Classification Models Wholesale Customer Analysis

Johnny Lee

2025-04-01

Libraries And Seed

```r
library(caret)
library(tidyverse)
library(yardstick)
set.seed(7890682)
options(scipen = 999)
```

Functions

```r
# Get validation scores on test data
validate = function(test, testZ, models) {

  results = data.frame(Metrics = c("Accuracy",
                                   "Recall",
                                   "Precision",
                                   "F",
                                   "ROC_AUC",
                                   "PR_AUC"))

  for (i in 1:length(models)) {

    if (names(models)[i] == "Random Forest") {
      test_data = test %>%
        mutate(preds = predict(models[[i]], test, type = "prob")[, "Horeca"],
          class_preds = predict(models[[i]], test, type = "raw"))

    } else {
      test_data = test %>%
        mutate(preds = predict(models[[i]], testZ, type = "prob")[, "Horeca"],
          class_preds = predict(models[[i]], testZ, type = "raw"))
    }


    metrics = metric_set(accuracy, recall, precision, f_meas, roc_auc, pr_auc)

    curr_test = metrics(test_data,
                        truth = Channel,
```

```r
                      estimate = class_preds,
                      preds,
                      event_level = "first")

    curr_cv = models[[i]]$results %>%
      filter(ROC == max(ROC)) %>%
      select(Accuracy, Recall, Precision, F, ROC, AUC) %>%
      pivot_longer(cols = everything(),
                   names_to = "Metric",
                   values_to = "Value")

    results = results %>%
    mutate(
      !!paste(names(models)[i], "Test") := curr_test$.estimate,
      !!paste(names(models)[i], "CV") := curr_cv$Value
    )
  }

  return (results)
}

# Z-score scale continuous variables
scaleZ = function(train, test) {
  scaled_train = train[,3:8]
  scaled_test = test[,3:8]
  for (i in 3:8) {
    mu_train = mean(train[,i])
    sd_train = sd(train[,i])
    scaled_train[,i-2] = (train[,i] - mu_train)/sd_train
    scaled_test[,i-2] = (test[,i] - mu_train)/sd_train
  }
  scaled_train = cbind(train[,1:2], scaled_train)
  scaled_test = cbind(test[,1:2], scaled_test)
  return (list(scaled_test = scaled_test, scaled_train = scaled_train))
}
```

Preparing Data

```r
data = read.csv("Wholesale_customers_data.csv")

# Rename channel levels
data$Channel = factor(
  data$Channel,
  levels = c(1, 2),
  labels = c("Horeca", "Retail")
)

data$Region = factor(
  data$Region,
  levels = c(1, 2, 3),
  labels = c("Lisbon", "Oporto", "Other")
)
```

```r
data = data %>%
  rename(
    'Delicatessen' = Delicassen
  )


# Split test and train data 30/70 split
trainRows = sample(nrow(data), 0.7*nrow(data))
train = data[trainRows, ]
test = data[-trainRows, ]

# Z-score scaled training and testing data
scaledData = scaleZ(train, test)
trainZ = scaledData$scaled_train
testZ = scaledData$scaled_test

# Custom function to calculate multiple metrics
customSummary = function(data, lev, model) {
  # ROC metrics
  roc_stats = twoClassSummary(data, lev, model)
  # Accuracy
  acc_stats = defaultSummary(data, lev, model)
  # Precision/Recall/F1
  f1_stats = prSummary(data, lev, model)

  c(roc_stats, acc_stats, f1_stats)
}
```

Logistic Regression Model

```r
# Cross validation parameters
ctrl = trainControl(
  method = "cv",
  number = 10,
  summaryFunction = customSummary,
  classProbs = TRUE,
  savePredictions = "final",
  sampling = "up",
)

# Train model
log_model = train(
  Channel ~ .,
  data = trainZ,
  method = "glm",
  family = "binomial",
  trControl = ctrl,
  metric = "ROC",
)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

KNN Model

```r
# Tune grid for optimal hyper-parameters
tuneGrid = expand.grid(k = c(3,5,7,9,11,13,15,17))

# Cross validation parameters
ctrl = trainControl(
  method = "cv",
  number = 10,
  summaryFunction = customSummary,
  classProbs = TRUE,
  savePredictions = "final",
  sampling = "up"
)

# Train model
knn_model = train(
  Channel ~ .,
  data = trainZ,
  method = "knn",
  trControl = ctrl,
  metric = "ROC",
  tuneGrid = tuneGrid
)
```

Random Forest Model

```r
# Tune grid for optimal hyper-parameters
tuneGrid = expand.grid(
  mtry = c(2, 3, 4, 5),
  splitrule = "gini",
  min.node.size = c(1,3,5,10)
)

# Cross validation parameters
ctrl = trainControl(
  method = "cv",
  number = 10,
  summaryFunction = customSummary,
  classProbs = TRUE,
```

```
  savePredictions = "final",
  sampling = "up",
)

