HW07_zilinw3

Zilin Wang (zilinw3)

2024-10-14

Contents

Question 1: SVM on Hand Written Digit Data (55 points)

We will again use the `MNIST` dataset. We will use the first 2400 observations of it:

```
# inputs to download file
fileLocation <- "https://pjreddie.com/media/files/mnist_train.csv"
numRowsToDownload <- 2400
localFileName <- paste0("mnist_first", numRowsToDownload, ".RData")

# download the data and add column names
mnist2400 <- read.csv(fileLocation, nrows = numRowsToDownload)
numColsMnist <- dim(mnist2400)[2]
colnames(mnist2400) <- c("Digit", paste("Pixel", seq(1:(numColsMnist - 1)), sep = ""))

# save file
# in the future we can read in from the local copy instead of having to redownload
save(mnist2400, file = localFileName)

# you can load the data with the following code
#load(file = localFileName)
```

a. [15 pts] Since a standard SVM can only be used for binary classification problems, let's fit SVM on digits 4 and 5. Complete the following tasks.

- Use digits 4 and 5 in the first 1200 observations as training data and those in the remaining part with digits 4 and 5 as testing data.
- Fit a linear SVM on the training data using the `e1071` package. Set the cost parameter $C = 1$.
- You will possibly encounter two issues: first, this might be slow (unless your computer is very powerful); second, the package will complain about some pixels being problematic (zero variance). Hence, reducing the number of variables by removing pixels with low variances is probably a good idea. Perform a marginal screening of variance on the pixels and select the top 250 Pixels with the highest marginal variance.
- Redo your SVM model with the pixels you have selected. Report the training and testing classification errors.

