**H2O GLM based models**

Instructions

1. Please understand and run this code
2. Upload the output at the assignment upload section
3. The exercise will have a quiz (another coding exercise) for GLM

In this section, we cover the GLM model in H2O. <http://s3.amazonaws.com/h2o-release/h2o/rel-nunes/2/docs-website/datascience/glm.html>

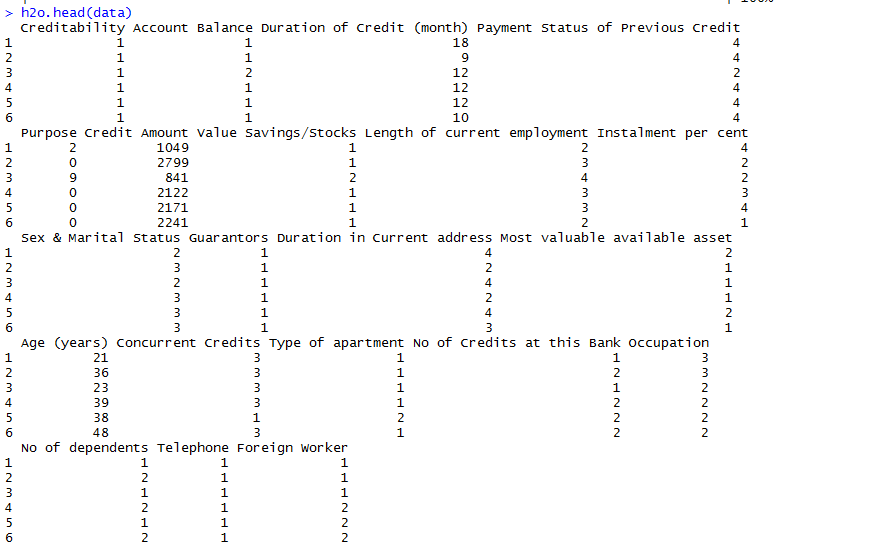
We discuss the problem of credit scoring using [German Credit data set](http://freakonometrics.free.fr/german_credit.csv). The German Credit Data contains data on 20 variables and the classification whether an applicant is considered a Good or a Bad credit risk for 1000 loan applicants.

Let's first import the data into H2O cluster using the below code snippet. You can also use  h2o.uploadFile  and  as.h2o  to load data. You can use h2o.uploadFileto upload a file from your local disk. It is a push from the client to the server and is only intended for smaller data sizes. h2o.importFile is another function which pulls information from the server from a location specified by the client. It is more suitable for big-data operations. Using h2o.importFile, you can also import files stored in HDFS as well as load an entire folder of files.

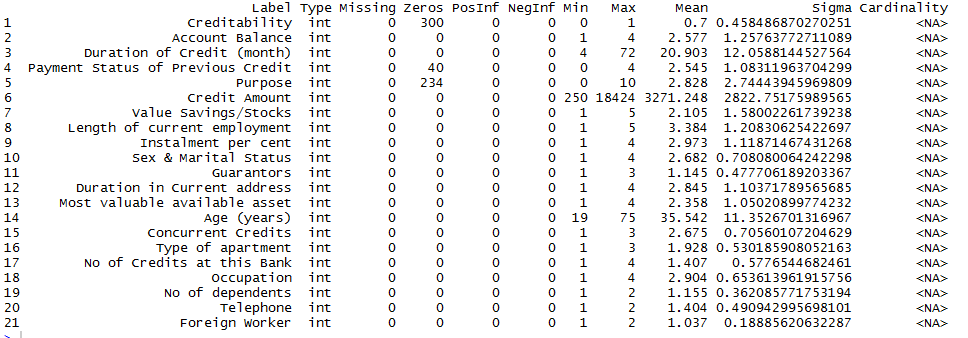
You can also use as.h2o() to import a local R data frame to the H2O. It serially uploads the file to H2O and so might be a bit slower than other file import methods.

data<-h2o.importFile("F:/git/credit\_scoring/german\_credit.csv")

Next, we use  h2o.head(data)  to view the first few rows of an H2OFrame object. It comes handy to inspect the data and get a look at how the data looks like after import. The h2o.head() is an optimized version of the R's head() to work on the H2OFrame object.



Next, we describe the aggregated information about the data using  h2o.describe(data) .The result contains the information about data types, min and max values, the missing values, count of zeros, the standard deviation of values for each column. It also displays the number of levels for the categorical/factor columns.



