Jeffrey Kerley
Jakcqc

a) Finding mahal squared distance from two points in same distribution

```
mahalTwoPoints <- function(x1,x2, sigma)
{
mahalD = sqrt(t(x1-x2) %*% solve(sigma) %*% (x1 - x2))
return(mahalD)
}
```

b) Finding generic descriminant function

```
getDiscrim <- function(x, mu, sigma, prior, d)
{
 dG = -0.5*(t(x-mu))%*% solve(sigma)%*%(x - mu)
     - (d/2)*log(2*pi)-0.5*log(det(sigma))
     + log(prior)

 return(dG)
}
```

c) Generate sample points based on above criteria:

```
generateSample <- function(eigMat, eigVal, d, n, mu)
{
 testX <- matrix(rnorm(d * n), n)

 eigValDiag1 = diag(sqrt(c(eigVal[,1],eigVal[,2])), d)

 testX = mu + eigMat %*% eigValDiag1 %*% t(testX)

 return(t(testX))
 }
```

Below we can see the two newly generated classes, as well as the probability distributions on an x and y. I could not figure out how to map the probability Z values to

my x,y and graph it in 3D, but I do see how it would be generated from this picture:

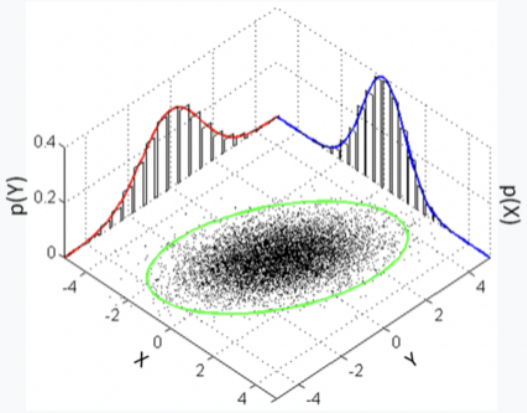
a

m the

elated

