Case Study 1 - Neural Networks Solutions

Today's Goal

This is the first case study in our series focusing on the application of neural networks in real-world scenarios. In this case study, we are going to explore how neural networks can be utilized to address a common health issue: obesity. Our main goal today is to build a neural network model that can predict obesity based on various demographic, physical, and lifestyle factors.

Dataset Overview

The dataset we will be working with is sourced from a popular Kaggle collection. It gathers comprehensive information on individuals' demographic characteristics, physical attributes, and lifestyle habits. These variables offer valuable insights into the factors influencing obesity outcomes. Here is the original dataset source: https://www.kaggle.com/datasets/mrsimple07/obesity-prediction/data.

For our case study, we will work on an extension of this dataset, which includes additional information on sleep hours and daily calorie intake. Please download the dataset 'obesity_dataset.csv' from the github folder 'case study'.

1.Getting Started

A. Initial Setup

(i) To begin, make sure you have the following libraries installed and loaded in your R environment: ggplot2, dplyr, caret, neuralnet, and tidyverse.

```r
# install.packages('neuralnet')
# install.packages('caret')
library(tidyverse)
library(neuralnet)
library(dplyr)
library(ggplot2)
library(caret)
```

(ii) Set the seed to '123' to ensure the reproducibility of your results. This will help you and others to obtain the same outcomes when running the same codes.

Learning Note: Setting a seed in R ensures that any analysis involving random operations (like sampling, partitioning data into training and test sets, or initializing random weights in machine learning algorithms) produces the same results each time it is run. This is crucial for scientific work, allowing others (or your future self) to replicate your analysis and get the same results.

```r
set.seed(123)
```

(iii) Now, please load the dataset that you've downloaded into your R environment. Assign this data to an object named 'my_data'.

```r
my_data <- read.csv('obesity_dataset.csv')
```

B. Data Cleanings

(i) Familiarize yourself with the dataset's structure.

Hint: You can use functions like `head()`, `summary()`, and `glimpse()` to get an overview of the data.

```r
head(my_data)
```

```
##   Age Gender   Height   Weight PhysicalActivityLevel ObesityCategory  Race
## 1  56   Male 173.5753 71.98205                     4   Normal weight Asian
## 2  69   Male 164.1273 89.95926                     2           Obese Asian
## 3  46 Female 168.0722 72.93063                     4      Overweight Black
## 4  32   Male 168.4596 84.88691                     3      Overweight Black
## 5  60   Male 183.5686 69.03895                     3   Normal weight Asian
## 6  25 Female 166.4056 61.14587                     4   Normal weight Other
##   MaritalStatus DailyCalorieIntake SleepHours
## 1      Divorced           1849.625   8.420019
## 2       Widowed           1876.121   8.359398
## 3       Married           2251.918   5.872235
## 4      Divorced           1642.401   8.031333
## 5       Widowed           1964.200   7.701170
## 6      Divorced           2051.847   6.495235
```

```r
# glimpse(my_data)
```

(ii) Identify how many variables (columns) and observations (rows) there are.

Our data contains 1000 observations and 10 variables.

(iii) Determine which variables are numerical and which are categorical.

- **Numerical**: Age, Height, Weight, DailyCalorieIntake, SleepHours
- **Categorical**: Gender, Physical Activity Level, ObesityCategory, Race, Marital Status

Now it's time to tidy up the dataset to make sure it's optimized for our analysis.

Through preliminary research and domain knowledge, we've identified that certain variables, such as 'MaritalStatus' and 'Race', might not significantly contribute to predicting obesity in our specific context.

(iv) Exclude the variables 'MaritalStatus' and 'Race' from the dataset.

Discussion: Explain why it's important to focus only on relevant variables, and how this step can improve model accuracy and reduce complexity.

```r
my_data <- my_data %>% select(-MaritalStatus, -Race)
```

Focusing on relevant variables is essential because it directly impacts the accuracy and complexity of our models. By excluding irrelevant or less significant features, we can enhance the model's ability to make accurate predictions, as it reduces the noise and distraction from unimportant data. This process not only simplifies the model, making it easier to interpret and faster to train, but also helps in mitigating the risk of overfitting.

(v) The ‘Gender’ variable is categorical and needs to be transformed for the neural network to process it. Please convert ‘Gender’ into its numeric format.

```r
my_data$Gender <- as.numeric(factor(my_data$Gender, levels = c("Male", "Female")))
```

(vi) Create a new variable named ‘BMI’ by using the weight(kg) and height(cm) information available in our data.