# Number of trees vector
num_tree_vec = c(100,200,300,400,500)
rf_model = NULL
best_AUC = -1
best_numTrees = -1

# Train model
for (ntree in num_tree_vec) {
  model = train(
    Channel ~ .,
    data = train,
    method = "ranger",
    trControl = ctrl,
    metric = "ROC",
    tuneGrid = tuneGrid,
    num.trees = ntree,
    importance = "impurity"
  )

  if (max(model$results$ROC) > best_AUC) {
    best_AUC = max(model$results$ROC)
    rf_model = model
    best_numTrees = ntree
  }
}

rf_model$bestNumTrees = best_numTrees
```

Validating Models

```
# List models
models_list = list(
  "Logistic" = log_model,
  "KNN" = knn_model,
  "Random Forest" = rf_model
)

# Validate
validation = validate(test, testZ, models_list) %>%
  mutate(across(where(is.numeric), ~ round(., 3)))

# Table
knitr::kable(
  validation,
  align = c("l", "r", "r", "r", "r",  "r", "r"),
  caption = "Model Performance Comparison",
)
```

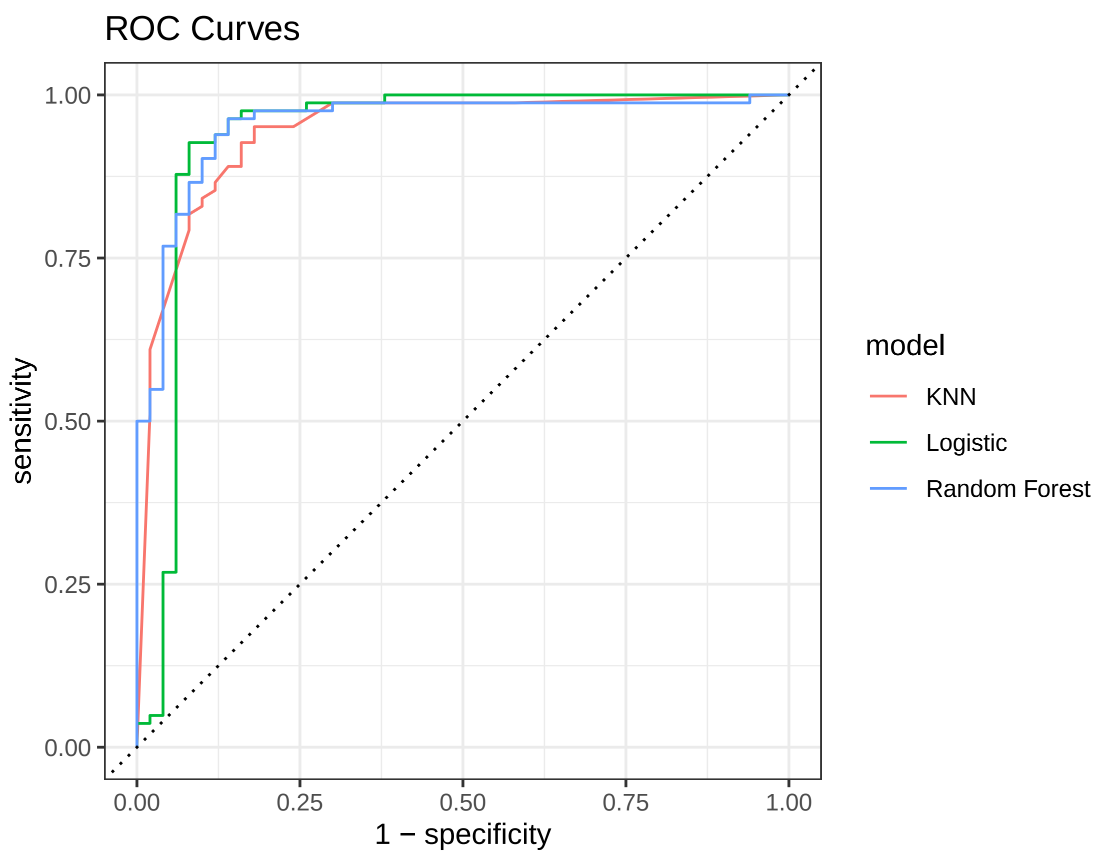
Table 1: Model Performance Comparison

Metrics	Logistic Test	Logistic CV	KNN Test	KNN CV	Random Forest Test	Random Forest CV
Accuracy	0.909	0.910	0.871	0.899	0.924	0.925
Recall	0.939	0.913	0.866	0.879	0.963	0.926
Precision	0.917	0.957	0.922	0.976	0.919	0.968
F	0.928	0.933	0.893	0.924	0.940	0.945
ROC_AUC	0.936	0.958	0.944	0.948	0.955	0.970
PR_AUC	0.913	0.930	0.963	0.464	0.971	0.907

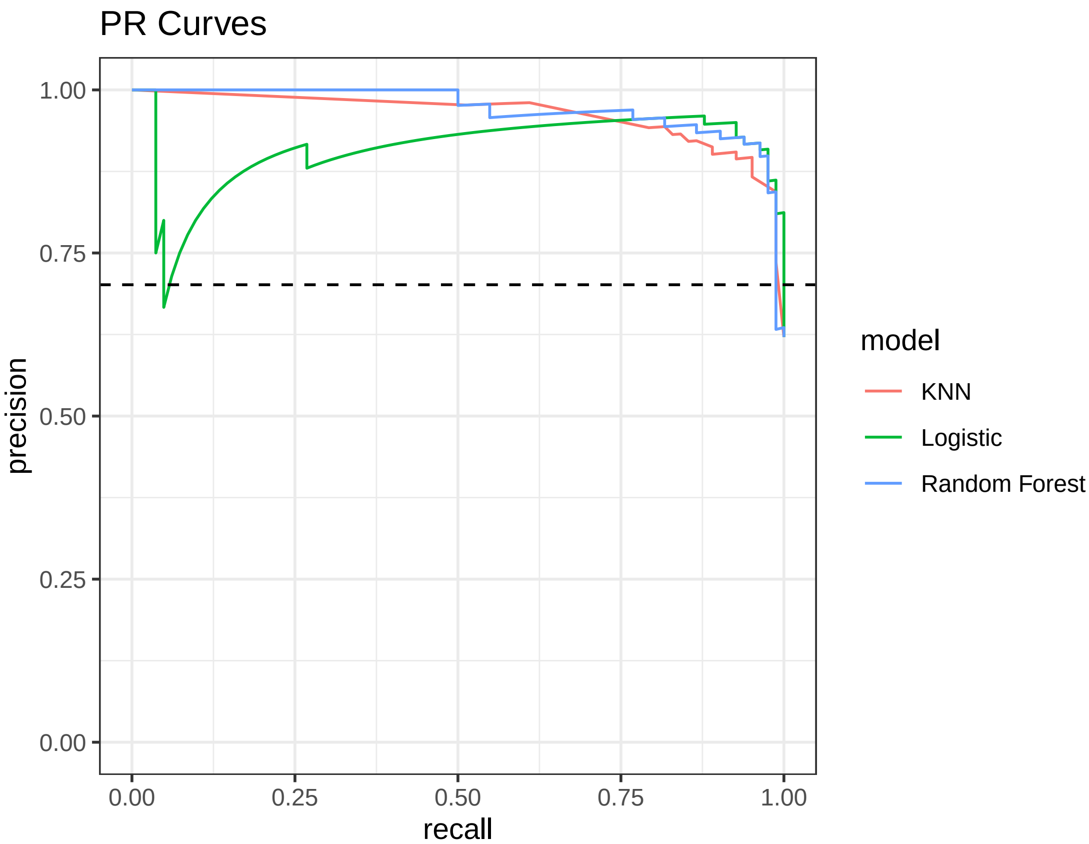
```r
# Saving Models
saveRDS(
  list(models_list = models_list, validate = validate, scaleZ = scaleZ),
  "class_models_and_functions.rds"
)
```

ROC and PR Curves

