PrecisionFDA Phase I

Segbehoe, Lawrence Sethor

October 17, 2018

Template for STAT 736, South Dakota State University, for phase I of the precisionFDA challenge.

Part A: Exploratory Data Analysis

- Import all the data set

The testing data set involve `test_cli.tsv` and `test_pro.tsv`

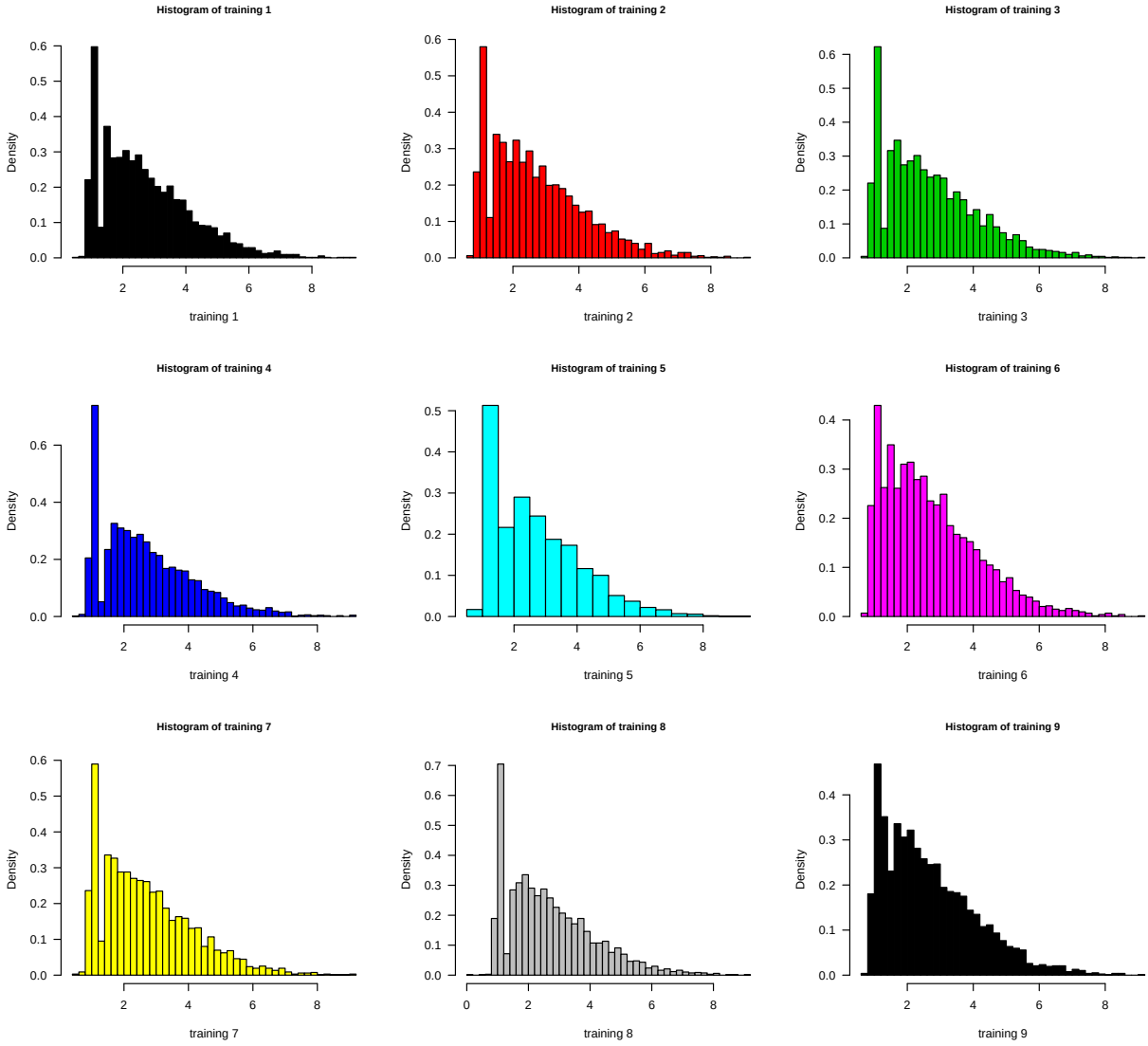
```
# test_pro data set
test_pro <- read.table("test_pro.tsv", sep = "")
# dim(test_pro) # 4118 * 80

# test_cli data set
test_cli <- read.table("test_cli.tsv",header = T,  sep = "")
# dim(test_cli) #  80 * 3
```

The training data sets are `train_cli.tsv` and `train_pro.tsv`

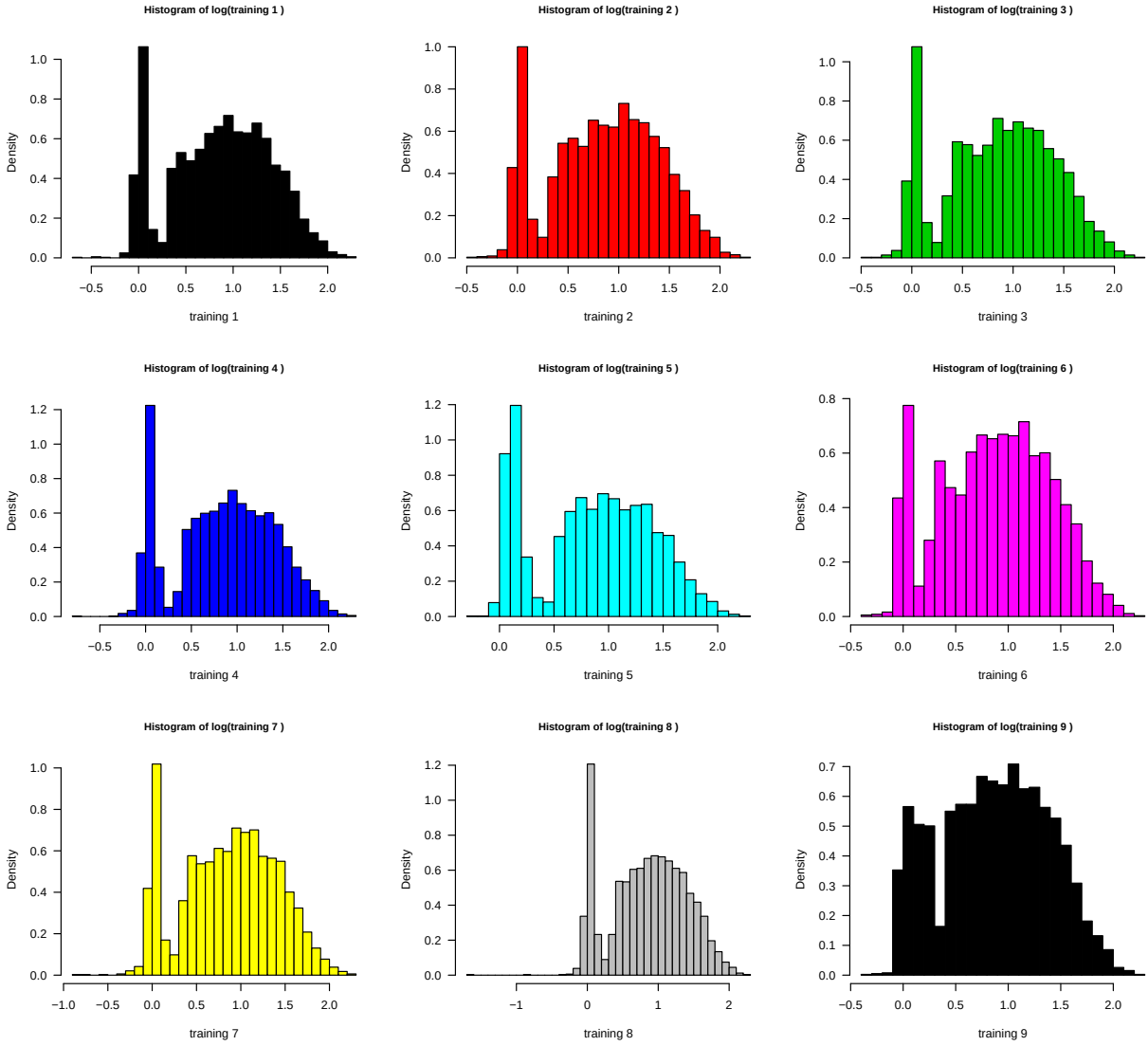
```
# training_pro data set
train_pro <- read.table(  "train_pro.tsv", sep = "")

# dim(t(train_pro)) #  80 * 4118

# train_cli data set
train_cli <- read.table(  "train_cli.tsv",header = T, sep = "")
```

1. Distribution by sample

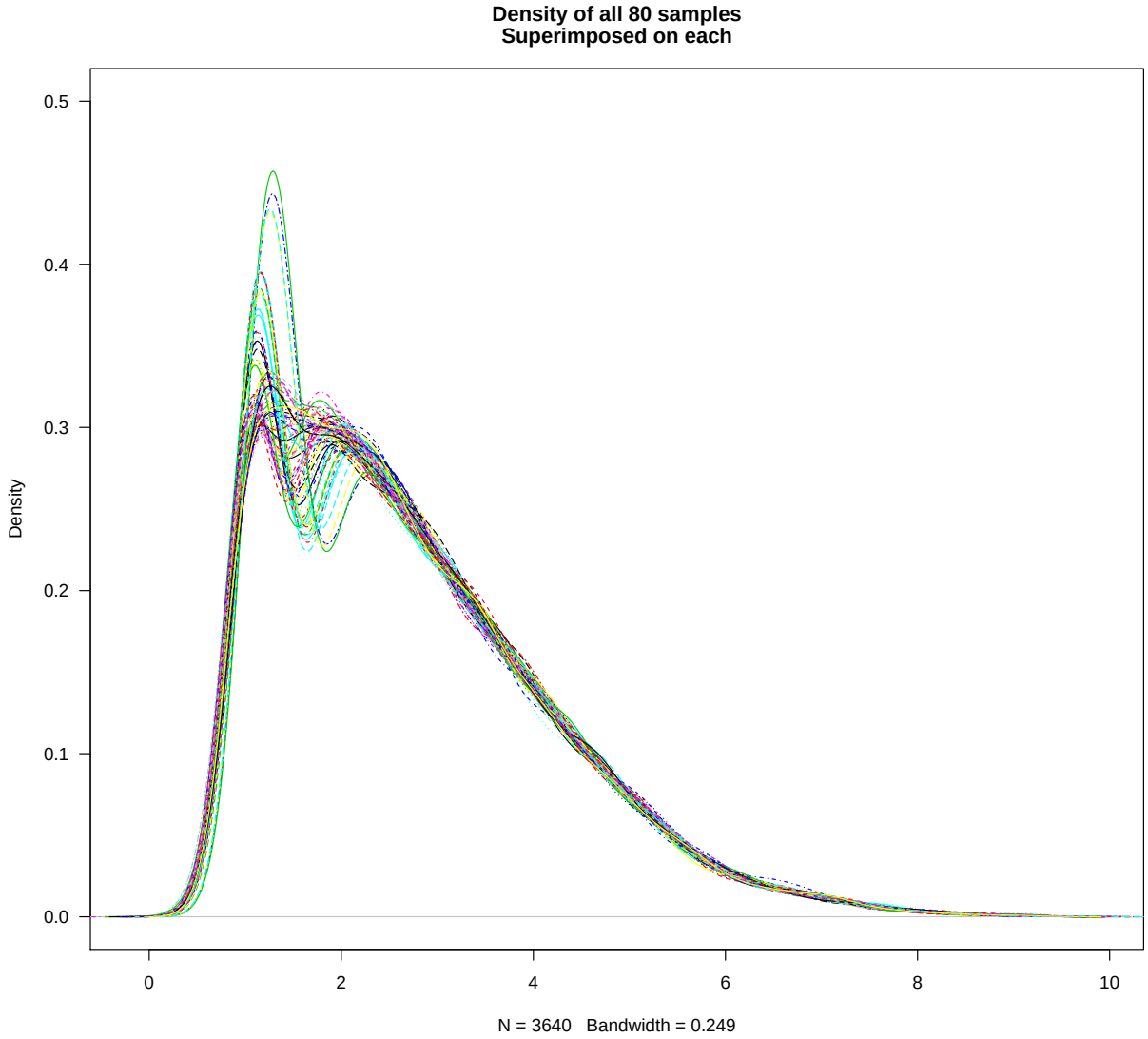
```
# Histogram for the train_pro
par(mfrow  = c(3,3))

for(i in 1:9){
hist(train_pro[, i], col = i, xlab = paste("training", i), las = 1, freq = F,
     breaks = "fd", main = paste("Histogram of training", i),  cex.main = 0.9)
}
```



