Universidade Federal de Minas Gerais
Aluno: Giovanni Martins de Sá Júnior
Matrícula: 2017001850

Exercício 2: Redes Neurais Artificiais:

1. Problema Não Linearmente Separável

O primeiro exercício da lista se trata de realizar a classificação de um problema não linear que pode ser visto pela seguinte imagem:

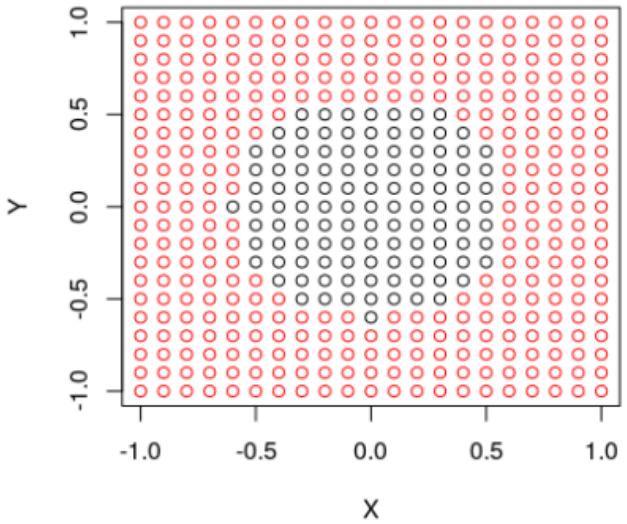


Figura 1. Resultado Esperado

Para isso, foi disponibilizado uma rotina que pudesse gerar os dados para a resolução do problema:

```
x = seq(-1,1,by = 0.1)
y = seq(-1,1,by = 0.1)
create_grid <- expand.grid(x,y)

circle <- function(x,y) {
  return(sqrt(x^2+y^2))
}

raio = 0.6

classe = 1*(circle(create_grid$Var1,create_grid$Var2)>raio)
```

Figura 2. Código disponibilizado para a geração dos dados

Com isso, se analisarmos a rotina mencionada acima, percebemos que ela irá gerar pontos distribuídos de maneira uniforme entre -1 e 1, formando um plano uniforme. Assim para diferenciarmos os pontos que estão dentro e fora da função do círculo, foram implementados dois loops for que percorre o vetor de elementos do círculo por meio de **k**, em que a cada execução do loop, os valores de **i** e **j** dos loops armazenam as posições **x** e **y**, e quando o elemento do vetor for igual a 1, é esperado que o ponto seja assinalado pela cor vermelha.

Assim, para ser possível diferenciar as duas regiões, foi aplicada uma função sigmoidal, que tem a função de nos retornar dois resultados: zero para quando x assume valores negativos, e um para valores positivos:

$$f(x) = (1 + e^{-x})^{-1}$$

Para que a solução atenda às condições de implementação, é fundamental que a função se aproxime do valor do raio quando x assumir o valor do raio. Para isso, foi implementada a relação de w1, um dos pesos a serem calculados, seguindo as condições mencionadas acima.

$$w_1(1 + e^{-0.6})^{-1} = 0.6 \Leftrightarrow w_1 = 0.6 + 0.6e^{-0.6} \Leftrightarrow w_1 = 0.92$$

Consequentemente, a função final será $f(x) = 0.92\,(1 + e^{-x})^{-1}$. A seguir, apresentamos a implementação da solução:

```
<<>>=
library('plotly')
rm(list = ls())

x = seq(-1, 1, by = 0.1)
y = seq(-1, 1, by = 0.1)
create_grid <- expand.grid(x, y)

circle <- function(x, y) {
  y = sqrt(x^2 + y^2)
  return((1 / (1 + exp(-y))))
}

raio = 0.6
classe = 1 * (circle(create_grid$Var1, create_grid$Var2) > raio)

plot(create_grid, xlab = 'x', ylab = 'y', xlim = c(min(x), max(x)), ylim = c(min(y), max(y)))

k = 1

for (i in x) {
  for (j in y) {
    if(classe[k] == 1){
      points(i, j, col = 'red')
    }
    k = k + 1
  }
}

classe_plot <- matrix(0, 21, 21)

for (lin in 1:21) {
  for (col in 1:21) {
    classe_plot[lin, col] <- classe[(lin - 1)*21 + col]
  }
}
@
```

