A logo on a black background

Description automatically generated

**Assignment for PMIM402**

|  |  |
| --- | --- |
| **Module number:** | PMIM402 |
| **Module name:** | Machine Learning |
| **Title of assignment:** | *Classification Assignment* |
| **Student ID number:** | 2333193 |
| **Word count:** |  |
| **Declaration:** | I understand the following conditions which apply throughout this course:   1. I confirm that I am the sole author of this work. 2. I understand that proof reading by a third party is discouraged, but if used, records should be available as per guidelines. 3. I understand the need for academic integrity and that all my submitted work will adhere to its principles. 4. I understand that the teaching team will take measures to deter, detect and report any academic misconduct. 5. I agree to my work being submitted to the TurnItIn academic database. 6. I understand the importance of assignment deadlines and the need to seek help in good time where personal circumstances interrupt my work. |
| **Please copy and paste this declaration onto the front of the submission.** | |

**Part 1:**

1. I undertook preprocessing and used the resulting data frame for both classification methods I chose: Logistic Regression and Random Forest. Preprocessing is crucial for aligning the data format with the requirements of machine learning models, enhancing model performance, and ensuring accurate prediction. For this project, the preprocessing steps included:

- Removing irrelevant or noisy features to reduce dimensionality and focus on meaningful predictors.

- I converted categorical variables into their respective labels because they are nominal and should not imply any inherent order. Then I applied one-hot encoding for model compatibility.

- I scaled continuous features to normalise their ranges. This step is particularly important for Logistic Regression. It ensures that all features contribute equally to the model’s performance.

1. Below is the code for preprocessing and running the models with default parameters.

library(tidyverse)

library(randomForest)

library(caret)

library(pROC)

modified\_data <- read\_csv("C:/Users/USER/Desktop/HDS/machine learning/data/heart\_disease\_modified.csv")

sum(is.na(modified\_data))

str(modified\_data)

modified\_data <- modified\_data %>% select(-Patient\_ID, -"...1",-pace\_maker,

-perfusion, -traponin)

# Recoding 'cp'

modified\_data$cp <- factor(modified\_data$cp, levels = c(1, 2, 3, 4), labels = c("typ\_angina", "atyp\_angina", "non\_anginal", "asympt"))

# Recoding 'restecg'

modified\_data$restecg <- factor(modified\_data$restecg, levels = c(0, 1, 2), labels = c("normal", "st\_t\_wave\_abnormality", "left\_vent\_hyper"))

# Recoding 'slope'

modified\_data$slope <- factor(modified\_data$slope, levels = c(1, 2, 3), labels = c("up", "flat", "down"))

# Recoding 'thal'

modified\_data$thal <- factor(modified\_data$thal, levels = c(3, 6, 7), labels = c("normal", "fixed\_defect", "reversable\_defect"))

# Recoding 'drug'

modified\_data$drug <- factor(modified\_data$drug, levels = c("Aspirin", "Both", "Clopidogrel", "None"))

# Recoding 'fam\_hist' (family history of heart disease)

modified\_data$fam\_hist <- ifelse(modified\_data$fam\_hist== "yes", 1, 0)

# Identify continuous features

continuous\_features <- c("age", "trestbps", "chol", "thalach", "oldpeak", "ca")

# Scale continuous features only

modified\_data[continuous\_features] <- scale(modified\_data[continuous\_features])

# One-hot encoding

dummy\_model <- dummyVars("~ .", data = modified\_data, fullRank = TRUE)

modified\_data\_transformed <- predict(dummy\_model, newdata = modified\_data)

data\_transformed <- as.data.frame(modified\_data\_transformed)

data\_transformed$class <- factor(data\_transformed$class, levels=c("0", "1"))

set.seed(123)

# Calculate indices for splitting

total\_rows <- nrow(data\_transformed)

train\_rows <- round(0.75 \* total\_rows)

val\_rows <- round(0.15 \* total\_rows)

# The remaining rows go to the test set

test\_rows <- total\_rows - train\_rows - val\_rows

# Create random indices

indices <- sample(1:total\_rows)

# Split data based on calculated indices

trainset <- data\_transformed[indices[1:train\_rows], ]

valset <- data\_transformed[indices[(train\_rows + 1):(train\_rows + val\_rows)], ]

testset <- data\_transformed[indices[(train\_rows + val\_rows + 1):total\_rows], ]

