Variable Selection

Group 5

December 12, 2015

In order to find the outliers, we will train a model using a section of the dataset that does not include our first 400 observations. We will then fit that model to the dataset we are interested in. This will avoid the 50 outliers from impacting the estimated coefficients in such a way as to mask themselves. We have therefore used observations 401 to 3000 of the synthetic regression data to train a model, which we will test on observations 1 to 400.

We will use a stepwise algorithm in order to find out which of the variables to include in the dataset. The major flaw in this method is that it fails to take into account correlations between the different variables. In order to get around this issue, we create groups of highly correlated variables. Therefore instead of, for instance, making a stepwise choice whether to include variable one, variable two, or variable three, the algorithm chooses whether to include variables one or two *and* three together.

```
file <- '/home/beeb/Documents/Data_Science/Data/Stat/'
library(ggplot2)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
##
## The following objects are masked from 'package:stats':
##
##     filter, lag
##
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
setwd(file)
synth.reg.test <- read.table('synthetic_regression.txt', header = TRUE, nrows= 400)
synth.reg.train <- read.table('synthetic_regression.txt', header = TRUE, nrows = 3000)
synth.reg.train <- synth.reg.train[401:3000,]
```

This section of code identifies the correlated x's:

```
# This bit of code figures out which of the xs need to be put in groups together
# The first thing we're going to do is see if any of the x's are correlated
cors <- cor(synth.reg.train[2:ncol(synth.reg.train)])
m <- ncol(cors)

# Pick out those that have correlation above 0.5
together <- apply(abs(cors)>0.5, 2, which)
```

We now have a list, 'together', which contains all the variables that should go together. We will scan through this list and see which groups of variables we can add to our current set of variables in order to provide a model with the highest log likelihood.

```r
# Initialise a dataframe for adding variables to and a list to show which groups were added
currentmod <- data.frame(t = synth.reg.train$t)
incvars <- list()

# Stepwise!
# Take the first 50 most important variables (we can examine subsets later on)
for(k in 1:50) {
  likelihoods <- sapply(together, function(y) {
      mod <- cbind(currentmod, select(synth.reg.train, one_of(names(y))))
      model <- lm(t ~ ., data = mod)
      return(logLik(model))
  })
  # Choose the variable which will give the best log likelihood
  bestvar <- which(likelihoods == max(likelihoods))
  incvars[[k]] <- bestvar

  # Move the best variables from the list of vars under consideration to the dataframe of selected vari
  together <- together[setdiff(1:length(together), bestvar)]
  currentmod <- cbind(currentmod, select(synth.reg.train, one_of(names(bestvar))))
}
```

We now have a list of the top 50 groups of variables (59 variables in total) that impact on t. We will now create a series of linear models on the training data, test them on the testing data, and choose a model which provides a high R^2 on the test data.

```r
  # This bit exists due to the complication of needing to treat certain variables in groups
  # That is, highly correlated variables should go together
  # 'Steps' will tell us; first take the first three vars; then the next two vars; then
  # one by itself ; etc etc.
  steps <- cumsum(sapply(incvars, length))
  incvars2 <- unlist(incvars)
  collect.r2 <- rep(0, length(steps))
  num.outliers <- rep(0, length(steps))
  j <- 1

  for(i in steps) {
    currentmod <- select(synth.reg.train, t, one_of(names(incvars2)[1:i]))
    model <- lm(t ~ ., data = currentmod)
    assign(paste0('model', j), model)
    synth.reg.test$predvals <- predict(model, newdata = synth.reg.test)

    #We collect the R^2 of using the training model on the testing data.
    # Not sure if there's an automatic way to do this.
    # Also mark out the points with low likelihood
    synth.reg.test$residuals <- synth.reg.test$predvals - synth.reg.test$t

    ressumsquare <- sum((synth.reg.test$residuals)**2)
    totsumsquare <- sum((synth.reg.test$t - mean(synth.reg.test$t))**2)
    r.squared <- 1 - (ressumsquare/totsumsquare)
    collect.r2[j] <- r.squared

    # We create a cutoff - a 1.96 standard deviation confidence interval, using the standard deviations
    cutoff <- sd(model$residuals) * 1.96
    synth.reg.test$cutoff <- 0
```

```r
    synth.reg.test$cutoff[abs(synth.reg.test$residuals) > cutoff] <- 1
    num.outliers[j] <- sum(synth.reg.test$cutoff)
    assign(paste0('synth.reg.test', j), synth.reg.test)

    # Now make a graph showing predicted values vs actual values
    plottest <- ggplot(data = synth.reg.test, aes(x = predvals, y = t, colour = cutoff)) +
    geom_point() +
    geom_abline(intercept = 0, slope = 1) +
    geom_abline(intercept = cutoff, slope = 1) +
    geom_abline(intercept = -cutoff, slope = 1) +
    ggtitle(paste('first', i, 'vars, R2:', round(r.squared, 3), 'Observations above cutoff:', sum(synth
   assign(paste0('plottest', j), plottest)

   # And another graph showing the size of the residuals
  plottest.res <- ggplot(data = synth.reg.test, aes(y = abs(residuals), x = rownames(synth.reg.test), c
    geom_hline(yintercept = 1.96 * sd(model$residuals), colour = 'deeppink2') +
    ggtitle(paste('first', i, 'vars, R2:', round(r.squared, 3), 'Observations above cutoff:', sum(synth
    xlab('') +
    ylab('residual size')
  assign(paste0('plottest.res', j), plottest.res)

  j <- j + 1
  }
```

