

SMM dual problem and simulations

Chanwoo Lee, April 02, 2020

1 SMM dual problem

Let us consider the following primal problem for SMM optimization.

$$(P) \quad \min_{U,V} \frac{1}{2} \|UV^T\|^2 + C \sum_{i=1}^N \xi_i$$

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

The Lagrange function with multiplier α and μ is

$$L_p = \frac{1}{2} \|UV^T\|^2 + C \sum_{i=1}^N \xi_i - \sum_{i=1}^N \alpha_i (y_i(\langle UV^T, X_i \rangle + b) - (1 - \epsilon_i)) - \sum_{i=1}^N \mu_i \epsilon_i, \quad (1)$$

which we minimize w.r.t. U, V, b , and ξ_i . Setting the respective derivatives to zero, we get

$$\begin{aligned} (V^T V)U &= \sum_{i=1}^N \alpha_i y_i X_i V, \\ (U^T U)V^T &= \sum_{i=1}^N \alpha_i y_i U^T X_i. \\ 0 &= \sum_{i=1}^N \alpha_i y_i, \\ \alpha_i &= C - \mu_i, \forall i. \end{aligned} \quad (2)$$

From this, we have

$$\begin{aligned} UV^T &= \left(\sum_{i=1}^N \alpha_i y_i X_i \right) H_V = H_U \left(\sum_{i=1}^N \alpha_i y_i X_i \right) \\ &= \sum_{i=1}^N \alpha_i y_i H_U X_i H_V, \end{aligned}$$

where $H_U = U(U^T U)^{-1} U^T$ and $H_V = V(V^T V)^{-1} V^T$.

By substituting (2) into (1), we obtain the Lagrangian dual objective function

$$\begin{aligned} L_d &= \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j \langle X_i H_V, H_U X_j \rangle \\ &= \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j \langle H_U X_i H_V, H_U X_j H_V \rangle \end{aligned} \quad (3)$$

where is the blue part? b = 0 ??

does strong duality hold?

We maximize L_d subject to $0 \leq \alpha_i \leq C$ and $\sum_{i=1}^N$. The dual problem (3) shows the intuition about how SMM model captures feature of predictor data X_i : SMM model projects X_i into row space and column space then use projected features as predictors. However, this dual problem is hard to find the optimizer α because U and V are unknown in L_d . Therefore, alternating updates for U and V fixing the other is reasonable.

2 Simulations

Our training data consists of N pairs $(X_1, y_1), \dots, (X_N, y_N)$, with $X_i \in \mathbb{R}^{10 \times 8}$ and $y_i \in \{-1, 1\}$. We define a hyper plane by $\{X : f(X) = \langle X, B \rangle + 0.1 = 0\}$ where the rank of B is five. A classification rule induced by $f(X)$ is $y_i = \text{sign}(f(X_i))$.

I perform three main simulations. In the first simulation, I check the consistency of SMM and SVM estimations. Figure 1 shows both SVM and SMM are consistent estimation because both estimations have small errors as N increases. In addition, we can check SMM outperforms SVM under B being low rank.

