Computer Assignment 11 - Tuning Regression Parameters and SVM

Rui Li

Greetings everyone! With the final project looming overhead, I have attempted to make this a lighter computer assignment. Since the amount that I need to cover does not really align with my intent here, I am going to provide some examples in this assignment and then simply require you to use the built in R functions on some data. Please don’t be intimidated by the number of questions, this assignment is mostly about reading examples and using pre-written functions from R. -Murph

Note: You’ll need to install the e1071, HDCI, and stringr packages before this document will compile.

# LASSO and Ridge Regression Tuning Parameter

Recall that both LASSO and Ridge have a penalty parameter that controls the amount of regularization in our procedure. As you might imagine, a good choice of is crucial to finding a good procedure fit. With our goal being to maximize our prediction accuracy, we can actually use what we have learned about cross-validation to choose a good . Recall that we use cross-validation to create an estimate for the actual test error of a procedure (the test error we would get for a general procedure, not of a particular classification rule we have created on the training data).

Each distinct value of the penalty parameter determines a distinct model we could use to fit our data and our aim is to pick the best procedure. The following algorithm uses cross-validation to choose by calculating estimates of prediction accuracy for a set of candidate s. The chosen is the one that has the best estimated accuracy.

#### K-fold cross validation for choosing

1. Divide the set in to subsets (i.e. folds) of roughly equal size,
2. For :

* Consider training on , and validating on
* For each value of the tuning parameter , compute the classification rule using the training set. Here is the LASSO/Ridge Regression model with tuning parameter trained on .
* For each , record the total testing error on the validation set:

1. For each tuning parameter , compute the average error over all folds,
2. Choose such that .

Further discussion of this process can be found [here](https://www.stat.cmu.edu/~ryantibs/datamining/lectures/18-val1.pdf). Luckily for us, R has built-in functionality to perform this entire process for us! We will practice using these built-in functions on the following data.

## A. Regression on Tree Leaf Images

We will attempt to identify trees based on image data of their leaves. This is a tough problem, though apps such as iNaturalist now do a pretty good job identifying plants from images taken on your phone.

The data set is from [here](https://www.kaggle.com/c/leaf-classification/data).

Images have been pre-processed, so the dataset inlcudes vectors for margin, shape and texture attributes for each of almost 1000 images.

We will start by loading the leaves dataset and dropping the id and species variables.

leaf = read.csv("leaves.csv")  
leaf$id = NULL  
leaf$species = NULL

1. Build a LASSO model using the Lasso function from the last homework. Make your response the margin1 variable and the rest of the variables your predictors. Set the parameter fix.lambda to FALSE.

library(MASS);library(HDCI);

## Warning: package 'HDCI' was built under R version 3.6.3

Y = leaf[,1];X = leaf[,-1]  
leaf.lasso = Lasso(X,Y,fix.lambda=F)

1. When fix.lambda is set to FALSE, the Lasso function tunes a using cross-validation. According to the manual page, what is the default number of folds the Lasso function uses?

The default number of folds is 10.

1. Print out the value that your LASSO model selects. Report how many non-zero parameter values there are in your LASSO model.

print(leaf.lasso$lambda)

## [1] 2.533211e-06

sum(leaf.lasso$beta!=0)

## [1] 127

There are 127 non-zero parameter values.

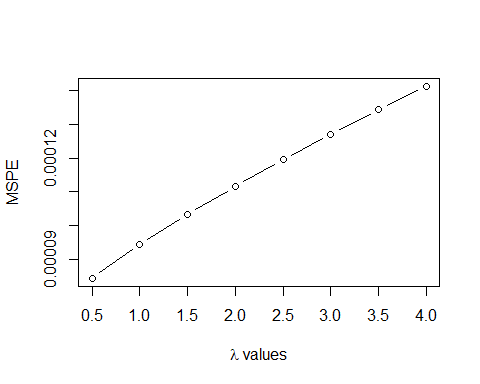
1. With the same predictors and response, let us do the same thing with a Ridge Regression model. Cross validation with Ridge Regression in R is done with the ridgereg.cv function in the MXM package. ridgereg.cv cross validates on every value of you provide and plots the Mean Square Prediction Error (MSPE) for each . Use ridgereg.cv for .

library(MXM)

## Warning: package 'MXM' was built under R version 3.6.3

## Registered S3 method overwritten by 'sets':  
## method from   
## print.element ggplot2

ridge.reg = ridgereg.cv(Y,data = X, lambda = seq(0.5,4, by=0.5))

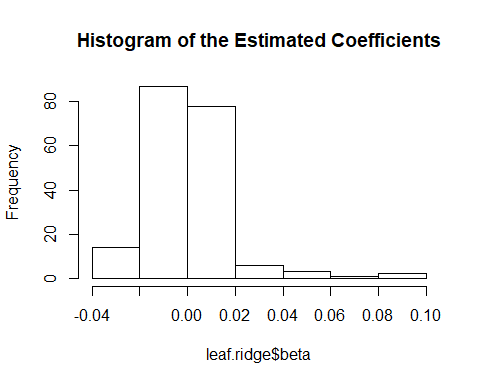


1. According to the output for ridgereg.cv, which value of should we choose?

From the plot above, we should choose lambda=0.5, which has the lowest `Mean Square Prediction Error`.

