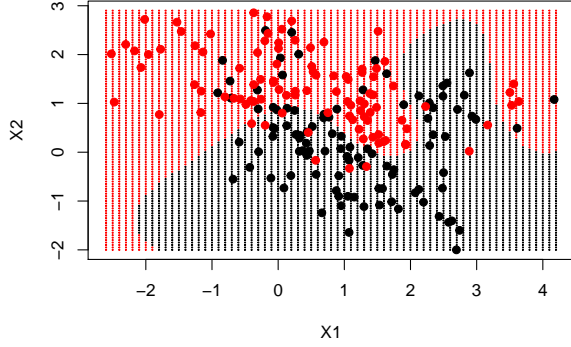
Chanwoo Lee, March 30, 2020
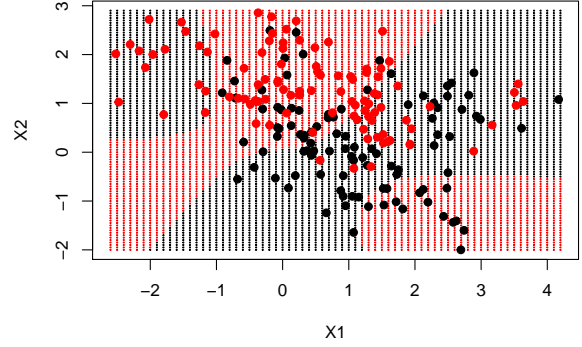
1 Kernel SVM discussion

(a) Soft SVM method: MCR = 0.73

(b) Polynomial kernel method: MCR = 0.67

(c) Radial kernel method: MCR = 0.83

(d) Sigmoid kernel method: MCR = 0.52

Figure 1: One linear SVM and Three nonlinear SVMs for the esl data. Mathematical expression of each boundary is $\langle \hat{\boldsymbol{w}}, h(\boldsymbol{x}) \rangle + \hat{b} = 0$ where h is the corresponding kernel to each type. MCR is misclassification rate

Explicit boundary with kernel function is,

$$\hat{f}(\boldsymbol{x}) = \sum_{i=1}^{m} \hat{\alpha}_i y_i K(\boldsymbol{x}, \boldsymbol{x}_i) + \hat{b}_0. \tag{1}$$

If we apply sigmoid kernel to (1),

$$\hat{f}(\boldsymbol{x}) = \sum_{i=1}^{m} \hat{\alpha}_i y_i \tanh(\gamma * \boldsymbol{x}^T * \boldsymbol{x}_j + \text{ coef0}) + \hat{b}_0.$$
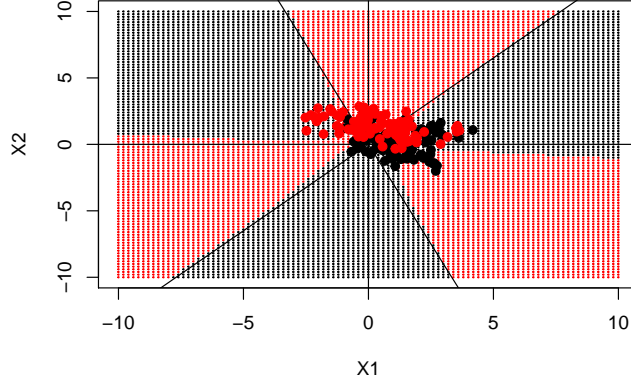
1

Figure 2: Sigmoid kernel classification. Range of x_1 and x_2 are $[-10, 10]$

Since the range of tanh -1 to 1, influence of $\langle \boldsymbol{x}, \boldsymbol{x}_i \rangle$ has limitation. This limitation can explain the phenomenon in Figure 2. Points in top-left and bottom-right do not match with the classifier. It happened that those points cannot dominate the $\hat{f}$ value because $|\tanh| \leq 1$ allowing the opposite labeled points impact to classifier.

