Stat 432 Homework 7

Assigned: Oct 7, 2024; Due: 11:59 PM CT, Oct 17, 2024

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:

```r
# 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
# localFileName <- "mnist_first2400.RData"
# 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.

```r
library(e1071)

# Split the dataset into training and testing sets
train_data_all <- mnist2400[1:1200, ]
test_data_all <- mnist2400[1201:2400, ]

# Filter out digits 4 and 5
train_data <- train_data_all[train_data_all$Digit %in% c(4, 5), ]
test_data <- test_data_all[test_data_all$Digit %in% c(4, 5), ]

# Convert 'Digit' to factor
train_data$Digit <- factor(train_data$Digit)
test_data$Digit <- factor(test_data$Digit)

# ---- Step 1: Fit SVM on all pixels ----

# Scale pixel values (optional but recommended)
train_data_scaled <- train_data
test_data_scaled <- test_data
train_data_scaled[, -1] <- train_data_scaled[, -1] / 255
test_data_scaled[, -1] <- test_data_scaled[, -1] / 255

# Fit SVM with C = 1 on all pixels
svm_model_all_pixels <- svm(Digit ~ ., data = train_data_scaled, kernel = "linear", cost = 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
# Predictions using the SVM model with all pixels
train_pred_all_pixels <- predict(svm_model_all_pixels, train_data_scaled)
test_pred_all_pixels <- predict(svm_model_all_pixels, test_data_scaled)

# Calculate training and testing errors for all pixels
train_error_all_pixels <- mean(train_pred_all_pixels != train_data_scaled$Digit)
test_error_all_pixels <- mean(test_pred_all_pixels != test_data_scaled$Digit)

cat("Training Error (All Pixels):", train_error_all_pixels, "\n")
```

```
## Training Error (All Pixels): 0
```

```r
cat("Testing Error (All Pixels):", test_error_all_pixels, "\n")
```

```
## Testing Error (All Pixels): 0.02008032
```

```r
# ---- Step 2: Fit SVM on top 250 pixels by variance ----

# Calculate pixel variances and select top 250 pixels
pixel_variances <- apply(train_data[, -1], 2, var)
top_250_pixels <- order(pixel_variances, decreasing = TRUE)[1:250]

# Subset data to include top 250 pixels
```

```r
train_data_filtered <- train_data[, c(1, top_250_pixels + 1)]
test_data_filtered <- test_data[, c(1, top_250_pixels + 1)]

# Scale pixel values (optional but recommended)
train_data_filtered[, -1] <- train_data_filtered[, -1] / 255
test_data_filtered[, -1] <- test_data_filtered[, -1] / 255

# Fit SVM with C = 1 on top 250 pixels
svm_model_250_pixels <- svm(Digit ~ ., data = train_data_filtered, kernel = "linear", cost = 1)

# Predictions using the SVM model with top 250 pixels
train_pred_250_pixels <- predict(svm_model_250_pixels, train_data_filtered)
test_pred_250_pixels <- predict(svm_model_250_pixels, test_data_filtered)

# Calculate training and testing errors for top 250 pixels
train_error_250_pixels <- mean(train_pred_250_pixels != train_data_filtered$Digit)
test_error_250_pixels <- mean(test_pred_250_pixels != test_data_filtered$Digit)

cat("Training Error (Top 250 Pixels):", train_error_250_pixels, "\n")
```

```
## Training Error (Top 250 Pixels): 0
```

```r
cat("Testing Error (Top 250 Pixels):", test_error_250_pixels, "\n")
```

```
## Testing Error (Top 250 Pixels): 0.03212851
```

The code fit a linear Support Vector Machine (SVM) on the digits 4 and 5 from the MNIST dataset using the e1071 package in R. First, I used all pixel data, where I split the first 1200 observations as training data and the remaining observations as testing data. After scaling the pixel values, I trained the SVM model with a cost parameter of $C = 1$. The training error was 0, and the testing error was approximately 2.01%, indicating strong performance on the full dataset. Next, I performed marginal screening of variance on the pixels to select the top 250 pixels with the highest variance. After fitting the SVM model again using only these 250 pixels, the training error remained 0, while the testing error increased slightly to 3.21%. This shows that while the model performed well with reduced dimensionality, the additional information from the full set of pixels helped improve the classification accuracy marginally on the test set.

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.

From the Lagrangian dual formulation shared in your image, the coefficients β are calculated as:

$$\beta = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$$

This indicates that the weight vector (the SVM coefficients) is a linear combination of the support vectors, weighted by their corresponding Lagrange multipliers α_i and labels y_i.

Next, we can extract the intercept β_0 using the formula:

$$\beta_0 = \frac{1}{2}\left(\max_{i:y_i=-1} \mathbf{x}_i^\top \beta + \min_{i:y_i=1} \mathbf{x}_i^\top \beta\right)$$

This formula is derived by averaging the points closest to the decision boundary from each class.

