Final Project Report

Runan Yao

May 4, 2018

0. Prepare

Load data and required library

```
load("STAT715_ink_data_training.rData")
library(mvtnorm)
```

1. Support Function

All functions below are based on given paper.

generate matrix given dim n

```
get.P.Matrix <- function(n){
  result <- NULL
  for(i in 1:n){
    subDim = n-i
    subMatrix_0 = matrix(0,ncol = n-1-subDim, nrow = subDim)
    subMatrix_1 = matrix(1,ncol = 1, nrow=subDim)
    subMatrix_Diag = diag(subDim)
    subMatrix <- cbind(subMatrix_0,subMatrix_1,subMatrix_Diag)
    result <- rbind(result, subMatrix)
  }
  return(result)
}
```

get score Matrix based on a kernel function

obsevation index is first index

```
get.Score <- function(data, kernelFun){
  n = dim(data)[1]
  v <- sapply(1:n, function(x){
    sapply(1:n, function(y){
      kernelFun(data[x,], data[y,])
    })
  })
  return(v)
}
```

calculate SSa

```
get.SSa <- function(Vec_Sn, Matrix_Vk, n){
    return(
      Reduce('+',
           sapply(2:n, function(x){
             ( t(Matrix_Vk[,x]) %*% Vec_Sn )^2
           })
           )
  )
}
```

calculate SSe

```
get.SSe <- function(Vec_Sn, Matrix_Vk, n){
  return(
    Reduce('+',
           sapply((n+1):dim(Matrix_Vk)[1], function(x){
             ( t(Matrix_Vk[,x]) %*% Vec_Sn )^2
           })
    )
  )
}
```

MSa

```
get.MSa <- function(SSa, n){
  return(SSa/(n-1))
}
```

MSe

```
get.MSe <- function(SSe, n){
  N <- n*(n-1)/2
  return(SSe/(N-n))
}
```

MSt

```
get.MSt <- function(SSa, SSe, n){
  N <- n*(n-1)/2
  return((SSa+SSe)/(N-1))
}
```

Calculate Capital Sigma

```
get.Cov <- function(var.a, var.e, n){
  P <- get.P.Matrix(n)
  return(P%*%t(P)*var.a + var.e * diag(n*(n-1)/2))
}
```

Parm calculate function

Parm calculate function combines all support functions.

This function are used in training function

Input: data table, kernel function

Output: var.a, var.e, mu, sn(vector)

```
## Calc parms by given data and kernel function
get.Parms <- function(dat, kernel_fun){
  ## observation count
  n <- dim(dat)[1]

  ## score matrix
  score <- get.Score(dat, kernel_fun)

  ## Sn
  Sn <- NULL
  for(j in 1:(n-1)){
    Sn <- c(Sn, score[j,][c((j+1):n)])
  }

  ## P matrix
  P_matrix <- get.P.Matrix(n)
  tmp <- P_matrix %*% t(P_matrix)

  ## eigen of P*Pt
  eigen <- eigen(tmp)

  ## SSa
  SSa <- get.SSa(Sn, eigen$vectors, n)

  ## SSe
  SSe <- get.SSe(Sn, eigen$vectors, n)

  ## MSa
  MSa <- get.MSa(SSa, n)

  ## MSe
  Mse <- get.MSe(SSe, n)

  ## mean hat
  mean <- mean(Sn)

  ## variance
```

```
  var.a <- ( MSa - Mse ) / ( n - 2 )
  if(var.a < 0)
  {
    var.a = 0
    var.e = get.MSt(SSa, SSe, n)
  }
  else
  {
    var.e = Mse
  }

  result <- NULL
  result$a <- var.a
  result$e <- var.e
  result$mu <- mean
  result$sn <-Sn

  return(result)
}
```

2. Kernel function

During test, I tried several kernel functions:

1. Sum of 2 objects. The result has very small variance which lead f(Y) = 0. Fail
2. Euclidean (and it's log type). This function will give us a huge e variance. Var.e = 5 * Var.a or higher. Which cannot give me a good classification.
3. Cor(x,y) this is the final kernel I used in this project. It gives me a good result.

```
ker_1 <- function(x,y){
  return(cor(x,y))
}
```

3. Main Function

3.1 Training

training will give us a model with:

a. 13 means,

b. 13 a_var,

c. 13 e_var

```
trainModel <- function(training.data){
  ## clean data
  my.table <- NULL
  n.class <- dim(training.data)[4] # this is not using now.
  n.sample <- dim(training.data)[3]
  for(i in 1:13){
    for(j in 1:n.sample){
```

```
      rawMatrix <- training.data[,,j,i]
      rawVector <- as.vector(rawMatrix)
      vectorWithLabel <- c(i,rawVector)
      my.table <- cbind(my.table, vectorWithLabel)
    }
  }
  colnames(my.table) <- NULL
  #dim(my.table)
  my.table<-t(my.table)
  my.table[,1]

  mu.array <- rep(1, 13)
  var.a.array <- rep(1, 13)
  var.e.array <- rep(1, 13)
  P.array <- rep(1, 13)

  ## calc mean, var a, var e for each class
  ## use this calc P(Sn)
  for(i in 1:13){
    ## data in class i
    class.dat <- my.table[which(my.table[,1] == i),-1]

    class.result <- get.Parms(class.dat, ker_1)

    ## assign result to matrix
    mu.array[i] = class.result$mu
    var.a.array[i] = class.result$a
    var.e.array[i] = class.result$e

    Sigma_Sn <- get.Cov(class.result$a, class.result$e, dim(class.dat)[1])

    P.array[i] <- dmvnorm(class.result$sn, mean=rep(class.result$mu, length(class.result$sn)), sigma=Si

  }

  #cat(var.e.array)

  result <- NULL
  result$mu <- mu.array
  result$a <- var.a.array
  result$e <- var.e.array
  result$p <- P.array
  result$table <- my.table

  return(result)
}
```

3.2 Classification

classification function will take 200*31*p*13 matrix

return a vector of 13 length. Shows which class each 200*31*p data block should belong to

```
classificationResult <- function(testData, model){
  ## check testData
  if(
    dim(testData)[1] != 200 ||
    dim(testData)[2] != 31 ||
    dim(testData)[4] != 13
  ){
    cat('Test Data does not fit model. We need 200*31*p*13 sized test data.')
    cat('\r\n')
    return(NULL)
  }

  ## clean test data, change into 6300 col ,  (13*p) row matrix
  testingSampleinEachGroup <- dim(testData)[3]
  testingTable <- NULL
  for(i in 1:13){
    for(j in 1:testingSampleinEachGroup){
      rawMatrix <- testData[,,j,i]
      rawVector <- as.vector(rawMatrix)
      vectorWithLabel <- c(i,rawVector)
      ## Here we add 'label' to indicate these sample are belong to same group. But, we don't know whic
      ## Be careful when using this label
      testingTable <- cbind(testingTable, vectorWithLabel)
    }
  }
  colnames(testingTable) <- NULL
  testingTable <- t(testingTable)

  ## read model data
  mu.array <- model$mu
  var.a.array <- model$a
  var.e.array <- model$e
  p.array <- model$p
  trainingTable <- model$table


  ## for each testing group i,
  ##   1.  Combine its data with training data with label j
  ##   2.  Calculate Cap_Sigma_s using var_a, var_e from class j
  ##   3.  Calculate the joint prob Pij( c( testingGroup_i, trainingGroup_j ) )
  ##       And pull the prob of Pj( trainingGroup_j ) from model
  ##   4.  Then conditional prob Pi|j = Pij / Pj
  ##   5.  Find the j, such that Pi|j has the biggest value
  ##   6.  j is the pen group i come from
  resultLabel <- rep(0, 13)
  highestProb <- rep(0, 13)
  for(i in 1:13){
    ## testing data on group i, without group label
    groupData <- testingTable[which(testingTable[,1] == i), -1]
    for( j in 1:13){

      var.a <- var.a.array[j]
      var.e <- var.e.array[j]
```

```
      mu <- mu.array[j]
      Pj <- p.array[j]

      ## training data for pen j, without pen label
      trainingData <- trainingTable[which(trainingTable[,1] == j), -1]

      ## combine data
      dat <- rbind(groupData, trainingData)
      dat.n <- dim(dat)[1]

      ## Calculate Cap_Sigma_s
      Sigma_s <- get.Cov(var.a, var.e, dim(dat)[1])

      ## Calculate s
      ## score matrix
      score <- get.Score(dat, ker_1)

      ## S
      S <- NULL
      for(k in 1:(dat.n-1)){
        S <- c(S, score[k,][c((k+1):dat.n)])
      }

      ## joint prob:
      Pij <- dmvnorm(S, mean=rep(mu, length(S)), sigma=Sigma_s, log = TRUE)

      ## Pi|j
      P_cond <- Pij - Pj

      if( ( P_cond > highestProb[i]) ){
        #we find a higher Pi|j, replace existing
        resultLabel[i] = j
        highestProb[i] = P_cond
      }
    }
  }
  return(resultLabel)
}
```