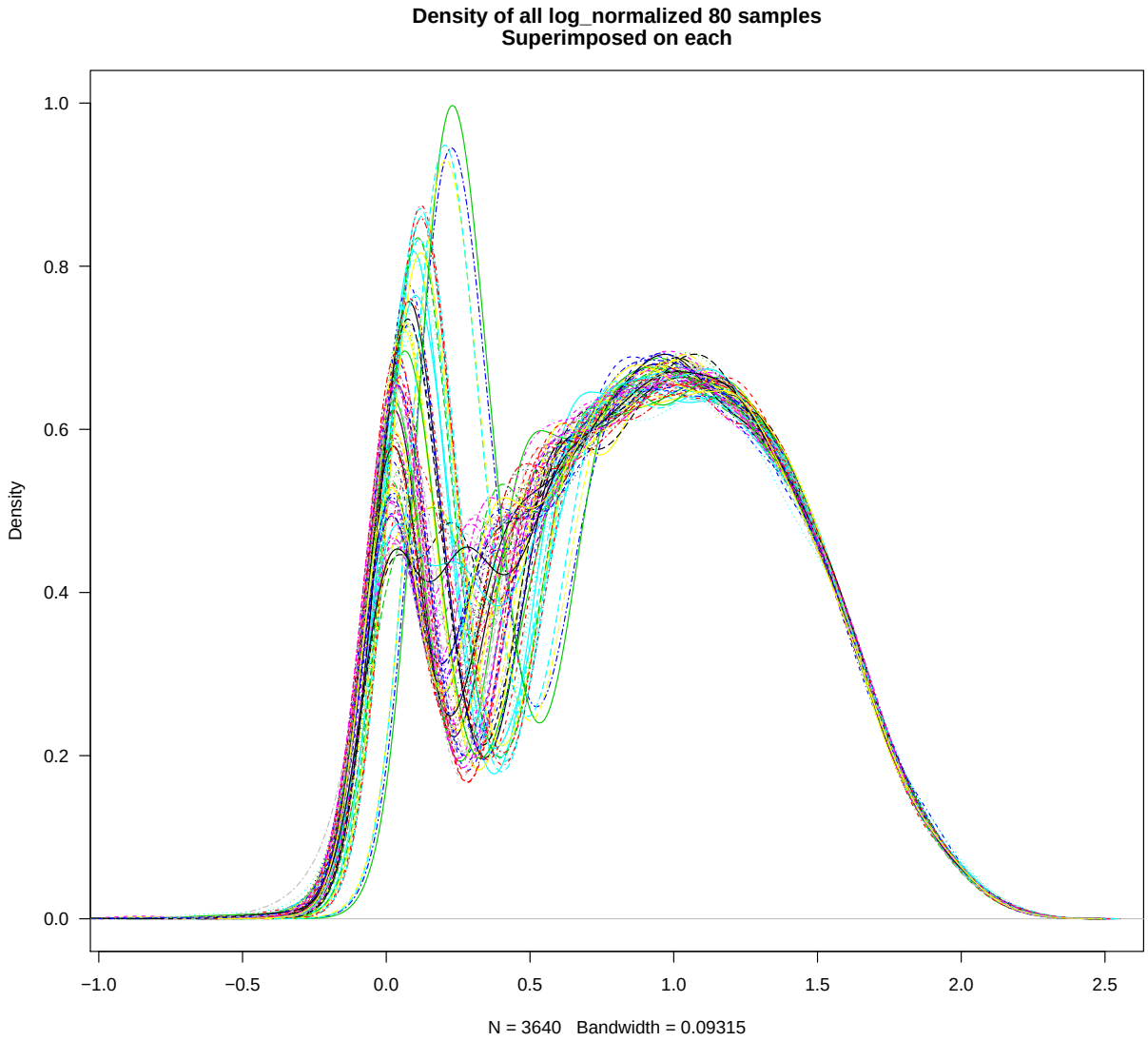

```
# Histogram after taking the natural log of the train_pro
par(mfrow  = c(3,3))

for(i in 1:9){
  hist(log(train_pro[, i]), col = i, xlab = paste("training", i), freq = F, las = 1,
       breaks = "fd", main = paste("Histogram of log(training", i,")"), cex.main = 0.9)
}
```




```
par(mfrow  = c(1,1))
## Density plot for all samples
plot(stats::density((train_pro[,1]),  na.rm = TRUE), main = c(paste( "Density of all 80 samples"),
                                                    paste("Superimposed on each")), type = "n",ylim = c(0,0.5), l
for (i in 1:80) { lines(density((train_pro[,i]),  na.rm = TRUE), col = i, lty = i)}
```



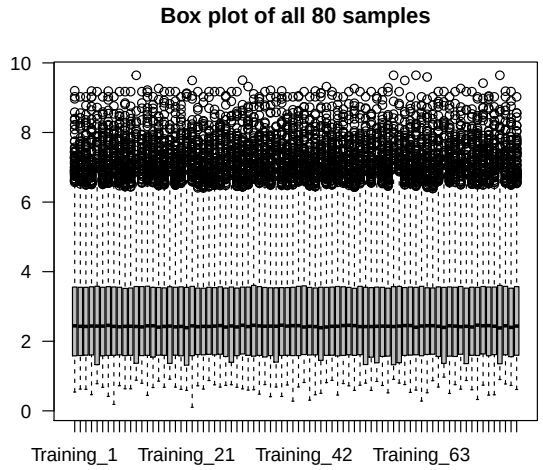

```
par(mfrow  = c(1,1))
## Density plot for all samples
plot(stats::density(log(train_pro[,1]),  na.rm = TRUE), main = c(paste( "Density of all log_normalized 8
                                                  paste("Superimposed on each")), type = "n",ylim = c(0,1),  la
for (i in 1:80) { lines(density(log(train_pro[,i]),  na.rm = TRUE), col = i, lty = i)}
```

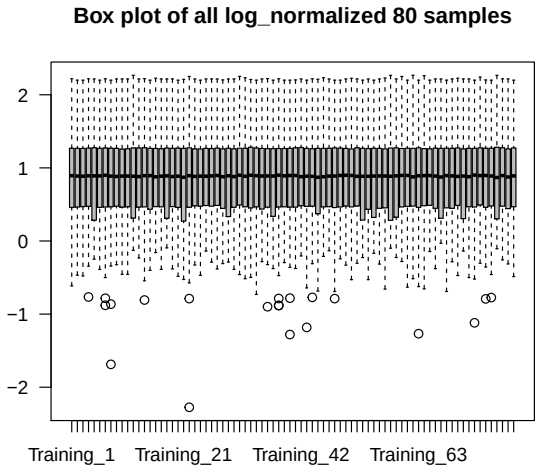


There are alot of zeros in the data set. Taking the natural log fairly normalizes the data set. This is almost the same for all the 80 samples in the training set.

Boxplot of the training Proteomic

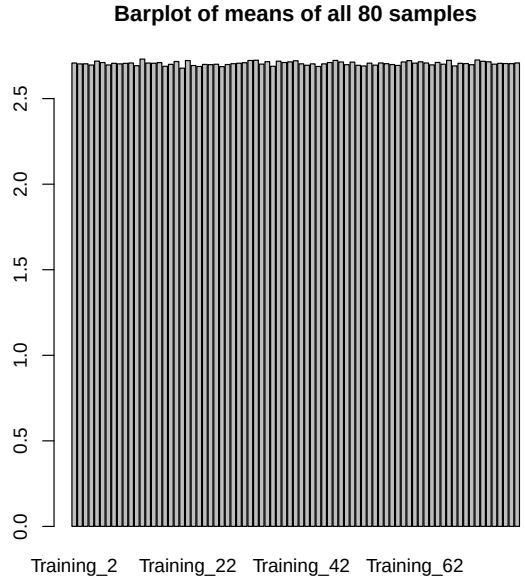
```
## boxplot
par(mfrow = c(1,2))
boxplot(train_pro, las = 1, col = "grey", main = "Box plot of all 80 samples")

## boxplot of the logtransformation

boxplot(log(train_pro), las = 1 , col = "grey", main = "Box plot of all log_normalized 80 samples")
```

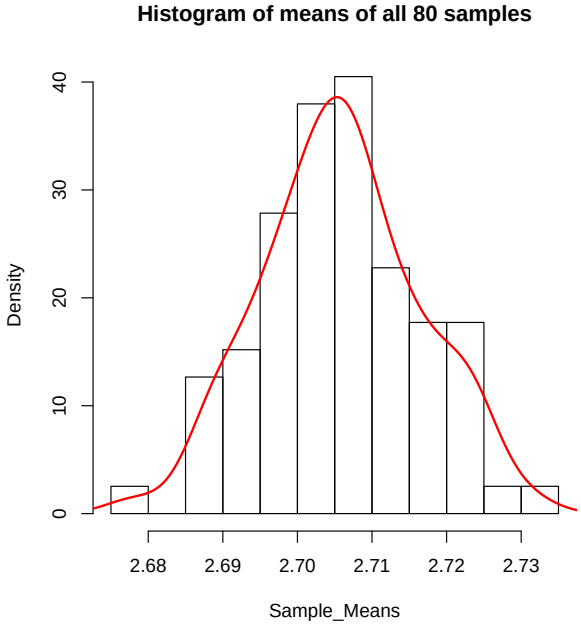





```
par(mfrow = c(1,1))
```

```
# colmeans
par(mfrow = c(1,2))
Sample_Means <- with(train_pro, colMeans(train_pro[,-1], na.rm = T))
barplot(Sample_Means, main = "Barplot of means of all 80 samples")
hist(Sample_Means, main = "Histogram of means of all 80 samples", freq = F)
lines(density(Sample_Means), lwd = 2, col = 2)
```

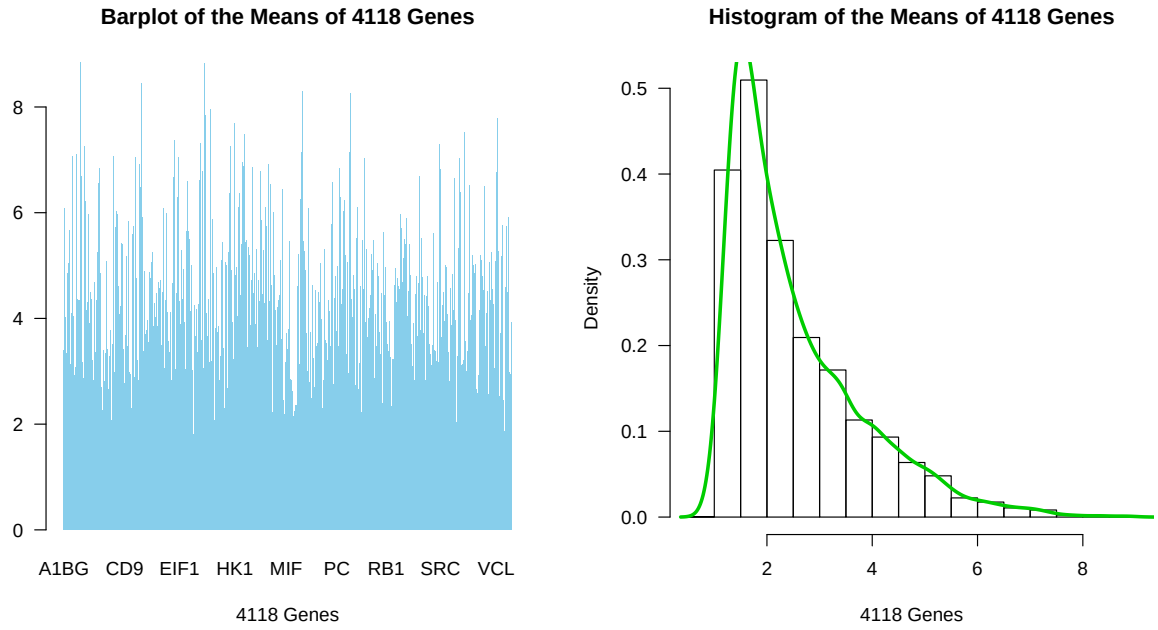




- In all, they have almost the **same average** (median).
- Taking log of the data fairly normalized the data sets with **fewer outliers** than before.
- Means of samples are fairly normal

Distribution based on the 4118 genes

```
Means.Genes = with(train_pro, rowMeans(train_pro[,-1], na.rm = T))
par(mfrow = c(1,2))
barplot(Means.Genes, border = NA, col = "skyblue", las = 1,
        xlab = "4118 Genes", main = "Barplot of the Means of 4118 Genes")
hist(Means.Genes,las = 1, xlab = "4118 Genes", freq = F,
  main = "Histogram of the Means of 4118 Genes")
lines( density(Means.Genes, na.rm = T), lwd = 3, col = 3)
```




```
par(mfrow = c(1,1))
```

