UNIVERSIDADE FEDERAL
DE MINAS GERAIS

ATIVIDADE AVALIATIVA
TREINAMENTO RBF 2 - BREAST CANCER

ALUNO: FELIPPE VELOSO MARINHO
MATRÍCULA: 2021072260
DISCIPLINA: REDES NEURAIS ARTIFICIAIS

Nesta atividade o objetivo é a aplicação de uma rede RBF ao problema do Câncer de mama (Breast Cancer). Esta base de dados pode ser carregada do pacote *mlbench*. Esta base de dados possui 9 variáveis de entrada, uma variável de saída com a classificação das 699 amostras em maligno e benigno.

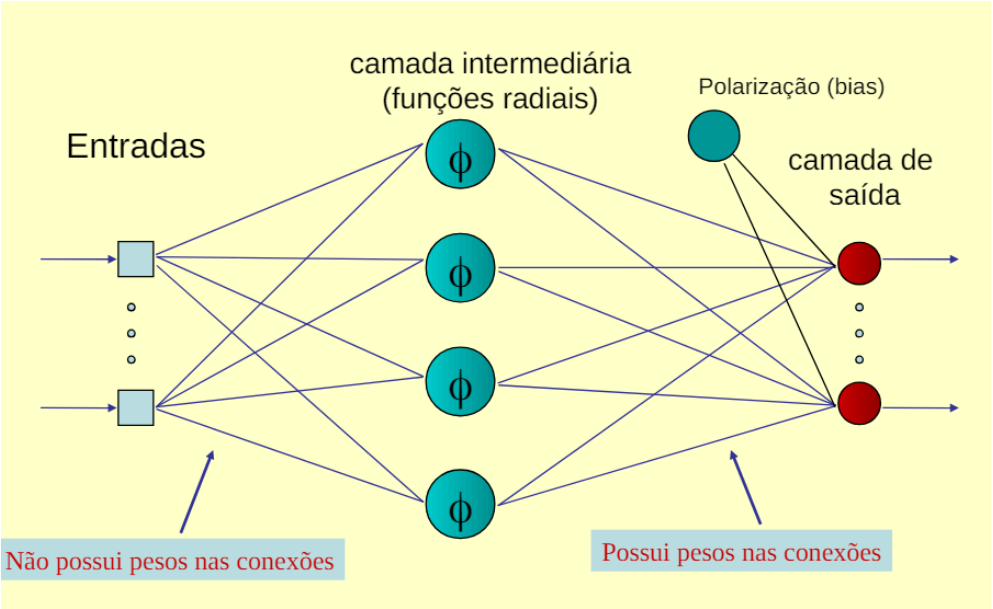
O treinamento da RBF irá consistir em separar as classes e avaliar o desempenho do mesmo.

Vale ressaltar, assim como no último problema, as redes com função de base radial apresentam 3 diferenças das redes MLP:

1) Elas sempre apresentam uma única camada intermediária;

2) Os neurônios de saída são sempre lineares;

3) Os neurônios da camada intermediária têm uma função de base radial como função de ativação, ao invés de uma função sigmoidal ou outras.



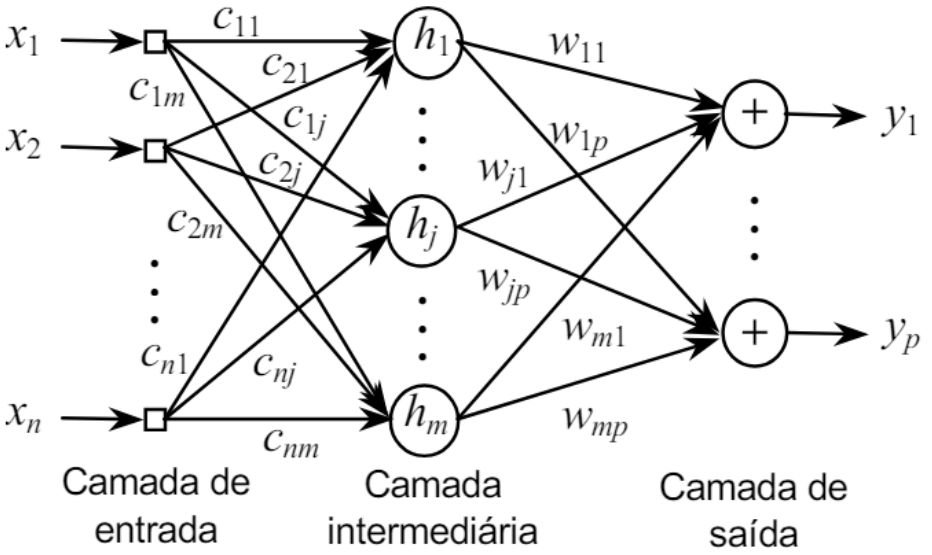


Figura 3 – Rede RBF com múltiplas saídas.

