INSH5301 Intro Computational Statistics

Ali Banijamali

04/13/2020

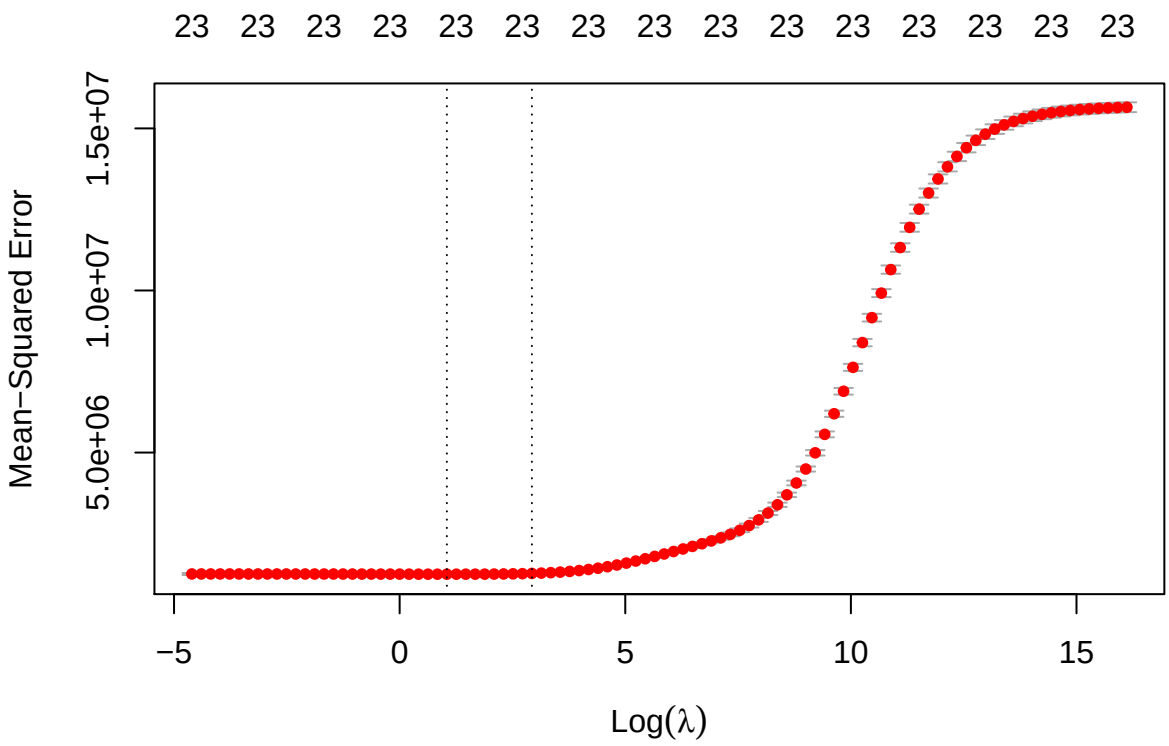
1. For this example, we are trying to predict the price of a diamond. You can get the data from the ggplot2 package using data(diamonds) -you need to load ggplot2 first with library(ggplot2)-.

```
# Required Packages:
# install.packages('glmnet')
# install.packages('e1071')
library(ggplot2)
library(glmnet)
library(e1071)

diamond <- data.frame(diamonds)
```