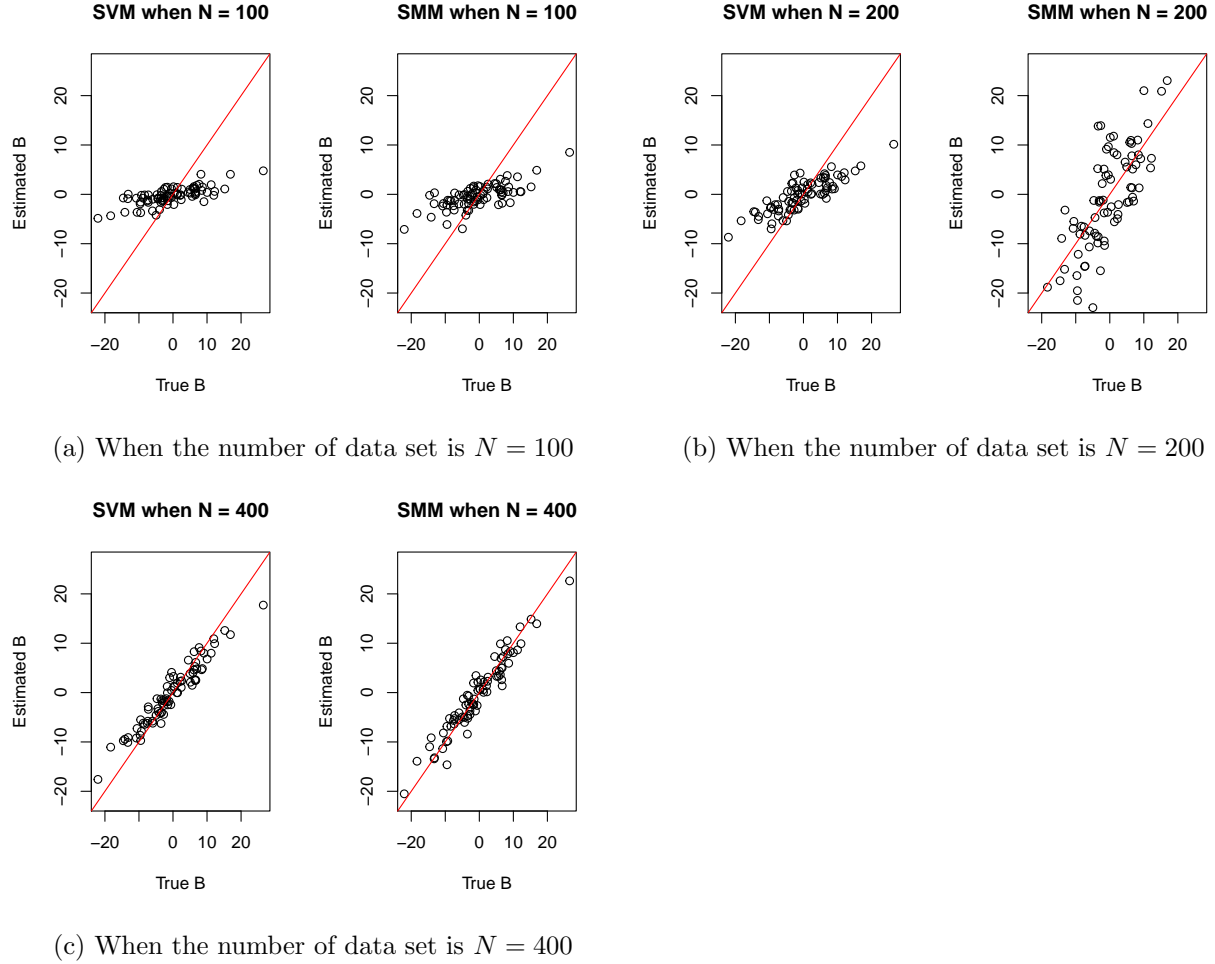


Figure 1: I compare true parameter B with estimated parameter \hat{B} under the several number of data sets $N \in \{100, 200, 400\}$. The horizontal axis is entries of B and the vertical axis is entries of \hat{B} . For each sub figure, the left figure is for SVM method and the right figure for SMM method.

In the second simulation, I check whether SMM method match with SVM when we assume B as a full rank matrix. The Figure 2 shows that SVM estimator and SMM estimator perfectly match each other under the full rank assumption.

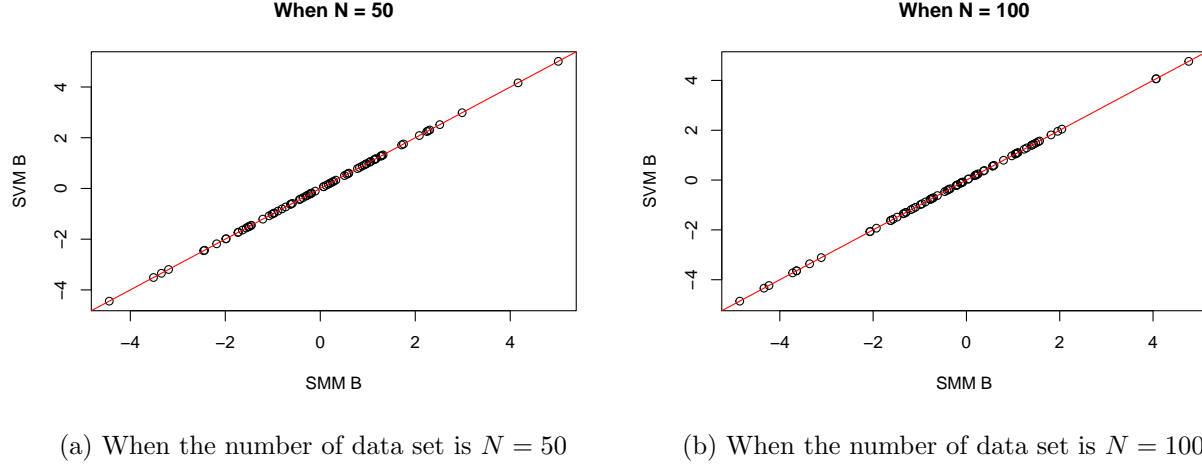


Figure 2: I compare SVM estimator \hat{B}_{svm} with SMM estimator \hat{B}_{smm} under the several number of data sets $N \in \{50, 100\}$. The horizontal axis is entries of \hat{B}_{smm} and the vertical axis is entries of \hat{B}_{svm} .

