A Primer on stats::glm and glmnet packages for classification based on Logistic Regression | Discussion and Hands-On.

Shaurya Jauhari (Email: shauryajauhari@gzhmu.edu.cn)

2019-05-31

This is a R Markdown document on *glmnet* and *stats::glm* packages. These packages provide functionalities to cater to logistic regression problems, amongst others. Let's begin with *glmnet* first.

glmnet

```
install.packages("glmnet",
                 repos = "https://cran.us.r-project.org")
```

```

The downloaded binary packages are in
    /var/folders/hm/c3_fjypn62v5xh5b5ygv267m0000gn/T//RtmpuoOOMS/downloaded_packages
```

```
library(glmnet)
```

```
Loading required package: Matrix

Loading required package: foreach

Loaded glmnet 2.0-18
```

For practical purposes, we shall now load the revered **Iris Dataset** that was released by **Ronald Fisher** in his publication in **1936**. This data is prebundled in R and we just need to call it. Let's look at the structure of the dataset.

```
data("iris")
mydata <- iris
rm(iris)
str(mydata)
```

```
'data.frame':   150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```
table(mydata$Species)
```

```

    setosa versicolor  virginica 
        50         50         50 
```

```
head(mydata)
```

```
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
```

```
1          5.1         3.5         1.4         0.2  setosa
2          4.9         3.0         1.4         0.2  setosa
3          4.7         3.2         1.3         0.2  setosa
4          4.6         3.1         1.5         0.2  setosa
5          5.0         3.6         1.4         0.2  setosa
6          5.4         3.9         1.7         0.4  setosa
```

You can always seek help in the R documentation with ?.

A good thing here is that the dataset is already clean in defining the response variable. The class labels are factors that are represented intutively. There could be situations where the class labels are naively defined in a dataset as numbers, or not defined at all. The homework that needs to be done then is convert them into factors and then define string labels that are more relatable. For example, let's assume that the class labels were represented by numbers 0,1,2. Then we would've got to do the following processing or something equivalent:

$irisSpecies < -factor(irisSpecies, levels = c(“0”,“1”,“2”), labels = c(“setosa”,“versicolor”,“virginica”))$

You would also see that there are no missing values in the dataset, so it's tidy and doesn't warrant any imputation strategy.

So far so good. Really? Here's a problem. We are exploring classification via logistic regression, which means that we can have only **two class labels** (binomial family) as per the definition of logistic regression model. Now, we could've considered another dataset, but I like the flower. Let's do something about it. For now, we'll create an arbitrary feature/column- “Class”, that'll hold the binary outcome. We'll randomly define 0: Leaf and 1:Flower. Let's see what happens when we do that.

```
mydata$Class <- as.factor(sample(c(0,1)))
str(mydata$Class)
```