2 SMM special case when B has full rank

Let $X_i \in \mathbb{R}^{m \times n} \forall i = 1, \cdots, N$ where $M > n$. We factorize B into $U \in \mathbb{R}^{m \times r}$ and $V \in \mathbb{R}^{n \times r}$ such that $B = UV^T$. Then primal and dual problem for updating U fixing V is as follows,

1. When fixing V,

$$(P_u) \qquad \min_{U, b, \boldsymbol{\xi}} \quad \frac{1}{2}\|UV^T\|^2 + C \sum_{i=1}^{N} \xi_i \tag{2}$$
$$\text{subject to} \quad y_i(\langle UV^T, X_i \rangle + b) \geq 1 - \xi_i,$$
$$\xi_i \geq 0, \quad i = 1, \cdots, N.$$

$$(D_u) \qquad \max_{\boldsymbol{\alpha} \in \mathbb{R}^m : \boldsymbol{\alpha} \geq 0} \left(\sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \langle X_i, X_j H_V \rangle \right)$$
$$\text{subject to} \quad \sum_{i=1}^{N} y_i \alpha_i = 0,$$
$$0 \leq \alpha_i \leq C, \quad i = 1. \cdots, N,$$

where $H_V = V(V^T V)^{-1} V^T$. We have the optimizer $U = \sum_{i=1}^{N} \alpha_i y_i X_i V(V^T V)^{-1}$.

2. When fixing U,

$$(P_v) \qquad \min_{V,b,\boldsymbol{\xi}} \quad \frac{1}{2}\|UV^T\|^2 + C\sum_{i=1}^{N}\xi_i \qquad\qquad (3)$$

$$\text{subject to} \quad y_i(\langle UV^T, X_i\rangle + b) \geq 1 - \xi_i,$$
$$\xi_i \geq 0, \quad i = 1, \cdots, N.$$

$$(D_v) \qquad \max_{\boldsymbol{\alpha}\in\mathbb{R}^m:\boldsymbol{\alpha}\geq 0} \left(\sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j \langle X_i, H_U X_j\rangle\right)$$

$$\text{subject to} \quad \sum_{i=1}^{N} y_i\alpha_i = 0,$$
$$0 \leq \alpha_i \leq C, \quad i = 1.\cdots, N,$$

where $H_U = U(U^TU)^{-1}U^T$. We have the optimizer $V^T = \sum_{i=1}^{N}\alpha_i y_i(U^TU)^{-1}U^T X_i$.

Suppose B has full rank such that $U \in \mathbb{R}^{m\times n}$ and $V \in \mathbb{R}^{n\times n}$ both of which are full rank. Notice that $H_V X^T = X^T$, for any $X \in \mathbb{R}^{m\times n}$. Therefore the dual problem (D_u) is reduced to

$$(D') \qquad \max_{\boldsymbol{\alpha}\in\mathbb{R}^m:\boldsymbol{\alpha}\geq 0} \left(\sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j \langle X_i, X_j\rangle\right)$$

$$\text{subject to} \quad \sum_{i=1}^{N} y_i\alpha_i = 0,$$
$$0 \leq \alpha_i \leq C, \quad i = 1.\cdots, N,$$

which is the dual problem of the primer,

$$\min_{B,b,\boldsymbol{\xi}} \frac{1}{2}\|\operatorname{Vec}(B)\|^2 + C\sum_{i=1}^{N}\xi_i \qquad\qquad (4)$$

$$\text{subject to } y_i(\langle \operatorname{Vec}(B), \operatorname{Vec}(X_i)\rangle + b) \geq 1 - \xi_i,$$
$$\xi_i \geq 0, \quad i = 1, \cdots, N.$$

Therefore, we have the same optimzer $\boldsymbol{\alpha}$ for dual problems in (2) and (4). The optimal value

$$U = \sum_{i=1}^{N}\alpha_i y_i X_i V(V^TV)^{-1} \quad \text{in (2)},$$

$$B = \sum_{i=1}^{N}\alpha_i y_i X_i \quad \text{in (4)},$$

coincide the fact $B = UV^T$ considering $H_V X_i^T = X_i^T$.

3 Algorithm implementation

I use matrix factorization technique, where instead of optimizing with respect to B, it is factorized into two matrices $U \in \mathbb{R}^{m \times r}$ and $V \in \mathbb{R}^{n \times r}$ such that $B = UV^T$. One then optimize with respect to U and V. Our main loss function is

$$\min_{U,V,b} \frac{1}{2} \|UV^T\|^2 + C \sum_{i=1}^{N} \max\{0, 1 - y_i(\langle UV^T, X_i \rangle + b)\}.$$

You can check the function `objv` for the loss function in R-codes. I update U fixing V from (2). I use `svm` function in the r-package `e1071` to solve the dual problem in (2). One can notice we can factorize H_V and H_U as $H_V = G_V G_V^T$ and $H_U = G_U G_U^T$. With this factorization we have,

$$\langle X_i, X_j H_V \rangle = \langle X_i G_V, X_j G_V \rangle, \quad \langle X_i, H_U X_j \rangle = \langle G_U^T X_i, G_U^T X_j \rangle.$$