Answer:

```
library(e1071)
```

```
## Warning:  'e1071' R 4.3.3
```

```
# Splitting the data into training and testing sets
train_data_1 <- mnist2400[1:1200, ]
test_data_1 <- mnist2400[1201:2400, ]

# Filter for digits 4 and 5 only
train_data <- train_data_1[train_data_1$Digit %in% c(4, 5),]
test_data <- test_data_1[test_data_1$Digit %in% c(4, 5),]

# Fit a linear SVM on the training data
svm_model <- svm(Digit ~ ., data = train_data, type = 'C-classification', kernel = 'linear', C = 1)
```

```
## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):
## Variable(s) 'Pixel1' and 'Pixel2' and 'Pixel3' and 'Pixel4' and 'Pixel5' and
## 'Pixel6' and 'Pixel7' and 'Pixel8' and 'Pixel9' and 'Pixel10' and 'Pixel11' and
## 'Pixel12' and 'Pixel13' and 'Pixel14' and 'Pixel15' and 'Pixel16' and 'Pixel17'
## and 'Pixel18' and 'Pixel19' and 'Pixel20' and 'Pixel21' and 'Pixel22' and
## 'Pixel23' and 'Pixel24' and 'Pixel25' and 'Pixel26' and 'Pixel27' and 'Pixel28'
## and 'Pixel29' and 'Pixel30' and 'Pixel31' and 'Pixel32' and 'Pixel33' and
## 'Pixel34' and 'Pixel35' and 'Pixel36' and 'Pixel37' and 'Pixel38' and 'Pixel39'
## and 'Pixel40' and 'Pixel41' and 'Pixel42' and 'Pixel43' and 'Pixel44' and
## 'Pixel45' and 'Pixel46' and 'Pixel47' and 'Pixel48' and 'Pixel49' and 'Pixel50'
## and 'Pixel51' and 'Pixel52' and 'Pixel53' and 'Pixel54' and 'Pixel55' and
## 'Pixel56' and 'Pixel57' and 'Pixel58' and 'Pixel59' and 'Pixel60' and 'Pixel61'
## and 'Pixel62' and 'Pixel63' and 'Pixel64' and 'Pixel65' and 'Pixel66' and
## 'Pixel67' and 'Pixel68' and 'Pixel69' and 'Pixel70' and 'Pixel71' and 'Pixel75'
## and 'Pixel76' and 'Pixel77' and 'Pixel78' and 'Pixel79' and 'Pixel80' and
## 'Pixel81' and 'Pixel82' and 'Pixel83' and 'Pixel84' and 'Pixel85' and 'Pixel86'
## and 'Pixel87' and 'Pixel88' and 'Pixel89' and 'Pixel90' and 'Pixel91' and
## 'Pixel92' and 'Pixel93' and 'Pixel94' and 'Pixel95' and 'Pixel110' and
## 'Pixel111' and 'Pixel112' and 'Pixel113' and 'Pixel114' and 'Pixel115' and
## 'Pixel116' and 'Pixel119' and 'Pixel120' and 'Pixel139' and 'Pixel140' and
## 'Pixel141' and 'Pixel142' and 'Pixel143' and 'Pixel167' and 'Pixel168' and
## 'Pixel169' and 'Pixel170' and 'Pixel171' and 'Pixel196' and 'Pixel197' and
## 'Pixel198' and 'Pixel199' and 'Pixel224' and 'Pixel225' and 'Pixel226' and
## 'Pixel227' and 'Pixel252' and 'Pixel253' and 'Pixel254' and 'Pixel255' and
## 'Pixel280' and 'Pixel281' and 'Pixel282' and 'Pixel283' and 'Pixel307' and
## 'Pixel308' and 'Pixel309' and 'Pixel310' and 'Pixel311' and 'Pixel335' and
## 'Pixel336' and 'Pixel337' and 'Pixel338' and 'Pixel339' and 'Pixel362' and
## 'Pixel363' and 'Pixel364' and 'Pixel365' and 'Pixel366' and 'Pixel367' and
## 'Pixel390' and 'Pixel391' and 'Pixel392' and 'Pixel393' and 'Pixel394' and
## 'Pixel395' and 'Pixel418' and 'Pixel419' and 'Pixel420' and 'Pixel421' and
## 'Pixel422' and 'Pixel423' and 'Pixel446' and 'Pixel447' and 'Pixel448' and
## 'Pixel449' and 'Pixel450' and 'Pixel451' and 'Pixel475' and 'Pixel476' and
## 'Pixel477' and 'Pixel478' and 'Pixel479' and 'Pixel503' and 'Pixel504' and
## 'Pixel505' and 'Pixel506' and 'Pixel507' and 'Pixel529' and 'Pixel530' and
## 'Pixel531' and 'Pixel532' and 'Pixel533' and 'Pixel534' and 'Pixel535' and
```

```
## 'Pixel536' and 'Pixel558' and 'Pixel559' and 'Pixel560' and 'Pixel561' and
## 'Pixel562' and 'Pixel563' and 'Pixel586' and 'Pixel587' and 'Pixel588' and
## 'Pixel589' and 'Pixel590' and 'Pixel591' and 'Pixel614' and 'Pixel615' and
## 'Pixel616' and 'Pixel617' and 'Pixel618' and 'Pixel619' and 'Pixel642' and
## 'Pixel643' and 'Pixel644' and 'Pixel645' and 'Pixel646' and 'Pixel647' and
## 'Pixel648' and 'Pixel670' and 'Pixel671' and 'Pixel672' and 'Pixel673' and
## 'Pixel674' and 'Pixel675' and 'Pixel676' and 'Pixel677' and 'Pixel678' and
## 'Pixel698' and 'Pixel699' and 'Pixel700' and 'Pixel701' and 'Pixel702' and
## 'Pixel703' and 'Pixel704' and 'Pixel705' and 'Pixel706' and 'Pixel708' and
## 'Pixel723' and 'Pixel724' and 'Pixel725' and 'Pixel726' and 'Pixel727' and
## 'Pixel728' and 'Pixel729' and 'Pixel730' and 'Pixel731' and 'Pixel732' and
## 'Pixel733' and 'Pixel734' and 'Pixel735' and 'Pixel736' and 'Pixel739' and
## 'Pixel740' and 'Pixel741' and 'Pixel743' and 'Pixel748' and 'Pixel749' and
## 'Pixel750' and 'Pixel751' and 'Pixel752' and 'Pixel753' and 'Pixel754' and
## 'Pixel755' and 'Pixel756' and 'Pixel757' and 'Pixel758' and 'Pixel759' and
## 'Pixel760' and 'Pixel761' and 'Pixel762' and 'Pixel763' and 'Pixel764' and
## 'Pixel765' and 'Pixel766' and 'Pixel767' and 'Pixel768' and 'Pixel769' and
## 'Pixel770' and 'Pixel771' and 'Pixel772' and 'Pixel773' and 'Pixel774' and
## 'Pixel775' and 'Pixel776' and 'Pixel777' and 'Pixel778' and 'Pixel779' and
## 'Pixel780' and 'Pixel781' and 'Pixel782' and 'Pixel783' and 'Pixel784'
## constant. Cannot scale data.
```

```r
# Calculate variances of each pixel and select the top 250 by variance
variances <- apply(train_data[, -1], 2, var)
top_250_indices <- order(variances, decreasing = TRUE)[1:250]
train_data_reduced <- train_data[, c(1, top_250_indices + 1)]
test_data_reduced <- test_data[, c(1, top_250_indices + 1)]

# Fit the SVM model
svm_model_refit <- svm(Digit ~ ., data = train_data_reduced, type = 'C-classification', kernel = 'linea

# Predictions and accuracy computation
train_predictions <- predict(svm_model_refit, train_data_reduced)
test_predictions <- predict(svm_model_refit, test_data_reduced)
train_ce <- mean(train_predictions != train_data_reduced$Digit)
test_ce <- mean(test_predictions != test_data_reduced$Digit)

# Output
cat("Training Classification Error: ", train_ce, "\n")
```

```
## Training Classification Error:  0
```

```r
cat("Testing Classification Error: ", test_ce, "\n")
```

```
## Testing Classification Error:  0.03212851
```

