Analysis-of-edibility-of-mushroom-on-basis-of-regression-models-in-R

2023-05-25

The dataset contains information about various attributes of mushrooms, including their edibility and assigned the target variable, "Edible," to the variable y. Goal is to find the best model for predicting ediblity based on all or any subset of the remaining attributes.To ensure proper modeling, converted the target variable y into a factor using the as.factor() function. Additionally, recorded the "Edible" variable into binary values by transforming the values "poisonous" to 0 and "edible" to 1 using the ifelse() function.

To maintain reproducibility and ensure consistent results, set a random seed of 123 using the set.seed() function.

```
mushrooms <- read.csv("C:/Users/saheli/Downloads/mushrooms.csv")
y <- mushrooms$Edible
y <- as.factor(y)
mushrooms$Edible <- ifelse(mushrooms$Edible == "poisonous", 0, 1)
set.seed(123)  # Set a seed for reproducibility
```

```
num_folds <- 5
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.2.3
```

```
## Loading required package: ggplot2
```

```
## Warning: package 'ggplot2' was built under R version 4.2.2
```

```
## Loading required package: lattice
```

```
# Create a cross-validation object
cv <- createFolds(y, k = num_folds, list = TRUE, returnTrain = FALSE)
```

To evaluate the performance of our models and assess their generalization ability, cross-validation is employed and set the number of folds for cross-validation to 5, indicating that the dataset was divided into five subsets or "folds."

To create the cross-validation object, utilized the createFolds() function from the caret package. This function generated a list of indices representing the partitioning of the data into the specified number of folds. The k parameter was set to 5, indicating the number of folds, and slo set list = TRUE to obtain a list of fold indices. Furthermore, by setting return Train = FALSE, it is ensured that the fold indices corresponded to the validation sets rather than the training sets.

```
set.seed(123)
#Convert categorical variables to dummy variables
X <- model.matrix(~ . - 1, data = mushrooms[, c("CapShape", "CapSurface", "CapColor", "Odor",
"Height")])

# Ensure proper data type
X <- as.matrix(X)

split <- createDataPartition(y, p = 0.6, list = FALSE)
```

```
# Split the data into training and testing sets
train_X <- X[split, ]
train_y <- y[split]
test_X <- X[-split, ]
test_y <- y[-split]
total_instances <- nrow(mushrooms)
training_instances <- nrow(train_X)
testing_instances <- nrow(test_X)

# Print the number of instances
cat("Total Instances:", total_instances, "\n")
```

```
## Total Instances: 8124
```

```
cat("Training Instances:", training_instances, "\n")
```

```
## Training Instances: 4875
```

```
cat("Testing Instances:", testing_instances, "\n")
```

```
## Testing Instances: 3249
```

Converted the categorical variables into dummy variables using the model.matrix() function. Formula ~ . - 1 to indicate that all variables except the intercept should be used.

To assess the performance of models, splitted the data into training and testing sets using createDataPartition() function from the caret package to randomly partition the data while preserving the class proportions. In this case, it is allocated 60% of the instances to the training set and the remaining 40% to the testing set. The indices generated by createDataPartition() were then used to split both the input features (X) and the target variable (y).

```
library(glmnet)
```

```
## Warning: package 'glmnet' was built under R version 4.2.3

## Loading required package: Matrix

## Warning: package 'Matrix' was built under R version 4.2.2

## Loaded glmnet 4.1-7
```

```
set.seed(123)
# Fit the logistic regression model with cross-validation
logistic_model <- cv.glmnet(x = train_X, y = train_y, family = "binomial")

# Tune the hyperparameters using cross-validation and grid search
cv_logistic <- cv.glmnet(x = train_X, y = train_y, family = "binomial", nfolds = num_folds)

# Get the optimal lambda value
best_lambda <- cv_logistic$lambda.min
# Fit the logistic regression model with the best lambda value
logistic_model <- glmnet(x = train_X, y = train_y, family = "binomial", lambda = best_lambda)
```

Logistic regression model is fitted with cross-validation using the cv.glmnet() function. This function applies the L1-regularization (LASSO) technique to the logistic regression model. By using cross-validation, it is aimed to estimate the model's performance on unseen data and prevent overfitting. Training features (train_X) and the corresponding target variable (train_y) are given as inputs to the function, specifying the family as "binomial" to indicate logistic regression.