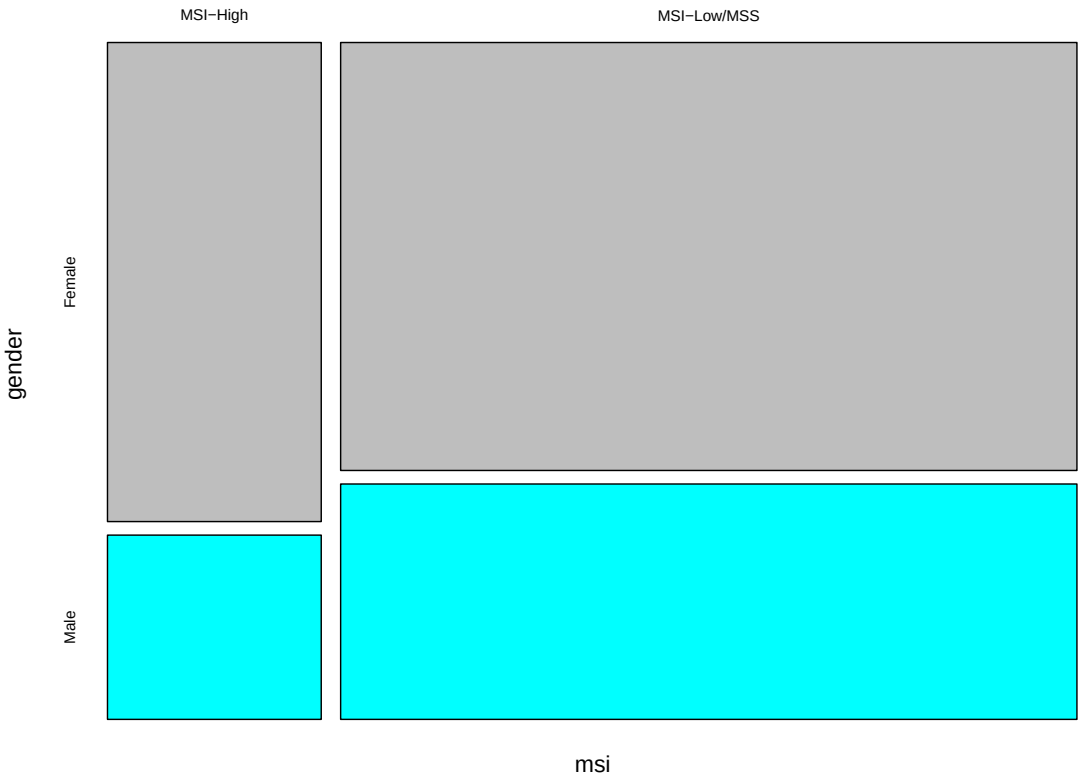
##2. Correlation between gender and MSI

```
library(knitr)

# table of training clinical data
kable(table(train_cli[,2:3]), caption = "Counts of Gender versus MSI")
```

Table 1: Counts of Gender versus MSI

	MSI-High	MSI-Low/MSS
Female	13	40
Male	5	22

```
## Mosaic Plot
par(mfrow = c(1,1))
mosaicplot(t(table(train_cli[,2:3])), type = "Pearson",
           main = "Mosaic Plot of Gender vs MSI", color = c(8,13))
```




```
### Contigency table with proportions of each variable and
###   A contigency table with chisquare test
library(gmodels)
with(train_cli, CrossTable(gender, msi, prop.r = T,chisq = T,
                           prop.c = T, prop.t = F, digits = 4,
                          resid = F ))
```

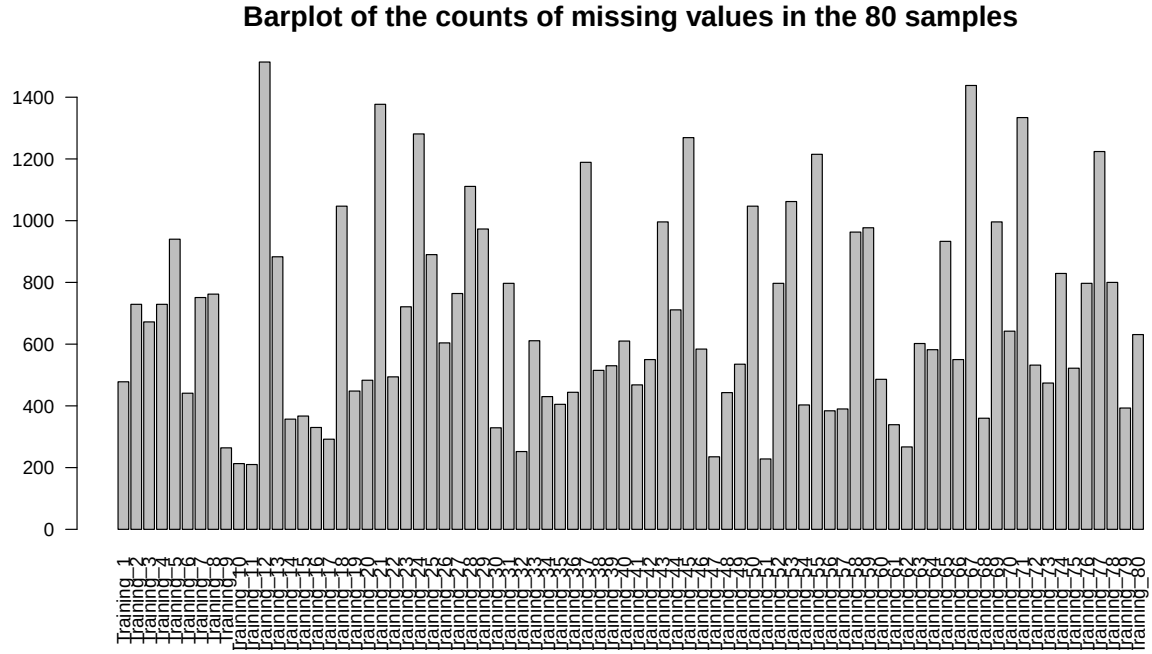
```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## | Chi-square contribution |
## |           N / Row Total |
## |           N / Col Total |
## |-------------------------|
##
##
## Total Observations in Table:  80
##
##
##              | msi
##       gender |    MSI-High | MSI-Low/MSS |   Row Total |
```

```
## -------------|-------------|-------------|-------------|
##       Female |          13 |          40 |          53 |
##              |      0.0969 |      0.0281 |             |
##              |      0.2453 |      0.7547 |      0.6625 |
##              |      0.7222 |      0.6452 |             |
## -------------|-------------|-------------|-------------|
##         Male |           5 |          22 |          27 |
##              |      0.1902 |      0.0552 |             |
##              |      0.1852 |      0.8148 |      0.3375 |
##              |      0.2778 |      0.3548 |             |
## -------------|-------------|-------------|-------------|
## Column Total |          18 |          62 |          80 |
##              |      0.2250 |      0.7750 |             |
## -------------|-------------|-------------|-------------|
##
##
## Statistics for All Table Factors
##
##
## Pearson's Chi-squared test
## ------------------------------------------------------------
## Chi^2 =  0.3704956     d.f. =  1     p =  0.5427342
##
## Pearson's Chi-squared test with Yates' continuity correction
## ------------------------------------------------------------
## Chi^2 =  0.105999     d.f. =  1     p =  0.744746
##
##
```

- 75% of Females have MSI-Low/MSS compared to only 25% with MSI-High
- 81% of Males have MSI-Low/MSS compared to only 19% with MSI-High
- 72% of MSI-High are Females compared to only 28% for Males
- With 62 patients in the MSI-Low/MSS category, 65% are Females
- There are more females that males for both MSI low and high based on the mosaic plot.
- The p-value of 0.745 shows from the chisquare test shows that there is **no association**. between gender and Microsatellite instability.

Checking Missing Values

```r
## check for protein gene that has mission values for the training sample


# based on the samples
barplot(apply(train_pro, 2, function(x){
  # sum of missing values
  SUMM = sum(is.na(x))
  return(SUMM)}), main = "Barplot of the counts of missing values in the 80 samples",
  las = 2, cex.main = 1.5)
```



##3. ...

##4. Observations

- Data follows normal distribution taken the natural log of the samples
- A lot of Missing values
- There is zero inflation in the data set (a lot of zeros in the data)

#Part B: Predict Gender

##1. Build first model

a). Read in data and preprocess (normalization, imputation, or digitalization)

Imputation with zero

```
# Write a CSV file in order to impute with zeros
write.csv(train_pro, "imputed_train_pro.csv", na = "0", row.names = T)
new_train_pro = read.table("imputed_train_pro.csv", header = T, sep = ",")
dim(new_train_pro) # 4118  * 81
```

```
## [1] 4118   81
```

```
kable(head(new_train_pro)[, 1:7], caption = "Imputed Train_pro with row names appended as a new column"
```

Table 2: Imputed Train_pro with row names appended as a new column

X	Training_1	Training_2	Training_3	Training_4	Training_5	Training_6
A1BG	2.919688	3.753851	3.513302	3.588087	3.405176	3.575759
A2M	5.737663	5.752416	5.601927	6.092091	5.664237	6.481125
AAAS	2.002401	1.090277	2.789359	2.085648	1.825888	1.073909
AACS	1.461113	0.000000	1.504314	1.007233	0.000000	1.987334
AAGAB	0.000000	0.000000	0.000000	0.000000	0.000000	1.895773

X	Training_1	Training_2	Training_3	Training_4	Training_5	Training_6
AAK1	1.731009	0.000000	0.000000	0.000000	1.066124	1.578639

```
# make the 1st column rownames again
Imputed_train_pro <- new_train_pro[, -1]
rownames(Imputed_train_pro) <- new_train_pro[, 1]
kable(head(Imputed_train_pro)[,1:6], caption = "Imputed Train_pro with row names corrected")
```

Table 3: Imputed Train_pro with row names corrected

	Training_1	Training_2	Training_3	Training_4	Training_5	Training_6
A1BG	2.919688	3.753851	3.513302	3.588087	3.405176	3.575759
A2M	5.737663	5.752416	5.601927	6.092091	5.664237	6.481125
AAAS	2.002401	1.090277	2.789359	2.085648	1.825888	1.073909
AACS	1.461113	0.000000	1.504314	1.007233	0.000000	1.987334
AAGAB	0.000000	0.000000	0.000000	0.000000	0.000000	1.895773
AAK1	1.731009	0.000000	0.000000	0.000000	1.066124	1.578639

```
dim(Imputed_train_pro) # 4118  * 80
```

```
## [1] 4118   80
```

```
class(Imputed_train_pro)
```

```
## [1] "data.frame"
```

```
# Write a CSV file in order to impute with zeros
write.csv(test_pro, "imputed_test_pro.csv", na = "0", row.names = T)
new_test_pro = read.table("imputed_test_pro.csv", header = T, sep = ",")
dim(new_test_pro) # 4118  * 81
```

```
## [1] 4118   81
```

```
kable(head(new_test_pro)[, 1:7], caption = "Imputed Test_pro with row names appended as a new column")
```

Table 4: Imputed Test_pro with row names appended as a new column

X	Testing_1	Testing_2	Testing_3	Testing_4	Testing_5	Testing_6
A1BG	3.446723	3.6695804	3.398472	3.112875	3.5359559	3.222556
A2M	5.994520	6.3710379	6.132440	5.645341	5.5328111	5.754886
AAAS	2.168001	2.4105428	0.000000	2.211822	0.9875613	1.737837
AACS	0.000000	1.0248385	0.000000	1.099892	1.0766151	1.001861
AAGAB	0.000000	0.9911145	1.096724	1.056902	0.0000000	0.000000
AAK1	1.613695	1.1920088	2.132646	2.060400	0.0000000	1.075423

```
# make the 1st column rownames again
Imputed_test_pro <- new_test_pro[, -1]
rownames(Imputed_test_pro) <- new_test_pro[, 1]
kable(head(Imputed_test_pro)[,1:6], caption = "Imputed Test_pro with row names corrected")
```

Table 5: Imputed Test_pro with row names corrected

	Testing_1	Testing_2	Testing_3	Testing_4	Testing_5	Testing_6
A1BG	3.446723	3.6695804	3.398472	3.112875	3.5359559	3.222556
A2M	5.994520	6.3710379	6.132440	5.645341	5.5328111	5.754886
AAAS	2.168001	2.4105428	0.000000	2.211822	0.9875613	1.737837
AACS	0.000000	1.0248385	0.000000	1.099892	1.0766151	1.001861
AAGAB	0.000000	0.9911145	1.096724	1.056902	0.0000000	0.000000
AAK1	1.613695	1.1920088	2.132646	2.060400	0.0000000	1.075423