We now have a series of graphs which show us how our residuals change depending on the size of our models. We choose the model which gives the highest R^2 when used on the testing dataset. We find it is model 7.

```r
which(collect.r2 == max(collect.r2))
```

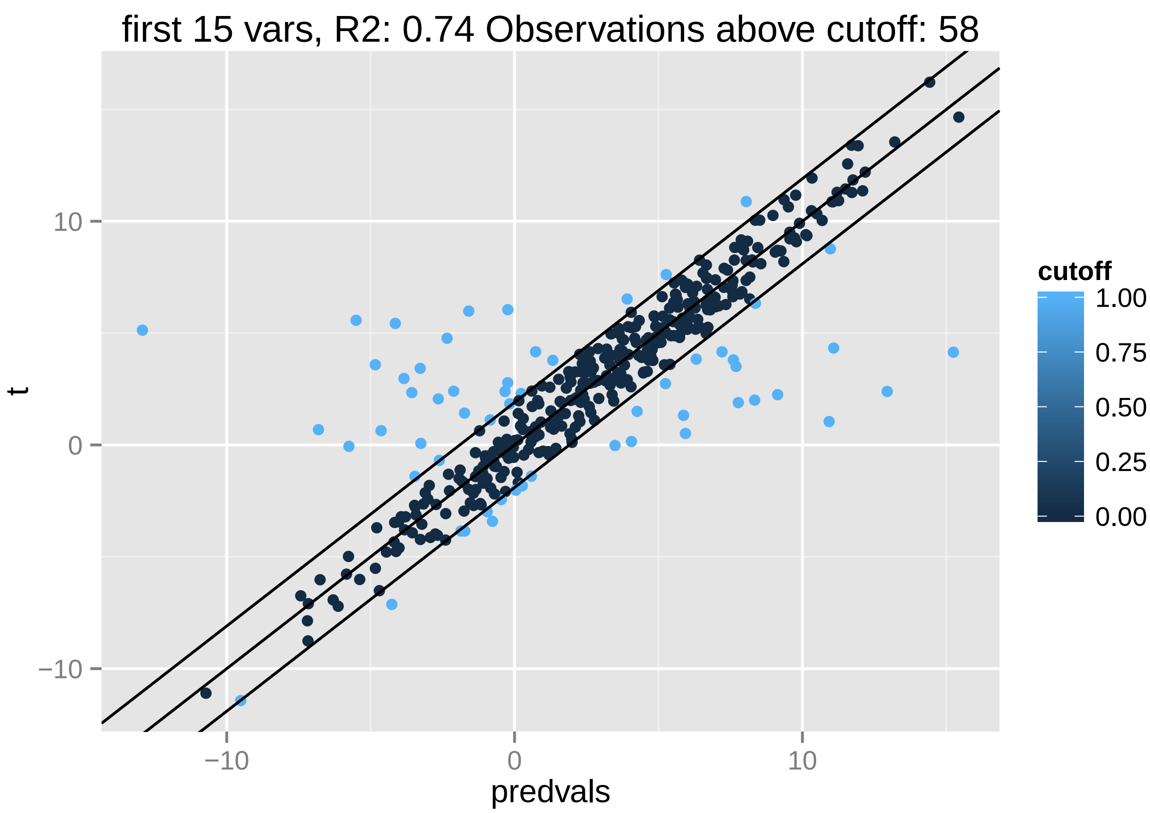
```
## [1] 7
```

```r
summary(model7)
```