Next, performed cross-validation and grid search to tune the hyperparameters. The function calculated the mean cross-validated error for different lambda values, which control the level of regularization. The optimal lambda value, denoted as lambda.min, was determined based on the minimum mean cross-validated error.

Finally, fitted the logistic regression model with the best lambda value using the glmnet() function. The function takes the training features (train_X), target variable (train_y), family as "binomial," and the selected lambda value as inputs. This resulted in the final logistic regression model that was optimized with respect to the lambda hyperparameter.

```
##########DECISION TREE##########
set.seed(123)
library(rpart)
```

```
## Warning: package 'rpart' was built under R version 4.2.3
```

```
# Define the decision tree model
tree_model <- rpart(formula = train_y ~ ., data = data.frame(train_X, train_y), method = "class")

# Tune the hyperparameters using cross-validation and grid search
cv_tree <- rpart.control(cp = 0.01)  # Adjust the complexity parameter as needed

# Fit the decision tree model
tree_model <- rpart(formula = train_y ~ ., data = data.frame(train_X, train_y),
                    method = "class", control = cv_tree)
```

Using the rpart() function, train_y ~ ., indicating that the target variable (train_y) should be predicted based on all the other features in the training set (train_X). The data was provided as a data frame with the target variable included. The method argument was set to "class" to indicate that the decision tree should be built for classification purposes.

Next, tuned the hyperparameters of the decision tree model using cross-validation and grid search and utilized the rpart.control() function to define the cross-validation parameters. In this case,complexity parameter (cp) to 0.01. The complexity parameter controls the level of tree complexity and helps to prevent overfitting.

Finally, fitted the decision tree model using the rpart() function once again.

```
#####Random Forests#######
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 4.2.3
```

```
## randomForest 4.7-1.1
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
set.seed(123)
# Define the range of mtry values to test
mtry_values <- seq(1, ncol(train_X), by = 1)

# Initialize variables to store results
best_mtry <- NULL
best_oob_error <- Inf
```

```r
# Iterate over different mtry values
for (mtry in mtry_values) {
  # Fit the random forest model
  rf_model <- randomForest(x = train_X, y = train_y, ntree = 100, mtry = mtry)

  # Get the out-of-bag (OOB) error rate
  oob_error <- rf_model$err.rate[nrow(rf_model$err.rate), "OOB"]

  # Check if the current model has a lower OOB error rate
  if (oob_error < best_oob_error) {
    best_mtry <- mtry
    best_oob_error <- oob_error
  }
}

# Fit the random forest model with the best mtry value
rf_model <- randomForest(x = train_X, y = train_y, ntree = 100, mtry = best_mtry)
```

The mtry parameter determines the number of randomly selected features considered at each split in the random forest model. In our case, used seq(1, ncol(train_X), by = 1) to iterate over all possible values of mtry, ranging from 1 to the total number of features in the training set (train_X) and initialized variables to store the best mtry value and the corresponding out-of-bag (OOB) error. The OOB error is an estimate of the model's performance on unseen data.

Then performed a loop over different mtry values. For each iteration, fitted a random forest model using the randomForest() function and provided the training features (train_X) and the target variable (train_y) as inputs, specifying ntree = 100 to construct 100 decision trees within the random forest ensemble. The mtry parameter was set to the current value of mtry in each iteration and calculated the OOB error rate using rf_model$err.rate[nrow(rf_m odel$err.rate), “OOB”]. This provided us with the OOB error rate for the current random forest model and checked if the current model had a lower OOB error rate compared to the best observed so far. If it did, we updated the best_mtry and best_oob_error variables accordingly.