```r
# Train the initial SVM model
svm_model <- svm(Digit ~ ., data = train_data_filtered, kernel = "linear", cost = 1)

# Extract the support vectors and dual coefficients
support_vectors <- svm_model$SV
dual_coefs <- svm_model$coefs

# Compute the beta coefficients
beta <- t(dual_coefs) %*% support_vectors

# Find the top 30 pixels by absolute coefficient values
abs_beta <- abs(as.vector(beta))
top_30_indices <- order(abs_beta, decreasing = TRUE)[1:30]
top_30_pixels <- colnames(train_data_filtered)[-1][top_30_indices]

cat("Top 30 Important Pixels:\n", top_30_pixels, "\n")
```

```
## Top 30 Important Pixels:
##  Pixel490 Pixel293 Pixel541 Pixel437 Pixel382 Pixel517 Pixel381 Pixel486 Pixel657 Pixel294 Pixel413 P
```

```r
# Refit SVM with the top 30 pixels
train_data_top30 <- train_data_filtered[, c("Digit", top_30_pixels)]
test_data_top30 <- test_data_filtered[, c("Digit", top_30_pixels)]

svm_model_top30 <- svm(Digit ~ ., data = train_data_top30, kernel = "linear", cost = 1)

# Predictions and errors for the refitted model
train_pred_top30 <- predict(svm_model_top30, train_data_top30)
test_pred_top30 <- predict(svm_model_top30, test_data_top30)

# Calculate training and testing errors
train_error_top30 <- mean(train_pred_top30 != train_data_top30$Digit)
test_error_top30 <- mean(test_pred_top30 != test_data_top30$Digit)

