Computer Assignment 11 - Tuning Regression Parameters and SVM

Machine Learning, Spring 2020

YOUR NAME

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 and stringr packages before this document will compile.

LASSO and Ridge Regression Tuning Parameter

Recall that both LASSO and Ridge have a penalty parameter λ that controls the amount of regularization in our procedure. As you might imagine, a good choice of λ 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 λ. 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 λ 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 λ by calculating estimates of prediction accuracy for a set of candidate λs. The λ chosen is the one that has the best estimated accuracy.

K-fold cross validation for choosing λ

1. Divide the set $\{1, \ldots, n\}$ in to K subsets (i.e. folds) of roughly equal size, $F_1, \ldots, F_K$
2. For $k = 1, \ldots, K$:

- Consider training on $(x_i, y_i), i \notin F_k$, and validating on $(x_i, y_i), i \in F_k$
- For each value of the tuning parameter $\lambda \in \{\lambda_1, \ldots, \lambda_m\}$, compute the classification rule $\hat{f}_\lambda^{-k}$ using the training set. Here $\hat{f}_\lambda^{-k}$ is the LASSO/Ridge Regression model with tuning parameter λ trained on $(x_i, y_i), i \notin F_k$.
- For each λ, record the total testing error on the validation set:

$$e_k(\lambda) = \frac{1}{|F_k|} \sum_{i \in F_k} I(y_i \neq \hat{f}_\lambda^{-k}(x_i)).$$

3. For each tuning parameter $\lambda \in \{\lambda_1, \ldots, \lambda_m\}$, compute the average error over all folds,

$$CV(\lambda) = \frac{1}{K} \sum_{k=1}^{K} e_k(\lambda).$$

4. Choose $\hat{\lambda}$ such that $\hat{\lambda} := \text{argmin}_{\lambda}\{CV(\lambda)\}$.

Further discussion of this process can be found here. 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.

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)
YOUR CODE HERE
```

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

YOUR ANSWER HERE

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

```
YOUR CODE HERE
```

4. 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 λ you provide and plots the Mean Square Prediction Error (MSPE) for each λ. Use `ridgereg.cv` for $\lambda \in \{0.5, 1.0, 1.5, \dots, 3.5, 4.0\}$.