After iterating through all mtry values, obtained the best mtry value and its corresponding OOB error which is then fitted to the final random forest model using the randomForest() function, specifying the best mtry value.

```r
set.seed(123)
# Define the count_correct_classifications() function
count_correct_classifications <- function(predictions, actual) {
  sum(predictions == actual)
}

# Perform cross-validation and evaluate model performance
for (fold in 1:num_folds) {

  # Logistic Regression
  logistic_predictions <- predict(logistic_model, newx = test_X, type = "class")
  correct_logistic <- count_correct_classifications(logistic_predictions, test_y)

  # Decision Trees
  tree_predictions <- predict(tree_model, newdata = data.frame(test_X), type = "class")
  correct_tree <- count_correct_classifications(tree_predictions, test_y)

  # Random Forests
  rf_predictions <- predict(rf_model, newdata = test_X, type = "class")
  correct_rf <- count_correct_classifications(rf_predictions, test_y)
}
```

Performed cross-validation to evaluate the performance of the logistic regression, decision tree, and random forest models and defined a helper function, count_correct_classifications(), to count the number of correctly classified instances given the predicted values and the actual values.

For each fold in the cross-validation process, evaluated the models using the testing data (test_X) and the corresponding target variable (test_y).

For logistic regression, used the predict() function with logistic_model as the model object and newx = test_X to obtain the predicted class labels. These labels were then compared to the actual class labels (test_y) using the count_correct_classifications() function, resulting in the count of correctly classified instances for logistic regression (correct_logistic).

Similarly, for decision trees, used the predict() function with tree_model as the model object and newdata = data.frame(test_X) to obtain the predicted class labels. The count of correctly classified instances for decision trees was computed using the count_correct_classifications() function and stored in correct_tree.

For random forests, used the predict() function with rf_model as the model object and newdata = test_X to obtain the predicted class labels. The count of correctly classified instances for random forests was computed using the count_correct_classifications() function and stored in correct_rf.

