SMM dual problem and simulations

Chanwoo Lee, April 02, 2020

1 SMM dual problem

Let us consider the following primal problem for SMM optimization.

$$
\begin{aligned}
(P) \quad & \min_{U,V} \frac{1}{2}\|UV^T\|^2 + C\sum_{i=1}^N \xi_i \\
\text{subject to} \quad & y_i(\langle UV^T, X_i\rangle + b) \geq 1 - \xi_i, \\
& \xi_i \geq 0, \quad i = 1, \ldots, N.
\end{aligned}
$$

The Laglange function with multiplier $\boldsymbol{\alpha}$ and $\boldsymbol{\mu}$ is

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

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

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

From this, we have

$$
\begin{aligned}
UV^T &= \left(\sum_{i=1}^N \alpha_i y_i X_i\right) H_V = H_U \left(\sum_{i=1}^N \alpha_i y_i X_i\right) \\
&= \sum_{i=1}^N \alpha_i y_i H_U X_i H_V, \\
&\text{where } H_U = U(U^TU)^{-1}U^T \text{ and } H_V = V(V^TV)^{-1}V^T.
\end{aligned}
$$

Using this UV^T expression, we have

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

$$\begin{aligned}\sum_{i=1}^N \alpha_i y_i \langle UV^T, X_i\rangle &= \langle UV^T, \sum_{i=1}^N \alpha_i y_i X_i\rangle = \langle (\sum_{i=1}^N \alpha_i y_i X_i) H_V, \sum_{j=1}^N \alpha_j y_j X_j\rangle \\ &= \langle (\sum_{i=1}^N \alpha_i y_i X_i) H_V, (\sum_{i=1}^N \alpha_i y_i X_i) H_V\rangle \\ &= \langle (\sum_{i=1}^N \alpha_i y_i X_i) H_V, H_U(\sum_{i=1}^N \alpha_i y_i X_i)\rangle \\ &= \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j \langle X_i H_V, H_U X_i\rangle.\end{aligned}$$

By substituting (3) and (2) into (1), we obtain the Lagrangian dual objective function

$$\begin{aligned}L_d &= \sum_{i=1}^N \alpha_i - \frac{1}{2}\sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j \langle X_i H_V, H_U X_j\rangle \\ &= \sum_{i=1}^N \alpha_i - \frac{1}{2}\sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j \langle H_U X_i H_V, H_U X_j H_V\rangle\end{aligned} \tag{4}$$

We maximize L_d subject to $0 \leq \alpha_i \leq C$ and $\sum_{i=1}^N$. The dual problem (4) 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 $\boldsymbol{\alpha}$ 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 $(X_1, y_1), \ldots, (X_N, y_N)$, 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.

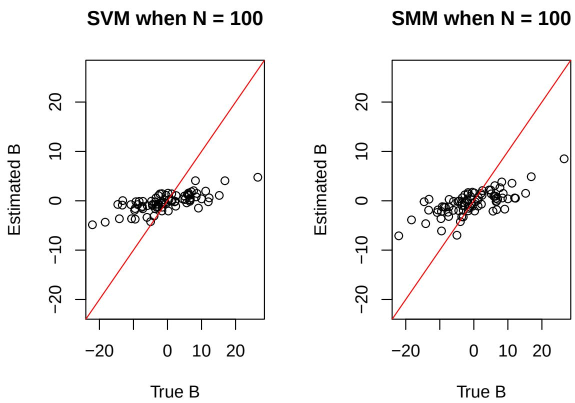
In the second simulation, I check whether SMM method match with SVM when when we assume B as a full rank matrix. The Figure 2 shows that SVM estimator and SMM estimator perfectly match each other under the full rank assumption.

(1) (For both simulation 1 & 2.) Design a way to visualize the classification results in a particular realization.
(Hint: the visualization should be as informative as your figure 1 in note "033020_SMM_implementation.pdf")