1.a. Divide the dataset into two equal-sized samples, the in-sample and the out-sample, the samples have to be random. Estimate an elastic net model using the in-sample data for three different values of α (0, 0.5, and 1), using cv.glmnet to find the best lambda level for each value of α. Remember that glmnet prefers that data be in a numeric matrix, therefore, you need to transform any factor into dummies manually.

```
# 1. Converting factors to dummy values:
# (Also, glmnet prefers numeric values)

diamonds.fixed <- cbind(diamond[c(1, 5:6, 8:10)],
                        cut.good=ifelse(diamond$cut=='Good', 1, 0),
                        cut.vgood=ifelse(diamond$cut=='Very Good', 1, 0),
                        cut.premium=ifelse(diamond$cut=='Premium', 1, 0),
                        cut.ideal=ifelse(diamond$cut=='Ideal', 1, 0),

                        col.E=ifelse(diamond$color=='E', 1, 0),
                        col.F=ifelse(diamond$color=='F', 1, 0),
                        col.G=ifelse(diamond$color=='G', 1, 0),
                        col.H=ifelse(diamond$color=='H', 1, 0),
                        col.I=ifelse(diamond$color=='I', 1, 0),
                        col.J=ifelse(diamond$color=='J', 1, 0),

                        clar.SI2=ifelse(diamond$clarity=='SI2', 1, 0),
                        clar.SI1=ifelse(diamond$clarity=='SI1', 1, 0),
                        clar.VS2=ifelse(diamond$clarity=='VS2', 1, 0),
                        clar.VS1=ifelse(diamond$clarity=='VS1', 1, 0),
                        clar.VVS2=ifelse(diamond$clarity=='VVS2', 1, 0),
                        clar.VVS1=ifelse(diamond$clarity=='VVS1', 1, 0),
                        clar.IF=ifelse(diamond$clarity=='IF', 1, 0),

                        diamond[7]
```

```
)

diamonds.fixed <- as.matrix(diamonds.fixed)

# 2. Preparing samples:
set.seed(1)
train.indices <- sample(1:nrow(diamonds), nrow(diamonds)/2)

d.specs <- diamonds.fixed[ ,1:23] # X
d.price <- diamonds.fixed[ ,24] # Y

train.specs <- d.specs[train.indices, ] # X_train
train.price <- d.price[train.indices] # Y_train

test.specs <- d.specs[-train.indices, ] # X_test
test.price <- d.price[-train.indices] # Y_test

# 3. Preparing list of candid alpha and lamdas:
alphas <- c(0, 0.5, 1)
lambdas <- 10^seq(7, -2, length=100) # 100 values from 10^-2 to 10^7

# 4. Using cv.glmnet to find the best value of lamda for our alphas:
# 4.a. alpha = 0: Purely Ridge model
ridge.model <- cv.glmnet(train.specs, train.price, alpha=0, lambda=lambdas)
# Choosing best lambda:
plot(ridge.model, xvar="lambda")
```




```
best.lambda.r <- ridge.model$lambda.min
cat('The best Lambda for alpha=0 (Pure Ridge model) is: ', best.lambda.r)
```