In the last simulation, I do 5 folded cross validation to check prediction performance. Simulation 1 generates a low-rank B and matrix ensembles $\{X_i\}$. I assign $y_i \in \{-1, 1\}$ based the rule, $y_i = \text{sign}(\langle X_i, B \rangle + b)$. Simulation 2 generates data set $(X_1, y_1), \dots, (X_{200}, y_{200})$ based on the following rule

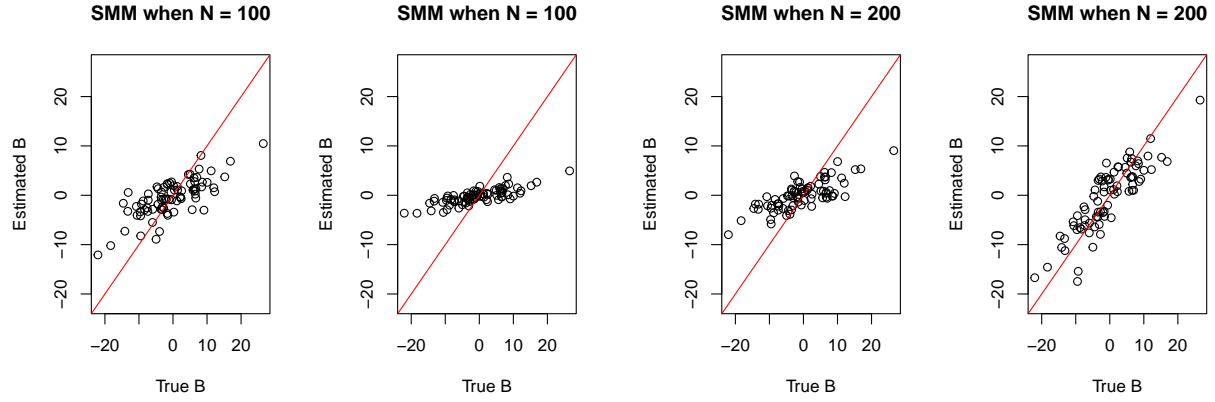
$$\{(X_i, 1) : X_i = u_1 v_1^T + E_i \quad i = 1, \dots, 100\} \text{ and } \{(X_j, -1) : X_j = u_2 v_2^T + E_i \quad j = 101, \dots, 200\},$$

whose entries of E_i i.i.d from $N(0, 4^2)$. This setting makes the data set inseparable space. The following table shows the cross validation results in Simulation 1 and 2. We can check SMM outperforms SVM when the data set is inseparable.

		CV1	CV2	CV3	CV4	CV5	Mean
Simulation 1	SVM	0.875	0.725	0.675	0.8	0.875	0.790
	SMM	0.725	0.800	0.725	0.8	0.875	0.785
Simulation 2	SVM	0.75	0.700	0.800	0.725	0.625	0.720
	SMM	0.75	0.725	0.775	0.725	0.700	0.735

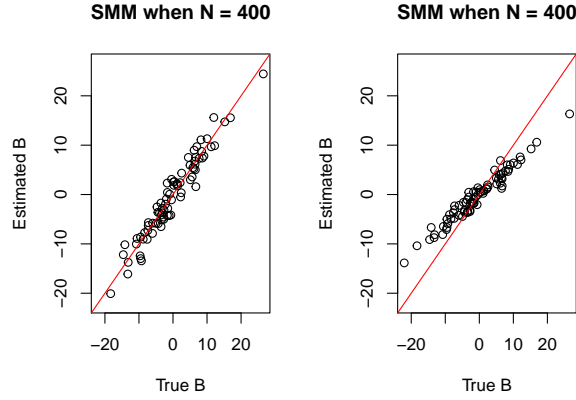
3 An issue for stability

One issue of the current algorithm is that the output depends on initial points quite a lot. Whenever I run the algorithm under the same settings, I get different output values. Figure 3 shows this issue well. Therefore, setting good initial points is needed.



