logistic Notebook

Code

Logistic regression is useful when we are predicting a binary outcome from a set of continuous predictor variables.

Hide

```
library(caTools)
library(ROCR)
```

```
package <U+393C><U+3E31>ROCR<U+393C><U+3E32> was built under R version 3.5.3Loading r
equired package: gplots
package <U+393C><U+3E31>gplots<U+393C><U+3E32> was built under R version 3.5.3
Attaching package: <U+393C><U+3E31>gplots<U+393C><U+3E32>

The following object is masked from <U+393C><U+3E31>package:stats<U+393C><U+3E32>:

    lowess
```

Hide

```
library(dplyr)
```

```


Attaching package: <U+393C><U+3E31>dplyr<U+393C><U+3E32>

The following objects are masked from <U+393C><U+3E31>package:stats<U+393C><U+3E32>:

    filter, lag

The following objects are masked from <U+393C><U+3E31>package:base<U+393C><U+3E32>:

    intersect, setdiff, setequal, union
```

caTools library is used for spliting the dataset

Hide

```
setwd("C:/Users/Veeru/Desktop/ML/logistic")
getwd()
```

```
[1] "C:/Users/Veeru/Desktop/ML/logistic"
```

Hide

```
data1 <- read.csv("LogisticData_1.csv",header=TRUE, sep = ",")
str(data1)
```

```
'data.frame':   1000 obs. of  3 variables:
 $ X1: num  1.141 1.408 -0.92 0.305 -1.667 ...
 $ X2: num  -0.202 -0.316 -1.697 -0.603 0.273 ...
 $ Y : int  1 1 1 1 1 1 1 1 1 1 ...
```

In the dataset, X1 and X2 are the independant variables and Y is dependant as well as categorical variable.

Hide

```
sum(is.na(data1))
```

```
[1] 0
```

checking for any NanN's

Hide

```
split <-  sample.split(data1, SplitRatio = 0.8)
split
```

```
[1] FALSE  TRUE  TRUE
```

80% of dataset is used as training and remaining 20% for testing.

Hide

```
training <-  subset(data1, split == "TRUE")
testing <-  subset(data1, split == "FALSE")
```

Hide

```
print(str(training))
```

```
'data.frame':   666 obs. of  3 variables:
 $ X1: num  1.408 -0.92 -1.667 -2.13 -0.232 ...
 $ X2: num  -0.316 -1.697 0.273 0.379 -2.009 ...
 $ Y : int  1 1 1 1 1 1 1 1 1 1 ...
NULL
```

Hide

```
print(str(testing))
```

```
'data.frame':   334 obs. of  3 variables:
 $ X1: num  1.141 0.305 1.848 -0.378 -0.323 ...
 $ X2: num  -0.202 -0.603 1.181 -0.834 0.569 ...
 $ Y : int  1 1 1 1 1 1 1 1 1 1 ...
NULL
```

Hide

```
model <-  glm(Y~. ,training, family="binomial")
```

Y is our response variable and X1 and X2 are the predictor variables.

In Generalized linear models(glm()), we are using family link as binomial for logit

Hide

```
summary(model)
```

```


Call:
glm(formula = Y ~ ., family = "binomial", data = training)

Deviance Residuals: 
    Min       1Q   Median       3Q      Max  
-3.0664   0.1548   0.2515   0.3927   1.3635  

Coefficients:
            Estimate Std. Error z value Pr(>|z|)    
(Intercept)  3.35037    0.24951  13.428  < 2e-16 ***
X1           0.07611    0.16431   0.463    0.643    
X2           1.31502    0.17829   7.376 1.63e-13 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 360.08  on 665  degrees of freedom
Residual deviance: 289.54  on 663  degrees of freedom
AIC: 295.54

Number of Fisher Scoring iterations: 6
```

The coefficients above of the model are the intercept i.e, B0 , weight1 corresponding to variable X1 i.e, B1 and weight2 corresponding to variable X2 i.e, B2

Hide

```
res <-  predict(model, testing, type="response")
str(res)
```

```
 Named num [1:334] 0.96 0.93 0.994 0.902 0.983 ...
 - attr(*, "names")= chr [1:334] "1" "4" "7" "10" ...
```

The response in the predict() function gives numerical results.

The model we created predicts the ‘Y’ variable from testing data set using X1 and X2 variables. we can check its accuracy by confusion matrix

Hide

```
conf_matrix_0.5 <- table(ActualValue=testing$Y, PredictedValue=res>0.5)
conf_matrix_0.5
```

```
           PredictedValue
ActualValue FALSE TRUE
          0     2   19
          1     0  313
```

In the above confusion matrix code, threshold is set to 50% for predicted values, the numbers below 0.5 is assigned 0 and number equal or greater are assigned to 1. Out of 334 ‘Y’ observations, model has predicted 2 of the 0’s correctly but 19 of the rest 0’s are predicted as 1’s. The model successfully has predicted all 1’s of ’Y’accurately.

