Stat 432 Homework 7 Jerry Liang

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

Contents

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:

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

```r
# 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
#Split into training (first 1200 observations) and testing (remaining observations)
training_data <- mnist2400[1:1200, ]
testing_data <- mnist2400[1201:2400, ]
training_data <- subset(training_data, Digit %in% c(4, 5))
testing_data <- subset(testing_data, Digit %in% c(4, 5))
# Separate features and labels
train_labels <- training_data$Digit
train_features <- training_data[, -1]
test_labels <- testing_data$Digit
test_features <- testing_data[, -1]

#Perform variance screening on the training features
variances <- apply(train_features, 2, var)
top_pixels <- order(variances, decreasing = TRUE)[1:250]

# Subset the features to include only the top 250 pixels
train_features_selected <- train_features[, top_pixels]
test_features_selected <- test_features[, top_pixels]

# Fit a linear SVM using the e1071 package
library(e1071)
svm_model <- svm(x = train_features_selected, y = as.factor(train_labels), type='C-classification', ker

# Predict and evaluate the model
# Training error
train_predictions <- predict(svm_model, train_features_selected)
train_error <- mean(train_predictions != train_labels)
cat("Training Classification Error: ", train_error, "\n")
```

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

```
# Testing error
test_predictions <- predict(svm_model, test_features_selected)
test_error <- mean(test_predictions != test_labels)
cat("Testing Classification Error: ", test_error, "\n")
```

```
## Testing Classification Error:  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.

```
svm_coefficients <- as.vector(t(svm_model$coefs) %*% svm_model$SV)

top_30_pixels <- order(abs(svm_coefficients), decreasing = TRUE)[1:30]

# Display the selected top 30 pixel indices
cat("Top 30 Pixels with the Largest Absolute Coefficients:\n", top_30_pixels, "\n")
```

```
## Top 30 Pixels with the Largest Absolute Coefficients:
##  139 100 48 68 235 123 244 1 133 86 35 151 42 78 213 184 106 30 197 190 248 103 229 172 154 6 18 214
```

```
train_features_top30 <- train_features_selected[, top_30_pixels]
test_features_top30 <- test_features_selected[, top_30_pixels]

svm_model_top30 <- svm(x = train_features_top30, y = as.factor(train_labels),
                       type = 'C-classification', kernel = "linear", scale = FALSE, cost = 1)

# Predict and evaluate the model using only the top 30 pixels
# Training error
train_predictions_top30 <- predict(svm_model_top30, train_features_top30)
train_error_top30 <- mean(train_predictions_top30 != train_labels)
cat("Training Classification Error with Top 30 Pixels: ", train_error_top30, "\n")
```

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

```
# Testing error
test_predictions_top30 <- predict(svm_model_top30, test_features_top30)
test_error_top30 <- mean(test_predictions_top30 != test_labels)
cat("Testing Classification Error with Top 30 Pixels: ", test_error_top30, "\n")
```

```
## Testing Classification Error with 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
library(glmnet)
```

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

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

```r
# Prepare the training data
x_train <- as.matrix(train_features_selected)
y_train <- as.factor(train_labels)

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

non_zero_counts <- apply(elastic_net_model$beta != 0, 2, sum)
lambda_30 <- elastic_net_model$lambda[which(non_zero_counts == 30)[1]] # Find the first lambda with 30

# Extract the selected pixels (features) corresponding to this lambda
selected_pixels_en <- which(elastic_net_model$beta[, which(non_zero_counts == 30)[1]] != 0)

# Display the lambda value and selected pixels
cat("Lambda value for model with 30 selected features:", lambda_30, "\n")
```

```
## Lambda value for model with 30 selected features: 0.2759404
```

```r
cat("Selected Pixels by Elastic Net Model:\n", selected_pixels_en, "\n")
```

```
## Selected Pixels by Elastic Net Model:
##  1 2 3 5 6 11 12 47 49 55 65 78 100 105 106 124 126 130 139 140 145 171 172 177 187 188 195 202 208 1
```

```r
# Compare with the top 30 pixels selected by the SVM
overlap <- intersect(selected_pixels_en, top_30_pixels)
cat("Number of overlapping pixels with SVM model:", length(overlap), "\n")
```