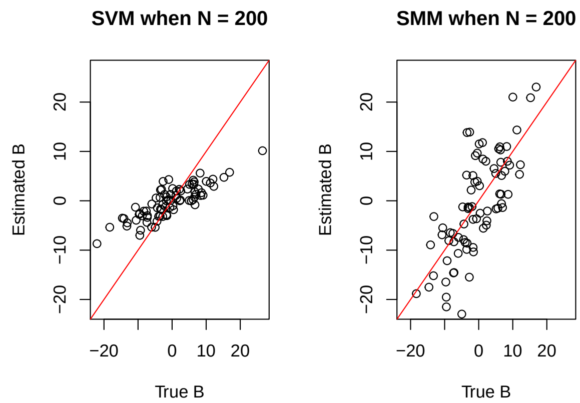
In the last simulation, I do 5 folded cross validation to check prediction performance. Simulation 1 generates a low-rank B and matrix ensembles $\{X_i\}$. I assign $y_i \in \{-1, 1\}$ based the rule, $y_i = \text{sign}(\langle X_i, B\rangle + b)$. Simulation 2 generates data set $(X_1, y_1), \ldots,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

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

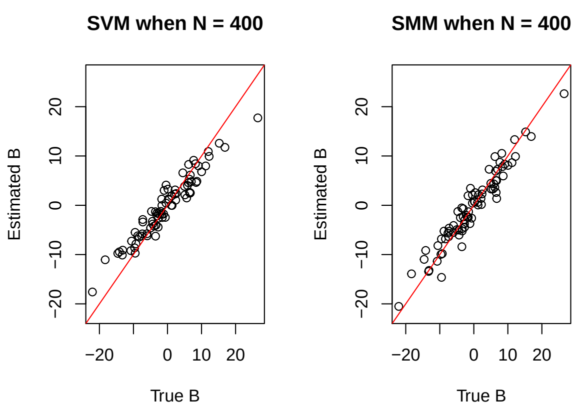
(2) What rank did you choose for Simulation 2? What is the ground truth rank?



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



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



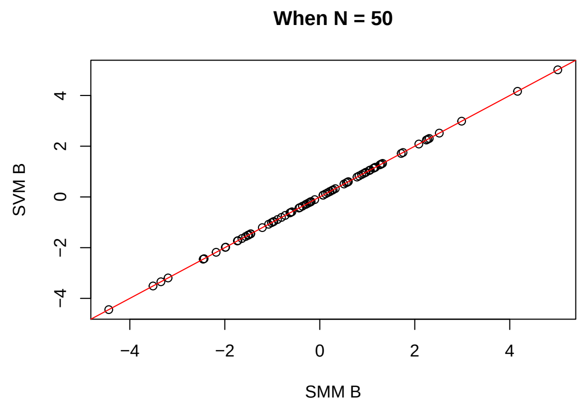
(c) When the number of data set is $N = 400$

Figure 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.

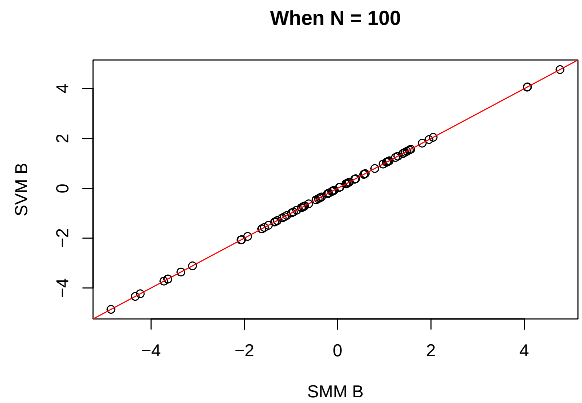
whose entries of E_i i.i.d from $N(0, 4^2)$. This setting makes the data set inseparable space The following table shows the cross validation results in Simulation 1 and 2. We can check SMM outperforms SVM when the data set is inseparable.

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 $N = 50$



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

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

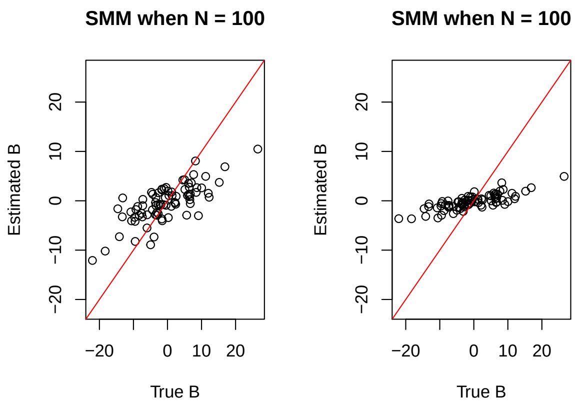
		CV1	CV2	CV3	CV4	CV5	Mean
Simulation 1	SVM	0.875	0.725	0.675	0.8	0.875	0.790
	SMM	0.725	0.800	0.725	0.8	0.875	0.785
Simulation 2	SVM	0.75	0.700	0.800	0.725	0.625	0.720
	SMM	0.75	0.725	0.775	0.725	0.700	0.735

