Exercício 8.
Nome: Lucas Kou Kinoshita
RM: 2019021557
data: 26/04/2025

Implementação:

De forma semelhante às atividades anteriores, foi implementada a rotina com um loop de n repetições para avaliar a acurácia média de teste e treino e suas variâncias. Desta vez, para o treinamento do RBF.

O RBF foi, inicialmente, implementado de duas formas distintas, a primeira consiste em utilizar o k-médias como meio de posicionamento dos clusters (sobre os quais a rede RBF é aplicada), já a segunda busca aleatorizar o centro dos clusters e seu raio (coeficiente de espalhamento) através da escolha de um par de pontos aleatórios do conjunto de dados de treinamento (assim como descrito no enunciado).

A primeira versão do RBF implementado já foi comentada anteriormente (no exercício 7) e permanece exatamente igual, portanto não será abordada neste relatório. Já a segunda implementação pode ser observada abaixo, note que o posicionamento dos centros pertencentes aos *clusters* agora é determinado pela média entre as coordenadas de dois pontos aleatórios e o coeficiente (*r*) de espalhamento é equivalente a metade da distância entre estes pontos. A função *YRBF()* também foi levemente alterada para acomodar o novo método.

```r
YRBF_aleatorio <- function(xin, modRBF) {
  radialnvar <- function(x, m, sigma) exp(-sum((x-m)^2) / (2*sig

  N <- dim(xin)[1]

  centers <- modRBF$centers
  radii <- modRBF$radii
  W <- modRBF$W
  p <- nrow(centers)

  xin <- as.matrix(xin)
  H <- matrix(nrow = N, ncol = p)

  for(j in 1:N) {
    for(i in 1:p) {
      H[j, i] <- radialnvar(xin[j, ], centers[i, ], radii[i])
    }
  }

  # Add bias term
  Haug <- cbind(1, H)

  # Calculate predictions
  Yhat <- Haug %*% W

  return(Yhat)
}
```

```r
RBF_aleatorio <- function(xin, yin, p) {
  radialnvar <- function(x, m, sigma) exp(-sum((x-m)^2) / (2*sigma^2))

  N <- dim(xin)[1]
  n <- dim(xin)[2]

  xin <- as.matrix(xin)
  yin <- as.matrix(yin)

  centers <- matrix(0, nrow = p, ncol = n)
  radii <- numeric(p)

  for(i in 1:p) {
    idx_pair <- sample(N, 2)
    point1 <- xin[idx_pair[1], ]
    point2 <- xin[idx_pair[2], ]

    centers[i, ] <- (point1 + point2) / 2

    radii[i] <- sqrt(sum((point1 - point2)^2))

    if(radii[i] < 1e-5) {
      radii[i] <- mean(radii[max(1, i-1)])
      if(is.na(radii[i])) radii[i] <- 1.0
    }
  }

  H <- matrix(nrow = N, ncol = p)
  for(j in 1:N) {
    for(i in 1:p) {
      H[j, i] <- radialnvar(xin[j, ], centers[i, ], radii[i])
    }
  }

  Haug <- cbind(1, H)
  W <- pseudoinverse(Haug) %*% yin

  return(list(centers = centers, radii = radii, W = W))
}
```

Por fim, de forma semelhante aos exercícios anteriores, o modelo implementado foi testado sobre os conjuntos de dados *"Breast Cancer"* e *"Statlog (Heart)"*, para um total de 15 repetições, com valores variados de número de centros (*k*)

```r
# Load BreastCancer
data("BreastCancer")
df <- BreastCancer
df <- df[, -1]  # drop 'Id'
df <- na.omit(df)
# Convert factor columns to numeric (except 'Class')
df[, 1:9] <- lapply(df[, 1:9], function(x) as.numeric(as.character(x)))
df$class <- ifelse(df$class == "malignant", 1, 0)
# Matrix version
X <- as.matrix(df[, 1:9])
y <- as.matrix(df$class)
N <- nrow(X)

# Number of centers
p <- 150
ntrain <- floor(nrow(df) * 0.7)
reps <- 15
```

```r
# Load Statlog (Heart)
# tratamento
df <- read.table("http://archive.ics.uci.edu/ml/machine-learning-databases/statlog/heart/heart
colnames(df) <- c(
  "age", "sex", "chest_pain", "chol", "fbs", "rest_ecg",
  "max_hr", "ex_angina", "oldpeak", "slope", "ca", "thal", "target"
)
df <- na.omit(df)
df$target <- ifelse(df$target == 1, 1, 0)

df[, 1:14] <- lapply(df[, 1:14], function(x) as.numeric(as.character(x)))

X <- as.matrix(df[, 1:14])
y <- as.matrix(df$target)
N <- nrow(X)
```

