Stat 432 Homework 7

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

- Instruction
- Question 1: SVM on Hand Written Digit Data (55 points)
- Question 2: SVM with Kernel Trick (45 points)

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 (https://studentcode.illinois.edu/article1/part4/1-401/). Any violation will be punished as severely as possible. Final submissions must be uploaded to Gradescope (https://www.gradescope.com/courses/570816). No email or hard copy will be accepted. For **late submission policy and grading rubrics** (https://teazrq.github.io/stat432/syllabus.html), 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.

Solution:

```r
dim(mnist2400)
```

```
## [1] 2400  785
```

```r
train <- mnist2400[1:1200,]
train <- train[train$Digit == 4 | train$Digit == 5,]
test <- mnist2400[1200:2400,]
test <- test[test$Digit == 4 | test$Digit == 5,]

library(e1071)

# perform marginal screening
var = apply(train[, -1], 2, var)
varuse = order(var, decreasing = TRUE)[1:250]
train = train[, c(1, varuse + 1)]
test = test[, c(1, varuse + 1)]

# redo svm
svmfit = svm(as.factor(Digit) ~ ., data = train, kernel = "linear", cost = 1)

# training error
trainError <- 1 - mean( predict(svmfit, train) == train$Digit )
trainError
```

```
## [1] 0
```

```
# testing error
testError <- 1 - mean( predict(svmfit, test) == test$Digit )
testError
```

```
## [1] 0.03212851
```

The training error is 0 and the testing error is 0.0321285.

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.

Solution:

The β parameters are defined as

$$\widehat{\beta} = \sum_{i=1}^{n} \alpha_i y_i x_i$$

Since in the `e1071` package provides the $\alpha_i y_i$ as the coefficients, we can extract the coefficients directly from the model by multiplying them to the x_i from the support vectors.

```
# Extract coefficients of the linear SVM model
coefficients <- svmfit$coefs
w <- t(svmfit$SV) %*% coefficients
b <- -svmfit$rho

# Find the top 20 pixels with the largest absolute coefficients
top_pixels <- w[order(abs(w), decreasing = TRUE)[1:30],]

top_names <- names(top_pixels)

# Refit the SVM using just these 30 pixels
train_sub = train[, c("Digit", top_names)]
test_sub = test[, c("Digit", top_names)]
svmfit_sub = svm(as.factor(Digit) ~ ., data = train_sub, kernel = "linear", cost = 1)

# training error
1 - mean( predict(svmfit_sub, train_sub) == train_sub$Digit )
```

```
## [1] 0
```

```
# testing error
1 - mean( predict(svmfit_sub, test_sub) == test_sub$Digit )
```

```
## [1] 0.04016064
```

Hence, after refitting the model with the top 30 pixels, the training error is 0 and the testing error is 0.0401606.

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.

Solution:

```
library(glmnet)
```

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

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

```
x = data.matrix(train[, -1])
y = as.numeric(train$Digit) - 4
enet.fit = glmnet(x, y, family = "binomial", alpha = 0.5)

# select the model with just 30 variables
enet.coef = coef(enet.fit)[-1, ]
enet.use = enet.coef[, colSums(coef(enet.fit)[-1, ] != 0) == 30]
enet.use = names(enet.use)[which(enet.use != 0)]

# extract the lambda value
lambda.use = which(colSums(coef(enet.fit)[-1, ] != 0) == 30)
lambda.value = enet.fit$lambda[lambda.use]
lambda.value
```

```
## [1] 0.2759404
```

```
# check overlap
sum(top_names %in% enet.use)
```

```
## [1] 7
```

Our findings suggest a overlap of 7 digits between the pixels selected by the SVM model in part b) and the glmnet model in part c). This is not that much overlap in the variables for the top 30.

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.