3. Redo the comparison using new code (multiple initializations).
Compared the results using (1) average classification accuracy, and (2) visualization that you designed earlier

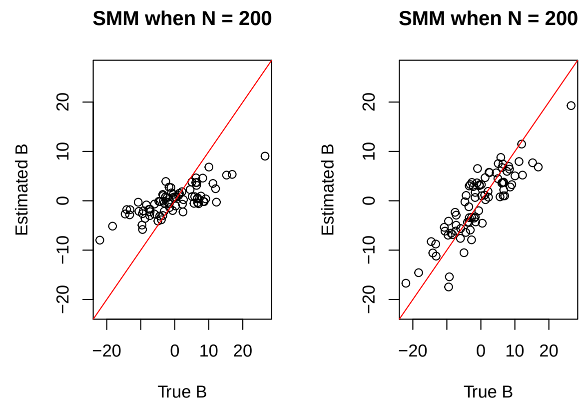
4 R-codes

4.1 Updated R functions

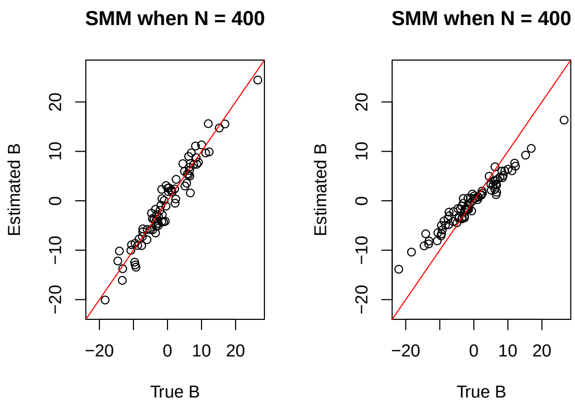
```
library(pracma)
library(quadprog)

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)
    ))
}

# 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)

```



(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.

```
# 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)

# predictor vector
x = matrix(nrow =N,ncol = m*n)
for(i in 1:N){
  x[i,] = as.vector(X[[i]])
}
```

```
  dat = data.frame(y = factor(y), x)

  result$B = B
  result$X = X; result$y = y; result$dat = dat
  return(result)
}


kernelm = function(X,H,y,type = c("u","v")){
  n = length(X)
  x = matrix(unlist(X),nrow = length(X),byrow = T)
  if (type == "u") {
    hx = matrix(unlist(lapply(X,function(x) x%*%H)),nrow = length(X),byrow = T)
  } else {
    hx = matrix(unlist(lapply(X,function(x) H%*%x)),nrow = length(X),byrow = T)
  }
  Q = matrix(nrow = n,ncol = n)
  for (i in 1:n) {
    for(j in i:n){
      Q[i,j] = sum(x[i,]*hx[j,])*y[i]*y[j]
      Q[j,i] = Q[i,j]
    }
  }
  h = eigen(Q)
  Q = (h$vectors)%*%diag(pmax(h$values,eps))%*%t(h$vectors)
  return(Q)
}



smm = function(X,y,r,cost = 10){
  result = list()
  error = 10
  iter = 0
  # SMM
  m= nrow(X[[1]]); n = ncol(X[[1]]); N = length(X)

  #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 <20)&(error>10^-4)){
    # update U fixing V
    Vs = V%*%solve(t(V)%*%V)
    H = Vs%*%t(V)
    dvec = rep(1,length(X))
    Dmat = kernelm(X,H,y,"u")
    Amat = cbind(y,diag(1,N),-diag(1,N))
    bvec = c(rep(0,1+N),rep(-cost,N))
    alpha = solve.QP(Dmat,dvec,Amat,bvec,meq =1)
    Bpart=matrix(t(y*alpha$solution)%*%matrix(unlist(X),nrow = length(X),byrow = T
    ),nrow = m)
    U = Bpart%*%Vs;U


    # update V fixing U
```

```
    Us = U%*%solve(t(U)%*%U)
    H = Us%*%t(U)
    Dmat = kernelm(X,H,y,"v")
    alpha = solve.QP(Dmat,dvec,Amat,bvec,meq = 1)
    Bpart=matrix(t(y*alpha$solution)%*%matrix(unlist(X),nrow = length(X),byrow = T
    ),nrow = m)
    V = t(Bpart)%*%Us;V


    ## intercept estimation
    Bhat = U%*%t(V);Bhat
    b0hat = -(min(unlist(lapply(X,function(x) sum(Bhat*x)))[which(y==1)])+
               max(unlist(lapply(X,function(x) sum(Bhat*x)))[which(y==-1)]))/2
    obj = c(obj,objv(Bhat,b0hat,X,y,cost));obj
    iter = iter+1
    error = abs(-obj[iter+1]+obj[iter])/obj[iter];error

  }
  predictor = function(x) sign(sum(Bhat*x)+b0hat)
  result$B = Bhat; result$b0 = b0hat; result$obj = obj; result$iter = iter
  result$error = error; result$predict = predictor
  return(result)
}


svm = function(X,y,cost = 10){
  result = list()
  error = 10
  iter = 0
  # SVM
  m= nrow(X[[1]]); n = ncol(X[[1]]); N = length(X)

  H = diag(1,n)
  dvec = rep(1,length(X))
  Dmat = kernelm(X,H,y,"u")
  Amat = cbind(y,diag(1,N),-diag(1,N))
  bvec = c(rep(0,1+N),rep(-cost,N))
  alpha = solve.QP(Dmat,dvec,Amat,bvec,meq =1)
  Bhat=matrix(t(y*alpha$solution)%*%matrix(unlist(X),nrow = length(X),byrow = T),
    nrow = m)
  b0hat = -(min(unlist(lapply(X,function(x) sum(Bhat*x)))[which(y==1)])+
             max(unlist(lapply(X,function(x) sum(Bhat*x)))[which(y==-1)]))/2
  obj = objv(Bhat,b0hat,X,y,cost)

  predictor = function(x) sign(sum(Bhat*x)+b0hat)
  result$B = Bhat; result$b0 = b0hat; result$obj = obj;
  result$predict = predictor
  return(result)
}
```