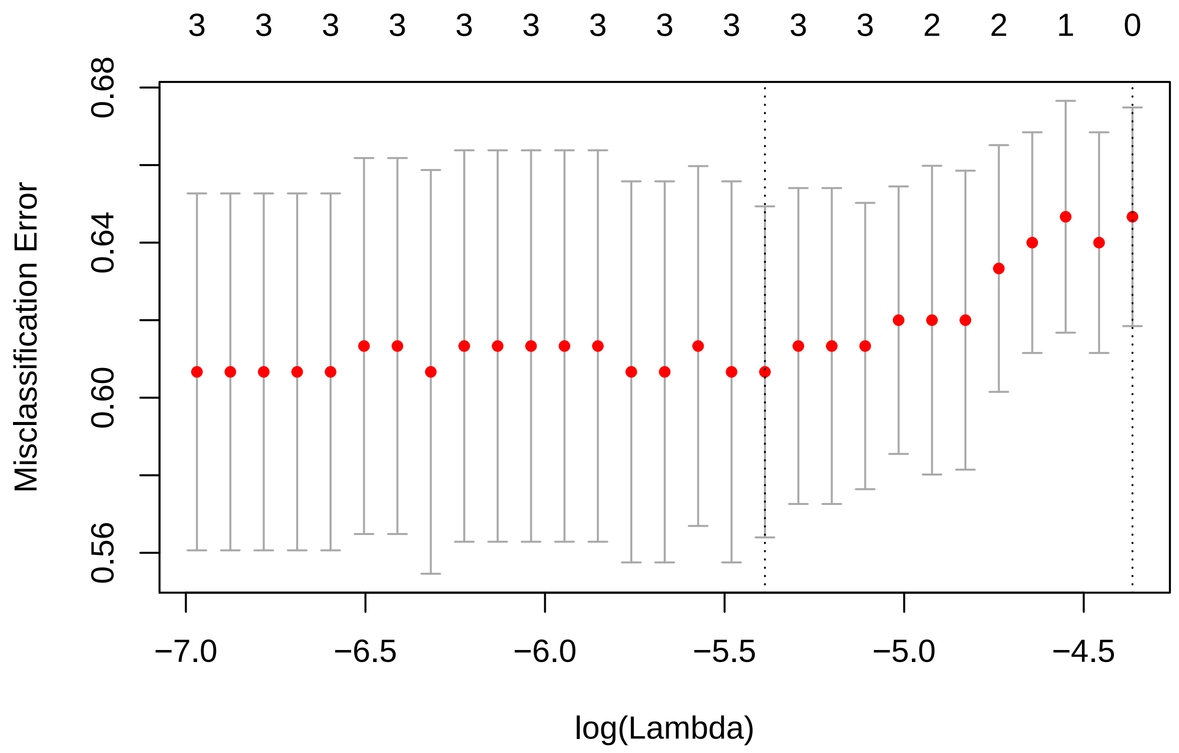
```
 Factor w/ 2 levels "0","1": 1 2 1 2 1 2 1 2 1 2 ...
```

```
# Let's label them for more clarity.

mydata$Class <- factor(mydata$Class,
                       levels = c("0","1"),
                       labels = c("Leaf", "Flower"))
mydata <- mydata[,-5]

# Cross-validation for finding the optimal lambda.
set.seed(1)
cv.modelfit <- cv.glmnet(as.matrix(mydata[,1:4]),
                         mydata$Class,
                         family = "binomial",
                         type.measure = "class",
                         alpha = 1,
                         nlambda = 100)
```

```
plot(cv.modelfit)
```



```r
cat("There are", length(cv.modelfit$lambda),
    "lambda values in all:",
    cv.modelfit$lambda,
    ", out of which",
    cv.modelfit$lambda.min,
    "is the minimum, while",
    cv.modelfit$lambda.1se,
    "denotes the value at which the model is optimized at one standard error.")
```

```
There are 29 lambda values in all: 0.01272444 0.01159404 0.01056406 0.009625576 0.008770466 0.007991322
```

The plot shows the models (with varying lambda values) that *glmnet* has fit, alongwith the misclassification error associated with each model.The first dotted line highlights the minimum misclassification error, while the second one is the highly regularized model within *1se* (one standard error).