```
library(MXM)
YOUR CODE HERE
```

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

YOUR ANSWER HERE

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

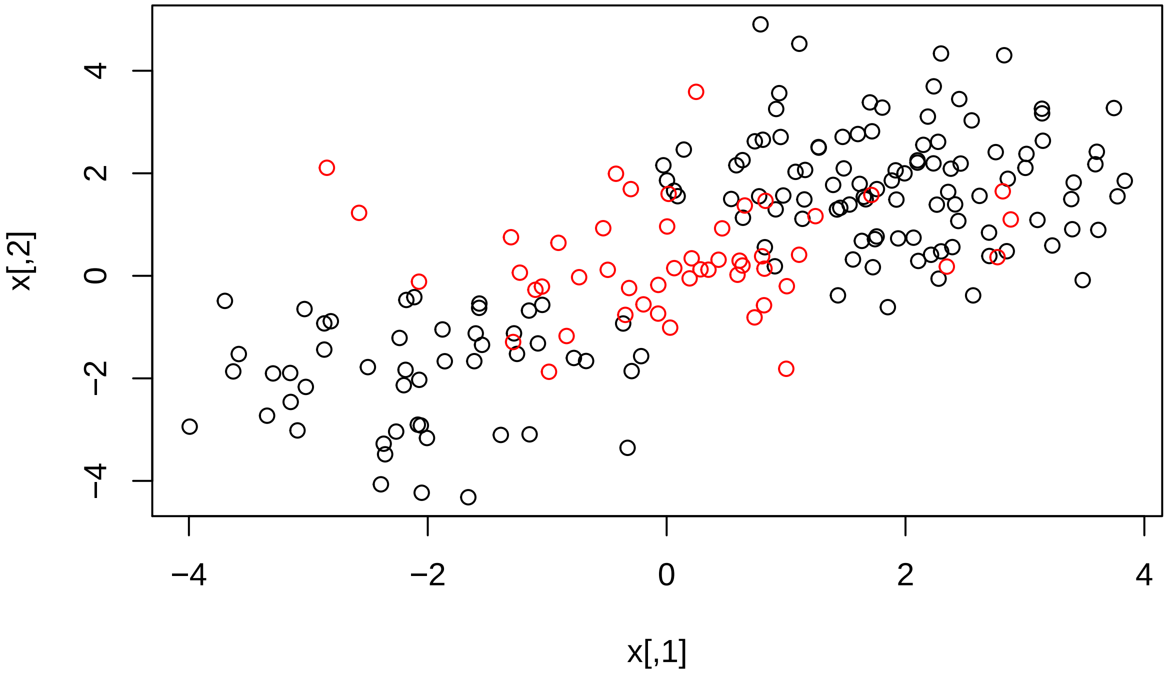
```
YOUR CODE HERE
```

B. SVM

For this section we will use the `svm` function in package `e1071`. Let us walk through a simple example (originally found here) to see how the `svm` function works:

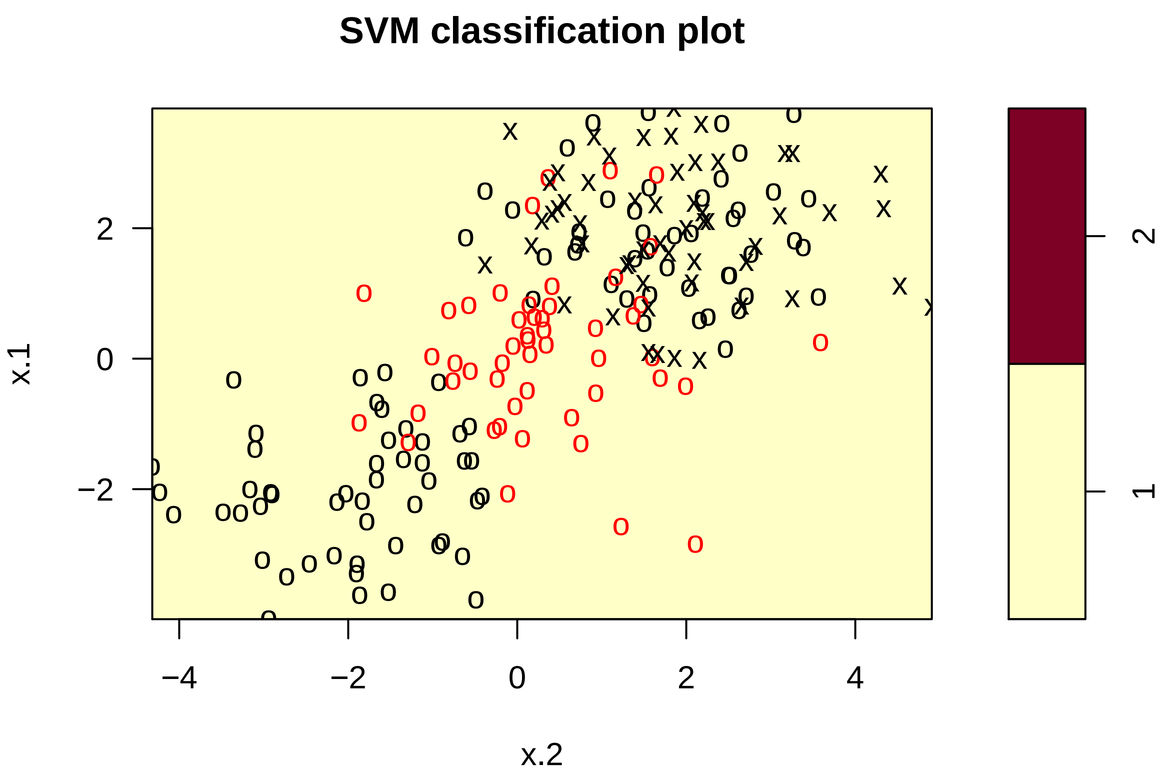
```r
set.seed(13)
x=matrix(rnorm(200*2) , ncol =2)
x[1:100 ,] = x[1:100 ,] + 2
x[101:150 ,]= x[101:150,] - 2
y=c(rep(1 ,150) ,rep(2 ,50) )
dat=data.frame(x=x,y=as.factor(y))
```

