Homework 2

Ruggeri D., Ercolino S., Tazza G.

25/5/2021

Exercise 1: The Bayes Classifier

1. Bayes strategy/classifier

Given the definition of the conditional (**pointwise**) risk $R(s \mid \mathbf{x})$ for a strategy $s(\mathbf{x})$, the **Bayes Strategy (predictor)** can be defined as the strategy s_opt which minimizes the conditional risk, as follows:

$$s_{\text{opt}}(\mathbf{x}) = \underset{s \in \mathcal{Y}}{\text{argmin}}\ R(s \mid \mathbf{x})$$

Which yields the **Bayes Risk**:

$$R_{opt} = R(s_{opt}) = \mathbb{E}_{\mathbf{X}} \inf_{s \in \mathcal{Y}} R(s \mid \mathbf{X})$$

So, in the case of Regression ($Y \in \mathcal{Y} = \mathbb{R}$ and $\mathbf{X} = [X_1, ..., X_p]^{\mathsf{T}}$), the Bayes strategy that minimizes the conditional risk can be shown to be the **Regression function**:

$$f_{\text{opt}}(\mathbf{X}) = \underset{\text{all } f}{\text{argmin}}\ R[f(\mathbf{X})] = \mathbb{E}(Y \mid \mathbf{X})$$

And the Bayes classifier (in the case of binary classification where $Y \in \mathcal{Y} = \{0, 1\}$) can be obtained by dicothomizing the regression function (which in this case becomes $f_{\text{opt}}(\mathbf{X}) = \mathbb{E}(Y \mid \mathbf{X}) = \mathbb{P}(Y = 1 \mid \mathbf{X})$) to define the classification rule h_{opt}, which can be proven to be again the strategy (classifier) that minimizes the (classification) risk:

$$h_{\text{opt}}(\mathbf{x}) = \underset{\text{all } h}{\text{argmin}}\ R\left[h(\mathbf{x})\right] = \mathbb{I}\left(f_{\text{opt}}(\mathbf{x}) \geq \frac{1}{2}\right)$$

This means, in general, that any classifier h can at best be expected to perform as well as the theoretical Bayes classifier, which could be build under the assumption of perfect information about the data generating process. So if a plug-in classifier was built $\hat{h}_n(\mathbf{x}) = \mathbb{I}(\hat{f}_n(\mathbf{x}) \geq 1/2)$, with $\hat{f}_n$ close to f_{opt}, then the plug-in classification risk would be close to the Bayes risk, thus being close to minimizing the classification risk.

2. Bayes Classification Rule $h_{opt}(x)$

The Bayes classification rule $h_{opt}(x)$ for a binary classification (s.t. $Y \in \mathcal{Y} = \{0, 1\}$) is given by:

$$h_{opt}(x) = \mathbb{I}\left[\mathbb{P}(Y = 1 \mid \mathbf{X} = \mathbf{x}) > \frac{1}{2}\right]$$

Where (simplifying the notation for the $\mathbf{X}$), we have the Regression Function:

$$\mathbb{P}(Y = 1 \mid \mathbf{X}) = (Bayes) = \frac{\mathbb{P}(\mathbf{X} \mid Y = 1) \cdot \mathbb{P}(Y = 1)}{\mathbb{P}(\mathbf{X})} = \frac{\mathbb{P}(\mathbf{X} \mid Y = 1) \cdot \mathbb{P}(Y = 1)}{\mathbb{P}(\mathbf{X} \mid Y = 1) \cdot \mathbb{P}(Y = 1) + \mathbb{P}(\mathbf{X} \mid Y = 0) \cdot \mathbb{P}(Y = 0)}$$