By evaluating the models on the testing data for each fold, obtained the counts of correctly classified instances for logistic regression, decision trees, and random forests. These counts serve as performance metrics to compare the predictive performance of the different models.

```
set.seed(123)
print(paste("Logistic Regression - Correctly Classified:", correct_logistic))
```

```
## [1] "Logistic Regression - Correctly Classified: 3211"
```

```
# Confusion matrix
confusion_matrix <- table(Actual = test_y, Predicted = logistic_predictions)
print(confusion_matrix )
```

```
##             Predicted
## Actual       Edible Poisonous
##   Edible       1683         0
##   Poisonous      38      1528
```

```
# Accuracy
log_accuracy <- sum(diag(confusion_matrix))/sum(confusion_matrix)
print(paste("For logistic regression model, accuracy on test set:", log_accuracy))
```

```
## [1] "For logistic regression model, accuracy on test set: 0.988304093567251"
```

```
# Sensitivity (True Positive Rate or Recall)
sensitivity <- confusion_matrix[2, 2] / sum(confusion_matrix[2, ])
print(paste("For logistic regression model , sensitivity on test set:", sensitivity))
```

```
## [1] "For logistic regression model , sensitivity on test set: 0.9757343550447"
```

```
# Specificity (True Negative Rate)
specificity <- confusion_matrix[1, 1] / sum(confusion_matrix[1, ])
print(paste("For logistic regression model , specificity on test set:", specificity))
```

```
## [1] "For logistic regression model , specificity on test set: 1"
```

```r
# Precision (Positive Predictive Value)
precision <- confusion_matrix[2, 2] / sum(confusion_matrix[, 2])
print(paste("For logistic regression model , precision on test set:", precision))
```

```
## [1] "For logistic regression model , precision on test set: 1"
```

```r
# F1 score
f1_score <- 2 * (precision * sensitivity) / (precision + sensitivity)
print(paste("For logistic regression model , F1 Score on test set:", f1_score))
```

```
## [1] "For logistic regression model , F1 Score on test set: 0.987718164188752"
```

```r
set.seed(123)
print(paste("Decision Tree - Correctly Classified for test set:", correct_tree))
```

```
## [1] "Decision Tree - Correctly Classified for test set: 3142"
```

```r
# Confusion matrix
confusion_matrix <- table(Actual = test_y, Predicted = tree_predictions)
print(confusion_matrix)
```

```
##            Predicted
## Actual      Edible Poisonous
##   Edible      1636        47
##   Poisonous     60      1506
```

```r
# Accuracy
descision_accuracy <- sum(diag(confusion_matrix))/sum(confusion_matrix)
print(paste("For decison tree, accuracy on test set:", descision_accuracy))
```

```
## [1] "For decison tree, accuracy on test set: 0.967066789781471"
```

```r
# Sensitivity (True Positive Rate or Recall)
sensitivity <- confusion_matrix[2, 2] / sum(confusion_matrix[2, ])
print(paste("For decison tree, sensitivity on test set:", sensitivity))
```

```
## [1] "For decison tree, sensitivity on test set: 0.961685823754789"
```

```r
# Specificity (True Negative Rate)
specificity <- confusion_matrix[1, 1] / sum(confusion_matrix[1, ])
print(paste("For decison tree , specificity on test set:", specificity))
```

```
## [1] "For decison tree , specificity on test set: 0.972073677956031"
```

```r
# Precision (Positive Predictive Value)
precision <- confusion_matrix[2, 2] / sum(confusion_matrix[, 2])
print(paste("For decison tree , precision on test set:", precision))
```

```
## [1] "For decison tree , precision on test set: 0.96973599484868"
```

```r
# F1 score
f1_score <- 2 * (precision * sensitivity) / (precision + sensitivity)
print(paste("For decison tree, F1 Score on test set:", f1_score))
```

```
## [1] "For decison tree, F1 Score on test set: 0.965694132734851"
```

```r
set.seed(123)
print(paste("Random Forest - Correctly Classified for test set:", correct_rf))
```

```
## [1] "Random Forest - Correctly Classified for test set: 3221"
```

```r
# Confusion matrix
confusion_matrix <- table(Actual = test_y, Predicted = rf_predictions)
print(confusion_matrix)
```

```
##            Predicted
## Actual      Edible Poisonous
##   Edible      1680         3
##   Poisonous     25      1541
```

```r
# Accuracy
rf_accuracy <- sum(diag(confusion_matrix))/sum(confusion_matrix)
print(paste("For random forest model , accuracy on test set:",rf_accuracy))
```

```
## [1] "For random forest model , accuracy on test set: 0.991381963681133"
```

```r
# Sensitivity (True Positive Rate or Recall)
sensitivity <- confusion_matrix[2, 2] / sum(confusion_matrix[2, ])
print(paste("For random forest model , sensitivity on test set:", sensitivity))
```

```
## [1] "For random forest model , sensitivity on test set: 0.984035759897829"
```

```r
# Specificity (True Negative Rate)
specificity <- confusion_matrix[1, 1] / sum(confusion_matrix[1, ])
print(paste("For random forest model , specificity on test set:", specificity))
```

```
## [1] "For random forest model , specificity on test set: 0.998217468805704"
```

```r
# Precision (Positive Predictive Value)
precision <- confusion_matrix[2, 2] / sum(confusion_matrix[, 2])
print(paste("For random forest model , precision on test set:", precision))
```

```
## [1] "For random forest model , precision on test set: 0.998056994818653"
```

```r
# F1 score
f1_score <- 2 * (precision * sensitivity) / (precision + sensitivity)
print(paste("For random forest model , F1 Score on test set:", f1_score))
```

```
## [1] "For random forest model , F1 Score on test set: 0.990996784565916"
```

```r
set.seed(123)
# Compare the number of correctly classified mushrooms
best_model <- "Logistic Regression"
max_correct_classifications <- correct_logistic

if (correct_tree > max_correct_classifications) {
  best_model <- "Decision Tree"
  max_correct_classifications <- correct_tree
}

if (correct_rf > max_correct_classifications) {
  best_model <- "Random Forest"
  max_correct_classifications <- correct_rf
}

cat("Best Model:", best_model, "\n")
```

```
## Best Model: Random Forest
```

```r
cat("Number of Correctly Classified Mushrooms:", max_correct_classifications, "\n")
```

```
## Number of Correctly Classified Mushrooms: 3221
```

