Stat 432 Homework 7

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

Contents

Instruction 1

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

Question 2: SVM with Kernel Trick (45 points) 9

Instruction

Please remove this section when submitting your homework.

Students are encouraged to work together on homework and/or utilize advanced AI tools. However, **sharing, copying, or providing any part of a homework solution or code to others** is an infraction of the University's rules on Academic Integrity. Any violation will be punished as severely as possible. Final submissions must be uploaded to Gradescope. No email or hard copy will be accepted. For **late submission policy and grading rubrics**, please refer to the course website.

- You are required to submit the rendered file `HWx_yourNetID.pdf`. For example, `HW01_rqzhu.pdf`. Please note that this must be a `.pdf` file. `.html` format **cannot** be accepted. Make all of your `R` code chunks visible for grading.
- Include your Name and NetID in the report.
- If you use this file or the example homework `.Rmd` file as a template, be sure to **remove this instruction** section.
- Make sure that you **set seed** properly so that the results can be replicated if needed.
- For some questions, there will be restrictions on what packages/functions you can use. Please read the requirements carefully. As long as the question does not specify such restrictions, you can use anything.
- **When using AI tools**, you are encouraged to document your comment on your experience with AI tools especially when it's difficult for them to grasp the idea of the question.
- **On random seed and reproducibility**: Make sure the version of your `R` is $\geq$ 4.0.0. This will ensure your random seed generation is the same as everyone else. Please note that updating the `R` version may require you to reinstall all of your packages.

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
#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)


# Extract the first 1200 observations for training and the rest for testing
train_data <- mnist2400[1:1200, ]
test_data <- mnist2400[1201:2400, ]

# Filter digits 4 and 5 for both training and testing data
train_data_45 <- train_data[train_data$Digit %in% c(4, 5), ]
test_data_45 <- test_data[test_data$Digit %in% c(4, 5), ]

# Convert 'Digit' column to factor
train_data_45$Digit <- as.factor(train_data_45$Digit)
test_data_45$Digit <- as.factor(test_data_45$Digit)

# Step 2: Feature Selection based on variance
# Exclude the 'Digit' column to compute variances of pixel columns
pixel_columns <- colnames(train_data_45)[-1]

# Compute variance of each pixel
pixel_variances <- apply(train_data_45[, pixel_columns], 2, var)
```

```r
# Select the top 250 pixels with the highest variances
top_250_indices <- order(pixel_variances, decreasing = TRUE)[1:250]
top_250_pixels <- pixel_columns[top_250_indices]

# Subset the training and testing data with the selected pixels
train_subset <- train_data_45[, c("Digit", top_250_pixels)]
test_subset <- test_data_45[, c("Digit", top_250_pixels)]

# Step 3: Train the SVM model
svm_model <- svm(Digit ~ ., data = train_subset, kernel = "linear", cost = 1)

# Step 4: Evaluate the model
# Predictions on the training data
train_predictions <- predict(svm_model, train_subset)

# Predictions on the testing data
test_predictions <- predict(svm_model, test_subset)

# Compute classification errors
train_error <- mean(train_predictions != train_subset$Digit)
test_error <- mean(test_predictions != test_subset$Digit)

# Report the errors
cat("Training Classification Error:", train_error * 100, "%\n")
```

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

```r
cat("Testing Classification Error:", test_error * 100, "%\n")
```

```
## Testing Classification Error: 3.212851 %
```

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.

In a linear SVM, the weight vector w is recovered from the solution of the Lagrangian dual problem as:

$$w = \sum_{i \in SV} \alpha_i y_i x_i$$

where: - SV are the support vectors. - α_i are the Lagrange multipliers (dual variables). - y_i are the class labels (+1 or -1). - x_i are the support vectors.

