# Logistic Regression and Classification

Logistic regression is the classification algorithm in machine learning which is used to observe the classes of a discrete variable based on the features. It has wide application in the area of statistics and machine learning like in detection of fraud transaction, predict whether the person will default on his loan based on various parameters, predict whether the person has disease or not based on various symptoms shown. It is more relevant and excessively used where the response desired is in two classes like True, False or Yes, No. In linear regression the relationship is based on the linear combination of features whereas in Logistic regression it is based on the exponential power of features.

The probability estimator for logistic regression can be calculated as:-

1. Consider the features of the set as *x1 , x2……. xk*i.e k features and the target variable as *y.* In logistic regression the probabilities are obtained to approximate the class to which the target variable belongs. As it gives out probabilities thus the range is between 0 and 1. The logistic transformation equation is given by :-

(1)

The equation (1) can further be modified.

Applying exponential on both sides of equation (1):-

Transforming the above equation gives the probability of y as:-

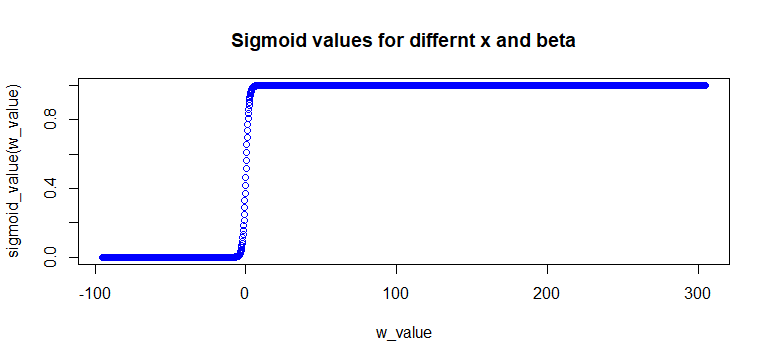
The right side of the above equation is the form of logistic sigmoid function *S(w)* where w = and the sigmoid function is given by:-

Thus the probability of y can be equated as:-

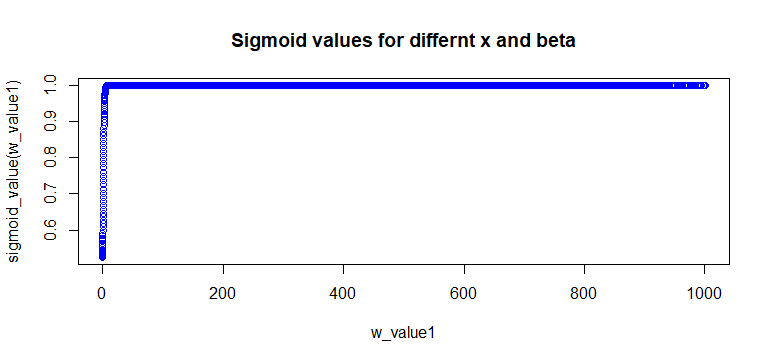
)

1. Considering the sigmoid function it can be observed that its value will always be between 0 and 1 as numerator contains exponential term which is positive and denominator is always greater than numerator due to addition. This can be proved graphically as well.

For the below graph x values were chosen between -100 and 100 and beta values from -100 to 100. As plotted the graph never crosses 1.



It was repeated again for x values between -1 and 1001 and beta values between 0 and 1. The resultant graph is :-



Thus it can be concluded that whatever be the values of x and beta the sigmoid function is never greater than 1.

Sigmoid function in logistic regression turns the score of the variables into the probability between 0 to 1. It became thus important for assigning label to the features by using this transformation. A probability value of say greater than 0.5 can be assigned to one class and the probability value of less than 0.5 can be assigned to other class. This way the result of probability estimator can be transformed to discrete classes.

# 2.1 ) Maximum Likelihood estimator of the model

To build the logistic regression model different approaches can be used. It can be made using the least square estimation which is based on minimizing the offset or residual of the features from the best fit curve of the datapoints and then assign the class labels to the features based on the best fit curve. Another method is to use the maximum likelihood estimator (MLE) of the features based on parameters that can be assigned to each feature. This method generates the probability of estimation of the target class based on various features, it is also a preferred method as it has better statistical properties. The objective of the MLE is to find the parameter values for features such that the labels can be assigned to them. To find the maximum likelihood estimator of a model following steps are processed:-

1. In logistic regression the class variable can take two values which are 1 if the feature belongs to that class else 0 if it doesn’t belong to that class. The distribution accounted for in logistic regression is the binomial distribution thus the probability function is given by:-

Thus the likelihood is then :-

Considering the dataset with n training examples with k features where is the class labels, are the features where (i) denotes the sample index and with k parameters . The likelihood can be generalised as:-

The log likelihood of the function is obtained by taking logs on both sides of the above equation. This will result into:-

As the products converts to summation after log is taken. The above equation will further simplified to :-

The value of as observed in the logistic regression equation is applied to the above equation. This further simplifies to by putting the value of p(x) :-

The above equation is the objective or the cost function for the log likelihood.