Hide

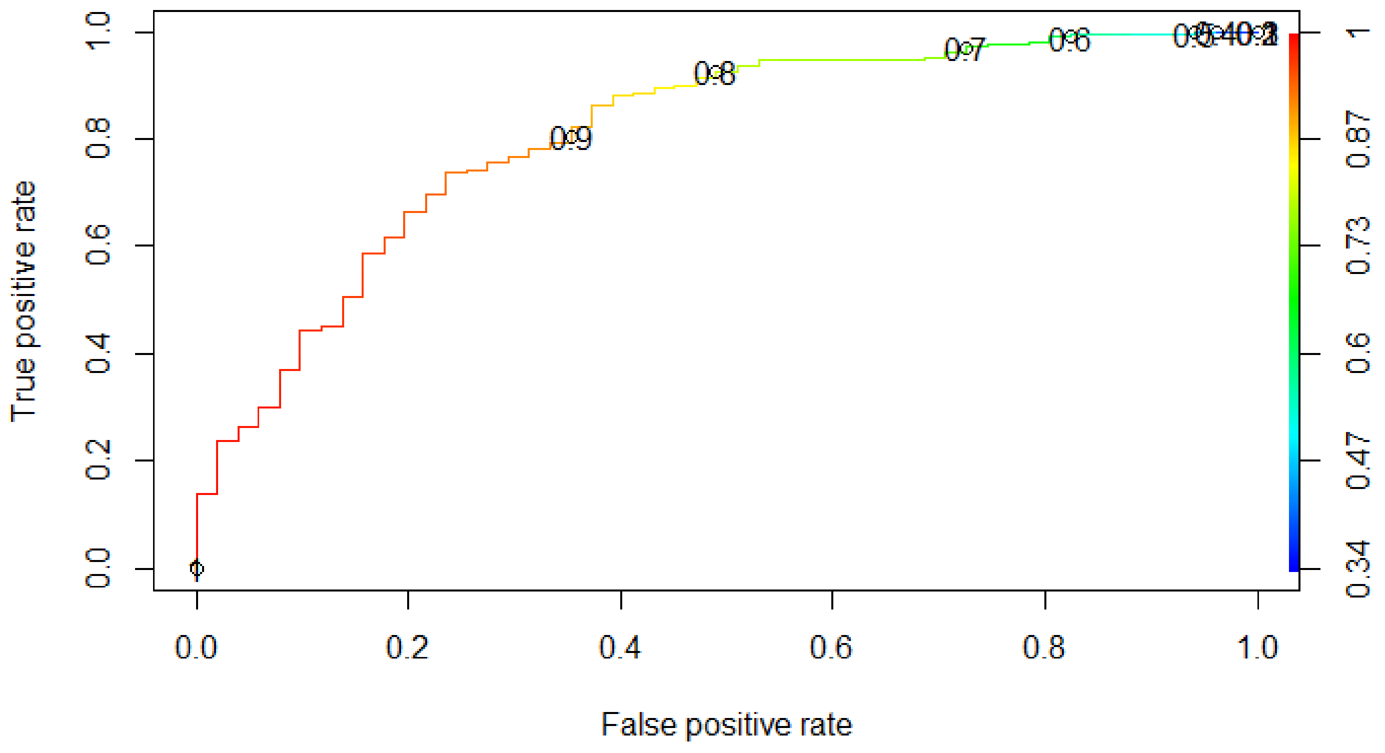
```
accuracy_0.5 <- conf_matrix_0.5
accuracy_0.5 <- (accuracy_0.5[1]+accuracy_0.5[4])/sum(accuracy_0.5)
accuracy_0.5*100
```

```
[1] 94.31138
```

Accuracy of the model is 94% which is a good fit. We can further find the optimal threshold for which the accuracy may increase by plotting ROC curve

Hide

```
res <-  predict(model, training, type="response")
ROCRPred <- prediction(res, training$Y)
ROCRPref <- performance(ROCRPred, "tpr", "fpr")
plot(ROCRPref, colorize=TRUE, print.cutoffs.at=seq(0.1,by=0.1))
```



To find the optimum threshold we should choose the value which has less false positive rate and more true positive rate. 0.9 and 0.8 seems to satisfy our case.