# Verify the sizes

print(nrow(trainset))

print(nrow(valset))

print(nrow(testset))

# Logistic Regression

model\_log <- glm(class ~ ., data = trainset, family = binomial)

predictions\_log <- predict(model\_log, valset, type = "response")

# Convert probabilities to binary outcome

predictedClass\_log <- ifelse(predictions\_log > 0.5, 1, 0)

# Evaluation

predictedClass\_log\_factor <- factor(predictedClass\_log, levels=c("0", "1"))

confusion\_log <- confusionMatrix(data=predictedClass\_log\_factor, reference=valset$class, positive="1")

sensitivity\_log <- confusion\_log$byClass["Sensitivity"]

sensitivity\_log

confusionMatrix\_log <- table(Predicted = predictedClass\_log\_factor, Actual = valset$class)

confusionMatrix\_log

# Random Forest

model\_rf <- randomForest(class ~ ., data = trainset)

predictions\_rf <- predict(model\_rf, valset)

# Evaluation

predictions\_rf\_factor <- factor(predictions\_rf, levels=c("0", "1"))

confusion\_rf <- confusionMatrix(data=predictions\_rf\_factor, reference=valset$class, positive="1")

sensitivity\_rf <- confusion\_rf$byClass["Sensitivity"]

sensitivity\_rf

confusionMatrix\_rf <- table(Predicted = predictions\_rf\_factor, Actual = valset$class)

confusionMatrix\_rf

1. The accuracy with which each classifier can predict patients who develop heart disease is quantified by the sensitivity metric. It indicates the proportion of actual positive cases (patients with heart disease) correctly identified by the model. For the Logistic Regression model, the sensitivity is approximately 78.1%, and the Random Forest model exhibits a sensitivity of approximately 82.2%. These values are found in the model’s output, specifically in the evaluation provided by the confusionMatrix() function.
2. People who are misclassified as developing heart disease are referred to as false positives (FP) – Patients who do not have the disease but are predicted by the model to have it.

The number of false positives is found in the confusion matrix output, which is displayed using table() function. The confusion matrix provides a detailed breakdown of the model’s prediction against the actual classes.

For the Logistic Regression: 0 = no heart disease, 1 = having heart disease.

Based on the confusion matrix:

|  |  |  |
| --- | --- | --- |
|  | Actual | |
| Predicted | 0 | 1 |
| 0 | 50 | 16 |
| 1 | 15 | 57 |

The number of false positives is 15.

For the Random Forest:

Based on the confusion matrix: 0 = no heart disease, 1 = having heart disease.

|  |  |  |
| --- | --- | --- |
|  | Actual | |
| Predicted | 0 | 1 |
| 0 | 47 | 13 |
| 1 | 18 | 60 |

The number of false positives is 18. The number is found in the cell that corresponds to Predicted = 0 and Actual = 1.

1. Below are the code and the ROC Curves of the Logistic regression model and Random Forest.

# ROC Curve for Logistic Regression

roc\_log <- roc(valset$class, predictions\_log)

plot(roc\_log, main="ROC Curve for Logistic Regression")

# ROC Curve for Random Forest

predictions\_rf\_prob <- predict(model\_rf, valset, type="prob")[,2] # Get probabilities for class=1

roc\_rf <- roc(valset$class, predictions\_rf\_prob)

plot(roc\_rf, main="ROC Curve for Random Forest")

