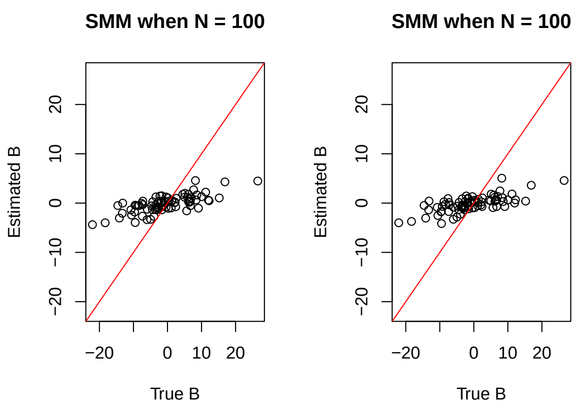
SMM kernel method and posterior distribution

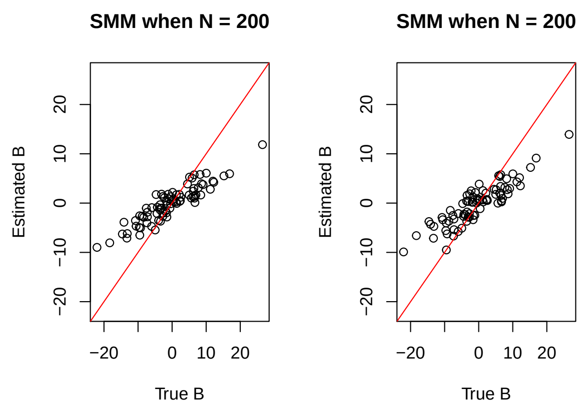
Chanwoo Lee, April 06, 2020

1 Instability issue of the SMM algorithm

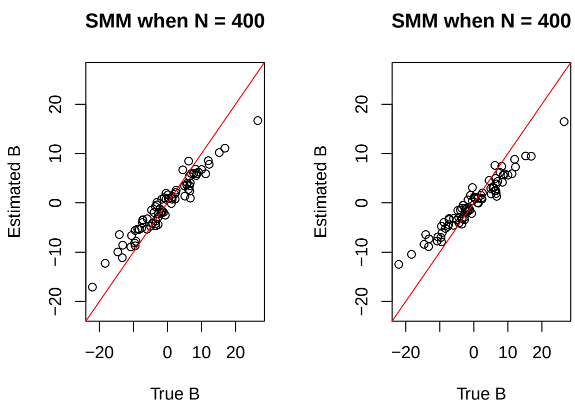
Multiple initialization can solve unstable issue of the SMM algorithm in the last meeting note. In the modified algorithm, we can set multiple initialization method. In this option, the algorithm choose the best output among multiple outputs in respect to loss function value. Figure 1 shows consistent outputs from repetitions.



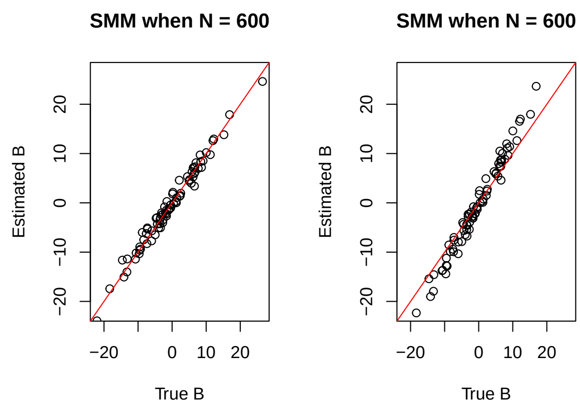
(a) When the number of data set is $N = 100$



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



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



(d) When the number of data set is $N = 600$

Figure 1: True parameter B is compared with multiple initialized SMM result $\hat{B}$ under the several number of data sets $N \in \{100, 200, 400, 600\}$. The horizontal axis is entries of B and the vertical axis is entries of $\hat{B}$. The number of initialization is 10. For each sub figure, we can check that the outputs are pretty much the same.