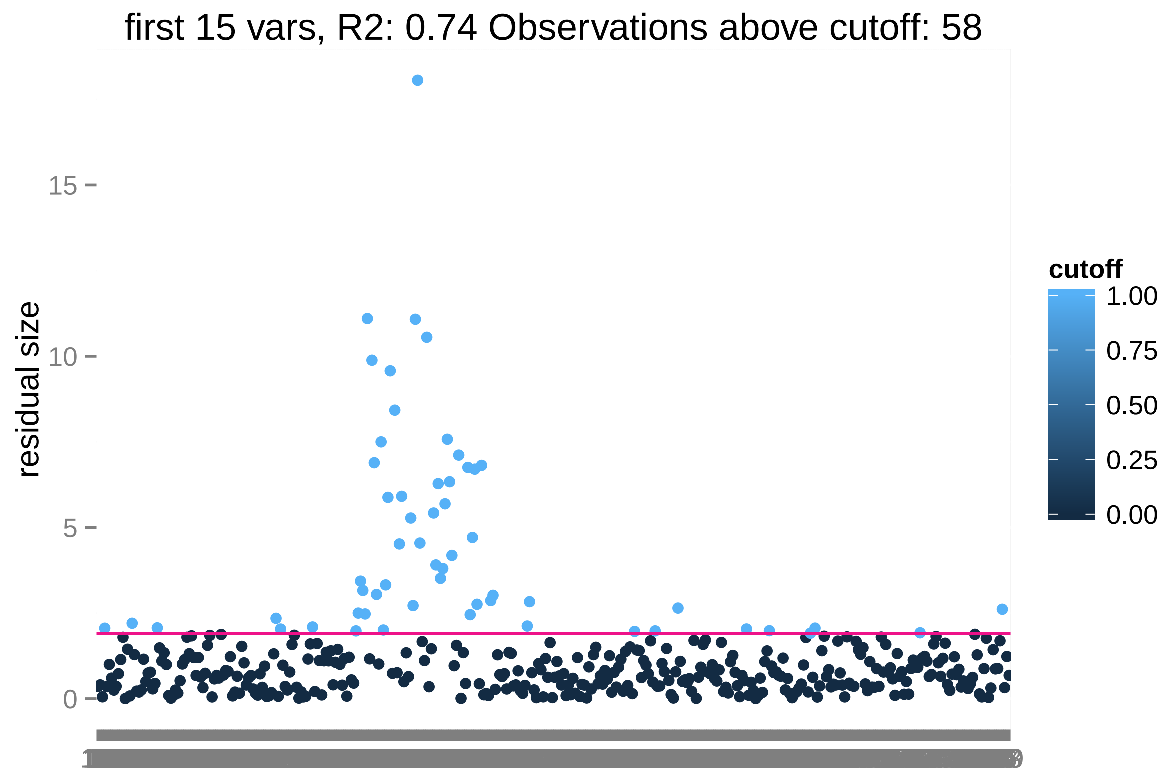
```
##
## Call:
## lm(formula = t ~ ., data = currentmod)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.0955 -0.6501 -0.0103  0.6579  3.4928
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.030107   0.019123 158.457  < 2e-16 ***
## X.1000       2.993732   0.019489 153.612  < 2e-16 ***
## X.5          2.035153   0.040357  50.429  < 2e-16 ***
## X.13        -0.054631   0.032010  -1.707 0.088002 .
## X.14         0.045564   0.031629   1.441 0.149817
## X.17        -0.035560   0.032495  -1.094 0.273912
## X.18        -0.003725   0.031949  -0.117 0.907197
## X.400        1.978385   0.041448  47.732  < 2e-16 ***
## X.15         0.007413   0.031439   0.236 0.813619
```

```
## X.16          0.037925   0.032087   1.182 0.237337
## X.100         0.979436   0.040354  24.271  < 2e-16 ***
## X.1           0.914478   0.040644  22.499  < 2e-16 ***
## X.11          0.043636   0.031538   1.384 0.166605
## X.12          0.054545   0.031117   1.753 0.079742 .
## X.588         0.065936   0.018965   3.477 0.000516 ***
## X.696        -0.053901   0.019022  -2.834 0.004639 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9739 on 2584 degrees of freedom
## Multiple R-squared:  0.9533, Adjusted R-squared:  0.953
## F-statistic:  3514 on 15 and 2584 DF,  p-value: < 2.2e-16
```

```
plottest7
```




```
plottest.res7
```