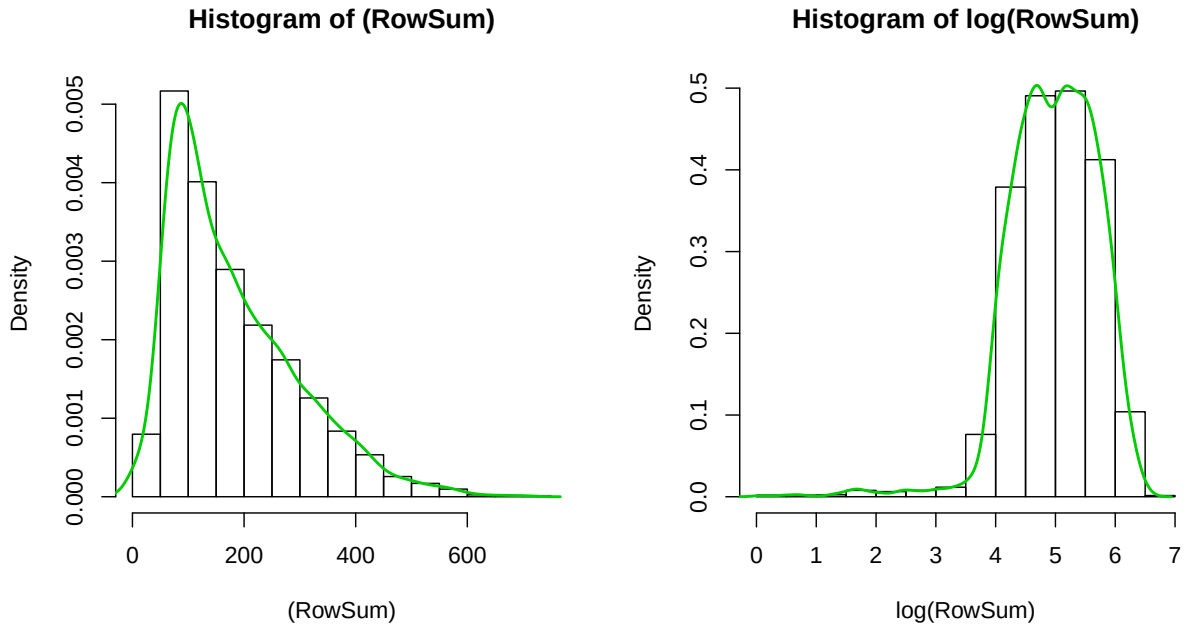
```
dim(Imputed_test_pro) # 4118  * 80
```

```
## [1] 4118   80
```

```
class(Imputed_test_pro)
```

```
## [1] "data.frame"
```

```
RowSum = apply(Imputed_train_pro, 1, sum)
par(mfrow = c(1,2))
hist((RowSum), freq = F)
lines(density((RowSum)), lwd = 2, col = 3)

hist(log(RowSum), freq = F)
lines(density(log(RowSum)), lwd = 2, col = 3)
```




```
par(mfrow = c(1,2))
```

b). Feature selection, if needed. (highly recommended as data is noisy)

- Row sum or genes with total proteomic reads of less than 75, is removed from the data set. This gives 3442 genes to work with.

```
new75 = Imputed_train_pro[RowSum >= 75,]
dim(new75)
```

```
## [1] 3442   80
```

c). Build one model as proof of concept (just make it work)

Gaussian Process Classification - Model 1

- The first model I am using is based on the chapter six(6) of the book; **Pattern Recognition and Machine Learning (PRML)** written by Christopher M. Bishop, 2006.
- Since this is not a usual R function, it suffices to check how the function accept the given data set. I have done a lot of examples which have proved to give 100% accuracy and many different choices of parameters, all give a very good result based on your choices of parameters. Further explanations follow.
- ** Caution**: I built the Gaussian Process classification `function` to take the accept the data just like what we have been given.
  - Do not transpose the `train_pro.tsv` data, input the function as it is.
  - Do not combine the classes or `train_cli.tsv` with the proteomics data. The function has a space in it to accept the

```
## Functions needed to be called in the Gaussian Process Classifier (GPC)


# Gaussian Kernel (GK)
GK <- function(Xn, Xm, theta = 0.5){
  kern = exp(-(theta)*(t(as.matrix(Xn-Xm))%*%(as.matrix(Xn-Xm))))
  return(kern)
}


## Correlation Kernel
CK <- function(Xn, Xm){
  cor(Xn,Xm) # the standard correlation function
}


# Equation 4.59: Sigmoid function
sig.fun <- function(x){
  1/(1 + exp(-x))
}


# Equation 6.80: Laplace Approximation
Laplace.aN <- function(a.N, t.N, C.N){

  a.N # vector of mode
  t.N # vector of target values
```

```
  C.N # Gram Matrix
  N = dim(C.N)[1]

  Laplace =  (
                -0.5*t(a.N)%*%solve(C.N)%*%a.N
                - (N/2)*log(2*pi)
                - 0.5*log(det(C.N))
                + t(t.N)%*%a.N
                - sum(log(1 + exp(a.N)))
              )

  return(Laplace)
}


# Equation 6.81: Gradient of the Laplace Approximaton
Gradient.a.N <- function(a.N, t.N, C.N){

  a.N # vector of mode
  t.N # vector of target values
  C.N # Gram Matrix

  Gradient =  (
                  t.N
                - sig.fun(a.N)
                - solve(C.N)%*%a.N
               )

  return(Gradient)
}
```

Things to note

- The function below takes five arguments
  - training data
  - test data
  - vector of Classes
  - K (an indicator of whick kernel function to be used). it has 6 options

```
##############################################################################
###############    Gaussian Process Classification (GPC)      #################
##############################################################################


## Function for the Gram Matrices


GPC.N_N2 <- function(train, test, K =1, theta = 15){


  # K # an indicator of which kernel to use
  # K = 1 for Gaussian Kernel
  # K = 2 for Correlation Kernel
  # K = 3 for exponential Kernel
```

```r
# K = 4 for the sum of both Gaussian and Correlation kernel-- this gives another valid kernel
# K = 5 for the sum of both exponential and Correlation kernel -- this gives another valid kernel
# K = 6 for the sum of both Gaussian and exponential kernel-- this gives another valid kernel


# extracting feactures from the training data
X.train <- data.frame(train) # training data
N <- dim(X.train)[2] # number of observations

# classes  <- c(classes) # converting to vectors
# target <- ifelse(classes == "Female", 1,0) # vector of classes


# Now extract the feature of the test data
X.test <- data.frame(test) # test data
N2 <- dim(X.test)[2]# number of observation of test data


# put togetther -- train and test data
X <- cbind(X.train,X.test) # both training and test data
N_N2 <- dim(X)[2] # total sample from training and test data

# get the Gram matrix  container for both training and test data
Gram.matrix.all <- matrix(NA,nrow = (N_N2), ncol= (N_N2))

## Create Gram matrix using Gaussian Kernel
for (i in 1:(N_N2)) {
  for (j in 1:(N_N2)) {
    Gram.matrix.all[i,j] <- GK(X[,i], X[,j])
  }
}


 # put togetther -- train and test data
X <- cbind(X.train,X.test) # both training and test data
N_N2 <- dim(X)[2]

# get the Gram matrix  container for both training and test data
Gram.matrix.all2 <- matrix(NA,nrow = (N_N2), ncol= (N_N2))

## Create Gram matrix using Correlation Kernel
for (i in 1:(N_N2)) {
  for (j in 1:(N_N2)) {
    Gram.matrix.all2[i,j] <- theta*cor(c(X[,i]), c(X[,j]))
  }
}



 # put togetther -- train and test data
X <- cbind(X.train,X.test) # both training and test data
N_N2 <- dim(X)[2]
```

```r
# get the Gram matrix  container for both training and test data
Gram.matrix.all3 <- matrix(NA,nrow = (N_N2), ncol= (N_N2))

## Create Gram matrix using Exponential Kernel
for (i in 1:(N_N2)) {
  for (j in 1:(N_N2)) {
    Gram.matrix.all3[i,j] <- exp(-theta*(norm(as.matrix(c(X[,i]) - c(X[,j])), type = "F")))
  }
}


# Gram matrix created by the Gaussian kernel
Gram.matrix.all <- Gram.matrix.all + 0.002*diag(N_N2)


# Gram matrix created by the correlation kernel
Gram.matrix.all2 <- Gram.matrix.all2 + 0.002*diag(N_N2)

# Gram matrix created by the exponential kernel
Gram.matrix.all3 <- Gram.matrix.all3 + 0.002*diag(N_N2)



GG = Gram.matrix.all + Gram.matrix.all2
GG2 = Gram.matrix.all2 + Gram.matrix.all3
GG3 = Gram.matrix.all3 + Gram.matrix.all



# choices for which Kernel or Kernel combination to use

if(K == 1){ # Gaussian Kernel

  # get the Gram matrix for training
  Gram.matrix <- as.matrix(Gram.matrix.all[1:N,1:N])
  # get k matrix associated with the test data
  matrix.k <- as.matrix(Gram.matrix.all[1:N, (N+1):(N_N2)])

} else  if(K == 2){ # Correlation Kernel

  # get the Gram matrix for training
  Gram.matrix <- as.matrix(Gram.matrix.all2[1:N,1:N])
  # get k matrix associated with the test data
  matrix.k <- as.matrix(Gram.matrix.all2[1:N, (N+1):(N_N2)])

} else  if(K == 3){ # Exponential Kernel

  # get the Gram matrix for training
  Gram.matrix <- as.matrix(Gram.matrix.all3[1:N,1:N])
  # get k matrix associated with the test data
  matrix.k <- as.matrix(Gram.matrix.all3[1:N, (N+1):(N_N2)])

} else  if(K == 4){ #  Sum of Gaussian and Correlation Kernel
```

```
    # get the Gram matrix for training
    Gram.matrix <- as.matrix(GG[1:N,1:N])
    # get k matrix associated with the test data
    matrix.k <- as.matrix(GG[1:N, (N+1):(N_N2)])

  } else  if(K == 5){ #  Sum of Correlation and  Exponential Kernel

    # get the Gram matrix for training
    Gram.matrix <- as.matrix(GG2[1:N,1:N])
    # get k matrix associated with the test data
    matrix.k <- as.matrix(GG2[1:N, (N+1):(N_N2)])

  } else if(K == 6){ #  Sum of Exponential and Gaussian Kernel

    # get the Gram matrix for training
    Gram.matrix <- as.matrix(GG3[1:N,1:N])
    # get k matrix associated with the test data
    matrix.k <- as.matrix(GG3[1:N, (N+1):(N_N2)])

  } else {
    # if(K !=1 & K !=2 & K !=3 & K != 4 & K != 5 & K != 6)
    STATEMENT = cat("Kernel function indicator is not specified", "\n",
        "You may use K = 1 for Gaussian Kernel", "\n",
        "K = 2 for Correlation Kernel", "\n",
        "k = 3 for the Exponential Kernel", "\n",
        "K = 4 for the sum of Gaussian and Correlation Kernel", "\n",
        "K = 5 for the sum of the Correlation and Exponential Kernel ", "\n" ,
        "K = 6 for the sum of the Exponential Kernel and Gaussian", "\n",
        "The default is Gaussian Kernel")

    # get the Gram matrix for training
    Gram.matrix <- as.matrix(Gram.matrix.all[1:N,1:N])
    # get k matrix associated with the test data
    matrix.k <- as.matrix(Gram.matrix.all[1:N, (N+1):(N_N2)])

    STATEMENT
  }
  return(list(CN = Gram.matrix, k.N2 = matrix.k))
}
```

Function for optimization

```
optim.GPC <- function(train, test, K, theta,  aN, tN){

  CN <- GPC.N_N2(train, test, K, theta)$CN

  # k is defined just like before
  # theta is also defined just like before

  # tN # vector of class labels
  # aN # set of initial value for the optim function
  # Note that length(aN) == dim(train_pro)[2] -- Number of samples of train
  tN <- ifelse(tN == "Female", 1, 0)
```

```
  # Optim function --- with gradient
  optim.values <- (optim(par    = (aN),
                         fn     = (Laplace.aN),
                         gr     = (Gradient.a.N),
                         # further arguments to be passed to fn and gr
                         t.N    = (tN),
                         C.N    = (CN),
                         # quasi-Newton method
                         method = "BFGS",
                         # set control parameters
                         control= list(maxit   = 3000000, # set iteration limit
                                       fnscale = -1  # change optim to maximize fn
                                       )
                         ))

  # extract mode from the optimation result
  mode.N <- optim.values$par # mode.N
  return(mode.N)
}
```

