Logisitic Regressions and regularization

Antoine Lizee

April 21, 2016

```
## Logisitcs Regressions
```

```
library(dplyr)
```

Prepare data

```
# load data ---------------------------------------------------------------

df <- ggplot2::diamonds %>%
  mutate(priceBinary = price > median(price)) %>%
  select(-(cut:clarity))
str(df)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    53940 obs. of  8 variables:
##  $ carat      : num  0.23 0.21 0.23 0.29 0.31 0.24 0.24 0.26 0.22 0.23 ...
##  $ depth      : num  61.5 59.8 56.9 62.4 63.3 62.8 62.3 61.9 65.1 59.4 ...
##  $ table      : num  55 61 65 58 58 57 57 55 61 61 ...
##  $ price      : int  326 326 327 334 335 336 336 337 337 338 ...
##  $ x          : num  3.95 3.89 4.05 4.2 4.34 3.94 3.95 4.07 3.87 4 ...
##  $ y          : num  3.98 3.84 4.07 4.23 4.35 3.96 3.98 4.11 3.78 4.05 ...
##  $ z          : num  2.43 2.31 2.31 2.63 2.75 2.48 2.47 2.53 2.49 2.39 ...
##  $ priceBinary: logi  FALSE FALSE FALSE FALSE FALSE FALSE ...
```

```
sds <- df %>% sapply(sd)
mus <- df %>% sapply(mean)

dfScale <- df %>% mutate_each(funs(scale), -price, -priceBinary)
```

glm from stats

```
# classic glms -------------------------------------------------------------

# Normal "full" logisitc regression
lr1 <- glm(priceBinary ~ depth + carat  + x + y + z, data = df, family = binomial)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
summary(lr1)
```

```
##
## Call:
## glm(formula = priceBinary ~ depth + carat + x + y + z, family = binomial,
##     data = df)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -8.4904  -0.0781  -0.0099   0.0559   3.2225
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -12.70543    1.55363  -8.178 2.89e-16 ***
## depth        -0.09366    0.01750  -5.351 8.73e-08 ***
## carat        14.93492    0.57250  26.087  < 2e-16 ***
## x            -1.81163    0.36602  -4.950 7.44e-07 ***
## y             3.17840    0.33712   9.428  < 2e-16 ***
## z             0.10385    0.07176   1.447    0.148
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 74777  on 53939  degrees of freedom
## Residual deviance: 12880  on 53934  degrees of freedom
## AIC: 12892
##
## Number of Fisher Scoring iterations: 9
```

```
# Natural coefficients
coeff_lr1 <- coef(lr1)
# Get the standardized transformative coefficients:
# Multiply the coefficients by the sds: (except the intercept)
std_coeff_lr1 <- coeff_lr1 * c(1, c(sds[names(coeff_lr1)][-1]))
# transform the intercept:
std_coeff_lr1[1] <- coeff_lr1[1] + sum(coeff_lr1[-1] * mus[names(coeff_lr1)][-1])

# logisitc regression on standardized data
std_lr1 <- glm(priceBinary ~ depth + carat  + x + y + z, data = dfScale, family = binomial)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
summary(std_lr1)
```

```
##
## Call:
## glm(formula = priceBinary ~ depth + carat + x + y + z, family = binomial,
##     data = dfScale)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -8.4904  -0.0781  -0.0099   0.0559   3.2225
##
## Coefficients:
```

```
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  1.63977    0.05320  30.824  < 2e-16 ***
## depth       -0.13418    0.02507  -5.351 8.73e-08 ***
## carat        7.07932    0.27137  26.087  < 2e-16 ***
## x           -2.03222    0.41059  -4.950 7.44e-07 ***
## y            3.63016    0.38504   9.428  < 2e-16 ***
## z            0.07329    0.05064   1.447    0.148
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 74777  on 53939  degrees of freedom
## Residual deviance: 12880  on 53934  degrees of freedom
## AIC: 12892
##
## Number of Fisher Scoring iterations: 9
```

```r
# Natural coefficients
coeff_std_lr1 <- coef(std_lr1)

# Compare -> they are the same !
print(cbind(std_coeff_lr1, coeff_std_lr1))
```

```
##             std_coeff_lr1 coeff_std_lr1
## (Intercept)    1.63976982    1.63976982
## depth         -0.13417684   -0.13417684
## carat          7.07932176    7.07932176
## x             -2.03221931   -2.03221931
## y              3.63016327    3.63016327
## z              0.07328749    0.07328749
```

```r
print(std_coeff_lr1 - coeff_std_lr1)
```

```
##   (Intercept)         depth         carat             x             y
##  8.526513e-14 -3.025358e-15  7.815970e-14 -7.549517e-14  2.886580e-14
##             z
## -1.026956e-15
```