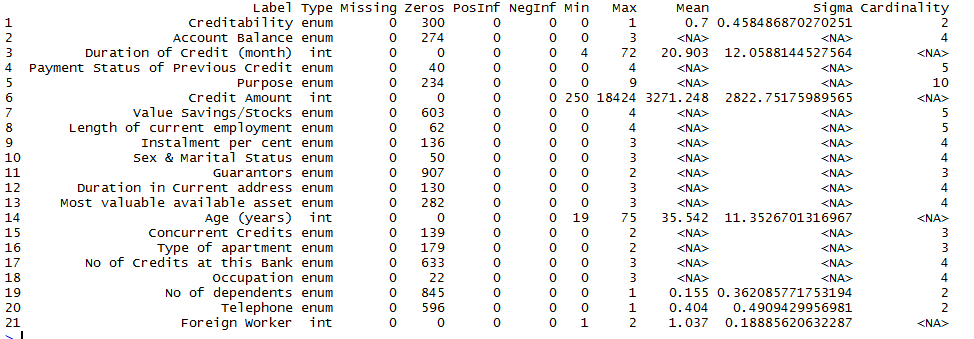
From the above result screenshot, we observe that there are some variables like **Creditability**and **Payment Status of Previous Credit** which should be of type factor/enum(in H2O). So, we type cast them using below code.  h2o.asfactor()  converts H2O Data to factors.

### Convert Numeric to Categorical ###

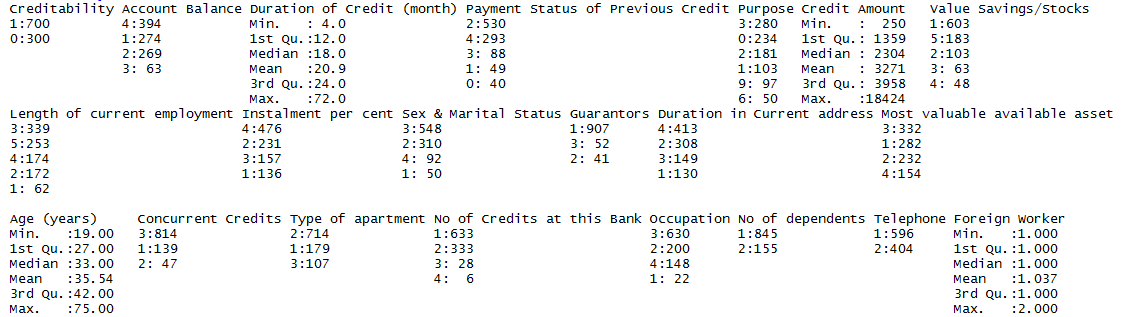
to\_factors <- c(1,2,4,5,7,8,9,10,11,12,13,15,16,17,18,19,20)

for(i in to\_factors) data[,i] <- h2o.asfactor(data[,i])

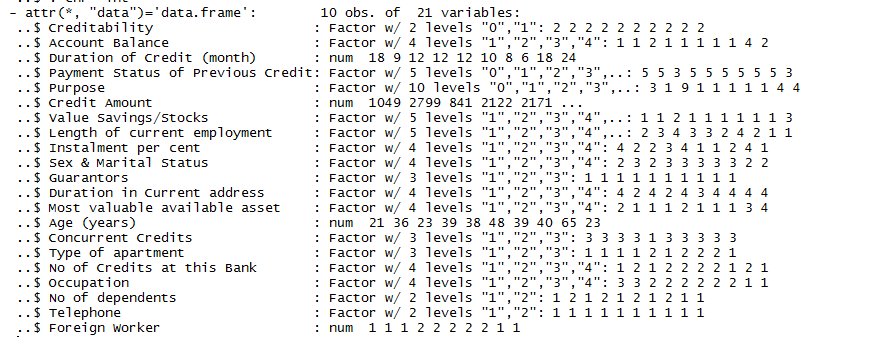
Again, we describe the data using  h2o.describe(data)  function. This time we see the data type as per our expectation and also the number of levels of each factor i.e., Cardinality available for all enum variables.



 h2o.summary(data)  is another useful summary function. It displays the minimum, 1st quartile, median, mean, 3rd quartile and maximum for each numeric column, and the levels and category counts of the levels in each categorical column.



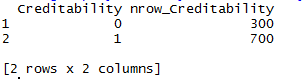
Next, we come to another handy function  h2o.str(data)  that displays the structure of an H2OFrame object.



We now check the number of data rows available to create the classification model for each class of the target variable. This also provides us with information useful to understand whether the data set is unbalanced. Modeling on unbalanced data set introduces error on the prediction and is more biased towards the class with more of records. So, generally, in such cases data is sampled again to be balanced or adjust the class weights during model creation. Not all algorithms support class weights and so we need to check algorithm definitions to find out.

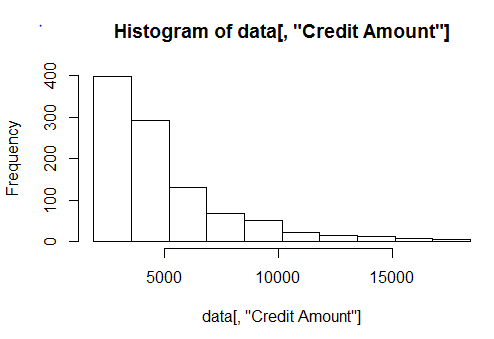
We can use  h2o.group\_by  function to count the number of rows for each target class as shown in below code.

h2o.group\_by(data, by="Creditability",nrow("Creditability"))



Another useful function from H2O is  h2o.log  which can be used for logarithmic transformations over the H2O data.