3.3 Deliver function

The deliverable needs to be a function that accepts TWO INPUTS: 1) 200 x 31 x 22 x 13 array of training data from which the model will learn its parameters 2) 200 x 31 x p x 13 array of testing data from which the model ONE OUTPUT: And then return the best class for each of the 13 sets of p matrices. Hence a 13 long vector is needed.

```
deliverFunction <- function(train.dat, test.dat){
  return( classificationResult(test.dat,
                               trainModel(train.dat)))
}
```

4. Example

4.1 All data training, all data testing

```
result1 <- deliverFunction(ink.training.dat, ink.training.dat)
print(result1)
```

```
##  [1]  1  3  2  4  5  6  7  8  9 10 11 12 13
```

Here, we use training data to test the model. We can see pen 2 and pen 3 are miss classified.

4.2 First 20 training, rest 2 testing

```
trainingData <- ink.training.dat[,,1:20,]
testingData <- ink.training.dat[,,c(21,22),]

result2 <- deliverFunction(trainingData, testingData)

print(result2)
```

```
##  [1]  1  2  3  4  1  6  7  4  9 10 11 12 13
```

Here, we use 20 samples in each class to train the model. And the rest 2 to do classification. Pen 5 and Pen 8 got miss classified.

5. Conclusion

Before implementing the method in given paper, I tried to apply mcLDA in given data. I got error when I try calculate eigen value and eigen vector: "Error in solve.default(mcLDA.within.cov): system is computationally singular: reciprocal condition number = 1.59585e-24". This happens because our training data is too small compare with the number of variable.

Then I turned to implementing the paper proposed method. In this final project, we use kernel function to convert sample data into a single distance value. By applying cross check, we can see the classification works well.

However, I still have a question about the this method.

1. For my understanding, when we training a model, we should abstract the most useful data from training data, turn it into a small data block. But, the model we got here, actually includes the original data.

Is it possible, to calculate the distance between sample from training data and sample from testing data without refering original training data?

2. The training function runs fast, but classification function runs slow . Is is possible to turn all computational load into training function, so we can make classification function runs faster?

6. Q2:

No for current method/kernel function.

The main requirement of using 5.5.1 is the data should have same covariance. But, if we check var.a value, it is different between classes. This leads the cov-matrix to be different.

But, a good news is the cov-matrix are following the same style: PPta + e*I. I believe there are some way (kernel function?) to transfer the difference in var.a into the difference in mu. However, whether we need to check our data is sufficent for modeling or not.

7. Reference/Link

All code and data are avaliable in Github: https://github.com/HalforcNull/MultiVarFinalProject

Source code can be found in final.R file.