Since we assume (X, Y) r.v. with $Y \in \{0, 1\}$ and $X \in \mathbb{R}$, having conditional distribution:

$$(X \mid Y = 1) \sim Unif(-1, 3) \quad and \quad (X \mid Y = 0) \sim Unif(-3, 1)$$

and:

$$\mathbb{P}(Y = 1) = \mathbb{P}(Y = 0) = \frac{1}{2}$$

But, since the conditional distributions of X are uniforms, we can rewrite the probabilities as:

$$\mathbb{P}(X \mid Y = 1) = \frac{1}{4}\mathbb{I}_{[-1,3]}(x) \quad and \quad \mathbb{P}(X \mid Y = 0) = \frac{1}{4}\mathbb{I}_{[-3,1]}(x)$$

So, by substituting the values, the Regression Function $\mathbb{P}(Y = 1|X)$ for this classification problem becomes:

$$\mathbb{P}(Y = 1 \mid X) = \frac{\frac{1}{8}\mathbb{I}_{[-1,3]}(x)}{\frac{1}{8}\mathbb{I}_{[-3,1]}(x) + \frac{1}{8}\mathbb{I}_{[-1,3]}(x)} = \frac{\mathbb{I}_{[-1,3]}(x)}{\mathbb{I}_{[-3,1]}(x) + \mathbb{I}_{[-1,3]}(x)} = \frac{\mathbb{I}_{[-1,1)}(x) + \mathbb{I}_{[1,3]}(x)}{\mathbb{I}_{[-3,-1]}(x) + 2\mathbb{I}_{[-1,1]}(x) + \mathbb{I}_{[1,3]}(x)} =$$

$$= \begin{cases} 0 & if \quad -3 \leq x < -1 \\ \frac{1}{2} & if \quad -1 \leq x < 1 \\ 1 & if \quad 1 \leq x \leq 3 \end{cases} \implies$$

$$\implies \mathbb{P}(Y = 1 \mid X) = \frac{1}{2}\mathbb{I}_{[-1,1)} + \mathbb{I}_{[1,3]}$$

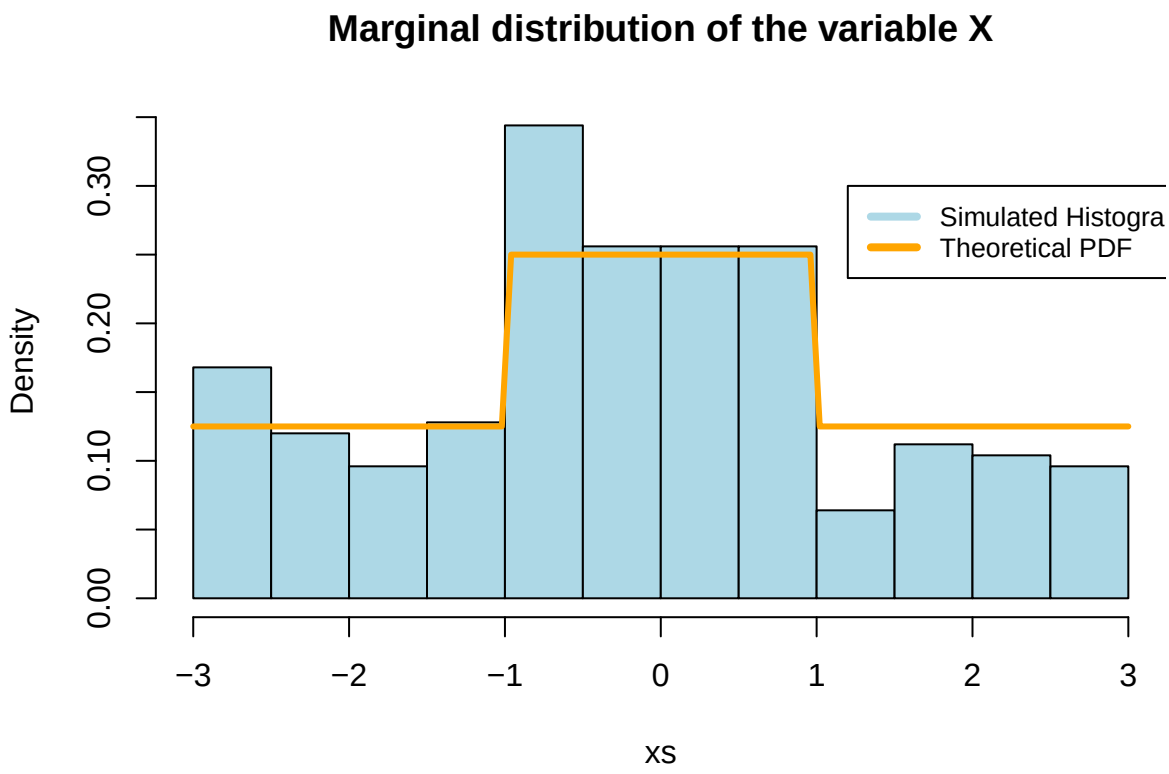
And the classification rule h_{opt} can be written as:

$$h_{opt}(x) = \mathbb{I}_{\left[\mathbb{P}(Y=1 \mid X) > \frac{1}{2}\right]}(x) = \mathbb{I}_{\left[\frac{1}{2}\mathbb{I}_{[-1,1)} + \mathbb{I}_{[1,3]} > \frac{1}{2}\right]}(x) = \mathbb{I}_{[1,3]}(x)$$

Which means that the observations are classified as $y = 1$ when the feature x has values in $[1, 3]$, and $y = 0$ otherwise.

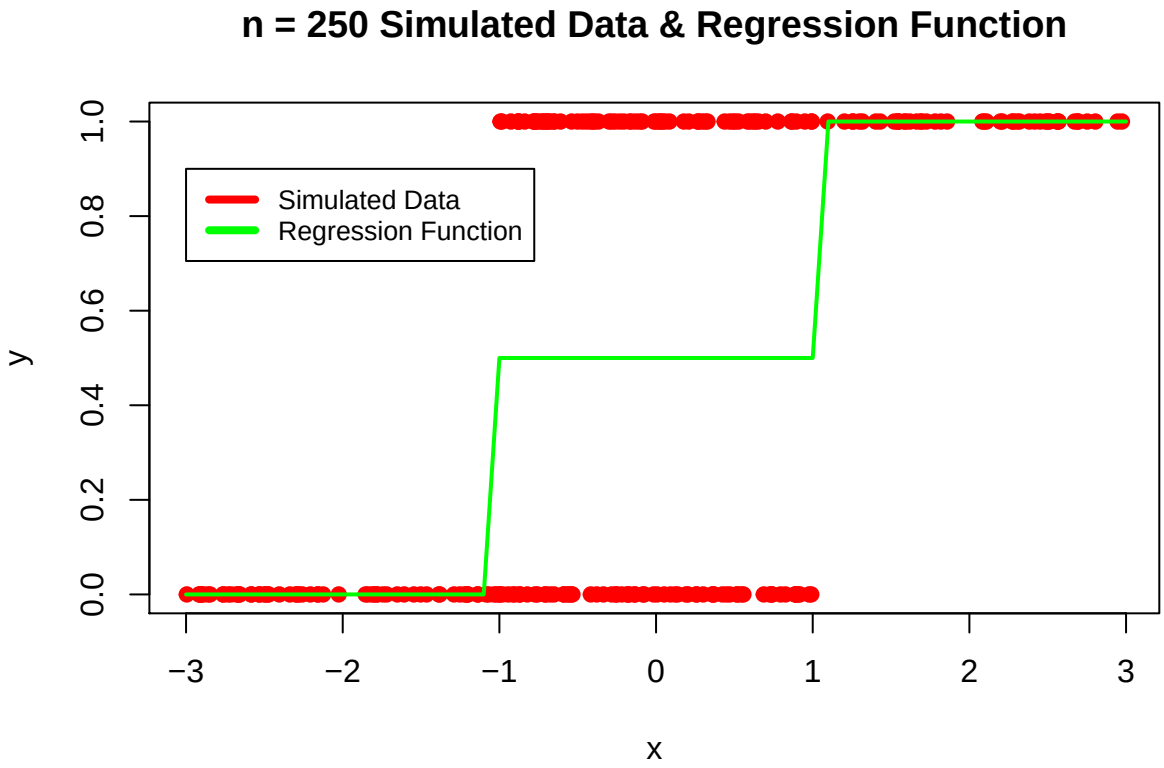
```
n = 250

#data model

p <- function(x, y){if(y==0){dunif(x, -3, 1) * 1/2}
    else if(y==1){dunif(x, -1, 3) * 1/2}
    else 0}
# curve(p(x,0) + p(x, 1), xlim = c(-5, 5),col = 'red')

ys <- rbinom(n, 1, 0.5)
ys_test <- rbinom(n, 1, 0.5)
gen <- function(y){if(y==0){runif(1, -3, 1)}
    else if(y==1){runif(1, -1, 3)}}
gen <- Vectorize(gen)
xs <- gen(ys)
xs_test <- gen(ys_test)
par(mfrow=c(1,1))
plot.new()
hist(xs, freq = F, col = 'light blue', main = 'Marginal distribution of the variable X')
curve(p(x,0) + p(x, 1),col = 'orange', add = T, lwd = 3)
legend(1.2, 0.3, legend = c('Simulated Histogram', 'Theoretical PDF'), col = c('light blue', 'orange'),
       lty = 1, lwd = 4, cex=0.8)
```