Figura 3. Implementação do Código

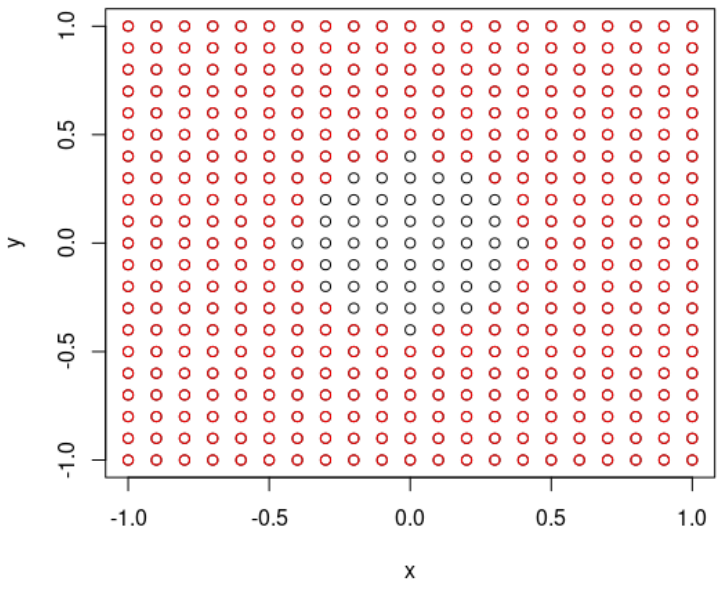


Figura 4. Resultado Obtido

2. Overfitting e Underfitting

2.1. Qual dos 3 modelos construídos (preto, vermelho e azul) parece ser a melhor aproximação da função geradora dos dados, sabendo-se que existe um ruído na amostragem?

A melhor aproximação aparenta ser a azul, uma vez que seu formato aparenta ser a função sem ruído.

2.2. Qual dos modelos apresenta menor erro de treinamento?

O preto, uma vez que a função se aproxima muito dos dados de entrada.

2.3. Qual dos modelos deve ter melhor desempenho para dados novos?

O azul, pois o melhor desempenho está associado com a melhor aproximação da função original.

2.4. Discuta sobre os efeitos do dimensionamento do modelo e ajuste aos dados (erro de treinamento) sobre a aproximação de funções com dados amostrados, como o da figura. Baixo erro de treinamento implica em um melhor desempenho no longo prazo? Discuta.

Para o vermelho, é notável que a função gerada não chega perto do objetivo, uma vez que ele foi subdimensionado. Com isso, ele não tem a capacidade de se aproximar dos dados de uma maneira aceitável.

Para o preto, o modelo foi superdimensionado, gerando uma saída que se aproxima demais dos dados de entrada, gerando overfitting.

Assim, o modelo ideal seria o de cor azul, que tem uma complexidade adequada para gerar uma função próxima da desejada, porém sem se aproximar demais dos dados de entrada a ponto dos ruídos influenciarem no sinal.

3. Aproximação Polinomial

Para a resolução deste terceiro exercício, foi apresentada uma função fg(x) em que é esperado obter uma aproximação polinomial por meio das amostras, com o seu grau polinomial variando de 1 a 8. A função fg(x) é denotada por:

$$f_g(x) = 0.5x^2 + 3x + 10$$

Nesse sentido, a parte inicial da aproximação polinomial implementada na linguagem R, pode ser vista logo abaixo:

```
<<>>=
library('corpcor')
library('pracma')

rm(list = ls())

xr <-runif(10, min = -15, max = 10)
fgx <- 0.5*xr^2 + 3*xr + 10
yr <- fgx + rnorm(10, 0, 4)

xt <- linspace(-15, 10, 20)
yt <- 0.5*xt^2 + 3*xt + 10
@
```

Figura 5. Primeira metade da Implementação

Em sequência, a segunda metade da implementação foi gradualmente reajustada, aumentando em cada caso, o grau do polinômio, como foi especificado no exercício:

P = 1

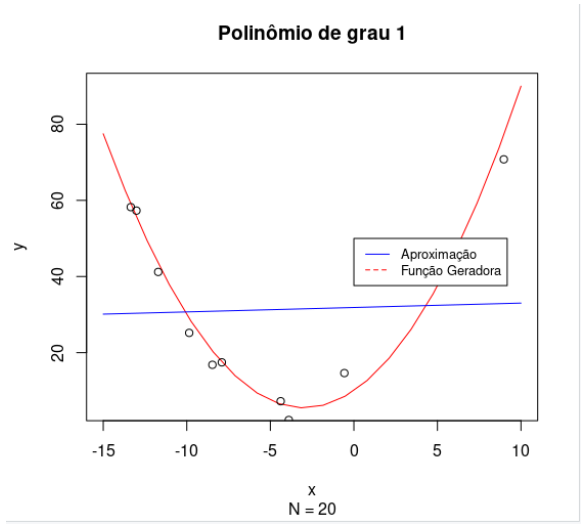
```
<<>>=
# P = 1

H1 <- cbind(xr^1, 1)
w1 <- pseudoinverse(H1) %*% yr

Ht <- cbind(xt^1, 1)
yt1 <- Ht %*% w1

plot(xt, yt, type = 'l', col = 'red', main = 'Polinômio de grau 1', sub = 'N = 20', xlab = 'x', ylab = 'y')

points (xr, yr)
lines (xt, yt1, col = 'blue')
legend(0, 50, legend = c('Aproximação', 'Função Geradora'), col = c('blue', 'red'), lty = 1:2, cex = 0.8)
@
```