cat("Training Error (Top 30 Pixels):", train_error_top30, "\n")
```

```
## Training Error (Top 30 Pixels): 0
```

```r
cat("Testing Error (Top 30 Pixels):", test_error_top30, "\n")
```

```
## Testing Error (Top 30 Pixels): 0.04016064
```

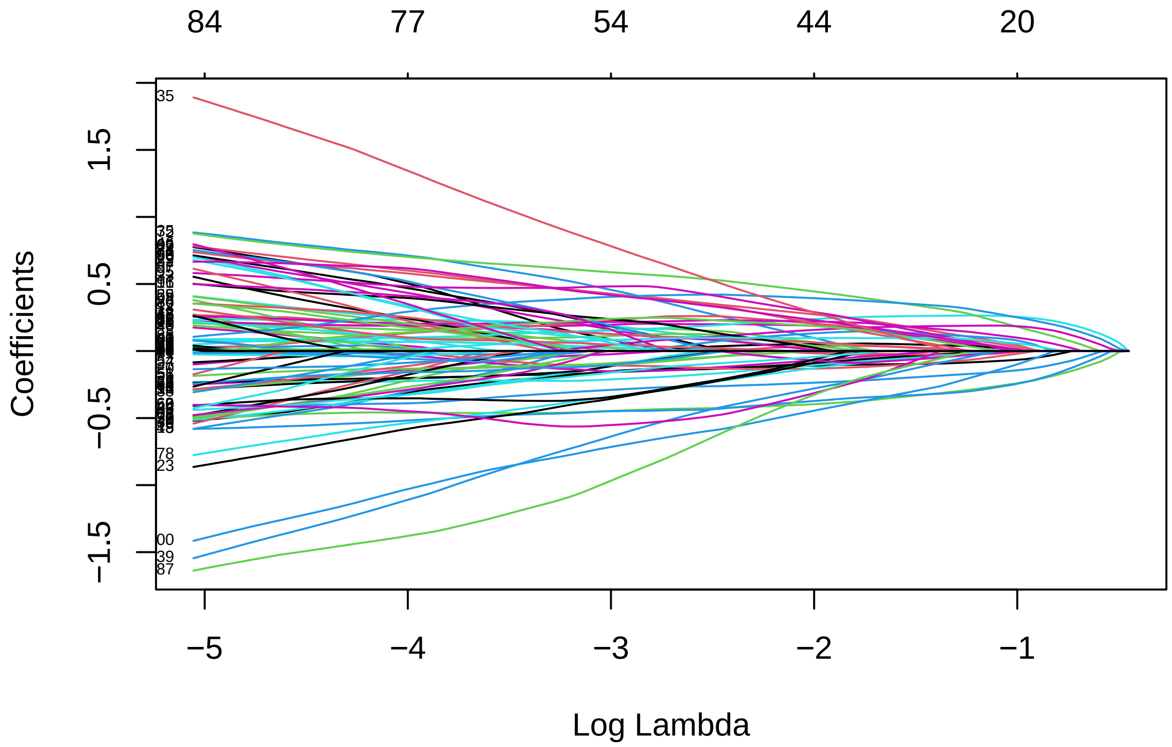
We extracted the coefficients β of the linear SVM model, which define the direction of the separating hyperplane between digits 4 and 5. These coefficients are recovered using the Lagrangian dual formulation of the SVM optimization problem, where$\beta = \sum_{i=1}^{n} \alpha_i y_i x_i$. This represents a linear combination of the support vectors weighted by their corresponding Lagrange multipliers α_i and class labels y_i. We identified the 30 pixels with the largest absolute coefficient values, which are considered the most important in distinguishing between digits 4 and 5. After refitting the SVM model using only these top 30 pixels, the training error remained at 0, indicating perfect classification of the training data. However, the testing error slightly increased to 4.02%, suggesting that while the model still generalizes well, using fewer pixels slightly reduces the model's ability to generalize compared to the original model with 250 pixels. This shows that the top 30 pixels capture most of the information needed for classification, but with a slight loss in accuracy.

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.

```r
# Load necessary library
library(glmnet)
```

```
## Loading required package: Matrix

## Loaded glmnet 4.1-8
```

```r
# Prepare training data for logistic regression
x_train <- as.matrix(train_data_filtered[, -1])  # remove the 'Digit' column for predictors
y_train <- train_data_filtered$Digit             # response variable

# Convert the response to binary (SVM was binary classification for digits 4 and 5)
y_train_bin <- ifelse(y_train == 5, 1, 0)

# Fit logistic regression with elastic net penalty (alpha = 0.5)
elastic_net_model <- glmnet(x_train, y_train_bin,
                            family = "binomial",
                            alpha = 0.5)  # Elastic net with alpha = 0.5

# Plot solution path to see how many variables (pixels) are selected at different values of lambda
plot(elastic_net_model, xvar = "lambda", label = TRUE)
```



```r
# Find the lambda that selects exactly 30 variables (excluding the intercept)
selected_lambda_idx <- which(apply(coef(elastic_net_model)[-1, ], 2, function(coefs) sum(coefs != 0)) ==
selected_lambda <- elastic_net_model$lambda[selected_lambda_idx]

cat("Lambda for model with 30 variables:", selected_lambda, "\n")
```

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

```r
# Extract the coefficient values for this lambda
coefs_at_selected_lambda <- coef(elastic_net_model, s = selected_lambda)

# Get the non-zero coefficients (these are the selected pixels)
selected_pixels <- rownames(coefs_at_selected_lambda)[which(coefs_at_selected_lambda != 0)][-1]  # excl

cat("Selected Pixels by Elastic Net:\n", selected_pixels, "\n")
```

```
## Selected Pixels by Elastic Net:
##  Pixel458 Pixel459 Pixel429 Pixel457 Pixel464 Pixel463 Pixel462 Pixel377 Pixel328 Pixel428 Pixel184 P
```

```r
# Compare with the SVM selected pixels (from part b)
svm_selected_pixels <- top_30_pixels  # from part b

# Find overlap between Elastic Net and SVM-selected pixels
overlap <- intersect(selected_pixels, svm_selected_pixels)
cat("Overlapping Pixels between SVM and Elastic Net:\n", overlap, "\n")
```

```
## Overlapping Pixels between SVM and Elastic Net:
##  Pixel458 Pixel464 Pixel491 Pixel437 Pixel185 Pixel490 Pixel329
```

