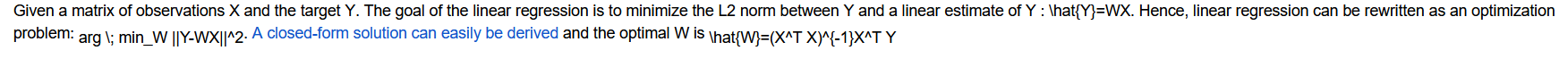
When dealing with Machine Learning problems in R, most of the time you rely on already existing libraries. This fastens the analysis process, but do you really understand what is behind the algorithms? Could you implement a logistic regression from scratch with R?  
The goal of this post is to create our own basic machine learning library from scratch with R. We will only use the linear algebra tools available in R. There will be three posts:

1. Linear and logistic regression (this one)
2. PCA and k-nearest neighbors classifiers and regressors
3. Tree-based methods and SVM

**Linear Regression (Least-Square)**

The goal of liner regression is to estimate a continuous variable given a matrix of observations . Before dealing with the code, we need to derive the solution of the linear regression.

**Solution derivation of linear regression**



**Linear regression in R**

Using the closed-form solution, we can easily code the linear regression. Our linear model object will have three methods, an init method where the model is fitted, a predict method to work with new data and a plot method to visualize the residuals’ distribution.

###Linear model

fit\_lm<-function(x,y,intercept=TRUE,lambda=0)

{

##Conversion to matrix if required

if (!is.matrix(x))

{

x=as.matrix(x)

}

if (!is.matrix(y))

{

y=as.matrix(y)

}

#Add the intercept coefficient

if (intercept)

{

x=cbind(x,1)

}

my\_lm=list(intercept=intercept)

##Compute coefficients estimates

my\_lm[['coeffs']]=solve(t(x) %\*% x) %\*% t(x) %\*% y

##Compute estimates for the train dataset

my\_lm[['preds']]=x %\*% my\_lm[['coeffs']]

my\_lm[['residuals']]=my\_lm[['preds']]-y

my\_lm[['mse']]=mean(my\_lm[['residuals']]^2)

attr(my\_lm, "class") <- "my\_lm"

return(my\_lm)

}

The fit function is simple, the first few lines transform the data to matrices and add an intercept if required. Then, the ‘my\_lm’ object is created and the coefficients are computed. The *solve()* function is used to invert the matrix and *%\*%* denotes matrix multiplication. A the end, the residuals and the estimates are computed and the class of the object is set as ‘my\_lm’.  
Now let’s implement the predict and plot methods for the my\_lm class:

predict.my\_lm<-function(my\_lm,x,..)

{

if (!is.matrix(x))

{

x=as.matrix(x)

}

if (my\_lm[["intercept"]])

{

x=cbind(x,1)

}

x%\*%my\_lm[["coeffs"]]

}

plot.my\_lm<-function(my\_lm,bins=30,..)

{

library(ggplot2)

qplot(my\_lm[["residuals"]], geom="histogram",bins=bins) + xlab('Residuals values') + ggtitle('Residual distribution')

}

You can test the code on some preinstalled R dataset such as the *car* one. The code will give you the same coefficients estimates as the *lm* function. For instance, on the *car* dataset:

my\_lm1=fit\_lm(cars[,1],cars[,2])

vanilla\_lm=lm(dist~speed,cars)

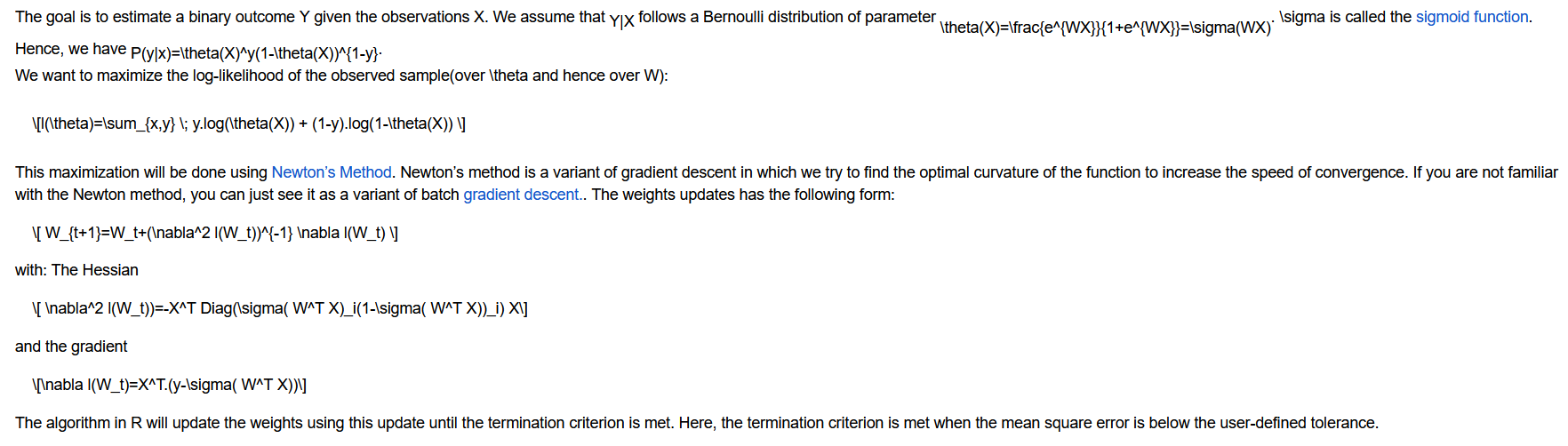
print(vanilla\_lm[['coefficients']])

print(my\_lm1[['coeffs']])

**Logistic regression**

Previously, we worked on regression and the estimation of a continuous variable. Now, with logistic regression, we try to estimate a binary outcome (for instance, ill vs healthy, pass vs failed, …). Again, let’s deal with the maths first:

**The mathematics of logistic regression**



**Logistic regression in R**

###Sigmoid function

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

###Fit logistic regression

fit\_logit=function(x,y,intercept=T,tol=10e-5,max\_it=100)

{

##Type conversion

if (!is.matrix(x))

{

x=as.matrix(x)

}

if (!is.matrix(y))

{

y=as.matrix(y)

}

##Add intercept is required

if (intercept)

{

x=cbind(x,1)

}

##Algorithm initialization

iterations=0

converged=F

##Weights are initialized to 1

coeffs=matrix(1,dim(x)[2])

##Updates the weight until the max number of iter

##Or the termination criterion is met

while (iterations0.5

}

}

The code is split into two part:

* The fit part, where the logistic model is fitted using Newton’s method.  
  This part has three main components. First, the data is put in the proper matrix format and, if required, the intercept is added. Then the algorithm parameters are updated (initially all the weights are set to one).  
  Finally, the algorithm updates the weights until the MSE goes below the tolerance. The most important line is probably the weights update one where the update formula is used to update the weights of the model.
* The predict method, where the outcome is estimated using the weights computed previously.

We can now use our logistic regression to predict the class of a flower from the iris dataset:

fit\_logit(iris[,1:4],iris[,5]=='setosa')

As expected, the algorithm can predict efficiently if a flower is a setosa or not.