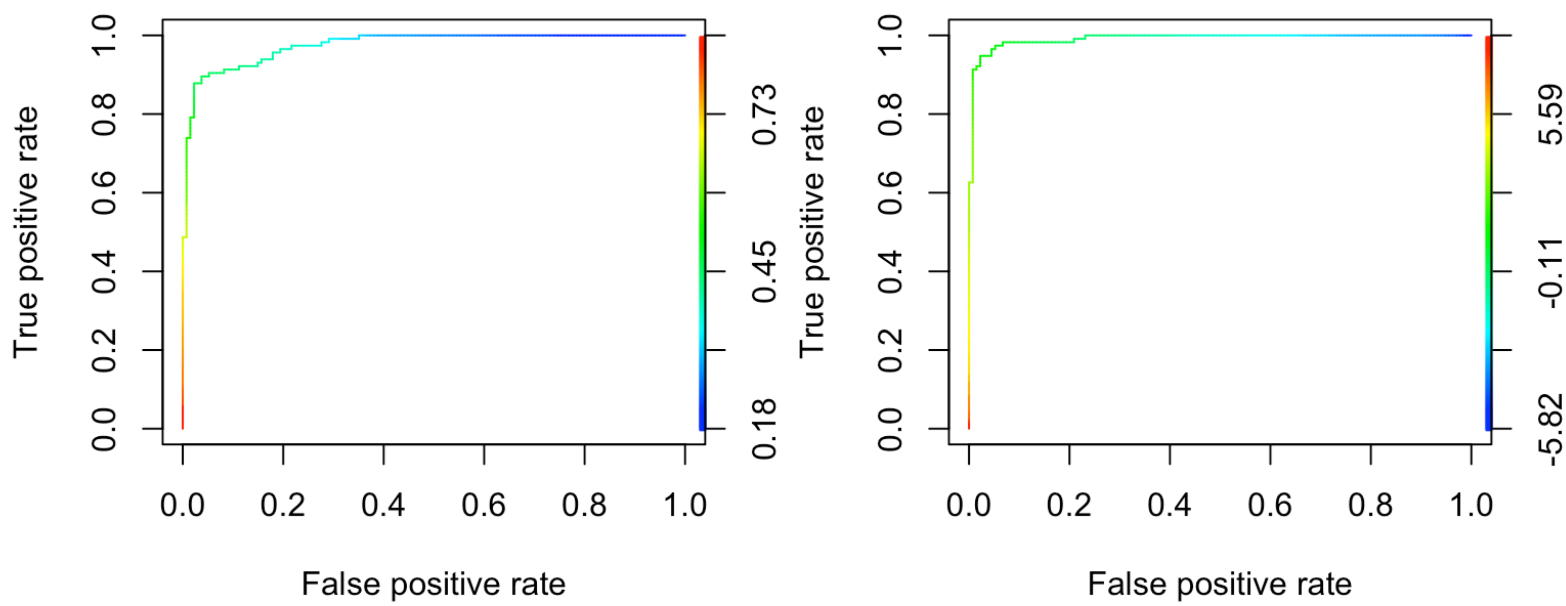
Solution:

```r
# calculate the prediction on the test set
x_test = data.matrix(test[, -1])
enet.pred = predict(enet.fit, newx = x_test, type = "response")[, lambda.use]

svm.pred = 1 - attr(predict(svmfit_sub, test_sub, decision.values = TRUE), "decision.values")

# plot ROC
library(ROCR)
par(mfrow=c(1,2))
enet.roc <- prediction(enet.pred, test$Digit - 4)
plot(performance(enet.roc,"tpr","fpr"), colorize=TRUE)

svm.roc <- prediction(svm.pred, test_sub$Digit)
plot(performance(svm.roc,"tpr","fpr"), colorize=TRUE)
```




```r
# calculate AUC
performance(enet.roc, measure = "auc")@y.values[[1]]
```

```
## [1] 0.9752758
```

```r
performance(svm.roc, measure = "auc")@y.values[[1]]
```

```
## [1] 0.991499
```

The two models perform almost the same, with SVM slightly better.

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.

Solution:

```
require(ISLR)
```

```
## Loading required package: ISLR
```

```
##
## Attaching package: 'ISLR'
```

```
## The following objects are masked from 'package:ISLR2':
##
##     Auto, Credit
```

```
require(e1071)
library(knitr)

svm.fit=svm(Purchase~.,data=train,cost=0.01,kernel='linear')

# train
svm.pred=predict(svm.fit,train)
kable(table(train[,'Purchase'],svm.pred))
```

	CH	MM
CH	424	60
MM	76	240

```
mean(train$Purchase != svm.pred)
```

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

```
# test
svm.pred=predict(svm.fit,test)
kable(table(test$Purchase,svm.pred))
```

	CH	MM
CH	154	15
MM	29	72

```
mean(test$Purchase != svm.pred)
```

```
## [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.

Solution:

```
set=c(0.01, 0.1, 1, 2, 5, 7, 10)
svm.tune = tune(svm,Purchase~.,data=train,ranges=data.frame(cost=set),kernel='linear')
summary(svm.tune)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 0.17
##
## - Detailed performance results:
##    cost   error dispersion
## 1  0.01 0.17625 0.04581439
## 2  0.10 0.17375 0.04348132
## 3  1.00 0.17000 0.04174992
## 4  2.00 0.17000 0.04257347
## 5  5.00 0.17000 0.04174992
## 6  7.00 0.17375 0.03928617
## 7 10.00 0.17375 0.04226652
```

```
svm.pred=predict(svm.tune$best.model,train)
kable(table(train$Purchase,svm.pred))
```

	CH	MM
CH	423	61
MM	73	243

```
mean(train$Purchase != svm.pred)
```

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

```
svm.pred=predict(svm.tune$best.model,test)
kable(table(test$Purchase,svm.pred))
```

	CH	MM
CH	153	16
MM	25	76

```
mean(test$Purchase != svm.pred)
```

```
## [1] 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.