Hide

```
res <-  predict(model, testing, type="response")
conf_matrix <- table(ActualValue=testing$Y, PredictedValue=res>0.8)
conf_matrix
```

```
           PredictedValue
ActualValue FALSE TRUE
          0     9   12
          1    12  301
```

Hide

```
accuracy <- conf_matrix
accuracy <- (accuracy[1]+accuracy[4])/sum(accuracy)
accuracy*100
```

```
[1] 92.81437
```

Hide

```
res <-  predict(model, testing, type="response")
conf_matrix <- table(ActualValue=testing$Y, PredictedValue=res>0.7)
accuracy <- conf_matrix
accuracy <- (accuracy[1]+accuracy[4])/sum(accuracy)
accuracy*100
```

```
[1] 93.71257
```

Even though the plot shows 0.8 and 0.9 both as better threshold options, The False positive rate is high on both of these threshold, the models seems to do great with 0.5 threshold and accuracy of 94%, so we will be keeping the same.

Hide

```
x <- as.matrix(training[-3])
x <- cbind(x, intercept=1)
y <- as.matrix(training[3])
```

Hide

```
###Sigmoid function
sigmoid=function(x) {
  1/(1+exp(-x))}
```

$$W_{t+1} = W_t + (\nabla^2 l(W_t))^{-1} \nabla l(W_t)$$

with: The Hessian

$$\nabla^2 l(W_t)) = -X^T Diag(\sigma(W^T X)_i (1 - \sigma(W^T X))_i) X$$

and the gradient

$$\nabla l(W_t) = X^T.(y - \sigma(W^T X))$$

Equation for weight updates:

The first line of formula above is divided into two parts which is presented in second and third line.

I will be incorporating this formula in the R code to calculate weight and update them using gradient descent.

Hide

```
# creating a function to predict the Y observations of the trainin
fit_logit=function(x,y,tol=1e-4,max_it=100, coeffs=1)
{
  #Algorithm initialization
  iterations=0
  converged=F
  #Weights are initialized to 1
  coeffs=matrix(coeffs,dim(x)[2])

  #Updates the weight until the max number of iter
  #Or the termination criterion is met
  while (iterations < max_it & !converged)
  {
    iterations=iterations+1
    predictions<-sigmoid(x %*% coeffs)
    predictions_diag=diag(predictions[,1])
    ##Weights update
    coeffs=coeffs + solve(t(x) %*% predictions_diag %*% x)%*% t(x) %*% (y-predictions
)
    ##compute mse to check termination
    mse=mean((y-sigmoid(x%*%coeffs))^2)
    ##Stop computation if tolerance is reached
    if (mse<tol)
    {
      converged=T
    }

  }
  ##Creates the logit objects
  my_logit=list("coeffs"=coeffs)
  my_logit[['preds']]=sigmoid(x%*%coeffs)
  my_logit[['residuals']]=my_logit[['preds']]-y
  my_logit[['mse']]=mean(my_logit[['residuals']]^2)
  my_logit[['conf_matrix']]=table(ActualValue=y, PredictedValue=my_logit[['preds']]>0
.5)
  my_logit[['accuracy']]= (sum(diag(my_logit[['conf_matrix']])))/(sum(my_logit[['conf
_matrix']]))
  return(my_logit)

}
```

“fit_logit” is the function which predicts the “y” observations of the training dataset.

“tol=10^-4” is the tolerance level i am choosing, it is the criteria for the while loop to end if the mean square error increases than 0.0001.

maximum iteration is kept to 100, if the weights dont decrease to 0.0001 by 100 iteraton the while loop will break and the value computed for weights will be used in sigmoid function to predict “y” values of training dataset.

iterations and convergance are the objects intilized with “0” and “False” first, I will be using it in the while loop for updating weights using gradient descent.

initial values of co-efficients is assigned to “1”, their dimension has to be having rows equal to x’s number of variables in 1 column.

while loop has two condition’s for number of iteration and not.converged status. If either of the conditions turns false, the while loop will stop executing

inside while loop we will execute the formula for weights/co-efficients updates, the sigmoid function gives out the prediction of “Y” values, using the formula (1/1+exp^-(x %*% coefficients)), %*% is the dot product used for matrix multiplication. dim(x) is (1000,3) and dim(coeffs) is (3,1) so the dot matrix will produce output of (1000,1), each of 1000 value will be substituted in sigmoid function and we get the output as an vector of 1000 observations.

we will need diag(sigmoid) as required in second line of the formula

solve() function is to find the values of co-effecients in linear or quadratic equation and also to find the inverse of given matrix. Mathematically inverse of an interger is equal to 1/the same integer eg: inverse of 10 is 1/10 or 10^-1.

mean square error is calculated by finding the mean of (y-predictions)^2. This mse will be used to compare with tolerance value mentioned, if it is greater than tolerance at any given iterations of 100, the while loop will break and the coefficients thus achieved wil be used for testing.

my_logit is the list which will store the values of co-efficients, predictions, residuals(which is difference between prediction and actual y values), mean square error. This list will be produced when we run the function with training x and y variables.