2 Kernel functions for matrices

We fit the SM classifier using input feature $h(X_i), i = 1, \ldots, N$. From this feature, we have the Lagrange dual problem

$$L_D = \sum_{i=1}^{N} - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \langle h(X_i), h(X_j) \rangle. \tag{1}$$

By solving (1), we obtain the nonlinear function $\hat{f}(X) = \sum_{i=1}^{N} \alpha_i y_i \langle h(X), h(X_i) \rangle$. Since all related equations require only knowledge of the kernel function,

$$K(X, X') = \langle h(X), h(X') \rangle.$$

Our goal is to define the kernel function which catches matrix structure well. In SVM case, three popular choices for K are

$$\begin{aligned} \text{dth-Degree polynomial}: \; & K(\boldsymbol{x}, \boldsymbol{x}') = (1 + \langle \boldsymbol{x}, \boldsymbol{x}' \rangle)^d, \\ \text{Radial basis}: \; & K(\boldsymbol{x}, \boldsymbol{x}') = \exp(-\gamma \|\boldsymbol{x} - \boldsymbol{x}'\|^2), \\ \text{Sigmoid}: \; & K(\boldsymbol{x}, \boldsymbol{x}') = \tanh(\gamma_1 \langle \boldsymbol{x}, \boldsymbol{x}' \rangle + \gamma_2). \end{aligned}$$

I define two measures to generalize $\langle \boldsymbol{x}, \boldsymbol{x}' \rangle$ and $\|\boldsymbol{x} - \boldsymbol{x}'\|^2$ into matrices case not vectorizing matrices. For two matrices $X, X' \in \mathbb{R}^{m \times n}$, we have singular value decomposition of two matrices.

$$X = \sum_{k=1}^{m \vee n} \sigma_k u_k v_k^T \quad \text{and} \quad X' = \sum_{k=1}^{m \vee n} \sigma'_k u'_k (v'_k)^T.$$

From this notation, I define weighted inner product between two matrices.

$$\langle X, X' \rangle_M = \sum_{k=1}^{m \vee n} \sigma_k \sigma'_k \langle u_k, u'_k \rangle \langle v_k, v'_k \rangle. \tag{2}$$

In (2), $\sigma\sigma'$ works as weight on principal inner products of subspace and $\langle u, u' \rangle, \langle v, v' \rangle$ represent principal inner product in column space and row space respectively. From this new definition, we can generalize d-th degree polynomial kernel and sigmoid kernel into matrices case.

$$\begin{aligned} \text{dth-Degree polynomial}: \; & K(X, X') = (1 + \langle X, X' \rangle_M)^d, \\ \text{Sigmoid}: & K(X, X') = \tanh(\gamma_1 \langle X, X' \rangle_M + \gamma_2). \end{aligned}$$

In addition to inner product, we can define weighted matrices distance as

$$\|X - X'\|_M^2 = \sum_{k=1}^{m \vee n} \sigma_k \sigma'_k (\|u_k - u'_k\|^2 + \|v_k - v'_k\|^2). \tag{3}$$

In (3), $\sigma\sigma'$ works as weight on principal row and column distances. $\|u_k - u'_k\|^2$ and $\|v_k - v'_k\|^2$ are column-wise and row-wise distances between principal vectors. With this definition we define generalized Radial basis kernel as

$$\text{Radial basis}: \; K(X, X') = \exp(-\gamma \|X - X'\|^2).$$

If two vectors $\boldsymbol{x}, \boldsymbol{x}'$ are expressed as

$$\boldsymbol{x} = \frac{\boldsymbol{x}}{\|\boldsymbol{x}\|}\|\boldsymbol{x}\| \cdot 1 \quad \text{and} \quad \boldsymbol{x}' = \frac{\boldsymbol{x}'}{\|\boldsymbol{x}'\|}\|\boldsymbol{x}'\| \cdot 1$$

We can check those definitions are consistent to vector case as follows

$$\langle \boldsymbol{x}, \boldsymbol{x}' \rangle_M = \|\boldsymbol{x}\|\|\boldsymbol{x}'\|\langle \frac{\boldsymbol{x}}{\|\boldsymbol{x}\|}, \frac{\boldsymbol{x}'}{\|\boldsymbol{x}'\|}\rangle\langle 1, 1\rangle = \langle \boldsymbol{x}, \boldsymbol{x}' \rangle,$$
$$\|\boldsymbol{x} - \boldsymbol{x}'\|_M = \|\boldsymbol{x}\|\|\boldsymbol{x}'\| \left(\left\| \frac{\boldsymbol{x}}{\|\boldsymbol{x}\|} - \frac{\boldsymbol{x}'}{\|\boldsymbol{x}'\|} \right\|_2 + \|1 - 1\| \right) = \|\boldsymbol{x} - \boldsymbol{x}'\|$$

3 Weighted binary classification

To obtain posterior distribution given feature data, we solve the regularization problem based on the weighted hinge loss.