Evaluated the predictive performance of logistic regression, decision trees, and random forests models for classifying the edibility of mushrooms. After conducting cross-validation and comparing the results, we found that the random forests model exhibited the best performance.

Among the three models tested, the random forests model achieved the highest number of correctly classified mushrooms. Specifically, it correctly classified 3,221 instances out of the total mushrooms in our testing set. This indicates that the random forests model was effective in accurately predicting the edibility of mushrooms based on the given attributes. Also the different classification metrics(accuracy, sensitivity, f1 score etc) are better compared to other two models.

These findings suggest that the random forests model is a promising approach and best model for predicting the edibility of mushrooms based on the given attributes.

It should be noted that the reported results are based on this specific dataset and evaluation methodology.

```r
logistic_accuracy <- numeric(num_folds)
tree_accuracy <- numeric(num_folds)
rf_accuracy <- numeric(num_folds)
# Perform cross-validation and evaluate model performance
for (fold in 1:num_folds) {
# Logistic Regression
logistic_predictions <- predict(logistic_model, newx = test_X, type = "class")
logistic_accuracy[fold] <- sum(logistic_predictions == test_y) / length(test_y)
# Decision Trees
tree_predictions <- predict(tree_model, newdata = data.frame(test_X), type = "class")
tree_accuracy[fold] <- sum(tree_predictions == test_y) / length(test_y)
# Random Forests
rf_predictions <- predict(rf_model, newdata = test_X, type = "class")
rf_accuracy[fold] <- sum(rf_predictions == test_y) / length(test_y)
}
# Create the model matrix using the accuracy values
model_matrix <- cbind(logistic_accuracy, tree_accuracy, rf_accuracy)
# Perform the Friedman test
friedman_result <- friedman.test(model_matrix)
# Print the test result
print(friedman_result)
```

```
##
##  Friedman rank sum test
##
## data:  model_matrix
## Friedman chi-squared = 10, df = 2, p-value = 0.006738
```

The Friedman rank sum test is a non-parametric statistical test used to determine if there are any differences between multiple related groups. In this case, the test is applied to the model_matrix that contains the accuracy values of three models (logistic regression, decision tree, and random forest) across multiple folds.

The test result indicates the following:

Friedman chi-squared: The calculated chi-squared test statistic is 10.

Degrees of freedom (df): The degrees of freedom associated with the chi-squared distribution are 2.

p-value: The p-value associated with the test statistic is 0.006738.

```r
# Perform t-tests

t_test <- t.test(model_matrix)

# Print the t-test results

print(t_test)
```

```
##
##  One Sample t-test
##
## data:  model_matrix
## t = 339.98, df = 14, p-value < 2.2e-16
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  0.9760544 0.9884475
## sample estimates:
## mean of x
## 0.9822509
```

The t-test results indicate that the mean accuracy across the models is significantly different from 0 (p-value < 2.2e-16). The 95% confidence interval for the mean accuracy ranges from 0.9760544 to 0.9884475. The estimated mean accuracy is 0.9822509.

The interpretation of the test result depends on the significance level (alpha) chosen for the test. If the chosen alpha level is 0.05, for example, for both the tests p-value is less than alpha, indicating statistical significance.