1. The above equation si the general equation the log likelihood of the event. To get the maximum likelihood of the event the above equation needs to be differentiated and then equate to zero. To do the partial differentiation of the log likelihood with β say where j is between 1 and k. The equation will reduce to :-

The above equation is arrived at by using the formula of and also partial differentiation of terms where there is no is equal to zero. The above equation further yields to :-

This finally comes to be, by using the equation of logistic function :-

That is :-

1. To maximize the above log likelihood function the differential is set to zero this gives:-

There are two possibilities of the above equation to be zero i.e the feature is zero for all i from 1 to n. this is not possible as this will give rise to equation without any feature . For the value of j = 0 , we know that the value of for all I, this gives the other possibility as :-

The above equation on simplification will yield the score equation which is :-

The above score equation is statistically important to calculate the maximum likelihood estimator. It should be noted that the above equation will give the critical points. That point could be either minimum or maximum. The probability estimator from the above equation results in the calculation by substituting the value of probability in the log likelihood equation.

1. The objective of logistic regression models is to find the probability of a target class to belong to a particular class based on the features and parameters defining the probabilities. To achieve this objectives various methods can be applied which includes methods of moments, newton raphson method, maximum likelihood method and by using various kind of distribution functions like binomial distribution, geometric distribution, Poisson distribution.

The objective of maximum likelihood method is to the find the probability estimator of the class. The class of the target variable can then be assigned to one which shows maximum probability estimate. Thus the log likelihood function when obtained with different features and parameters should give maximum probability.

This is done by doing the partial derivative of log likelihood function and equating that to zero. It is known that the values so obtained will give the critical points. These critical points include the maximum and minimum value. Thus the above equation gives value which could be minimum likelihood or the maximum likelihood that’s why these equations are known as transcendental equations and not the closed form equation because in the closed form equation the value so obtained is assigned to the variable.

# Implementation of Gradient Ascent in R

## In this section the implementation of Gradient Ascent algorithm on the heart dataset will be done.

The first part is to import the dataset on which the algorithm will be implemented. following code is used for importing the dataset into R

library(readr)  
SAheart <- read\_csv("SAheart.data")

## Parsed with column specification:  
## cols(  
## row.names = col\_double(),  
## sbp = col\_double(),  
## tobacco = col\_double(),  
## ldl = col\_double(),  
## adiposity = col\_double(),  
## famhist = col\_character(),  
## typea = col\_double(),  
## obesity = col\_double(),  
## alcohol = col\_double(),  
## age = col\_double(),  
## chd = col\_double()  
## )

View(SAheart)

The above code is used for the importing of dataset. It is observed that it has 462 observations with 11 variable. Firstly a brief look at the dataset is required which can be achieved through below option.

**str**(SAheart)

## tibble [462 x 11] (S3: spec\_tbl\_df/tbl\_df/tbl/data.frame)  
## $ row.names: num [1:462] 1 2 3 4 5 6 7 8 9 10 ...  
## $ sbp : num [1:462] 160 144 118 170 134 132 142 114 114 132 ...  
## $ tobacco : num [1:462] 12 0.01 0.08 7.5 13.6 6.2 4.05 4.08 0 0 ...  
## $ ldl : num [1:462] 5.73 4.41 3.48 6.41 3.5 6.47 3.38 4.59 3.83 5.8 ...  
## $ adiposity: num [1:462] 23.1 28.6 32.3 38 27.8 ...  
## $ famhist : chr [1:462] "Present" "Absent" "Present" "Present" ...  
## $ typea : num [1:462] 49 55 52 51 60 62 59 62 49 69 ...  
## $ obesity : num [1:462] 25.3 28.9 29.1 32 26 ...  
## $ alcohol : num [1:462] 97.2 2.06 3.81 24.26 57.34 ...  
## $ age : num [1:462] 52 63 46 58 49 45 38 58 29 53 ...  
## $ chd : num [1:462] 1 1 0 1 1 0 0 1 0 1 ...  
## - attr(\*, "spec")=  
## .. cols(  
## .. row.names = col\_double(),  
## .. sbp = col\_double(),  
## .. tobacco = col\_double(),  
## .. ldl = col\_double(),  
## .. adiposity = col\_double(),  
## .. famhist = col\_character(),  
## .. typea = col\_double(),  
## .. obesity = col\_double(),  
## .. alcohol = col\_double(),  
## .. age = col\_double(),  
## .. chd = col\_double()  
## .. )

The above structure of the dataset tells that all the variables are numeric in nature except ‘famhist’ which is a character type. It is also observed that the response variable which is ‘chd’ is also numeric.