b. [15 pts] Some researchers might be interested in knowing what pixels are more important in distinguishing the two digits. One way to do this is to extract the coefficients of the (linear) SVM model (they are fairly comparable in our case since all the variables have the same range). Keep in mind that the coefficients are those β parameter used to define the direction of the separation line, and they are can be recovered from the solution of the Lagrangian. Complete the following tasks.

- Extract the coefficients of the linear SVM model you have fitted in part 1. State the mathematical formula of how these coefficients are recovered using the solution of the Lagrangian.

- Find the top 30 pixels with the largest absolute coefficients.
- Refit the SVM using just these 30 pixels. Report the training and testing classification errors.

Answer:

```
# Extract the SVM model coefficients
coefs <- t(svm_model_refit$coefs) %*% svm_model_refit$SV

# Calculate the weights
weights <- colSums(coefs)

# Assign names to these weights based on the pixel columns used
names(weights) <- colnames(train_data_reduced)[-1]  # Excluding the 'Digit' column

# Order by absolute value and select top 30
top_30_indices <- order(abs(weights), decreasing = TRUE)[1:30]
top_30_pixels <- names(weights)[top_30_indices]
cat("Top 30 pixel names: ", top_30_pixels, "\n")
```

```
## Top 30 pixel names:  Pixel490 Pixel293 Pixel541 Pixel437 Pixel382 Pixel517 Pixel381 Pixel486 Pixel65
```

```
# Subset data to only include these top 30 pixels plus the 'Digit' column
train_data_top_30 <- train_data[, c("Digit", top_30_pixels)]
test_data_top_30 <- test_data[, c("Digit", top_30_pixels)]

# Refit the SVM model using just the top 30 pixels
svm_model_top_30 <- svm(Digit ~ ., data = train_data_top_30, type = 'C-classification', kernel = 'linea

# Predict on the training and testing data
train_predictions_top_30 <- predict(svm_model_top_30, train_data_top_30)
test_predictions_top_30 <- predict(svm_model_top_30, test_data_top_30)

# Calculate and report classification errors
train_error_top_30 <- mean(train_predictions_top_30 != train_data_top_30$Digit)
test_error_top_30 <- mean(test_predictions_top_30 != test_data_top_30$Digit)

# Print the errors
cat("Training Classification Error with Top 30 Pixels: ", train_error_top_30, "\n")
```

```
## Training Classification Error with Top 30 Pixels:  0
```

```
cat("Testing Classification Error with Top 30 Pixels: ", test_error_top_30, "\n")
```

```
## Testing Classification Error with Top 30 Pixels:  0.04016064
```

The mathematical foundation for the extraction of the coefficients β in a linear SVM hinges on the relationship between the dual and primal forms of the SVM's optimization problem. The primal objective function can be stated as:

$$\min_{\beta,b} \frac{1}{2}\|\beta\|^2$$

Subject to:

$$y_i(\beta^\top x_i + b) \geq 1 - \xi_i$$

for all i.

