Leave one out Method

Daivd Leslie

Monday, March 24, 2015

The purpose of the code is to evaluate the effectiveness of the leave one out method for determining the predictive power of each model given. In order to acomplish this, the model given will be trained with data that has been reduced by one sample. The sample that is left out will then be used as a new data point to be predicted by the given model. By looking at the differences in the predicted value by the model and the actual value recorded, a greater insight will be gained as to how well the model preforms when predicting new data. Let's first start by generating a popluation, which we will sample from.

```
# Create predictor variables
set.seed(10)
pop = 9000
x1 = rnorm(pop)
x2 = rnorm(pop)
x3 = rnorm(pop)

# Create noise term
noise = rnorm(pop,0,3)

# Generate responce: additive model plus noise, intercept = 0
y = 2*x1 + x2 + 3*x3 + noise

# Organize predictors and responce in data frame
pop_data = data.frame(y, x1, x2, x3)

head(pop_data)
```

```
##            y          x1          x2         x3
## 1 -0.9350841  0.01874617 -0.97718623 -0.9620658
## 2 -5.0348547 -0.18425254 -1.15456052 -0.7386388
## 3 -3.3519495 -1.37133055 -0.05577223  0.4370131
## 4 -6.7564424 -0.59916772  0.61778059 -0.1895484
## 5  1.8298094  0.29454513  1.38595893  0.2462712
## 6  1.6258103  0.38979430  1.72930724 -0.1739482
```

Now that we have our population, lets take a sample of our data and verify that the results generated by the model are reasonalbe estimates of the true model's coeffcients (Remebering that they should be close to the exspected values of: x1=2, x2=1, and x3=3).

```
# Create sample population
n = 375
samp = pop_data[sample(nrow(pop_data), n), ]
# Create model(s)
mod = lm(y ~ x1 + x2 + x3, data=samp)

summary(mod)
```

```
##
```

```
## Call:
## lm(formula = y ~ x1 + x2 + x3, data = samp)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.8807 -1.9683  0.0068  2.0143  8.3742
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -0.2321     0.1596  -1.454    0.147
## x1            1.9598     0.1501  13.057  < 2e-16 ***
## x2            1.2881     0.1576   8.172 4.83e-15 ***
## x3            2.9242     0.1590  18.394  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.083 on 371 degrees of freedom
## Multiple R-squared:  0.6039, Adjusted R-squared:  0.6007
## F-statistic: 188.5 on 3 and 371 DF,  p-value: < 2.2e-16
```

Looking at the model summary, it appears that calculated coeffeicents are fairly close to actual coefficents. It also appears that approximately 60% of the variability can be explained by the predictor variables. This r squared value seems reasonalbe since there is an extra error term incorporated into the model (noise). Now that we feel cofident about the sample taken, lets see how good the models generated will be at predicting new data. In order to do this, lets apply the leave one out method to the data. The two things to note for this method are:

1. How close are the predicted values to the actual vales.
2. How much the coeffcients vary between models

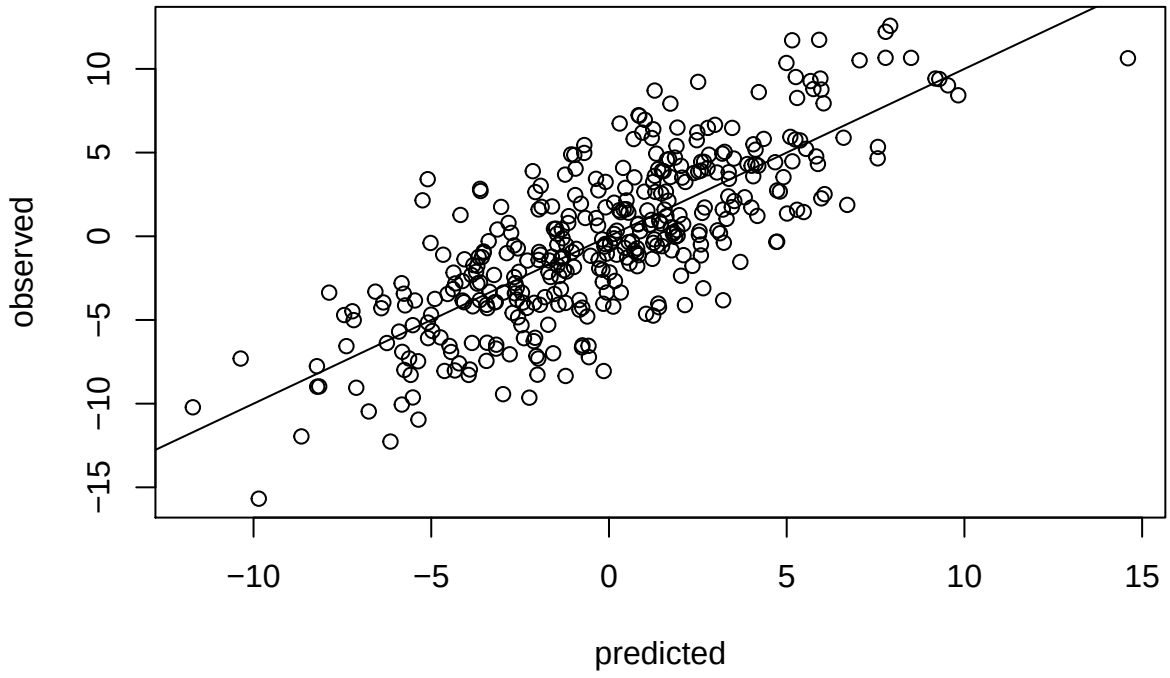
```
# Function(s)
get_R2 = function(obs, pred, na.rm=FALSE) {
  if (na.rm) {
    true = !(is.na(obs) | is.na(pred))
    obs = obs[true]
    pred = pred[true]
  }
  SSerr = sum((obs - pred)^2)
  SStot = sum((obs - mean(obs))^2)
  R2 = 1 - SSerr / SStot
  return(R2)
}

Leave_one_out = function(data, model) {
  matrixCoef = matrix(NA, nrow=nrow(data), ncol=length(coef(model)))
  colnames(matrixCoef) = names(coef(model))
  predicted = NULL
  rSquared = NULL
  for(i in 1:nrow(data)) {
    trainSet = data[-i,]
    testSet = data[i,]
    trainMod = update(model, data=trainSet)
    matrixCoef[i, ] = coef(trainMod)
```

```
    predicted[i] = predict(trainMod, newdata = testSet)
    rSquared[i] = summary(trainMod)$r.squared
    rSquaredCalc = get_R2(residuals(trainMod) + predict(trainMod), predict(trainMod))
  }
  observed = residuals(mod) + predict(mod)
  finalMatrix = data.frame(matrixCoef, predicted, observed, rSquared, rSquaredCalc)
  names(finalMatrix)[1] = 'intercept'
  return(finalMatrix)
}

fit_mod = lm(y ~ x1 + x2 + x3, data=samp)
resultsMatrix = Leave_one_out(samp, fit_mod)

plot(observed ~ predicted, data=resultsMatrix)
abline(a=0, b=1)
```




```
head(resultsMatrix)
```

```
##       intercept       x1       x2       x3  predicted    observed
## 2854 -0.2281022 1.954284 1.285724 2.930600  0.7098677 -0.85620190
## 2649 -0.2441764 1.951138 1.307448 2.914592  1.2452626  6.39325405
## 4048 -0.2369202 1.959112 1.296238 2.923439 -1.5478888  0.44202209
## 7097 -0.2270942 1.965208 1.290371 2.922847  1.9310070  0.03740653
## 777  -0.2315083 1.960572 1.288177 2.924740  4.6747159  4.42426709
## 8754 -0.2276340 1.954337 1.291563 2.923383 -2.4242752 -3.96262376
```

```
##       rSquared rSquaredCalc
## 2854 0.6041157     0.603014
## 2649 0.6048970     0.603014
## 4048 0.6042899     0.603014
## 7097 0.6042638     0.603014
## 777  0.6029303     0.603014
## 8754 0.6034851     0.603014
```

From the results, it appears that the coeffeicents are not varying too much, which would suggest that the models are not overfitting the data. However, the differences in the predicted and actual values do vary slightly with each model prediction. Despite this varation, all the models constructed have relatively the same r squared value of roughtly 60%, which would suggest that all of the models have about the same predictive power.