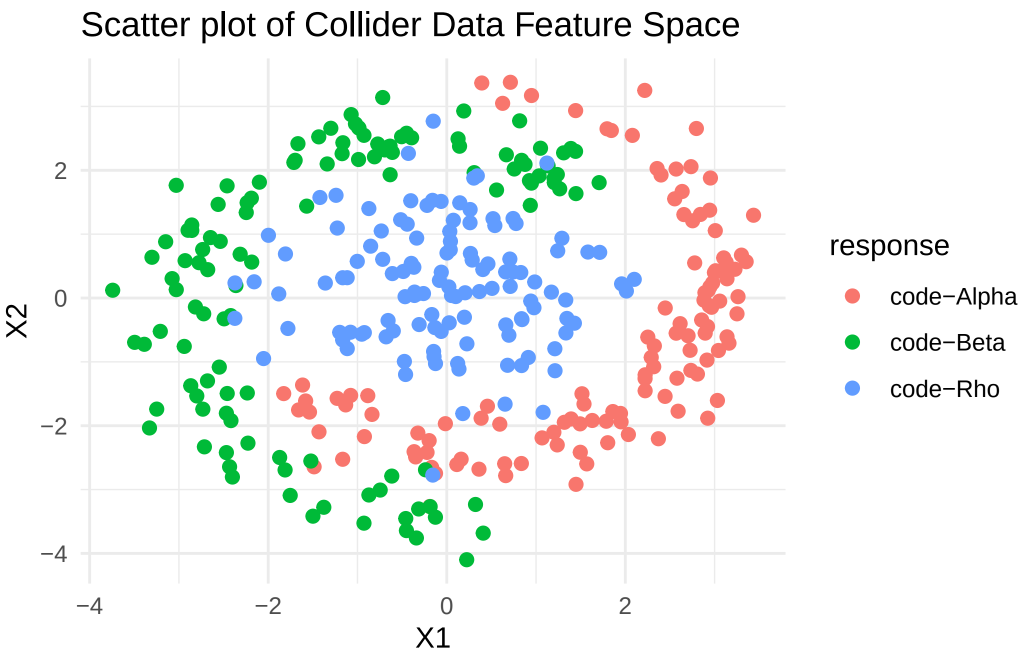
Assignment 2

```
[1] 360   6
```

```
          X1         X2 X3 Y1 Y2 Y3
1  3.2509506 -0.2478462  0  1  0  0
2  0.1130716 -2.6077716  1  1  0  0
3  2.0345077 -2.1380898  0  1  0  0
4 -1.6595216 -1.7533739  1  1  0  0
5  3.1474216  0.4139403  1  1  0  0
6 -0.1972298 -2.2335683  0  1  0  0
```

a)

```
dat$response <- apply(dat[, c("Y1", "Y2", "Y3")], 1, function (x){
  if (x[1] == 1) return("code-Alpha")
  if (x[2] == 1) return("code-Beta")
  if (x[3] == 1) return("code-Rho")
})
ggplot(dat, aes(x=X1, y=X2, color=response ))+geom_point(size=2)+ggtitle("Scatter plot of Co
```



The scatter plot shows how the particles are distributed in the X1-X2 plane. We can see that the process creates a complex and non-linear separation in the points plotted. Therefore, the use of a neural network is justified as they are known for their ability to learn non-linear decision boundaries which is presented in the scatter plot.

b)

```
softmax <- function(Z)
  {

  Z_shift <- Z - matrix(apply(Z, 2, max),           # subtract column-wise maxima
                        nrow = 3, ncol = ncol(Z),
                        byrow = TRUE)

  expZ    <- exp(Z_shift)                           # exponentiate
  denom   <- matrix(colSums(expZ),                  # column-wise sums
                    nrow = 3, ncol = ncol(Z),
                    byrow = TRUE)

  expZ / denom                                      # element-wise division
  }
```

c)

```r
calc_Ci <- function(y_true, y_pred) {
  # y_true is assumed to be a scalar (either 0 or 1)
  if (y_true == 1) {
    return(-log(y_pred))
  } else {
    return(0)
  }
}
```

Evaluating only the component corresponding to the actual class simplifies calculations and enhances numerical stability by avoiding evaluating terms that are known to be 0.

d)

```r
g <- function(Yhat, Y, eps = 1e-15) {
  # Yhat, Y : N × q matrices   (rows = obs, cols = classes)
  N <- nrow(Y)
  -sum( Y * log( pmax(Yhat, eps) ) ) / N
}
```

e)

number of parameters = $2p^2+2p+2pm+2m+m^2+mq+q$

f)

```r
af_forward <- function(X, Y, theta, m, nu)
{
  N <- nrow(X)
  p <- ncol(X)
  q <- ncol(Y)
```

```r
index <- 1:(2(p^2)) #W1 : p(p+p)
W1 <- matrix(theta[index], nrow=p)

index <- max(index)+1:(2*p) #b1 : (p+p)
b1 <- theta[index]

index <- max(index)+1:((2*p)*m) #W2 : (p+p)*m
W2 <- matrix(theta[index], nrow=2*p)

index <- max(index)+1:m #b2 : m
b2 <- theta[index]

index <- max(index)+1:(m*m) #W3 : (m*m)
W3 <- matrix(theta[index], nrow=m)

index <- max(index)+1:m #b3 : m
b3 <- theta[index]

index <- max(index)+1:(m*q) #W4 : (m*q)
W4 <- matrix(theta[index], nrow=m)

index <- max(index)+1:q #b4 : q
b4 <- theta[index]

#softmax function
softmax <- function(Z)
  {

Z_shift <- Z - matrix(apply(Z, 2, max),          # subtract column-wise maxima
                      nrow = 3, ncol = ncol(Z),
                      byrow = TRUE)

expZ    <- exp(Z_shift)                          # exponentiate
denom   <- matrix(colSums(expZ),                 # column-wise sums
                  nrow = 3, ncol = ncol(Z),
                  byrow = TRUE)

expZ / denom                                     # element-wise division
}

#forward propagation
H1 <- tanh( X  %*% W1 + matrix(b1, N, 2*p, TRUE) )       # aug-layer
```

```r
  H2 <- tanh( H1 %*% W2 + matrix(b2, N, m, TRUE) )      # 2nd hidden
  H3 <- tanh(H2 %*% W3 + matrix(b3, N, m, TRUE))
  Z <- H3 %*% W4 + matrix(b4, N, q, TRUE)          # logits
  probs <- softmax(Z)                                          # softmax

  #losses & objective
  loss <- g(probs, Y)                 # cross-entropy
  obj  <- loss + (nu / 2) * sum(theta^2)
  list(probs = probs, loss = loss, obj = obj)

}
```