Where b is the bias, β are the coefficients of the decision boundary, y_i are the class labels, x_i are the feature vectors, and ξ_i are the slack variables.

In the dual form, which is often used to solve the optimization problem more efficiently for certain classes of problems (including when using kernels), the solution involves support vectors and their corresponding dual coefficients α_i, leading to:

$$\hat{\beta} = \sum_{i=1}^{N} \alpha_i y_i x_i$$

where α_i is the Lagrange multiplier for each support vector.

c. [15 pts] Perform a logistic regression with elastic net penalty ($\alpha = 0.5$) on the training data. Start with the 250 pixels you have used in part a). You do not need to select the best λ value using cross-validation. Instead, select the model with just 30 variables in the solution path (what is this? you can refer to our lecture note on Lasso). What is the λ value corresponding to this model? Extract the pixels being selected by your elastic net model. Do these pixels overlap with the ones selected by the SVM model in part b)? Comment on your findings.

Answer:

```
library(glmnet)
```

```
## Warning:   'glmnet' R 4.3.3
```

```
##      Matrix
```

```
## Warning:   'Matrix' R 4.3.2
```

```
## Loaded glmnet 4.1-8
```

```
# Prepare the data
x_matrix <- as.matrix(train_data_reduced[-1])  # Exclude the label column
y_vector <- train_data_reduced$Digit

# Fit the elastic net model
elastic_net_fit <- glmnet(x_matrix, y_vector, family = "binomial", alpha = 0.5)

# Extract the coefficients for each lambda value
num_coeffs <- as.matrix(coef(elastic_net_fit))

# Count the number of non-zero coefficients for each lambda
non_zero_counts <- apply(num_coeffs[-1, ], 2, function(coef) sum(coef != 0))

# Find the lambda that corresponds to 30 non-zero coefficients
```

```r
lambda_30 <- elastic_net_fit$lambda[which(non_zero_counts == 30)][1]

# Display the lambda value
cat("Lambda for 30 variables:", lambda_30, "\n")
```

```
## Lambda for 30 variables: 0.2759404
```

```r
# Extract the coefficients for this lambda
coefficients_30 <- coef(elastic_net_fit, s = lambda_30)

# Extract the names of the pixels selected
selected_pixels <- rownames(coefficients_30)[coefficients_30[,1] != 0]
cat("Pixels selected by the Elastic Net Model:", selected_pixels, "\n")
```

```
## Pixels selected by the Elastic Net Model: (Intercept) Pixel458 Pixel459 Pixel429 Pixel457 Pixel464 P
```

```r
# Overlap with SVM-selected pixels from Part B
svm_selected_pixels <- top_30_pixels
overlap_pixels <- intersect(selected_pixels, svm_selected_pixels)
cat("Overlapping pixels:", overlap_pixels, "\n")
```

```
## Overlapping pixels: Pixel458 Pixel464 Pixel491 Pixel437 Pixel185 Pixel490 Pixel329
```

The selected pixels are spread across various positions, as indicated by their indices. These pixels were identified as significant after enforcing a regularization that combines both L1 and L2 penalties, highlighting their potential key roles in the visual features distinguishing the two digits.

The consistency in pixel selection across different models enhances confidence in the robustness and reliability of the feature selection process. It suggests that despite varying methodologies (SVM's margin maximization versus logistic regression's probabilistic approach with regularization), there is a core subset of features that are fundamentally informative.

There is a notable overlap of pixels between those selected by the SVM in part B and those identified by the logistic regression model with an elastic net penalty. This overlap is crucial as it underscores the importance of these particular pixels in effectively classifying the digits '4' and '5'. Their consistent selection across different models and methods strongly suggests that these pixels capture critical visual characteristics essential for accurate digit recognition.

