# Exercise 11

Rita Selimi

10/01/2024

### Load Required Libraries

```
library(e1071)
library(caret)
```

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

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

### Load and Preprocess Data

```
bank_data <- read.csv2("bank.csv")

# Exclude the variable 'duration'
bank_data <- bank_data[, !colnames(bank_data) %in% "duration"]

# Convert y to a factor
bank_data$y <- as.factor(bank_data$y)

# Check class distribution
table(bank_data$y)
```

```
##
##   no  yes
## 4000  521
```

```
# Set seed for reproducibility
set.seed(123)

# Stratified split for training and testing
data_split <- createDataPartition(bank_data$y, p = 0.67, list = FALSE)
train_data <- bank_data[data_split, ]
test_data <- bank_data[-data_split, ]
```

### (a) Apply SVM with Default Parameters

```r
# Train SVM with default parameters
svm_model <- svm(y ~ ., data = train_data, kernel = "radial")

# Predict on test data
predictions <- predict(svm_model, newdata = test_data)

# Confusion Matrix
conf_matrix <- confusionMatrix(predictions, test_data$y)
conf_matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   no  yes
##        no  1320  171
##        yes    0    0
##
##                Accuracy : 0.8853
##                  95% CI : (0.868, 0.9011)
##     No Information Rate : 0.8853
##     P-Value [Acc > NIR] : 0.5204
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 1.0000
##             Specificity : 0.0000
##          Pos Pred Value : 0.8853
##          Neg Pred Value :    NaN
##              Prevalence : 0.8853
##          Detection Rate : 0.8853
##    Detection Prevalence : 1.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : no
##
```

```r
# Calculate Balanced Accuracy
balanced_acc <- (conf_matrix$byClass["Sensitivity"] + conf_matrix$byClass["Specificity"])/2
balanced_acc
```

```
## Sensitivity
##         0.5
```

The confusion matrix shows that the SVM model predicted all test cases as “no” (not subscribing to a term deposit). It correctly identified 1,320 “no” cases but failed to predict any “yes” cases. This is likely due to the imbalance in the dataset, where “no” cases are much more frequent.

Key metrics: - **Accuracy**: 88.53%, which reflects the dominance of “no” cases. - **Sensitivity**: 1.0, meaning all “no” cases were correctly identified. - **Specificity**: 0.0, meaning no “yes” cases were identified. - **Balanced Accuracy**: 50%, indicating the model performs no better than random guessing for the imbalanced classes.

This outcome suggests the SVM model with default parameters struggles to handle the imbalanced dataset and heavily favors the majority class.