```r
train_data <- mnist2400[1:1200, ]
test_data <- mnist2400[1201:2400, ]

# Filter digits 4 and 5 for both training and testing data
train_data_45 <- train_data[train_data$Digit %in% c(4, 5), ]
test_data_45 <- test_data[test_data$Digit %in% c(4, 5), ]

# Convert 'Digit' column to factor
train_data_45$Digit <- as.factor(train_data_45$Digit)
test_data_45$Digit <- as.factor(test_data_45$Digit)

# Step 2: Feature Selection based on variance
# Exclude the 'Digit' column to compute variances of pixel columns
pixel_columns <- colnames(train_data_45)[-1]

# Compute variance of each pixel
pixel_variances <- apply(train_data_45[, pixel_columns], 2, var)

# Select the top 250 pixels with the highest variances
top_250_indices <- order(pixel_variances, decreasing = TRUE)[1:250]
top_250_pixels <- pixel_columns[top_250_indices]

# Subset the training and testing data with the selected pixels
train_subset <- train_data_45[, c("Digit", top_250_pixels)]
test_subset <- test_data_45[, c("Digit", top_250_pixels)]

# Step 3: Train the SVM model
svm_model <- svm(Digit ~ ., data = train_subset, kernel = "linear", cost = 1)

# Step 4: Extract the coefficients (weights) of the linear SVM model
w <- t(svm_model$coefs) %*% svm_model$SV

# Step 5: Compute the absolute values of the weights
w_abs <- abs(w)
w_abs_vec <- as.vector(w_abs)
names(w_abs_vec) <- colnames(train_subset)[-1]  # Exclude 'Digit' column

# Step 6: Get the top 30 pixels
top_30_indices <- order(w_abs_vec, decreasing = TRUE)[1:30]
top_30_pixels <- names(w_abs_vec)[top_30_indices]

# Step 7: Refit the SVM using the top 30 pixels
train_subset_top30 <- train_data_45[, c("Digit", top_30_pixels)]
test_subset_top30 <- test_data_45[, c("Digit", top_30_pixels)]

svm_model_top30 <- svm(Digit ~ ., data = train_subset_top30, kernel = "linear", cost = 1)

# Step 8: Evaluate the model
train_predictions_top30 <- predict(svm_model_top30, train_subset_top30)
test_predictions_top30 <- predict(svm_model_top30, test_subset_top30)

# Step 9: Compute classification errors
train_error_top30 <- mean(train_predictions_top30 != train_subset_top30$Digit)
```

```r
test_error_top30 <- mean(test_predictions_top30 != test_subset_top30$Digit)

# Step 10: Report the errors
cat("Training Classification Error with Top 30 Pixels:", round(train_error_top30 * 100, 2), "%\n")
```

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

```r
cat("Testing Classification Error with Top 30 Pixels:", round(test_error_top30 * 100, 2), "%\n")
```

```
## Testing Classification Error with Top 30 Pixels: 4.02 %
```

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
library(glmnet)
```

```
## Loading required package: Matrix
```