1. Build a Ridge Regression model using this . Print a histogram of the estimated coefficients for the fitted model

leaf.ridge=ridge.reg(Y, X, lambda = 0.5)  
hist(leaf.ridge$beta, main = "Histogram of the Estimated Coefficients")

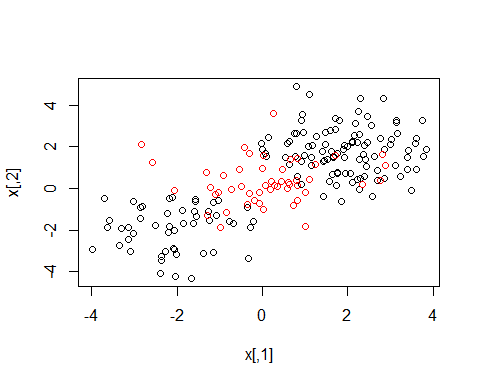


# B. SVM

For this section we will use the svm function in package e1071. Let us walk through a simple example (originally found [here](https://rstudio-pubs-static.s3.amazonaws.com/271792_96b51b7fa2af4b3f808d04f3f3051516.html)) to see how the svm function works:

set.seed(13)  
x=matrix(rnorm(200\*2) , ncol =2)  
x[1:100 ,] = x[1:100 ,] + 2  
x[101:150 ,]= x[101:150,] - 2  
y=c(rep(1 ,150) ,rep(2 ,50) )  
dat=data.frame(x=x,y=as.factor(y))

plot(x, col =y)

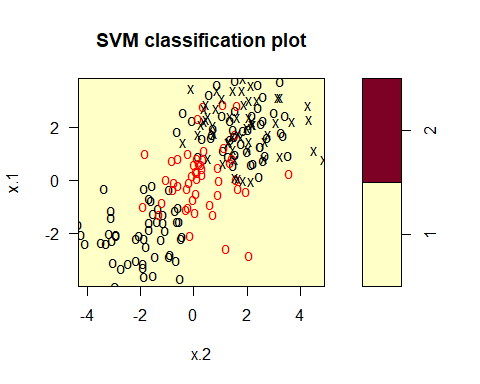


Note here that our data are NOT linearly seperable! In fact, it does not appear that a linear seperation rule will work here. Let us verify this using a linear kernel with our SVM:

library(e1071)

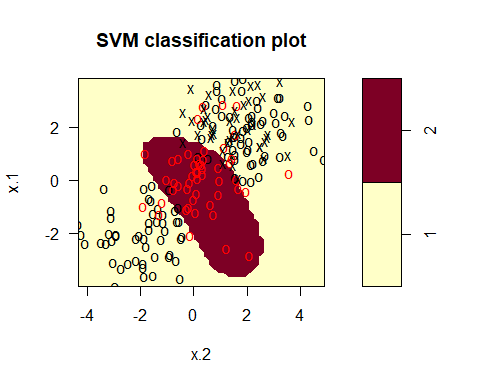
## Warning: package 'e1071' was built under R version 3.6.3

train=sample(200 ,100)  
svmfit = svm(y~., data=dat[train ,], kernel ="linear", gamma = 1, cost = 1)  
plot(svmfit , dat)



It would appear that all observations are classified as 1s, confirming our suspicions. Luckily, SVMs are not limited to linear kernels:

library(e1071)  
train=sample(200 ,100)  
svmfit = svm(y~., data=dat[train ,], kernel ="radial", gamma = 1, cost = 1)  
plot(svmfit , dat)



The above is a special form of SVM where we used a radial kernel. While the use of non-linear kernels is an interesting topic to explore, we merely introduce it here. For the following example and exercise, we will use a linear kernel.

