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

```
## Warning: package 'e1071' was built under R version 4.3.3
```

```r
# Step 1: Subset MNIST data
mnist_filtered_1 <- mnist2400[1:1200, ]
mnist_filtered_2 <- mnist2400[1201:nrow(mnist2400), ]
# Split data into training and testing sets
train_data <- mnist_filtered_1[mnist_filtered_1$Digit %in% c(4, 5), ]
test_data <- mnist_filtered_2[mnist_filtered_2$Digit %in% c(4, 5), ]

# Step 2: Fit a linear SVM on the original data
svm_original <- svm(Digit ~ ., data = train_data, type = 'C-classification', kernel = "linear", scale =

# Make predictions on training and testing data
train_pred_original <- predict(svm_original, train_data)
test_pred_original <- predict(svm_original, test_data)

# Calculate the training and testing errors
train_error_original <- mean(train_pred_original != train_data$Digit)
test_error_original <- mean(test_pred_original != test_data$Digit)

cat("Training Error (Original): ", train_error_original, "\n")
```

```
## Training Error (Original):  0
```

```r
cat("Testing Error (Original): ", test_error_original, "\n")
```

```
## Testing Error (Original):  0.02008032
```

```r
# Step 3: Marginal variance screening to select top 250 pixels
pixel_variances <- apply(train_data[, -1], 2, var)
top_pixels <- order(pixel_variances, decreasing = TRUE)[1:250]

# Subset the data to include only these top 250 pixels
train_data_filtered <- train_data[, c(1, top_pixels + 1)]  # Include Digit column
test_data_filtered <- test_data[, c(1, top_pixels + 1)]

# Step 4: Refit the linear SVM on the filtered dataset
svm_filtered <- svm(Digit ~ ., data = train_data_filtered, type = 'C-classification', kernel = "linear"

# Make predictions on the filtered training and testing data
train_pred_filtered <- predict(svm_filtered, train_data_filtered)
test_pred_filtered <- predict(svm_filtered, test_data_filtered)

# Calculate the training and testing errors on the filtered data
train_error_filtered <- mean(train_pred_filtered != train_data_filtered$Digit)
test_error_filtered <- mean(test_pred_filtered != test_data_filtered$Digit)

# Report the results
cat("Training Error (Filtered): ", train_error_filtered, "\n")
```

```
## Training Error (Filtered):  0
```

```r
cat("Testing Error (Filtered): ", test_error_filtered, "\n")
```

```
## Testing Error (Filtered):  0.02811245
```

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.

```r
# Step 1: Extract the coefficients from the fitted linear SVM model
svm_coefs <- svm_filtered$coefs  # The coefficients for the support vectors
support_vectors <- svm_filtered$SV  # The support vectors themselves

# Calculate the overall coefficients (beta) for each pixel
# Matrix multiplication of the coefficients and the support vectors
beta <- t(svm_coefs) %*% support_vectors

# Step 2: Find the top 30 pixels with the largest absolute coefficients
# Get absolute values of the coefficients
beta_abs <- abs(beta)

# Sort by the largest absolute coefficients and get the indices of the top 30 pixels
top_30_indices <- order(beta_abs, decreasing = TRUE)[1:30]

# Step 3: Refit the SVM using only these 30 pixels
train_data_top30 <- train_data_filtered[, c(1, top_30_indices + 1)]  # Include the digit column
test_data_top30 <- test_data_filtered[, c(1, top_30_indices + 1)]

# Fit a new SVM model using just these 30 pixels
svm_top30 <- svm(Digit ~ ., data = train_data_top30, type = 'C-classification', kernel = "linear", scal

# Make predictions and calculate errors
train_pred_top30 <- predict(svm_top30, train_data_top30)
test_pred_top30 <- predict(svm_top30, test_data_top30)

train_error_top30 <- mean(train_pred_top30 != train_data_top30$Digit)
test_error_top30 <- mean(test_pred_top30 != test_data_top30$Digit)

# Print the training and testing errors for the top 30 pixel model
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):  0.03212851
```

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
# Step 1: Prepare the data for glmnet
# Use the top 250 pixels from part a) for logistic regression with elastic net penalty
X_train <- as.matrix(train_data_filtered[, -1])  # Exclude the Digit column
y_train <- as.numeric(train_data_filtered$Digit == 5)  # Convert labels to binary (1 for '5', 0 for '4'

# Step 2: Fit logistic regression with elastic net penalty (alpha = 0.5)
elastic_net_model <- glmnet(X_train, y_train, family = "binomial", alpha = 0.5)

# Step 3: Identify the lambda that corresponds to a model with exactly 30 variables
# Check the number of non-zero coefficients for each lambda
num_nonzero <- apply(coef(elastic_net_model), 2, function(coef) sum(coef != 0))

