vector case: Given a kernel, we know the mapping
h: R^m \maps to R^m'

SMM Kernel Method

Chanwoo Lee, April 23, 2020

Claim: if we define h_matrix in this way,
then the induced matrix-valued kernel satisfies
the properties in Remark 2

1 Kernel method

I am suggesting new kernel method which makes optimization easier. Define feature mapping $h : \mathbb{R}^{m\times n} \to \mathbb{R}^{m'\times n}$ where $m < m'$. Kernels for matrix case can be define as

h_matrix: R^{m\times n} —> to R^{m'\times n}

$$K(X, X') = h(X)^T h(X') \in \mathbb{R}^{n\times n}.$$

dX=[x1,…x_n] —> h_matrix(X)=[h(x1), … h(xn)]

Our objective primal problem is

here, h(.) is the classical mapping in vector space

$$\begin{aligned}
&\min_{U\in\mathbb{R}^{m'\times r}, V\in\mathbb{R}^{n\times r}, \boldsymbol{\xi}} \frac{1}{2}\|UV^T\|^2 + c\sum_{i=1}^N \xi_i \qquad (1)\\
&\text{subject to } y_i(\langle UV^T, h(X_i)\rangle + b) \leq 1 - \xi_i\\
&\xi_i \geq 0, \quad i = 1,\ldots,N.
\end{aligned}$$

First, fix V and solve (1) with respect to U. We have the following dual problem.

$$\begin{aligned}
&\min_{\boldsymbol{\alpha}} -\sum_{i=1}^N \alpha_i + \frac{1}{2}\sum_{i=1}^N\sum_{j=1}^N \alpha_i\alpha_j y_i y_j \langle h(X_i)H_V, h(X_j)H_V\rangle\\
&\text{subject to } \sum_{i=1}^N y_i\alpha_i = 0\\
&0 \leq \alpha_i \leq C, \quad i = 1,\ldots,N.
\end{aligned}$$

Notice $\langle h(X_i)H_V, h(X_j)H_V\rangle = \mathrm{tr}\left(H_V h(X_i)^T h(X_j)\right) = \mathrm{tr}\left(H_V K(X_i, X_j)\right)$. Therefore, we can update U as

$$U = \sum_{i=1}^N \alpha_i y_i h(X_i) V (V^T V)^{-1} \qquad (2)$$

where h function is not known. We are going to borrow this formula to update V in the next step. Now, fix U and solve (1) with respect to V. We have the following dual problem.

$$\begin{aligned}
&\min_{\boldsymbol{\beta}} -\sum_{i=1}^N \beta_i + \frac{1}{2}\sum_{i=1}^N\sum_{j=1}^N \beta_i\beta_j y_i y_j \langle H_U h(X_i), H_U h(X_j)\rangle \qquad (3)\\
&\text{subject to } \sum_{i=1}^N y_i\beta_i = 0\\
&0 \leq \beta_i \leq C, \quad i = 1,\ldots,N.
\end{aligned}$$

To get an optimal $\boldsymbol{\beta}$ in (3), we need the information of $\langle H_U h(X_i), H_U h(X_j)\rangle$. Notice

$$\begin{aligned}
\langle H_U h(X_i), H_U h(X_j)\rangle &= \mathrm{tr}\left(H_U h(X_j) h(X_i)^T\right) = \mathrm{tr}\left(U(U^TU)^{-1}U^T h(X_j)h(X_i)^T\right) \qquad (4)\\
&= \mathrm{tr}\left((U^TU)^{-1}U^T h(X_j)h(X_i)^T U\right)
\end{aligned}$$

Using the (2), we have the following expression of the component in (4).

$$
\begin{aligned}
U^T U &= \sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j (V^T V)^{-1} V^T h(X_i)^T h(X_j) V (V^T V)^{-1} \\
&= \sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j (V^T V)^{-1} V^T K(X_i, X_j) V (V^T V)^{-1} \\
&= (V^T V)^{-1} V^T \left(\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j K(X_i, X_j) \right) V (V^T V)^{-1},
\end{aligned} \tag{5}
$$

$$
\begin{aligned}
U^T h(X_j) &= \sum_{l=1}^{N} \alpha_l y_l (V^T V)^{-1} V^T h(X_l)^T h(X_j) \\
&= \sum_{l=1}^{N} \alpha_l y_l (V^T V)^{-1} V^T K(X_l, X_j) \\
&= (V^T V)^{-1} V^T \sum_{l=1}^{N} \alpha_l y_l K(X_l, X_j).
\end{aligned}
$$

