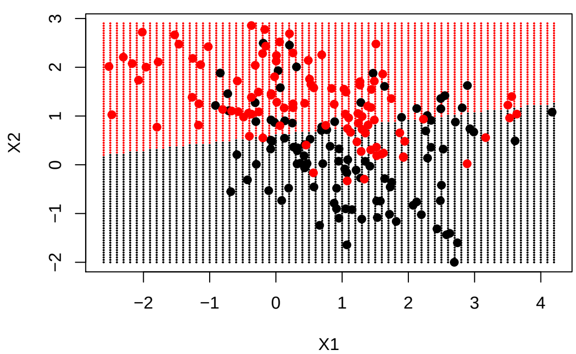
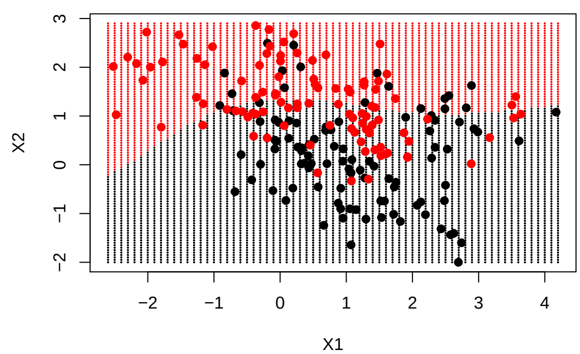
Support Matrix Machine Implementation

Chanwoo Lee, March 30, 2020

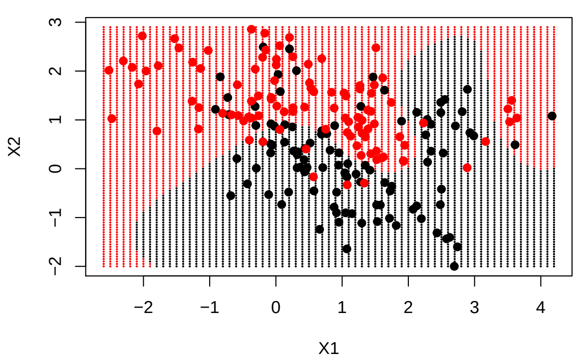
1 Kernel SVM discussion



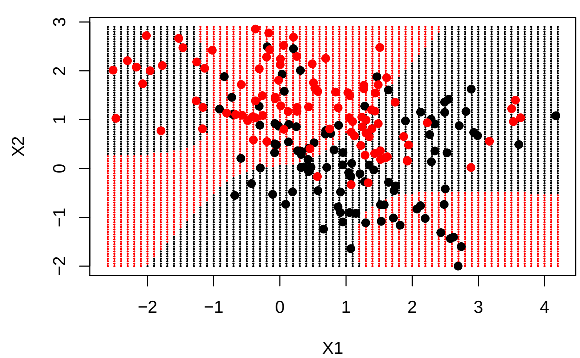
(a) Soft SVM method: MCR = 0.73



(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),

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

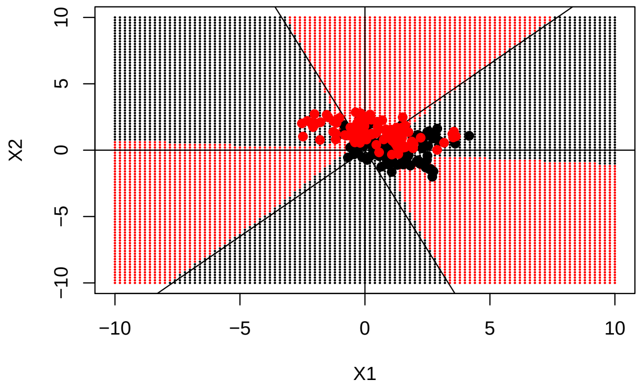


Figure 2: Sigmoid kernel classification. Range of x_1 and x_2 are $[-10, 10]$

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,

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

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

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

2. When fixing U,

$$(P_v) \qquad \begin{aligned} \min_{V,b,\boldsymbol{\xi}} \quad & \frac{1}{2}\|UV^T\|^2 + C\sum_{i=1}^{N}\xi_i \\ \text{subject to} \quad & y_i(\langle UV^T, X_i\rangle + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0, \quad i = 1, \cdots, N. \end{aligned} \tag{3}$$

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

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

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

which is the dual problem of the primer,

$$\begin{aligned} \min_{B,b,\boldsymbol{\xi}} & \frac{1}{2}\|\operatorname{Vec}(B)\|^2 + C\sum_{i=1}^{N}\xi_i \\ \text{subject to } & y_i(\langle \operatorname{Vec}(B), \operatorname{Vec}(X_i)\rangle + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0, \quad i = 1, \cdots, N. \end{aligned} \tag{4}$$

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

$$\begin{aligned} U &= \sum_{i=1}^{N}\alpha_i y_i X_i V(V^TV)^{-1} \quad \text{ in (2)}, \\ B &= \sum_{i=1}^{N}\alpha_i y_i X_i \quad \text{in (4)} , \end{aligned}$$

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 $(X_1G_V, y_1), \cdots, (X_NG_V, y_N)$ 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.