Classification Function

```
GPC <- function(train, test, K, theta, aN, tN, Index.sample.test)  {

  # tN # vector of class labels
  target <- ifelse(tN == "Female", 1, 0)

  # mode from the optim function
  mode.N <- optim.GPC(train, test, K, theta, aN, tN)

  # k matrix associated with the test data
  matrix.k <- GPC.N_N2(train, test, K, theta)$k.N2

  # get Equation 6.87: The Expection -- mean
  E.a.Nplus1 <- t(matrix.k)%*%(target - sig.fun(mode.N))

  # Note that variance is also given by the GPC but not useful
  # so variance use not included in this code

  # we now put the mean in sigmoid function -- sig.fun
  # to get the required classification

  # get Equation 4.155: Probability of CLASS 1
  Prob.class1.due.to.mean <- sig.fun(E.a.Nplus1) # Required output

  # Classification by mean
  class.observation <- apply(Prob.class1.due.to.mean, 1, function(x)ifelse(x<0.5,"Male","Female"))


  # Create a data frame for the prob. of class of the classification due to the prob.
  Prob.out <- data.frame(Prob.CLASS = Prob.class1.due.to.mean,
                         CLASSES    = class.observation)

  # Index.sample.test # vector of ordered indices of which samples are being tested(predicted)
```

```
                    # this vector corresponds with index of samples to be tested

  tN <- data.frame(tN)
  tab <- table(Pred = Prob.out[,2], True_Class.of.train = tN[Index.sample.test,1])
  #
  return(list(tab = tab,
              Prob.out = Prob.out,
              train = train,
              test = test,
              K = K,
              theta = theta,
              aN = aN,
              tN = tN,
              Index = Index.sample.test
              ))
}
```

Function for getting the Error rate from the Gaussian Process

- It works for two situations:
  - Getting error rate from the model without a test data
  - Getting error rate given the test data set.

```
Error_rate = function(model , test.data = NULL){

  model # a model of calls "GPC" --- Gaussian Process Classifier
  test.data # this is a dataframe with sample labels as columns

  if(sum(test.data)==0){

  tab <- model$tab
  Pred <- model$Prob.out[,2]

   if(length(levels(Pred)) == 1 & all(levels(Pred) == "Male")){

      Error.rate1 =1- tab[2]/length(Pred)
      paste( "The error rate is ",Error.rate1)

    } else if(length(levels(Pred)) == 1 & all(levels(Pred) == "Female")){

      Error.rate2 = 1-tab[1]/length(Pred)
      paste( "The error rate is ",Error.rate2)

    } else if(length(levels(Pred)) == 2){
      Error.rate3 = 1 -  ((tab[1,1]+tab[2,2])/(length(train_cli[,2])))
    paste( "The error rate is ",Error.rate3)
    }

  } else if(class(test.data) == "data.frame" | sum(test.data) != 0) {

    test.er <- data.frame(test.data)
    train.er <- model$train
    K.er <- model$K
    theta.er <- model$theta
    aN.er <- model$aN
```

```
    tN.er <- model$tN
    Index.er <- model$Index.sample.test

    model.new <- GPC(train =  train.er,  test =  test.er, K =  K.er,
                     theta =  theta.er,aN =  aN.er, tN = tN.er,
                     Index.sample.test = Index.er)

    tab <- model.new$tab
    Pred <- model.new$Prob.out[,2]

   if(length(levels(Pred)) == 1 & all(levels(Pred) == "Male")){

      Error.rate1 =1- tab[2]/length(Pred)
      paste( "The error rate is ",Error.rate1)

    } else if(length(levels(Pred)) == 1 & all(levels(Pred) == "Female")){

      Error.rate2 = 1-tab[1]/length(Pred)
      paste( "The error rate is ",Error.rate2)

    } else if(length(levels(Pred)) == 2){
      Error.rate3 = 1 -  ((tab[1,1]+tab[2,2])/(length(train_cli[,2])))
    paste( "The error rate is ",Error.rate3)
    }

  }


}
```

Applying the Gaussian Process Classifier

- Gaussian Process Classifier then applied on the new data set with criterion of genes with a total of $\geqslant$ 100 reading of proteomics.
- The different examples are based on different choices or combination of kernels used and theta parameter which has a great influence on the classificaion.

```
######### checking the GPC classifier using -- train.pro as the test data ####################


# Refreshing our memory of the counts Gender classes
summary(train_cli[, 2])
```

```
## Female   Male
##     53     27
```

```
############## Testing the Classification  ##############

training_6 = new75[,6] # testing for just a single sample using the exponential kernel
res.GPC = GPC(new75, training_6, K = 3, theta = 1,
              aN = rep(0, dim(new75)[2]), tN = train_cli[,2], Index.sample.test = 6)
res.GPC$tab
```

```
##       True_Class.of.train
## Pred   Female Male
```

```
##   Male      0    1
```

```
Error_rate(res.GPC)
```

```
## [1] "The error rate is  0"
```

```
## Gaussian Kernel
res.GPC1 = GPC(new75, new75, K = 1, theta = 1,
               aN = rep(0,dim(new75)[2]), tN = train_cli[,2], 1:80 )
Pred1 <- res.GPC1$Prob.out[,2]
(tab1 <- res.GPC1$tab)
```

```
##         True_Class.of.train
## Pred     Female Male
##   Female     53    0
##   Male        0   27
```

```
Error_rate(res.GPC1)
```

```
## [1] "The error rate is  0"
```

```
## Correlation Kernel
res.GPC2 = GPC(new75, new75, K = 2, theta = 50,
               aN = rep(0,dim(new75)[2]), tN = train_cli[,2], 1:80)
Pred2 = res.GPC2$Prob.out[,2]
## Table of misclassification
(tab2 <- res.GPC2$tab)
```

```
##         True_Class.of.train
## Pred     Female Male
##   Female     53    0
##   Male        0   27
```

```
## Probability for checking how well the
## discrimination went
Error_rate(res.GPC2)
```

```
## [1] "The error rate is  0"
```

```
## Exponential Kernel
res.GPC3 = GPC(new75, new75, K = 3, theta = 5,
               aN = rep(0,dim(new75)[2]), tN = train_cli[,2], 1:80)
Pred3 = res.GPC3$Prob.out[,2]
## Table of misclassification
(tab3 <- res.GPC3$tab)
```

```
##         True_Class.of.train
## Pred     Female Male
##   Female     53    0
##   Male        0   27
```

```
## Probability for checking how well the
## discrimination went
Error_rate(res.GPC3)
```

```
## [1] "The error rate is  0"
```

```
## Sum of Gaussian and Correlation Kernel
res.GPC4 = GPC(new75, new75, K = 2, theta = 43,
               aN = rep(0,dim(new75)[2]), tN = train_cli[,2] , 1:80)
```

```
Pred4 = res.GPC4$Prob.out[,2]
## Table of misclassification
(tab4 <- res.GPC4$tab)
```

```
##         True_Class.of.train
## Pred     Female Male
##   Female     53    0
##   Male        0   27
```

```
## Probability for checking how well the
## discrimination went
Error_rate(res.GPC4)
```

```
## [1] "The error rate is  0"
```

```
## Sum of Correlation and Exponential Kernel
res.GPC5 = GPC(new75, new75, K = 5, theta = 1,
               aN = rep(0,dim(new75)[2]), tN = train_cli[,2] , 1:80)
Pred5 = res.GPC5$Prob.out[,2]
## Table of misclassification
(tab5 <- res.GPC5$tab)
```

```
##         True_Class.of.train
## Pred     Female Male
##   Female     53   10
##   Male        0   17
```

```
## Probability for checking how well the
## discrimination went
Error_rate(res.GPC5)
```

```
## [1] "The error rate is  0.125"
```

```
## Sum of Exponential  and Gaussian Kernel
res.GPC6 = GPC(new75, new75, K = 6, theta = 1,
               aN = rep(0,dim(new75)[2]), tN = train_cli[,2], 1:80 )
Pred6 = res.GPC6$Prob.out[,2]
## Table of misclassification
(tab6 <- res.GPC6$tab)
```

```
##         True_Class.of.train
## Pred     Female Male
##   Female     53    0
##   Male        0   27
```

```
## Probability for checking how well the
## discrimination went
Error_rate(res.GPC6)
```

```
## [1] "The error rate is  0"
```

- I have a 100% classification from my model when I used the **Gaussian Process Classification** method which involve using **Gaussian kernel function** below;

$$k(\boldsymbol{x}_n, \boldsymbol{x}_m) = exp\Big(-0.5(\boldsymbol{x}_n - \boldsymbol{x}_m)^T(\boldsymbol{x}_n - \boldsymbol{x}_m)\Big),$$

where $\boldsymbol{x}_n$,$\boldsymbol{x}_m$ are multidimensional vectors or $\boldsymbol{x}_n, \boldsymbol{x}_m \in R^D$.

in this case the dimension of the $\boldsymbol{x}_n$ and $\boldsymbol{x}_m$ are both 4118.

- The correlation kernel is the based on the standard Pearson correlation between two random variables. `R` calculates the Pearson correlation by default which is what we want.

$$Corr = \frac{cov(X, Y)}{\sigma_X \sigma_Y}$$

where cov is the covariance, σ_X is the standard deviation of X and σ_Y is the standard deviation of Y.

- The exponential kernel is given by the expression below

$$k(\boldsymbol{x}_n, \boldsymbol{x}_m) = exp\Big(- \theta |\boldsymbol{x}_n - \boldsymbol{x}_m|\Big)$$

where $n, m = 1, ..., D$, $D = 4118$ in the case for the number of dimensions of the date given.

Note:

- Tunning the value of θ in the exponential kernel has a strong influence of on the classification output of this Gaussian process.

Correlation Kernel

- The correlation kernel is not giving desired results. My suggestion for this poor performance of the correlation kernel is that by definition correlation searches for **linear relationship or association** between variables two random variables. Hence, if two variables in question a related by some quadratic or cubic relations, the correlation can't give good result for that case. This situation is true in our case. A quick check on the density plot of all the 80 samples reveal that the samples are **not linear related**.
- However, the whole results turned around when the a contant greater than one is used to multiplied the correlation kernel. So that we have

$$\theta \; Corr = \theta \; \frac{cov(X, Y)}{\sigma_X \sigma_Y}$$

'* The Gaussian process classifier does increasingly better and better until perfect prediction as the value, θ is rising. For my model, I realised that as I increased the value of the θ the classier moves form not recognising any males at all for $\theta = 1$, to missclassification of 7 males at $\theta = 10$ to misclassification of 4 of males at $\theta = 15$ and $\theta = 25$ to just 1 misclassfied male at $\theta = 30$ to perfect classification at $\theta = 35$.

- Of courses this is what we want but I'm now concerned about the **problem of overfitting*.

The test above using the Gaussian Process Classifier has been done using the reduced data set by criteria of row sum or total of proteomic reading for genes with sum $\geqslant 100$. This criterion result gives total of **2890** genes.

- Both the expential and Gaussian do perfect either alone or if added and both almost same prediction with used with the correlation kernel at around 90% success rates
- When correlation kernel is used the classifier see all samples as Females.

