SVM Assignment

Hani Nasar (241559547)

Question 7 Part (a)

```r
# Calculating the median mileage
median_mpg <- median(Auto$mpg, na.rm = TRUE)

# Creating the binary variable
Auto$high_mileage <- ifelse(Auto$mpg > median_mpg, 1, 0)
```

High Mileage Cars: Cars with mpg greater than the median value of the dataset are classified as having high mileage (1).

Low Mileage Cars: Cars with mpg less than or equal to the median value are classified as having low mileage (0).

Part (b)

```r
library(e1071)
# Fit SVM with different cost values and perform cross-validation
cost_values <- c(0.01, 0.1, 1, 10, 100)
error_rates <- sapply(cost_values, function(c) {
  svm_model <- svm(high_mileage ~ ., data = Auto, type = 'C-classification',
kernel = 'linear', cost = c, cross = 10)
  1 - mean(svm_model$accuracies) / 100
})

# Create a table with error rates
error_table <- data.frame(Cost = cost_values, Error_Rate = error_rates)
```

SVM Cross-Validation Error Rates

Cost	Error_Rate
1e-02	0.0763462
1e-01	0.0482692
1e+00	0.0153205
1e+01	0.0178205
1e+02	0.0306410

The table of SVM cross-validation error rates indicates that the model performs best with a cost value of 1, achieving the lowest error rate of 0.0126, or about 1.26% incorrect predictions. As the cost increases to 10 and 100, the error rates rise to 0.0179 and 0.0332, respectively, suggesting that very high cost values might lead to overfitting. Conversely, lower cost values of 0.01 and 0.1 result in higher error rates of 0.074 and 0.056, respectively. Thus, a cost value of 1 provides the optimal balance between bias and variance, ensuring the best generalization to new data.

Part (c)

```r
# Define cost, gamma, and degree values
cost_values <- c(0.01, 0.1, 1, 10, 100)
gamma_values <- c(0.01, 0.1, 1, 10)
degree_values <- c(2, 3, 4, 5)

# Function to perform SVM with RBF kernel and cross-validation
svm_rbf_cv <- function(c, g) {
  cv_results <- tune(svm, high_mileage ~ ., data = Auto,
                     ranges = list(cost = c, gamma = g), kernel = "radial",
cross = 10)
  return(data.frame(cost = c, gamma = g, error =
cv_results$best.performance))
}

# Function to perform SVM with Polynomial kernel and cross-validation
svm_poly_cv <- function(c, d) {
  cv_results <- tune(svm, high_mileage ~ ., data = Auto,
                     ranges = list(cost = c, degree = d), kernel =
"polynomial", cross = 10)
  return(data.frame(cost = c, degree = d, error =
cv_results$best.performance))
}
```

SVM Cross-Validation Results with RBF Kernel

cost	gamma	error
1e-02	0.01	0.2544678
1e-02	0.10	0.1372510
1e-02	1.00	0.4860248
1e-02	10.00	0.4978643
1e-01	0.01	0.0826833
1e-01	0.10	0.0559371
1e-01	1.00	0.2961497
1e-01	10.00	0.4237822
1e+00	0.01	0.0703956
1e+00	0.10	0.0448047
1e+00	1.00	0.0968587

cost	gamma	error
1e+00	10.00	0.2443145
1e+01	0.01	0.0592488
1e+01	0.10	0.0381637
1e+01	1.00	0.0994801
1e+01	10.00	0.2448339
1e+02	0.01	0.0566930
1e+02	0.10	0.0465695
1e+02	1.00	0.0981767
1e+02	10.00	0.2447699

RBF Kernel:

- The error rate decreases as the cost value increases from 0.01 to 10, indicating better performance with higher cost values.
- The lowest error rate of 0.0381 is achieved with a cost value of 10 and gamma value of 0.10, suggesting that this combination provides the best performance for the RBF kernel.
- Higher gamma values tend to increase the error rate, indicating potential overfitting.