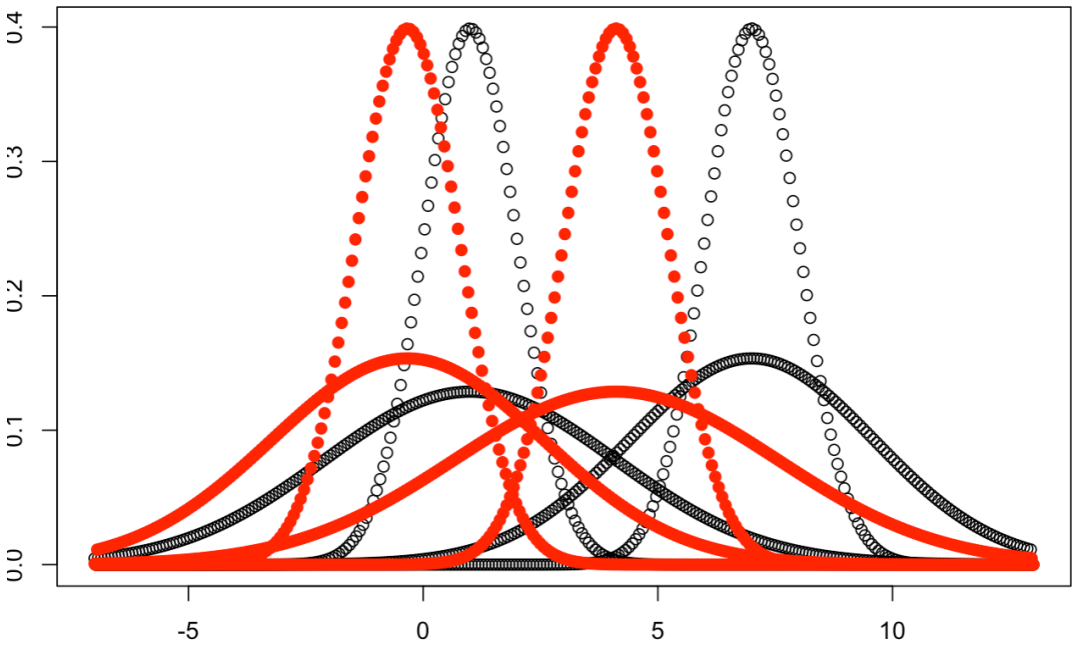
Multivariate normal

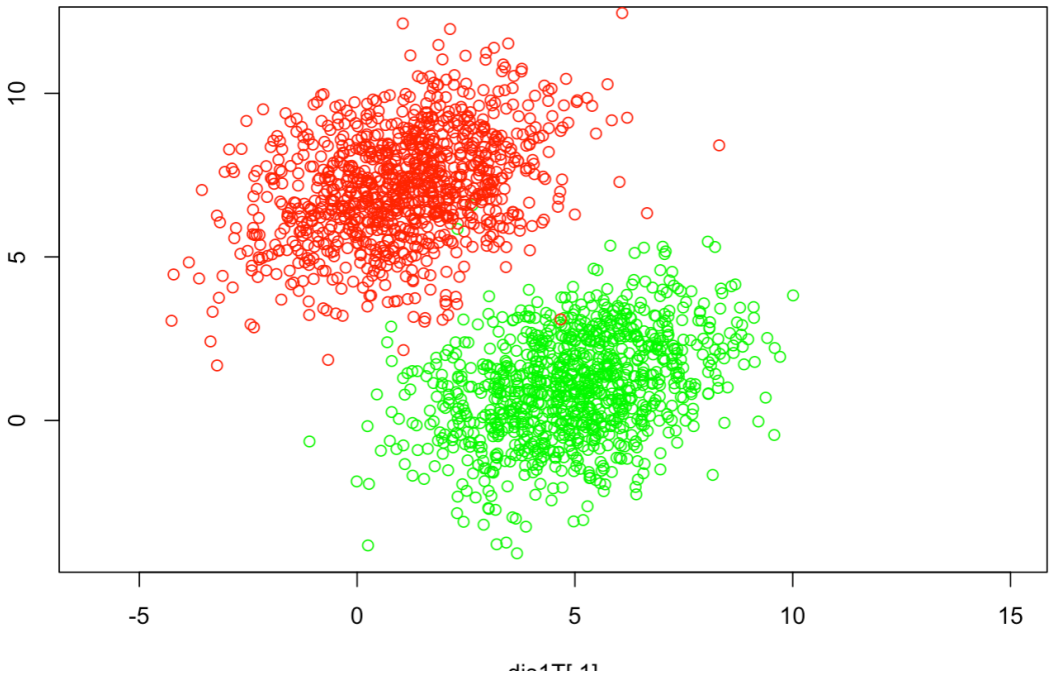
Probability density function



Many sample points from a multivariate normal distribution with $\boldsymbol{\mu} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ and $\boldsymbol{\Sigma} = \begin{bmatrix} 1 & 3/5 \\ 3/5 & 2 \end{bmatrix}$, shown along with the 3-sigma ellipse, the two marginal distributions, and the two 1-d histograms.

Thus, all we would need to do is project the z axis from the y value of our probabilities.


0.0
0.1
0.2
0.3
0.4
-5
0
5
10
seq(-7, 13, length = 1099)


0
5
10
-5
0
5
10
15

d) To find our decision boundary for the two classes, g, we do g = g1 - g2, where g1 and g2 are the generic discriminant function for each class. This can then be projected into 2d space as a decision boundary.

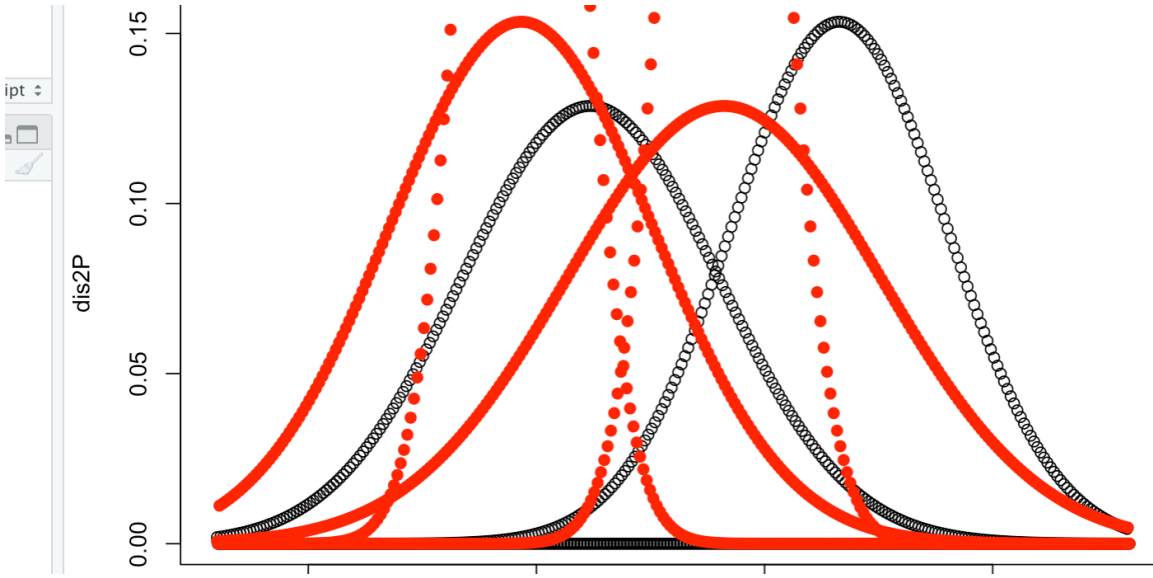
e) For plotting the posterior probabilities, we simply prior probability, which is .8 for class 2, and .2 for class 1, and add our evidence (which is our likelihood)
We can define this as:

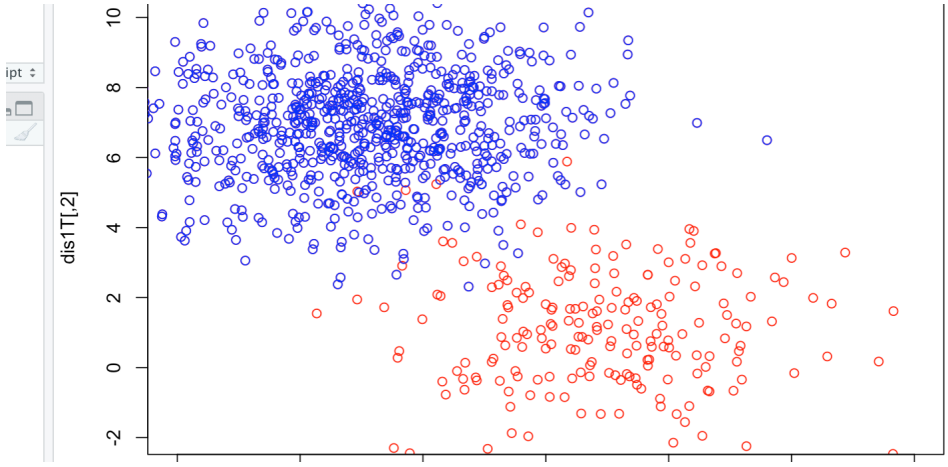
By Bayes' theorem, the posterior distribution can be written as

$$p(\theta \mid X_1, \ldots, X_n) = \frac{p(X_1, \ldots, X_n \mid \theta)\pi(\theta)}{p(X_1, \ldots, X_n)} = \frac{\mathcal{L}_n(\theta)\pi(\theta)}{c_n}$$

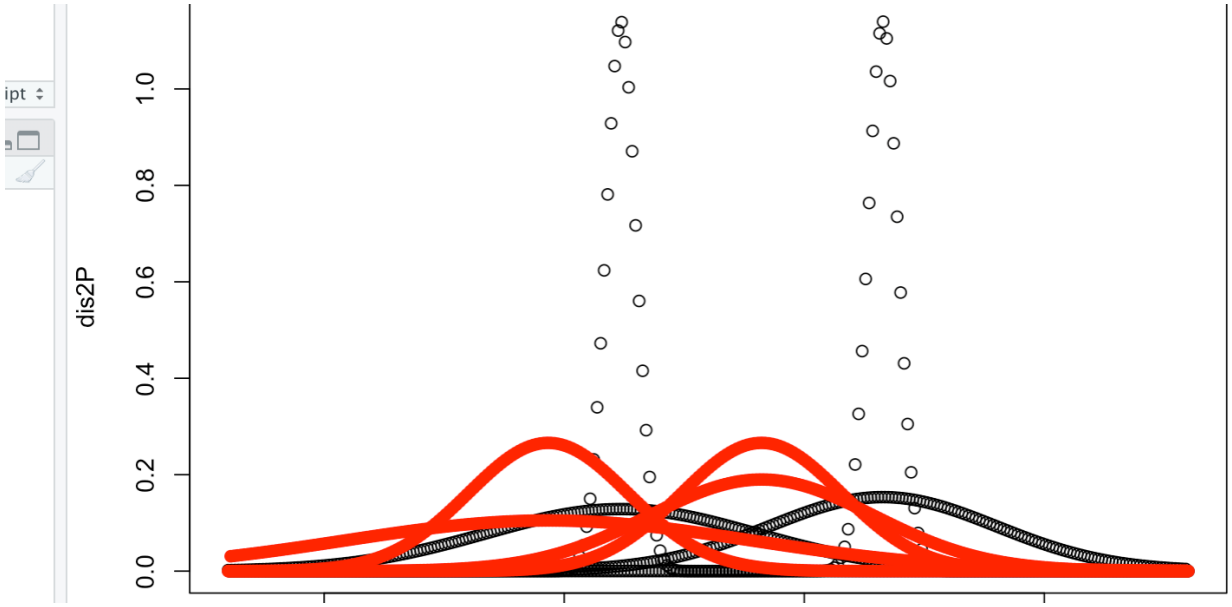
We can do this by substituting into this function our probabilities and probability sum of our generated data. So p(X1…Xn) would be the sum of all our data points probability given our prior distribution.