I first directly apply `svm` function with $(X_1 G_V, y_1), \cdots, (X_N G_V, y_N)$ to get the optimizer $\boldsymbol{\alpha}$ for the dual problem (2) and vice versa for (3). SMM algorithm is summarized in Algorithm 1. You can check the r function `SMM` in R-codes. However, this algorithm does not work well. I find that the function `svm` sometimes fail to find the optimizer in the dual problems. This unstable performance happens in the new r function `SVM` too.

In this reason, I changed the approach. Define $G_v(i,j) = y_i y_j \langle X_i, X_j H_V \rangle$ then, the dual problem becomes quadratic programming.

$$\min_{\boldsymbol{\alpha}} \frac{1}{2} \boldsymbol{\alpha}^T G_v \boldsymbol{\alpha} - \mathbf{1}^T \boldsymbol{\alpha} \quad \text{s.t.} \quad \boldsymbol{y}^T \boldsymbol{\alpha} = 0, \quad 0 \le \alpha_i \le C \quad \forall i = 1, \cdots, N$$

Therefore, I solve this constraint quadratic programming solution with `solve.QP` function in `quadprod` package. From this optimizer, I can update $U = \sum_{i=1}^{N} \alpha_i y_i X_i V (V^T V)^{-1}$. You can update V by the same way. You can check `smm` for this method in R-codes. From some simulations, I checked that this approach has stable performance and has smaller loss than at true parameters.

In R-codes section, I attached function codes needed for SMM implementation and simulation codes.

Algorithm 1: SMM algorithm

Input: $(X_1, y_1), \cdots, (X_N, y_N)$, rank r
Parameter: U,V
Initizlize: $U^{(0)}, V^{(0)}$
Do until converges
   Update U fixing V :
   Solve $(D_u) : \max_{\boldsymbol{\alpha}} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \langle X_i, X_j H_V \rangle$.
   $U = \sum_{i=1}^{N} \alpha_i y_i X_i V (V^T V)^{-1}$.
   Update V fixing U :
   Solve $(D_v) : \max_{\boldsymbol{\alpha}} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \langle X_i, H_U X_j \rangle$.
   $V = \sum_{i=1}^{N} \alpha_i y_i X_i^T U (U^T U)^{-1}$.
Output: $B = UV^T$

---

4 Simulation results

In Simulation 1, I first generate a low-rank B and any matrix ensembles $\{X_i\}$ (not necessarily low-rank) as you like. and simulate Y_i based $Y_i = sign(\langle X_i, B \rangle + b)$. In simulation 2, I simulate two classes of matrices $\{u_1 v_1^T + E_i\}, \{u_2 v_2^T + E_i\}$ together with -1, 1 labels. Then assess the misclassification rate. The following shows the cross validation results.

		CV1	CV2	CV3	CV4	CV5
Simulation 1	SVM	0.70	0.55	0.75	0.60	0.70
	SMM	0.55	0.45	0.65	0.45	0.75
Simulation 2	SVM	0.95	0.95	1	0.925	0.95
	SMM	0.95	0.95	0.975	0.95	1