4.2 Simulation codes

```
source("SMMfunctions.R")
set.seed(1818)
m = 10; n = 8; r = 5; N = 800; b0 = 0.1
result = gendat(m,n,r,N,b0)
X = result$X; y = result$y; dat = result$dat
B = result$B
k = 400
```

```r
svmres = svm(X[1:k],y[1:k])
smmres = smm(X[1:k],y[1:k],5);smmres

lim = c(min(B/b0),max(B/b0))

par(mfrow = c(1,2))
plot(B/b0,svmres$B/abs(svmres$b0),xlab = "True B",ylab = "Estimated B",main = paste("SVM when N =", k),
     xlim = lim, ylim = lim)
abline(0,1,col = "red")
plot(B/b0,smmres$B/abs(smmres$b0),xlab = "True B",ylab = "Estimated B",main = paste("SMM when N =", k),
     xlim = lim, ylim = lim)
abline(0,1,col = "red")


#### consistency test
k = 50
par(mfrow = c(1,1))
svmres = svm(X[1:k],y[1:k])
fsmmres = smm(X[1:k],y[1:k],8)
plot(fsmmres$B/abs(fsmmres$b0),svmres$B/abs(svmres$b0),xlab = "SMM B",ylab = "SVM B",main = paste("When N =", k),
     xlim = c(min(fsmmres$B/abs(fsmmres$b0)),max(fsmmres$B/abs(fsmmres$b0))),
     ylim = c(min(svmres$B/abs(svmres$b0)),max(svmres$B/abs(svmres$b0))))
abline(0,1,col = "red")




#### Stability test
k = 100
smmres = smm(X[1:k],y[1:k],5);smmres

lim = c(min(B/b0),max(B/b0))

par(mfrow = c(1,2))
plot(B/b0,smmres$B/abs(smmres$b0),xlab = "True B",ylab = "Estimated B",main = paste("SMM when N =", k),
     xlim = lim, ylim = lim)
abline(0,1,col = "red")


smmres = smm(X[1:k],y[1:k],5);smmres
plot(B/b0,smmres$B/abs(smmres$b0),xlab = "True B",ylab = "Estimated B",main = paste("SMM when N =", k),
     xlim = lim, ylim = lim)
abline(0,1,col = "red")
```