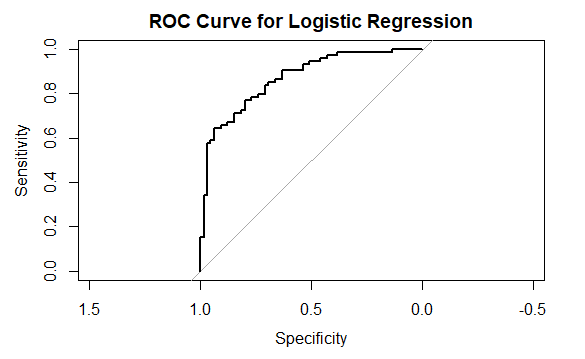
# AUC to compare the models

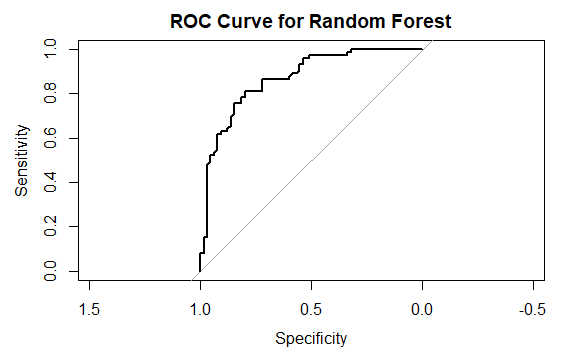
auc\_log <- auc(roc\_log)

auc\_rf <- auc(roc\_rf)

auc\_log

auc\_rf





ROC curve is a visual representation that illustrates the trade-off between sensitivity and false positive rate (1- specificity) at various threshold settings. Another commonly used measure of accuracy is the Area Under the Curve (AUC). It summarises the ROC curve’s information into a single value (from 0 to 1) where higher values indicate a better model performance in terms of distinguishing between classes (having the disease or not).

**Part 2:**

1. I chose Random Forest over Logistic Regression by comparing the following measures metrics:

- Random Forest demonstrated higher sensitivity (86.2%), more accurately predicting patients with heart disease, and thereby minimizing false negatives. This characteristic is crucial in medical diagnostics, where overlooking a case can lead to severe consequences (1).

- Although the difference in AUC values is very small, with Random Forest at 0.8685 and Logistic Regression at 0.8683. Random Forest performs slightly better in distinguishing between patients with and without heart disease across all possible classification thresholds.

1. Theoretical foundation:

Random Forest (RF) operates on the principle of ensemble learning, which combines multiple decision trees to improve model accuracy and robustness.

The algorithm begins by creating multiple decision trees, each trained on a random subset of the original dataset with replacement (a bootstrap sample). This bootstrap sampling process introduces variability among the trees, which helps to reduce the overall variance of the model without increasing bias.

At each decision node within a tree, a subset of features is randomly selected as candidates for the split. This randomness enhances the model’s robustness to noise and overfitting by ensuring that the trees are less correlated with each other. Consequently, this leads to more independent errors among the trees. When the predictions from the individual trees are aggregated, these independent errors are more likely to cancel each other out, leading to a more accurate and stable overall prediction. This process effectively reduces the model’s variance. Leo Breiman, the creator of RF, introduced this technique to de-correlate the trees, enhancing the ensemble’s predictive performance.

The individual trees are grown using a “top-down” greedy approach, starting from the root node and splitting down towards the leaves. The “greedy” approach means that at each decision node, the algorithm selects the best split from the randomly selected subset of features. This selection is based on the criterion like entropy or Gini impurity. This process aims to choose the split that maximizes the homogeneity (or purity) of the target variable within the resulting subsets, thereby enhancing the accuracy of predictions. The process continues recursively until terminal nodes (leaves) are reached. These leaves represent the predictions—indicating the most frequent class in classification tasks or the average outcome in regression tasks.

Prediction mechanism:

For classification tasks, the prediction is determined by majority voting among all the trees. Each tree provides a vote based on its prediction, and the class with the majority of votes is chosen as the model’s prediction.

For regression tasks, the final prediction is the average of the predictions from all the trees in the forest. (1–3)

**Advantages:**

- By averaging the results from numerous trees, RF mitigates overfitting, leading to better generalisation to unseen data.

- RF can handle large datasets with high dimensionality and can accommodate both quantitative and qualitative input variables. It is relatively immune to the presence of outliers and redundant variables.

- RF provides insight into the importance of each feature in making predictions, which is valuable for understanding the factors driving the model’s decisions.