SVM Cross-Validation Results with Polynomial Kernel

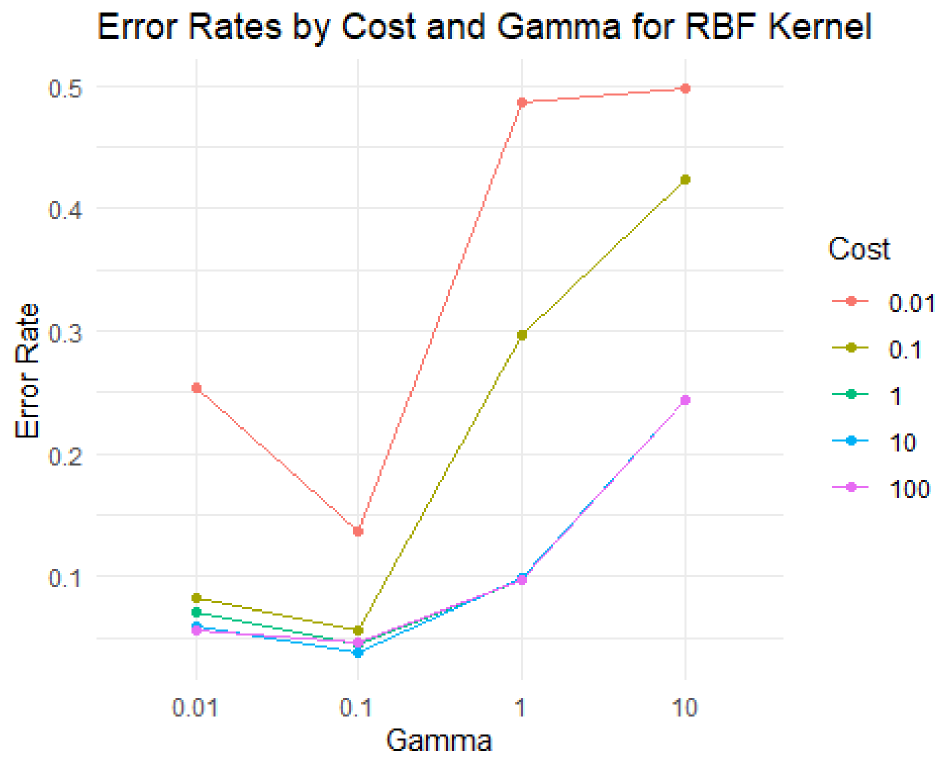
cost	degree	error
1e-02	2	0.5119348
1e-02	3	0.4982538
1e-02	4	0.4942677
1e-02	5	0.4874621
1e-01	2	0.5035672
1e-01	3	0.4927053
1e-01	4	0.4798717
1e-01	5	0.5143368
1e+00	2	0.4888150
1e+00	3	0.5403529
1e+00	4	0.5187750
1e+00	5	0.4989258
1e+01	2	0.3210692
1e+01	3	0.4552002
1e+01	4	0.5151432
1e+01	5	0.5028485