Therefore, we can get an optimal $\boldsymbol{\beta}$ in (3) with (5). Finally, we update V with the help of Equation (5) as

$$
V^T = \sum_{i=1}^{N} \beta_i y_i (U^T U)^{-1} U^T h(X_i).
$$

Remark 1. We can get explicit update formula V while U cannot be expressed as explicit value. This does make sense because in feature mapping $h : \mathbb{R}^{m\times n} \to \mathbb{R}^{m'\times n}$, dimension m' can be increased to arbitrary dimension while n being fixed.

Remark 2. There are some kernel functions that might be used often.

$$
\begin{aligned}
\text{Linear:} \quad & K(X, X') = X^T X' \\
\text{Polynomial:} \quad & K(X, X') = (X^T X' + I_n)^d \\
\text{Radial:} \quad & K(X, X') = \exp\left((X - X')^T (X - X')/\sigma\right),
\end{aligned}
$$

where $\exp(A) = \sum_{n=0}^{\infty} \frac{1}{n!} A^n$. Notice that when $X \in \mathbb{R}^{m\times 1}$ i.e. X is a vector, all those definitions are reduced to SVM case. From this way, we can generalize linear SMM method to Kernel SMM with tractable algorithm.

2 Algorithm construction and simulation

2.1 Algorithm construction: objective function

Since we do not have explicit U formula in SMMK algorithm, we do not evaluate the following objective function in the SMM algorithm.

$$L(U,V,b) = \frac{1}{2}\|UV^T\|^2 + C\sum_{i=1}^{N}\left(1 - y_i(\langle UV^T, h(X_i)\rangle + b)\right)_+ \tag{6}$$

Instead, we are using the same objective function with different parameter,

$$L(V,\boldsymbol{\alpha},b) = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j \mathrm{tr}\left(H_V K(X_i,X_j)\right) + C\sum_{i=1}^{N}\left(1 - y_i(\sum_{j=1}^{N}\alpha_j y_j \mathrm{tr}\left(H_V K(X_i,X_j)\right) + b)\right)_+ . \tag{7}$$

This objective function in Equation (7) can be obtained by plugging $U = \sum_{i=1}^{N}\alpha_i y_i h(X_i)V(V^TV)^{-1}$ in Equation (6). In addition, I evaluate the objective value after updates of U because if we evaluate after updates of V, U expressed by V is changed resulting in wrong evaluation.

2.2 Simulation

I generate random matrix $X_1,\ldots,X_{100} \in \mathbb{R}^{5\times 4}$ whose entries are from i.i.d.Unif$(-1,1)$. I set ground truth matrix $B = UV^T \in \mathbb{R}^{5\times 4}$ such that $U \in \mathbb{R}^{5\times 3}, V \in \mathbb{R}^{4\times 3}$ whose entries are from i.i.d. Unif$(-1,1)$. Our classification rule is

$$y = \mathrm{sign}(\langle B, X\rangle + 0.1).$$

First, I perform five folded cross validation. Table 1 shows the miss classification rate (MCR) of SMM and SMMK (linear kernel) method at each test. Table 2 shows the loss function value at the last iteration at each test. This shows that two methods are pretty much the same. To be more accurate, only thing that makes two methods different is random initialization. I checked they have the same update of V from the same initialization.

	1st	2nd	3rd	4th	5th	average
SMMK(linear)	0.90	0.85	0.90	0.90	0.90	0.89
SMM	0.95	0.85	0.85	0.95	0.85	0.89

Table 1: Miss classification rate in Simulation

	1st	2nd	3rd	4th	5th	average
SMMK(linear)	32.41501	17.92673	18.18015	25.83209	24.44279	23.75935
SMM	22.31593	15.56181	17.99429	26.34058	32.12252	22.86703

Table 2: Loss function value in Simulation

2.3 Simulation with new kernel

I generate feature data matrix with the following rule.