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

Therefore, I solve this constraint quadratic programming solution with `solve.QP` function in `quadprod` package. From this optimizer, I can update $U = \sum_{i=1}^N \alpha_i y_i X_i V(V^TV)^{-1}$. 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 $(D_u) : \max_{\boldsymbol{\alpha}} \sum_{i=1}^N \alpha_i - \frac{1}{2}\sum_{i=1}^N\sum_{j=1}^N \alpha_i\alpha_j y_i y_j\langle X_i, X_j H_V\rangle$.
    $U = \sum_{i=1}^N \alpha_i y_i X_i V(V^TV)^{-1}$.
  Update V fixing U :
    Solve $(D_v) : \max_{\boldsymbol{\alpha}} \sum_{i=1}^N \alpha_i - \frac{1}{2}\sum_{i=1}^N\sum_{j=1}^N \alpha_i\alpha_j y_i y_j\langle X_i, H_U X_j\rangle$.
    $V = \sum_{i=1}^N \alpha_i y_i X_i^T U(U^TU)^{-1}$.
Output: $B = UV^T$

4 R-codes

4.1 Functions

```
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)
#     x = matrix(unlist(lapply(X,function(x) x%*%sqrtH(Vs,V))),nrow = length(X),byrow = T)
#     dat = data.frame(y= factor(y),x)
#     fit = svm(factor(y) ~ ., data = dat, scale = FALSE, kernel = "linear", cost=cost)
#     Bpart=matrix(t(fit$coefs)%*%matrix(unlist(X),nrow = length(X),byrow = T)[fit$index,],nrow = m)
#     U = Bpart%*%Vs;U
#
#
#     # update V fixing U
#     Us = U%*%solve(t(U)%*%U)
#     x = matrix(unlist(lapply(X,function(x) t(sqrtH(Us,U))%*%x)),nrow = length(X),byrow = T)
#     dat = data.frame(y= factor(y),x)
#     fit = svm(factor(y) ~ ., data = dat, scale = FALSE, kernel = "linear", cost=cost)
#     Bpart=matrix(t(fit$coefs)%*%matrix(unlist(X),nrow = length(X),byrow = T)[fit
```

```
    $index,],nrow = m)
#      V = t(Bpart)%*%Us;V
#
#
#      ## intercept estimation
#      Bhat = U%*%t(V);Bhat
#      b0hat = -(min(unlist(lapply(X,function(x) sum(Bhat*x)))[which(y==1)])+
#                    max(unlist(lapply(X,function(x) sum(Bhat*x)))[which(y==-1)]))/2
#      obj = c(obj,objv(Bhat,b0hat,X,y,cost));obj
#      iter = iter+1
#      error = abs(-obj[iter+1]+obj[iter])/obj[iter];error
#
#    }
#    predictor = function(x) sign(sum(Bhat*x)+b0hat)
#    result$B = Bhat; result$b0 = b0hat; result$obj = obj; result$iter = iter
#    result$error = error; result$predict = predictor
#    return(result)
# }


# 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)

  # 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$B
  result$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)
}


#
# obj = objv(U%*%t(V),0,X,y,cost);obj
#
#
# U = matrix(runif(m*r,-1,1),nrow = m)
# V = randortho(n)[,1:r]
# # V = matrix(runif(n*r,-1,1),nrow = n)
# Vs = V%*%solve(t(V)%*%V)
# H = Vs%*%t(V)
# dvec = rep(1,length(X))
# Dmat = kernelm(X,H,y,"u")
# Amat = cbind(y,-y,diag(1,N),-diag(1,N))
# bvec = c(rep(0,2+N),rep(-cost,N))
# result = solve.QP(Dmat,dvec,Amat,bvec)
# Bpart=matrix(t(y*result$solution)%*%matrix(unlist(X),nrow = length(X),byrow = T)
  ,nrow = m)
# U = Bpart%*%Vs;U
# obj = objv(U%*%t(V),0,X,y,cost);obj
#
# alph = rep(0,N)
# alph[fit$index] = y[fit$index]*fit$coefs
# t(alph)%*%Dmat%*%alph/2-sum(alph)

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)
    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)
}
```

4.2 Simulations

```
source("SMMfunctions.R")


set.seed(18)
m = 20; n = 10; r = 5; N = 40; b0 = 0.1
result = gendat(m,n,r,N,b0)
X = result$X; y = result$y; dat = result$dat
B = result$B


# SVM
dat = data.frame(y = factor(y), x)
fit = svm(factor(y) ~ ., data = dat, scale = FALSE, kernel = "linear", cost = 10)
fit$coefs
fit$rho
Bvec = t(fit$coefs)%*%fit$SV
hatB = matrix(Bvec,nrow = m,ncol = n)
#training result
length(which(y == predict(fit,dat[,-1])))/N


# SMM
result = smm(X,y,r,10);result
```

```
#traing result
length(which(unlist(lapply(X,result$predict))==y))/N


#obj value using svm
objv(hatB,fit$rho,X,y)
#obj value using smm
objv(result$B,result$b0,X,y)
#obj value at true B,b0
objv(B,b0,X,y)



# obj at true
objv(B,b0,X,y,10)
```