Application of Logistic Regression via packages stats::glm and glmnet for Enhancer Prediction data.

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

2019-10-14

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//RtmpZeTOgN/downloaded_packages
```

```
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loading required package: foreach
```

```
## Loaded glmnet 2.0-18
```

The dataset derivation has been lengthily described here

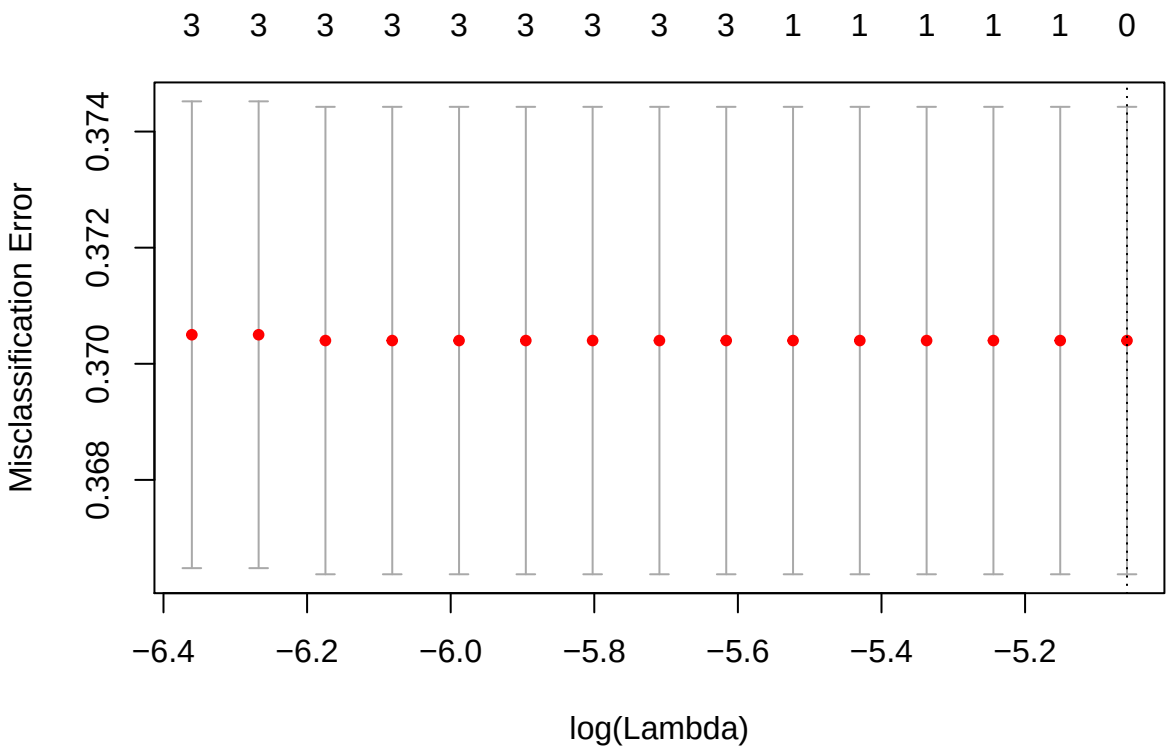
```
epdata <- readRDS("../../Machine_Learning_Deep_Learning/data/ep_data_sample.rds")
```

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

The class labels 0 and 1 represent "Enhancer" and "Non-Enhancer" categories.

```
set.seed(1)
cv.modelfit <- cv.glmnet(as.matrix(epdata[,1:4]),
                         epdata$class,
                         family = "binomial",
                         type.measure = "class",
                         alpha = 1,
                         nlambda = 100)
```

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




