Bagging

Bagging, which stands for Bootstrap Aggregating, is an ensemble method which consists in generating m bootstrap samples from the training set and training a model on each of them. The final prediction is obtained by leveraging a voting mechanism to combine the predictions of the m models in one final prediction. It has been prooved that bagging provides stability to learning algorithms and reduces variance, therefore we will apply it to our case study. We are going to use both Soft Voting and Hard Voting to combine the predictions of the m models and compare their performance.

Load the train data set on which ROSE was applied

```
train_rose <- read.csv("data/train_rose.csv")
test <- read.csv("data/test.csv")

SEED <- 42
set.seed(SEED)
```

Perform the train validation split with 80% of the data for training and 20% for validation and set some parameters for the bootstrap samples.

```
train_index = createDataPartition(
  train_rose$CARAVAN,
  p = 0.8,
  list = FALSE,
  times = 1
)

train = train_rose[train_index, ]
val = train_rose[-train_index, ]
```

Generate m bootstrap samples from the training set.

```
m = 20
perc = 0.8
n = round(nrow(train) * perc, 0)
bootstrap_samples <- list()
for (i in 1:m) {
  bootstrap_samples[[i]] <- train[sample(1:n, n, replace = TRUE), ]
}
```

```
balanced_accuracy <- function(true_labels, predicted_labels) {
  # Ensure that the inputs are factors and have the same levels
  true_labels <- factor(true_labels)
  predicted_labels <- factor(predicted_labels, levels = levels(true_labels))

  # Calculate confusion matrix
  cm <- confusionMatrix(predicted_labels, true_labels)

  # Calculate True Positive Rate (TPR) and True Negative Rate (TNR)
  TPR <- cm$byClass['Sensitivity']
  TNR <- cm$byClass['Specificity']

  # Calculate balanced accuracy
  balanced_accuracy <- (TPR+TNR)/2

  return(balanced_accuracy)
}
```

Hard Voting

Compute the predictions of the m models on the validation and test sets and combine them using hard voting.

```
y_pred_val <- matrix(0, nrow = nrow(val), ncol = m)
y_pred_test <- matrix(0, nrow = nrow(test), ncol = m)

for(i in 1:m) {
  model <- glm(formula = CARAVAN ~ ., data = bootstrap_samples[[i]], family = "binomial")
  y_pred_val[,i] <- predict(model, newdata = val)
  y_pred_test[,i] <- predict(model, newdata = test)
}

y_pred_val <- ifelse(rowMeans(y_pred_val) > 0.5, 1, 0)
y_pred_test <- ifelse(rowMeans(y_pred_test) > 0.5, 1, 0)
```

Compute performance metrics.

```
acc_val_hard <- mean(y_pred_val == val$CARAVAN)
acc_test_hard <- mean(y_pred_test == test$CARAVAN)
acc_val_hard <- round(acc_val_hard*100, 2)
acc_test_hard <- round(acc_test_hard*100, 2)

bacc_val_hard <- balanced_accuracy(val$CARAVAN, y_pred_val)
bacc_test_hard <- balanced_accuracy(test$CARAVAN, y_pred_test)
bacc_val_hard <- round(bacc_val_hard*100, 2)
bacc_test_hard <- round(bacc_test_hard*100, 2)

f1_val_hard <- F1_Score(val$CARAVAN, y_pred_val)
f1_test_hard <- F1_Score(test$CARAVAN, y_pred_test)
f1_val_hard <- round(f1_val_hard*100, 2)
f1_test_hard <- round(f1_test_hard*100, 2)
```

Soft Voting

Compute the predictions of the m models on the validation and test sets and combine them using soft voting.

```
y_pred_val <- matrix(0, nrow = nrow(val), ncol = m)
y_pred_test <- matrix(0, nrow = nrow(test), ncol = m)

for(i in 1:m) {
  model <- glm(formula = CARAVAN ~ ., data = bootstrap_samples[[i]], family = "binomial")
  y_pred_val[,i] <- predict(model, newdata = val, type = "response")
  y_pred_test[,i] <- predict(model, newdata = test, type = "response")
}

y_pred_val <- ifelse(rowMeans(y_pred_val) > 0.5, 1, 0)
y_pred_test <- ifelse(rowMeans(y_pred_test) > 0.5, 1, 0)
```

Compute performance metrics.

```
acc_val_soft <- mean(y_pred_val == val$CARAVAN)
acc_test_soft <- mean(y_pred_test == test$CARAVAN)
acc_val_soft <- round(acc_val_soft*100, 2)
acc_test_soft <- round(acc_test_soft*100, 2)

bacc_val_soft <- balanced_accuracy(val$CARAVAN, y_pred_val)
```

```r
bacc_test_soft <- balanced_accuracy(test$CARAVAN, y_pred_test)
bacc_val_soft <- round(bacc_val_soft*100, 2)
bacc_test_soft <- round(bacc_test_soft*100, 2)

f1_val_soft <- F1_Score(val$CARAVAN, y_pred_val)
f1_test_soft <- F1_Score(test$CARAVAN, y_pred_test)
f1_val_soft <- round(f1_val_soft*100, 2)
f1_test_soft <- round(f1_test_soft*100, 2)
```

Comparison

Hereafter we compare the performance of the two voting mechanisms.

```r
comparison <- data.frame(
  "Metric" = c("Accuracy", "Balanced Accuracy", "F1 Score"),
  "Hard - Validation" = c(acc_val_hard, bacc_val_hard, f1_val_hard),
  "Soft - Validation" = c(acc_val_soft, bacc_val_soft, f1_val_soft),
  "Hard - Test" = c(acc_test_hard, bacc_test_hard, f1_test_hard),
  "Soft - Test" = c(acc_test_soft, bacc_test_soft, f1_test_soft)
)
kable(comparison, format = "markdown", caption = "Comparison of the performance of the two voting mecha
```

Table 1: Comparison of the performance of the two voting mechanisms

Metric	Hard... Validation	Soft... Validation	Hard... Test	Soft... Test
Accuracy	64.08	69.06	88.83	80.70
Balanced Accuracy	64.08	69.06	61.19	63.96
F1 Score	71.71	72.84	93.97	88.99

Conclusion

As we can see the Soft voting mechanism outperforms the Hard voting mechanism in every metric on the validation set. In the test set, Soft voting is better only in the balanced accuracy, however the accuracy is not a reliable metrics for the test set since it is imbalanced. However, Hard voting performs better in the F1 score.