3. Simulate n = 250 data from the joint data model p(y, x) = p(x|y) · p(y) described above:




```
# Regression function
plot(xs, ys, col='red', pch=19,
     main = 'n = 250 Simulated Data & Regression Function',
     xlab = 'x', ylab = 'y')
reg.fun <- function(x){ p(x, 1)/(p(x,  0) + p(x, 1))}
curve(reg.fun, xlim = c(-5,5), lwd=2, col = 'green', add = T)
legend(-3, 0.9, legend = c('Simulated Data', 'Regression Function'),
       col = c('red', 'green'),
       lty = 1, lwd = 4, cex=0.8)
```

· ***Plot the data together with the regression function that defines*** $h_{opt}(x)$ ***and Evaluate the performance of the Bayes Classifiers on these simple***




```
h.opt <- function(x) {if(reg.fun(x) > 0.5) {1}
                      else 0}
h.opt <- Vectorize(h.opt)
accuracy.Bayes = sum(h.opt(xs_test) == ys_test)/length(ys_test)
sprintf("Accuracy of the Bayes Classifier:  %s", accuracy.Bayes)
```

```
## [1] "Accuracy of the Bayes Classifier:  0.76"
```

By looking at the marginal distribution of X, we know that $\sim 50\%$ of the simulated data points lie in the central interval $(-1, 1)$, where we know that the probability that observations in this interval belong to class $Y = 1$ is exactly $\mathbb{P}(Y = 1 \mid X \in (-1, 1) = 1/2$ (also the value of the regression function in this interval). This means that, in average, around half of the $\sim 50\%$, so $\sim 25\%$ of the simulated data points would be misclassified by the Bayes classifier. So the classification accuracy we expect to see is $\sim 75\%$ (with fluctuation due to the estimation based on $n = 250$ simulated data points), and this is confirmed by the accuracy obtained through simulation.