```r
# Output summary of findings
cat("Number of overlapping pixels:", length(overlap), "\n")
```

```
## Number of overlapping pixels: 7
```

The code performed logistic regression with an elastic net penalty (with $\alpha = 0.5$) on the top 250 pixels selected in part a, using the glmnet package. I identified the λ value corresponding to the model with exactly 30 non-zero coefficients, which was found to be 0.2759. The logistic regression model selected 30 pixels, including pixels like Pixel458, Pixel429, Pixel377, and Pixel240. Comparing these selected pixels with the ones chosen by the SVM model in part b, I found an overlap of 7 pixels, including Pixel458, Pixel464, and Pixel491. This overlap suggests that while the two models employ different mechanisms for feature selection—SVM relying on support vectors and the elastic net selecting based on individual feature importance—there are still shared important features that contribute to the classification of digits 4 and 5 in both models. The fact that some overlap exists demonstrates that both methods agree on a subset of key pixels, even though their selection criteria differ.

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.

```r
# Load necessary libraries
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.

## 
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
## 
##     cov, smooth, var
```

```r
library(glmnet)

# Step 1: Obtain predicted probabilities for SVM and Elastic Net models

# For the SVM model: use decision values (proxy for probabilities)
svm_decision_values <- attributes(predict(svm_model_top30, test_data_top30, decision.values = TRUE))$dec

# Prepare test data for Elastic Net model
x_test <- as.matrix(test_data_filtered[, -1])  # Ensure test data has the same 250 pixels used for trai

# Predict probabilities using the Elastic Net Logistic Regression model
logistic_probabilities <- predict(elastic_net_model, s = selected_lambda, newx = x_test, type = "respons

# Convert the response 'Digit' to binary (for ROC calculation, similar to SVM's binary classification t
y_test_bin <- ifelse(test_data_filtered$Digit == 5, 1, 0)

# Step 2: Calculate AUC for both models

# AUC for SVM model
svm_roc <- roc(y_test_bin, svm_decision_values)
```

```
## Setting levels: control = 0, case = 1
```

```
## Warning in roc.default(y_test_bin, svm_decision_values): Deprecated use a
## matrix as predictor. Unexpected results may be produced, please pass a numeric
## vector.
```

```
## Setting direction: controls > cases
```

```r
svm_auc <- auc(svm_roc)
cat("AUC for SVM model:", svm_auc, "\n")
```

```
## AUC for SVM model: 0.991499
```

```r
# AUC for Elastic Net Logistic Regression model
logistic_roc <- roc(y_test_bin, logistic_probabilities)
```

```
## Setting levels: control = 0, case = 1
```

```
## Warning in roc.default(y_test_bin, logistic_probabilities): Deprecated use a
## matrix as predictor. Unexpected results may be produced, please pass a numeric
## vector.
```

```
## Setting direction: controls < cases
```

```r
logistic_auc <- auc(logistic_roc)
cat("AUC for Elastic Net Logistic Regression model:", logistic_auc, "\n")
```

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