The overlap might indicate regions in the images that contain distinct edges, curves, or intersections unique to digits '4' and '5'. For example, the curvature at the top of '4' or the loop of '5' could be areas where these pixels are located. The specific pixels that did not overlap might still be important but could be capturing more subtle or less universally critical features that one model may leverage better than the other. This highlights the potential utility of combining insights from multiple modeling approaches to enhance feature selection strategies in complex tasks like image recognition.

d. [10 pts] Compare the two 30-variable models you obtained from part b) and c). Use the area under the ROC curve (AUC) on the testing data as the performance metric.

Answer:

```r
library(ROCR)
```

```
## Warning:  'ROCR' R 4.3.3
```

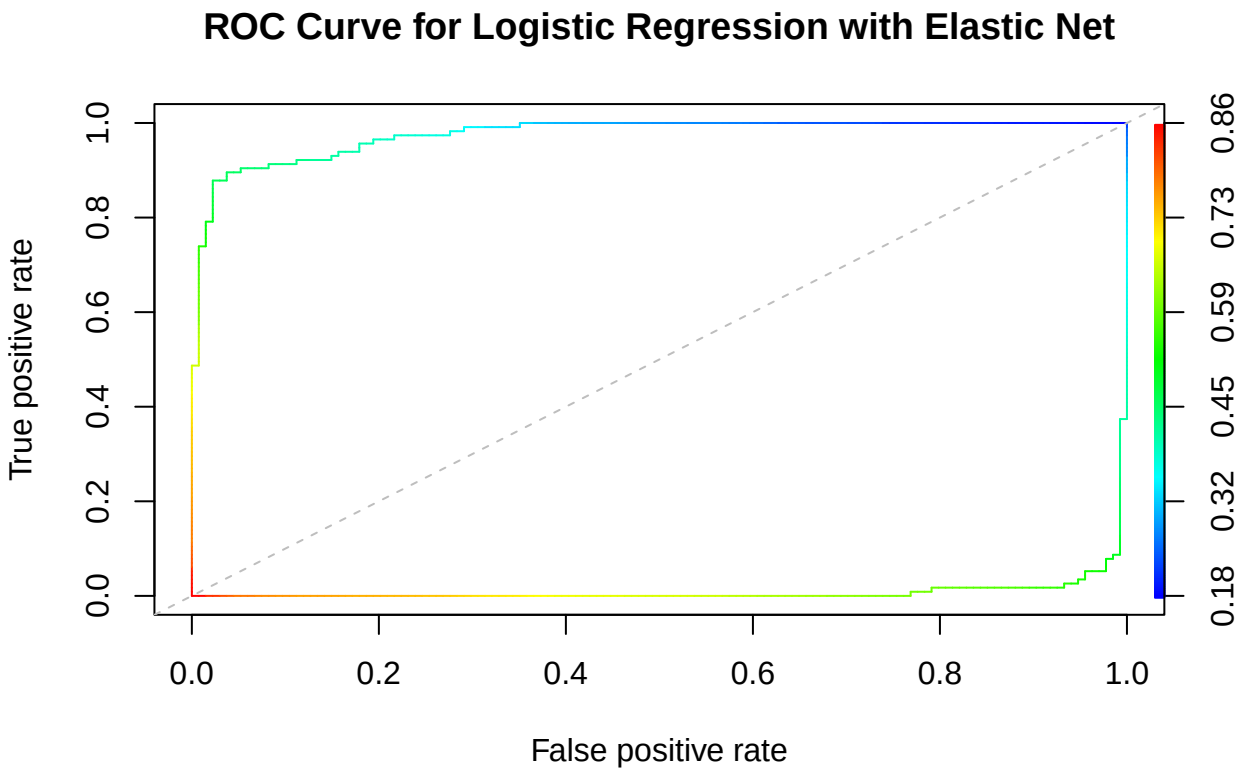
```r
# SVM
# Predicting with SVM Model
svm_predictions <- predict(svm_model_top_30, test_data_top_30, decision.values = TRUE)

# Extract the decision values
dec_values_svm <- attributes(svm_predictions)$decision.values

# Create a prediction object for ROCR
svm_pred <- prediction(dec_values_svm, test_data_top_30$Digit)

# Calculate AUC
auc_svm <- performance(svm_pred, "auc")@y.values[[1]]

# Print AUC
cat("AUC for SVM with Top 30 Pixels: ", auc_svm, "\n")
```

```
## AUC for SVM with Top 30 Pixels:  0.008500973
```

```r
# ENM
# Ensure all columns are in the correct order:
test_features <- colnames(train_data_reduced)[-1]  # Remove the 'Digit' label, use the exact order of t

# Verify if all required features are present in the test data
missing_features <- setdiff(test_features, colnames(test_data_reduced))
if (length(missing_features) > 0) {
    for (feat in missing_features) {
        test_data_reduced[[feat]] <- rep(0, nrow(test_data_reduced))  # Fill missing features with zero
    }
}

# Make sure to select and order columns as they were during model training
x_test_matrix <- as.matrix(test_data_reduced[test_features])

# Ensure binary outcome setup is correct
y_test_vector <- test_data_reduced$Digit

# Predicting with elastic net model
enm_predictions <- predict(elastic_net_fit, newx = x_test_matrix, s = lambda_30, type = "response")

# Convert predictions to a format suitable for ROCR
enm_pred <- prediction(enm_predictions, y_test_vector)

# Calculate AUC for elastic net model
auc_enm <- performance(enm_pred, "auc")@y.values[[1]]

# Print AUC
cat("AUC for Logistic Regression with Elastic Net: ", auc_enm, "\n")
```

```
## AUC for Logistic Regression with Elastic Net:  0.9752758
```

```r
# Plot the ROC curve
roc_performance_enm <- performance(enm_pred, "tpr", "fpr")
plot(roc_performance_enm, main = "ROC Curve for Logistic Regression with Elastic Net", colorize = TRUE)

# Plot ROC curve for the SVM model
roc_performance_svm <- performance(svm_pred, "tpr", "fpr")
plot(roc_performance_svm, main = "ROC Curve for SVM", colorize = TRUE,add=TRUE)
abline(a = 0, b = 1, lty = 2, col = "gray")  # Adding a diagonal line for reference
```