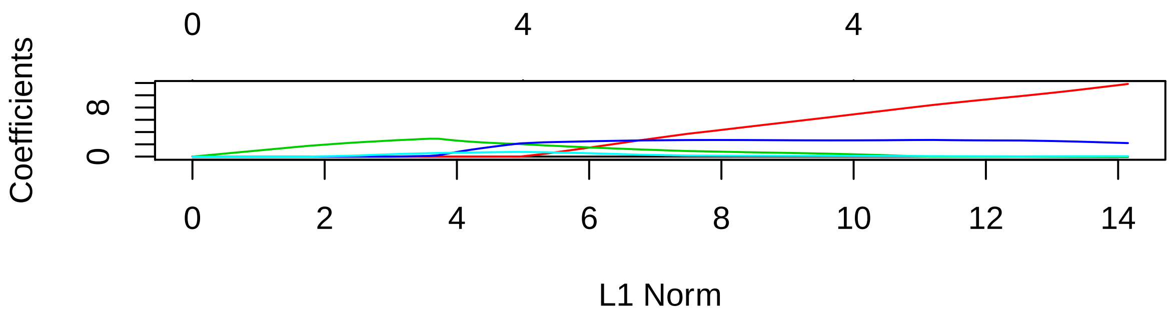
Using the glmnet package

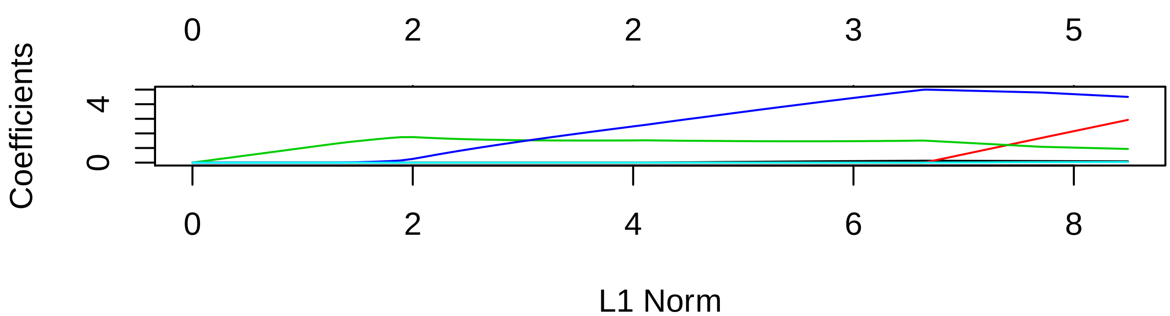
```r
# glmnet ------------------------------------------------------------------

## Compute all models **SLOW** (we directly compute the cross-validated optimums)
# default with standardization & coefficients reported in the feature units
cvlr2 <- glmnet::cv.glmnet(x = data.matrix(df[c("depth", "carat", "x", "y", "z")]),
                           y = df$priceBinary,
                           family = "binomial")
lr2 <- cvlr2$glmnet.fit
# no standardization
cvlr2_raw <- glmnet::cv.glmnet(x = data.matrix(df[c("depth", "carat", "x", "y", "z")]),
```

```r
                                   y = df$priceBinary,
                                   family = "binomial", standardize = FALSE)
lr2_raw <- cvlr2_raw$glmnet.fit
# inputting standardized data
std_cvlr2 <- glmnet::cv.glmnet(x = data.matrix(dfScale[c("depth", "carat", "x", "y", "z")]),
                              y = df$priceBinary,
                              family = "binomial")
std_lr2 <- std_cvlr2$glmnet.fit
```

We can see that the standaradized and non-standardized fits lead to completely different results.