Hide

```
training_results <- fit_logit(x,y)
```

Hide

```
training_results[['coeffs']]
```

```
                   Y
X1        0.06839269
X2        1.24248674
intercept 3.23936760
```

Hide

```
training_results[['accuracy']]
```

```
[1] 0.9249249
```

Hide

```
training_results[['conf_matrix']]
```

```
           PredictedValue
ActualValue FALSE TRUE
          0     3   48
          1     2  613
```

I did try with 200 iterations, the accuracy is same as for 100 iteration, this means that the weights reach convergance where mse becomes greater than tolerance level before the iterations are completed. I feel the tolerance level is best as the value of 0.0001, if the mean difference between predictions and observation is less or equal to 0.0001 it is better for the model's predictions.

I am going to increase/decrease the tolerance level to check if it creats better accuracy models.

Hide

```
training_results_trial1 <- fit_logit(x,y,tol=1e-1)
training_results_trial1[['accuracy']]
```

```
[1] 0.9069069
```

the accuracy decreased

Hide

```
training_results_trial2 <- fit_logit(x,y,tol=1e-6)
training_results_trial2[['accuracy']]
```

```
[1] 0.9249249
```

the accuracy is same as before

Hide

```
training_results_trial3 <- fit_logit(x,y,coeffs = -0.5)
training_results_trial3[['accuracy']]
```

```
[1] 0.9264264
```

Even after changing weights=0 the accuracy is the same.

Hide

```
x_test <- NULL
x_test <- as.matrix(testing[-3])
x_test <- cbind(x_test, intercept=1)
y_test <- as.matrix(testing[3])
coefficients_train <- training_results[['coeffs']]
```

Hide

```
predict.my_logit<-function(coefficients_train,x_test)
{
  pred=sigmoid(x_test %*% coefficients_train)
  logit_test=list("predictions"= pred)
  logit_test[['residuals']]=logit_test[['predictions']]-y_test
  logit_test[['mse']]=mean(logit_test[['residuals']]^2)
  logit_test[['conf_matrix']]=table(ActualValue=y_test, PredictedValue=logit_test[['p
redictions']]>0.5)
  logit_test[['accuracy']]= (sum(diag(logit_test[['conf_matrix']])))/(sum(logit_test[
['conf_matrix']]))
  return(logit_test)

}
```

Hide

```
predict.my_logit(coefficients_train,x_test)[['accuracy']]
```

```
[1] 0.9431138
```

Hide

```
predict.my_logit(coefficients_train,x_test)[['conf_matrix']]
```

```
           PredictedValue
ActualValue FALSE TRUE
          0     2   19
          1     0  313
```

Hide

```
comparison <- data.frame(glm_accuracy = accuracy_0.5*100 , my_logit_accuracy = predic
t.my_logit(coefficients_train,x_test)[['accuracy']]*100)
print(comparison)
```

glm_accuracy <dbl>	my_logit_accuracy <dbl>
94.31138	94.31138

1 row

the glm() model prediction are same as the predict.my_logit model created from scratch

Hide

```
comparison_conf_mat <- list(glm_conf_mat = conf_matrix_0.5 , my_logit_conf_mat = pred
ict.my_logit(coefficients_train,x_test)[['conf_matrix']])
print(comparison_conf_mat)
```

```
$`glm_conf_mat`
           PredictedValue
ActualValue FALSE TRUE
          0     2   19
          1     0  313

$my_logit_conf_mat
           PredictedValue
ActualValue FALSE TRUE
          0     2   19
          1     0  313
```

The confusion matrix matches too.

references: http://enhancedatascience.com/2018/01/30/your-own-machine-learning-library-from-scratch-with-r/ (http://enhancedatascience.com/2018/01/30/your-own-machine-learning-library-from-scratch-with-r/)

https://rpubs.com/junworks/Understanding-Logistic-Regression-from-Scratch (https://rpubs.com/junworks /Understanding-Logistic-Regression-from-Scratch)

https://www.youtube.com/watch?v=Z5WKQr4H4Xk&t=3545s (https://www.youtube.com /watch?v=Z5WKQr4H4Xk&t=3545s)

https://www.youtube.com/watch?v=yIYKR4sgzI8&t=1s (https://www.youtube.com/watch?v=yIYKR4sgzI8&t=1s)