# Find the lambda value that corresponds to exactly 30 non-zero coefficients
lambda_30 <- elastic_net_model$lambda[which(num_nonzero == 31)]  # 31 because of intercept

# Step 4: Refit the model using this lambda value and extract the selected pixels
coef_30 <- coef(elastic_net_model, s = lambda_30)  # Get the coefficients at lambda_30
selected_pixels_elastic_net <- which(coef_30[-1] != 0)  # Exclude the intercept

# Step 5: Compare to the top 30 pixels selected by the SVM
overlap_pixels <- intersect(selected_pixels_elastic_net, top_30_indices)

# Print the results
cat("Lambda value for 30-variable model: ", lambda_30, "\n")
```

```
## Lambda value for 30-variable model:  0.2759404
```

```r
cat("Selected pixels from Elastic Net: ", selected_pixels_elastic_net, "\n")
```

```
## Selected pixels from Elastic Net:  1 2 3 5 6 11 12 47 49 55 65 78 100 105 106 124 126 130 139 140 14
```

```r
cat("Overlap with SVM-selected pixels: ", overlap_pixels, "\n")
```

```
## Overlap with SVM-selected pixels:  1 6 78 100 106 139 172 187
```

Yes, there is an overlap between the pixels selected by the elastic net logistic regression and the SVM model. The pixels that overlap between the two models are:

Pixels: 1, 6, 78, 100, 106, 139, 172, 187

This means that 8 out of 30 pixels are shared between the two models.

The overlap of 8 pixels suggests both models identify key features for distinguishing digits 4 and 5, indicating these pixels are significant for classification. However, the remaining 22 pixels differ between models, reflecting that SVM and elastic net focus on different aspects of the data. Combining both selections could enhance performance.

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
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: Prepare the data for glmnet
X_train <- as.matrix(train_data_filtered[, -1])  # Exclude the Digit column
y_train <- as.numeric(train_data_filtered$Digit == 5)  # Convert labels to binary (1 for '5', 0 for '4'

# Step 2: Fit logistic regression with elastic net penalty (alpha = 0.5)
elastic_net_model <- glmnet(X_train, y_train, family = "binomial", alpha = 0.5)

# Step 3: Identify the lambda that corresponds to a model with exactly 30 variables
# Check the number of non-zero coefficients for each lambda
num_nonzero <- apply(coef(elastic_net_model), 2, function(coef) sum(coef != 0))

# Find the lambda value that corresponds to exactly 30 non-zero coefficients
lambda_30 <- elastic_net_model$lambda[which(num_nonzero == 31)]  # 31 because of intercept

# Step 4: Refit the model using this lambda value and extract the selected pixels
elastic_net_selected <- glmnet(X_train, y_train, family = "binomial", alpha = 0.5, lambda = lambda_30)


# Prepare the test data
X_test <- as.matrix(test_data_filtered[, -1])  # Test data with the top 250 pixels from part a)
y_test <- as.numeric(test_data_filtered$Digit == 5)  # Binary response for test data
```

```r
# Step 1: Predictions for SVM model with top 30 pixels
svm_pred_prob <- attr(predict(svm_top30, test_data_top30, decision.values = TRUE), "decision.values")

# Step 2: Predictions for Elastic Net model
enet_pred_prob <- predict(elastic_net_selected, X_test, type = "response")

# Step 3: Calculate AUC for SVM model
roc_svm <- roc(y_test, svm_pred_prob)
```

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

## Warning in roc.default(y_test, svm_pred_prob): 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)
cat("AUC for SVM model: ", auc_svm, "\n")
```

```
## AUC for SVM model:  0.9935756
```

```r
# Step 4: Calculate AUC for Elastic Net model
roc_enet <- roc(y_test, enet_pred_prob)
```

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

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

## Setting direction: controls < cases
```

```r
auc_enet <- auc(roc_enet)
cat("AUC for Elastic Net model: ", auc_enet, "\n")
```

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

```r
# Comparison
if (auc_svm > auc_enet) {
  cat("SVM model performs better with a higher AUC.\n")
} else {
  cat("Elastic Net model performs better with a higher AUC.\n")
}
```

```
## SVM model performs better with a higher AUC.
```

The SVM model achieved a higher AUC of 0.9936 compared to 0.9753 for the Elastic Net model, indicating that the SVM performs better in distinguishing between digits 4 and 5 on the testing data. Both models performed very well, but the SVM's slightly superior AUC suggests it is more effective at classification in this case. Converting decision values and probabilities to numeric vectors resolved any warnings.