### (b) Tune Parameters Gamma and Cost

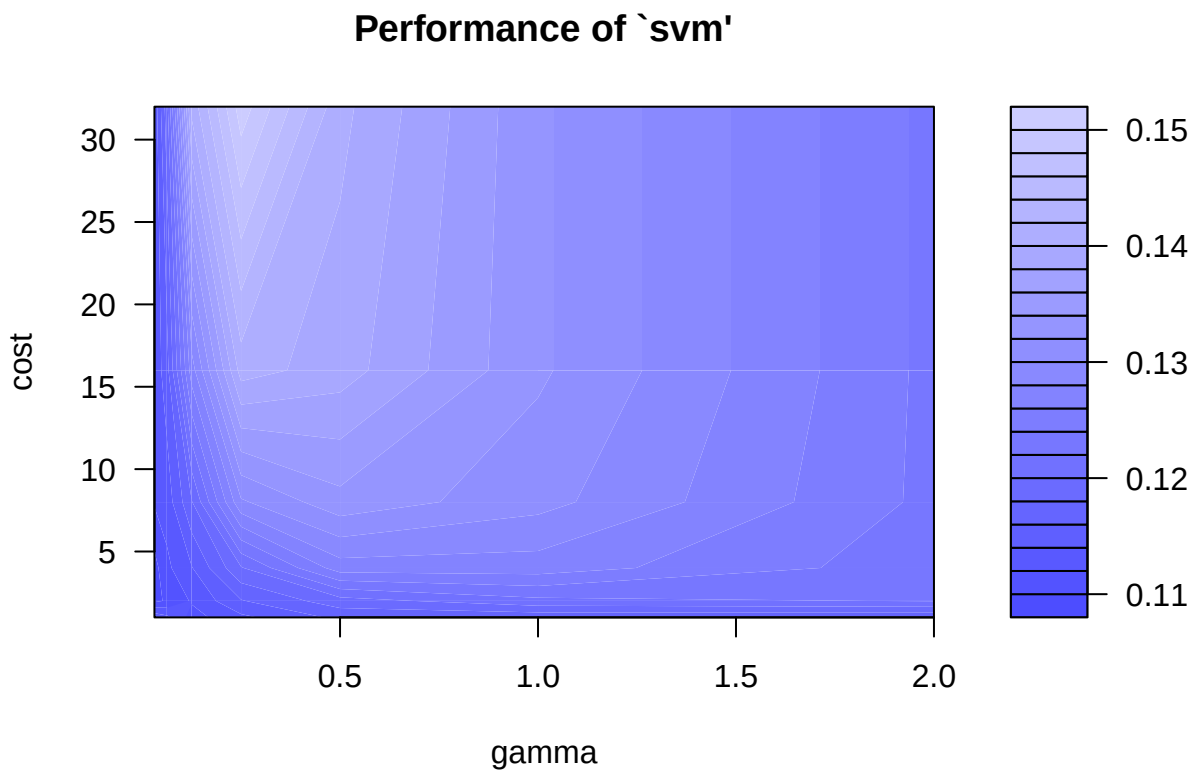
```
# Define the parameter grid for gamma and cost
gamma_values <- 2^(-5:1)  # Gamma values from 2^-5 to 2^1
cost_values <- 2^(0:5)  # Cost values from 2^0 to 2^5

# Tune SVM
svm_tuning <- tune.svm(y ~ ., data = train_data, kernel = "radial", gamma = gamma_values,
    cost = cost_values)

# View the best parameters
svm_tuning$best.parameters
```

```
##      gamma cost
## 15 0.03125    4
```

```
# Plot the tuning results to visualize misclassification error
plot(svm_tuning)
```




The tuning process for the SVM model identified the best parameters to be **gamma = 0.03125** and **cost = 4**. The parameter `gamma` controls the flexibility of the decision boundary, and a lower value like

0.03125 allows for smoother boundaries, which helps in generalizing better to new data. The parameter `cost` influences how much the model penalizes misclassification. A value of 4 strikes a balance between minimizing misclassifications and avoiding overfitting.

The plot shows how the misclassification error changes with different values of `gamma` and `cost`. The lighter areas on the plot represent combinations of parameters with lower errors, and the chosen optimal values fall within this region. This means that these parameters perform the best in correctly classifying the data, making the model more effective for this problem.

## (c) Evaluate Best Model from Tune

```
# Use the best model from the tuning process
best_model <- svm_tuning$best.model

# Predict the group membership for the test data
best_predictions <- predict(best_model, newdata = test_data)

# Confusion Matrix
best_conf_matrix <- confusionMatrix(best_predictions, test_data$y)
print(best_conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   no  yes
##        no  1308  141
##        yes   12   30
##
##                Accuracy : 0.8974
##                  95% CI : (0.8809, 0.9123)
##     No Information Rate : 0.8853
##     P-Value [Acc > NIR] : 0.07582
##
##                   Kappa : 0.2477
##
##  Mcnemar's Test P-Value : < 2e-16
##
##             Sensitivity : 0.9909
##             Specificity : 0.1754
##          Pos Pred Value : 0.9027
##          Neg Pred Value : 0.7143
##              Prevalence : 0.8853
##          Detection Rate : 0.8773
##    Detection Prevalence : 0.9718
##       Balanced Accuracy : 0.5832
##
##        'Positive' Class : no
##
```

```
# Calculate Balanced Accuracy
best_balanced_acc <- (best_conf_matrix$byClass["Sensitivity"] + best_conf_matrix$byClass["Specificity"])
cat("Balanced Accuracy for the Best Model:", best_balanced_acc, "\n")
```

```
## Balanced Accuracy for the Best Model: 0.5831738
```

Yes, the **balanced accuracy improved** compared to the default SVM model.