```
## Number of overlapping pixels with SVM model: 8
```

```r
cat("Overlapping pixel indices:\n", overlap, "\n")
```

```
## Overlapping pixel indices:
##  1 6 78 100 106 139 172 187
```

the model with just 30 variables in the solution path is the model with 30 non-zero coefficients, lambda value is 0.276. There are 8 selected pixel overlapped. 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
```

```
# Calculate decision values for SVM model
svm_decision_values <- attributes(predict(svm_model_top30, test_features_top30, decision.values = TRUE))
svm_roc <- roc(response = as.factor(test_labels), predictor = as.numeric(svm_decision_values))
```

```
## Setting levels: control = 4, case = 5

## Setting direction: controls > cases
```

```
auc_svm <- auc(svm_roc)
cat("AUC for SVM Model (Top 30 Pixels): ", auc_svm, "\n")
```

```
## AUC for SVM Model (Top 30 Pixels):  0.9935756
```

```
# Prepare the testing data for Elastic Net
x_test <- as.matrix(test_features_selected)

# Predict probabilities using the Elastic Net model
elastic_net_probabilities <- predict(elastic_net_model, s = lambda_30, newx = x_test, type = "response")
elastic_net_roc <- roc(response = as.factor(test_labels), predictor = elastic_net_probabilities)
```

```
## Setting levels: control = 4, case = 5

## Setting direction: controls < cases
```

```
auc_elastic_net <- auc(elastic_net_roc)
cat("AUC for Elastic Net Model (30 Selected Features): ", auc_elastic_net, "\n")
```

```
## AUC for Elastic Net Model (30 Selected Features):  0.9752758
```

```
# Compare the AUC values
cat("Difference in AUC between SVM and Elastic Net: ", abs(auc_svm - auc_elastic_net), "\n")
```

```
## Difference in AUC between SVM and Elastic Net:  0.01829981
```

Seems that 0.99>0.97, and SVM model have a better performance according to 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.

```
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.

```
set.seed(432)

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

# Fit a linear SVM model using the 'svm' function from the e1071 package
svm_model <- svm(Purchase ~ ., data = train, kernel = "linear", cost = 0.01)

# Make predictions on the training data
train_pred <- predict(svm_model, train)

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

# Make predictions on the test data
test_pred <- predict(svm_model, test)

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

# Output the training and test errors
cat("Training Error:", train_error, "\n")
```

```
## Training Error: 0.17625
```

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

```
## Test Error: 0.137037
```

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
set.seed(432)
tune_result <- tune(svm, Purchase ~ ., data = train,
                    kernel = "linear",
                    ranges = list(cost = c(0.01, 0.1, 1, 2, 5, 7, 10)))

# Extract the best model
best_model <- tune_result$best.model

# Display the best cost value
cat("Best cost value:", tune_result$best.parameters$cost, "\n")
```

```
## Best cost value: 0.1
```

```r
# Compute the training error using the best model
train_pred_best <- predict(best_model, train)
train_error_best <- mean(train_pred_best != train$Purchase)

# Compute the test error using the best model
test_pred_best <- predict(best_model, test)
test_error_best <- mean(test_pred_best != test$Purchase)

# Output the training and test errors
cat("Training Error with best cost:", train_error_best, "\n")
```

```
## Training Error with best cost: 0.17375
```

```r
cat("Test Error with best cost:", test_error_best, "\n")
```

```
## Test Error with best cost: 0.1296296
```

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(432)
tune_result_radial <- tune(svm, Purchase ~ ., data = train,
                           kernel = "radial",
                           ranges = list(cost = c(0.01, 0.1, 1, 2, 5, 7, 10)))

# Extract the best model for the radial kernel
best_model_radial <- tune_result_radial$best.model

# Display the best cost value for the radial kernel
cat("Best cost value (radial):", tune_result_radial$best.parameters$cost, "\n")
```

```
## Best cost value (radial): 1
```

```r
# Compute the training error using the best radial model
train_pred_radial <- predict(best_model_radial, train)
train_error_radial <- mean(train_pred_radial != train$Purchase)
```

```r
# Compute the test error using the best radial model
test_pred_radial <- predict(best_model_radial, test)
test_error_radial <- mean(test_pred_radial != test$Purchase)

# Output the training and test errors for the radial kernel
cat("Training Error (radial):", train_error_radial, "\n")
```

```
## Training Error (radial): 0.15625
```

```r
cat("Test Error (radial):", test_error_radial, "\n")
```

```
## Test Error (radial): 0.1555556
```

```r
# Part c: Polynomial Kernel (degree 2) SVM
# Tune the SVM model using a polynomial kernel (degree 2) to find the optimal 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)))

# Extract the best model for the polynomial kernel
best_model_poly <- tune_result_poly$best.model

# Display the best cost value for the polynomial kernel
cat("Best cost value (polynomial):", tune_result_poly$best.parameters$cost, "\n")
```

```
## Best cost value (polynomial): 10
```

```r
# Compute the training error using the best polynomial model
train_pred_poly <- predict(best_model_poly, train)
train_error_poly <- mean(train_pred_poly != train$Purchase)

# Compute the test error using the best polynomial model
test_pred_poly <- predict(best_model_poly, test)
test_error_poly <- mean(test_pred_poly != test$Purchase)

# Output the training and test errors for the polynomial kernel
cat("Training Error (polynomial):", train_error_poly, "\n")
```

```
## Training Error (polynomial): 0.14625
```

```r
cat("Test Error (polynomial):", test_error_poly, "\n")
```

```
## Test Error (polynomial): 0.1777778
```

Since 0.01 is included in the list, we can find out that 0.01 is not the best cost. The best cost seems to be linear kernel SVM with cost value of 0.1 in part b since it has the lowest test error of 0.1296