Carregamento e tratamento dos dados

Primeiramente , assim como no enunciado do exercício, foram carregados os dados do problema. Removemos as linhas com valores faltantes (NA) e a coluna ID que não é útil para o treinamento do modelo. Para a realização de operações matemáticas ao longo do código, convertemos as colunas necessárias em dados numéricos.

```
# Converter variáveis de fatores para numéricas
data2[, 1:9] <- lapply(data2[, 1:9], as.numeric)
```

Foi criado a rotulação das amostras das classes com o valor de 0 para maligno e 1 para benigno. Assim como requisitado no tópico 2 da atividade.

```
# Rotular as amostras das Classes com o valor de 0 (maligno) e 1 (benigno)
data2$Class <- ifelse(data2$Class == "malignant", 0, 1)
```

Terminando essa parte, assim como no primeiro exemplo feito em redes RBF, utilizamos a função gaussiana novamente como função radial na camada intermediária.

Ajuste e treinamento das redes RBF

```
# Função para ajustar e treinar a RBF Network
fit_rbf_network <- function(X_train, y_train, n_clusters, sigma) {
  # Normalizar os dados
  scaler <- preProcess(X_train, method = c("center", "scale"))
  X_train_scaled <- predict(scaler, X_train)
```

Para a normalização dos dados utilizamos o preProcess da biblioteca “caret” usada justamente para criar pré-processamento para centrar os dados. O método “predict” é utilizado para normatizar considerando o conjunto de dados da entrada. Esse processo funciona da seguinte forma:

Supondo que o conjunto de dados originais é:

```
X_train <- data.frame(
        V1 = c(10, 20, 30),
        V2 = c(40, 50, 60)
)
scaler <- preProcess(X_train, method = c("center", "scale"))
X_train_scaled <- predict(scaler, X_train)
```

Após a criação do objeto “preProcess” e aplicarmos a normalização com o “predict” o que temos é basicamente.

- Para V1, a média é 202020 e o desvio padrão é 10.
- Para V2, a média é 505050 e o desvio padrão é 10.

Portanto os dados normalizados são:

```
X_train_scaled
# V1 V2
# -1.0 -1.0
# 0.0 0.0
# 1.0 1.0
```

Aplicar essas transformações aos dados (ou novos dados) usando predict assegura que todas as características têm média 0 e desvio padrão 1, facilitando o treinamento e melhorando o desempenho do código.

```
# Encontrar os centros dos clusters
  kmeans_model <- kmeans(X_train_scaled, centers = n_clusters, nstart = 20)
  centers <- kmeans_model$centers
```

Utilizando o kmeans da biblioteca “cluster”, especificamos o número de clusters ou centros da RBF. O kmeans é rodado 20 vezes com os dados normalizados e os centros são armazenados na variável “centers”.

Momento revisão: “O k-means é um algoritmo de aprendizado não supervisionado usado para particionar um conjunto de dados em kkk clusters. Cada cluster é representado pelo seu centroide, que é a média dos pontos do cluster.”

```
  # Transformar os dados usando a RBF
  transformed_X_train <- t(apply(X_train_scaled, 1, function(row) gauss(row, centers, sigma)))
```

Por fim, com os centros dos clusters encontrados, transformamos os dados de entrada onde cada ponto de dado é transformado usando uma função gaussiana baseada na distância aos centros. Para isso, aplicamos a função gaussiana em cada linha dos dados normalizados passando.

```
  # Adicionar nomes de colunas ao data frame transformado
  colnames(transformed_X_train) <- paste0("C", 1:n_clusters) # Os dados são convertidos para data frame e adicionamos rótulos para cada coluna, representando cada centro (neurônio na camada escondida)

  # Treinar o modelo de regressão logística
  logistic_model <- multinom(y_train ~ ., data = as.data.frame(transformed_X_train))

  list(scaler = scaler, centers = centers, logistic_model = logistic_model, sigma = sigma)
}
```

No loop de simulações separo todos os conjuntos para treino e teste, sendo 90% da amostras para o conjunto de treinamento e 10% para o conjunto de teste. Depois disso, podemos treinar o usando o modelo criado. Dentro da função “*predict_rbf_network”*, fazemos a normalização, transformação com RBFs, e finalmente usamos o modelo de regressão logística para fazer a previsão:

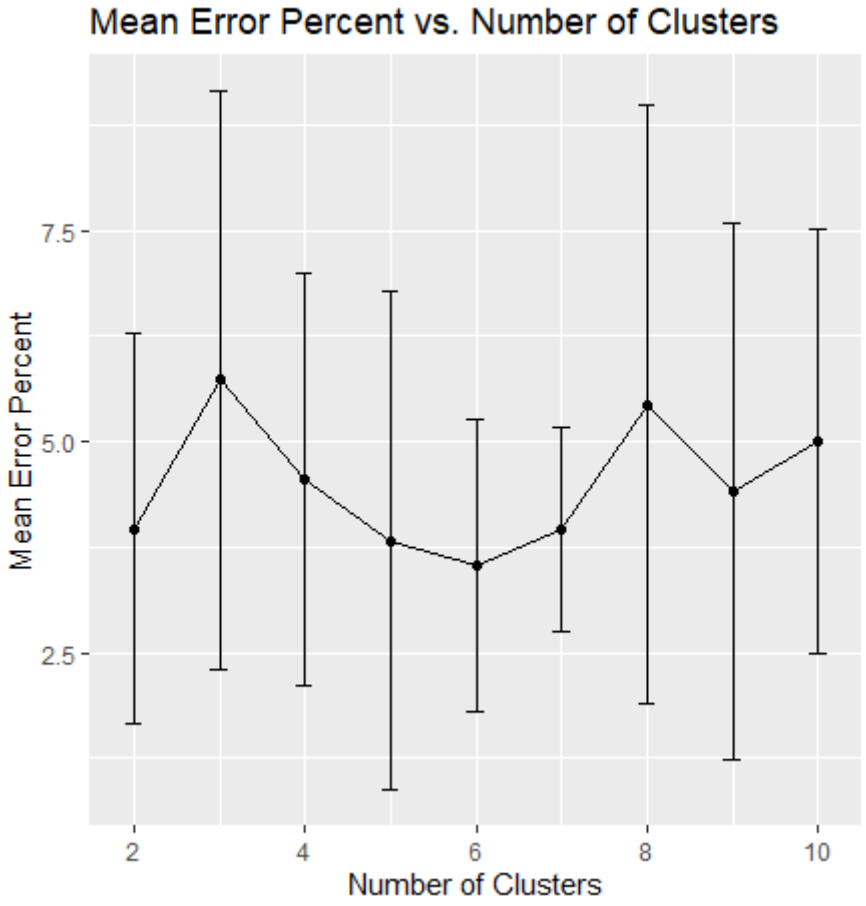
```
# Função para prever usando a RBF Network
```

```
predict_rbf_network <- function(model, X_test) {
  X_test_scaled <- predict(model$scaler, X_test)
  transformed_X_test <- t(apply(X_test_scaled, 1, function(row) gauss(row, model$centers,
model$sigma)))

  # Adicionar nomes de colunas ao data frame transformado
  colnames(transformed_X_test) <- paste0("C", 1:ncol(transformed_X_test))

  predict(model$logistic_model, as.data.frame(transformed_X_test))
}
```

No final do loop para realizar as 10 simulações, os dados de maior percentual de erros de cada iteração são salvos para serem exibidos em um gráfico de comparação de número de clusters e maior erro percentual.



É visível que o cluster que revelou o menor erro percentual médio foi o 5, porém ele possui uma alta porcentagem de erro. Sendo assim, o melhor cluster nessa análise foi o 6, devido a melhor relação entre porcentagem de erro (refletida pelo tamanho das linhas do gráfico).

Sendo a relação final:

	Clusters	MeanAccuracy	StdDevAccuracy	MeanErrorPercent	StdDevErrorPercent
5	6	0.9647059	0.01726159	3.529412	1.726159
4	5	0.9617647	0.02957471	3.823529	2.957471

```
1         2    0.9602941     0.02304443         3.970588           2.304443
6         7    0.9602941     0.01210695         3.970588           1.210695
8         9    0.9558824     0.03176834         4.411765           3.176834
3         4    0.9544118     0.02446074         4.558824           2.446074
9        10    0.9500000     0.02518673         5.000000           2.518673
7         8    0.9455882     0.03538251         5.441176           3.538251
2         3    0.9426471     0.03427869         5.735294           3.427869
```

Observações: “Para essa análise o foco foi no erro percentual devido aos requisitos do exercício. Porém, foram exibidos as acurácias, salvas anteriormente ao final do loop também. A ideia desse relatório não foi somente a interpretação dos resultados mas também um resumo para que possa ser revisitado posteriormente.
Optei por exibir somente um gráfico com a comparação de acordo com o número de clusters pois já bastava para realizar os requisitos do exercício.“