· ***Apply any other classifier of your choice to these data and comparatively comment its performance*** As another classifier we chose logistic regression one:

```
train= data.frame(X = xs, y =  ys)
test= data.frame(X = xs_test, y =  ys_test)


model = glm(y ~ X, data = train, family = "binomial")
probabilities <- predict(model, newdata=test, type='response')
predicted.classes <- ifelse(probabilities > 0.5, 1, 0)
accuracy.Log = sum(predicted.classes == test$y)/length(test$y)
sprintf("Accuracy of the Log classifier:  %s", accuracy.Log)
```

```
## [1] "Accuracy of the Log classifier:  0.744"
```

Although the Bayes classifier is the one that minimizes the risk, we can expect that another classifier may outperform it in a single test limited to $n = 250$ observations, but this is not the case in this specific test. Nonetheless, since we can simulate exactly from the data generating process, the best idea is to perform a test through repeated sampling to confirm (if successful) that the Bayes classifier is indeed the "best", in the sense that it is the one that minimizes the misclassification error, and that another classifier h (in our case a logistic regression), can at best match the average performance of the classifier h_{opt}.

4. Repeat the sampling M = 1000 times keeping n = 250 fixed

```
M = 1000
n = 250
total_acc_Bayes = c()
total_acc_Log = c()
for(i in 1:M){
    # generating samples of n=250 observations
    ys <- rbinom(n, 1, 0.5)
    ys_test <- rbinom(n, 1, 0.5)
    xs <- gen(ys)
    xs_test <- gen(ys_test)
    total_acc_Bayes = cbind(total_acc_Bayes,  c(sum(h.opt(xs_test) == ys_test)/length(ys_test)))
    # Logistic Regression
    train= data.frame(X = xs, y =  ys)
    test= data.frame(X = xs_test, y =  ys_test)
    model = glm(y ~ X, data = train, family = "binomial")
    probabilities <- predict(model, newdata=test, type='response')
    predicted.classes <- ifelse(probabilities > 0.5, 1, 0)
    total_acc_Log = cbind(total_acc_Log, c(sum(predicted.classes == test$y)/length(test$y)))
}
avg_acc_Bayes = mean(total_acc_Bayes)
avg_acc_Log = mean(total_acc_Log)
sd_acc_Bayes = sd(total_acc_Bayes)
sd_acc_Log = sd(total_acc_Log)
sprintf("Bayes classifier, Mean:  %s, Std: %s ", avg_acc_Bayes, sd_acc_Bayes)
```

```
## [1] "Bayes classifier, Mean:  0.750524, Std: 0.0270473840691662 "
```

```
sprintf("Log classifier, Mean:  %s, Std: %s ", avg_acc_Log, sd_acc_Log)
```

```
## [1] "Log classifier, Mean:  0.748936, Std: 0.0281091093068782 "
```

The accuracy of both the models are very close, even thought the performance of the Bayes Classifier is slightly higher then the one of the Logistic Regression. This may due by the fact that in this setup the main source of error is the irreducible part, since in the intervals $[-1, 1]$ the model doesn't have any information for discriminate between the two classes. Then, in one end we have the Bayes Classifier that was realized knowing the data generating process, so having as error only the irreducible component, and on the other end the Logistic Regression that approximate to the same solution of the Bayes Classifier, but with a small approximation error that gets added to the irreducible component.

=======

Exercise 2: Go Gradient, Go!

Loading the dataset and encoding the responses right after:

```
load("data/amazon_review_clean.RData")

# changing labels of y vectors to book = TRUE, movie = FALSE
y_tr = (y_tr == "book")
y_te = (y_te == "book")
```

Creating the validation dataset from train:

```
val.indices = sample(seq(1, length(y_tr)), 50000, replace = F)
X_val = X_tr[val.indices, ]
y_val = y_tr[val.indices]
X_tr = X_tr[-val.indices, ]
y_tr = y_tr[-val.indices]
```

· ***Gradient*** Defining the gradient computation and computing the $X.y$ matrix in advance:

```
# differentiation variables:
gradient = function(n, X, X.y, X.t.beta){
    1/n * (t(X) %*% (1/(1+exp(-X.t.beta))) - X.y)
}

X.y = t(X_tr)%*%y_tr
```

· ***Parameters*** Now is time to initialize the parameters and hyper parameters:

```
# regression parameters
beta = rnorm(ncol(X_tr), sd=0.1)

# hyper parameters:
alpha = 0.2 # learning rate
tolerance = 0.00001 # tolerance for the convergence condition
epochs = 350 # early stop on iterations

n = nrow(X_tr)
```