- While RF works well with defaults settings, performance can often be further optimized by tuning parameters such as the number of trees (ntree) and the number of features considered for splitting at each node (mtry). (1–3)

1. a- The importance() function on the Random Forest model (“importance(model\_rf)“) yields the following output :

|  |  |
| --- | --- |
|  | MeanDecreaseGini |
| sex | 11.866866 |
| age | 28.720394 |
| Cp = Chest pain | |
| cp.atyp\_angina | 21.414288 |
| cp.non\_anginal | 5.841845 |
| cp.asympt | 37.063057 |
| trestbps = resting blood pressure | |
| trestbps | 21.583687 |
| chol | 34.590645 |
| fbs = (fasting blood sugar > 120 mg/dl) | 4.984844 |
| restecg = resting electrocardiographic results | |
| restecg.st\_t\_wave\_abnormality | 5.808499 |
| restecg.left\_vent\_hyper | 4.522956 |
| Thalach = maximum heart rate achieved | 36.494600 |
| exang = exercise induced angina ( | 25.561004 |
| oldpeak ST depression induced by exercise relative to rest perfusion: results of perfusion scan | 28.417772 |
| slope = the slope of the peak exercise ST segment | |
| slope.flat | 7.318732 |
| slope.down | 2.969730 |
| ca = number of major vessels (0-3) colored by flourosopy | 6.404980 |
| thal = Thallium Stress Test | |
| thal.fixed\_defect | 6.819749 |
| thal.reversable\_defect | 14.669412 |
| smoker | 5.385206 |
| drug = any prescribed cardiovascular drugs | |
| drug.Both | 4.269747 |
| drug.Clopidogrel | 4.643150 |
| drug.None | 4.110259 |
| fam\_hist = family history of heart disease | 5.110556 |

The MeanDecreaseGini column in the output provides a measure of how each feature contributes to the homogeneity of the nodes and leaves in the Random Forest model. Higher values indicate more important features (4). After aggregating the importance scores of all categories belonging to single features, the analysis indicates that ‘fbs’, ‘fam\_hist’ and ‘smoking’ are the least important variables in the model. Consequently, they could be considered for removal from the model. However, this decision should be carefully weighed against the domain knowledge and the potential of these features to influence the model in interaction with other variables.

b- Sampling optimisation:

K-crossfold validation is implemented to refine RF model training process. This involves to repeatedly dividing the dataset into ‘k’ groups, using each in turn for validation while training on the remaining groups.

The following code illustrates how I set-up the crossfold validation with a focus on sensitivity as the primary metric.

# Specifying the control using k-fold cross-validation

trainset$class <- factor(trainset$class, levels = c("1", "0"), labels = c("Positive", "Negative"))

control <- trainControl(method = "repeatedcv",

number = 10,

repeats = 3,

summaryFunction = twoClassSummary,

classProbs = TRUE,

savePredictions = "final")

# Random Forest training with k-fold cross-validation focused on sensitivity

rfModel <- train(class ~ ., data = trainset,

method = "rf",

trControl = control,

metric = "Sensitivity")

# Sensitivity analysis

testset$class <- factor(testset$class, levels = c("1", "0"), labels = c("Positive", "Negative"))

predictions2 <- predict(rfModel, newdata = testset)

confMatrix <- confusionMatrix(predictions2, testset$class)

sensitivity2 <- confMatrix$byClass['Sensitivity']

print(sensitivity2)

The sensitivity of the model increased from 82.2% to 88.9% following the optimization of the sampling strategy. This improvement indicates that the optimized model is performing better to accurately identify positive cases of heart disease.

1. Hyperparameter tuning:

The “mtry” hyperparameter, which determines the number of variables considered for splitting at each tree node, was tuned using grid search across a range of values (1 to 15). Adjusting this internal parameter can significantly impact model accuracy. A lower mtry increases tree diversity by using different subset of features, potentially reducing model’s variance, and improving accuracy on unseen data. Conversely, a higher mtry makes the trees more similar, which might increase model bias if too many irrelevant features are included. Finding the optimal mtry value is crucial to balancing bias and variance, thereby maximizing the model’s accuracy (3,5). The code below demonstrates how I conduct hyperparameter tuning:

# Specifying the control using k-fold cross-validation

control1 <- trainControl(method = "repeatedcv",

number = 10,

repeats = 3,

search = "grid",

summaryFunction = twoClassSummary,

classProbs = TRUE,

savePredictions = "final")

# Defining a tuning grid

tunegrid <- expand.grid(.mtry=c(1:15))

# Hyperparameter tuning with grid search on 'mtry'

rfTuned <- train(class ~ ., data = trainset,

method = "rf",

trControl = control1,

tuneGrid = tunegrid,

metric = "Sensitivity")

# Re-evaluation of sensitivity

predictions3 <- predict(rfTuned, newdata = testset)

confMatrix2 <- confusionMatrix(predictions3, testset$class)

sensitivity3 <- confMatrix2$byClass['Sensitivity']

print(sensitivity3)

After the hyperparameter tuning process, the model’s sensitivity further improved to 91.1%. This increase highlights the efficacy of optimizing the model’s internal parameters to better identify positive cases of heart disease.

