HW_12_Gupta_S

Sumit Gupta

December 2, 2017

I am using the mtcars dataset which is a default dataset in R. It has 11 variables and 32 observations. The data conprises fuel consumption and 10 aspects of automobile design and performance for 32 automobiles.

1. Use your dataset with a continuous dependent variable:

```r
# Loading the default dataset mtcars which has 11 variables
datasets::mtcars
```

```
##                      mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4           21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710          22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Valiant             18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 240D           24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230            22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2       26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

```r
library(MASS)
library(glmnet)
```

```
## Warning: package 'glmnet' was built under R version 3.4.3

## Loading required package: Matrix

## Loading required package: foreach
```

```
## Loaded glmnet 2.0-13
```

```r
x <- model.matrix(mpg ~ ., data=mtcars)[,-1]
y <- mtcars$mpg

# # Dividing the dataset into train and test set
set.seed(1)
train <- sample (1:nrow(x),nrow(x)/2)
test <- (-train)
trainx <- x[train,]
testx <- x[test,]
trainy <- y[train]
testy <- y[test]
```

```r
# Estimating the elastic net model with alpha as 0,0.5 and 1
fit.lasso <- cv.glmnet(trainx, trainy, alpha=1)
```

```
## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations
## per fold
```

```r
fit.ridge <- cv.glmnet(trainx, trainy, alpha=0)
```

```
## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations
## per fold
```

```r
fit.elnet <- cv.glmnet(trainx, trainy, alpha=.5)
```

```
## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations
## per fold
```

```r
fit.lasso$lambda.min
```

```
## [1] 0.2652519
```

```r
fit.ridge$lambda.min
```

```
## [1] 1.320165
```

```r
fit.elnet$lambda.min
```

```
## [1] 0.0905757
```

So, for elastic net regression, when $\alpha = 0.5$ will give the best value of $\lambda = 1$ which is 0.09

b. Choose the alpha (and corresponding lambda) with the best results (lowest error), and then test that model out-of-sample using the out-sample data.

```r
yhat.e <- predict(fit.elnet$glmnet.fit, s=fit.elnet$lambda.min, newx=testx)
yhat.e
```

```
##                             1
## Mazda RX4           21.085107
## Datsun 710          26.028455
## Duster 360          11.502748
## Merc 280            20.114704
## Merc 280C           20.114704
## Merc 450SL          15.925471
## Cadillac Fleetwood 10.927687
## Honda Civic         36.446083
## Toyota Corona       26.376805
## Dodge Challenger    16.813817
```

```
## AMC Javelin        20.608146
## Camaro Z28         16.561853
## Fiat X1-9          29.716761
## Lotus Europa       19.393899
## Ferrari Dino        7.794487
## Maserati Bora      -3.804029
```

c. Compare your out-of-sample results to regular multiple regression: fit the regression model in-sample, predict yhat out-of-sample, and estimate the error. Which works better?

```
lmout <- lm(trainy~trainx)
yhat.r <- cbind(1,testx) %*% lmout$coefficients

# Mean Square error for Multiple regression fit
mse.reg <- sum((testy - yhat.r)^2)/nrow(testx)
mse.reg
```

```
## [1] 337.9995
```

```
# Mean square error for Elastic net regression fit
mse.e <- sum((testy - yhat.e)^2)/nrow(testx)
mse.e
```

```
## [1] 46.92981
```

Thus, we observe that MSE for Elastic net model is 46 way less than for Linear regression model with value 337. Hence, Elastic net works much better than Multiple Linear Regression.

d. Which coefficients are different between the multiple regression and the elastic net model? What, if anything, does this tell you substantively about the effects of your independent variables on your dependent variable?

```
coef(fit.elnet, fit.elnet$lambda.min)
```

```
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                       1
## (Intercept)  0.62843292
## cyl          1.18059668
## disp         .
## hp          -0.05372684
## drat         9.72904942
## wt           .
## qsec         .
## vs           .
## am           0.46653508
## gear        -2.48914955
## carb        -2.29254591
```

```
lmout$coefficients
```

```
##   (Intercept)     trainxcyl    trainxdisp      trainxhp    trainxdrat
## -127.77340584   10.29870129   -0.01960153   -0.22132294   24.96599634
##      trainxwt    trainxqsec      trainxvs      trainxam    trainxgear
##    9.63213344    0.27590333    3.83935521    1.71842469    1.88720664
##    trainxcarb
##   -6.13805866
```

On comparing, the coefficients of the Elastic net model with the Multiple Linear Regression model we observe that, 4 coefficients of Elastic net which are disp, wt, qsec, vs have been shrunk to 0 by the model unlike

Linear regression. Thus, Elastic net has a property of coefficient shrinkage and thereby feature reduction.

Thus, this means that these 4 variables donot affect the dependent variable y (mpg).

2. Repeat the same process using your dataset with a binary dependent variable:

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

Heart Disease data set consists of 14 attributes data. All the attributes consist of numeric values. First 13 variables will be used for predicting 14th variables. The target variable is at index 14 which represents Absence/Presence of heart disease (0/1)