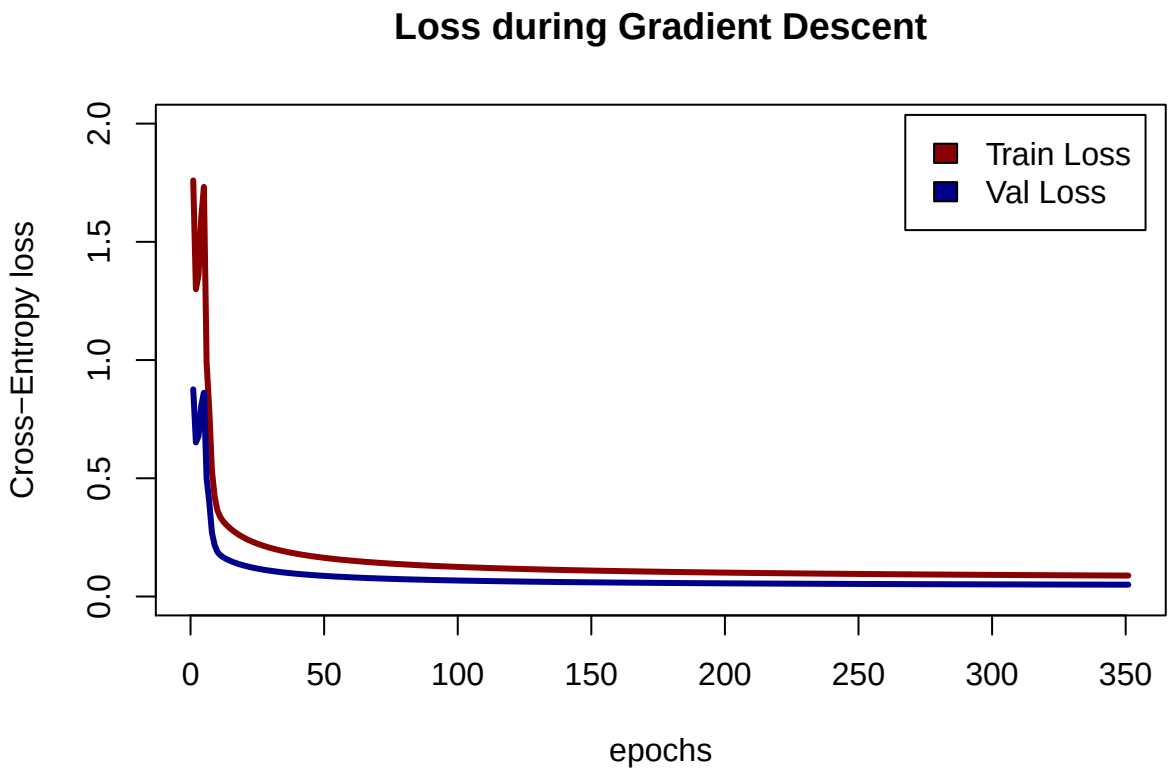
· ***Starting the training with the Gradient Descent*** The chunk of code below executes the training of the classifier by gradient descent.