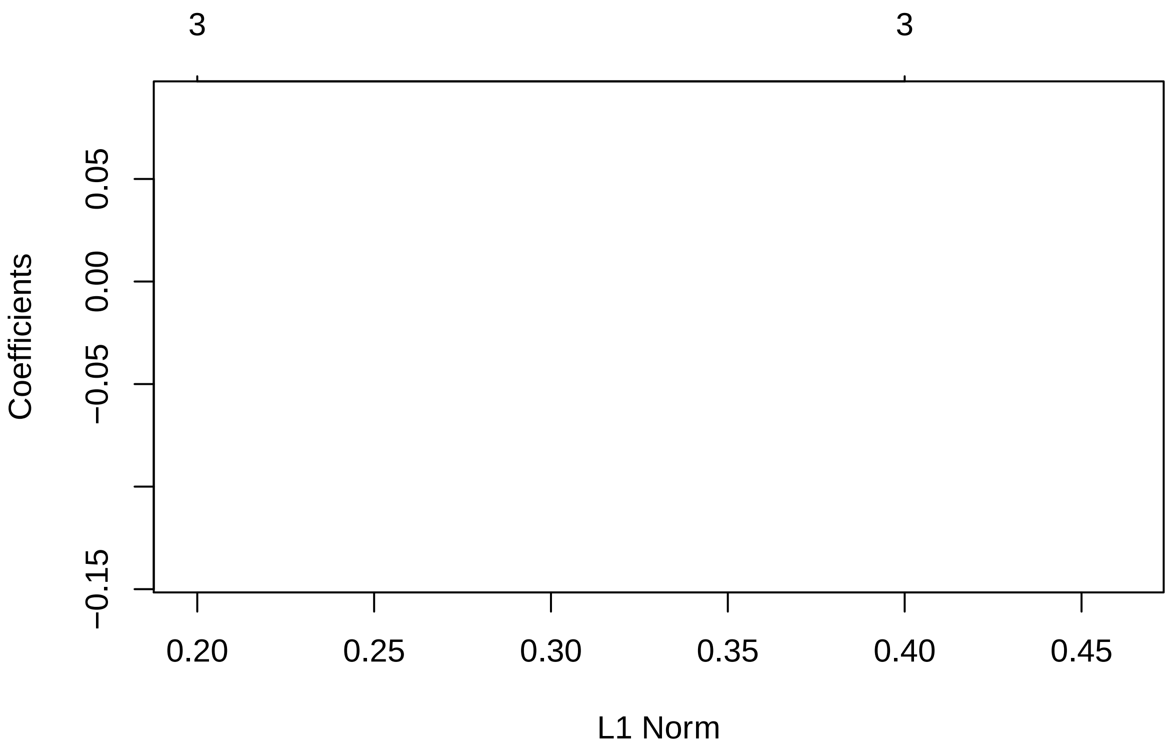
```r
set.seed(2)
modelfit <- glmnet(as.matrix(mydata[,1:4]),
                   mydata$Class,
                   family = "binomial",
                   alpha = 1,
                   lambda = cv.modelfit$lambda.min)

# Listing non-zero coefficients

print(modelfit$beta[,1])
```

```
Sepal.Length  Sepal.Width Petal.Length  Petal.Width
  0.08836100  -0.09974821   0.00000000  -0.14235687
```

```r
plot(modelfit)
```



Note that the **features must be presented as a data matrix**, while the **response variable is a factor with two levels**. On calling the *glmnet*, we get information under 3 heads: *Df* signifies the number of non-zero coefficients from left to right, i.e. in this case coefficients for Sepal.Length, Sepal.Width, Petal.Length, Petal.Width; *%Dev* represents deviation; and *Lambda* represents the penalties imposed by the model. They would typically be limited to 100, but could even halt early if insufficient deviation is observed. Also, by default elastic-net (lasso+ridge) is used for regularization task by the glmnet, which could be set to lasso (alpha=1) or ridge (alpha=0).

```r
coef(modelfit)[,1]
```

```
 (Intercept) Sepal.Length  Sepal.Width Petal.Length  Petal.Width 
 -0.04063797   0.08836100  -0.09974821   0.00000000  -0.14235687 
```

```r
predict(modelfit, type="coef")
```

```
5 x 1 sparse Matrix of class "dgCMatrix"
                      s0
(Intercept)  -0.04063797
Sepal.Length  0.08836100
Sepal.Width  -0.09974821
Petal.Length  .         
Petal.Width  -0.14235687
```

"." here symbolizes 0.

Excercises.