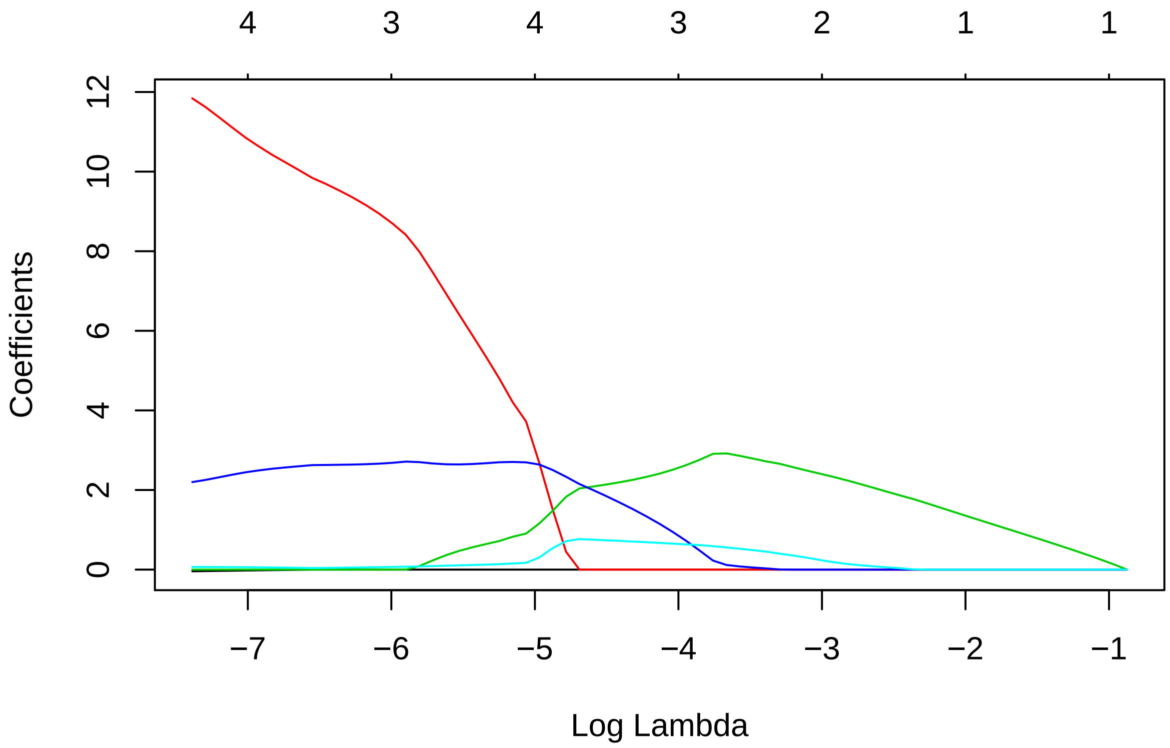
```r
layout(matrix(1:2))
plot(lr2)
plot(lr2_raw)
```