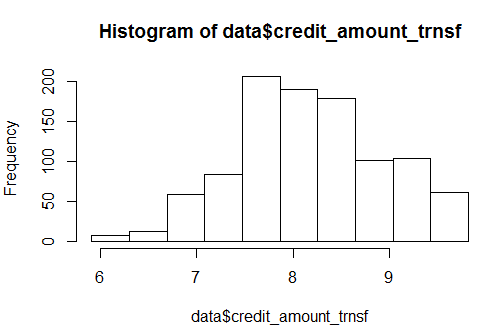
Let's plot the histogram of the Credit Column data using  h2o.hist(data[,"Credit Amount"]) . We can alternatively use positional notion to refer the column like h2o.hist(data[,6])



Apply the log function to the Credit Amount, store it in a new variable credit\_amount\_trnsf and plot the histogram.

data$credit\_amount\_trnsf <- h2o.log(data[,"Credit Amount"])

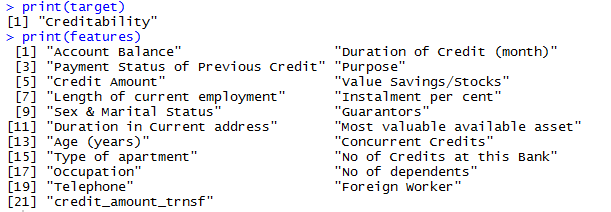
h2o.hist(data$credit\_amount\_trnsf)



After the log transformation, data looks somewhat close to normal. Let's stick to it for now. To create a supervised model such as classification, we need to annotate the target and input predictors so that it can make predictions.

For that, let us define the column **Creditability**(Good/Bad credit) as the Target for modeling.   
 target <- "Creditability"

Now that we have assigned Creditability as a Target, let us remove it from the column list and assign remaining to a new variable called features which serve as a list of input predictors.



To train and test the model, let us partition the data into training(60%) and test set(40%). Setting a seed will guarantee reproducibility.

Now that we have prepared our data, let us build a Generalized Linear Model(glm) using  h2o.glm function. We will detail about it in next articles. You can run the below code snippet to create a GLM model and refer H2O's [GLM Docs page](http://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/glm.html) for more information.

glm\_model1 <- h2o.glm(x = features,

y = target,

training\_frame = credit\_train,

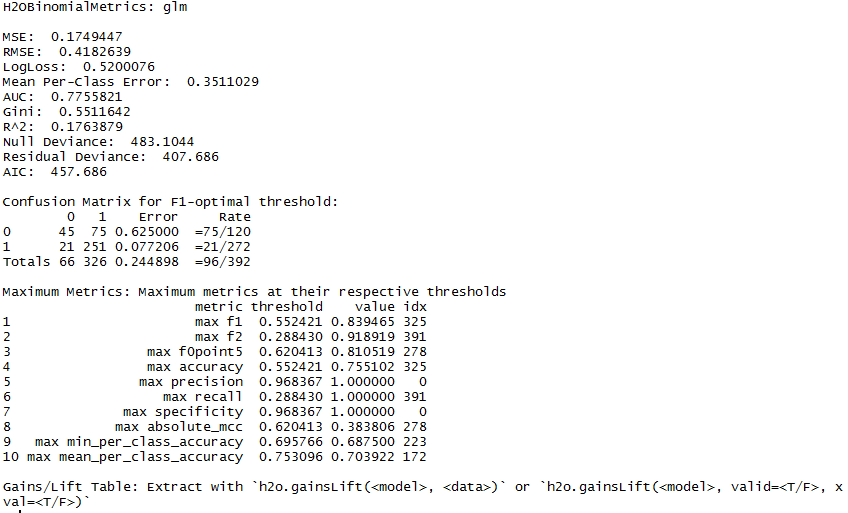
model\_id = "glm\_model1",

family = "binomial")

Once the model building is completed successfully, you can view the model build summary using print(summary(glm\_model1)) . Model build summary displays information about the fields, build settings and model estimation process.

The next important step is to evaluate whether the model can be as accurate on new data as it is in the training data. For this step, we had already held out some portion of the data as a test data. Use h2o.performance to evaluate the model performance on the hold-out test dataset. Usually to validate the model on a test data, we first do the predictions using predict methods in R and then compute the metrics using other functions. However, using h2o packages, we can just use the h2o.performance method to validate the performance of the test data. The result has all the performance measures that can be computed for that model. Track and choose the best metrics you want to evaluate your model. This is a cool feature and saves a lot of time.

perf\_obj <- h2o.performance(glm\_model1, newdata = credit\_test)



Alternatively, you can use the [H2O Model metric accessor functions](https://rdrr.io/cran/h2o/man/h2o.metric.html) to print out only the evaluation metric of your choice. The complete list of these functions is available here. For our example, let's view the accuracy of the model at 0.95 threshold.

h2o.accuracy(perf\_obj,0.944485973651485)

Use the  h2o.predict  method to do predictions on the new data. For this example, we stick to the predictions on the same test data set.

pred\_creditability <- h2o.predict(glm\_model1,credit\_test)

pred\_creditability