1. Try to fit the model with varying lamba, say *cv.modelfit$lambda.min*.

2. Try the above for alpha = 0, i.e. ridge penalty.
3. Try the above for any value between 0 and 1; that's **elastic-net** regularization.
4. Try the above template for several available datasets at http://archive.ics.uci.edu/ml/index.php.

Now, let's move to stats::glm.

stats::glm()

The *stats* package is preloaded in R. We are particularly interested in the generalised linear models , glm() function. To begin, we shall customarily bifurcate our dataset into training data and testing data. The training data shall be used to build our linear model, while the testing data shall be used for its validation. Arbitrary proportions can be considered for splitting the data, however, usually 80-20 partition is reasonable.

```r
set.seed(123) # for results reproducibility.
part <- sample(2, nrow(mydata),
               replace = TRUE,
               prob = c(0.8,0.2))
train <- mydata[part==1,]
test <- mydata[part==2,]
cat("So, now we have",
    nrow(train),
    "training rows and",
    nrow(test),
    "testing rows")
```

```
So, now we have 121 training rows and 29 testing rows
```

```r
mymodel <- glm(formula = Class ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width,
               data = train,
               family = "binomial")
summary(mymodel)
```

```

Call:
glm(formula = Class ~ Sepal.Length + Sepal.Width + Petal.Length +
    Petal.Width, family = "binomial", data = train)

Deviance Residuals:
   Min       1Q   Median       3Q      Max
-1.338  -1.164  -1.035   1.176   1.337

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)    1.6751     2.2360   0.749    0.454
Sepal.Length  -0.1058     0.5880  -0.180    0.857
Sepal.Width   -0.3209     0.6225  -0.516    0.606
Petal.Length  -0.1228     0.5782  -0.212    0.832
Petal.Width    0.2965     0.9697   0.306    0.760

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 167.73  on 120  degrees of freedom
Residual deviance: 167.07  on 116  degrees of freedom
AIC: 177.07
```

```
Number of Fisher Scoring iterations: 3
```

Here, we are taking into account all the variables as responses to the predictor variable - *Class*. Although, it can be interpreted straightforwardly, that none of the estimated coefficients of the model are statistically significant (See Pr (>|z|)); but that's just the nature of this data, and in general terms it's better to reject all variables that have insignificant coefficients. Had we chosen to do that here, we would've left with nothing. Never mind. This demonstration is to highlight the protocol of logistic regression. Let's continue with whatever we have here, taking all.

Nonetheless, we musn;t ignore an important aspect of *multicollinearity*. Out of many ways to access that, rms::vif() provides an effective way to seek multicollinearity problem. vif stands for Variance Inflation Factor, and by norm if vif() > 10, we must omit the corresponding column (variable) as it does not add much to the model due to redundancy.

```r
library(rms)
```

```
Loading required package: Hmisc

Loading required package: lattice

Loading required package: survival

Loading required package: Formula

Loading required package: ggplot2


Attaching package: 'Hmisc'

The following objects are masked from 'package:base':

    format.pval, units

Loading required package: SparseM


Attaching package: 'SparseM'

The following object is masked from 'package:base':

    backsolve
```

```r
vif(mymodel)
```

```
Sepal.Length  Sepal.Width Petal.Length  Petal.Width
    6.765915     2.092696    30.104356    16.024227
```

Without getting into the mathematics of it, we see here that *Petal.Length* has a huge vif score and so does *Petal.Width*. That suggests us to remove *Petal.Length*. Let's reevaluate the model after doing it.

```r
mymodel <- glm(formula = Class ~ Sepal.Length + Sepal.Width + Petal.Width,
               data = train,
               family = "binomial")
summary(mymodel)
```

```

Call:
glm(formula = Class ~ Sepal.Length + Sepal.Width + Petal.Width,
    family = "binomial", data = train)

Deviance Residuals:
   Min      1Q  Median      3Q     Max
-1.334  -1.168  -1.027   1.184   1.331

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)    1.6747     2.2349   0.749    0.454
Sepal.Length  -0.1911     0.4298  -0.445    0.657
Sepal.Width   -0.2386     0.4866  -0.490    0.624
Petal.Width    0.1190     0.4916   0.242    0.809

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 167.73  on 120  degrees of freedom
Residual deviance: 167.12  on 117  degrees of freedom
AIC: 175.12

Number of Fisher Scoring iterations: 3
```

```
vif(mymodel)
```

```
Sepal.Length  Sepal.Width  Petal.Width
    3.617677     1.279805     4.119004
```