Question 2: SVM with Kernel Trick (45 points)

This problem involves the `OJ` data set which is part of the `ISLR2` package. We create a training set containing a random sample of 800 observations, and a test set containing the remaining observations. In the dataset, `Purchase` variable is the output variable and it indicates whether a customer purchased Citrus Hill or Minute Maid Orange Juice. For the details of the datset you can refer to its help file.

```
  library(ISLR2)
```

```
## Warning:   'ISLR2' R 4.3.3
```

```
  data("OJ")
  set.seed(7)
  id=sample(nrow(OJ),800)
  train=OJ[id,]
  test=OJ[-id,]
```

a. [15 pts]** Fit a (linear) support vector machine by using `svm` function to the training data using `cost`= 0.01 and using all the input variables. Provide the training and test errors.

Answer:

```
library(e1071)

# Fit a linear SVM with a specified cost of 0.01
svm_linear <- svm(Purchase ~ ., data = train, kernel = "linear", cost = 0.01)

# Predictions and error calculation
train_pred <- predict(svm_linear, train)
test_pred <- predict(svm_linear, test)
train_error <- mean(train_pred != train$Purchase)
test_error <- mean(test_pred != test$Purchase)

# Print out the errors
cat("Training Error: ", train_error, "\n")
```

```
## Training Error:  0.17
```

```
cat("Test Error: ", test_error, "\n")
```

```
## Test Error:  0.162963
```

b. [15 pts]** Use the `tune()` function to select an optimal cost, C in the set of $\{0.01, 0.1, 1, 2, 5, 7, 10\}$. Compute the training and test errors using the best value for cost.