Open Link (Ctrl+Click)

```r
for (r in 1:reps) {
  # Shuffle indices
  idx <- sample(N)

  # Train/Test split
  train_idx <- idx[1:ntrain]
  test_idx <- idx[(ntrain + 1):N]

  xin <- X[train_idx, ]
  yd <- y[train_idx]
  xinteste <- X[test_idx, ]
  yteste <- y[test_idx]

  # Train both models
  # 1. Original K-means based RBF
  retlist_kmeans <- RBF(xin, yd, p = p, r = 1.0)

  # 2. New random centers RBF
  retlist_random <- RBF_aleatorio(xin, yd, p = p)

  # Predictions for k-means RBF
  yhat_train_kmeans <- YRBF(xin, retlist_kmeans)
  yhat_test_kmeans <- YRBF(xinteste, retlist_kmeans)

  # Predictions for random RBF
  yhat_train_random <- YRBF_aleatorio(xin, retlist_random)
  yhat_test_random <- YRBF_aleatorio(xinteste, retlist_random)

  # Binarize predictions
  yhat_train_class_kmeans <- ifelse(yhat_train_kmeans >= 0.5, 1, 0)
  yhat_test_class_kmeans <- ifelse(yhat_test_kmeans >= 0.5, 1, 0)
  yhat_train_class_random <- ifelse(yhat_train_random >= 0.5, 1, 0)
  yhat_test_class_random <- ifelse(yhat_test_random >= 0.5, 1, 0)

  # Calculate accuracies
  acc_treino_kmeans[r] <- mean(yhat_train_class_kmeans == yd)
  acc_teste_kmeans[r] <- mean(yhat_test_class_kmeans == yteste)
  acc_treino_random[r] <- mean(yhat_train_class_random == yd)
  acc_teste_random[r] <- mean(yhat_test_class_random == yteste)
}
```

```r
# Print results
cat("\n====== RBF com k-médias ======\n")
cat("Acurácia média (treino):", round(mean(acc_treino_kmeans), 4),
    "±", round(sd(acc_treino_kmeans), 4), "\n")
cat("Acurácia média (teste):", round(mean(acc_teste_kmeans), 4),
    "±", round(sd(acc_teste_kmeans), 4), "\n")

cat("\n====== RBF aleatório ======\n")
cat("Acurácia média (treino):", round(mean(acc_treino_random), 4),
    "±", round(sd(acc_treino_random), 4), "\n")
cat("Acurácia média (teste):", round(mean(acc_teste_random), 4),
    "±", round(sd(acc_teste_random), 4), "\n")
```

Além das duas implementações anteriores, foi proposto no exercício que o aluno desenvolvesse um método próprio para atribuição de centros e raios. Para isso, utilizamos um algoritmo evolucionário conhecido como PSO (Particle Swarm Optimization).
Algoritmos evolucionários simulam processos biológicos para explorar grandes espaços de solução, buscando ótimos para problemas pouco convexos. No PSO, a otimização ocorre através da movimentação de partículas (ou indivíduos) em busca de minimizar uma função de fitness, que mede a qualidade de cada solução.

Neste caso, a função de fitness é definida pela média das distâncias entre cada amostra e o centro mais próximo. O objetivo do algoritmo é minimizar essa média — sendo a distância média zero a solução ideal (embora, na prática, quase inatingível). Por isso, o PSO é limitado a um número máximo de interações

Para ilustrar o funcionamento do PSO, imagine um bando de urubus em busca da refeição perfeita. Cada urubu é influenciado por três fatores: um coeficiente social (influência dos outros), um coeficiente pessoal (confiança em sua própria experiência) e um coeficiente de inércia (tendência de manter sua velocidade). A cada interação, os urubus ajustam sua trajetória com base na sua melhor experiência e na melhor experiência do grupo, atualizando suas posições no espaço de soluções até atingir o número máximo de interações ou encontrar uma solução ótima.