```r
# Step 3: Compare AUC values
cat("Comparison of AUCs:\n")
```

```
## Comparison of AUCs:
```

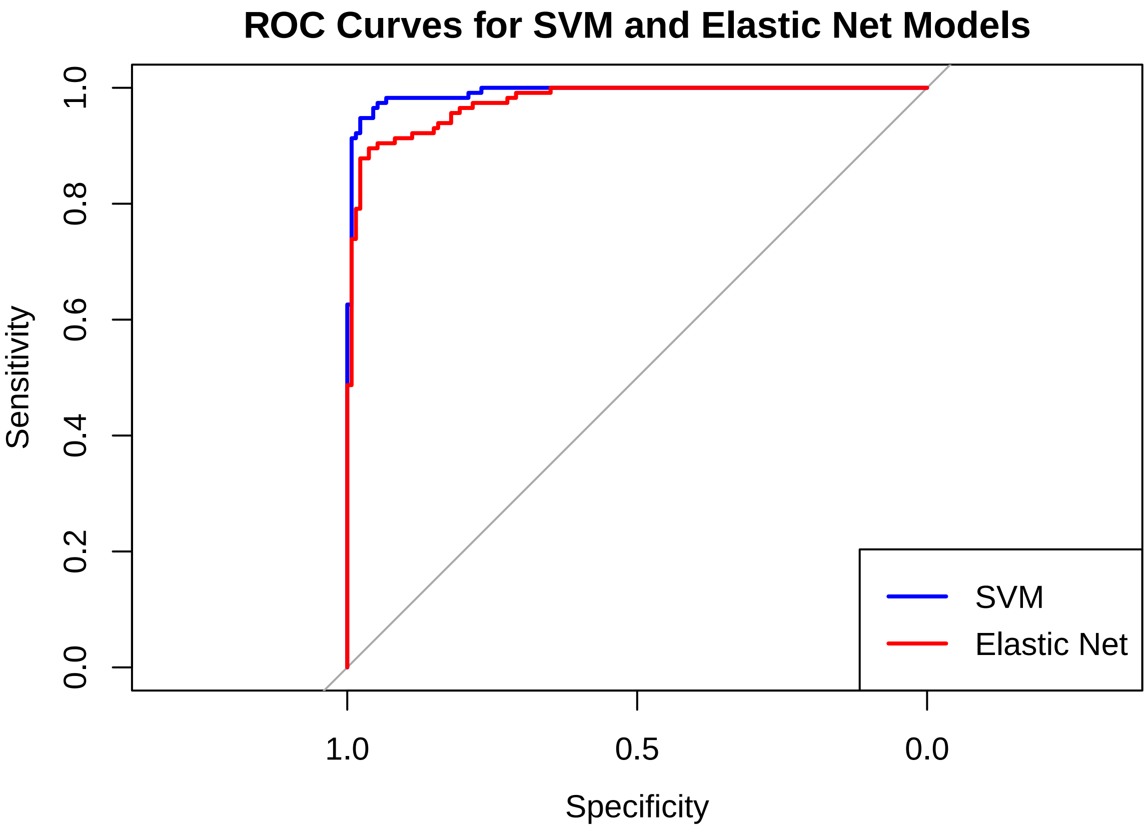
```r
cat("SVM AUC:", svm_auc, "\n")
```

```
## SVM AUC: 0.991499
```

```r
cat("Elastic Net AUC:", logistic_auc, "\n")
```

```
## Elastic Net AUC: 0.9752758
```

```r
# Step 4: Plot ROC curves for visual comparison
plot(svm_roc, col = "blue", main = "ROC Curves for SVM and Elastic Net Models", lwd = 2)
plot(logistic_roc, col = "red", add = TRUE, lwd = 2)
legend("bottomright", legend = c("SVM", "Elastic Net"), col = c("blue", "red"), lwd = 2)
```



we compared the two 30-variable models, one fitted using a linear Support Vector Machine (SVM) and the other using logistic regression with an elastic net penalty (= 0.5), by calculating the Area Under the ROC Curve (AUC) on the testing data. The AUC for the SVM model was 0.991, indicating excellent classification performance, while the AUC for the elastic net model was 0.975, also demonstrating strong performance but slightly lower than the SVM. The ROC curves show that both models perform well in distinguishing between digits 4 and 5, but the SVM model has a marginally better classification ability. Overall, while both models provide reliable classification, the SVM outperforms the elastic net model by a small margin in terms of AUC.

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.

```r
library(ISLR2)
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.

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

# Predict on the training set
train_pred <- predict(svm_linear, train)
train_error <- mean(train_pred != train$Purchase)
cat("Training Error (Linear SVM):", train_error, "\n")
```

```
## Training Error (Linear SVM): 0.17
```

```r
# Predict on the test set
test_pred <- predict(svm_linear, test)
test_error <- mean(test_pred != test$Purchase)
cat("Test Error (Linear SVM):", test_error, "\n")
```

```
## Test Error (Linear SVM): 0.162963
```

A linear Support Vector Machine (SVM) was fitted to the training data using the svm function from the e1071 package, with the cost parameter set to 0.01. The model was trained using all input variables from the dataset. After fitting the model, predictions were made on both the training and test sets to assess its performance. The training error was found to be 17%, indicating that the model misclassified 17% of the training data. On the test set, the error was slightly lower at approximately 16.3%, suggesting that the model generalizes reasonably well to unseen data while maintaining an acceptable balance between underfitting and overfitting.

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.