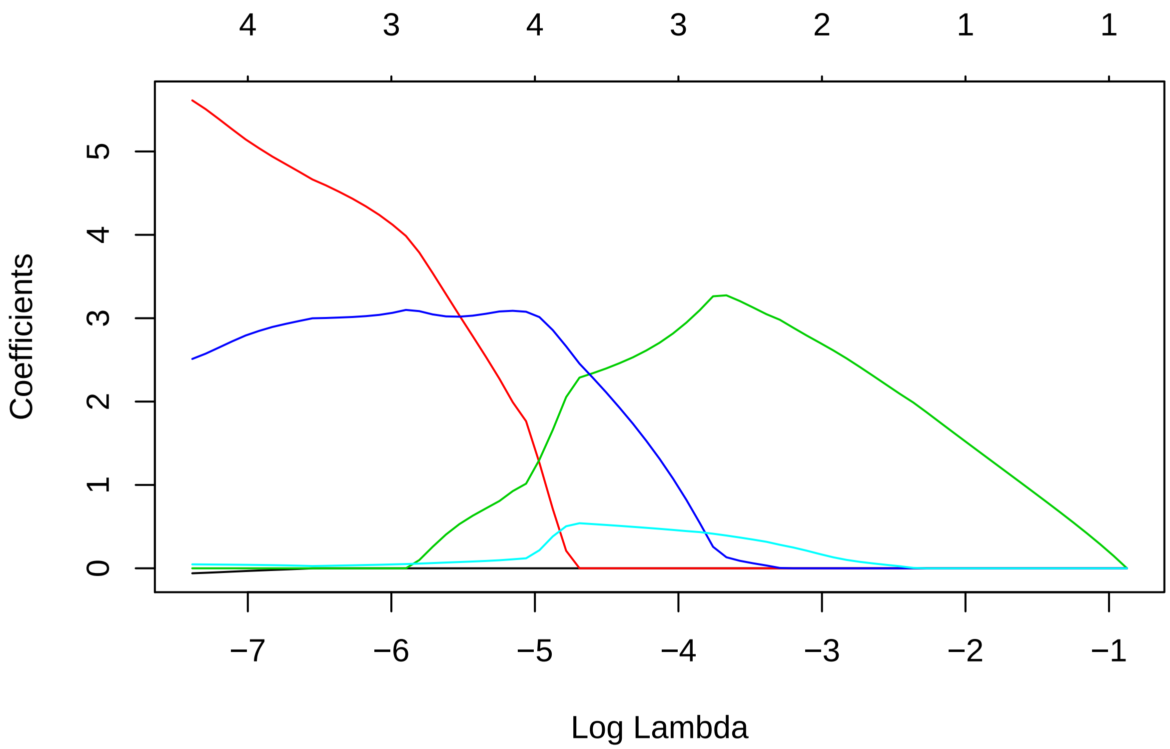


On the other hand, relying on the scaling from glmnet or scaling the features beforehand leads to the same model

```r
plot(lr2, xvar = "lambda")
```




```r
plot(std_lr2, xvar = "lambda")
```




```r
layout(1)

# RQ: looking in the "norm" space for the coefficient variation will
```

```r
# show different graphs because the L1 norm of the coeffs is also
# affected by their value. Try:
#plot(lr2)
#plot(std_lr2)
```

Of course, the optimal lambda will be only slightly different because of the randomness in the fold generation for the CVs:

```r
c(cvlr2$lambda.1se, std_cvlr2$lambda.1se)
```

```
## [1] 0.002743277 0.002743277
```

How different?

```r
coeff_lr2 <- coef(cvlr2, s = "lambda.1se")[,1]
coeff_std_lr2 <- coef(std_cvlr2, s = "lambda.1se")[,1]
# Same transformations as above:
std_coeff_lr2 <- coeff_lr2 * c(1, c(sds[names(coeff_lr2)][-1]))
std_coeff_lr2[1] <- coeff_lr2[1] + sum(coeff_lr2[-1] * mus[names(coeff_lr2)][-1])
cbind(coeff_std_lr2, std_coeff_lr2)
```

```
##             coeff_std_lr2 std_coeff_lr2
## (Intercept)    1.04466981    1.04466981
## depth          0.00000000    0.00000000
## carat          3.98669734    3.98669734
## x              0.00000000    0.00000000
## y              3.09931574    3.09931574
## z              0.05097746    0.05097746
```

Let's check that if we take the same lambda for the standard model than the one from the :

```r
cbind(coeff_std_lr2 = coef(std_cvlr2, s = cvlr2$lambda.1se)[,1],
      std_coeff_lr2)
```

```
##             coeff_std_lr2 std_coeff_lr2
## (Intercept)    1.04466981    1.04466981
## depth          0.00000000    0.00000000
## carat          3.98669734    3.98669734
## x              0.00000000    0.00000000
## y              3.09931574    3.09931574
## z              0.05097746    0.05097746
```