(a) When the number of data set is $N = 100$

(b) When the number of data set is $N = 200$



(c) When the number of data set is $N = 400$

Figure 3: I compare true parameter B with the alternating algorithm for SMM result \hat{B} under the several number of data sets $N \in \{100, 200, 400\}$. The horizontal axis is entries of B and the vertical axis is entries of \hat{B} . For each sub figure, We can check that the output differs at each run.

4 R-codes

4.1 Updated R functions

```

1 library(pracma)
2 library(quadprog)
3
4 eps = 10^-5
5
6 sqrtH = function(Us,U){
7   h = eigen(Us%*%t(U))
8   return(h$vectors%*%diag(sqrt(pmax(h$values,eps))))
9 }
10
11 objv = function(B,b0,X,y,cost = 10){
12   return(sum(B*B)/2+cost*sum(pmax(1-y*unlist(lapply(X,function(x) sum(B*x)+b0)),0)
13   ))
14 }

```

```

14
15 # Generating dataset
16 gendat = function(m,n,r,N,b0){
17   result = list()
18   # simulation
19   # Weight
20   rU = matrix(runif(m*r,-1,1),nrow = m)
21   rV = matrix(runif(n*r,-1,1),nrow = n)
22   B = rU%*%t(rV)
23
24   # predictor matrix
25   X = list()
26   for (i in 1:N) {
27     X[[i]] <- matrix(runif(m*n,-1,1),nrow = m,ncol=n)
28   }
29
30   # classification
31   y = list()
32   for (i in 1:N) {
33     y[[i]] = sign(sum(B*X[[i]]))+b0)
34   }
35   y = unlist(y)
36
37   # predictor vector
38   x = matrix(nrow =N,ncol = m*n)
39   for(i in 1:N){
40     x[i,] = as.vector(X[[i]])
41   }
42   dat = data.frame(y = factor(y), x)
43
44   result$B = B
45   result$X = X; result$y = y; result$dat = dat
46   return(result)
47 }
48
49
50 kernelm = function(X,H,y,type = c("u","v")){
51   n = length(X)
52   x = matrix(unlist(X),nrow = length(X),byrow = T)
53   if (type == "u") {
54     hx = matrix(unlist(lapply(X,function(x) x%*%H)),nrow = length(X),byrow = T)
55   } else {
56     hx = matrix(unlist(lapply(X,function(x) H%*%x)),nrow = length(X),byrow = T)
57   }
58   Q = matrix(nrow = n,ncol = n)
59   for (i in 1:n) {
60     for(j in i:n){
61       Q[i,j] = sum(x[i,]*hx[j,])*y[i]*y[j]
62       Q[j,i] = Q[i,j]
63     }
64   }
65   h = eigen(Q)
66   Q = (h$vectors)%*%diag(pmax(h$values,eps))%*%t(h$vectors)
67   return(Q)
68 }
69
70
71
72 smm = function(X,y,r,cost = 10){

```

```

73 result = list()
74 error = 10
75 iter = 0
76 # SMM
77 m= nrow(X[[1]]); n = ncol(X[[1]]); N = length(X)
78
79 #initialization
80 U = randortho(m)[,1:r]
81 # U = matrix(runif(m*r,-1,1),nrow = m)
82 V = randortho(n)[,1:r]
83 # V = matrix(runif(n*r,-1,1),nrow = n)
84 obj = objv(U%*%t(V),0,X,y,cost);obj
85
86 while((iter <20)&(error>10^-4)){
87   # update U fixing V
88   Vs = V%*%solve(t(V)%*%V)
89   H = Vs%*%t(V)
90   dvec = rep(1,length(X))
91   Dmat = kernelm(X,H,y,"u")
92   Amat = cbind(y,diag(1,N),-diag(1,N))
93   bvec = c(rep(0,1+N),rep(-cost,N))
94   alpha = solve.QP(Dmat,dvec,Amat,bvec,meq=1)
95   Bpart=matrix(t(y*alpha$solution)%*%matrix(unlist(X),nrow = length(X),byrow = T
96   ),nrow = m)
97   U = Bpart%*%Vs;U
98
99   # update V fixing U
100   Us = U%*%solve(t(U)%*%U)
101   H = Us%*%t(U)
102   Dmat = kernelm(X,H,y,"v")
103   alpha = solve.QP(Dmat,dvec,Amat,bvec,meq = 1)
104   Bpart=matrix(t(y*alpha$solution)%*%matrix(unlist(X),nrow = length(X),byrow = T
105   ),nrow = m)
106   V = t(Bpart)%*%Us;V
107
108   ## intercept estimation
109   Bhat = U%*%t(V);Bhat
110   b0hat = -(min(unlist(lapply(X,function(x) sum(Bhat*x)))[which(y==1)]))+
111             max(unlist(lapply(X,function(x) sum(Bhat*x)))[which(y==1)]))/2
112   obj = c(obj,objv(Bhat,b0hat,X,y,cost));obj
113   iter = iter+1
114   error = abs(-obj[iter+1]+obj[iter])/obj[iter];error
115
116 }
117 predictor = function(x) sign(sum(Bhat*x)+b0hat)
118 result$B = Bhat; result$b0 = b0hat; result$obj = obj; result$iter = iter
119 result$error = error; result$predict = predictor
120 return(result)
121 }
122
123
124 svm = function(X,y,cost = 10){
125   result = list()
126   error = 10
127   iter = 0
128   # SVM
129   m= nrow(X[[1]]); n = ncol(X[[1]]); N = length(X)

