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:

```
# 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)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
trainingData <- mnist2400[1:1200, ]
trainingData_45 <- subset(trainingData, Digit %in% c(4, 5))

testingData <- mnist2400[1201:2400, ]
testingData_45 <- subset(testingData, Digit %in% c(4, 5))

pixelColumns <- names(trainingData_45)[-1]
pixelVariances <- apply(trainingData_45[, pixelColumns], 2, var)

top250Pixels <- names(sort(pixelVariances, decreasing = TRUE))[1:250]

trainingData_svm <- trainingData_45[, c("Digit", top250Pixels)]
testingData_svm <- testingData_45[, c("Digit", top250Pixels)]

trainingData_svm$Digit <- as.factor(trainingData_svm$Digit)
testingData_svm$Digit <- as.factor(testingData_svm$Digit)

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

pred_train <- predict(model, trainingData_svm)
training_error <- mean(pred_train != trainingData_svm$Digit)

print(paste("train error", training_error))
```

```
## [1] "train error 0"
```

```r
pred_test <- predict(model, testingData_svm)
testing_error <- mean(pred_test != testingData_svm$Digit)

print(paste("test error", testing_error))
```

```
## [1] "test error 0.0321285140562249"
```

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
w <- t(model$coefs) %*% model$SV
w_vector <- as.vector(w)
names(w_vector) <- colnames(model$SV)
abs_w <- abs(w_vector)
top30_pixels <- names(sort(abs_w, decreasing = TRUE))[1:30]

trainingData_top30 <- trainingData_45[, c("Digit", top30_pixels)]
testingData_top30 <- testingData_45[, c("Digit", top30_pixels)]

trainingData_top30$Digit <- as.factor(trainingData_top30$Digit)
testingData_top30$Digit <- as.factor(testingData_top30$Digit)

model_top30 <- svm(Digit ~ ., data = trainingData_top30,
                   kernel = "linear", cost = 1, probability = TRUE)

pred_train_top30 <- predict(model_top30, trainingData_top30,
                            probability = TRUE)
training_error_top30 <- mean(pred_train_top30 != trainingData_top30$Digit)
cat("Training classification error with top 30 pixels:", training_error_top30, "\n")
```

```
## Training classification error with top 30 pixels: 0
```

```r
pred_test_top30 <- predict(model_top30, testingData_top30,
                           probability = TRUE)
testing_error_top30 <- mean(pred_test_top30 != testingData_top30$Digit)
cat("Testing classification error with top 30 pixels:", testing_error_top30, "\n")
```

```
## Testing classification error with top 30 pixels: 0.04417671
```

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
y_train <- trainingData_svm$Digit
x_train <- as.matrix(trainingData_svm[, -1])

fit_glmnet <- glmnet(x_train, y_train, family = "binomial", alpha = 0.5)

num_nonzero <- fit_glmnet$df
index_30 <- which(num_nonzero == 30)
lambda_30 <- fit_glmnet$lambda[index_30]
print(paste("lambda", lambda_30))
```

```
## [1] "lambda 0.275940371272523"
```

```r
coef_30 <- coef(fit_glmnet, s = lambda_30)

nonzero_indices <- which(coef_30 != 0)
```

```
selected_pixels <- rownames(coef_30)[nonzero_indices]
selected_pixels <- selected_pixels[selected_pixels != "(Intercept)"]

print("selected c")
```

```
## [1] "selected c"
```

```
print(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"
```

```
print("top 30")
```

```
## [1] "top 30"
```

```
print(top30_pixels)
```

```
##  [1] "Pixel490" "Pixel293" "Pixel541" "Pixel437" "Pixel382" "Pixel517"
##  [7] "Pixel381" "Pixel486" "Pixel657" "Pixel294" "Pixel413" "Pixel458"
## [13] "Pixel512" "Pixel494" "Pixel511" "Pixel187" "Pixel491" "Pixel185"
## [19] "Pixel248" "Pixel629" "Pixel207" "Pixel329" "Pixel353" "Pixel268"
## [25] "Pixel464" "Pixel181" "Pixel487" "Pixel435" "Pixel412" "Pixel493"
```

```
print(paste("overlap", length(intersect(selected_pixels, top30_pixels))))
```

```
## [1] "overlap 7"
```

```
intersect(selected_pixels, top30_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
```

```
probabilities_svm <- attr(pred_test_top30, "probabilities")
prob_svm <- probabilities_svm[, "5"]

roc_svm <- roc(response = testingData_top30$Digit,
               predictor = prob_svm, levels = c("4", "5"),
               direction = "<")
auc_svm <- auc(roc_svm)
cat("AUC for b model on testing data:", auc_svm, "\n")
```

```
## AUC for b model on testing data: 0.991499
```

```
x_test_net <- as.matrix(testingData_svm[, -1])
y_test_net <- testingData_svm$Digit
prob_net <- predict(fit_glmnet, newx = x_test_net,
                    s = lambda_30, type = "response")
roc_net <- roc(response = y_test_net, predictor = as.vector(prob_net))
```