```
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 15 lambda values in all: 0.006358038 0.005793207 0.005278555 0.004809622 0.004382349 0.003
```

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).

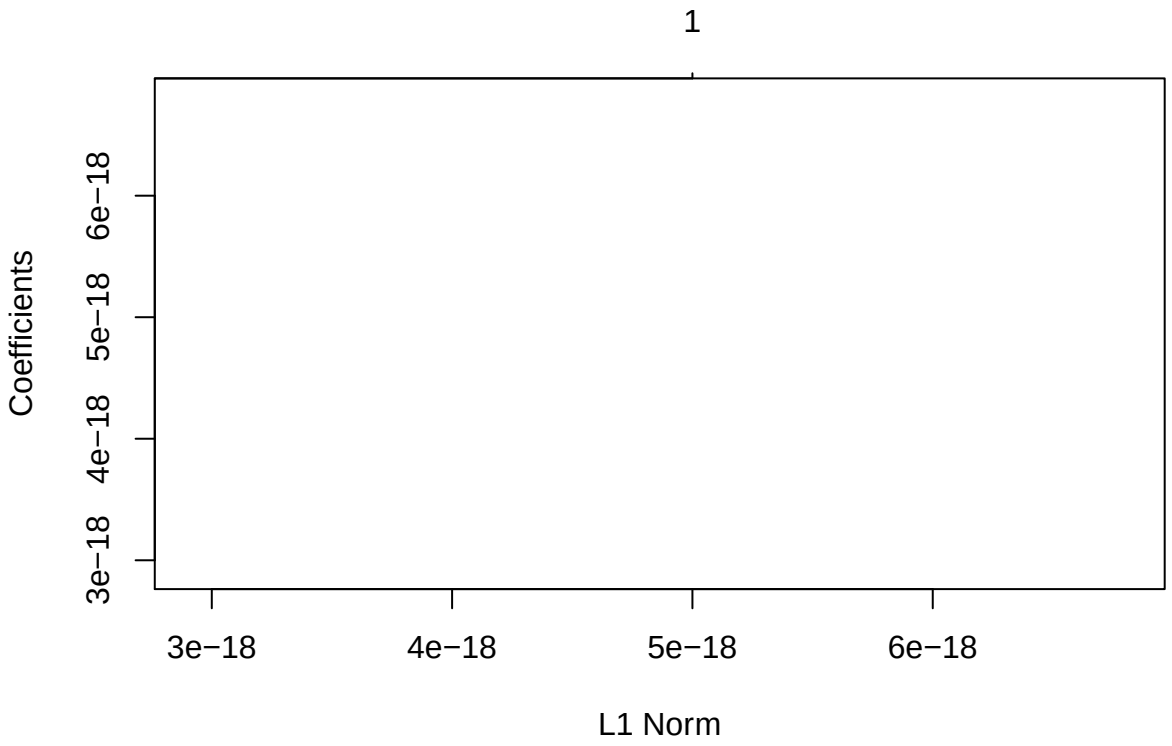
```
set.seed(2)
modelfit <- glmnet(as.matrix(epdata[,1:4]),
                   epdata$class,
                   family = "binomial",
                   alpha = 1,
                   lambda = cv.modelfit$lambda.min)

# Listing non-zero coefficients

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

```
## peaks_h3k27ac peaks_h3k4me3 peaks_h3k4me2 peaks_h3k4me1
##  0.000000e+00  0.000000e+00  4.863933e-18  0.000000e+00
```

```
plot(modelfit)
```

```
## Warning in plotCoef(x$beta, lambda = x$lambda, df = x$df, dev =
```

```
## x$dev.ratio, : 1 or less nonzero coefficients; glmnet plot is not
## meaningful
```



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).

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

```
##   (Intercept) peaks_h3k27ac peaks_h3k4me3 peaks_h3k4me2 peaks_h3k4me1
##  5.305012e-01  0.000000e+00  0.000000e+00  4.863933e-18  0.000000e+00
```

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

```
## 5 x 1 sparse Matrix of class "dgCMatrix"
##                          s0
## (Intercept)   5.305012e-01
## peaks_h3k27ac .
## peaks_h3k4me3 .
## peaks_h3k4me2 4.863933e-18
## peaks_h3k4me1 .
```