$$\begin{aligned} \min_{\boldsymbol{\beta}, \boldsymbol{\xi}} \frac{1}{2}\|\boldsymbol{\beta}\|^2 + C \left[(1-\pi) \sum_{y_i = 1} \xi_i + \pi \sum_{y_i = -1} \xi_i \right] \\ \text{subject to } y_i(\langle x_i, \boldsymbol{\beta} \rangle + b) \geq 1 - \xi_i, \\ \xi_i \geq 0. \end{aligned} \tag{4}$$

The related dual problem for (4) is

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle, \\ \text{subject to } 0 \leq \alpha_i \leq C(1-\pi) \text{ for } y_i = 1, \\ 0 \leq \alpha_i \leq C\pi \text{ for } y_i = -1, \\ \sum_{i=1}^{N} \alpha_i y_i = 0. \end{aligned} \tag{5}$$

From the solution of (5), we can find the primal solution as $\boldsymbol{\beta} = \sum_{i=1}^{N} y_i \alpha_i \boldsymbol{x}_i$. With this relation, `smm` in R-codes section solves the equation (4). Figure 3 shows the weighted hinge loss SVM classifier with $\pi \in \{0.001, 0.5, 0.999\}$. It is known that the minimizer $\text{sign}(f_\pi(x))$ to Equation (4) is a consistent estimate of $\text{sign}(\mathbb{P}(y = 1|x) - \pi)$. Therefore solving Equation (4) using different π values such that $\pi_1 < \cdots < \pi_m$, we can estimate

$$\hat{\mathbb{P}}(y = 1|x) = \frac{1}{2} \left(\arg\max_{\pi_j}\{\text{sign}(f_{\pi_j}(x)) = 1\} + \arg\max_{\pi_j}\{\text{sign}(f_{\pi_j}(x)) = -1\} \right). \tag{6}$$

Figure 2 shows the posterior probability estimation with the rule of (6).

4 One issue for posterior estimation

I found one issue to estimate posterior probability $\mathbb{P}(y = 1|\boldsymbol{x})$. There are some points $\boldsymbol{x}_i$'s such that $\text{sign}(\mathbb{P}(y|\boldsymbol{x}_i) - \pi)$ is not decreasing in respect to π. We can check that the red point in Figure 3

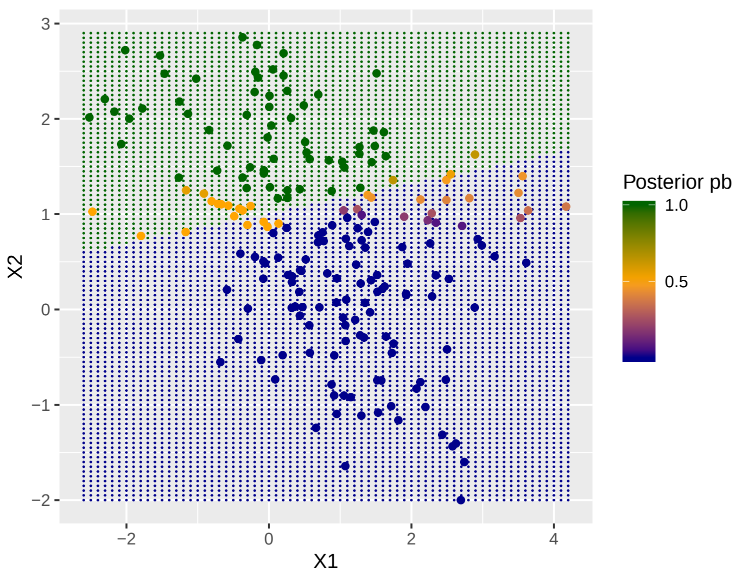


Figure 2: The green area is labeled as 1 with the SVM and blue area is -1. We can obtain non trivial posterior probability around the classification boundary.

has $\text{sign}(\mathbb{P}(y=1|\boldsymbol{x})-0.0001)=-1$ but $\text{sign}(\mathbb{P}(y=1|\boldsymbol{x})-0.9999)=1$ which does not make sense. This phenomenon happens in all points located below classification boundary when $\pi=0.0001$ and above the boundary when $\pi=0.9999$ at the same time. In addition, this area is inevitable unless two classification boundaries are parallel which is hard to be satisfied. If I stick to the rule in (6), all points in the area has 0.5 as posterior probability.