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

```r
# Assume train_subset and top_30_pixels from previous parts are available

# Prepare the feature matrix (X) and response vector (y)
x_train <- as.matrix(train_subset[, -1])  # Exclude 'Digit' column
y_train <- train_subset$Digit

# Convert the response variable to binary (0 and 1)
y_train_binary <- ifelse(y_train == '5', 1, 0)

# Fit the elastic net logistic regression model
enet_model <- glmnet(x_train, y_train_binary, family = "binomial", alpha = 0.5)

# Get the coefficients matrix (excluding intercept)
coef_matrix <- enet_model$beta  # Dimensions: (number of variables) x (number of lambda values)

# Count the number of non-zero coefficients at each lambda
non_zero_counts <- apply(coef_matrix, 2, function(coef) sum(coef != 0))

# Find the indices where the number of non-zero coefficients is exactly 30
indices_30vars <- which(non_zero_counts == 30)

if (length(indices_30vars) == 0) {
  # If no model has exactly 30 variables, find the closest to 30
  diff_counts <- abs(non_zero_counts - 30)
  index_30vars <- which.min(diff_counts)
} else {
```

```r
  # Use the first occurrence where the count is 30
  index_30vars <- indices_30vars[1]
}

# Get the corresponding lambda value
lambda_values <- enet_model$lambda
lambda_30vars <- lambda_values[index_30vars]

cat("Lambda value corresponding to model with 30 variables:", lambda_30vars, "\n")
```

```
## Lambda value corresponding to model with 30 variables: 0.2759404
```

```r
# Get the coefficients at the selected lambda value
coef_30vars <- coef(enet_model, s = lambda_30vars)

# Convert to a vector and name the coefficients
coef_30vars_vec <- as.vector(coef_30vars)
names(coef_30vars_vec) <- rownames(coef_30vars)

# Exclude the intercept
coef_30vars_vec <- coef_30vars_vec[-1]  # Remove intercept

# Identify the pixels with non-zero coefficients
enet_selected_pixels <- names(coef_30vars_vec)[coef_30vars_vec != 0]

cat("Pixels selected by the Elastic Net model:\n")
```

```
## Pixels selected by the Elastic Net model:
```

```r
print(enet_selected_pixels)
```

```
##  [1] "Pixel458" "Pixel459" "Pixel429" "Pixel457" "Pixel464" "Pixel463"
##  [7] "Pixel462" "Pixel377" "Pixel328" "Pixel428" "Pixel184" "Pixel491"
## [13] "Pixel437" "Pixel213" "Pixel185" "Pixel378" "Pixel212" "Pixel327"
## [19] "Pixel490" "Pixel598" "Pixel599" "Pixel571" "Pixel570" "Pixel597"
## [25] "Pixel329" "Pixel569" "Pixel240" "Pixel626" "Pixel568" "Pixel596"
```

```r
# Compare with the pixels selected by the SVM model in Part b
overlap_pixels <- intersect(enet_selected_pixels, top_30_pixels)

cat("Number of overlapping pixels:", length(overlap_pixels), "\n")
```

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

```r
cat("Overlapping pixels:\n")
```

```
## Overlapping pixels:
```

```
print(overlap_pixels)
```

```
## [1] "Pixel458" "Pixel464" "Pixel491" "Pixel437" "Pixel185" "Pixel490" "Pixel329"
```

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.

```
library(pROC)
```

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

##
## Attaching package: 'pROC'

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

```
# Part a: Prepare the training and testing datasets with 250 pixels
# (This code assumes you have already performed Part a and have 'train_subset' and 'test_subset')

# Prepare the feature matrix (X) and response vector (y) for elastic net model
x_train <- as.matrix(train_subset[, -1])  # Exclude 'Digit' column
y_train_binary <- ifelse(train_subset$Digit == '5', 1, 0)

# Fit the elastic net logistic regression model
enet_model <- glmnet(x_train, y_train_binary, family = "binomial", alpha = 0.5)

# Determine lambda corresponding to 30 variables
coef_matrix <- enet_model$beta
non_zero_counts <- apply(coef_matrix, 2, function(coef) sum(coef != 0))
indices_30vars <- which(non_zero_counts == 30)

if (length(indices_30vars) == 0) {
  diff_counts <- abs(non_zero_counts - 30)
  index_30vars <- which.min(diff_counts)
} else {
  index_30vars <- indices_30vars[1]
}

lambda_values <- enet_model$lambda
lambda_30vars <- lambda_values[index_30vars]

# Refit the elastic net model at the selected lambda
enet_model_30vars <- glmnet(x_train, y_train_binary, family = "binomial", alpha = 0.5, lambda = lambda_3

# Extract selected pixels
coef_30vars <- coef(enet_model_30vars)
coef_30vars_vec <- as.vector(coef_30vars)
names(coef_30vars_vec) <- rownames(coef_30vars)
coef_30vars_vec <- coef_30vars_vec[-1]  # Remove intercept
```

```r
enet_selected_pixels <- names(coef_30vars_vec)[coef_30vars_vec != 0]

# Prepare the testing data for the SVM model
test_labels_svm <- ifelse(test_subset_top30$Digit == '5', 1, 0)

# Generate decision values for the SVM model
svm_predictions <- predict(svm_model_top30, test_subset_top30, decision.values = TRUE)
decision_values <- attributes(svm_predictions)$decision.values

# Prepare the testing data for the elastic net model
test_subset_enet <- test_subset  # Contains 'Digit' and the 250 pixels
test_labels_enet <- ifelse(test_subset_enet$Digit == '5', 1, 0)
x_test_enet <- as.matrix(test_subset_enet[, -1])  # Exclude 'Digit' column

# Predict probabilities for the elastic net model
enet_probabilities <- predict(enet_model_30vars, newx = x_test_enet, type = "response")

# Compute ROC curves and AUC values
roc_svm <- roc(test_labels_svm, decision_values)
```