5 R-codes

5.1 Functions

```r
library(e1071)
library(quadprog}
library(pracma)

library(e1071)
library(pracma)

eps = 10^-5

sqrtH = function(Us,U){
  h = eigen(Us%*%t(U))
  return(h$vectors%*%diag(sqrt(pmax(h$values,eps))))
}

objv = function(B,b0,X,y,cost = 10){
  return(sum(B*B)/2+cost*sum(pmax(1-y*unlist(lapply(X,function(x) sum(B*x)+b0)),0)
    ))
}

#
# SMM = function(X,y,r,cost = 10){
#    result = list()
#    error = 10
#    iter = 0
#    # SMM
#    m= nrow(X[[1]]); n = ncol(X[[1]])
#
#    #initialization
#    U = randortho(m)[,1:r]
#    # U = matrix(runif(m*r,-1,1),nrow = m)
#    V = randortho(n)[,1:r]
#    # V = matrix(runif(n*r,-1,1),nrow = n)
#    obj = objv(U%*%t(V),0,X,y,cost);obj
#
#    while((iter <1000)&(error>10^-4)){
#       # update U fixing V
#       Vs = V%*%solve(t(V)%*%V)
```

```r
37  #        x = matrix(unlist(lapply(X,function(x) x%*%sqrtH(Vs,V))),nrow = length(X),
            byrow = T)
38  #        dat = data.frame(y= factor(y),x)
39  #        fit = svm(factor(y) ~ ., data = dat, scale = FALSE, kernel = "linear", cost=
            cost)
40  #        Bpart=matrix(t(fit$coefs)%*%matrix(unlist(X),nrow = length(X),byrow = T)[fit
            $index,],nrow = m)
41  #        U = Bpart%*%Vs;U
42  #
43  #
44  #        # update V fixing U
45  #        Us = U%*%solve(t(U)%*%U)
46  #        x = matrix(unlist(lapply(X,function(x) t(sqrtH(Us,U))%*%x)),nrow = length(X)
            ,byrow = T)
47  #        dat = data.frame(y= factor(y),x)
48  #        fit = svm(factor(y) ~ ., data = dat, scale = FALSE, kernel = "linear", cost=
            cost)
49  #        Bpart=matrix(t(fit$coefs)%*%matrix(unlist(X),nrow = length(X),byrow = T)[fit
            $index,],nrow = m)
50  #        V = t(Bpart)%*%Us;V
51  #
52  #
53  #        ## intercept estimation
54  #        Bhat = U%*%t(V);Bhat
55  #        b0hat = -(min(unlist(lapply(X,function(x) sum(Bhat*x)))[which(y==1)])+
56  #                    max(unlist(lapply(X,function(x) sum(Bhat*x)))[which(y==-1)]))/2
57  #        obj = c(obj,objv(Bhat,b0hat,X,y,cost));obj
58  #        iter = iter+1
59  #        error = abs(-obj[iter+1]+obj[iter])/obj[iter];error
60  #
61  #    }
62  #    predictor = function(x) sign(sum(Bhat*x)+b0hat)
63  #    result$B = Bhat; result$b0 = b0hat; result$obj = obj; result$iter = iter
64  #    result$error = error; result$predict = predictor
65  #    return(result)
66  # }


# Generating dataset
gendat = function(m,n,r,N,b0){
  result = list()
  # simulation
  # Weight
  rU = matrix(runif(m*r,-1,1),nrow = m)
  rV = matrix(runif(n*r,-1,1),nrow = n)
  B = rU%*%t(rV)

  # predictor matrix
  X = list()
  for (i in 1:N) {
    X[[i]] <- matrix(runif(m*n,-1,1),nrow = m,ncol=n)
  }

  # classification
  y = list()
  for (i in 1:N) {
    y[[i]] = sign(sum(B*X[[i]])+b0)
  }
  y = unlist(y)
```

```r
90
91    # predictor vector
92    x = matrix(nrow =N,ncol = m*n)
93    for(i in 1:N){
94      x[i,] = as.vector(X[[i]])
95    }
96    dat = data.frame(y = factor(y), x)
97
98    result$B
99    result$X = X; result$y = y; result$dat = dat
100   return(result)
101 }
102
103
104 kernelm = function(X,H,y,type = c("u","v")){
105   n = length(X)
106   x = matrix(unlist(X),nrow = length(X),byrow = T)
107   if (type == "u") {
108     hx = matrix(unlist(lapply(X,function(x) x%*%H)),nrow = length(X),byrow = T)
109   } else {
110     hx = matrix(unlist(lapply(X,function(x) H%*%x)),nrow = length(X),byrow = T)
111   }
112   Q = matrix(nrow = n,ncol = n)
113   for (i in 1:n) {
114     for(j in i:n){
115       Q[i,j] = sum(x[i,]*hx[j,])*y[i]*y[j]