The last step is to create a list of which observations are above the cutoff value. As we see from the graphs above, there is not a clear point separation between outliers and non-outliers. I have therefore included in my list all the observations above the cutoff value, although at the margins the choice of whether to list a certain observation as an outlier or not is somewhat arbitrary. 'Index' captures the row numbers of these observations.

```
outliers <- mutate(synth.reg.test7, index = rownames(synth.reg.test7)) %>%
  filter(cutoff == 1) %>%
  arrange(residuals) %>%
  select(t, predvals, residuals, index)

outliers
```

```
##                t    predvals  residuals index
## 1    5.13038686 -12.9272312 -18.057618   225
## 2    5.57352935  -5.5059209 -11.079450   224
## 3    5.43193017  -4.1421851  -9.574115   214
## 4    3.58643325  -4.8384388  -8.424872   216
## 5    5.98378389  -1.5923733  -7.576157   237
## 6    0.68500482  -6.8143459  -7.499351   210
## 7    4.76906714  -2.3442211  -7.113288   241
## 8    2.97237251  -3.8412829  -6.813655   250
## 9    3.42484644  -3.2783785  -6.703225   248
## 10   6.04695065  -0.2328766  -6.279827   233
## 11   2.34123534  -3.5688562  -5.910092   219
## 12  -0.06240902  -5.7536238  -5.691215   236
## 13   0.63800906  -4.6373636  -5.275373   222
## 14   2.05923419  -2.6489700  -4.708204   247
## 15   2.40337699  -2.1155492  -4.518926   218
```

```
## 16    4.16469739   0.7310353  -3.433662   202
## 17    0.06785632  -3.2555304  -3.323387   212
## 18    1.42297647  -1.7377345  -3.160711   203
## 19    2.78259043  -0.2373898  -3.019980   255
## 20   10.88127644   8.0478079  -2.833469    27
## 21    2.38861024  -0.3308418  -2.719452   223
## 22    6.52431337   3.9118550  -2.612458    96
## 23    3.78334115   1.3289009  -2.454440   246
## 24    7.61622474   5.2683424  -2.347882    17
## 25    2.29778061   0.2285178  -2.069263   122
## 26   -1.40251745  -3.4618085  -2.059291   101
## 27    1.83774358  -0.1683987  -2.006142   211
## 28    1.11821124  -0.8469641  -1.965175   310
## 29   -0.68954119  -2.6150373  -1.925496    63
## 30  -11.42769535  -9.5142400   1.913455   380
## 31   -2.44093139  -0.4602417   1.980690   319
## 32   -1.39735820   0.5835125   1.980871   200
## 33   -3.84842106  -1.8621365   1.986285   364
## 34   -2.98347051  -0.9494068   2.034064   171
## 35    6.33162247   8.3676256   2.036003   355
## 36   -2.01978256   0.0416753   2.061458   382
## 37   -1.82629680   0.2685642   2.094861   184
## 38   -3.85149810  -1.7288872   2.122611   269
## 39    8.76679008  10.9710138   2.204224   112
## 40    3.83583840   6.3108818   2.475043   204
## 41    2.74229596   5.2428260   2.500530   201
## 42   -3.41430550  -0.7675908   2.646715   328
## 43    1.49979513   4.2577856   2.757990   249
## 44   -7.12906538  -4.2638940   2.865171   254
## 45    4.16193898   7.2066081   3.044669   209
## 46   -0.02311067   3.4902847   3.513395   234
## 47    3.80123003   7.6025581   3.801328   235
## 48    0.15339674   4.0600361   3.906639   232
## 49    3.50907818   7.6947921   4.185714   239
## 50    1.32443465   5.8695333   4.545099   226
## 51    0.51382339   5.9365025   5.422679   231
## 52    1.89304611   7.7726707   5.879625   213
## 53    2.00328029   8.3388120   6.335532   238
## 54    4.33193989  11.0857370   6.753797   245
## 55    2.24823310   9.1385439   6.890311   208
## 56    1.04502780  10.9280584   9.883031   207
## 57    2.39203999  12.9459801  10.553940   229
## 58    4.14485011  15.2448676  11.100018   205
```