P = 2

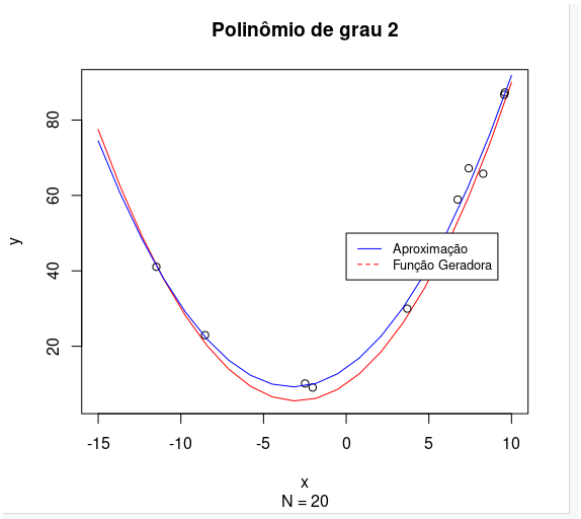
```
<<>>=
# P = 2

H2 <- cbind(xr^2, xr^1, 1)
w2 <- pseudoinverse(H2) %*% yr

Ht <- cbind(xt^2, xt^1, 1)
yt2 <- Ht %*% w2

plot(xt, yt, type = 'l', col = 'red', main = 'Polinômio de grau 2', sub = 'N = 20', xlab = 'x', ylab = 'y')

points (xr, yr)
lines (xt, yt2, col = 'blue')
legend(0, 50, legend = c('Aproximação', 'Função Geradora'), col = c('blue', 'red'), lty = 1:2, cex = 0.8)
@
```



P = 3

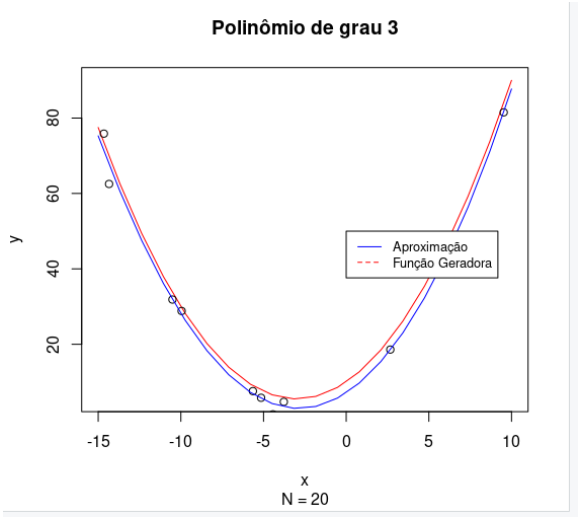
```
<<>>=
# P = 3

H3 <- cbind(xr^3, xr^2, xr^1, 1)
w3 <- pseudoinverse(H3) %*% yr

Ht <- cbind(xt^3, xt^2, xt^1, 1)
yt3 <- Ht %*% w3

plot(xt, yt, type = 'l', col = 'red', main = 'Polinômio de grau 3', sub = 'N = 20', xlab = 'x', ylab = 'y')

points (xr, yr)
lines (xt, yt3, col = 'blue')
legend(0, 50, legend = c('Aproximação', 'Função Geradora'), col = c('blue', 'red'), lty = 1:2, cex = 0.8)
@
```