Using all the train_pro.tsv data set

- The different examples are based on different choices or combination of kernels used and also the choice of θ parameter used

```
# Freshing our memory of the counts given classes
summary(train_cli[, 2])
```

```
## Female   Male
##     53     27
```

```
## Gaussian Kernel
res.GPC = GPC(Imputed_train_pro, Imputed_train_pro,  K = 1, theta = 30,
              aN = rep(0 ,dim(Imputed_train_pro)[2]), tN = train_cli[,2] , 1:80)
Pred = res.GPC$Prob.out[,2]
## Table of misclassification
(tab <- res.GPC$tab)
```

```
##         True_Class.of.train
## Pred     Female Male
##   Female     53    0
##   Male        0   27
```

```
## Probability for checking how well the
## discrimination went
Error_rate(res.GPC)
```

```
## [1] "The error rate is  0"
```

```
## Correlation Kernel
system.time((res.GPC = GPC(Imputed_train_pro, Imputed_train_pro,  K = 2, theta = 1,
                           aN = rep(0 ,dim(Imputed_train_pro)[2]), tN = train_cli[,2], 1:80 )))
```

```
##    user  system elapsed
##   20.42    0.00   20.42
```

```
Pred = res.GPC$Prob.out[,2]
## Table of misclassification
(tab <- res.GPC$tab)
```

```
##         True_Class.of.train
## Pred     Female Male
##   Female     53   27
```

```
Error_rate(res.GPC)
```

```
## [1] "The error rate is  0.3375"
```

```
## Exponential Kernel
res.GPC = GPC(Imputed_train_pro, Imputed_train_pro,  K = 3, theta = 30,
              aN = rep(0 ,dim(Imputed_train_pro)[2]), tN = train_cli[,2] , 1:80 )
Pred = res.GPC$Prob.out[,2]
## Table of misclassification
(tab <- res.GPC$tab)
```

```
##         True_Class.of.train
## Pred     Female Male
##   Female     53    0
##   Male        0   27
```

```
## Probability for checking how well the
## discrimination went
Error_rate(res.GPC)
```

```
## [1] "The error rate is  0"
```

```
## Sum of Gaussian and Correlation Kernel
res.GPC = GPC(Imputed_train_pro, Imputed_train_pro,  K = 4, theta = 30,
              aN = rep(0 ,dim(Imputed_train_pro)[2]), tN = train_cli[,2] , 1:80)
Pred = res.GPC$Prob.out[,2]
```

```
## Table of misclassification
(tab <- res.GPC$tab)
```

```
##         True_Class.of.train
## Pred     Female Male
##   Female     53    0
##   Male        0   27
```

```
## Probability for checking how well the
## discrimination went
Error_rate(res.GPC)
```

```
## [1] "The error rate is  0"
```

```
## Sum of Correlation and Exponential Kernel
res.GPC = GPC(Imputed_train_pro, Imputed_train_pro, K = 5, theta = 30,
              aN = rep(0 ,dim(Imputed_train_pro)[2]), tN = train_cli[,2], 1:80 )
Pred = res.GPC$Prob.out[,2]
## Table of misclassification
(tab <- res.GPC$tab)
```

```
##         True_Class.of.train
## Pred     Female Male
##   Female     53    0
##   Male        0   27
```

```
## Probability for checking how well the
## discrimination went
 Error_rate(res.GPC)
```

```
## [1] "The error rate is  0"
```

```
## Sum of Exponential  and Gaussian Kernel
res.GPC = GPC(Imputed_train_pro, Imputed_train_pro, K = 6, theta = 30,
              aN = rep(0 ,dim(Imputed_train_pro)[2]), tN = train_cli[,2] , 1:80)
Pred = res.GPC$Prob.out[,2]
## Table of misclassification
(tab <- res.GPC$tab)
```

```
##         True_Class.of.train
## Pred     Female Male
##   Female     53    0
##   Male        0   27
```

```
## Probability for checking how well the
## discrimination went
Error_rate(res.GPC)
```

```
## [1] "The error rate is  0"
```

- The success rate goes down from 90% to 82.5% for the combined kernel correlation and Gaussian, Correlation and Exponention. However, the good nows is that both Guassian and exponential have 100% when used individual and when added togethor.

d). Evaluate model using leave-one-out cross validation (LOOCV). Compute error rate.

** Caution**: I built the Gaussian Process classification `function` to take the accept the data just like what we have been given. * Do not transpose the `train_pro.tsv` data, input the function as it is. * Do not

combine the classes or `train_cli.tsv` with the proteomics data. The function has a space in it to accept the

```
# Function for LOOCV for Gaussian Process Classification

LOOCV.GPC = function(train.CV, K.CV, theta.CV, aN.CV, tN.CV)
  {

  ## container for the error rate
  Error.rate = NULL
  Predict.GPC = NULL
  ERR = NULL
  GPC.model.CV = NULL
  tabble.list = list()
  Pred.list = list()

  for(i in 1 :dim(train.CV)[2]
      ){

          trainCV <- train.CV[,-i]
          test.CV <- data.frame(train.CV[,i])
          aNCV <- aN.CV[-i]
          tNCV <- tN.CV[-i]


        GPC.model.CV = GPC(trainCV, test = test.CV, K = K.CV, theta = theta.CV, aN = aNCV , tN = tNCV,i


        Predict.GPC = GPC.model.CV$Prob.out[,2]

        ## Table of misclassification
        (tab = GPC.model.CV$tab)
        ## Probability for checking how well the
        ## discrimination went
    if(length(levels(Predict.GPC)) == 1 & all(levels(Predict.GPC) == "Male")){

      Error.rate[i] =1- tab[2]/length(Predict.GPC)
      # update (Error.rate)

    } else if(length(levels(Predict.GPC)) == 1 & all(levels(Predict.GPC) == "Female")){

      Error.rate[i] = 1-tab[1]/length(Predict.GPC)
      # update (Error.rate)

    } else if(length(levels(Predict.GPC)) == 2){

      Error.rate[i] =1-  (tab[1,1] + tab[2,2])/length(Predict.GPC)
    # update (Error.rate)
    }

    tabble.list[[i]]  <- tab
    Pred.list[[i]] <- Predict.GPC
  }

  ER.mean = mean(Error.rate)
```

```
  AVG.error.rate <-  paste( "The average error rate is ",   ER.mean )

  return( list(AVG.error.rate = AVG.error.rate,
               Error.rate = Error.rate,
               tablelist = tabble.list, #list of tables for all the predictions
               PredList = Pred.list # list of all predictions
               ))

}


## system.time used to get the time spent to run the LOOCV function or loop
system.time((LOOCV.GPC.result1 = LOOCV.GPC( train = Imputed_train_pro, K.CV = 1, theta.CV = 40,
                                        aN.CV = rep(0, dim(Imputed_train_pro)[2]), tN.CV = train_cli[,2]
```

```
##    user  system elapsed
##  488.75    2.03  519.68
```

```
LOOCV.GPC.result1$Error.rate # vector of error each time it makes the prediction
```

```
##  [1] 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 1 0 0 1 0 1 0 1 1 0 0 1
## [36] 1 0 0 1 0 0 0 1 1 0 0 0 1 0 1 0 1 0 0 0 0 0 0 0 1 1 0 0 0 1 0 0 0 1 1 0
## [71] 0 1 1 0 1 0 0 1 0 1
```

```
LOOCV.GPC.result1$AVG.error.rate # average error rate
```

```
## [1] "The average error rate is  0.35"
```

```
# LOOCV.GPC.result1$tablelist # list of all the 80 tables of predictions
# LOOCV.GPC.result1$PredList # list of all the 80  predictions
table(LOOCV = unlist(LOOCV.GPC.result1$PredList), True_Class = train_cli[,2])
```

```
##         True_Class
## LOOCV    Female Male
##   Female     53   27
```

- Base on the LOOCV for all the dataset, it is noticed that the model makes a wrong prediction any time it is given a male sample to predict. However, each time given a female, it predicts correctly. The error rate is 33.75%.
- Using the reduce data set, the error rate was was 37.5%. This shows that the Gaussian Process Classification model does better in prediction when all the genes in the data set are used.

##2. Build alternative model

a). Read in data and preprocess (normalization, imputation, or digitalization)

- I have made use of random forest (rf) and support vector machines (svm). The svm performs gives perfect predictions most of the time and much more faster than any of the algorithm I have tried in this project.

b). Feature selection, if needed. (highly recommended as data is noisy)

- I have tried using the criterion of selecting only genes with a sum of 100 proteomic readings. However, my classification model still give a very good prediction with or without this criterion.

```r
library(randomForest)

data_Gender = data.frame(train_cli[,2])
rownames(data_Gender) <- train_cli[,1]
data_TRANSPOSED = data.frame(Gender = data_Gender, t(Imputed_train_pro))
kable(head(data_TRANSPOSED, 3)[1:6 , 1:6], caption = "First 6 rows and columns of the transposed data s
```

Table 6: First 6 rows and columns of the transposed data set

	train_cli...2.	A1BG	A2M	AAAS	AACS	AAGAB
Training_1	Female	2.919688	5.737663	2.002401	1.461113	0
Training_2	Female	3.753851	5.752416	1.090277	0.000000	0
Training_3	Male	3.513302	5.601927	2.789359	1.504314	0
NA	NA	NA	NA	NA	NA	NA
NA.1	NA	NA	NA	NA	NA	NA
NA.2	NA	NA	NA	NA	NA	NA

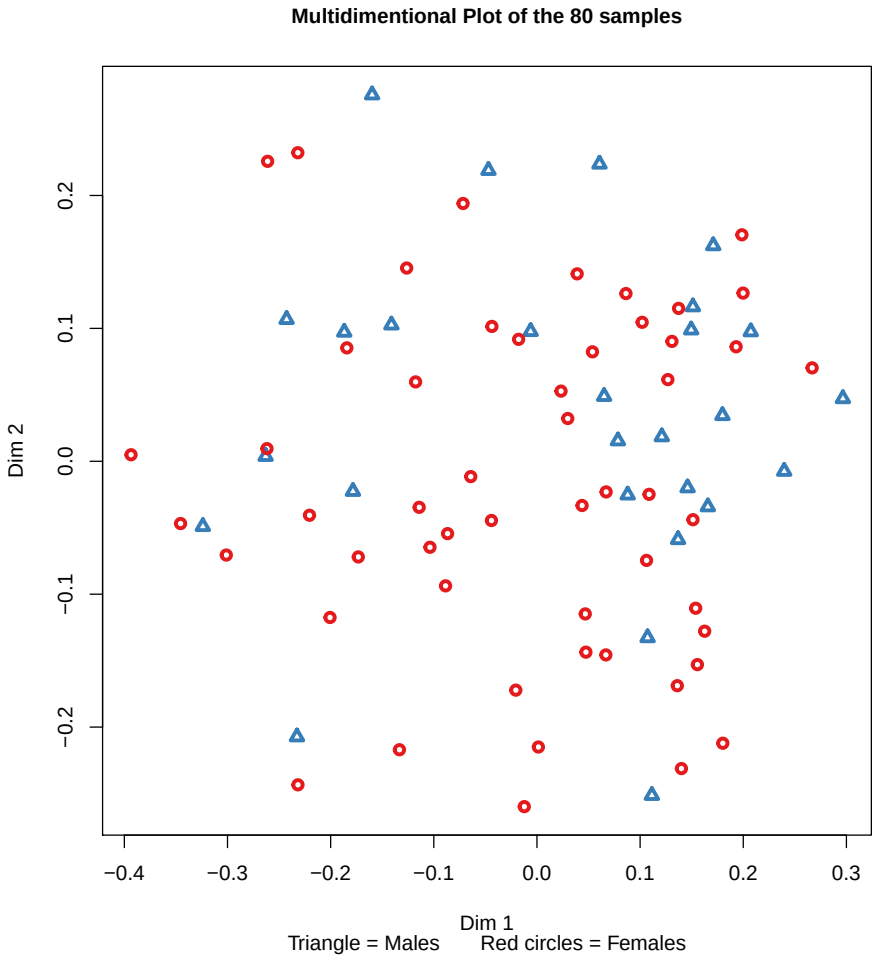
```r
system.time((train_RF <- randomForest(train_cli...2. ~ .,
                        data=data_TRANSPOSED, importance=TRUE,
                       proximity=TRUE)))
```

```
##    user  system elapsed
##   16.08    0.31   17.21
```

```r
print(train_RF)
```

```
##
## Call:
##  randomForest(formula = train_cli...2. ~ ., data = data_TRANSPOSED,      importance = TRUE, proximity
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 64
##
##         OOB estimate of  error rate: 31.25%
## Confusion matrix:
##        Female Male class.error
## Female     52    1  0.01886792
## Male       24    3  0.88888889
```

```r
## Unsupervised Random Forest
train_random = randomForest(t(Imputed_train_pro))

# summary(train_random)
MDSplot(train_random, train_cli[,2],
        main = "Multidimentional Plot of the 80 samples",cex.main = 1,
        sub = expression(paste("Triangle = Males", "       ", "Red circles = Females" )),
        pch = as.numeric(train_cli[,2]), lwd = 3 )
```



c). Build one model as proof of concept (just make it work)

- My model is based on support vector machines(svm)

Using Support Vector Machines

```
library("e1071")

train_SVM = svm(y = train_cli[,2], x = t(new75), kernel = "radial", gamma = 15)
# Predi = predict(train_SVM, train_cli[,2])
summary(train_SVM)
```

```
##
## Call:
## svm.default(x = t(new75), y = train_cli[, 2], kernel = "radial",
##     gamma = 15)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
```

```
##       gamma:  15
##
## Number of Support Vectors:  80
##
##  ( 53 27 )
##
##
## Number of Classes:  2
##
## Levels:
##  Female Male
```

```
pred <- fitted(train_SVM)
(tab = table(pred, True_Class = train_cli[,2]))
```

```
##         True_Class
## pred     Female Male
##   Female     53    0
##   Male        0   27
```

```
## error rate
paste("The error rate is  " , 1 - sum(tab[1,1] , tab[2,2])/length(pred))
```

```
## [1] "The error rate is   0"
```

```
## What if the kernel is changed to sigmoid
base::system.time((train_SVM = svm(y = train_cli[,2], x = t(new75), kernel = "polynomial", gamma = 15))
```

```
##    user  system elapsed
##    0.66    0.00    0.66
```

```
# Predi = predict(train_SVM, train_cli[,2])
summary(train_SVM)
```

```
##
## Call:
## svm.default(x = t(new75), y = train_cli[, 2], kernel = "polynomial",
##     gamma = 15)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  1
##      degree:  3
##       gamma:  15
##      coef.0:  0
##
## Number of Support Vectors:  80
##
##  ( 53 27 )
##
##
## Number of Classes:  2
##
## Levels:
##  Female Male
```

```r
pred <- fitted(train_SVM)
(tab = table(pred, train_cli[,2]))
```

```
##
## pred     Female Male
##   Female     53    0
##   Male        0   27
```

```r
## error rate
paste("The error rate is  " ,  1 - sum(tab[1,1] , True_Class = tab[2,2])/length(pred))
```

```
## [1] "The error rate is   0"
```

- There is perfect prediction by the support vector machine algorithm.