Código completo:

```
# Carregar bibliotecas necessárias
library(mlbench)
library(caret)
library(cluster)
library(nnet)
library(ggplot2)

# Carregar e limpar os dados
data("BreastCancer")
data2 <- BreastCancer
data2 <- data2[complete.cases(data2),]

# Remover coluna de ID
data2 <- data2[, -1]

# Converter variáveis de fatores para numéricas
data2[, 1:9] <- lapply(data2[, 1:9], as.numeric)

# Rotular as amostras das Classes com o valor de 0 (malígno) e 1 (benígno)
data2$Class <- ifelse(data2$Class == "malignant", 0, 1)

# Função Gaussiana
gauss <- function(x, centers, sigma) {
  apply(centers, 1, function(center) {
    exp(-0.5 * sum((x - center)^2) / (sigma^2))
  })
}
```

```
# Função para ajustar e treinar a RBF Network
fit_rbf_network <- function(X_train, y_train, n_clusters, sigma) {
  # Normalizar os dados
  scaler <- preProcess(X_train, method = c("center", "scale"))
  X_train_scaled <- predict(scaler, X_train)

  # Encontrar os centros dos clusters
  kmeans_model <- kmeans(X_train_scaled, centers = n_clusters, nstart = 20)
  centers <- kmeans_model$centers

  # Transformar os dados usando a RBF
  transformed_X_train <- t(apply(X_train_scaled, 1, function(row) gauss(row, centers,
sigma)))

  # Adicionar nomes de colunas ao data frame transformado
  colnames(transformed_X_train) <- paste0("C", 1:n_clusters)

  # Treinar o modelo de regressão logística
  logistic_model <- multinom(y_train ~ ., data = as.data.frame(transformed_X_train))

  list(scaler = scaler, centers = centers, logistic_model = logistic_model, sigma =
sigma)
}

# Função para prever usando a RBF Network
predict_rbf_network <- function(model, X_test) {
  X_test_scaled <- predict(model$scaler, X_test)
  transformed_X_test <- t(apply(X_test_scaled, 1, function(row) gauss(row,
model$centers, model$sigma)))

  # Adicionar nomes de colunas ao data frame transformado
  colnames(transformed_X_test) <- paste0("C", 1:ncol(transformed_X_test))

  predict(model$logistic_model, as.data.frame(transformed_X_test))
}

# Variáveis
n_simulations <- 10
sigma <- 1.0
results <- data.frame()

# Loop de simulações
for (n_clusters in 2:10) {
```

```
  accuracies <- c()
  error_percents <- c()

  for (i in 1:n_simulations) {
    # Separar os dados em conjuntos de treino e teste
    train_index <- createDataPartition(data2$Class, p = 0.9, list = FALSE)
    X_train <- data2[train_index, 1:9]
    y_train <- data2[train_index, 10]
    X_test <- data2[-train_index, 1:9]
    y_test <- data2[-train_index, 10]

    # Treinar a RBF Network
    model <- fit_rbf_network(X_train, y_train, n_clusters, sigma)

    # Prever no conjunto de teste
    y_pred <- predict_rbf_network(model, X_test)

    # Calcular a acurácia
    accuracy <- mean(y_pred == y_test)
    accuracies <- c(accuracies, accuracy)

    # Calcular o erro percentual
    error_percent <- mean(y_pred != y_test) * 100
    error_percents <- c(error_percents, error_percent)
  }

  # Armazenar resultados
  mean_accuracy <- mean(accuracies)
  std_accuracy <- sd(accuracies)
  mean_error_percent <- mean(error_percents)
  std_error_percent <- sd(error_percents)

  results <- rbind(results, data.frame(
    Clusters = n_clusters,
    MeanAccuracy = mean_accuracy,
    StdDevAccuracy = std_accuracy,
    MeanErrorPercent = mean_error_percent,
    StdDevErrorPercent = std_error_percent
  ))
}

# Melhor número de clusters
best_clusters <- results[which.max(results$MeanAccuracy), "Clusters"]
```

```
# Exibir resultados
print(results)

# Melhor número de clusters
best_clusters <- results[which.min(results$MeanError), "Clusters"]

# Exibir resultados ordenados por erro percentual
results <- results[order(results$MeanError), ]
print(results)

# Plotar acurácia
ggplot(results, aes(x = Clusters, y = MeanAccuracy)) +
  geom_point() +
  geom_line() +
  geom_errorbar(aes(ymin = MeanAccuracy - StdDevAccuracy, ymax =
MeanAccuracy + StdDevAccuracy), width = 0.2) +
  labs(title = "Mean Accuracy vs. Number of Clusters", x = "Number of Clusters", y =
"Mean Accuracy")

# Plotar erro percentual
ggplot(results, aes(x = Clusters, y = MeanErrorPercent)) +
  geom_point() +
  geom_line() +
  geom_errorbar(aes(ymin = MeanErrorPercent - StdDevErrorPercent, ymax =
MeanErrorPercent + StdDevErrorPercent), width = 0.2) +
  labs(title = "Mean Error Percent vs. Number of Clusters", x = "Number of Clusters",
y = "Mean Error Percent")

# Melhor modelo
print(paste("Best number of clusters: ", best_clusters))
```