```r
# Create prediction data frame
data_pred = tibble()

for (i in 1:length(models_list)) {

  if (names(models_list)[i] == "Random Forest") {
    estimate = predict(models_list[[i]], test, type = "prob")[, "Horeca"]
  } else {
    estimate = predict(models_list[[i]], testZ, type = "prob")[, "Horeca"]
  }

  preds = tibble(
    truth = test$Channel,
    estimate = estimate,
    model = names(models_list)[i]
    )
  data_pred = bind_rows(data_pred, preds)
}

# ROC Curves
data_pred %>%
  group_by(model) %>%
  roc_curve(truth, estimate,
            event_level = "first") %>%
  autoplot() +
  labs(title = "ROC Curves")
```




```
# PR Curves
data_pred %>%
  group_by(model) %>%
  pr_curve(truth, estimate,
           event_level = "first") %>%
  autoplot() +
  labs(title = "PR Curves")  +
  geom_hline(yintercept = mean(train$Channel == "Horeca"),
             linetype = "dashed")
```



Summary

Best Performing Model: Random Forest (Test ROC-AUC = 0.948, CV ROC-AUC = 0.973)

Most Stable Model: Logistic Regression and Random Forest (ΔROC-AUC = 0.025 Test vs CV)

Worst Model: KNN (Test Accuracy = 0.864, CV PR-AUC = 0.430)

Most Overfit Model: KNN (ΔPR-AUC = 0.501 Test vs CV)

Final Model Selection

We select the Random Forest model due to its:

Best ROC-AUC performance (ability to distinguish between classes)

Robustness across metrics (High Recall, Precision, F, PR-AUC, and stability)

Handling complex interactions as it captures non-linear patterns

While Logistic Regression offers interpretability, Random Forest excels in our primary metric (ROC-AUC), which prioritizes overall classification performance over precision/recall for a specific class.