Hint: BMI = Weight(kg)/Height(m)ˆ2

```r
my_data <- my_data %>%
  mutate(BMI = Weight/(Height/100)^2)
```

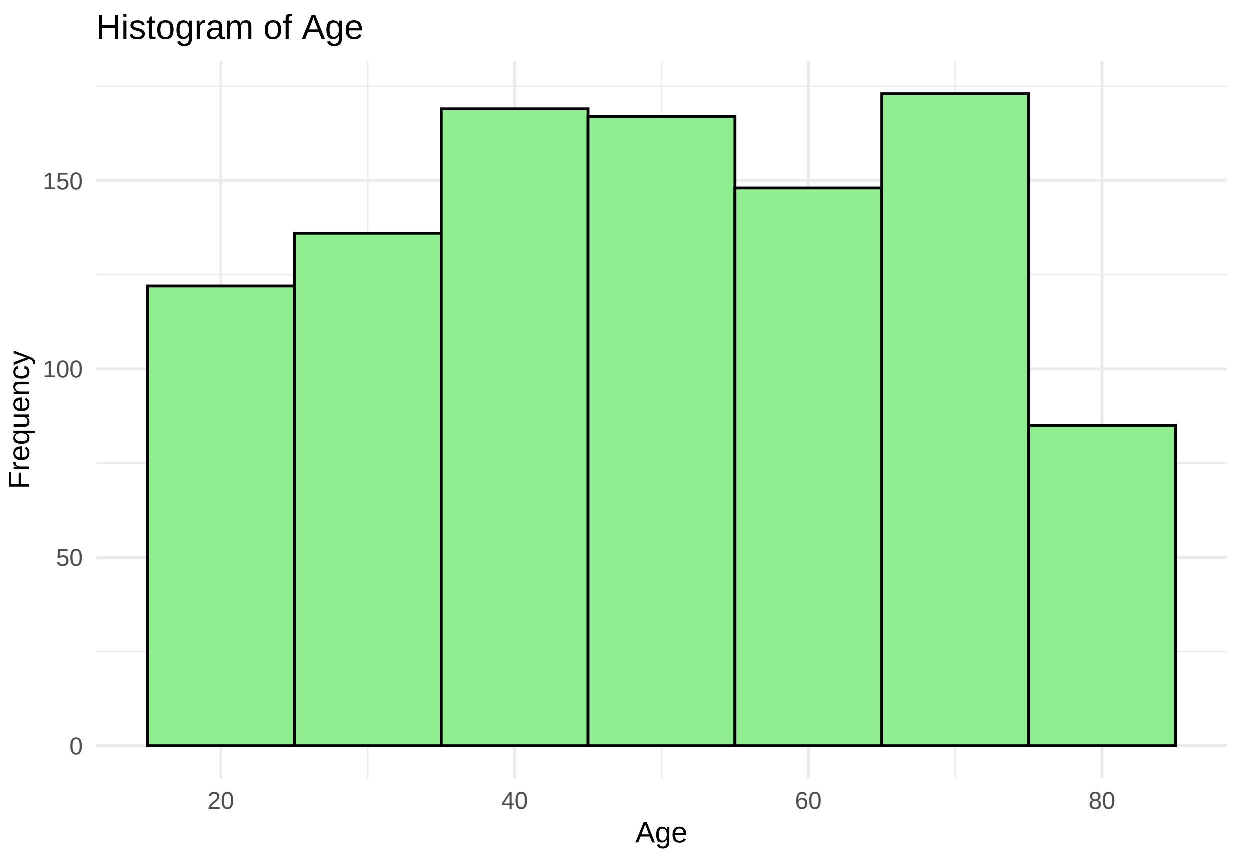
(vii) Round the numerical values of Weight, Height, BMI, DailyCalorieIntake, and SleepHours to one decimal place for consistency.

```r
my_data$Weight <- round(my_data$Weight, 1)
my_data$Height <- round(my_data$Height, 1)
my_data$BMI <- round(my_data$BMI, 1)
my_data$DailyCalorieIntake <- round(my_data$DailyCalorieIntake, 1)
my_data$SleepHours <- round(my_data$SleepHours, 1)
```