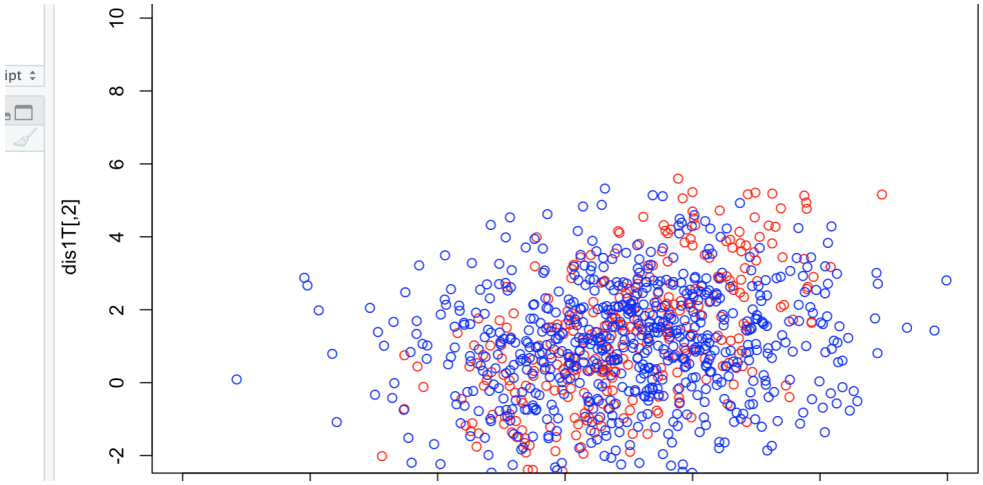
f) This sigma returns a [-1,0]
[0,-1] eigen vector matrix. As a result, our values are the same as those listed in the sigma.
Below we have the generated classes and likelihood distribution





g) Plotting the final set of sigmas with a 1/3P + 2/3P = 1





Here we see how relationship between variables can effect a generated distribution.