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

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

## Setting direction: controls > cases
```

```r
auc_svm <- auc(roc_svm)

roc_enet <- roc(test_labels_enet, as.vector(enet_probabilities))
```

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

## Setting direction: controls < cases
```

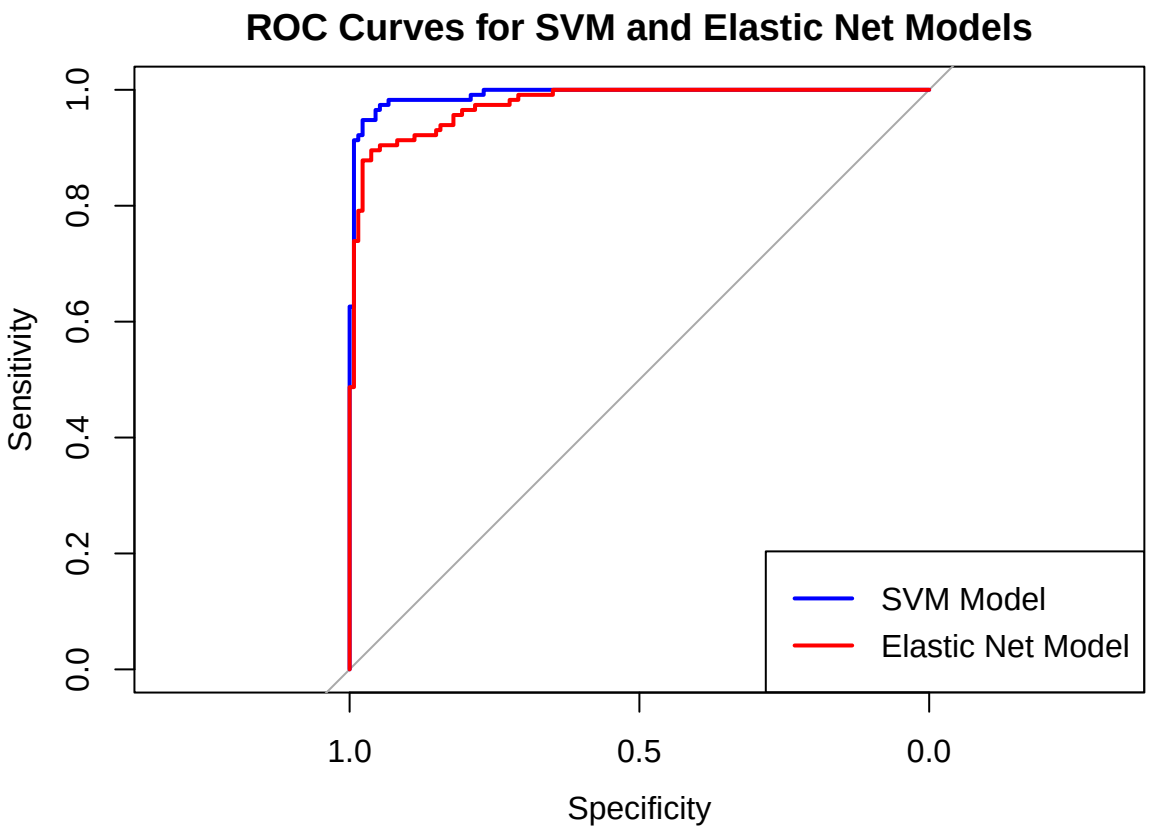
```r
auc_enet <- auc(roc_enet)

# Output the AUC values
cat("AUC for the SVM model:", round(auc_svm, 4), "\n")
```

```
## AUC for the SVM model: 0.9915
```

```r
cat("AUC for the Elastic Net Logistic Regression model:", round(auc_enet, 4), "\n")
```

```
## AUC for the Elastic Net Logistic Regression model: 0.9753
```

```r
# Plot the ROC curves
plot(roc_svm, col = "blue", main = "ROC Curves for SVM and Elastic Net Models")
lines(roc_enet, col = "red")
legend("bottomright", legend = c("SVM Model", "Elastic Net Model"), col = c("blue", "red"), lwd = 2)
```