116       Q[j,i] = Q[i,j]
117     }
118   }
119   h = eigen(Q)
120   Q = (h$vectors)%*%diag(pmax(h$values,eps))%*%t(h$vectors)
121   return(Q)
122 }
123
124
125 #
126 # obj = objv(U%*%t(V),0,X,y,cost);obj
127 #
128 #
129 # U = matrix(runif(m*r,-1,1),nrow = m)
130 # V = randortho(n)[,1:r]
131 # # V = matrix(runif(n*r,-1,1),nrow = n)
132 # Vs = V%*%solve(t(V)%*%V)
133 # H = Vs%*%t(V)
134 # dvec = rep(1,length(X))
135 # Dmat = kernelm(X,H,y,"u")
136 # Amat = cbind(y,-y,diag(1,N),-diag(1,N))
137 # bvec = c(rep(0,2+N),rep(-cost,N))
138 # result = solve.QP(Dmat,dvec,Amat,bvec)
139 # Bpart=matrix(t(y*result$solution)%*%matrix(unlist(X),nrow = length(X),byrow = T)
       ,nrow = m)
140 # U = Bpart%*%Vs;U
141 # obj = objv(U%*%t(V),0,X,y,cost);obj
142 #
143 # alph = rep(0,N)
144 # alph[fit$index] = y[fit$index]*fit$coefs
145 # t(alph)%*%Dmat%*%alph/2-sum(alph)
146
147 smm = function(X,y,r,cost = 10){
```

```
148    result = list()
149    error = 10
150    iter = 0
151    # SMM
152    m= nrow(X[[1]]); n = ncol(X[[1]]); N = length(X)
153
154    #initialization
155    U = randortho(m)[,1:r]
156    # U = matrix(runif(m*r,-1,1),nrow = m)
157    V = randortho(n)[,1:r]
158    # V = matrix(runif(n*r,-1,1),nrow = n)
159    obj = objv(U%*%t(V),0,X,y,cost);obj
160
161    while((iter <1000)&(error >10^-4)){
162      # update U fixing V
163      Vs = V%*%solve(t(V)%*%V)
164      H = Vs%*%t(V)
165      dvec = rep(1,length(X))
166      Dmat = kernelm(X,H,y,"u")
167      Amat = cbind(y,diag(1,N),-diag(1,N))
168      bvec = c(rep(0,1+N),rep(-cost,N))
169      alpha = solve.QP(Dmat,dvec,Amat,bvec,meq =1)
170      Bpart=matrix(t(y*alpha$solution)%*%matrix(unlist(X),nrow = length(X),byrow = T
       ),nrow = m)
171      U = Bpart%*%Vs;U
172
173
174      # update V fixing U
175      Us = U%*%solve(t(U)%*%U)
176      H = Us%*%t(U)
177      Dmat = kernelm(X,H,y,"v")
178      alpha = solve.QP(Dmat,dvec,Amat,bvec,meq = 1)
179      Bpart=matrix(t(y*alpha$solution)%*%matrix(unlist(X),nrow = length(X),byrow = T
       ),nrow = m)
180      V = t(Bpart)%*%Us;V
181
182
183      ## intercept estimation
184      Bhat = U%*%t(V);Bhat
185      b0hat = -(min(unlist(lapply(X,function(x) sum(Bhat*x)))[which(y==1)])+
186              max(unlist(lapply(X,function(x) sum(Bhat*x)))[which(y==-1)]))/2
187      obj = c(obj,objv(Bhat,b0hat,X,y,cost));obj
188      iter = iter+1
189      error = abs(-obj[iter+1]+obj[iter])/obj[iter];error
190
191    }
192    predictor = function(x) sign(sum(Bhat*x)+b0hat)
193    result$B = Bhat; result$b0 = b0hat; result$obj = obj; result$iter = iter
194    result$error = error; result$predict = predictor
195    return(result)
196 }
```

5.2 Simulations

```
1  source("SMMfunctions.R")
2
3
4  set.seed(18)
5  m = 20; n = 10; r = 5; N = 40; b0 = 0.1
```

```r
 6  result = gendat(m,n,r,N,b0)
 7  X = result$X; y = result$y; dat = result$dat
 8  B = result$B
 9
10
11  # SVM
12  dat = data.frame(y = factor(y), x)
13  fit = svm(factor(y) ~ ., data = dat, scale = FALSE, kernel = "linear", cost = 10)
14  fit$coefs
15  fit$rho
16  Bvec = t(fit$coefs)%*%fit$SV
17  hatB = matrix(Bvec,nrow = m,ncol = n)
18  #training result
19  length(which(y == predict(fit,dat[,-1])))/N
20
21
22  # SMM
23  result = smm(X,y,r,10);result
24  #traing result
25  length(which(unlist(lapply(X,result$predict))==y))/N
26
27
28  #obj value using svm
29  objv(hatB,fit$rho,X,y)
30  #obj value using smm
31  objv(result$B,result$b0,X,y)
32  #obj value at true B,b0
33  objv(B,b0,X,y)
34
35
36
37  # obj at true
38  objv(B,b0,X,y,10)
```