2. Data Visualizations

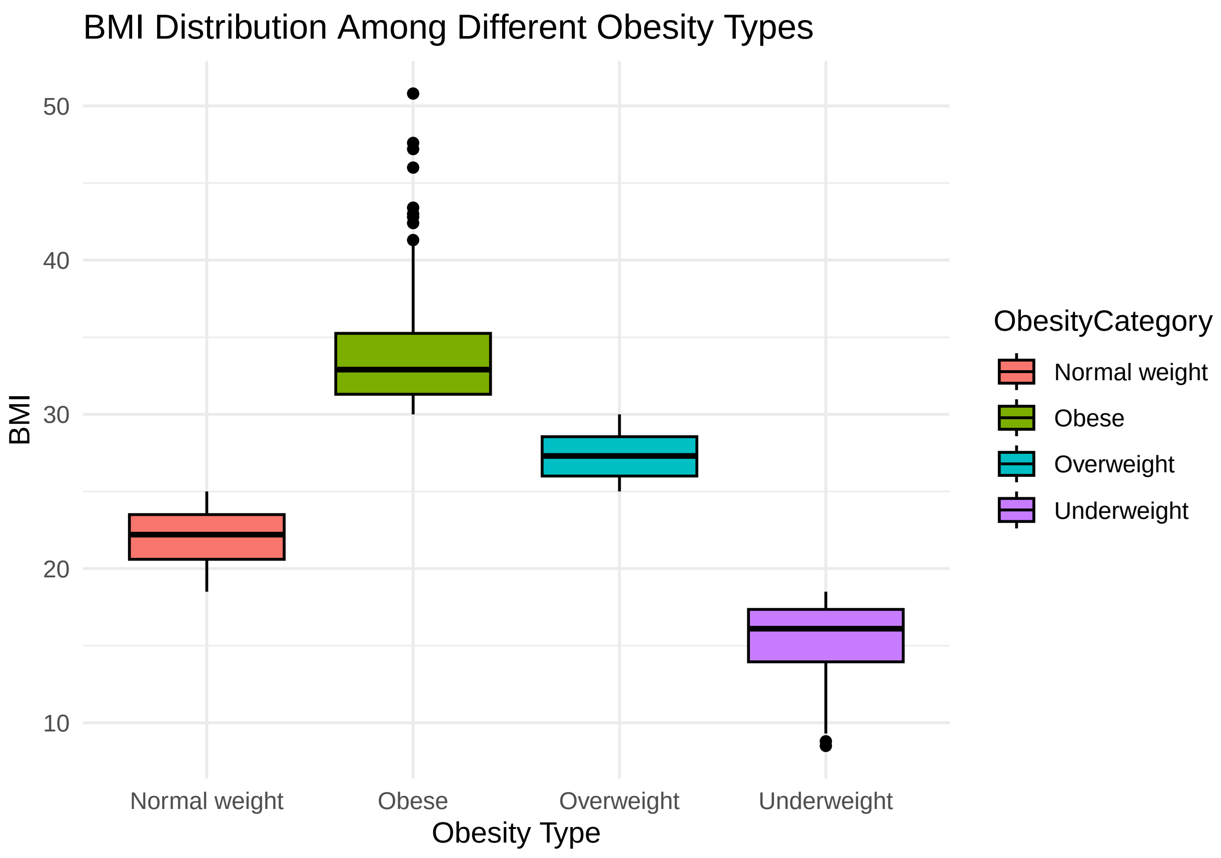
(i) Create a histogram in lightgreen color to visualize the age distribution in the dataset. Make sure you generate a plot with clear title and axis label.

```r
ggplot(my_data, aes(x = Age)) +
  geom_histogram(binwidth = 10, fill = "lightgreen", color = "black") +
  theme_minimal() +
  labs(title = "Histogram of Age", x = "Age", y = "Frequency")
```



(ii) Generate boxplots to compare the distribution of BMI values across different obesity categories. What can you infer from the generated boxplots?

```r
ggplot(my_data, aes(x = ObesityCategory, y = BMI, fill = ObesityCategory)) +
  geom_boxplot(color = "black") +
  labs(title = "BMI Distribution Among Different Obesity Types", x = "Obesity Type", y = "BMI") +
  theme_minimal()
```



3. Model Building

(i) Before building our neural network models, we need to first scale all numerical variables in our dataset to ensure they all contribute equally to the analysis. Use the `scale()` function to normalize these variables.

```
variables_to_scale <- c("Age", "Height", "Weight", "BMI", "DailyCalorieIntake", "SleepHours")
my_data[variables_to_scale] <- scale(my_data[variables_to_scale])
```

(ii) Divide the dataset into 70% training and 30% testing sets.

```
data_rows <- floor(0.7 * nrow(my_data))
train_indices <- sample(c(1:nrow(my_data)), data_rows)
train_data <- my_data[train_indices,]
test_data <- my_data[-train_indices,]
```

(iii) Constructing the Neural Network Model:

- Use the `neuralnet()` function to build a neural network model. Recall that our goal is to classify individuals into specific obesity categories.
- Use Age, Gender, Weight, Height, BMI, PhysicalActivityLevel, SleepHours, and DailyCalorieIntake as predictors.
- Set up the network with one hidden layer containing 5 neurons.
- To address potential convergence issues, set the learning rate to 0.001 by including learningrate = 0.001 in your model configuration.

