Advanced Methods for Regression and Classification

Exercise 11

Bosse Behrens , st.id: 12347333

First we load the data and convert y to a factor.

```r
data <- read.csv("bank.csv", header = TRUE, sep = ";", stringsAsFactors = FALSE)

data$y <- as.factor(data$y)
```

Now we do the stratified data splitting.

```r
library(dplyr)
```

```
##
## Attache Paket: 'dplyr'

## Die folgenden Objekte sind maskiert von 'package:stats':
##
##     filter, lag

## Die folgenden Objekte sind maskiert von 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
set.seed(12347333)

train_data <- data %>%
  group_by(y) %>%
  slice_sample(prop = 2/3) %>%
  ungroup()

test_data <- anti_join(data, train_data)
```

```
## Joining with `by = join_by(age, job, marital, education, default, balance,
## housing, loan, contact, day, month, duration, campaign, pdays, previous,
## poutcome, y)`
```

Task 1

part a)

```r
library(e1071)
```

We train the model with default parameters.

```r
model_default <- svm(y~., data=train_data, kernel="radial")
```

Now we predict the test data classes.

```r
test_predictions <- predict(model_default, newdata = test_data)
```

Now we compute the confusion matrix and balanced accuracy.

```r
conf_matrix <- table(Predicted = test_predictions, Actual = test_data$y)
print(conf_matrix)
```

```
##          Actual
## Predicted   no  yes
##       no  1325  158
##       yes    9   16
```

```r
tp <- diag(conf_matrix)
actual_totals <- colSums(conf_matrix)

sensitivity <- tp / actual_totals

balanced_accuracy <- mean(sensitivity)
print(paste("Balanced Accuracy:", balanced_accuracy))
```

```
## [1] "Balanced Accuracy: 0.542603698150925"
```

As we can see the balanced accuracy shows the model is performing very poorly with a BA of close to 0.5, which is akin to random guessing. The confusion matrix also shows that especially the predictions for the yes class are very inaccurate, while for the no class they are better. This is again most probably due to the imbalanced classes.

part b)

First we define grids for the two parameters.

```r
gamma_values <- c(0.1,1,10,100)
cost_values <- c(0.1,1,10,100)
```

Now we tune the parameters using tune.svm().

```r
tuned_model <- tune.svm(
  y ~ .,
  data = train_data,
  gamma = gamma_values,
  cost = cost_values
)
```

part c)

Retrieving the best model and predicting on the test data again.

```r
best_model <- tuned_model$best.model

test_predictions_best <- predict(best_model, newdata = test_data)
```

Now we again compute the confusion matrix and balanced accuracy.

```r
conf_matrix_best <- table(Predicted = test_predictions_best, Actual = test_data$y)
print("Confusion Matrix (Best Model):")
```

```
## [1] "Confusion Matrix (Best Model):"
```

```r
print(conf_matrix_best)
```

```
##          Actual
## Predicted   no  yes
##       no  1318  143
##       yes   16   31
```

```r
tp_best <- diag(conf_matrix_best)
actual_totals_best <- colSums(conf_matrix_best)

sensitivity_best <- tp_best / actual_totals_best

balanced_accuracy_best <- mean(sensitivity_best)
print(paste("Balanced Accuracy (Best Model):", balanced_accuracy_best))
```

```
## [1] "Balanced Accuracy (Best Model): 0.583083458270865"
```

As we can observe the performance stay almost similar, only under 10 observations more from the 1500 test data samples now get classified correctly in comparison to the tuned model from part a).

part d)

First we define the error function that takes 1-balanced accuracy as the criteria.

```r
custom_error <- function(true_labels, predicted_labels) {
  cm <- table(Predicted = predicted_labels, Actual = true_labels)
  tp <- diag(cm)
  actual_totals <- colSums(cm)
  sensitivity <- tp / actual_totals

  balanced_accuracy <- mean(sensitivity)

  return(1 - balanced_accuracy)
}
```

Now we define the grid for weights. We tested for multiple weights. Sicne the tune() fucntion only takes one list a time, we only show here the one that gave the best results in output.

```r
# "inverse"
# class_weights <- list("no" = 1, "yes" = 2)
#class_weights <- list("no" = 1, "yes" = 3)
class_weights <- list("no" = 1, "yes" = 5)
```

Now we tune the model using the grid of weights and the tune() function.

```r
tuned_model_weights <- tune(
  svm,
  y ~ .,
  data = train_data,
  class.weights = class_weights,
  tunecontrol = tune.control(error.fun = custom_error)
)
```

We extract the model with the best weights and predict on the test data again.

```r
best_model_weights <- tuned_model_weights$best.model

test_predictions <- predict(best_model_weights, newdata = test_data)
```

As a last step we again compute the confusion matrix and balanced accuracy for the model with tuned weights.

```r
conf_matrix <- table(Predicted = test_predictions, Actual = test_data$y)
print("Confusion Matrix:")
```

```
## [1] "Confusion Matrix:"
```

```r
print(conf_matrix)
```

```
##          Actual
## Predicted   no  yes
##       no  1169   46
##       yes  165  128
```

```r
tp <- diag(conf_matrix)
actual_totals <- colSums(conf_matrix)
sensitivity <- tp / actual_totals

balanced_accuracy <- mean(sensitivity)
print(paste("Balanced Accuracy:", balanced_accuracy))
```

```
## [1] "Balanced Accuracy: 0.805972013993004"
```

As we can see, the balanced accuracy has improved significantly to 0.8. Therefore the weight tuning was very effective in improving the model quality. Further improvement can possibly made by enlarging the grid for the class weights, e.g. trying to assign the minority class even more weight or fine tuning further to the exact weights, though this might be computationally too expensive and therefore unfeasible.