Also as the prediction and training of the dataset is to be done using a single ‘x’ variable which is ldl(low density lipoprotein cholesterol) thus the dataset will be modified and only these two variables will be retained for further calculation and building of model. The following code is used to achieve the above objective.

SAheart\_data <- SAheart[,c("ldl","chd")]

## Normalization of feature variable

The feature variable has values that could range from very low to very high. To make the estimator of response so that no one feature value overwhelm the value of small one, and to make the convergence faster normalization of feature is used. There are many ways through which it can be achieved like to have values of feature from 0 to 1 the actual value can be subtracted from the minimum value and the result can be divided by the difference between the minimum and maximum value. Another is to have the zero mean of the feature variable which can be achieved by subtracting the value of feature with the mean and dividing the result with the standard deviation. For this analysis the later approach will be used.

mu\_x <- **mean**(SAheart\_data**$**ldl)  
mu\_x

## [1] 4.740325

sigma\_x <- **sd**(SAheart\_data**$**ldl)  
sigma\_x

## [1] 2.070909

The above code gets the value of mean as 4.740325 and the value of standard deviation as 2.070909. The next step is to normalize the feature.

SAheart\_data$ldl <- (SAheart\_data$ldl - mu\_x)/sigma\_x

To check the feature variable is normalized:-

**mean**(SAheart\_data**$**ldl)

## [1] -8.420712e-17

The mean value is -8.420712e-17 which is close to zero. Thus the normalization of feature is achieved.

## Splitting the dataset in test and train

To know the accuracy of the model created and to verify the dataset is divided into two groups with train dataset and test dataset. For training more points are used as this will generalize the model and to get the effective values of parameter. This is done with following code.

train\_data = SAheart\_data[1:100,]  
test\_data = SAheart\_data[101:130,]

In the train dataset there are 100 rows of the original dataset and in the test dataset there are 30 rows after.

## Initialization of parameter values.

As there is only one feature to be used the parameter will have two values which are beta\_0 and beta\_1. initializing the value of parameter to some number is achieved through following code.

beta <- rep(0,2)

## Value of objective function l(beta)

The value is achieved by substituting the values of feature and parameter first in the probability estimator of class variable.

X <- as.matrix(train\_data$ldl)  
y <- as.matrix(train\_data$chd)  
  
p = exp(beta[1]+beta[ 2 ]\*X ) /(1+exp(beta[1]+beta[ 2 ]\*X ) )

## defining the learning rate and tolerance limit.

How the model is learned at each stage of processing and deciding the speed of convergence learning rate is used. It should be such a value that the maximum likelihood can be observed so it has to be that slow enough and should be that fast that it shoots to the other side of maximum likelihood value without interpreting that value. Tolerance limit will define the error with which the convergence can be made.

learning\_rate <- seq(0,1,0.2)  
epsilon = 0.001

The above code generates 6 values of learning rate from 0 to 1 and set the tolerance limit as 0.001

## Calculating the Gradient

The next step is to maximize the objective function. This is achieved through the loop where the initial values are fed the gradient is calculated for each feature and then the values of parameter are updated till the convergence is achieved within the tolerance limits.

sigmoid <- function(z) 1/(1 + exp(-1\*z))  
  
beta = c(0.1,2)  
  
ascent\_algo<- function(beta,X,y,learning\_rate)  
{  
 for ( i in 1 : 100 )  
 {   
   
 p = exp(beta[1]+beta[ 2 ]\*X[i] ) /(1+exp(beta[1]+beta[ 2 ]\*X[i] ) )  
 g = colMeans ( cbind( y[i]−p , ( y[i]−p)\*X[i] ) )  
 beta = beta + learning\_rate\*g   
   
 }  
 return(beta)  
}

The above code is for the gradient ascent algorithm, where first the sigmoid function is defined using its definition. Then the ascent\_algo functions is defined which takes the value of training data, response variable of that training data and learning rate. First the probabilities of the class labels is calculated. The probabilities are then fed to the maximum likelihood function and values of gradient is calculated. The values are assigned to the vector g. The parameters value are then updated based on the learning rate and the gradient value. The above function returns the parameters based on the gradient ascent algorithm