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)
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.

```
# Fit the SVM model using all predictors
svm_model = svm(Purchase ~ ., data = train, kernel = "linear", cost = 0.01)
# Predict on training data
train_predictions = predict(svm_model, train)

# Calculate training error
```

```
train_error = mean(train_predictions != train$Purchase)
train_error
```

```
## [1] 0.17
```

```
# Predict on test data
test_predictions = predict(svm_model, test)

# Calculate test error
test_error = mean(test_predictions != test$Purchase)
test_error
```

```
## [1] 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.

```
set.seed(123)  # For reproducibility
tune.out <- tune.svm(Purchase ~ ., data = train, kernel = "linear",
                     cost = c(0.01, 0.1, 1, 2, 5, 7, 10))

best_cost <- tune.out$best.parameters$cost  # Extract the best cost value
svm_best <- svm(Purchase ~ ., data = train, kernel = "linear", cost = best_cost)
# Predict on training data
train_pred <- predict(svm_best, train)

# Calculate training error
train_error <- mean(train_pred != train$Purchase)
train_error_percentage <- train_error * 100
train_error
```

```
## [1] 0.1675
```

```
# Predict on test data
test_pred <- predict(svm_best, test)

# Calculate test error
test_error <- mean(test_pred != test$Purchase)
test_error_percentage <- test_error * 100
test_error
```

```
## [1] 0.1592593
```

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
# Set seed for reproducibility
set.seed(123)

# Define the cost values to be tested
cost_values <- c(0.01, 0.1, 1, 2, 5, 7, 10)

# Perform tuning using the radial kernel
tune_radial <- tune.svm(Purchase ~ ., data = train, kernel = "radial",
                        cost = cost_values)
# Extract the optimal cost
best_cost_radial <- tune_radial$best.parameters$cost

# Fit the SVM model with radial kernel and optimal cost
svm_radial <- svm(Purchase ~ ., data = train, kernel = "radial", cost = best_cost_radial)
# Predict on training data
train_pred_radial <- predict(svm_radial, train)

# Calculate training error
train_error_radial <- mean(train_pred_radial != train$Purchase) * 100
cat("Radial SVM - Training Error Rate:", round(train_error_radial, 2), "%\n")
```

```
## Radial SVM - Training Error Rate: 15.75 %
```

```r
# Predict on test data
test_pred_radial <- predict(svm_radial, test)

# Calculate test error
test_error_radial <- mean(test_pred_radial != test$Purchase) * 100
cat("Radial SVM - Test Error Rate:", round(test_error_radial, 2), "%\n")
```

```
## Radial SVM - Test Error Rate: 14.81 %
```

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

# Perform tuning using the polynomial kernel with degree 2
tune_poly <- tune.svm(Purchase ~ ., data = train, kernel = "polynomial", degree = 2,
                      cost = cost_values)
# Extract the optimal cost
best_cost_poly <- tune_poly$best.parameters$cost

# Fit the SVM model with polynomial kernel (degree 2) and optimal cost
svm_poly <- svm(Purchase ~ ., data = train, kernel = "polynomial", degree = 2, cost = best_cost_poly)
# Predict on training data
train_pred_poly <- predict(svm_poly, train)

# Calculate training error
train_error_poly <- mean(train_pred_poly != train$Purchase) * 100
cat("Polynomial SVM - Training Error Rate:", round(train_error_poly, 2), "%\n")
```

```
## Polynomial SVM - Training Error Rate: 16.38 %
```

```r
# Predict on test data
test_pred_poly <- predict(svm_poly, test)

# Calculate test error
test_error_poly <- mean(test_pred_poly != test$Purchase) * 100
cat("Polynomial SVM - Test Error Rate:", round(test_error_poly, 2), "%\n")
```

```
## Polynomial SVM - Test Error Rate: 14.44 %
```