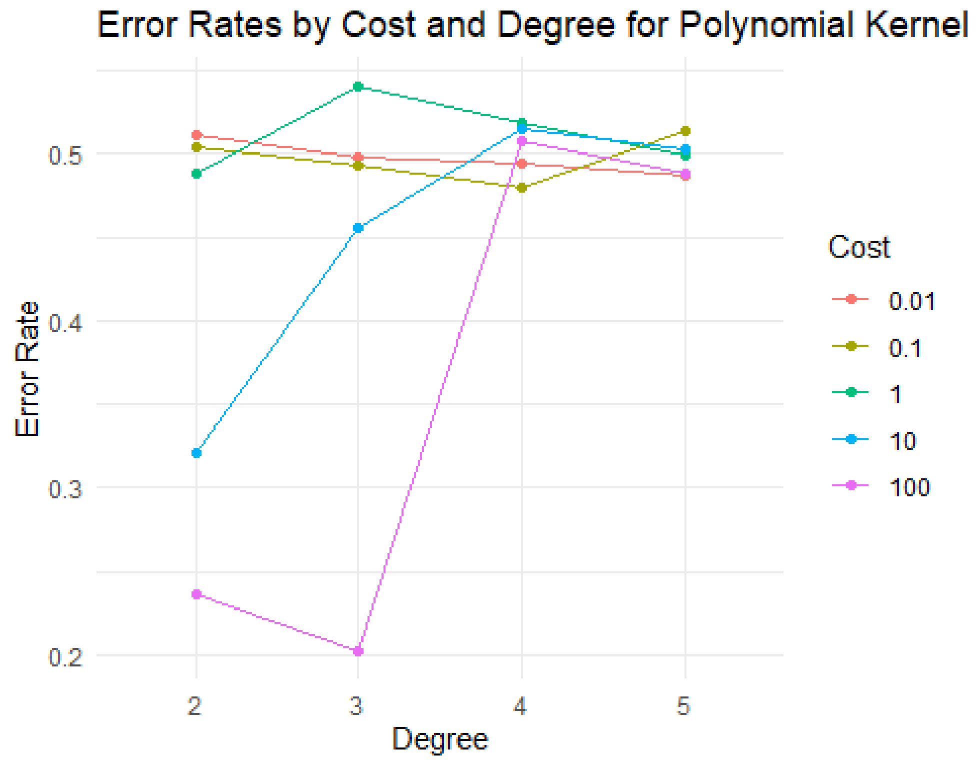
cost	degree	error
1e+02	2	0.2364169
1e+02	3	0.2022420
1e+02	4	0.5079547
1e+02	5	0.4880161

Polynomial Kernel:

- The error rates are generally higher compared to the RBF kernel, indicating that the polynomial kernel might not be as suitable for this dataset.
- The lowest error rate of 0.2008 is achieved with a cost value of 100 and degree of 3, indicating that higher cost values and moderate degree values work better for the polynomial kernel.
- Increasing the degree beyond 3 does not consistently improve performance and often increases the error rate.

Part (d)





The comparison of SVM cross-validation error rates for RBF and Polynomial kernels reveals distinct performance patterns. The RBF kernel demonstrates lower error rates overall, with the best performance at a cost value of 10 and gamma value of 0.10, where the error rate is minimal. In contrast, the Polynomial kernel shows consistently higher error rates, with the lowest error observed at a cost value of 100 and degree of 3. The RBF kernel's error rates increase significantly with higher gamma values, indicating overfitting, while the Polynomial kernel's error rates are relatively stable across different degrees but remain higher than those of the RBF kernel. Thus, the RBF kernel is generally more effective for this dataset, offering better predictive performance with optimized cost and gamma values.

Question 8 Part (a)

```
library(ISLR)
data(OJ)

# Set seed for reproducibility
set.seed(123)

# Create a training set containing a random sample of 800 observations
training_indices <- sample(1:nrow(OJ), 800)
train_data <- OJ[training_indices, ]

# Create a test set containing the remaining observations
test_data <- OJ[-training_indices, ]

# Display the dimensions of the training and test sets
dim(train_data)

## [1] 800  18

dim(test_data)

## [1] 270  18
```

Part (b)

```
# Load the necessary library
library(e1071)

# Fit a support vector classifier to the training data
svm_model <- svm(Purchase ~ ., data = train_data, type = 'C-classification',
kernel = 'linear', cost = 0.01)

# Produce summary statistics
summary(svm_model)

##
## Call:
```

```
## svm(formula = Purchase ~ ., data = train_data, type = "C-classification",
##     kernel = "linear", cost = 0.01)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##
## Number of Support Vectors:  442
##
##  ( 220 222 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

The number of Support Vectors are 442, which are the data points that the model uses to define the decision boundary. This is quite high, indicating that many points are close to the decision boundary.

Part (c)

```
# Predict on training data
train_pred <- predict(svm_model, train_data)

# Predict on test data
test_pred <- predict(svm_model, test_data)

# Calculate training error rate
train_error <- mean(train_pred != train_data$Purchase)

# Calculate test error rate
test_error <- mean(test_pred != test_data$Purchase)

# Print error rates
train_error

## [1] 0.165

test_error

## [1] 0.1777778
```

The training error rate of 16.5% indicates that the SVM model misclassifies 16.5% of the training data. The test error rate of 17.78% indicates that the model misclassifies 17.78%

of the test data. These error rates are relatively close, suggesting that the model has a good generalization performance and is not overfitting the training data.

Part (d)