$$
\begin{aligned}
\mathbb{P}\left((X_{\cdot 1}^T, X_{\cdot 2}^T)^T | y = 1\right) &\sim N((1,1,-1,-1)^T, I_4),\\
\mathbb{P}\left((X_{\cdot 1}^T, X_{\cdot 2}^T)^T | y = 1\right) &\sim N((0,0,0,0)^T, I_4).
\end{aligned}
$$

With this rule, we have a test data set $(X^1, 1), \ldots, (X^{20}, 1), (X^{21}, -1), \ldots, (X^{40}, -1)$.

$$
Z_1 = X_{11} + X_{21} \text{ and } Z_2 = X_{12} + X_{22}.
$$

Figure 1 shows how generated data looks like with $Z1$ and $Z2$ axes.

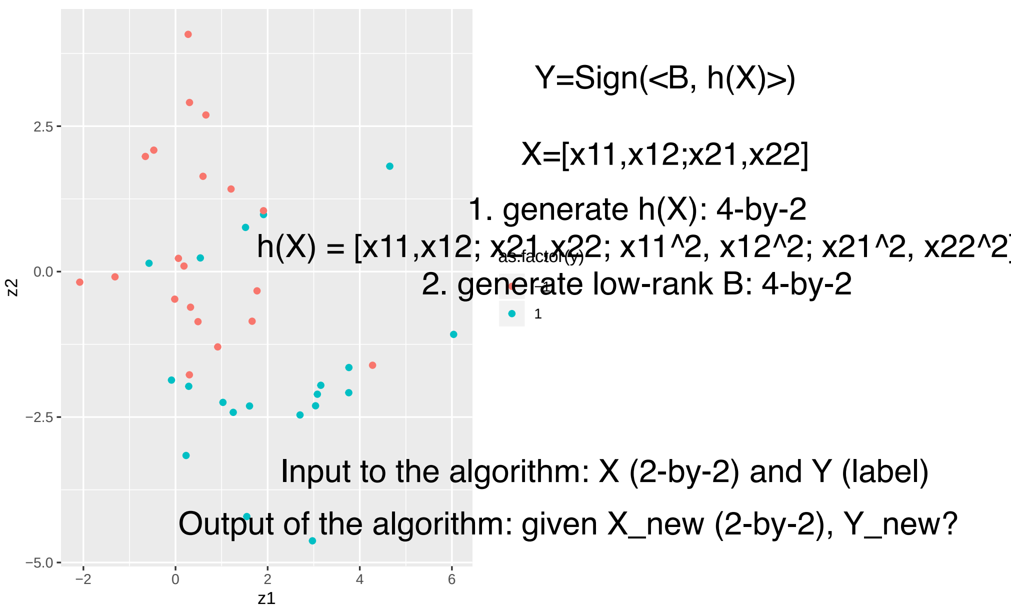

Figure 1: Visualization of the feature matrices. $Z1$ is the sum of the first column and $Z1$ is of the second one.

I implement SMMK method with linear kernel and polynomial kernel with degree three. I do not use exponential kernel I suggested above. The reason is that algorithm does not work well getting exploding objective values. I think that there is no feature h that corresponding to the exponential kernel. Figure 2 shows the classification results using the linear kernel and the polynomial kernel. One thing to note is that there are blurry part around the boundary in each figure. This is because for each point $(Z1, Z2)$, there are infinitely many possible combinations of $(X1, X2, X3, X4)$ which might give different value of y for each classifier.

3 Things to do

1. Construct algorithm that provides weighted loss function for conditional probability estimation.
2. Find good condition to make us easily check if given kernel is available i.e. has feature mapping such taht $K(X, X') = h(X)^T h(X')$.