## Updating parameters and convergence

The ascent function is then used to optimize the values of parameters based on the tolerance limit and for a defined learning rate. Below code has taken the initial value of tolerance as 0.9 and the desired value of tolerance as 0.001. The algorithm converges when the difference between the input parameters and the calculated parameters is 0.001.

beta\_test <- beta  
epsilon\_e <- 0.9  
while (epsilon\_e > 0.001) {  
 answer <- ascent\_algo(beta\_test,X,y,learning\_rate = 0.2)  
 beta\_new <- answer  
 epsilon\_e <- max(abs(beta\_test - beta\_new))  
 beta\_test <- beta\_new   
}

## Completeing the implementation

To complete the implementation of gradient ascent algorithm, the model prediction is used. If the accuracy is not goo enough then learning rate can be changed accordingly. For its implementation it is assumed that for probability values of greater than 0.5 the class label assigned to them will be 1 and if the probability value is less than 0.5 then it will assigned the class 0.

for (i in 1:100)  
{p[i] = exp(beta\_new[1]+beta\_new[ 2 ]\*X[i] ) /(1+exp(beta\_new[1]+beta\_new[ 2 ]\*X[i] ))  
}  
   
p[p>0.5] <- 1  
p[p<0.5] <- 0  
  
# Finding accuracy of the model  
accuracy <- (sum(p==y, na.rm=T))/ length(p)\*100  
accuracy

## [1] 63

The above model at learning rate of 0.2 gives accuracy of only 63%. To optimize it different value of learning rate and tolerance will be used.

beta\_test <- rep(0,2)  
epsilon\_e <- 1  
while (epsilon\_e > 0.001) {  
 answer <- ascent\_algo(beta\_test,X,y,learning\_rate = 0.5)  
 beta\_new <- answer  
 epsilon\_e <- max(abs(beta\_test - beta\_new))  
 beta\_test <- beta\_new  
   
}  
  
for (i in 1:100)  
{p[i] = exp(beta\_new[1]+beta\_new[ 2 ]\*X[i] ) /(1+exp(beta\_new[1]+beta\_new[ 2 ]\*X[i] ))  
}  
  
p[p>0.5] <- 1  
p[p<0.5] <- 0  
  
# Finding accuracy of the model  
accuracy <- (sum(p==y, na.rm=T))/ length(p)\*100  
accuracy

## [1] 65

For learning rate of 0.5 and tolerance at 0.001 the accuracy achieved is 65%, whereas for learning rate of 0.3 it is 64%. So for the completion of model training the value of learning\_rate as 0.5 is chosen which gives the parameters as:- beta\_0 = -0.97 and beta\_1 = 1.1

## Predict the labels for set of test examples

The above parameter values will be used now on the training set to determine how well the gradient descent model is able to predict the labels. It is achieved through following code

X\_feature <- as.matrix(test\_data$ldl)  
y\_label <- as.matrix(test\_data$chd)  
row\_test <- nrow(X\_feature)  
col\_test <- ncol(X\_feature)  
p\_y <- matrix(NA,row\_test,col\_test)  
  
# Calculating the probabilities based on the parameter value  
  
for (i in 1:row\_test)  
{p\_y[i] = exp(beta\_new[1]+beta\_new[ 2 ]\*X\_feature[i] ) /(1+exp(beta\_new[1]+beta\_new[ 2 ]\*X\_feature[i] ))  
}  
  
p\_y[p\_y>0.5] <- 1  
p\_y[p\_y<0.5] <- 0  
  
# Finding accuracy of the model  
accuracy\_test <- (sum(p\_y==y\_label, na.rm=T))/ length(p\_y)\*100  
accuracy\_test

## [1] 73.33333

# [1] 73.33333

Testing the accuracy of the model on the test dataset gives 73.33 % which is a good accuracy score.

# Conclusion

Logistic regression has many applications and is widely used in some critical areas of decision making. The financial analysis and to determine the eligibility of a person, to check for the disease and model it are mostly done with the use of logistic regression. Th method of loglikelihood can be used for finding the probabilities of the target variable based on the features available.

The gradient ascent algorithm was used for predicting the disease based on the single feature of the dataset. There can be various modifications that could be done like including other features to determine the disease. This can modify the model to a large extent and also include various symptoms that could be used for the determination of disease.