```r
plot(x, col =y)
```



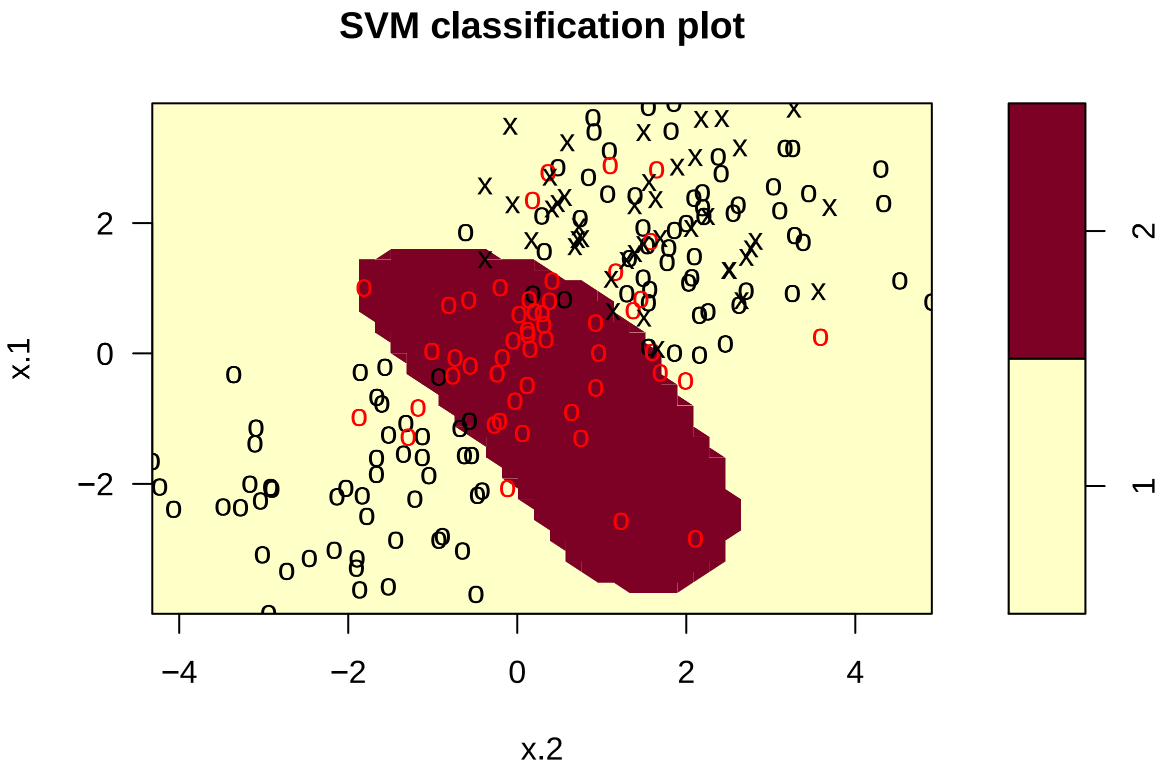
Note here that our data are NOT linearly seperable! In fact, it does not appear that a linear seperation rule will work here. Let us verify this using a linear kernel with our SVM:

```r
library(e1071)
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 C (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 C. 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)
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?

```
YOUR CODE AND ANALYSIS HERE
```

2. 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.**

```
YOUR CODE HERE
```

3. Tune a SVM model with a linear kernel on the training set. Use $cost \in \{0.001, 0.01, 0.1, 1, 5, 10, 100\}$ (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

```
YOUR CODE HERE
```