The support vector machine the fastest when it comes to system running time with a speed of relatively $\frac{1}{15}$ the time used by random Frorest and $\frac{1}{10}$ the time used by the Gaussian process classifier. It's gives perfect predictions when the γ parameter is increased. Here, I used two basis functions, the `radial basis` (which is link to the Euclidean distance) function or kernel and the the polynomial kernel. The polynomial also gives perfect prediction when the γ parameter is increased.

d). Evaluate model using leave-one-out cross validation (LOOCV). Compute error rate.

Here us we use the LOOCV for two cases: where radial basis function is used and where sigmoid function is used. The LOOCV shows that support vector machine with radial basis gives are perfect prediction with no prediction errors. The γ parameter also plays a role in the prediction of classes.

```r
LOOCV.SVM = function(train, test = 1,tN, KERN = "radial", GAM = 100){
  ### Note
  # Both train and test are dataframe with sample labels as columns

  ## container for the error rate
  Error.rate = NULL
  train_SVM = NULL
  predict.SVM = NULL
  tab = NULL
  tN = data.frame(tN)
  transposed.dat = data.frame(t(train))
  test.transposed = t(test)

  for(i in 1 : dim(train)[2]
      ){

  train_SVM = svm(y = tN[-i,1], x = transposed.dat[-i, ],kernel = KERN, gamma = GAM)

    # summary(train_SVM)
    if(sum(is.null( dim(test)[2])) == 1){
    predict.SVM <- fitted(train_SVM)

    } else if (any(class(test) == "data.frame")| sum(test) != 1){

      testdat = test.transposed[-i,]

      predict.SVM <- predict(train_SVM, testdat )
    }
```

```
    # length(predict.SVM)

    tab = table(predict.SVM, train_cli[-i,2])

    ## error rate
    Error.rate[i] =   (1 - sum(tab[1,1] , tab[2,2])/length(predict.SVM))

  }


  return(list( Error.rate  = Error.rate,
               ER.mean = mean(Error.rate)))
}
```

```
## system.time is used to know the time used by the system to finish the LOOCV
system.time((result.LOOCV = LOOCV.SVM(train = new75, test = new75, tN = train_cli[,2])))
result.LOOCV
# $`Error.rate`
#  [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
# [57] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
#
# $ER.mean
# [1] 0


## when Radial basis is used and Gamma set to 15
system.time((result.LOOV = LOOCV.SVM(train = new75, test = new75, KERN = "polynomial", GAM = 15, tN = t
result.LOOV
```

NB

- The code for LOOCV using the svm function gives error so I looped through the samples dropping one at a time and predicting the rest.

##3. Final prediction.

a). Choose and train best model. As the training dataset contains mislabeled samples, you may want to exclude a small number of training samples that are predicted wrong in the LOOCV.

- The model I that works best for me is the Gaussian Process Classifier.
- The training above:
  - Increasing the theta parameter works better
  - Predictions was better with more genes than less genes based on the LOOCV
  - More males we predicted as females in from the LOOCV.
  - Females we most of the times predicted correctly

```
system.time((LOOCV.GPC.result2 = LOOCV.GPC( train = Imputed_train_pro, K.CV = 2, theta.CV = 60,
                                          aN.CV = rep(0, dim(new75)[2]), tN.CV = train_cli[,2])))
LOOCV.GPC.result2$Error.rate # vector of error each time it makes the prediction
LOOCV.GPC.result2$AVG.error.rate # average error rate
# LOOCV.GPC.result$tablelist # list of all the 80 tables of predictions
# LOOCV.GPC.result$PredList # list of all the 80  predictions
```

```r
## Using the reduced reduced data set
## A table showing how each sample taken out was predicted

table(LOOCV = unlist(LOOCV.GPC.result2$PredList), True_Class = train_cli[,2])




system.time((LOOCV.GPC.result3 = LOOCV.GPC(train = Imputed_train_pro, K.CV = 3, theta.CV = 60,
                                           aN.CV = rep(0, dim(new75)[2]), tN.CV = train_cli[,2])))
LOOCV.GPC.result3$Error.rate # vector of error each time it makes the prediction
LOOCV.GPC.result3$AVG.error.rate # average error rate
# LOOCV.GPC.result$tablelist # list of all the 80 tables of predictions
# LOOCV.GPC.result$PredList # list of all the 80  predictions

## Using the reduced reduced data set
## A table showing how each sample taken out was predicted

table(LOOCV = unlist(LOOCV.GPC.result3$PredList), True_Class = train_cli[,2])


system.time((LOOCV.GPC.result4 = LOOCV.GPC(train = Imputed_train_pro, K.CV = 4, theta.CV = 60,
                                           aN.CV = rep(0, dim(new75)[2]), tN.CV = train_cli[,2])))
LOOCV.GPC.result4$Error.rate # vector of error each time it makes the prediction
LOOCV.GPC.result4$AVG.error.rate # average error rate
# LOOCV.GPC.result$tablelist # list of all the 80 tables of predictions
# LOOCV.GPC.result$PredList # list of all the 80  predictions

## Using the reduced reduced data set
## A table showing how each sample taken out was predicted

table(LOOCV = unlist(LOOCV.GPC.result4$PredList), True_Class = train_cli[,2])




system.time((LOOCV.GPC.result5 = LOOCV.GPC(train = Imputed_train_pro, K.CV = 5, theta.CV = 60,
                                           aN.CV = rep(0, dim(new75)[2]), tN.CV = train_cli[,2])))
LOOCV.GPC.result5$Error.rate # vector of error each time it makes the prediction
LOOCV.GPC.result5$AVG.error.rate # average error rate
# LOOCV.GPC.result$tablelist # list of all the 80 tables of predictions
# LOOCV.GPC.result$PredList # list of all the 80  predictions

## Using the reduced reduced data set
## A table showing how each sample taken out was predicted

table(LOOCV = unlist(LOOCV.GPC.result5$PredList), True_Class = train_cli[,2])




system.time((LOOCV.GPC.result6 = LOOCV.GPC(train = Imputed_train_pro, K.CV = 6, theta.CV = 40,
                                           aN.CV = rep(0, dim(new75)[2]), tN.CV = train_cli[,2])))
```

```
LOOCV.GPC.result6$Error.rate # vector of error each time it makes the prediction
LOOCV.GPC.result6$AVG.error.rate # average error rate
# LOOCV.GPC.result$tablelist # list of all the 80 tables of predictions
# LOOCV.GPC.result$PredList # list of all the 80  predictions

## Using the reduced reduced data set
## A table showing how each sample taken out was predicted

table(LOOCV = unlist(LOOCV.GPC.result6$PredList), True_Class = train_cli[,2])


indices = which(rowSums( cbind(LOOCV.GPC.result1$Error.rate, LOOCV.GPC.result2$Error.rate,
               LOOCV.GPC.result3$Error.rate,LOOCV.GPC.result4$Error.rate,
               LOOCV.GPC.result5$Error.rate, LOOCV.GPC.result6$Error.rate)) == 6)
# result of above code
# indices = c(2,  5 ,14, 19 ,23 ,24 ,27 ,29 ,32 ,35 ,36, 43, 44 ,48 ,50, 52, 59, 64 ,68, 69 ,72, 73, 80
write.csv(indices, "indices.csv")
```

Since the LOOCV takes a lot of time to finish running, I have set `eval = F` for five of the LOOCV and left only one run in order for you to see the result but once you have my file, you can run it by your self to see all the result.

- 23 training samples selected based on the LOOCV
- The samples selected based on some voting process. These are samples for which the six different kernel approaches wrongly predict in the LOOCV stage.
- Two options to handle this is to either drop them or switch their gender labels since we know that some of the training samples were wrongly labeled.
- the following lines of code just create a file for the gender labels that were predicted wrongly

```
library(dplyr)
indices = c(2,  5 ,14, 19 ,23 ,24 ,27 ,29 ,32 ,35 ,36, 43, 44 ,48 ,50, 52, 59, 64 ,68, 69 ,72, 73, 80)
length(indices)
```

```
## [1] 23
```

```
WronglyPredictedSamples <- paste0("Training_", indices)

gender.correctly.predicted <- slice(train_cli[,1:2], -indices)
gender.correctly.predicted$mismatch <- rep(0, 57)
gender.wrongly.predicted <- slice(train_cli[,1:2], indices)
gender.wrongly.predicted$mismatch <- rep(1, 23)
gender_mismatch_training <- rbind(gender.wrongly.predicted, gender.correctly.predicted )
gender_mismatch_training <- gender_mismatch_training[,c(1,3)]
gender_mismatch_training <- arrange(gender_mismatch_training, sample)
sliced.sorted11 = slice(gender_mismatch_training, c(1,12,23,34,45,56,67,78,80))
sliced.sorted22 = slice(gender_mismatch_training, -c(1,12,23,34,45,56,67,78,80))
(gender_mismatch_training <- rbind(sliced.sorted11,sliced.sorted22))
```

```
##         sample mismatch
## 1   Training_1        0
## 2   Training_2        1
## 3   Training_3        0
## 4   Training_4        0
## 5   Training_5        1
```

```
## 6   Training_6        0
## 7   Training_7        0
## 8   Training_8        0
## 9   Training_9        0
## 10 Training_10        0
## 11 Training_11        0
## 12 Training_12        0
## 13 Training_13        0
## 14 Training_14        1
## 15 Training_15        0
## 16 Training_16        0
## 17 Training_17        0
## 18 Training_18        0
## 19 Training_19        1
## 20 Training_20        0
## 21 Training_21        0
## 22 Training_22        0
## 23 Training_23        1
## 24 Training_24        1
## 25 Training_25        0
## 26 Training_26        0
## 27 Training_27        1
## 28 Training_28        0
## 29 Training_29        1
## 30 Training_30        0
## 31 Training_31        0
## 32 Training_32        1
## 33 Training_33        0
## 34 Training_34        0
## 35 Training_35        1
## 36 Training_36        1
## 37 Training_37        0
## 38 Training_38        0
## 39 Training_39        0
## 40 Training_40        0
## 41 Training_41        0
## 42 Training_42        0
## 43 Training_43        1
## 44 Training_44        1
## 45 Training_45        0
## 46 Training_46        0
## 47 Training_47        0
## 48 Training_48        1
## 49 Training_49        0
## 50 Training_50        1
## 51 Training_51        0
## 52 Training_52        1
## 53 Training_53        0
## 54 Training_54        0
## 55 Training_55        0
## 56 Training_56        0
## 57 Training_57        0
## 58 Training_58        0
## 59 Training_59        1
```

```
## 60 Training_60        0
## 61 Training_61        0
## 62 Training_62        0
## 63 Training_63        0
## 64 Training_64        1
## 65 Training_65        0
## 66 Training_66        0
## 67 Training_67        0
## 68 Training_68        1
## 69 Training_69        1
## 70 Training_70        0
## 71 Training_71        0
## 72 Training_72        1
## 73 Training_73        1
## 74 Training_74        0
## 75 Training_75        0
## 76 Training_76        0
## 77 Training_77        0
## 78 Training_78        0
## 79 Training_79        0
## 80 Training_80        1
```

```
write.csv(gender_mismatch_training, "gender_mismatch_training.csv", sep = ",")
```

The following 23 samples out of 80 are countinuously predicted wrongly by all choices of kernel.

(Training_2, Training_5, Training_14, Training_19, Training_23, Training_24, Training_27, Training_29, Training_32, Training_35, Training_36, Training_43, Training_44, Training_48, Training_50, Training_52, Training_59, Training_64, Training_68, Training_69, Training_72, Training_73, Training_80).

b) Make prediction on test data.