Solution:

SVM with Radial Kernel:

```
svm.fit = svm(Purchase~.,data=train,cost=0.01,kernel='radial')
summary(svm.fit)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = train, cost = 0.01, kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.01
##
## Number of Support Vectors:  636
##
##  ( 320 316 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
# train
svm.pred=predict(svm.fit,train)
kable(table(train[,'Purchase'],svm.pred))
```

	CH	MM
CH	484	0
MM	316	0

```
mean(train$Purchase != svm.pred)
```

```
## [1] 0.395
```

```
# test
svm.pred=predict(svm.fit,test)
kable(table(test$Purchase,svm.pred))
```

	CH	MM
CH	169	0
MM	101	0

```
mean(test$Purchase != svm.pred)
```

```
## [1] 0.3740741
```

```
set=c(0.01, 0.1, 1, 2, 5, 7, 10)
svm.tune=tune(svm,Purchase~.,data=train,ranges=data.frame(cost=set),kernel='radial')
summary(svm.tune)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 0.18
##
## - Detailed performance results:
##    cost   error dispersion
## 1  0.01 0.39500 0.03184162
## 2  0.10 0.19500 0.03872983
## 3  1.00 0.18000 0.04495368
## 4  2.00 0.18625 0.04656611
## 5  5.00 0.18750 0.04370037
## 6  7.00 0.18750 0.04487637
## 7 10.00 0.18375 0.04168749
```

```
svm.pred=predict(svm.tune$best.model,train)
kable(table(train$Purchase,svm.pred))
```

	CH	MM
CH	433	51
MM	75	241

```
mean(train$Purchase != svm.pred)
```

```
## [1] 0.1575
```

```
svm.pred=predict(svm.tune$best.model,test)
kable(table(test$Purchase,svm.pred))
```

	CH	MM
CH	158	11
MM	29	72

```
mean(test$Purchase != svm.pred)
```

```
## [1] 0.1481481
```

SVM with Polynomial Kernel (degree 2):

```
svm.fit=svm(Purchase~.,data=train,cost=0.01,kernel='polynomial', degree = 2)
summary(svm.fit)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = train, cost = 0.01, kernel = "polynomial",
##     degree = 2)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  0.01
##      degree:  2
##      coef.0:  0
##
## Number of Support Vectors:  637
##
##  ( 321 316 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
# train
svm.pred=predict(svm.fit,train)
kable(table(train[,'Purchase'],svm.pred))
```

	CH	MM
CH	484	0
MM	315	1

```
mean(train$Purchase != svm.pred)
```

```
## [1] 0.39375
```

```
# test
svm.pred=predict(svm.fit,test)
kable(table(test$Purchase,svm.pred))
```

	CH	MM
CH	169	0
MM	101	0

```
mean(test$Purchase != svm.pred)
```

```
## [1] 0.3740741
```

```
set=c(0.01, 0.1, 1, 2, 5, 7, 10)
 svm.tune=tune(svm,Purchase~.,data=train,ranges=data.frame(cost=set),kernel='polynomial', degree
=2)
summary(svm.tune)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     5
##
## - best performance: 0.195
##
## - Detailed performance results:
##    cost   error dispersion
## 1  0.01 0.39500 0.05277047
## 2  0.10 0.32375 0.06440163
## 3  1.00 0.20500 0.04174992
## 4  2.00 0.20125 0.02913689
## 5  5.00 0.19500 0.04377975
## 6  7.00 0.19625 0.04489571
## 7 10.00 0.19875 0.04348132
```

```r
svm.pred=predict(svm.tune$best.model,train)
kable(table(train$Purchase,svm.pred))
```

	CH	MM
CH	441	43
MM	88	228

```r
mean(train$Purchase != svm.pred)
```

```
## [1] 0.16375
```

```r
svm.pred=predict(svm.tune$best.model,test)
kable(table(test$Purchase,svm.pred))
```

	CH	MM
CH	161	8
MM	35	66

```r
mean(test$Purchase != svm.pred)
```

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

```r
svm_results <- data.frame(
  Kernel = c("Linear", "Radial", "Polynomial (Degree 2)"),
  Training_Error_tuned = c(0.1675, 0.1575, 0.1630),
  Test_Error_tuned = c(0.152, 0.148, 0.159),
  Optimal_cost=c(5,1,5)
)
svm_results
```

Kernel <chr>	Training_Error_tuned <dbl>	Test_Error_tuned <dbl>	Optimal_cost <dbl>
Linear	0.1675	0.152	5
Radial	0.1575	0.148	1
Polynomial (Degree 2)	0.1630	0.159	5

3 rows

It seems like svm with radial kernel, default gamma value, and cost parameter of 1 give the best train and test error.