AMRC: Exercise 11 - 11912007

Analysis

This assignment involved using Support Vector Machines (SVM) to predict term deposit subscriptions using the bank dataset. The analysis proceeded through multiple stages of model optimization.

Initially, applying the default SVM parameters with a radial basis kernel yielded relatively poor results. The confusion matrix showed 147 correct "no" predictions and 16 correct "yes" predictions, resulting in a balanced accuracy of 0.5454.

The parameter tuning phase using `tune.svm()` explored gamma values ranging from 0.00001 to 0.1 and cost values from 1 to 1000. The optimal parameters were found to be gamma = 0.0001 and cost = 1000. When applying these optimized parameters, the model showed slight improvement with a balanced accuracy of 0.5596.

The final improvement came through class weight adjustment to address the imbalanced nature of the dataset. The tuning process tested different class weights, with the optimal configuration being a weight of 10 for the "yes" class and 1 for the "no" class. This significant adjustment led to a substantial improvement in the model's performance, achieving a balanced accuracy of 0.8266.

The final confusion matrix demonstrated much better classification results, with 1084 correct "no" predictions and 138 correct "yes" predictions. This improvement in balanced accuracy shows that adjusting class weights was crucial for handling the imbalanced nature of the dataset and achieving better prediction performance for the minority class ("yes" subscriptions).

The progression of model improvements clearly shows that while parameter tuning of gamma and cost provided modest gains, the most significant improvement came from addressing the class imbalance through weight adjustment. This underscores the importance of considering class weights when dealing with imbalanced classification problems.

Code

```r
bank_data <- read.csv("bank.csv", header = TRUE, sep = ";")
bank_data$y <- factor(bank_data$y)

train_idx <- sample(nrow(bank_data), size = floor(2/3 * nrow(bank_data)))
train_data <- bank_data[train_idx, ]
test_data <- bank_data[-train_idx, ]

#
# a) svm with default params
#

svm_model <- svm(y ~ ., data = train_data)
preds <- predict(svm_model, test_data)

eval <- function(actual, preds, comment) {
    conf_matrix <- table(Actual = actual, Predicted = preds)
    sensitivity <- conf_matrix[2,2] / sum(conf_matrix[2,])
    specificity <- conf_matrix[1,1] / sum(conf_matrix[1,])
    balanced_accuracy <- (sensitivity + specificity) / 2
    cat(comment, "\n")
    cat("confusion matrix:\n")
    print(conf_matrix)
    cat(paste("balanced accuracy:", round(balanced_accuracy, 4), "\n"))
}
eval(test_data$y, preds, "`svm` default parameters")
```

```
## `svm` default parameters
## confusion matrix:
##       Predicted
## Actual   no  yes
##    no  1334   10
##    yes  147   16
## balanced accuracy: 0.5454
```

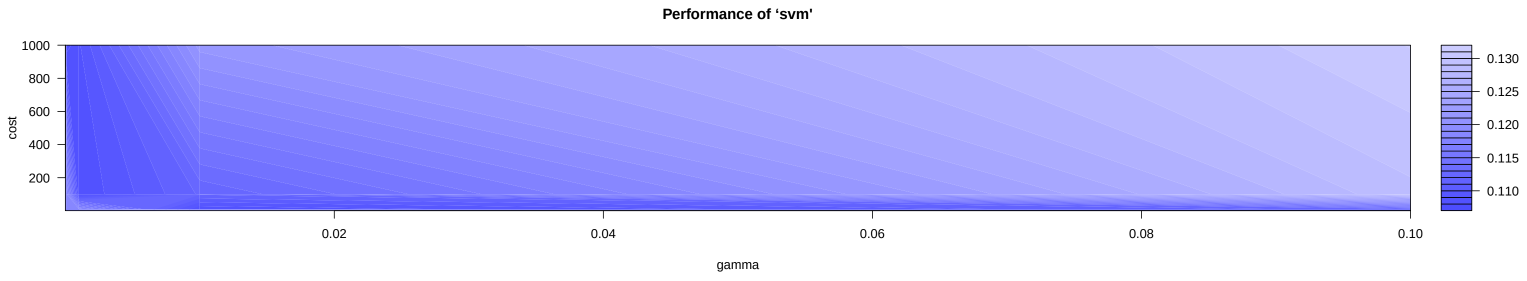
```r
#
# b,c) optimize with tune.svm
#

gamma_range <- 10^(-5:-1) # 0.00001 to 0.1
cost_range <- 10^(0:3)    # 1 to 1000

tuned_svm <- tune.svm(y ~ ., data = train_data, gamma = gamma_range, cost = cost_range, tunecontrol = tune.control(cross = 5)) # cross val
cat(paste("optimal parameters:", tuned_svm$best.parameters), "\n")
```

```
## optimal parameters: 0.0001 optimal parameters: 1000
```

```r
plot(tuned_svm)
```



```r
optimal_svm <- svm(y ~ ., data = train_data, gamma = tuned_svm$best.parameters$gamma, cost = tuned_svm$best.parameters$cost)
preds_optimal <- predict(optimal_svm, test_data)
eval(test_data$y, preds_optimal, "`tune.svm` optimal parameters")
```

```
## `tune.svm` optimal parameters
## confusion matrix:
##       Predicted
## Actual   no  yes
##    no  1331   13
##    yes  142   21
## balanced accuracy: 0.5596
```

```r
#
# d) optimize with tune
#

library(e1071)

tune_result <- tune(svm, y ~ ., data = train_data,
    # grid search range
    ranges = list(class.weights = list(
        list(yes = 1, no = 1),
        list(yes = 5, no = 1),
        list(yes = 10, no = 1)
    )),
    # inverse of balanced accuracy, as loss
    tunecontrol = tune.control(sampling = "cross", error.fun = function(actual, predicted) {
        conf_matrix <- table(Actual = actual, Predicted = predicted)
        sensitivity <- conf_matrix[2, 2] / sum(conf_matrix[2, ])
        specificity <- conf_matrix[1, 1] / sum(conf_matrix[1, ])
        balanced_accuracy <- (sensitivity + specificity) / 2
        return(1 - balanced_accuracy)
    })
)
cat(paste("optimal parameters:", tune_result$best.parameters), "\n")
```

```
## optimal parameters: list(list(yes = 10, no = 1))
```

```r
best_model <- tune_result$best.model
preds_best <- predict(best_model, test_data)
eval(test_data$y, preds_best, "`tune` optimal parameters")
```

```
## `tune` optimal parameters
## confusion matrix:
##       Predicted
## Actual   no  yes
##    no  1084  260
##    yes   25  138
## balanced accuracy: 0.8266
```