5 R-codes

5.1 Updated functions

```
library(pracma)
library(quadprog)

eps = 10^-5



objv = function(B,b0,X,y,cost = 10,prob = F){
  if (prob == F) {
    value = sum(B*B)/2+cost*sum(pmax(1-y*unlist(lapply(X,function(x) sum(B*x)+b0)),0))
  }else{
    ind = which(y==1)
    value = sum(B*B)/2 +
      (1-prob)*cost*sum(pmax(1-y[ind]*unlist(lapply(X[ind],function(x) sum(B*x)+b0)),0)) +
      prob*cost*sum(pmax(1-y[-ind]*unlist(lapply(X[-ind],function(x) sum(B*x)+b0)),0))

  }
  return(value)
}
```

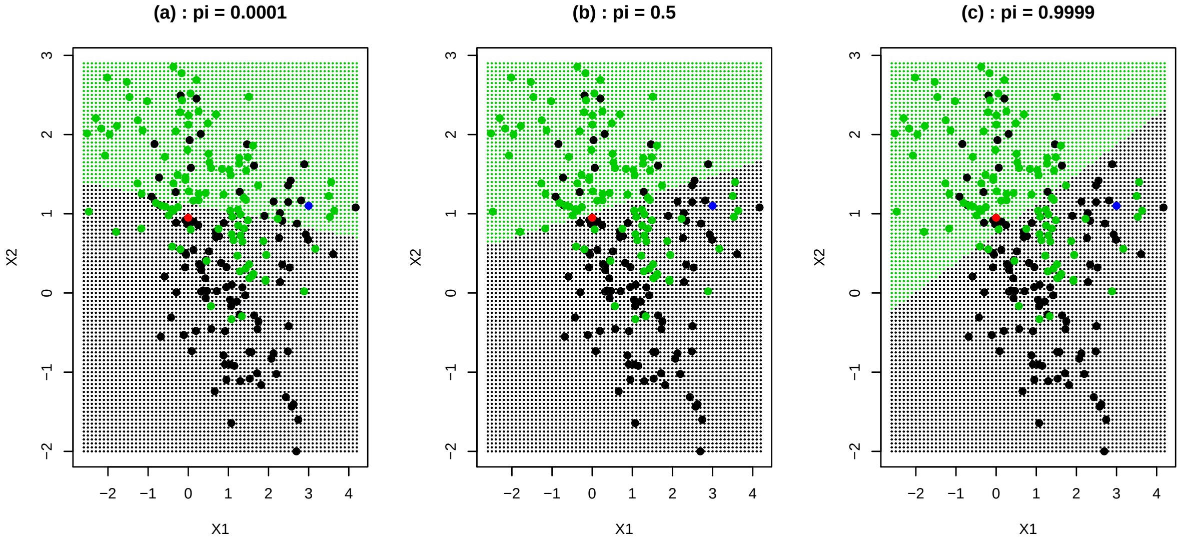


Figure 3: The sub figures show weighted hinge loss SVM classifier when $\pi = 0.001, 0.5, 0.999$. Green points represent for $y = 1$ and black for $y = -1$. The middle sub figure ($\pi = 0.5$) is the regular SVM. The red point changes its label from -1 to 1 as π increases. The blue point shows vice versa.

```

# 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$B = B
  result$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 with multiple initialization
smm = function(X,y,r,cost = 10,rep = 10){
  result = list()

  # SMM
  m= nrow(X[[1]]); n = ncol(X[[1]]); N = length(X)

  compareobj = 10^100
  for (i in 1:rep) {
    error = 10
    iter = 0
    #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^-3)){
      # 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


```

```
      # 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


      ## intercept estimation
      Bhat = U%*%t(V);Bhat
      positiv = min(unlist(lapply(X,function(x) sum(Bhat*x)))[which(y==1)])
      negativ = max(unlist(lapply(X,function(x) sum(Bhat*x)))[which(y==-1)])
      if ((1-positiv)<(-1-negativ)) {
        b0hat = -(positiv+negativ)/2
      }else{
        gridb0 = seq(from = -1-negativ,to = 1-positiv,length = 100)
        b0hat = gridb0[which.min(sapply(gridb0,function(b) objv(Bhat,b,X,y)))]
      }
      obj = c(obj,objv(Bhat,b0hat,X,y,cost));obj
      iter = iter+1
      error = abs(-obj[iter+1]+obj[iter])/obj[iter];error

    }
    if (compareobj>obj[iter+1]) {
      compareobj = obj[iter+1]
      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)
}


kernelmat = function(x,y,kernels = function(x1,x2) sum(x1*x2)){
  N = length(y)
  Q = matrix(nrow = N,ncol = N)
  for (i in 1:N) {
    for(j in i:N){
      Q[i,j] =kernels(x[i,],x[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)
}


# SVM with kernel functions and weighted cost function
svm = function(X,y,cost = 10, kernels = function(x1,x2) sum(x1*x2), p = .5){
  if (p==.5) {
    cost = 2*cost
  }
  result = list()
  error = 10
```

```
  iter = 0
  # SVM
  m= nrow(X[[1]]); n = ncol(X[[1]]); N = length(X)

  x = matrix(unlist(X),nrow = N,byrow = T)
  dvec = rep(1,length(X))
  Dmat = kernelmat(x,y,kernels)
  Amat = cbind(y,diag(1,N),-diag(1,N))
  bvec = c(rep(0,1+N),ifelse(y==1,-cost*(1-p),-cost*p))
  alpha = solve.QP(Dmat,dvec,Amat,bvec,meq =1)

  Bhat=matrix(t(y*alpha$solution)%*%x,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,prob = p)

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




posterior = function(X,y,cost = 10,test,kernels = function(x1,x2) sum(x1*x2)){
  a = 1:99
  for(i in 1:99){
    fit = svm(X,y,cost, kernels, p = i*0.01)$predict
    a[i] = fit(test)
  }
  if (all(a==1)) {
    return(1)
  }else if(all(a==-1)){
    return(0)
  }else{
    return((max(which(a==1))+min(which(a==-1)))/200)
  }
}
```