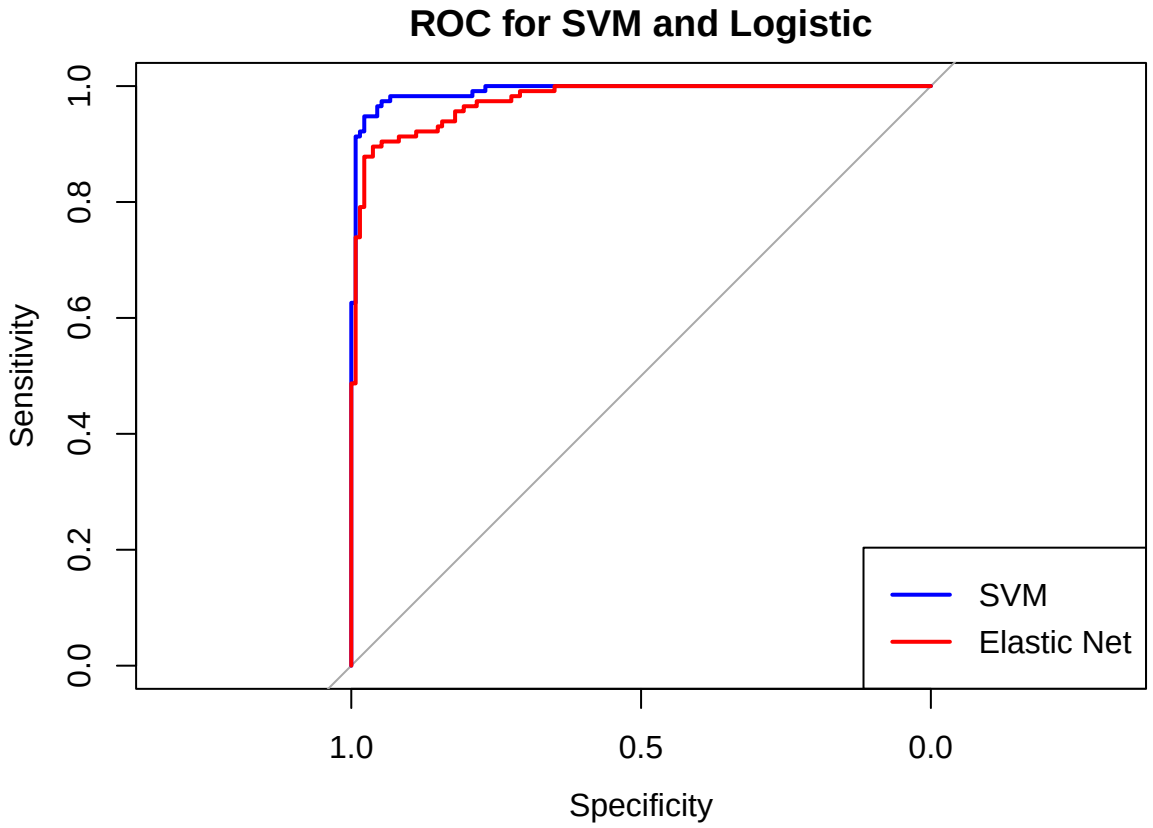
```
## Setting levels: control = 4, case = 5
```

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

```
auc_net <- auc(roc_net)
cat("AUC for Elastic Net model on testing data:", auc_net, "\n")
```

```
## AUC for Elastic Net model on testing data: 0.9752758
```

```
plot(roc_svm, col = "blue", main = "ROC for SVM and Logistic")
lines(roc_net, col = "red")
legend("bottomright", legend = c("SVM", "Elastic Net"),
       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.

```
library(e1071)
set.seed(7)

svm_linear = svm(Purchase ~ ., data = train, kernel = "linear", cost = 0.01)

train_pred = predict(svm_linear, train)
train_table = table(Predicted = train_pred, Actual = train$Purchase)
train_error = 1 - sum(diag(train_table)) / sum(train_table)

test_pred = predict(svm_linear, test)
test_table = table(Predicted = test_pred, Actual = test$Purchase)
test_error = 1 - sum(diag(test_table)) / sum(test_table)

print(paste("train_error", train_error))
```

```
## [1] "train_error 0.17"
```

```
print(paste("test_error", test_error))
```

```
## [1] "test_error 0.162962962962963"
```

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(7)
tune_linear = tune(svm, Purchase ~ ., data = train, kernel = "linear",
                   ranges = list(cost = c(0.01, 0.1, 1, 2, 5, 7, 10)))

best_linear_model = tune_linear$best.model
best_cost = tune_linear$best.parameters$cost

train_pred_best = predict(best_linear_model, train)
train_table_best = table(Predicted = train_pred_best, Actual = train$Purchase)
train_error_best = 1 - sum(diag(train_table_best)) / sum(train_table_best)

test_pred_best = predict(best_linear_model, test)
test_table_best = table(Predicted = test_pred_best, Actual = test$Purchase)
test_error_best = 1 - sum(diag(test_table_best)) / sum(test_table_best)

print(paste("best cost", best_cost))
```

```
## [1] "best cost 5"
```

```r
print(paste("train error", train_error_best))
```

```
## [1] "train error 0.1675"
```

```r
print(paste("test error", test_error_best))
```

```
## [1] "test error 0.162962962962963"
```