"." 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.

```
set.seed(3) # for results reproducibility.
part <- sample(2, nrow(epdata),
               replace = TRUE,
               prob = c(0.8,0.2))
train <- epdata[part==1,]
test <- epdata[part==2,]
cat("So, now we have",
    nrow(train),
    "training rows and",
    nrow(test),
    "testing rows")
```

```
## So, now we have 8016 training rows and 1984 testing rows
```

```
epmodel <- glm(formula = class ~ peaks_h3k27ac + peaks_h3k4me3 + peaks_h3k4me2 + peaks_h3k4me1,
               data = train,
               family = "binomial")
summary(epmodel)
```

```
##
## Call:
## glm(formula = class ~ peaks_h3k27ac + peaks_h3k4me3 + peaks_h3k4me2 +
##     peaks_h3k4me1, family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.8280  -1.4100   0.9574   0.9614   1.2157
##
## Coefficients:
##                 Estimate Std. Error z value Pr(>|z|)
## (Intercept)    0.5319361  0.0260284  20.437   <2e-16 ***
## peaks_h3k27ac -0.0217276  0.0201602  -1.078    0.281
## peaks_h3k4me3  0.0048529  0.0174500   0.278    0.781
## peaks_h3k4me2  0.0200793  0.0154109   1.303    0.193
## peaks_h3k4me1 -0.0006598  0.0079202  -0.083    0.934
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 10558  on 8015  degrees of freedom
## Residual deviance: 10555  on 8011  degrees of freedom
## AIC: 10565
##
## Number of Fisher Scoring iterations: 4
```

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.

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

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

```
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
```

```
vif(epmodel)
```

```
## peaks_h3k27ac peaks_h3k4me3 peaks_h3k4me2 peaks_h3k4me1
##      1.351508      1.236176      2.122616      2.003160
```

Neither of the variables have high vif score, so they all qualify for the model.

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

```
##  11034519   2449475  16635621  10889006  16312109   3709751
## 0.6262084 0.6278333 0.6366687 0.6312187 0.6260835 0.6299346
```

```
head(train)
```

```
##          peaks_h3k27ac peaks_h3k4me3 peaks_h3k4me2 peaks_h3k4me1 class
## 11034519      0.734144       0.00000      0.000000      0.000000     1
## 2449475       1.468290       1.13595      0.865892      0.000000     1
## 16635621      0.734144       5.67974      0.865892      0.000000     0
## 10889006      0.000000       1.13595      0.000000      0.000000     1
## 16312109      0.734144       0.00000      0.000000      0.809057     1
## 3709751       0.000000       0.00000      0.000000      0.000000     1
```

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 "Enhancer" and 1 for "Non-Enhancer". If you see the first observation, 0.6262084 ~ 62.6 % chance of being a non-enhancer, and in actuality if you look at the original data frame it is a non-enhancer. The probability (62.6 %) can be calculated by fitting values of coefficients in the model. Try doing that.

y = 0.5319361 + (-0.0217276 x peaks_h3k27ac) + (0.0048529 x peaks_h3k4me3) + (0.0200793 x peaks_h3k4me2) + (-0.0006598 x peaks_h3k4me3) = 0.5319361 + (-0.0217276 x 0.734144) + (0.0048529 x 0.00000) + (0.0200793 x 0.00000) + (-0.0006598 x 0.00000) = 0.5319361 + (-0.0159511872) + 0 + 0 + 0 = 0.515984913 != 0.6262084 CROSS-CHECK

There is some discrepancy in resultant probabilities, yet the classification is accurate.

```
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    0    1
##         0    0    1
##         1 2960 5055
```

```
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 37 percent
```

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

```
y_test <- predict(epmodel, 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    0    1
##         1  744 1240
```

```
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 62 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(epmodel,
                  pchisq(null.deviance-deviance,
```

```
                          df.null-df.residual,
                          lower.tail = FALSE))
cat("The statistical significance for the model is", overall_p, "\n")
```

```
## The statistical significance for the model is 0.5030704
```

```
cat("The confidence level for this model is",
    ((1-overall_p)*100), "percent")
```

```
## The confidence level for this model is 49.69296 percent
```

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