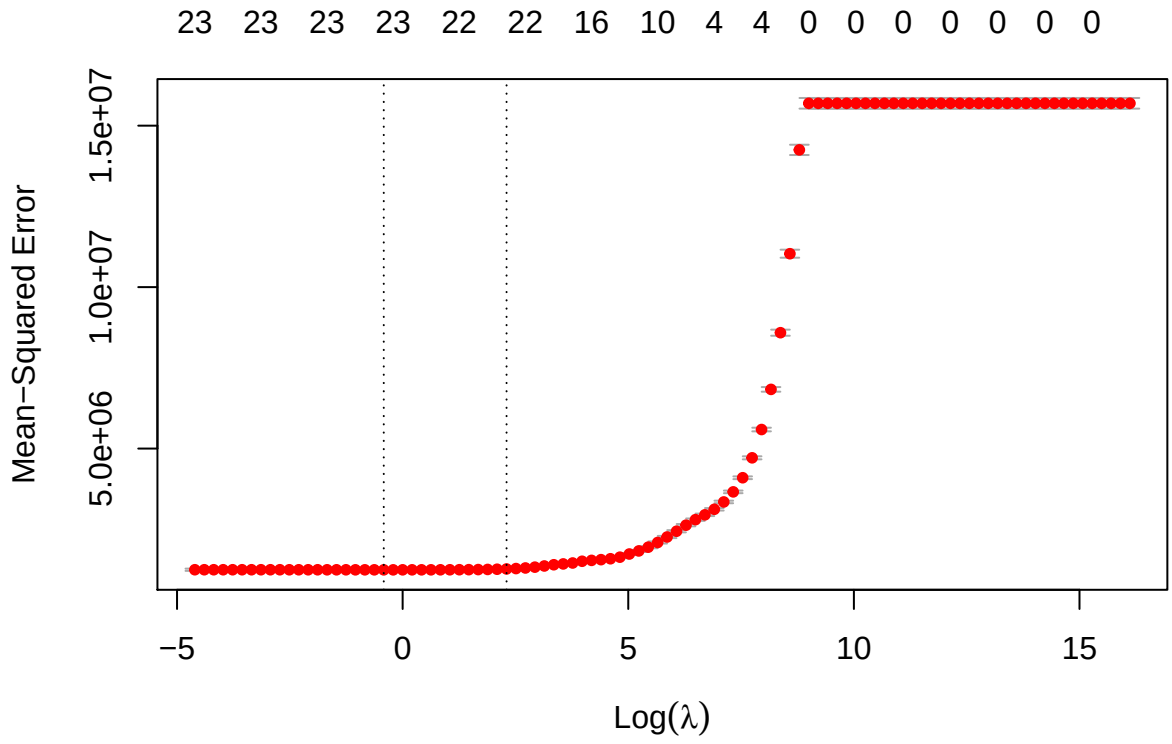
```
## The best Lambda for alpha=0 (Pure Ridge model) is:  2.848036
```

```
# coeff's for best lamda:
predict(ridge.model, type="coefficients", s=best.lambda.r)
```

```
## 24 x 1 sparse Matrix of class "dgCMatrix"
##                         1
## (Intercept)  2937.37717
## carat       11099.80880
## depth         -71.68237
## table         -28.96250
## x            -988.46536
## y              49.27181
## z             -45.62187
## cut.good      543.54768
## cut.vgood     713.28221
## cut.premium   755.23884
## cut.ideal     803.90914
## col.E        -190.50129
## col.F        -274.91753
## col.G        -465.29484
## col.H        -961.84157
## col.I       -1444.25711
## col.J       -2349.57942
```

```
## clar.SI2     2363.00493
## clar.SI1     3301.73169
## clar.VS2     3916.10238
## clar.VS1     4211.17816
## clar.VVS2    4592.45003
## clar.VVS1    4642.48545
## clar.IF      5032.09575
```

```
# 4.b. alpha = 0.5: Half Ridge model, Half Lasso model
rl.model <- cv.glmnet(train.specs, train.price, alpha=0.5, lambda=lambdas)
# Choosing best lambda:
plot(rl.model, xvar="lambda")
```




```
best.lambda.rl <- rl.model$lambda.min
cat('The best Lambda for alpha=0.5 (0.5Ridge/0.5Lasso model) is: ', best.lambda.rl)
```