P = 4

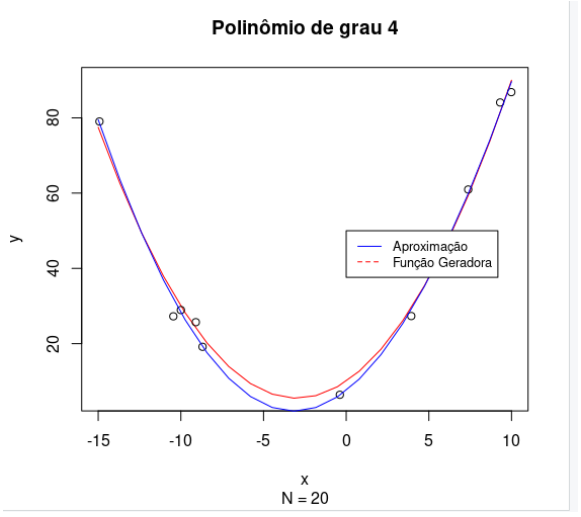
```
<<>>=
# P = 4

H4 <- cbind(xr^4, xr^3, xr^2, xr^1, 1)
w4 <- pseudoinverse(H4) %*% yr

Ht <- cbind(xt^4, xt^3, xt^2, xt^1, 1)
yt4 <- Ht %*% w4

plot(xt, yt, type = 'l', col = 'red', main = 'Polinômio de grau 4', sub = 'N = 20', xlab = 'x', ylab = 'y')

points (xr, yr)
lines (xt, yt4, col = 'blue')
legend(0, 50, legend = c('Aproximação', 'Função Geradora'), col = c('blue', 'red'), lty = 1:2, cex = 0.8)
@
```



P = 5

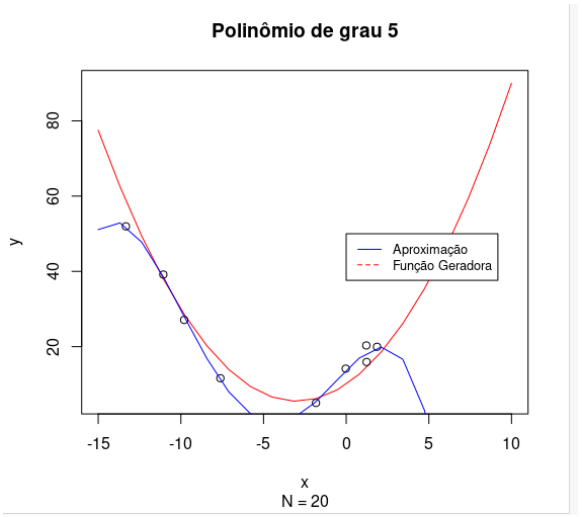
```
<<>>=
# P = 5

H5 <- cbind(xr^5, xr^4, xr^3, xr^2, xr^1, 1)
w5 <- pseudoinverse(H5) %*% yr

Ht <- cbind(xt^5, xt^4, xt^3, xt^2, xt^1, 1)
yt5 <- Ht %*% w5

plot(xt, yt, type = 'l', col = 'red', main = 'Polinômio de grau 5', sub = 'N = 20', xlab = 'x', ylab = 'y')

points (xr, yr)
lines (xt, yt5, col = 'blue')
legend(0, 50, legend = c('Aproximação', 'Função Geradora'), col = c('blue', 'red'), lty = 1:2, cex = 0.8)
@
```



P = 6

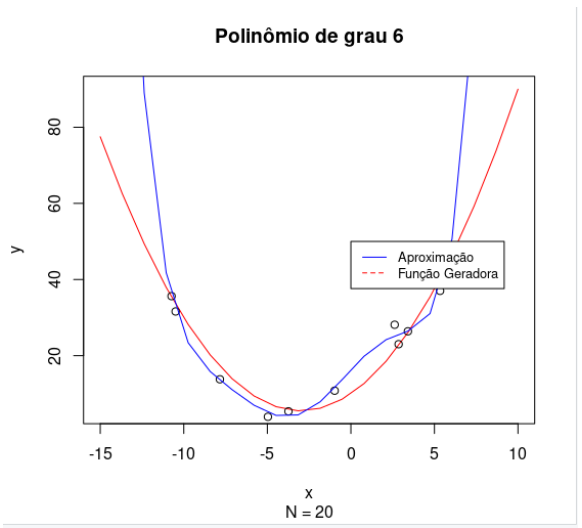
```
<<>>=
# P = 6

H6 <- cbind(xr^6, xr^5, xr^4, xr^3, xr^2, xr^1, 1)
w6 <- pseudoinverse(H6) %*% yr

Ht <- cbind(xt^6, xt^5, xt^4, xt^3, xt^2, xt^1, 1)
yt6 <- Ht %*% w6

plot(xt, yt, type = 'l', col = 'red', main = 'Polinômio de grau 6', sub = 'N = 20', xlab = 'x', ylab = 'y')

points (xr, yr)
lines (xt, yt6, col = 'blue')
legend(0, 50, legend = c('Aproximação', 'Função Geradora'), col = c('blue', 'red'), lty = 1:2, cex = 0.8)
@
```