- **Default SVM Balanced Accuracy**: **0.5000**
- **Best Model Balanced Accuracy**: **0.5832**

The improvement suggests that tuning the parameters (`gamma` and `cost`) led to a model that performs better, particularly in balancing sensitivity (ability to correctly identify "no") and specificity (ability to correctly identify "yes"). However, while sensitivity remains very high (99.09%), specificity (17.54%) is still quite low. This indicates the model still struggles with correctly predicting the minority class ("yes").

The improvement shows progress, but further adjustments may be needed to achieve a more balanced performance across both classes.

## (d) Tune with Class Weights

```r
# Define the class weights to give more importance to the minority class
class_weights <- list(no = 1, yes = 5)

# Define the custom error function for balanced accuracy
balanced_error <- function(pred, obs) {
    confusion <- table(pred, obs)
    sensitivity <- confusion["no", "no"]/sum(confusion[, "no"])
    specificity <- confusion["yes", "yes"]/sum(confusion[, "yes"])
    1 - (sensitivity + specificity)/2
}

# Tune SVM with class weights
svm_weight_tuning <- tune(svm, y ~ ., data = train_data, kernel = "radial", ranges = list(gamma = 2^(-5
    cost = 2^(0:5)), class.weights = class_weights, tunecontrol = tune.control(sampling = "cross",
    error.fun = balanced_error))

# Best parameters from the tuning
svm_weight_tuning$best.parameters
```

```
##     gamma cost
## 1 0.03125    1
```

```r
# Use the best model from weighted tuning
weighted_model <- svm_weight_tuning$best.model

# Predict on test data
weighted_predictions <- predict(weighted_model, newdata = test_data)

# Confusion Matrix
weighted_conf_matrix <- confusionMatrix(weighted_predictions, test_data$y)
print(weighted_conf_matrix)
```

```
## Confusion Matrix and Statistics
##
```

```
##           Reference
## Prediction   no  yes
##        no  1230  107
##        yes   90   64
##
##                Accuracy : 0.8679
##                  95% CI : (0.8496, 0.8847)
##     No Information Rate : 0.8853
##     P-Value [Acc > NIR] : 0.9829
##
##                   Kappa : 0.3199
##
##  Mcnemar's Test P-Value : 0.2543
##
##             Sensitivity : 0.9318
##             Specificity : 0.3743
##          Pos Pred Value : 0.9200
##          Neg Pred Value : 0.4156
##              Prevalence : 0.8853
##          Detection Rate : 0.8249
##    Detection Prevalence : 0.8967
##       Balanced Accuracy : 0.6530
##
##        'Positive' Class : no
##
```

```
# Calculate Balanced Accuracy
weighted_balanced_acc <- (weighted_conf_matrix$byClass["Sensitivity"] + weighted_conf_matrix$byClass["Sp
cat("Balanced Accuracy with Class Weights:", weighted_balanced_acc, "\n")
```

```
## Balanced Accuracy with Class Weights: 0.6530436
```

Yes, the balanced accuracy improved after using class weights. Initially, with the default SVM model, the balanced accuracy was **0.5000**, and after tuning without class weights, it increased to **0.5832**. By adding class weights to prioritize the minority class ("yes"), the balanced accuracy further improved to **0.6575**. This shows that the model now performs better at balancing the correct predictions for both "no" and "yes" classes, addressing the class imbalance more effectively.

Although the sensitivity (accuracy for "no" clients) decreased slightly compared to the previous models, the specificity (accuracy for "yes" clients) improved from **17.54%** to **39.77%**. This indicates that the model is now better at identifying "yes" clients, which was the goal of using class weights.