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.

```
# Load necessary libraries
library(e1071)
library(ISLR2)

# Load the data
data("OJ")

# Set a seed for reproducibility
set.seed(7)

# Split the data into training and test sets
id = sample(nrow(OJ), 800)
train = OJ[id, ]
test = OJ[-id, ]

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

# Predict on training data
train_pred <- predict(svm_model, train)
train_error <- mean(train_pred != train$Purchase)

# Predict on test data
test_pred <- predict(svm_model, test)
test_error <- mean(test_pred != test$Purchase)

# Output 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.

```r
# Load necessary libraries
library(e1071)
library(ISLR2)

# Load the data
data("OJ")

# Set a seed for reproducibility
set.seed(7)

# Split the data into training and test sets
id = sample(nrow(OJ), 800)
train = OJ[id, ]
test = OJ[-id, ]

# Tune the SVM model to find the best cost
tune_result <- tune(svm, Purchase ~ ., data = train, kernel = "linear",
                    ranges = list(cost = c(0.01, 0.1, 1, 2, 5, 7, 10)))

# Best model
best_model <- tune_result$best.model

# Compute training and test errors with the best model
train_pred_best <- predict(best_model, train)
train_error_best <- mean(train_pred_best != train$Purchase)

test_pred_best <- predict(best_model, test)
test_error_best <- mean(test_pred_best != test$Purchase)

# Output the best cost, training error, and test error
cat("Best Cost: ", tune_result$best.parameters$cost, "\n")
```

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

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

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

```r
cat("Test Error with Best Cost: ", test_error_best, "\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.

```r
# Load necessary libraries
library(e1071)
library(ISLR2)

# Load the data
data("OJ")

# Set a seed for reproducibility
set.seed(7)

# Split the data into training and test sets
id = sample(nrow(OJ), 800)
train = OJ[id, ]
test = OJ[-id, ]

# Radial Kernel SVM with tuning for cost
tune_result_radial <- tune(svm, Purchase ~ ., data = train, kernel = "radial",
                           ranges = list(cost = c(0.01, 0.1, 1, 2, 5, 7, 10)))

best_model_radial <- tune_result_radial$best.model

# Predict and calculate errors for radial kernel
train_pred_radial <- predict(best_model_radial, train)
train_error_radial <- mean(train_pred_radial != train$Purchase)

test_pred_radial <- predict(best_model_radial, test)
test_error_radial <- mean(test_pred_radial != test$Purchase)

# Polynomial Kernel (degree 2) SVM with tuning for cost
tune_result_poly <- tune(svm, Purchase ~ ., data = train, kernel = "polynomial", degree = 2,
                         ranges = list(cost = c(0.01, 0.1, 1, 2, 5, 7, 10)))

best_model_poly <- tune_result_poly$best.model

# Predict and calculate errors for polynomial kernel
train_pred_poly <- predict(best_model_poly, train)
train_error_poly <- mean(train_pred_poly != train$Purchase)

test_pred_poly <- predict(best_model_poly, test)
test_error_poly <- mean(test_pred_poly != test$Purchase)

# Output the errors for radial and polynomial kernels
cat("Radial Kernel - Best Cost: ", tune_result_radial$best.parameters$cost, "\n")
```

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

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

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

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

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

```r
cat("Polynomial Kernel (degree 2) - Best Cost: ", tune_result_poly$best.parameters$cost, "\n")
```

```
## Polynomial Kernel (degree 2) - Best Cost:  7
```

```r
cat("Polynomial Kernel (degree 2) - Training Error: ", train_error_poly, "\n")
```

```
## Polynomial Kernel (degree 2) - Training Error:  0.15875
```

```r
cat("Polynomial Kernel (degree 2) - Test Error: ", test_error_poly, "\n")
```

```
## Polynomial Kernel (degree 2) - Test Error:  0.1444444
```

Comparison and Conclusion:

Both radial and polynomial (degree 2) kernels perform similarly in terms of training and test errors. The polynomial kernel has a marginally lower test error (14.44%) than the radial kernel (14.81%), indicating a slight advantage in predicting unseen data.

The difference between training and test errors is small across all kernels, suggesting that there is no significant overfitting. The polynomial kernel may be slightly better at capturing relationships in the data due to its lower test error, while the radial kernel offers a flexible, non-linear boundary with comparable performance.

Linear Kernel results (from part b) can serve as a baseline, but based on the radial and polynomial kernels, the data appears to benefit from non-linear decision boundaries, as both the radial and polynomial kernels outperform the linear kernel.

In summary, the polynomial kernel (degree 2) with a cost of 7 provides the best generalization, but the performance difference between radial and polynomial is minor. Both kernels perform better than the linear kernel, indicating non-linear relationships in the data.