P = 7

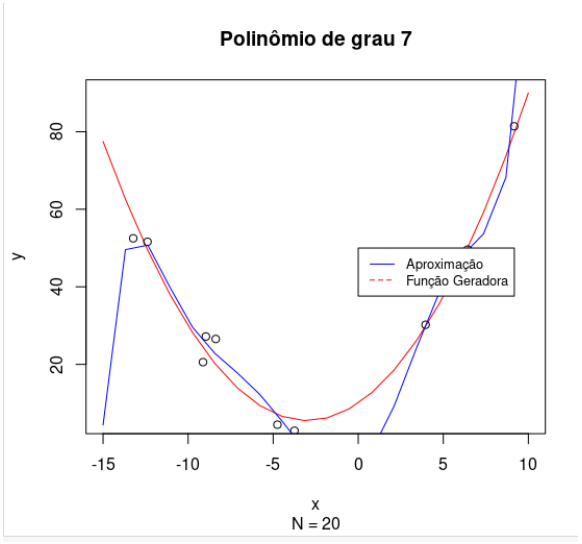
```
<<>>=
# P = 7

H7 <- cbind(xr^7, xr^6, xr^5, xr^4, xr^3, xr^2, xr^1, 1)
w7 <- pseudoinverse(H7) %*% yr

Ht <- cbind(xt^7, xt^6, xt^5, xt^4, xt^3, xt^2, xt^1, 1)
yt7 <- Ht %*% w7

plot(xt, yt, type = 'l', col = 'red', main = 'Polinômio de grau 7', sub = 'N = 20', xlab = 'x', ylab = 'y')

points (xr, yr)
lines (xt, yt7, col = 'blue')
legend(0, 50, legend = c('Aproximação', 'Função Geradora'), col = c('blue', 'red'), lty = 1:2, cex = 0.8)
@
```



P = 8

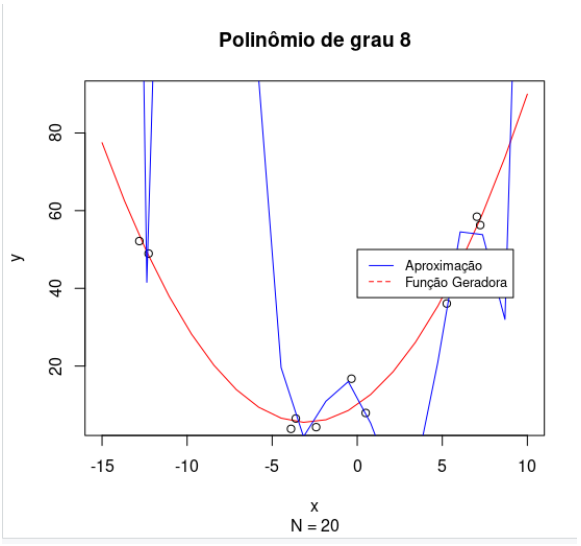
```
<<>>=
# P = 8

H8 <- cbind(xr^8, xr^7, xr^6, xr^5, xr^4, xr^3, xr^2, xr^1, 1)
w8 <- pseudoinverse(H8) %*% yr

Ht <- cbind(xt^8, xt^7, xt^6, xt^5, xt^4, xt^3, xt^2, xt^1, 1)
yt8 <- Ht %*% w8

plot(xt, yt, type = 'l', col = 'red', main = 'Polinômio de grau 8', sub = 'N = 20', xlab = 'x', ylab = 'y')

points (xr, yr)
lines (xt, yt8, col = 'blue')
legend(0, 50, legend = c('Aproximação', 'Função Geradora'), col = c('blue', 'red'), lty = 1:2, cex = 0.8)
@
```



A partir de todas essas variações, foi possível observar que após cada aumento de grau em relação ao polinômio da função geradora, o overfitting ficava cada vez mais nítido, sobretudo a partir do grau P = 5.

Em um caso específico, sendo este o primeiro exemplo (P = 1), ficou explícito um exemplo de underfitting, em que o grau da função de aproximação não é suficiente para a resolução do problema.

Por fim, era de se esperar que o grau P = 2 da função aproximadora fosse o que melhor satisfizesse o problema proposto uma vez que apresentava um grau análogo à função geradora fg(x), apesar dos modelos P = 3 e P = 4 também apresentarem um resultado satisfatório.