Learning Note: The learning rate determines how much we adjust the model's parameters in response to the error it made on the last prediction. It's like setting the pace for a model's learning; too fast and the model might overshoot the target (optimal solution), too slow and it might take too long to reach it, or get stuck in a less optimal solution. An optimal learning rate helps the model to converge more quickly and reliably to a good solution. This is because it balances the need for large enough steps to make noticeable progress but small enough to refine the solution as it gets closer to the optimum.

```r
model_1 <- neuralnet(
    ObesityCategory~Age + Gender + Weight + Height + BMI +
                    PhysicalActivityLevel + SleepHours + DailyCalorieIntake,
    data=train_data,
    hidden=c(5),
    linear.output=FALSE,
    learningrate = 0.001
)
```

```r
plot(model_1)
```

(iv) Evaluate the model's performance by making predictions using the `predict()` function. Store the results in an object named 'test_predictions'.

Hint: Before applying the `predict()` function, make sure to remove the 'ObesityCategory' from the test dataset using `test <- test_data %>% select(-ObesityCategory)`. This ensures that the test dataset only contains predictor variables.

Run the provided code to convert the predictions into categorical factors:

```r
max_indices <- max.col(test_predictions, ties.method = "first")
categories <- c('Normal weight', 'Obese', 'Overweight', 'Underweight')
test_predictions <- factor(categories[max_indices], levels = categories)
```

The first line of code identifies the index of the highest value in each row of test_predictions. In cases of ties, where two or more columns in the same row share the maximum value, the first occurrence is selected. This step is crucial for determining each observation's predicted category based on the highest probability.

Next, a vector named categories is created, listing the category names in alignment with the column order of test_predictions. The indices in max_indices are then mapped to the corresponding category names using this vector. Finally, these names are transformed into a factor, with levels explicitly defined by the categories vector to maintain the predefined order of categories.

```r
# Compute predictions
test <- test_data %>% select(-ObesityCategory)

# Predict with the model
test_predictions <- predict(model_1, test)

# Convert matrix of predictions to categorical predictions
max_indices <- max.col(test_predictions, ties.method = "first")
categories <- c('Normal weight', 'Obese', 'Overweight', 'Underweight')
test_predictions <- factor(categories[max_indices], levels = categories)
```

(v) Compute the accuracy of your model and construct a confusion matrix to evaluate its performance.

Hint: Use the `confusionMatrix()` function, ensuring that both the predictions and the true labels are in factor format.

Discussion: Comment on the model's performance. What does the accuracy and the confusion matrix tell you about the model's predictive capabilities?

```r
# Check if the predicted classes match the actual classes
check_1 <- test_data$ObesityCategory == test_predictions
```

```r
# Calculate accuracy as a percentage
accuracy_1 <- (sum(check_1) / nrow(test_data)) * 100

print(paste("Accuracy:", accuracy_1, "%"))
```

```
## [1] "Accuracy: 23.6666666666667 %"
```

```r
# Confusion Matrix
confusionMatrix <- confusionMatrix(as.factor(test_predictions), as.factor(test_data$ObesityCategory))
print(confusionMatrix)
```

```
## Confusion Matrix and Statistics
##
##                  Reference
## Prediction         Normal weight Obese Overweight Underweight
##   Normal weight              0     0          0           0
##   Obese                     70     2         20          42
##   Overweight                38    54         69           5
##   Underweight                0     0          0           0
##
## Overall Statistics
##
##                  Accuracy : 0.2367
##                    95% CI : (0.1897, 0.2889)
##     No Information Rate : 0.36
##     P-Value [Acc > NIR] : 1
##
##                     Kappa : -0.0144
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                       Class: Normal weight Class: Obese Class: Overweight
## Sensitivity                           0.00     0.035714            0.7753
## Specificity                           1.00     0.459016            0.5403
## Pos Pred Value                         NaN     0.014925            0.4157
## Neg Pred Value                        0.64     0.674699            0.8507
## Prevalence                            0.36     0.186667            0.2967
## Detection Rate                        0.00     0.006667            0.2300
## Detection Prevalence                  0.00     0.446667            0.5533
## Balanced Accuracy                     0.50     0.247365            0.6578
##                       Class: Underweight
## Sensitivity                       0.0000
## Specificity                       1.0000
## Pos Pred Value                       NaN
## Neg Pred Value                    0.8433
## Prevalence                        0.1567
## Detection Rate                    0.0000
## Detection Prevalence              0.0000
## Balanced Accuracy                 0.5000
```

- The model has an accuracy rate of 23.67%, which is significantly lower than what would be considered effective for predictive tasks. This low accuracy suggests that the model struggles to correctly classify the majority of instances into the correct categories of weight status (Normal weight, Obese, Overweight, Underweight).