- The sample lables are then switched
- The following lines of code just switch the gender labels that were predicted wrongly

```
# the following lines of code just switch the gender labels that were predicted wrongly
library(dplyr)
samples.mismatch = train_cli[indices, 1:2]
samples.switch = as.factor(ifelse(train_cli[indices,2] == "Female", "Male", "Female"))
samples.mismatch$correct.match = samples.switch
samples.mismatch = samples.mismatch[, c(1,3)]
samples.filtered = dplyr::slice(train_cli[,1:2], -indices)
samples.filtered = rename(samples.filtered, correct.match = gender)
train_cli_corrected_gender = dplyr::union(samples.filtered, samples.mismatch )
sorted = arrange(train_cli_corrected_gender, sample)
sliced.sorted1 = slice(sorted, c(1,12,23,34,45,56,67,78,80))
sliced.sorted2 = slice(sorted, -c(1,12,23,34,45,56,67,78,80))
Gender.switched = rbind(sliced.sorted1, sliced.sorted2)
```

- Train Model using corrected sample labels

```
## Gaussian Kernel
train.GPC = GPC(Imputed_train_pro, Imputed_train_pro, K = 1, theta = 30,
              aN = rep(0 ,dim(Imputed_train_pro)[2]), tN = Gender.switched[,2], 1:80 )
Pred = train.GPC$Prob.out[,2]
```

```
## Table of misclassification
(tab <- train.GPC$tab)
```

```
##         True_Class.of.train
## Pred     Female Male
##   Female     42    0
##   Male        0   38
```

```
## Probability for checking how well the
## discrimination went
Error_rate(res.GPC)
```

```
## [1] "The error rate is  0"
```

```
table(samples.mismatch[,2])
```

```
##
## Female   Male
##      6     17
```

- Making predictions using Gaussian Kernel class labels as given in the train

```
## Gaussian Kernel
train.GPC = GPC(Imputed_train_pro, Imputed_test_pro, K = 1, theta = 30,
              aN = rep(0 ,dim(Imputed_train_pro)[2]), tN = train_cli[,2], 1:80 )
Pred = train.GPC$Prob.out[,2]
## Table of misclassification

tab <- train.GPC$tab

## Probability for checking how well the
## discrimination went
Error_rate(train.GPC)
```

```
## [1] "The error rate is  0.3375"
```

```
## Compare Predictions with labels of the test
tab;table(Pred, TrueClass.of.test = test_cli[,2])
```

```
##         True_Class.of.train
## Pred     Female Male
##   Female     53   27
```

```
##         TrueClass.of.test
## Pred     Female Male
##   Female     31   49
```

- Making predictions using Gaussian Kernel class labels switched

```
## Gaussian Kernel
train.GPC1 = GPC(Imputed_train_pro, Imputed_test_pro, K = 1, theta = 30,
              aN = rep(0 ,dim(Imputed_train_pro)[2]), tN = Gender.switched[,2], 1:80 )
Pred1 = train.GPC1$Prob.out[,2]
## Table of misclassification
tab1 <- train.GPC1$tab
## Probability for checking how well the
## discrimination went
Error_rate(train.GPC1)
```

```
## [1] "The error rate is  0.475"
```

```
## Compare Predictions with labels of the test
tab1;table(Pred1, TrueClass.of.test = test_cli[,2])
```

```
##         True_Class.of.train
## Pred     Female Male
##   Female     42   38
```

```
##         TrueClass.of.test
## Pred1    Female Male
##   Female     31   49
```

- In all the two cases the model classify every sample as **Female**.
- However, the model shows perfect predictions when trained and tested using the test_pro.tsv alone. This the same when you use the train_pro.tsv alone.

Created the mismatch by just doing a one time prediction on the test data

```
mismatch <- ifelse(test_cli[,2] == "Female", 0,1)
sample <- as.factor(paste0("Training_", 1:80))


gender_mismatch_testing <- data.frame(sample,mismatch)

# write.csv(gender_mismatch_testing, "gender_mismatch_testing.csv", sep = ",")
#
# tttt = read.table("~/Renvironment/gender_mismatch_testing.csv", header = T, sep = ",")
# gender_mismatch_testing = tttt[,2:3]
```

- I thought it is good to use the same LOOCV to find the mismatched samples in the testing data.
- I used the LOOCV function to get LOOCV prediction on the testing samples.

```
system.time((LOOCV.GPC.result.test = LOOCV.GPC(train = Imputed_test_pro, K.CV = 2, theta.CV = 40,
                                        aN.CV = rep(0, dim(new75)[2]), tN.CV = test_cli[,2])))
```

```
##    user  system elapsed
##  460.40    2.61  479.83
```

```
(LOOCV.GPC.result.test$Error.rate )# vector of error each time it makes the prediction
```

```
##  [1] 0 1 0 1 0 1 1 0 1 0 1 0 0 1 1 1 0 1 0 1 1 0 0 0 0 1 0 1 0 0 0 0 1 1 1
## [36] 0 0 1 1 1 1 0 0 1 1 1 0 1 0 0 0 1 0 0 1 0 1 0 1 0 1 1 0 0 0 0 0 1 0 0
## [71] 0 1 0 0 1 0 0 0 1 1
```

```
LOOCV.GPC.result.test$AVG.error.rate # average error rate
```

```
## [1] "The average error rate is  0.45"
```

```
# LOOCV.GPC.result.test$tablelist # list of all the 80 tables of predictions
# LOOCV.GPC.result.test$PredList # list of all the 80  predictions

## A table showing how each sample taken out was predicted
LOOCV.Pred.test = unlist(LOOCV.GPC.result.test$PredList)
table(LOOCV.Pred.test = unlist(LOOCV.GPC.result.test$PredList), True_Class.test = test_cli[,2])
```

```
##                 True_Class.test
## LOOCV.Pred.test Female Male
##          Male       26   40
##          Female      5    9
```

```r
(Prediction.table.test = data.frame( test_cli[,1:2], LOOCV.Pred.test))
```

```
##        sample gender LOOCV.Pred.test
## 1   Testing_1 Female            Male
## 2   Testing_2   Male            Male
## 3   Testing_3 Female            Male
## 4   Testing_4   Male            Male
## 5   Testing_5 Female            Male
## 6   Testing_6   Male          Female
## 7   Testing_7   Male          Female
## 8   Testing_8   Male          Female
## 9   Testing_9 Female            Male
## 10 Testing_10 Female            Male
## 11 Testing_11   Male          Female
## 12 Testing_12   Male            Male
## 13 Testing_13   Male            Male
## 14 Testing_14   Male            Male
## 15 Testing_15 Female            Male
## 16 Testing_16 Female            Male
## 17 Testing_17 Female            Male
## 18 Testing_18   Male            Male
## 19 Testing_19 Female            Male
## 20 Testing_20   Male            Male
## 21 Testing_21 Female            Male
## 22 Testing_22 Female            Male
## 23 Testing_23   Male            Male
## 24 Testing_24   Male            Male
## 25 Testing_25   Male            Male
## 26 Testing_26   Male            Male
## 27 Testing_27 Female            Male
## 28 Testing_28   Male            Male
## 29 Testing_29 Female            Male
## 30 Testing_30   Male            Male
## 31 Testing_31   Male            Male
## 32 Testing_32   Male            Male
## 33 Testing_33   Male            Male
## 34 Testing_34 Female            Male
## 35 Testing_35 Female          Female
## 36 Testing_36   Male          Female
## 37 Testing_37 Female            Male
## 38 Testing_38   Male            Male
## 39 Testing_39 Female          Female
## 40 Testing_40   Male            Male
## 41 Testing_41 Female            Male
## 42 Testing_42 Female          Female
## 43 Testing_43 Female            Male
## 44 Testing_44   Male            Male
## 45 Testing_45 Female            Male
## 46 Testing_46 Female            Male
## 47 Testing_47 Female            Male
## 48 Testing_48   Male            Male
## 49 Testing_49 Female            Male
## 50 Testing_50   Male            Male
## 51 Testing_51   Male            Male
```

```
## 52 Testing_52   Male          Female
## 53 Testing_53   Male            Male
## 54 Testing_54   Male          Female
## 55 Testing_55 Female            Male
## 56 Testing_56 Female            Male
## 57 Testing_57   Male          Female
## 58 Testing_58   Male            Male
## 59 Testing_59   Male          Female
## 60 Testing_60   Male            Male
## 61 Testing_61   Male            Male
## 62 Testing_62 Female          Female
## 63 Testing_63   Male            Male
## 64 Testing_64   Male            Male
## 65 Testing_65   Male            Male
## 66 Testing_66   Male            Male
## 67 Testing_67   Male            Male
## 68 Testing_68   Male            Male
## 69 Testing_69 Female            Male
## 70 Testing_70   Male            Male
## 71 Testing_71   Male            Male
## 72 Testing_72   Male            Male
## 73 Testing_73 Female            Male
## 74 Testing_74   Male            Male
## 75 Testing_75   Male            Male
## 76 Testing_76 Female            Male
## 77 Testing_77   Male            Male
## 78 Testing_78   Male            Male
## 79 Testing_79   Male            Male
## 80 Testing_80 Female          Female
```

```
mismatch.test <- ifelse(Prediction.table.test$gender == Prediction.table.test$LOOCV.Pred.test, 0,1)
sample <- as.factor(paste0("Training_", 1:80))
(gender_mismatch_testing <- data.frame(sample, mismatch = mismatch.test))
```

```
##         sample mismatch
## 1   Training_1        1
## 2   Training_2        0
## 3   Training_3        1
## 4   Training_4        0
## 5   Training_5        1
## 6   Training_6        1
## 7   Training_7        1
## 8   Training_8        1
## 9   Training_9        1
## 10 Training_10        1
## 11 Training_11        1
## 12 Training_12        0
## 13 Training_13        0
## 14 Training_14        0
## 15 Training_15        1
## 16 Training_16        1
## 17 Training_17        1
## 18 Training_18        0
## 19 Training_19        1
## 20 Training_20        0
```

```
## 21 Training_21        1
## 22 Training_22        1
## 23 Training_23        0
## 24 Training_24        0
## 25 Training_25        0
## 26 Training_26        0
## 27 Training_27        1
## 28 Training_28        0
## 29 Training_29        1
## 30 Training_30        0
## 31 Training_31        0
## 32 Training_32        0
## 33 Training_33        0
## 34 Training_34        1
## 35 Training_35        0
## 36 Training_36        1
## 37 Training_37        1
## 38 Training_38        0
## 39 Training_39        0
## 40 Training_40        0
## 41 Training_41        1
## 42 Training_42        0
## 43 Training_43        1
## 44 Training_44        0
## 45 Training_45        1
## 46 Training_46        1
## 47 Training_47        1
## 48 Training_48        0
## 49 Training_49        1
## 50 Training_50        0
## 51 Training_51        0
## 52 Training_52        1
## 53 Training_53        0
## 54 Training_54        1
## 55 Training_55        1
## 56 Training_56        1
## 57 Training_57        1
## 58 Training_58        0
## 59 Training_59        1
## 60 Training_60        0
## 61 Training_61        0
## 62 Training_62        0
## 63 Training_63        0
## 64 Training_64        0
## 65 Training_65        0
## 66 Training_66        0
## 67 Training_67        0
## 68 Training_68        0
## 69 Training_69        1
## 70 Training_70        0
## 71 Training_71        0
## 72 Training_72        0
## 73 Training_73        1
## 74 Training_74        0
```

```
## 75 Training_75        0
## 76 Training_76        1
## 77 Training_77        0
## 78 Training_78        0
## 79 Training_79        0
## 80 Training_80        0
```

```
write.csv(gender_mismatch_testing, "gender_mismatch_testing.csv", sep = ",")
```

Part C. Reproducibility check

Part D. Predict microsatellite instability (MSI) status in cancer

Follow the same steps for Part B and submit 4 files. The two CSV files should be named ???MSI_mismatch_training.csv??? and ???MSI_mismatch_testing.csv???.

##1. Build first model

a). Read in data and preprocess (normalization, imputation, or digitalization)

b). Feature selection, if needed. (highly recommended as data is noisy)

c). Build one model as proof of concept (just make it work)

d). Evaluate model using leave-one-out cross validation (LOOCV). Compute error rate.

##2. Build alternative model

a). Read in data and preprocess (normalization, imputation, or digitalization)

b). Feature selection, if needed. (highly recommended as data is noisy)

c). Build one model as proof of concept (just make it work)

d). Evaluate model using leave-one-out cross validation (LOOCV). Compute error rate.

##3. Final prediction.

a). Choose and train best model. As the training dataset contains mislabeled samples, you may want to exclude a small number of training samples that are predicted wrong in the LOOCV.

b) Make prediction on test data.

Part E. Combine results of gender and MSI status predictions

##1. Compare LOOCV results from gender model and MSI model to see if the same training samples are mislabeled for both gender and MSI status. ##2. Combine predictions results of both gender and

MSI status models and generate one file with mislabeled test samples. Use this format and name this file: final_mismatch.csv