```r
# Tune SVM with different cost values
set.seed(7)
tune_linear <- tune(svm, Purchase ~ ., data = train, kernel = "linear",
                    ranges = list(cost = c(0.01, 0.1, 1, 2, 5, 7, 10)))

# Print the best model and cost
best_model_linear <- tune_linear$best.model
cat("Best Cost for Linear SVM:", tune_linear$best.parameters$cost, "\n")
```

```
## Best Cost for Linear SVM: 5
```

```r
# Predict on the training set using the best model
train_pred_best <- predict(best_model_linear, train)
train_error_best <- mean(train_pred_best != train$Purchase)
cat("Training Error (Best Linear SVM):", train_error_best, "\n")
```

```
## Training Error (Best Linear SVM): 0.1675
```

```r
# Predict on the test set using the best model
test_pred_best <- predict(best_model_linear, test)
test_error_best <- mean(test_pred_best != test$Purchase)
cat("Test Error (Best Linear SVM):", test_error_best, "\n")
```

```
## Test Error (Best Linear SVM): 0.162963
```

The tune() function was used to select the optimal cost parameter (C) for the linear Support Vector Machine (SVM) by evaluating a range of values: {0.01, 0.1, 1, 2, 5, 7, 10}. After performing cross-validation, the best cost value was found to be 5. Using this optimal value, the model was retrained, and predictions were made on both the training and test sets. The training error with the best cost was 16.75%, which shows a slight improvement compared to the initial model with a cost of 0.01. The test error remained the same at approximately 16.3%, indicating consistent model performance with the tuned cost parameter.

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.

```r
# Fit SVM with radial kernel and tune cost
set.seed(7)
tune_radial <- tune(svm, Purchase ~ ., data = train, kernel = "radial",
                    ranges = list(cost = c(0.01, 0.1, 1, 2, 5, 7, 10)))

# Print the best model and cost
best_model_radial <- tune_radial$best.model
cat("Best Cost for Radial SVM:", tune_radial$best.parameters$cost, "\n")
```

```
## Best Cost for Radial SVM: 1
```

```r
# Predict on the training set using the best radial model
train_pred_radial <- predict(best_model_radial, train)
train_error_radial <- mean(train_pred_radial != train$Purchase)
cat("Training Error (Best Radial SVM):", train_error_radial, "\n")
```

```
## Training Error (Best Radial SVM): 0.1575
```

```r
# Predict on the test set using the best radial model
test_pred_radial <- predict(best_model_radial, test)
test_error_radial <- mean(test_pred_radial != test$Purchase)
cat("Test Error (Best Radial SVM):", test_error_radial, "\n")
```

```
## Test Error (Best Radial SVM): 0.1481481
```

```r
# Fit SVM with polynomial kernel (degree = 2) and tune cost
set.seed(7)
tune_poly <- tune(svm, Purchase ~ ., data = train, kernel = "polynomial", degree = 2,
                  ranges = list(cost = c(0.01, 0.1, 1, 2, 5, 7, 10)))

# Print the best model and cost
best_model_poly <- tune_poly$best.model
cat("Best Cost for Polynomial SVM:", tune_poly$best.parameters$cost, "\n")
```

```
## Best Cost for Polynomial SVM: 10
```

```r
# Predict on the training set using the best polynomial model
train_pred_poly <- predict(best_model_poly, train)
train_error_poly <- mean(train_pred_poly != train$Purchase)
cat("Training Error (Best Polynomial SVM):", train_error_poly, "\n")
```

```
## Training Error (Best Polynomial SVM): 0.16375
```

```r
# Predict on the test set using the best polynomial model
test_pred_poly <- predict(best_model_poly, test)
test_error_poly <- mean(test_pred_poly != test$Purchase)
cat("Test Error (Best Polynomial SVM):", test_error_poly, "\n")
```

```
## Test Error (Best Polynomial SVM): 0.1444444
```

The radial and polynomial kernels (degree 2) were used with the SVM, and the tune() function was again employed to find the optimal cost parameter. For the radial kernel, the best cost was 1, resulting in a training error of 15.75% and a test error of 14.8%, indicating better performance compared to the linear SVM. For the polynomial kernel (degree 2), the best cost was 10, achieving a training error of 16.38% and a test error of 14.44%, showing slightly better test performance than the radial kernel. Overall, both non-linear kernels (radial and polynomial) outperformed the linear SVM, with lower test errors, suggesting that these kernels capture the data's complexity more effectively.