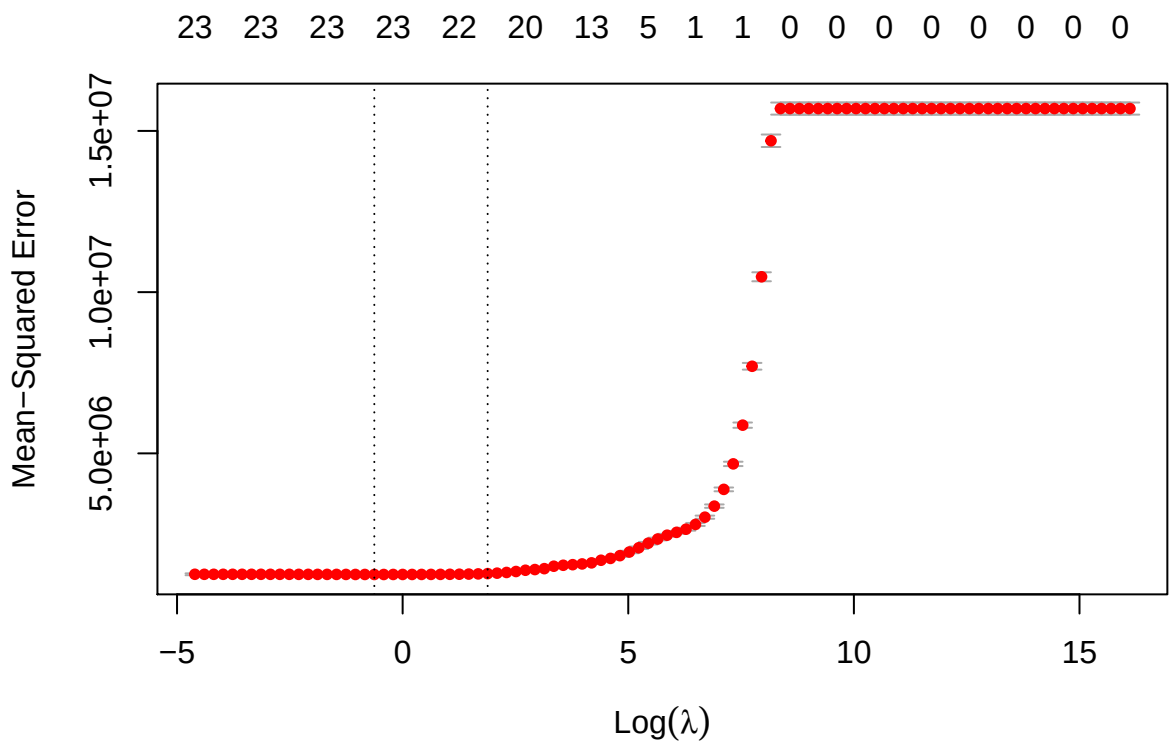
```
## The best Lambda for alpha=0.5 (0.5Ridge/0.5Lasso model) is:  0.6579332
```

```
# coeff's for best lamda:
predict(rl.model, type="coefficients", s=best.lambda.rl)
```

```
## 24 x 1 sparse Matrix of class "dgCMatrix"
##                       1
## (Intercept)  3333.29997
## carat       11263.27779
## depth         -74.98530
## table         -29.07363
## x           -1042.32512
```

```
## y              33.42100
## z             -40.15362
## cut.good      524.80451
## cut.vgood     693.31779
## cut.premium   735.24678
## cut.ideal     784.96686
## col.E        -186.89753
## col.F        -270.18167
## col.G        -462.12571
## col.H        -960.81262
## col.I       -1446.50731
## col.J       -2355.54465
## clar.SI2     2443.33896
## clar.SI1     3386.45024
## clar.VS2     3999.10497
## clar.VS1     4294.99645
## clar.VVS2    4673.65739
## clar.VVS1    4721.95875
## clar.IF      5111.02221
```

```
# 4.c. alpha = 0.5: Purely Lasso model
lasso.model <- cv.glmnet(train.specs, train.price, alpha=1, lambda=lambdas)
# Choosing best lambda:
plot(lasso.model, xvar="lambda")
```




```
best.lambda.l <- lasso.model$lambda.min
cat('The best Lambda for alpha=1 (Pure Lasso model) is: ', best.lambda.l)
```

```
## The best Lambda for alpha=1 (Pure Lasso model) is:  0.5336699
```

```
# coeff's for best lamda:
predict(lasso.model, type="coefficients", s=best.lambda.l)
```

```
## 24 x 1 sparse Matrix of class "dgCMatrix"
##                        1
## (Intercept)  3488.63000
## carat       11274.05152
## depth         -76.49123
## table         -29.27922
## x           -1038.03413
## y              16.82762
## z             -28.73017
## cut.good      514.00106
## cut.vgood     682.52735
## cut.premium   723.56235
## cut.ideal     773.73930
## col.E        -183.82763
## col.F        -267.01351
## col.G        -458.92758
## col.H        -957.74497
## col.I       -1443.26678
## col.J       -2352.08605
## clar.SI2     2422.21750
## clar.SI1     3365.72014
## clar.VS2     3978.11603
## clar.VS1     4273.76897
## clar.VVS2    4651.95829
## clar.VVS1    4699.83022
## clar.IF      5088.41438
```

1.b. Choose the value of α and λ that minimize the MSE, and then test the model using the outsample data. That is, compute the MSE using the out-of-sample data.