CODE BELOW:

```
##libs
library(MASS)
library(vegan)
library(plot3D)
##functions
#QR decomposition to solve for 5 eigen vectors
getEigOrtho = function(covMat, colSize)
{
  Q = calcQMatrix(covMat, colSize)
  counterNew = 0
  while(counterNew < 50000)
  {
    baseQ = Q
    cMat = covMat %*% Q
    Q = calcQMatrix(cMat, colSize)
    checker = all.equal(Q, baseQ )
    if(checker[1]==TRUE)
    {
      print(counterNew)
      counterNew = 100000000
    }
    counterNew = counterNew + 1
  }

  return(Q)
}
## finding Q decom at each iteration for eigen vector algorithm
calcQMatrix = function(cMat, colSize)
{
  cont = 1
  calcW = 2
  while(cont <= colSize)
  {
    if(cont == 1)
    {
      cU = cMat[,1]
      eMat = matrix(cU / sqrt(sum(cU^2)), ncol =1)
    }
    if(cont > 1)
    {
      cStep = cont - 1
      cA = cMat[,cont]
```

```
      cU = cA - eMat[,cStep]*(cA %*% eMat[,cStep])
      cStep = cStep-1
      while(cStep > 0)
      {
        cU = cU - eMat[,cStep]*(cA %*% eMat[,cStep])
        cStep = cStep - 1
      }
      newCol = matrix(cU / sqrt(sum(cU^2)), ncol = 1)
      eMat = cbind(eMat, newCol)

    }
    cont = cont + 1
  }
  return (eMat)

}
#eigen values from each vector
getEigenValueFromVec = function(covMat, eigenMat, cClassValue, colSize)
{
  if(colSize == 0)
  {
    return(cClassValue)
  }
  currentAX = covMat %*% eigenMat[,colSize]
  cEigenValue = (currentAX / eigenMat[,colSize])[1]
  cClassValue[,colSize] = cEigenValue
  print(cClassValue)
  getEigenValueFromVec(covMat, eigenMat, cClassValue, colSize-1)

}

mahalTwoPoints <- function(x1,x2, sigma)
{
  mahalD = sqrt(t(x1-x2) %*% solve(sigma) %*% (x1 - x2))
  return(mahalD)
}

getDiscrim <- function(x, mu, sigma, prior, d)
{
  dG = -0.5*(t(x-mu))%*% solve(sigma)%*%(x - mu)
       - (d/2)*log(2*pi)-0.5*log(det(sigma))
       + log(prior)

  return(dG)
```

```
}
generateSample <- function(eigMat, eigVal, d, n, mu)
{
  testX <- matrix(rnorm(d * n), n)

  eigValDiag1 = diag(sqrt(c(eigVal[,1],eigVal[,2])), d)

  testX = mu + eigMat %*% eigValDiag1 %*% t(testX)

  return(t(testX))

}

##set mus
mu1 = c(5,1)
mu2 = c(1,7)

##set sigma
sigma1 = sigma2 = matrix(c(3.1,1,1,2.6), nrow = 2)

##testing

mahalTwoPoints(c(5.1,1.1), c(4.9,.9), sigma1)

## graphing ?
####

##get eigen
eigMat1 = getEigOrtho(sigma1, 2)
eigMat2 = getEigOrtho(sigma2, 2)

##get lambda values ie - eigen values
eigOneVal = t(c(0,0))
eigOneVal = getEigenValueFromVec(sigma1,eigMat1,eigOneVal, 2)

eigTwoVal = t(c(0,0))
eigTwoVal = getEigenValueFromVec(sigma2,eigMat2,eigTwoVal, 2)

###generate sample data
dis1T = generateSample(eigMat1, eigOneVal, 2, 200, mu1)
dis2T = generateSample(eigMat2, eigTwoVal, 2, 800, mu2)


##bivariate normal distribution generation
```

```
plot(dis1T, xlim = range(-6,15), ylim = range(-4,12), col = "green")
points(dis2T, col = "red")



##3d plotting
count = dim(dis1T)[1]

dis1Z = NULL
dis2Z = NULL
getDiscrim(dis1T, mu1, sigma1, .2, 2)

while(count > 0)
{
  dis1Z = append(dis1Z, getDiscrim(dis1T[count,], mu1, sigma1, .2, 2))
  count = count - 1
}
count = dim(dis2T)[1]
while(count > 0)
{
  dis2Z = append(dis2Z, getDiscrim(dis2T[count,], mu2, sigma2, .2, 2))
  count = count - 1
}
dBound = dis1Z - dis2Z

dis1P = dnorm(seq(-5,13, length = 1099), mu1, sigma1)
plot(seq(-5,13, length = 1099),dis1P)

dis2P = dnorm(seq(-8,14, length = 1099), mu2, sigma2)
plot(seq(-7,13, length = 1099),dis2P)
points(seq(-7,13, length = 1099), dis1P, col = "red", pch =19)

## rest of project

##set new sigma
sigma1 = sigma2 = matrix(c(3.1,0,0,2.6), nrow = 2)
##get eigen
eigMat1 = getEigOrtho(sigma1, 2)
eigMat2 = getEigOrtho(sigma2, 2)
##get lambda values ie - eigen values
eigOneVal = t(c(0,0))
eigOneVal = getEigenValueFromVec(sigma1,eigMat1,eigOneVal, 2)

eigTwoVal = t(c(0,0))
```

```
eigTwoVal = getEigenValueFromVec(sigma2,eigMat2,eigTwoVal, 2)

eigMat1 = matrix(c(-1,0,0,-1), nrow = 2)
eigOneVal = t(c(3.1,2.6))
dis1T = generateSample(eigMat1, eigOneVal, 2, 200, mu1)
dis2T = generateSample(eigMat1, eigOneVal, 2, 800, mu2)

plot(dis1T, xlim = range(-2,10), ylim = range(-2,10), col = "red")
points(dis2T, col = "blue")

sigma1 =  matrix(c(2.1,1.5,1.5,3.8), nrow = 2)
sigma2 = matrix(c(3.1,.35,.35,2.6), nrow = 2)
##get eigen
eigMat1 = getEigOrtho(sigma1, 2)
eigMat2 = getEigOrtho(sigma2, 2)
##get lambda values ie - eigen values
eigOneVal = t(c(0,0))
eigOneVal = getEigenValueFromVec(sigma1,eigMat1,eigOneVal, 2)

eigTwoVal = t(c(0,0))
eigTwoVal = getEigenValueFromVec(sigma2,eigMat2,eigTwoVal, 2)

dis2T = generateSample(eigen(sigma2)$vectors, t(eigen(sigma2)$values), 2, 660, mu1)
dis1T = generateSample(eigen(sigma1)$vectors, t(eigen(sigma1)$values), 2, 330, mu1)

plot(dis1T, xlim = range(-2,10), ylim = range(-2,10), col = "red")
points(dis2T, col = "blue")
```