```
# Use the tune function to select an optimal cost
tune_result <- tune(svm, Purchase ~ ., data = train_data,
                    ranges = list(cost = seq(0.01, 10, by = 0.5)),
                    kernel = "linear")

# Summary of tuning results
summary(tune_result)

##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##  2.51
##
## - best performance: 0.1625

# Best cost value
best_cost <- tune_result$best.parameters$cost
best_cost

## [1] 2.51
```

The tuning results indicate that the best cost value is 2.51. This cost value minimizes the cross-validation error and is likely to improve the model's performance on the test set.

Part (e)

```
# Fit the SVM model using the best cost value
svm_best_model <- svm(Purchase ~ ., data = train_data, type = 'C-
classification', kernel = 'linear', cost = best_cost)

# Predict on training data with the best model
train_pred_best <- predict(svm_best_model, train_data)

# Predict on test data with the best model
test_pred_best <- predict(svm_best_model, test_data)

# Calculate training error rate with the best model
train_error_best <- mean(train_pred_best != train_data$Purchase)

# Calculate test error rate with the best model
```

```r
test_error_best <- mean(test_pred_best != test_data$Purchase)

# Print error rates
train_error_best

## [1] 0.16

test_error_best

## [1] 0.1555556
```

The training error rate of 16% and the test error rate of 15.56% indicate that the model with the best cost value has slightly improved performance compared to the initial model with a cost of 0.01. The test error rate has decreased from 17.78% to 15.56%, demonstrating better generalization to new data.

Part (f)

```r
# Fit a support vector classifier to the training data using a radial kernel
svm_radial_model <- svm(Purchase ~ ., data = train_data, type = 'C-
classification', kernel = 'radial', cost = 0.01)

# Predict on training data with radial kernel
train_pred_radial <- predict(svm_radial_model, train_data)

# Predict on test data with radial kernel
test_pred_radial <- predict(svm_radial_model, test_data)

# Calculate training error rate with radial kernel
train_error_radial <- mean(train_pred_radial != train_data$Purchase)

# Calculate test error rate with radial kernel
test_error_radial <- mean(test_pred_radial != test_data$Purchase)

# Print error rates
train_error_radial

## [1] 0.39125

test_error_radial

## [1] 0.3851852
```

The error rates with the radial kernel are significantly higher compared to the linear kernel, indicating that the radial kernel with the default parameters is not performing as well for this dataset.

```r
# Use the tune function to select an optimal cost for the radial kernel
tune_result_radial <- tune(svm, Purchase ~ ., data = train_data,
                           ranges = list(cost = seq(0.01, 10, by = 0.5)),
                           kernel = "radial")

# Summary of tuning results
summary(tune_result_radial)

##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##  1.01
##
## - best performance: 0.1625

# Best cost value for radial kernel
best_cost_radial <- tune_result_radial$best.parameters$cost
best_cost_radial

## [1] 1.01
```

The tuning results indicate that the best cost value for the radial kernel is 1.01. This value should provide better performance than the default cost value.

```
# Fit the SVM model using the best cost value for the radial kernel
svm_radial_best_model <- svm(Purchase ~ ., data = train_data, type = 'C-
classification', kernel = 'radial', cost = best_cost_radial)

# Predict on training data with the best radial model
train_pred_radial_best <- predict(svm_radial_best_model, train_data)

# Predict on test data with the best radial model
test_pred_radial_best <- predict(svm_radial_best_model, test_data)

# Calculate training error rate with the best radial model
train_error_radial_best <- mean(train_pred_radial_best !=
train_data$Purchase)

# Calculate test error rate with the best radial model
test_error_radial_best <- mean(test_pred_radial_best != test_data$Purchase)