Recall from class that SVM requires a tuning parameter (which, if you check the manual page, the svm function calls cost). Like the ridgereg.cv function, the e1071 library has a built-in cross-validation function for choosing a good value of . Observe the following:

set.seed(13)  
tune.out=tune(svm ,y~.,data=dat ,kernel ="linear",   
 ranges =list(cost=c(0.001,0.01,0.1, 1,5,10,100)))  
summary(tune.out)

##   
## Parameter tuning of 'svm':  
##   
## - sampling method: 10-fold cross validation   
##   
## - best parameters:  
## cost  
## 0.001  
##   
## - best performance: 0.25   
##   
## - Detailed performance results:  
## cost error dispersion  
## 1 1e-03 0.25 0.0745356  
## 2 1e-02 0.25 0.0745356  
## 3 1e-01 0.25 0.0745356  
## 4 1e+00 0.25 0.0745356  
## 5 5e+00 0.25 0.0745356  
## 6 1e+01 0.25 0.0745356  
## 7 1e+02 0.25 0.0745356

According to the output, our best choice of the cost parameter would be 0.001. The tune() function stores the best model obtained, which can be accessed as follows:

bestmod =tune.out$best.model

# C. Applying SVM to Tree Leaf Images

We begin by loading the leaves dataset again and this time only dropping the id variable. We will further extract the genus of each observation using the stringr package in R.

library(stringr)

## Warning: package 'stringr' was built under R version 3.6.3

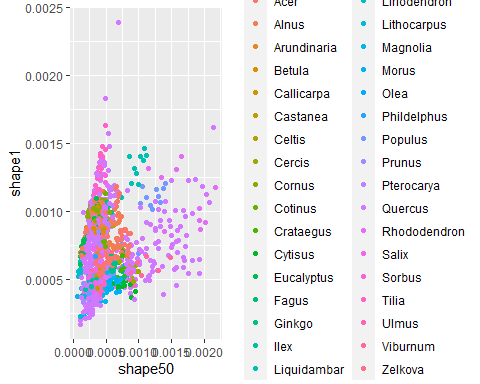
leaf = read.csv("leaves.csv", stringsAsFactors = FALSE)  
leaf$id = NULL  
leaf$genus <- str\_split(leaf$species, "\_", simplify = TRUE)[, 1]  
leaf$genus = as.factor(leaf$genus)  
leaf$species = NULL

1. Make a scatter plot of shape1 by shape50, coloring by genus label. Does this data look linearly seperable?

library(ggplot2)

## Warning: package 'ggplot2' was built under R version 3.6.3

ggplot(leaf, aes(x = shape50, y = shape1, color = genus)) + geom\_point()



This data does not look linearly seperable.

1. Randomly split your data into test and training sets. About 35 percent of the data should be in the test set. Display a summary of genus labels in the training set. **Note: In the rare event that one class in the training data is not represented, you may reduce the test set percentage to 30 percent and resample.**

training\_size <- round(.65 \* nrow(leaf))   
indices = sample(1:nrow(leaf), training\_size)   
training\_set <- leaf[indices,]   
testing\_set <- leaf[-(indices),]  
summary(training\_set$genus)

## Acer Alnus Arundinaria Betula Callicarpa Castanea   
## 58 26 7 14 8 6   
## Celtis Cercis Cornus Cotinus Crataegus Cytisus   
## 7 7 20 5 7 6   
## Eucalyptus Fagus Ginkgo Ilex Liquidambar Liriodendron   
## 20 6 7 13 7 7   
## Lithocarpus Magnolia Morus Olea Phildelphus Populus   
## 12 13 8 7 3 24   
## Prunus Pterocarya Quercus Rhododendron Salix Sorbus   
## 8 8 249 6 16 6   
## Tilia Ulmus Viburnum Zelkova   
## 22 9 14 8

1. Tune a SVM model with a linear kernel on the training set. Use (this may take a while for your computer to run). Report the cost value of the best model. Use the predict function to classify the data from the test set and the training set. Report both the testing and training errors

tune.out=tune(svm ,genus~.,data=training\_set ,kernel ="linear", ranges =list(cost=c(0.001,0.01,0.1,1,5,10,100)))  
summary(tune.out)

##   
## Parameter tuning of 'svm':  
##   
## - sampling method: 10-fold cross validation   
##   
## - best parameters:  
## cost  
## 0.1  
##   
## - best performance: 0.02175481   
##   
## - Detailed performance results:  
## cost error dispersion  
## 1 1e-03 0.54653846 0.04186424  
## 2 1e-02 0.06980769 0.02935388  
## 3 1e-01 0.02175481 0.01498601  
## 4 1e+00 0.02175481 0.01498601  
## 5 5e+00 0.02175481 0.01498601  
## 6 1e+01 0.02175481 0.01498601  
## 7 1e+02 0.02175481 0.01498601

bestmod =tune.out$best.model  
best\_cost = bestmod$cost  
print(paste0("The cost of the best model is ",best\_cost,"."))

## [1] "The cost of the best model is 0.1."

train\_pre = predict(bestmod, training\_set[,1:192])  
test\_pre = predict(bestmod, testing\_set[,1:192])

train\_error = sum(train\_pre!=training\_set$genus)/length(training\_set$genus)  
test\_error = sum(test\_pre!=testing\_set$genus)/length(testing\_set$genus)  
  
print(paste0("The training error is ",train\_error,"."))

## [1] "The training error is 0."

print(paste0("The testing error is ",test\_error,"."))

## [1] "The testing error is 0.0260115606936416."