Answer:

```r
# Set the range of cost values
cost_values <- c(0.01, 0.1, 1, 2, 5, 7, 10)

# Tune the SVM model
tuned_svm <- tune(svm,
                  train.x = Purchase ~ .,
                  data = train,
                  kernel = "linear",
                  ranges = list(cost = cost_values))

# Best model, training and test error
best_svm <- tuned_svm$best.model
best_train_pred <- predict(best_svm, train)
best_test_pred <- predict(best_svm, test)
best_train_error <- mean(best_train_pred != train$Purchase)
best_test_error <- mean(best_test_pred != test$Purchase)

# Print out the results
cat("Best Cost: ", tuned_svm$best.parameters$cost, "\n")
```

```
## Best Cost:  1
```

```r
cat("Training Error with Best Cost: ", best_train_error, "\n")
```

```
## Training Error with Best Cost:  0.1675
```

```r
cat("Test Error with Best Cost: ", best_test_error, "\n")
```

```
## Test Error with Best Cost:  0.1518519
```

c. [15 pts]** Repeat parts 1 and 2 using a support vector machine with `radial` and `polynomial` (with degree 2) kernel. Use the default value for `gamma` in the `radial` kernel. Comment on your results from parts b and c.

Answer:

```r
# Tuning radial kernel
tuned_radial <- tune(svm,
                     train.x = Purchase ~ .,
                     data = train,
                     kernel = "radial",
                     ranges = list(cost = cost_values))

# Best model from radial tuning
best_radial <- tuned_radial$best.model
best_radial_train_pred <- predict(best_radial, train)
best_radial_test_pred <- predict(best_radial, test)
best_radial_train_error <- mean(best_radial_train_pred != train$Purchase)
best_radial_test_error <- mean(best_radial_test_pred != test$Purchase)
```

```r
# Tuning polynomial kernel
tuned_poly <- tune(svm,
                   train.x = Purchase ~ .,
                   data = train,
                   kernel = "polynomial",
                   degree = 2,
                   ranges = list(cost = cost_values))

# Best model from polynomial tuning
best_poly <- tuned_poly$best.model
best_poly_train_pred <- predict(best_poly, train)
best_poly_test_pred <- predict(best_poly, test)
best_poly_train_error <- mean(best_poly_train_pred != train$Purchase)
best_poly_test_error <- mean(best_poly_test_pred != test$Purchase)

# Output results
cat("Radial Kernel Best Cost: ", tuned_radial$best.parameters$cost, "\n")
```

```
## Radial Kernel Best Cost:  1
```

```r
cat("Radial Kernel Training Error: ", best_radial_train_error, "\n")
```

```
## Radial Kernel Training Error:  0.1575
```

```r
cat("Radial Kernel Test Error: ", best_radial_test_error, "\n")
```

```
## Radial Kernel Test Error:  0.1481481
```

```r
cat("Polynomial Kernel Best Cost: ", tuned_poly$best.parameters$cost, "\n")
```

```
## Polynomial Kernel Best Cost:  5
```

```r
cat("Polynomial Kernel Training Error: ", best_poly_train_error, "\n")
```

```
## Polynomial Kernel Training Error:  0.16375
```

```r
cat("Polynomial Kernel Test Error: ", best_poly_test_error, "\n")
```

```
## Polynomial Kernel Test Error:  0.1592593
```

The linear kernel SVM model performed best with a cost of 1. This indicates a moderate level of penalization for misclassification that balances bias and variance reasonably well, reducing overfitting. The training error is slightly higher than the test error, which is relatively unusual but may suggest that the model, while slightly more generalized, might also be underfitting the training data.

The radial kernel SVM also identifies the cost of 1 as optimal, which is consistent with the linear kernel's result. This cost level appears to maintain a good balance for this dataset across different kernel functions. Both training and test errors are relatively lower compared to the linear kernel, suggesting a better fit. The

radial kernel is adept at handling non-linear relationships, which might be indicative of the relationships among variables in the OJ dataset.

The polynomial kernel, with a degree of 2, performs best at a higher cost value of 5. This suggests a need for a stronger penalty against misclassifications to manage the complexity introduced by the polynomial transformations of the features. Errors for the polynomial kernel are slightly higher compared to the radial kernel, indicating it might be slightly less effective at modeling this particular dataset or potentially overfitting due to the increased complexity of the model.

Among the kernels, the radial kernel demonstrates the lowest test error, making it potentially the most suitable model for this dataset, assuming the goal is to minimize misclassification on unseen data. The polynomial kernel, despite its higher complexity, does not outperform the simpler radial kernel, suggesting that the additional complexity does not capture the underlying patterns more effectively than the radial basis function.