It looks more promising now and we can safely include these three variables in our final model.

```
y_train <- predict(mymodel,
                   train,
                   type = "response")
head(y_train)
```

```
        1         2         3         6         7         9
0.4722284 0.5115800 0.5092034 0.4402444 0.5050253 0.5413376
```

```
head(train)
```

```
  Sepal.Length Sepal.Width Petal.Length Petal.Width  Class
1          5.1         3.5          1.4         0.2   Leaf
2          4.9         3.0          1.4         0.2 Flower
3          4.7         3.2          1.3         0.2   Leaf
6          5.4         3.9          1.7         0.4 Flower
7          4.6         3.4          1.4         0.3   Leaf
9          4.4         2.9          1.4         0.2   Leaf
```

These are the estimates of the class variable. To calculate the accuracy of the model we need to compare these to the orginal values of the response variable, 0 for "Leaf" and 1 for "Flower". If you see the first observation, 0.4722284 ~ 47.2 % chance of being a flower, and in actuality if you look at the original data frame it is not a flower, i.e. a leaf. The probability (47.2%) can be calculated by fitting values of coefficients in the model. Try doing that.

y = 1.6747 + (-0.1911 x Sepal.Length) + (-0.2386 x Sepal.Width) + (0.1190 x Petal.Width) = 1.6747 + (-0.1911 x 5.1) + (-0.2386 x 3.5) + (0.1190 x 0.2) = 1.6747 + (-0.97461) + (-0.8351) + 0.0238 = -0.11121 != 0.4722284 CROSS-CHECK

```
prediction_probabilities_train <- ifelse(y_train > 0.5, 1, 0) # Probabilities to Labels conversion
confusion_matrix_train <- table(Predicted = prediction_probabilities_train, Actual = train$Class)
print(confusion_matrix_train)
```

```
         Actual
```

```
Predicted Leaf Flower
        0   33     32
        1   28     28
```

```
misclassfication_error_train <- 1- sum(diag(confusion_matrix_train))/sum(confusion_matrix_train)
cat("The misclassification error in train data is",
    (round(misclassfication_error_train*100)), "percent")
```

```
The misclassification error in train data is 50 percent
```

Now, we can repeat the same procedure for the test data.

```
y_test <- predict(mymodel, test, type = "response")
prediction_probabilities_test <- ifelse(y_test > 0.5, 1, 0)
confusion_matrix_test <- table(Predicted = prediction_probabilities_test, Actual = test$Class)
print(confusion_matrix_test)
```

```
         Actual
Predicted Leaf Flower
        0    8     11
        1    6      4
```

```
misclassfication_error_test <- 1- sum(diag(confusion_matrix_test))/sum(confusion_matrix_test)
cat("The misclassification error in test data is",
    (round(misclassfication_error_test*100)), "percent")
```

```
The misclassification error in test data is 59 percent
```

Finally, there is also a way to ascertain if our model on the whole is statistically significant. We refer this as the Goodness-Of-Fit test.

```
overall_p <- with(mymodel,
                  pchisq(null.deviance-deviance,
                         df.null-df.residual,
                         lower.tail = FALSE))
cat("The confidence level for this model is",
    ((1-overall_p)*100), "percent")
```

```
The confidence level for this model is 10.68387 percent
```

Our model achieved a fairly large p-value and a low confidence level suggesting that the model is unsuitable for current classification task.