Assignment2Problemsolving.R

shadankhan

2024-08-27

```
##################################
#Question 1 - Understand the Data
##################################

# Load the raw data
dataRaw <- as.matrix(read.table("RedWine.txt"))

# Set seed using your student ID for reproducibility
set.seed(222623809)  # Replace with your actual student ID

# Subset the data (500 samples and first 6 columns)
# Subset the raw data to include 500 random samples
dataSubset <- dataRaw[sample(1:1599, 500), c(1:6)]
dataSubset <- dataRaw

# Define the variable names
variableNames <- c("citric acid", "chlorides", "total sulfur dioxide", "pH", "alcohol", "quality")

# Create histograms for each X variable and Y
png("histogramsPlots.png", width=1920, height=5000, res=200)
par(mfrow=c(6,1))
for (index in 1:6) {
  hist(dataSubset[,index],
       main = paste("Histogram of", variableNames[index]),
       xlab = variableNames[index],
       col = "skyblue",
       border = "black")
}
dev.off()  # Close the PNG device
```

```
## pdf
##   2
```

```
# Create scatterplots for each X variable against Y (Quality)
png("scatterPlots.png", width=1920, height=5000, res=200)
par(mfrow=c(5,1))
for (index in 1:5) {
  plot(dataSubset[,index], dataSubset[,6],
       xlab = variableNames[index],
       ylab = "Quality",
       col = "green",
```

```
      main = paste("Scatterplot of", variableNames[index], "vs Quality"),
      cex = 2
  )
}
dev.off()  # Close the PNG device
```

```
## pdf
##   2
```

```
################################
#Question 2 - Transform the Data
# Define the standardization function `unitZ`
# Define the standardization function `unitZ`
unitZ <- function(x) {
  0.5 + 0.15 * ((x - mean(x)) / sd(x))
}

# Define the Min-Max scaling function
minMaxScale <- function(x) {
  return ((x - min(x)) / (max(x) - min(x)))
}

originalVariables <- dataSubset

# Find the minimum and maximum values for each column (variable), mean values, and standard deviation f
outputMinValue <- min(originalVariables[,6])
outputMaxValue <- max(originalVariables[,6])
minValues <- apply(originalVariables, 2, min)
maxValues <- apply(originalVariables, 2, max)

# Define the index for the selected variables
selectedIndices <- c(1, 2, 3, 4, 5)
outputVariable <- dataSubset[,6]

# Extract the selected variables
selectedVariables <- dataSubset[, selectedIndices]

# Apply Min-Max scaling
scaledVariables <- apply(selectedVariables, 2, minMaxScale)

# Apply transformations based on the type of variable
# 1. Citric Acid (Column 1) - Power Transformation
power <- 0.5
scaledVariables[,1] <- scaledVariables[,1]^power

# Handle missing values for Chlorides (Column 2)
scaledVariables[,2][is.na(scaledVariables[,2])] <- 0  # Replace NAs with 0 or another small value

# 3. Total Sulfur Dioxide (Column 3) - Power Transformation
scaledVariables[,3] <- scaledVariables[,3]^power

# Calculate mean and standard deviation after scaling
meanValues <- apply(scaledVariables, 2, mean)
```

```r
sdValues <- apply(scaledVariables, 2, sd)

# Apply unitZ Standardization after Transformations
transformedData <- apply(scaledVariables, 2, unitZ)

# Apply Min-Max scaling and standardize the output variable
outputVariable <- minMaxScale(outputVariable)
outputMeanValue <- mean(outputVariable)
outputSDValue <- sd(outputVariable)
outputVariable <- unitZ(outputVariable)

# Combine the transformed data and standardized output
finalData <- cbind(transformedData, outputVariable)

# Function to calculate Negative Log-Likelihood for a Normal distribution
nll_normal <- function(x) {
  mu <- mean(x)
  sigma <- sd(x)
  -sum(dnorm(x, mean = mu, sd = sigma, log = TRUE))
}

# Function to calculate Negative Log-Likelihood for a Log-Normal distribution
nll_lognormal <- function(x) {
  mu <- mean(log(x))