ANS.

```
# Calculating the MSE for the 3 models using train data (in-sample):
# 1. Pure Ridge model (alpha=0):
pred_price.train.r <- predict(ridge.model$glmnet.fit, s=best.lambda.r, newx=train.specs)
mse.train.r <- sum((train.price - pred_price.train.r)^2) / nrow(train.specs)
mse.train.r
```

```
## [1] 1239784
```

```
# 2. Half Ridge/Half Lasso Model (alpha=0.5):
pred_price.train.rl <- predict(rl.model$glmnet.fit, s=best.lambda.rl, newx=train.specs)
mse.train.rl <- sum((train.price - pred_price.train.rl)^2) / nrow(train.specs)
mse.train.rl
```

```
## [1] 1238833
```

```
# 3. Pure Lasso Model (alpha=1):
pred_price.train.l <- predict(lasso.model$glmnet.fit, s=best.lambda.l, newx=train.specs)
mse.train.l <- sum((train.price - pred_price.train.l)^2) / nrow(train.specs)
mse.train.l
```

```
## [1] 1238966
```

As we can see, the lowest MSE for training set, is for the second model (Half Ridge/Half Lasso, $\alpha = 0.5$). However, the models are very close. I actually tried different test samples by changing the seeds, and I got different results with different training sets. Now let's calculate the out-sample MSE using this model:

```
# MSE for Half Ridge/Half Lasso Model (alpha=0.5) w/ out-sample (test) data:
pred_price.test.rl <- predict(ridge.model$glmnet.fit, s=best.lambda.rl, newx=test.specs)
mse.test.rl <- sum((test.price - pred_price.test.rl)^2) / nrow(test.specs)
mse.test.rl
```

```
## [1] 1317116
```

1.c. Compare your out-of-sample results to regular a multiple regression using all the variables in the dataset. That is; (1) fit the standard regression model using the in-sample data and using all the variables, (2) predict the out-of-sample using the estimated paramters in (1), and, (3) compute MSE. Which model works best out-of sample, the multivariate regression or the one estimated in (b)?

ANS.

```
# 1. Fitting our data w/ a multiple regression model:
mult_regr.model <- lm(train.price ~ train.specs)

# 2. Predicting outsample using our multiple regression model:
pred_price.test.mult_regr <- cbind(1, test.specs) %*% mult_regr.model$coefficients

# 3. Calculating MSE for test (out-sample) data:
mse.test.mult_regr <- sum((test.price - pred_price.test.mult_regr)^2) / nrow(test.specs)
mse.test.mult_regr
```

```
## [1] 1317169
```

As we can see, the Half Ridge/Half Lasso model did marginally better than the multiple regression model (It has a lower MSE).

2. For this example, we are going to predict the quality of the diamond. For that you need to create a dummy that is equal to one if the quality is Premium or ideal and zero otherwise.

```
diamond <- data.frame(diamonds)

diamonds.fixed <- cbind(diamond[c(1, 5:10)],

                        col.E=ifelse(diamond$color=='E', 1, 0),
                        col.F=ifelse(diamond$color=='F', 1, 0),
                        col.G=ifelse(diamond$color=='G', 1, 0),
                        col.H=ifelse(diamond$color=='H', 1, 0),
                        col.I=ifelse(diamond$color=='I', 1, 0),
                        col.J=ifelse(diamond$color=='J', 1, 0),

                        clar.SI2=ifelse(diamond$clarity=='SI2', 1, 0),
                        clar.SI1=ifelse(diamond$clarity=='SI1', 1, 0),
                        clar.VS2=ifelse(diamond$clarity=='VS2', 1, 0),
                        clar.VS1=ifelse(diamond$clarity=='VS1', 1, 0),
```

```
                        clar.VVS2=ifelse(diamond$clarity=='VVS2', 1, 0),
                        clar.VVS1=ifelse(diamond$clarity=='VVS1', 1, 0),
                        clar.IF=ifelse(diamond$clarity=='IF', 1, 0),

                        cut=as.factor(ifelse(diamonds$cut=='Premium' |
                                             diamonds$cut=='Ideal', 1, 0))
)
```

2.a. Divide the data into an in-sample and out-sample as before, and estimate an SVM using at least two different kernels and use tune to find the best cost level for each.