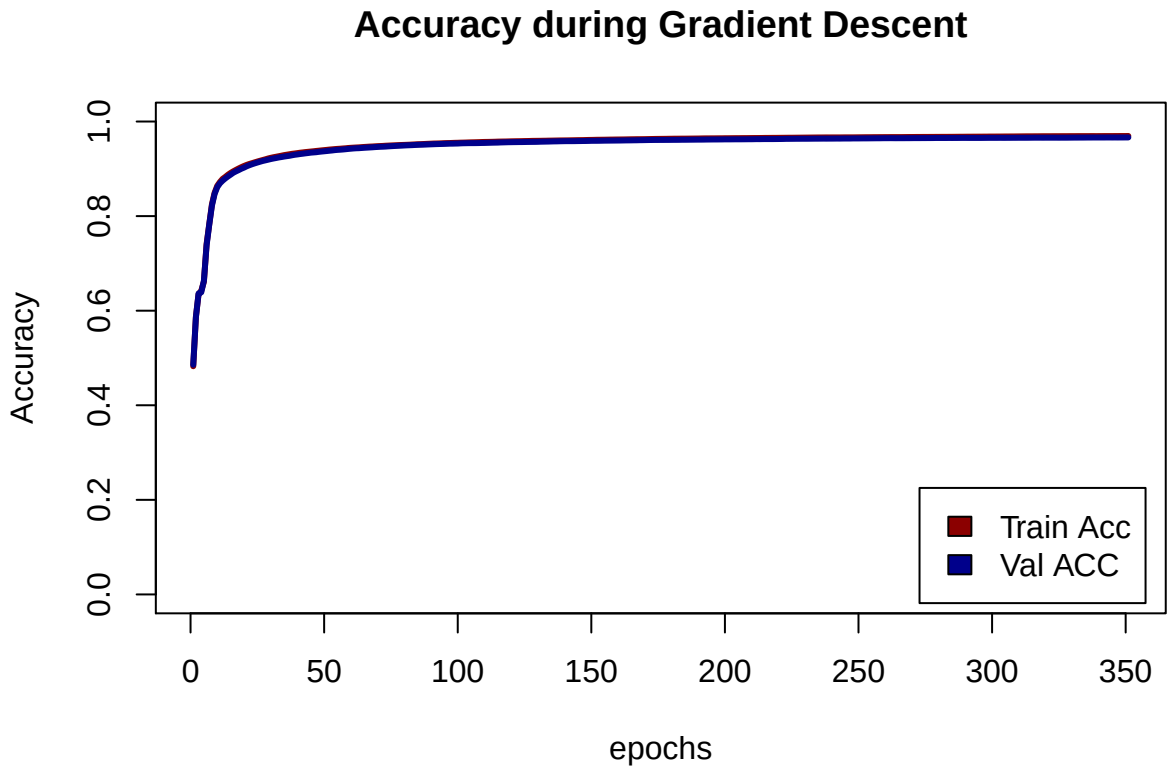
```
it = 1
continue = T
while (continue) {

  # updating the parameters
  beta = beta - alpha * gradient(n=nrow(X_tr), X_tr, X.y, X.t.beta)
  X.t.beta = X_tr%*%beta
  X.v.beta = X_val%*%beta
  old.val.loss = val.loss
  loss = 1/n * sum(log(1+exp(X.t.beta)) - y_tr*X.t.beta)
  val.loss <- 1/n * sum(log(1+exp(X.v.beta)) - y_val*X.v.beta)
```

```
  # logging the metrics for plotting
  loss.hist <- cbind(loss.hist, c(loss))
  val.hist <- cbind(val.hist, c(val.loss))
  train.acc = Accuracy(y_pred = X.t.beta>0, y_true = y_tr)
  val.acc = Accuracy(y_pred = X.v.beta>0, y_true = y_val)
  train.accs <- cbind(train.accs, c(train.acc))
  val.accs <- cbind(val.accs, c(val.acc))

  # conditions for stop iterating
  if (((it>50) && (old.val.loss <= val.loss + tolerance)) || (it==epochs)){continue = F}
  it = it + 1
}
```

· ***Plots***

Here the descending curves:





At the beginning the Loss is pretty unstable, but after 20 epochs it descents rapidly ad gets to a plateau slightly decreasing; We can also observe that we didn't reach convergence -with tolerance of 10^{-5}-, because the early stop, set at 350 epochs, was reached.

· ***Final results***

And here the final results on test and train:

Train loss: 0.0884921, Test loss: 0.052067
Train accuracy: 0.9694276, Test accuracy: 0.9644772
Number of iterations: 350