  sigma <- sd(log(x))
  -sum(dlnorm(x, meanlog = mu, sdlog = sigma, log = TRUE))
}

# Function to calculate Negative Log-Likelihood for an Exponential distribution
nll_exponential <- function(x) {
  lambda <- 1 / mean(x)
  -sum(dexp(x, rate = lambda, log = TRUE))
}

# Calculate NLL for each variable in the transformed data
nll_values <- apply(transformedData, 2, function(x) {
  list(
    normal = nll_normal(x),
    lognormal = nll_lognormal(x),
    exponential = nll_exponential(x)
  )
})
```

```
## Warning in log(x): NaNs produced
## Warning in log(x): NaNs produced
```

```r
# Apply NLL calculations for the output variable as well
output_nll <- list(
  normal = nll_normal(outputVariable),
  lognormal = nll_lognormal(outputVariable),
  exponential = nll_exponential(outputVariable)
)
```

```r
# Print out the NLL values for each distribution
print("NLL values for each variable:")
```

```
## [1] "NLL values for each variable:"
```

```r
print(nll_values)
```

```
## $V1
## $V1$normal
## [1] -765.1121
##
## $V1$lognormal
## [1] -549.2343
##
## $V1$exponential
## [1] 490.6577
##
##
## $V2
## $V2$normal
## [1] -765.1121
##
## $V2$lognormal
## [1] -1455.108
##
## $V2$exponential
## [1] 490.6577
##
##
## $V3
## $V3$normal
## [1] -765.1121
##
## $V3$lognormal
## [1] -799.576
##
## $V3$exponential
## [1] 490.6577
##
##
## $V4
## $V4$normal
## [1] -765.1121
##
## $V4$lognormal
## [1] NA
##
## $V4$exponential
## [1] Inf
##
##
## $V5
## $V5$normal
```

```
## [1] -765.1121
##
## $V5$lognormal
## [1] -908.8311
##
## $V5$exponential
## [1] 490.6577
```

```
print("NLL values for the output variable:")
```

```
## [1] "NLL values for the output variable:"
```

```
print(output_nll)
```

```
## $normal
## [1] -765.1121
##
## $lognormal
## [1] -300.0656
##
## $exponential
## [1] 490.6577
```

```
# Interpretation of NLL values:
# For each variable, the distribution with the smallest NLL value is considered the best fit.
# - Compare the 'normal', 'lognormal', and 'exponential' NLL values for each variable.
# - The distribution with the lowest NLL value indicates that it is the most likely model for that vari

for (i in 1:length(nll_values)) {
  nll <- nll_values[[i]]
  best_fit <- names(nll)[which.min(unlist(nll))]
  cat(sprintf("Variable %d best fits with %s distribution\n", i, best_fit))
}
```

```
## Variable 1 best fits with normal distribution
## Variable 2 best fits with lognormal distribution
## Variable 3 best fits with lognormal distribution
## Variable 4 best fits with normal distribution
## Variable 5 best fits with lognormal distribution
```

```
# Also check for the output variable:
output_best_fit <- names(output_nll)[which.min(unlist(output_nll))]
cat(sprintf("Output variable best fits with %s distribution\n", output_best_fit))
```

```
## Output variable best fits with normal distribution
```

```
# Save the transformed data to a text file
write.table(finalData, "shadantransformed-data.txt", row.names=FALSE, col.names=FALSE)
```

```r
########################################
#Question 3 - Build models and investigate
########################################

# Source the function file
# Load the necessary functions from AggWaFit718.R
source("AggWaFit718.R")

# Import your saved transformed data into a matrix format
data.transformed_copy <- as.matrix(read.table("shadantransformed-data.txt"))

# Get weights for the Weighted Arithmetic Mean (QAM) using fit.QAM()

# 1. Fitting QAM with Arithmetic Mean (AM)
weights_QAM <- fit.QAM(data.transformed_copy)

# 2. Fitting QAM with Power Mean (PM) with P=2
# The function fit.QAM is used with a power mean (PM) where P=2.
# This means the aggregation will be biased towards higher values.
weights_QAM <- fit.QAM(data.transformed_copy, g=PM2, g.inv=invPM2)