5.2 Simulations

```
load(file = "ESL.mixture.rda")
names(ESL.mixture)
rm(x,y)
attach(ESL.mixture)
y = ifelse(y==1,1,-1)
X = lapply(seq_len(nrow(x)),function(i) x[i,,drop = F])
par(mfrow = c(1,1))
plot(x, col = y + 3)
dat = data.frame(y = factor(y), x)



a1 = matrix(nrow = 2, ncol = 99)
a1[1,] = (1:99)*0.01
for(i in 1:99){
  fit = svm(X,y,cost, kernels, p = i*0.01)$predict
  a1[2,i] = fit(test)
```

```
}

a2 = matrix(nrow = 2, ncol = 99)
a2[1,] = (1:99)*0.01
for(i in 1:99){
  fit = svm(X,y,cost, kernels, p = i*0.01)$predict
  a2[2,i] = fit(test2)
}

a1



### Changing svm according to weight
par(mfrow = c(1,3))

fit = svm(X,y,cost = 10,p=0.0001)$predict
xgrid = expand.grid(X1 = px1, X2 = px2)
ygrid = apply(xgrid,1,fit)
plot(xgrid, col = as.numeric(ygrid+2), pch = 20, cex = .2,main = "(a) : pi =
    0.0001")
points(x, col = y + 2, pch = 19)
points(test,col = 'red',pch = 19)
points(test2,col = 'blue',pch = 19)

fit = svm(X,y,cost = 10,p=0.5)$predict
xgrid = expand.grid(X1 = px1, X2 = px2)
ygrid = apply(xgrid,1,fit)
plot(xgrid, col = as.numeric(ygrid+2), pch = 20, cex = .2,main = "(b) : pi = 0.5")
points(x, col = y + 2, pch = 19)
points(test,col = 'red',pch = 19)
points(test2,col = 'blue',pch = 19)


fit = svm(X,y,cost = 10,p=0.9999)$predict
xgrid = expand.grid(X1 = px1, X2 = px2)
ygrid = apply(xgrid,1,fit)
plot(xgrid, col = as.numeric(ygrid+2), pch = 20, cex = .2,main = "(c) : pi =
    0.9999")
points(x, col = y + 2, pch = 19);
points(test,col = "red",pch = 19)
points(test2,col = "blue",pch = 19)



### posterior
posterior(X,y,cost = 10,test)
posterior(X,y,cost = 10,test2)
yposterior = vector(length = 200)
for(i in 1:200){
  yposterior[i] = posterior(X,y,cost = 10,x[i,])
  print(paste(i,"th point is done lol"))
}
ypost = ifelse(yposterior==-1,0,yposterior)


par(mfrow = c(1,1))
xgrid = expand.grid(X1 = px1, X2 = px2)
ygrid = apply(xgrid,1,fit)
```

```r
datgrid = cbind(ygrid,xgrid)

colnames(x) = c("X1","X2 ")
realdat =as.data.frame(cbind(ypost,x))
breaks <- c(0.5,1,3.2)
ggplot(data = datgrid,aes(x = X1,y= X2,colour = ifelse(ygrid==-1,0,ygrid)))+geom_
    point(size = 0.001)+
  geom_point(data = realdat,aes(x = realdat[,2],y = realdat[,3],colour = ypost))+
  labs(colour = "Posterior pb")+
  scale_colour_gradientn(colours = c("darkblue","orange","darkgreen"),
                                                              breaks = breaks, labels =
    format(breaks))
```