```

```

130
131 H = diag(1,n)
132 dvec = rep(1,length(X))
133 Dmat = kernelm(X,H,y,"u")
134 Amat = cbind(y,diag(1,N),-diag(1,N))
135 bvec = c(rep(0,1+N),rep(-cost,N))
136 alpha = solve.QP(Dmat,dvec,Amat,bvec,meq =1)
137 Bhat=matrix(t(y*alpha$solution)%*%matrix(unlist(X),nrow = length(X),byrow = T),
138           nrow = m)
139 b0hat = -(min(unlist(lapply(X,function(x) sum(Bhat*x)))[which(y==1)]))+
140          max(unlist(lapply(X,function(x) sum(Bhat*x)))[which(y==-1)]))/2
141 obj = objv(Bhat,b0hat,X,y,cost)
142 predictor = function(x) sign(sum(Bhat*x)+b0hat)
143 result$B = Bhat; result$b0 = b0hat; result$obj = obj;
144 result$predict = predictor
145 return(result)
146 }

```

4.2 Simulation codes

```

1 source("SMMfunctions.R")
2 set.seed(1818)
3 m = 10; n = 8; r = 5; N = 800; b0 = 0.1
4 result = gendat(m,n,r,N,b0)
5 X = result$X; y = result$y; dat = result$dat
6 B = result$B
7 k = 400
8 svmres = svm(X[1:k],y[1:k])
9 smmres = smm(X[1:k],y[1:k],5);smmres
10
11 lim = c(min(B/b0),max(B/b0))
12
13 par(mfrow = c(1,2))
14 plot(B/b0,svmres$B/abs(svmres$b0),xlab = "True B",ylab = "Estimated B",main =
15      paste("SVM when N =", k),
16      xlim = lim, ylim = lim)
17 plot(B/b0,smmres$B/abs(smmres$b0),xlab = "True B",ylab = "Estimated B",main =
18      paste("SMM when N =", k),
19      xlim = lim, ylim = lim)
20
21
22 #### consistency test
23 k = 50
24 par(mfrow = c(1,1))
25 svmres = svm(X[1:k],y[1:k])
26 fsmmres = smm(X[1:k],y[1:k],8)
27 plot(fsmmres$B/abs(fsmmres$b0),svmres$B/abs(svmres$b0),xlab = "SMM B",ylab = "SVM
28      B",main = paste("When N =", k),
29      xlim = c(min(fsmmres$B/abs(fsmmres$b0)),max(fsmmres$B/abs(fsmmres$b0))),
30      ylim = c(min(svmres$B/abs(svmres$b0)),max(svmres$B/abs(svmres$b0))))
31 abline(0,1,col = "red")
32
33
34
35 #### Stability test

```

```

36 k = 100
37 smmres = smm(X[1:k],y[1:k],5);smmres
38
39 lim = c(min(B/b0),max(B/b0))
40
41 par(mfrow = c(1,2))
42 plot(B/b0,smmres$B/abs(smmres$b0),xlab = "True B",ylab = "Estimated B",main =
    paste("SMM when N =", k),
43       xlim = lim, ylim = lim)
44 abline(0,1,col = "red")
45
46
47 smmres = smm(X[1:k],y[1:k],5);smmres
48 plot(B/b0,smmres$B/abs(smmres$b0),xlab = "True B",ylab = "Estimated B",main =
    paste("SMM when N =", k),
49       xlim = lim, ylim = lim)
50 abline(0,1,col = "red")

```