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(7)

svm_radial = svm(Purchase ~ ., data = train, kernel = "radial", cost = 0.01)

train_pred_radial = predict(svm_radial, train)
train_table_radial = table(Predicted = train_pred_radial,
                           Actual = train$Purchase)
train_error_radial = (1 - sum(diag(train_table_radial)) /
                        sum(train_table_radial))

test_pred_radial = predict(svm_radial, test)
test_table_radial = table(Predicted = test_pred_radial,
                          Actual = test$Purchase)
test_error_radial = 1 - sum(diag(test_table_radial)) / sum(test_table_radial)

print(paste("radial train error", train_error_radial))
```

```
## [1] "radial train error 0.395"
```

```r
print(paste("radial test error", test_error_radial))
```

```
## [1] "radial test error 0.374074074074074"
```

```r
tune_radial = tune(svm, Purchase ~ ., data = train, kernel = "radial",
                   ranges = list(cost = c(0.01, 0.1, 1, 2, 5, 7, 10)))

best_radial_model = tune_radial$best.model
best_cost_radial = tune_radial$best.parameters$cost

train_pred_radial_best = predict(best_radial_model, train)
train_table_radial_best = table(Predicted = train_pred_radial_best,
                                Actual = train$Purchase)
train_error_radial_best = (1 - sum(diag(train_table_radial_best)) /
                             sum(train_table_radial_best))

test_pred_radial_best = predict(best_radial_model, test)
test_table_radial_best = table(Predicted = test_pred_radial_best,
                               Actual = test$Purchase)
test_error_radial_best = (1 - sum(diag(test_table_radial_best)) /
                            sum(test_table_radial_best))

print(paste("radial best cost", best_cost_radial))
```

```
## [1] "radial best cost 1"
```

```
print(paste("radial best train error", train_error_radial_best))
```

```
## [1] "radial best train error 0.1575"
```

```
print(paste("radial best test error", test_error_radial_best))
```

```
## [1] "radial best test error 0.148148148148148"
```

```
svm_poly = svm(Purchase ~ ., data = train, kernel = "polynomial",
               degree = 2, cost = 0.01)

train_pred_poly = predict(svm_poly, train)
train_table_poly = table(Predicted = train_pred_poly, Actual = train$Purchase)
train_error_poly = 1 - sum(diag(train_table_poly)) / sum(train_table_poly)

test_pred_poly = predict(svm_poly, test)
test_table_poly = table(Predicted = test_pred_poly, Actual = test$Purchase)
test_error_poly = 1 - sum(diag(test_table_poly)) / sum(test_table_poly)

print(paste("poly train error", train_error_poly))
```

```
## [1] "poly train error 0.39375"
```

```
print(paste("poly test error", test_error_poly))
```

```
## [1] "poly test error 0.374074074074074"
```

```
tune_poly = tune(svm, Purchase ~ ., data = train,
                 kernel = "polynomial", degree = 2,
                 ranges = list(cost = c(0.01, 0.1, 1, 2, 5, 7, 10)))

best_poly_model = tune_poly$best.model
best_cost_poly = tune_poly$best.parameters$cost

train_pred_poly_best = predict(best_poly_model, train)
train_table_poly_best = table(Predicted = train_pred_poly_best,
                              Actual = train$Purchase)
train_error_poly_best = (1 - sum(diag(train_table_poly_best)) /
                           sum(train_table_poly_best))

test_pred_poly_best = predict(best_poly_model, test)
test_table_poly_best = table(Predicted = test_pred_poly_best,
                              Actual = test$Purchase)
test_error_poly_best = (1 - sum(diag(test_table_poly_best)) /
                          sum(test_table_poly_best))

print(paste("poly best cost", best_cost_poly))
```

```
## [1] "poly best cost 5"
```

```
print(paste("poly best train error", train_error_poly_best))
```

```
## [1] "poly best train error 0.16375"
```

```
print(paste("poly best test error", test_error_poly_best))
```

```
## [1] "poly best test error 0.159259259259259"
```

From the output in part b and c, we can find that the training and testing error after tune for linear kernel is arount 16.75% and 16.3% respectively, for radial kernel is around 15.75% and 14.81% respectively, for polynomial kernel is around 16.38% and 15.93% respectively. Overall, the ability of fitting data improved significantly after tune for radial and polynomial kernel, but not for linear kernel. The error of radial kernal after tune is the lowest among three, followed by polynomial kernel after tune, and linear kernel after tune has the highest among three. These two kernels captures non linear relationship better than the linear kernel.