```
library(e1071)
# Loading the data set
heart.data <- read.csv("heart_tidy.csv", header = T, sep = ",")

x <- heart.data[,c(1:13)]
y <- heart.data$Target

dat <- data.frame(x=x, y=as.factor(y))
# Dividing into train and test set
set.seed(1)
train <- sample (1:nrow(x),nrow(x)/2)
test <- (-train)
traindat <- dat[train,]
testdat <- dat[test,]


# Estimating using linear Kernel
costvalues <- 10^seq(-3,2,1)
tuned.svm_1 <- tune(svm, y~., data=traindat, ranges=list(cost=costvalues), kernel="linear")
yhat_1 <- predict(tuned.svm_1$best.model, newdata = testdat)
sum(yhat_1==testdat$y)/length(testdat$y)
```

```
## [1] 0.7666667
```

```
summary(tuned.svm_1)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   0.1
##
## - best performance: 0.08666667
##
## - Detailed performance results:
##    cost      error dispersion
## 1 1e-03 0.40666667 0.10159226
## 2 1e-02 0.10666667 0.08999314
## 3 1e-01 0.08666667 0.05488484
## 4 1e+00 0.11333333 0.06324555
## 5 1e+01 0.12000000 0.06885304
```

```
## 6 1e+02 0.12666667 0.07336700
```

```r
# Estimating using Radial Kernel

tuned.svm_2 <- tune(svm, y~., data=traindat, ranges=list(cost=costvalues), kernel="radial")
yhat_2 <- predict(tuned.svm_2$best.model, newdata = testdat)
sum(yhat_2==testdat$y)/length(testdat$y)
```

```
## [1] 0.78
```

```r
summary(tuned.svm_2)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 0.12
##
## - Detailed performance results:
##    cost     error dispersion
## 1 1e-03 0.4066667 0.16762686
## 2 1e-02 0.4066667 0.16762686
## 3 1e-01 0.1866667 0.13259052
## 4 1e+00 0.1200000 0.08777075
## 5 1e+01 0.1800000 0.08344437
## 6 1e+02 0.1800000 0.08344437
```

Thus radial kernel in this case outperforms linear kernel with a cost of 1 and performance of 0.78 better than linear with 0.76

b. Chose the kernel and cost with the best results, and then test that model out-of-sample using the out-sample data.

```r
tuned.svm_3 <- tune(svm, y~., data=traindat, ranges=list(cost=1), kernel="radial")
yhat <- predict(tuned.svm_3$best.model,newdata=testdat)
table(predicted=yhat,truth=testdat$y)
```

```
##          truth
## predicted  0  1
##         0 63 23
##         1 10 54
```

```r
sum(yhat==testdat$y)/length(testdat$y)
```

```
## [1] 0.78
```

Accuracy comes as 78% with cost taken as 1 for radial kernel from previous result.

c. Compare your results to a logistic regression: fit the logit in-sample, predict yhat out-of-sample, and estimate the accuracy. Which works better?

```r
logit.reg2 <- glm(y~., data = traindat, family = "binomial")

logit.reg.pred <- predict(logit.reg2, testdat,  type = "response")
```

```r
yhat_3 <- round(logit.reg.pred) # rounding off

sum(yhat_3==testdat$y)/length(testdat$y)
```

```
## [1] 0.7733333
```

Thus, we observe that the Radial kernel (0.78) gives slightly better result than the logistic regression model.

d. Can you make any guesses as to why the SVM works better (if it does)? Feel to speculate, or to research a bit more the output of svm, the meaning of the support vectors, or anything else you can discover about SVMs (no points off for erroneous speculations!).

Ans: If you restrict yourself to linear kernels, both SVMs and Logistic regression (LR) will give almost identical performance and in some cases, LR will beat SVM. If the data is linearly separable in the input space, then LR is usually preferred as it outputs probabilities instead of hard labels and you can fine tune your performance by plotting the ROC curve and figuring out the right threshold.

The natural advantage that SVMs have over LRs is the non-linearity obtained via the use of non-linear kernels. If we compare logistic regression with SVMs with non-linear kernels, then SVMs beat LRs hands down. If the data is linearly separable in the input space, then LRs give performance comparable to SVMs, but if the data is non-linearly separable, then LRs gradually worsen depending on how bad the non-linearity is in the data and SVMs win out.

Also, Non-regularized logistic regression techniques don't work well (in fact, the fitted coefficients diverge) when there's a separating hyperplane, because the maximum likelihood is achieved by any separating plane, and there's no guarantee that you'll get the best one. What you get is an extremely confident model with poor predictive power near the margin.

SVMs get you the best separating hyperplane, and they're efficient in high dimensional spaces. They're similar to regularization in terms of trying to find the lowest-normed vector that separates the data, but with a margin condition that favors choosing a good hyperplane. A hard-margin SVM will find a hyperplane that separates all the data (if one exists) and fail if there is none; soft-margin SVMs (generally preferred) do better when there's noise in the data.

Additionally, SVMs only consider points near the margin (support vectors). Logistic regression considers all the points in the data set. Which you prefer depends on your problem.

Logistic regression is great in a low number of dimensions and when the predictors don't suffice to give more than a probabilistic estimate of the response. SVMs do better when there's a higher number of dimensions, and especially on problems where the predictors do certainly (or near-certainly) determine the responses.