The ‘ntree’ hyperparameter specifies the number of trees in the model. More trees will increase the model’s stability and accuracy up to a certain point, beyond which improvements may plateau and increase computational cost.

# Define a vector of ntree values you want to explore

ntree\_values <- seq(100, 1000, by=100)

# Initialize a vector to store the sensitivity for each ntree value

sensitivities <- numeric(length(ntree\_values))

# Loop over ntree values, train a model for each, and evaluate its performance

for (i in seq\_along(ntree\_values)) {

# Train the model using the current ntree value

rfModel4 <- randomForest(class ~ ., data = trainset, ntree = ntree\_values[i])

# Predict on the test set

predictions4 <- predict(rfModel4, newdata = testset)

# Compute the confusion matrix and extract the sensitivity

confMatrix <- confusionMatrix(predictions4, testset$class)

sensitivities[i] <- confMatrix$byClass['Sensitivity']

}

# Combine the ntree values and their corresponding sensitivities

results <- data.frame(ntree = ntree\_values, Sensitivity = sensitivities)

# Print the results

print(results)

|  |  |
| --- | --- |
| ntree | Sensitivity |
| 100 | 93% |
| 200 | 91% |
| 300 | 91% |
| 400 | 91% |
| 500 | 95% |
| 600 | 93% |
| 700 | 93% |
| 800 | 93% |
| 900 | 93% |
| 1000 | 93% |

This analysis indicates that setting ntree to 500 optimizes the model’s ability to accurately identify cases with heart disease, achieving 95% sensitivity.

1. To train a good classifier, especially non-parametric ones like Random Forest, training should meet the following requirements:

- Adequate data volume relative to the number of features (dimensionality) is essential to avoid overfitting. Without enough data, the model might not capture the actual patterns and instead focus on the noise, leading to a poor performance on new data.

- The training set should include diverse examples covering as many different scenarios as possible. It enables the classifier to accurately identify the boundaries separating different classes. Ultimately, a diverse and well distributed dataset ensures that the model can generalize to unseen data and enhances its performance.

- Features included in the data set should be relevant to the problem at hand. The presence of noise can significantly impact classifier performance (6).

Implementing classifiers in healthcare settings can improve data collection processes. By analyzing classifier’s performance, we can reveal underrepresented patient groups or conditions in the data. Additionally, classifiers can help identify features that most effectively predict health outcomes. These insights can guide healthcare providers towards the most valuable data to collect and monitor.

References:

1. Kirasich K, Smith T, Sadler B. Random Forest vs Logistic Regression: Binary Classification for Heterogeneous Datasets. 2018;1(3).

2. Couronné R, Probst P, Boulesteix AL. Random forest versus logistic regression: a large-scale benchmark experiment. BMC Bioinformatics. 2018 Dec;19(1):270.

3. Probst P, Wright MN, Boulesteix AL. Hyperparameters and tuning strategies for random forest. WIREs Data Mining and Knowledge Discovery. 2019;9(3):e1301.

4. The SIESTA (SEAAV Integrated evaluation sedation tool for anaesthesia) project: Initial development of a multifactorial sedation assessment tool for dogs. PLOS ONE. 2020 Apr 1;15(4):e0230799.

5. Bernard S, Heutte L, Adam S. Influence of Hyperparameters on Random Forest Accuracy. In: Benediktsson JA, Kittler J, Roli F, editors. Multiple Classifier Systems. Berlin, Heidelberg: Springer; 2009. p. 171–80. (Lecture Notes in Computer Science).

6. Van Der Walt CM, Barnard E. Data characteristics that determine classifier performance. SAIEE Afr Res J. 2007 Sep;98(3):87–93.