ANS.

```
# Since the dataset is large and therefore the model takes so long to converge,
# I take %25 of the data for training and %25 percent for test:
set.seed(1)
down.sample <- sample(1:nrow(diamonds), nrow(diamonds)/2)
diamonds.fixed.ds <- diamonds.fixed[down.sample, ]
row.names(diamonds.fixed.ds) <- NULL


# 1. Selecting the test/train samples:
train.indices.ds <- sample(1:nrow(diamonds.fixed.ds), nrow(diamonds.fixed.ds)/2)
train.data <- diamonds.fixed.ds[train.indices.ds, ]
test.data <- diamonds.fixed.ds[-train.indices.ds, ]

# 2. SVM:
# C values:
costvalues <- 10^seq(-3,2,1)
# 2.1. Linear Kernl:
svm.linear <- tune(svm, cut~., data=train.data,
                   ranges=list(cost=costvalues), kernel="linear")
summary(svm.linear)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##    10
##
## - best performance: 0.2250652
##
## - Detailed performance results:
##    cost     error  dispersion
## 1 1e-03 0.2810542 0.006465206
## 2 1e-02 0.2587335 0.007860592
## 3 1e-01 0.2399716 0.009221235
## 4 1e+00 0.2261777 0.009487211
## 5 1e+01 0.2250652 0.011762375
## 6 1e+02 0.3963678 0.222798368
```

```r
# 2.2. Radial Kernel:
svm.radial <- tune(svm, cut~., data=train.data,
                   ranges=list(cost=costvalues), kernel="radial")
summary(svm.radial)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   100
##
## - best performance: 0.1520967
##
## - Detailed performance results:
##    cost     error  dispersion
## 1 1e-03 0.3439390 0.018613235
## 2 1e-02 0.3339279 0.018132210
## 3 1e-01 0.2246949 0.010127872
## 4 1e+00 0.1965902 0.009019012
## 5 1e+01 0.1675961 0.012551599
## 6 1e+02 0.1520967 0.008916130
```

Based on our samller samples, for the linear model we have the smallest error (%23.01068) for C=10, and for the radial model, the smallest error is %15.22426 for C=100. This shows that the radial model is much better than the linear model.

2.b. Choose the kernel and cost with the best in-sample performance, and then test that model outofsample using the out-sample data. That is, compute the percentage of correct predictions using the out-of-sample data.

ANS.

```r
svm.bestmodel.pred <- predict(svm.radial$best.model, newdata=test.data)

# The percentage of correct predictions using the out-of-sample data:
sum(svm.bestmodel.pred==test.data$cut)*100/length(test.data$cut)
```

```
## [1] 85.25028
```

~%85 correct prediction.

2.c. Compare your out-of-sample results with a logistic regression using all the variables in the dataset. That is; (1) fit the standard logistic regression model using the in-sample data and using all the variables, (2) predict the out-of-sample data using the estimated paramters in (1), and, (3) compute the model accuracy as the percentage of correct predictions. Which model works best out-of sample, the logistic regression or the one estimated in (b)?

ANS.

```r
# 1. Logistic regression model using in-sample (train data):
logit <- glm(cut~., data=train.data, family="binomial")
```

```r
# 2. Making predictions using the out-sample (test data):
logit.p <- predict(logit, newdata=test.data, type="response")
logit.pred <- round(logit.p)
# The "reponse" option was used to the get predicted probabilities,
# and then the results were rounded, so that any predicted prob > 0.5 is a 1,
# and vice versa for 0.

# 3. Percentage of correct predictions:
sum(logit.pred==test.data$cut)*100/length(test.data$cut)
```

```
## [1] 77.51576
```

Based on the percentage of correct predictions, the SVM model did a better job (~%85 correct prediction), compared to the logistic regression model with ~%77 correct prediction rate on test data.