O algoritmo foi escolhido por sua capacidade de minimização e exploração de um espaço essencialmente ilimitado de soluções e testado exatamente da mesma forma que os dois anteriores, com 10 indivíduos por grupo e limite de 20 interações.

```r
# Main RBF function with PSO optimization
RBF_PSO <- function(xin, yin, p, swarm_size = 20, max_iter = 50) {
  radialnvar <- function(x, m, sigma) exp(-sum((x-m)^2) / (2*sigma^2))

  N <- dim(xin)[1]
  n <- dim(xin)[2]

  xin <- as.matrix(xin)
  yin <- as.matrix(yin)

  # Optimize centers using PSO
  cat("Optimizing RBF centers with PSO...\n")
  centers <- optimize_centers_PSO(xin, yin, p, swarm_size, max_iter)

  # Calculate appropriate radii for the centers
  radii <- calc_optimal_radii(centers, xin)

  # Create activation matrix H
  H <- matrix(nrow = N, ncol = p)
  for(j in 1:N) {
    for(i in 1:p) {
      H[j, i] <- radialnvar(xin[j, ], centers[i, ], radii[i])
    }
  }

  # Add bias term
  Haug <- cbind(1, H)

  # Calculate weights using pseudoinverse
  W <- pseudoinverse(Haug) %*% yin

  # Return model parameters
  return(list(centers = centers, radii = radii, W = W))
}
```

```r
YRBF_PSO <- function(xin, modRBF) {
  radialnvar <- function(x, m, sigma) exp(-sum((x-m)^2) / (2*sigma^2))

  N <- dim(xin)[1]

  centers <- modRBF$centers
  radii <- modRBF$radii
  W <- modRBF$W
  p <- nrow(centers)

  xin <- as.matrix(xin)
  H <- matrix(nrow = N, ncol = p)

  for(j in 1:N) {
    for(i in 1:p) {
      H[j, i] <- radialnvar(xin[j, ], centers[i, ], radii[i])
    }
  }

  # Add bias term
  Haug <- cbind(1, H)

  # Calculate predictions
  Yhat <- Haug %*% W

  return(Yhat)
}
```

```r
calc_optimal_radii <- function(centers, xin) {
  p <- nrow(centers)
  n <- ncol(centers)
  N <- nrow(xin)

  # Calculate average distance to nearest neighbor center for each center
  radii <- numeric(p)

  for (i in 1:p) {
    dist_to_other_centers <- numeric(p-1)
    idx <- 1
    for (j in 1:p) {
      if (i != j) {
        dist_to_other_centers[idx] <- sqrt(sum((centers[i, ] - centers[j, ])^2))
        idx <- idx + 1
      }
    }
    # Set radius as a fraction of average distance to other centers
    radii[i] <- mean(dist_to_other_centers) / 2
  }

  return(radii)
}
```

```r
optimize_centers_PSO <- function(xin, yin, p, swarm_size = 20, max_iter = 50, c1 = 1.5, c2 = 1.5, w = 0.7) {
  N <- dim(xin)[1]
  n <- dim(xin)[2]

  # Initialize particles (each particle represents a full set of centers)
  # Each particle has p centers, each center has n dimensions
  particles <- array(0, dim = c(swarm_size, p, n))
  velocities <- array(0, dim = c(swarm_size, p, n))

  # Initialize particles with random positions within the bounds of the data
  min_vals <- apply(xin, 2, min)
  max_vals <- apply(xin, 2, max)

  for (i in 1:swarm_size) {
    for (j in 1:p) {
      for (k in 1:n) {
        particles[i, j, k] <- runif(1, min_vals[k], max_vals[k])
        velocities[i, j, k] <- runif(1, -abs(max_vals[k] - min_vals[k])/10,
                                     abs(max_vals[k] - min_vals[k])/10)
      }
    }
  }

  particle_best_pos <- particles
  particle_best_fitness <- rep(Inf, swarm_size)
  global_best_pos <- particles[1, , ]
  global_best_fitness <- Inf
  # Define fitness function (average distance of samples to nearest center)
  calc_fitness <- function(centers) {
    distances <- matrix(0, nrow = N, ncol = p)

    for (i in 1:N) {
      for (j in 1:p) {
        distances[i, j] <- sqrt(sum((xin[i, ] - centers[j, ])^2))
      }
    }
    # For each sample, find distance to nearest center
    min_distances <- apply(distances, 1, min)
    return(mean(min_distances))
  }
```

```r
  for (iter in 1:max_iter) {
    # Evaluate fitness for each particle
    for (i in 1:swarm_size) {
      current_fitness <- calc_fitness(particles[i, , ])

      # Update particle's best position if needed
      if (current_fitness < particle_best_fitness[i]) {
        particle_best_pos[i, , ] <- particles[i, , ]
        particle_best_fitness[i] <- current_fitness

        # Update global best if needed
        if (current_fitness < global_best_fitness) {
          global_best_pos <- particles[i, , ]
          global_best_fitness <- current_fitness
        }
      }
    }

    # Update velocities and positions
    for (i in 1:swarm_size) {
      for (j in 1:p) {
        for (k in 1:n) {
          # Cognitive component
          r1 <- runif(1)
          cognitive <- c1 * r1 * (particle_best_pos[i, j, k] - particles[i, j, k])

          # Social component
          r2 <- runif(1)
          social <- c2 * r2 * (global_best_pos[j, k] - particles[i, j, k])

          # Update velocity with inertia
          velocities[i, j, k] <- w * velocities[i, j, k] + cognitive + social

          # Update position
          particles[i, j, k] <- particles[i, j, k] + velocities[i, j, k]

          # Keep within bounds
          particles[i, j, k] <- max(min_vals[k], min(particles[i, j, k], max_vals[k]))
        }
      }
    }

    # Optional: print progress
    if (iter %% 10 == 0) {
      cat("Iteration", iter, "- Best fitness:", global_best_fitness, "\n")
    }
  }

  # Return optimized centers
  return(global_best_pos)
}
```