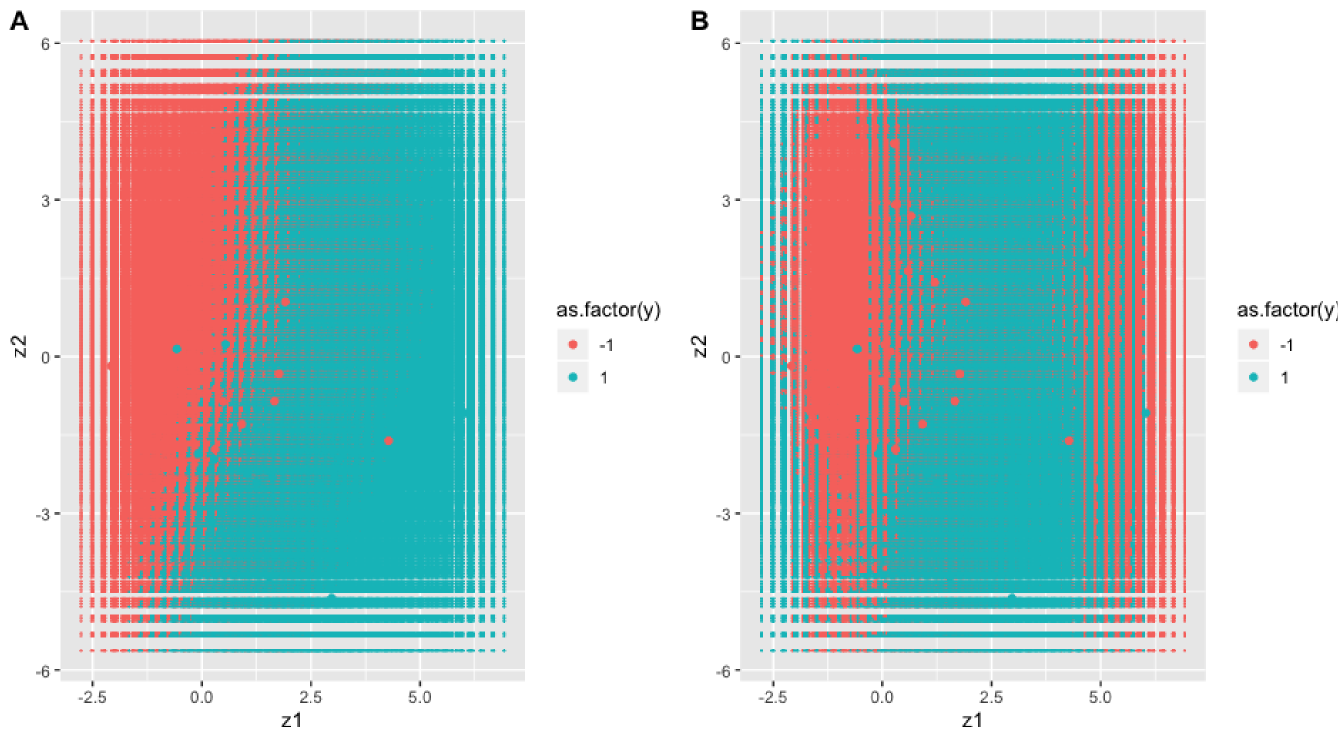

Figure 2: Figure A shows the classification rule of SMM with linear kernel and Figure B shows SMM with polynomial kernel.