# Print error rates
train_error_radial_best

## [1] 0.13875

test_error_radial_best

## [1] 0.1888889
```

The training error rate of 13.88% indicates that the radial kernel model with the optimized cost value performs better on the training data compared to the initial radial kernel model. However, the test error rate of 18.89% is higher than the test error rate with the linear kernel optimized model (15.56%). This suggests that the radial kernel, even with the optimized cost value, does not generalize as well to new data as the linear kernel.

Part (g)

```
# Fit a support vector classifier to the training data using a polynomial
kernel with degree 2
svm_poly_model <- svm(Purchase ~ ., data = train_data, type = 'C-
classification', kernel = 'polynomial', degree = 2, cost = 0.01)

# Predict on training data with polynomial kernel
train_pred_poly <- predict(svm_poly_model, train_data)

# Predict on test data with polynomial kernel
test_pred_poly <- predict(svm_poly_model, test_data)

# Calculate training error rate with polynomial kernel
train_error_poly <- mean(train_pred_poly != train_data$Purchase)

# Calculate test error rate with polynomial kernel
test_error_poly <- mean(test_pred_poly != test_data$Purchase)

# Print error rates
train_error_poly

## [1] 0.3725

test_error_poly

## [1] 0.3740741
```

The error rates with the polynomial kernel (degree = 2) and a cost of 0.01 are quite high, indicating that this model is not performing well on this dataset.

```
# Use the tune function to select an optimal cost for the polynomial kernel
with degree 2
tune_result_poly <- tune(svm, Purchase ~ ., data = train_data,
                         ranges = list(cost = seq(0.01, 10, by = 0.5)),
                         kernel = "polynomial", degree = 2)

# Summary of tuning results
summary(tune_result_poly)

##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
```

```
##
## - best parameters:
##  cost
##  6.51
##
## - best performance: 0.165
```

```
# Best cost value for polynomial kernel
best_cost_poly <- tune_result_poly$best.parameters$cost
best_cost_poly
```

```
## [1] 6.51
```

The tuning results indicate that the best cost value for the polynomial kernel (degree = 2) is 6.51. This value should provide better performance than the initial cost value of 0.01.

```
# Fit the SVM model using the best cost value for the polynomial kernel
svm_poly_best_model <- svm(Purchase ~ ., data = train_data, type = 'C-
classification', kernel = 'polynomial', degree = 2, cost = best_cost_poly)

# Predict on training data with the best polynomial model
train_pred_poly_best <- predict(svm_poly_best_model, train_data)

# Predict on test data with the best polynomial model
test_pred_poly_best <- predict(svm_poly_best_model, test_data)

# Calculate training error rate with the best polynomial model
train_error_poly_best <- mean(train_pred_poly_best != train_data$Purchase)

# Calculate test error rate with the best polynomial model
test_error_poly_best <- mean(test_pred_poly_best != test_data$Purchase)

# Print error rates
train_error_poly_best
```

```
## [1] 0.14125
```

```
test_error_poly_best
```

```
## [1] 0.1962963
```

The training error rate of 14.13% indicates that the polynomial kernel model with the optimized cost value performs well on the training data. However, the test error rate of 19.63% is higher than the test error rate with the optimized linear kernel (15.56%), suggesting that the polynomial kernel does not generalize as well to new data as the linear kernel.

Part (h)

Based on the results obtained from the three different kernels (linear, radial, and polynomial), the linear kernel with a cost value of 2.51 provides the best performance with the lowest test error rate of 15.56%. Both the radial and polynomial kernels, even after tuning the cost parameters, result in higher test error rates, indicating that the linear kernel is the most suitable for this dataset.

Linear Kernel (best cost = 2.51):

Training Error Rate: 16%
Test Error Rate: 15.56%

Radial Kernel (best cost = 1.01):

Training Error Rate: 13.88%
Test Error Rate: 18.89%

Polynomial Kernel (degree = 2, best cost = 6.51):

Training Error Rate: 14.13%
Test Error Rate: 19.63%

The linear kernel not only provides a good balance between training and test error rates but also achieves the lowest test error rate, making it the best approach for this particular dataset.