Resultados:

Dataset: Breast Cancer

Número de centros (k)	RBF (k-means) treino	RBF (aleatorizado) treino	RBF (k-means) teste	RBF (aleatorizado) teste
10	0.8551 ± 0.0122	0.9696 ± 0.0048	0.8511 ± 0.0243	0.9678 ± 0.0102
15	0.8682 ± 0.011	0.9714 ± 0.0037	0.8611 ± 0.0245	0.9704 ± 0.0105
30	0.9017 ± 0.0089	0.9756 ± 0.0055	0.8706 ± 0.0246	0.9688 ± 0.0118
60	0.9335 ± 0.0071	0.9791 ± 0.0045	0.8836 ± 0.0184	0.9707 ± 0.0108
150	0.9669 ± 0.007	0.993 ± 0.0027	0.9083 ± 0.0262	0.9564 ± 0.0173
250	0.9817 ± 0.0053	1 ± 0	0.9089 ± 0.019	0.8959 ± 0.0269

Tabela 1: Médias e variâncias das acurácias obtidas para diferentes valores de *k* no *dataset Breast Cancer*

Dataset: Statlog (Heart)

Número de centros (k)	RBF (k-means) treino	RBF (aleatorizado) treino	RBF (k-means) teste	RBF (aleatorizado) teste
10	0.5506 ± 0.0206	0.7093 ± 0.0174	0.5679 ± 0.0485	0.6535 ± 0.0386
15	0.5587 ± 0.0185	0.7189 ± 0.027	0.5523 ± 0.047	0.679 ± 0.0448
30	0.5693 ± 0.0179	0.8261 ± 0.0288	0.5416 ± 0.0376	0.744 ± 0.0416
60	0.5887 ± 0.0161	0.9397 ± 0.0237	0.572 ± 0.0475	0.8115 ± 0.0475
150	0.8836 ± 0.019	1 ± 0	0.5243 ± 0.0697	0.9679 ± 0.0246
180	0.9743 ± 0.0082	1 ± 0	0.4872 ± 0.0759	0.9663 ± 0.0206

Tabela 2: Médias e variâncias das acurácias obtidas para diferentes valores de *k* no *dataset Statlog (Heart)*

Dataset: Breast Cancer

Número de centros (k)	RBF (PSO) treino	RBF (PSO) teste
10	0.9703 ± 0.0048	0.9678 ± 0.0112
15	0.9745 ± 0.0027	0.9629 ± 0.0082
30	0.9736 ± 0.0038	0.9707 ± 0.0077
60	0.9774 ± 0.0054	0.959 ± 0.0184

Tabela 3: Médias e variâncias das acurácias obtidas para diferentes valores de *k* no *dataset Breast Cancer*

Dataset: Statlog (Heart)

Número de centros (k)	RBF (PSO) treino	RBF (PSO) teste
10	0.6772 ± 0.015	0.6815 ± 0.0684
15	0.7175 ± 0.0286	0.6691 ± 0.0354
30	0.7905 ± 0.0445	0.679 ± 0.0509
60	0.8847 ± 0.0165	0.7284 ± 0.063

Tabela 4: Médias e variâncias das acurácias obtidas para diferentes valores de *k* no *dataset Statlog (Heart)*

Conclusão:

Analisando os valores de centro testados, especialmente no *dataset Statlog (Heart),* observamos que tanto o PSO quanto o método aleatorizado apresentaram desempenhos satisfatórios, com acurácias médias de 88,5% (60 centros, PSO, 10 indivíduos e 20 interações) e 94% (método aleatorizado, também 60 centros).

Para números elevados de centros (acima de 150), o método k-médias aproxima-se do desempenho do método aleatorizado. No entanto, ao observar os resultados obtidos sobre o *dataset Statlog (Heart)*, podemos observar que o k-médias não aparenta convergir para bons resultados no conjunto de testes mesmo com a elevação do número de centros, deixando a desejar quando comparado aos demais métodos testados.

Já no *dataset Breast Cancer*, a variação no número de centros para o método de atribuição por meio do PSO não impactou significativamente a performance. Nesse caso, o PSO foi capaz de encontrar soluções relativamente ótimas mesmo com poucos centros disponíveis — comportamento diferente do Statlog (Heart), onde o aumento do número de centros resultou em melhoria de desempenho.

Apêndice:

https://github.com/LucasKouKinoshita/RNA - Códigos elaborados