- The model has a very low sensitivity (true positive rate) for all classes except 'Overweight', which has a sensitivity of 77.53%. This means it is relatively better at identifying 'Overweight' instances but fails to accurately identify most instances of 'Normal weight', 'Obese', and 'Underweight'.

(vi) You may have noticed that the initial model shows poor performance, try adding an extra layer and see if it improves. Build a new model with the same predictors but with two hidden layers – the first with 5 neurons and the second with 4 neurons.

```
model_2 <- neuralnet(
    ObesityCategory~Age + Gender + Weight + Height + BMI +
                    PhysicalActivityLevel + SleepHours + DailyCalorieIntake,
    data=train_data,
    hidden=c(5, 4),
    linear.output=FALSE,
    learningrate = 0.001
)
```

```
plot(model_2)
```

(vii) Repeat the prediction process using the new model. Calculate the accuracy and construct a confusion matrix for this updated model and analyze how it performs compared to the initial model.

```
# Compute predictions
test <- test_data %>% select(-ObesityCategory)

# Predict with the model
test_predictions <- predict(model_2, test)

# Convert matrix of predictions to categorical predictions
max_indices <- max.col(test_predictions, ties.method = "first")
categories <- c('Underweight', 'Overweight', 'Normal weight', 'Obese')
test_predictions <- factor(categories[max_indices], levels = categories)
```

```
# Check if the predicted classes match the actual classes
check_2 <- test_data$ObesityCategory == test_predictions

# Calculate accuracy as a percentage
accuracy_2 <- (sum(check_2) / nrow(test_data)) * 100

print(paste("Accuracy:", accuracy_2, "%"))
```

```
## [1] "Accuracy: 26 %"
```

```
# Confusion Matrix
confusionMatrix <- confusionMatrix(as.factor(test_predictions), as.factor(test_data$ObesityCategory))
```

```
## Warning in confusionMatrix.default(as.factor(test_predictions),
## as.factor(test_data$ObesityCategory)): Levels are not in the same order for
## reference and data. Refactoring data to match.
```

```
print(confusionMatrix)
```

```
## Confusion Matrix and Statistics
##
##                  Reference
## Prediction        Normal weight Obese Overweight Underweight
##   Normal weight              17    24         28           1
##   Obese                     0     0          0           0
##   Overweight               91    32         61          46
##   Underweight               0     0          0           0
##
## Overall Statistics
##
##                Accuracy : 0.26
##                  95% CI : (0.2113, 0.3135)
##     No Information Rate : 0.36
##     P-Value [Acc > NIR] : 0.9999
##
##                   Kappa : -0.0747
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: Normal weight Class: Obese Class: Overweight
## Sensitivity                       0.15741       0.0000            0.6854
## Specificity                       0.72396       1.0000            0.1991
## Pos Pred Value                    0.24286          NaN            0.2652
## Neg Pred Value                    0.60435       0.8133            0.6000
## Prevalence                        0.36000       0.1867            0.2967
## Detection Rate                    0.05667       0.0000            0.2033
## Detection Prevalence              0.23333       0.0000            0.7667
## Balanced Accuracy                 0.44068       0.5000            0.4422
##                      Class: Underweight
## Sensitivity                      0.0000
## Specificity                      1.0000
## Pos Pred Value                      NaN
## Neg Pred Value                   0.8433
## Prevalence                       0.1567
## Detection Rate                   0.0000
## Detection Prevalence             0.0000
## Balanced Accuracy                0.5000
```

After we adding an extra layer to the neural network, there's a slight decrease in overall accuracy to approximately 26% compared with model 1. This suggests that the additional complexity introduced by the extra layer has not improved the model's ability to correctly classify instances across the categories of weight status.

What we can infer from these results:

- Overfitting: The model might be overfitting the training data, suggesting that our data size might be small relative to the model's complexity.
- Irrelevant Features: If the dataset contains irrelevant features, increasing model complexity might exacerbate the model's focus on noise rather than underlying patterns related to obesity categories.