4 New Algorithms

```
#SMMK

# Make sure Q matrix is positive definite
Makepositive = function(mat){
  h = eigen(mat,symmetric = T)
  nmat = (h$vectors)%*%diag(pmax(h$values,10^-4))%*%t(h$vectors)
  return(nmat)
}

# save kernel values
Karray = function(X,kernel = function(X1,X2) t(X1)%*%X2){
  m= nrow(X[[1]]); n = ncol(X[[1]]); N = length(X)
  K = array(dim = c(N,N,n,n))
  for(i in 1:N){
    for(j in 1:N){
      K[i,j, , ] = kernel(X[[i]],X[[j]])
    }
  }
  return(K)
}


# objective value function
objm = function(X,y,alpha,V,b,K,cost = 10){
  m= nrow(X[[1]]); n = ncol(X[[1]]); N = length(X)
  Hv = V%*%solve(t(V)%*%V)%*%t(V)
```

```
  Kv = matrix(nrow =N,ncol = N)
  for(i in 1:N){
    for(j in 1:N){
      Kv[i,j] = sum(Hv*K[i,j,,])
    }
  }
  coef = as.matrix(alpha*y)
  obj = t(coef)%*%Kv%*%coef/2 + cost*sum(pmax(1-y*(Kv%*%coef+b),0))
  return(obj)
}



# Main function (not available for weighed loss yet)
SMM = function(X,y,r,kernel = function(X1,X2) t(X1)%*%X2, cost = 10,rep = 1,p =
    .5){
  result = list()
  m= nrow(X[[1]]); n = ncol(X[[1]]); N = length(X)
  K = Karray(X,kernel)

  compareobj = 10^10
  for(i in 1:rep){
    error = 10; iter = 0; V = randortho(n)[,1:r,drop = F]
    obj = compareobj


    ### update U fixing V  ###
    vtvi = solve(t(V)%*%V)
    Hv = V%*%vtvi%*%t(V)
    Kv = matrix(nrow =N,ncol = N)
    for(i in 1:N){
      for(j in 1:N){
        Kv[i,j] = sum(Hv*K[i,j,,])
      }
    }
    dvec = rep(1,length(X))
    Dmat = Makepositive((y%*%t(y))*Kv)
    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)$solution


    while((iter < 20)&(error >10^-3)){


      ### update V fixing U ###

      # sum_{i,j} alpha_ialpha_jy_iy_jK(X_i,X_j)
      aayyK = 0
      for(i in 1:N){
        for(j in 1:N){
          aayyK = aayyK + alpha[i]*alpha[j]*y[i]*y[j]*K[i,j,,]
        }
      }

      # sum_j alpha_iy_iK(X_i,X_j)
      ayK = array(0,dim = c(N,n,n))
      for(i in 1:N){
        for(j in 1:N){
```

```
        ayK[i,,] = ayK[i,,] + alpha[j]*y[j]*K[j,i,,]
      }
    }

    # (U^TU)^{-1}
    utui = t(V)%*%V%*%solve(t(V)%*%aayyK%*%V)%*%t(V)%*%V

    # U^th(X_i)
    uth = array(dim = c(N,r,n))
    for(i in 1:N){
      uth[i,,] = vtvi%*%t(V)%*%ayK[i,,]
    }

    # Ku[i,j] = tr(H_uh(X_i),H_uh(X_j))
    Ku = matrix(nrow = N,ncol = N)
    for(i in 1:N){
      for(j in 1:N){
        Ku[i,j] = sum(uth[i,,]*(utui%*%uth[j,,]))
      }
    }
    dvec = rep(1,length(X))
    Dmat = Makepositive((y%*%t(y))*Ku)
    Amat = cbind(y,diag(1,N),-diag(1,N))
    bvec = c(rep(0,1+N),ifelse(y==1,-cost*(1-p),-cost*p))
    beta = solve.QP(Dmat,dvec,Amat,bvec,meq =1)$solution

    # update V
    V = 0
    for(i in 1:N){
      V = V+as.matrix(uth[i,,],nrow = r)*beta[i]*y[i]
    }
    V = V%*%utui


    ### update U fixing V  ###
    vtvi = solve(t(V)%*%V)
    Hv = V%*%vtvi%*%t(V)
    Kv = matrix(nrow =N,ncol = N)
    for(i in 1:N){
      for(j in 1:N){
        Kv[i,j] = sum(Hv*K[i,j,,])
      }
    }
    dvec = rep(1,length(X))
    Dmat = Makepositive((y%*%t(y))*Kv)
    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)$solution



    # slope part estimation

    slope = function(nX){
      coef <-  as.matrix(alpha*y)
      sp <-  t(coef)%*%unlist(lapply(X,function(x) sum(Hv*kernel(nX,x))))
      return(sp)
    }

```

```r
      # intercept part estimation (update b)
      positiv = min(unlist(lapply(X,slope))[which(y==1)])
      negativ = max(unlist(lapply(X,slope))[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) objm(X,y,alpha,V,b,K,
    cost)))]
      }
      obj = c(obj,objm(X,y,alpha,V,b0hat,K,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(nX) sign(slope(nX)+b0hat)
      result$slope = slope; result$b0 = b0hat; result$obj = obj[-1]; result$iter =
     iter
      result$error = error; result$predict = predictor; result$V = V
    }


  }
  return(result)
}


## Some kernels (Expkernel does not work)
Expkernel = function(Y,Z){
  return(exp(-t(Y-Z)%*%(Y-Z)))
}

polykernel = function(Y,Z,deg = 3){
  n = ncol(Y)
  return((t(Y)%*%Z+diag(1,n))^deg)
}
```