# 3. Fitting QAM with Power Mean (PM) with P=0.5
# The function fit.QAM is now used with a power mean (PM) where P=0.5.
# This means the aggregation will be biased towards lower values.

# Identify rows that do not contain any negative values
rows_without_negatives <- apply(data.transformed_copy >= 0, 1, all)

# Filter the dataset to keep only the rows that do not contain negative values
data.transformed_copy <- data.transformed_copy[rows_without_negatives, ]

# Fit QAM using the filtered data with P=0.5 in the power mean
weights_QAM <- fit.QAM(data.transformed_copy, g=PM05, g.inv=invPM05)

# 4. Get weights for the Choquet integral
# The Choquet integral is a more general aggregation operator that can handle interactions among criter
weights_choquet <- fit.choquet(data.transformed_copy)

# 5. Get weights for the Ordered Weighted Average (OWA)
# The Ordered Weighted Average (OWA) is an aggregation operator that emphasizes certain orders of value
weights_OWA <- fit.OWA(data.transformed_copy)



########################################
# Question 4 - Use Model for Prediction
########################################

# Define the min-max scaling function
scale_minmax <- function(x, min_val, max_val) {
  # Scale x to the range [0, 1] using min-max scaling
  return ((x - min_val) / (max_val - min_val))
}
```

```r
# Define the reverse min-max scaling function to revert scaling
reverse_scale_minmax <- function(x, min_val, max_val) {
  # Revert the scaling from the range [0, 1] back to the original range
  return (x * (max_val - min_val) + min_val)
}

# Define the unitZ standardization function that stores mean and sd
standardize_unitZ <- function(x, mean_val, sd_val){
  # Standardize x using unitZ scaling, which is a custom scaling to a specific range
  return (0.15 * ((x - mean_val) / sd_val) + 0.5)
}

# Define the reverse of the unitZ standardization function
reverse_standardize_unitZ <- function(x, mean_val, sd_val) {
  # Revert the unitZ scaling back to the original scale
  return (((x - 0.5) / 0.15) * sd_val + mean_val)
}

# Define the indices for the selected variables (citric acid, chlorides, total sulfur dioxide, pH, alco
indices <- c(1, 2, 3, 4, 5, 6)

# Subset your data (example data subset, replace with actual data)
# Extracting the relevant columns from your data
variables_to_transform <- dataSubset[, indices]

# Define the power transformation parameter
power_transform <- 0.5

# Define the new input for prediction (example values, replace with actual)
# The order of variables should match the order used in transformations: citric acid, chlorides, total
input_data <- c(0.9, 0.65, 38, 2.53, 7.1)

# Select the same variables as in Question 2 for transformation
# Initialize a numeric vector to store the transformed input
input_transformed <- numeric(length(input_data))

# 1. Apply transformations to the first variable (citric acid)
scaled_citric_acid <- scale_minmax(input_data[1], minValues[1], maxValues[1])
# Apply unitZ standardization after power transformation
input_transformed[1] <- standardize_unitZ(scaled_citric_acid^power_transform, meanValues[1], sdValues[1]

# 2. Apply transformations to the second variable (chlorides)
scaled_chlorides <- scale_minmax(input_data[2], minValues[2], maxValues[2])
# Apply unitZ standardization without power transformation
input_transformed[2] <- standardize_unitZ(scaled_chlorides, meanValues[2], sdValues[2])

# 3. Apply transformations to the third variable (total sulfur dioxide)
scaled_total_sulfur_dioxide <- scale_minmax(input_data[3], minValues[3], maxValues[3])
# Apply unitZ standardization after power transformation
input_transformed[3] <- standardize_unitZ(scaled_total_sulfur_dioxide^power_transform, meanValues[3], s

# 4. Apply transformations to the fourth variable (pH)
scaled_pH <- scale_minmax(input_data[4], minValues[4], maxValues[4])
```

```r
# Apply unitZ standardization without power transformation
input_transformed[4] <- standardize_unitZ(scaled_pH, meanValues[4], sdValues[4])

# 5. Apply transformations to the fifth variable (alcohol)
scaled_alcohol <- scale_minmax(input_data[5], minValues[5], maxValues[5])
# Apply unitZ standardization without power transformation
input_transformed[5] <- standardize_unitZ(scaled_alcohol, meanValues[5], sdValues[5])
```

QAM' is the prediction function that takes the transformed input and weights

```r
predicted_output <- QAM(input_transformed, weights_QAM)

# Reverse the transformations on the predicted output
reversed_output <- reverse_standardize_unitZ(predicted_output, outputMeanValue, outputSDValue)

# Final prediction on the original scale (reversing min-max scaling)
final_prediction <- round(reverse_scale_minmax(reversed_output, outputMinValue, outputMaxValue))

# Print the final prediction
print(final_prediction)
```

```
## [1] 5
```

```r
# Continue with the prediction process...

##########################################
# Save the script as "name-code.R"
##########################################
```