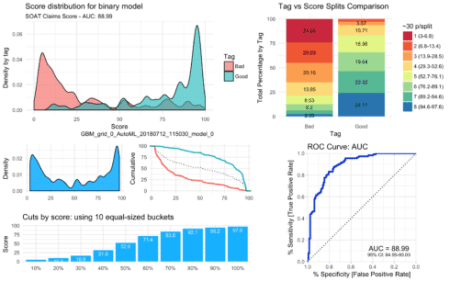
To automate the process of modeling selection and evaluate the results with visualization, I have created some functions into my personal library and today I’m sharing the codes with you. I run them to evaluate and compare Machine Learning models as fast and easily as possible. Currently, they are designed to evaluate binary classification models results. Before we start, let me show you the final outcome so you know what we are trying to achieve here with just a simple R function:  
[](https://i1.wp.com/datascienceplus.com/wp-content/uploads/2018/07/viz_full.png?ssl=1)

So, let’s start!

**The results object**

First of all, we need to have a single list with all the results to facilitate the next steps. I am assuming on this step that you already designed a model and can calculate the predictions out of your test set. So, on my list I have the following objects:

 Project name (i.e. Fraud Score)

 Model (the object with our model)

 Test Scores:

* Index (row id, it can be a user\_id, email, lead\_id…)
* Tag (known label)
* Score (calculated with the model we are studying)

 Datasets:

* Train set
* Test set

 Parameters:

* nfolds, ntrees, max\_depth, seed, sample\_rate….

 Variable importance

 Metrics:

* log\_loss
* auc

 Notes (anything you’d like to write to give you a reference later on)

Once we automate our results object, we can start with our beautiful plots!

**Density Plot**

I have always given importance to the density plot because it gives us visual information on skewness, distribution and our model’s facility to distinguish each class. Here we can see how the model has distributed both our categories, our whole test set and the cumulative of each category (the more separate, the better).

mplot\_density <- function(tag, score, model\_name = NA, subtitle = NA,

save = FALSE, file\_name = "viz\_distribution.png") {

require(ggplot2)

require(gridExtra)

if (length(tag) != length(score)) {

message("The tag and score vectors should be the same length.")

stop(message(paste("Currently, tag has",length(tag),"rows and score has",length(score))))

}

if (length(unique(tag)) != 2) {

stop("This function is for binary models. You should only have 2 unique values for the tag value!")

}

out <- data.frame(tag = as.character(tag),

score = as.numeric(score),

norm\_score = lares::normalize(as.numeric(score)))

p1 <- ggplot(out) + theme\_minimal() +

geom\_density(aes(x = 100 \* score, group = tag, fill = as.character(tag)),

alpha = 0.6, adjust = 0.25) +

guides(fill = guide\_legend(title="Tag")) +

xlim(0, 100) +

labs(title = "Score distribution for binary model",

y = "Density by tag", x = "Score")

p2 <- ggplot(out) + theme\_minimal() +

geom\_density(aes(x = 100 \* score),

alpha = 0.9, adjust = 0.25, fill = "deepskyblue") +

labs(x = "", y = "Density")

p3 <- ggplot(out) + theme\_minimal() +

geom\_line(aes(x = score \* 100, y = 100 \* (1 - ..y..), color = as.character(tag)),

stat = 'ecdf', size = 1) +

geom\_line(aes(x = score \* 100, y = 100 \* (1 - ..y..)),

stat = 'ecdf', size = 0.5, colour = "black", linetype="dotted") +

ylab('Cumulative') + xlab('') + guides(color=FALSE)

if(![is.na](http://is.na)(subtitle)) {

p1 <- p1 + labs(subtitle = subtitle)

}

if(![is.na](http://is.na)(model\_name)) {

p1 <- p1 + labs(caption = model\_name)

}

if(save == TRUE) {

png(file\_name, height = 1800, width = 2100, res = 300)

grid.arrange(

p1, p2, p3,

ncol = 2, nrow = 2, heights = 2:1,

layout\_matrix = rbind(c(1,1), c(2,3)))

dev.off()

}

return(

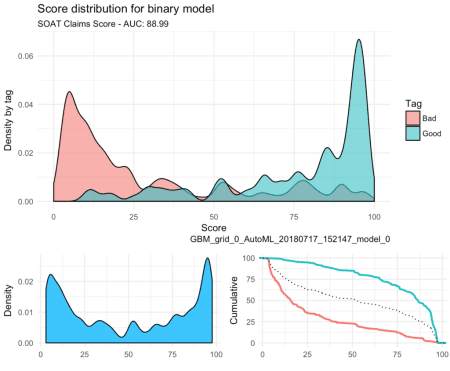
grid.arrange(

p1, p2, p3,

ncol = 2, nrow = 2, heights = 2:1,

layout\_matrix = rbind(c(1,1), c(2,3))))

}

Gives this plot:  
[](https://i2.wp.com/datascienceplus.com/wp-content/uploads/2018/07/Captura-de-pantalla-2018-07-17-a-las-4.09.35-p.-m..png?ssl=1)

**ROC Curve**

The ROC curve will give us an idea of how our model is performing with our test set. You should know by now that if the AUC is close to 50% then the model is as good as a random selector; on the other hand, if the AUC is near 100% then you have a “perfect model” (wanting or not, you must have been giving the model the answer this whole time!). So it is always good to check this plot and check that we are getting a reasonable Area Under the Curve with a nice and closed 95% confidence range.

# ROC Curve

mplot\_roc <- function(tag, score, model\_name = NA, subtitle = NA, interval = 0.2, plotly = FALSE,

save = FALSE, file\_name = "viz\_roc.png") {

require(pROC)

require(ggplot2)

if (length(tag) != length(score)) {

message("The tag and score vectors should be the same length.")

stop(message(paste("Currently, tag has",length(tag),"rows and score has",length(score))))

}

roc <- pROC::roc(tag, score, ci=T)

coords <- data.frame(

x = rev(roc$specificities),

y = rev(roc$sensitivities))

ci <- data.frame(roc$ci, row.names = c("min","AUC","max"))

p <- ggplot(coords, aes(x = x, y = y)) +

geom\_line(colour = "deepskyblue", size = 1) +

geom\_point(colour = "blue3", size = 0.9, alpha = 0.4) +

geom\_segment(aes(x = 0, y = 1, xend = 1, yend = 0), alpha = 0.2, linetype = "dotted") +

scale\_x\_reverse(name = "% Specificity [False Positive Rate]", limits = c(1,0),

breaks = seq(0, 1, interval), expand = c(0.001,0.001)) +

scale\_y\_continuous(name = "% Sensitivity [True Positive Rate]", limits = c(0,1),

breaks = seq(0, 1, interval), expand = c(0.001, 0.001)) +

theme\_minimal() +

theme(axis.ticks = element\_line(color = "grey80")) +

coord\_equal() +

ggtitle("ROC Curve: AUC") +

annotate("text", x = 0.25, y = 0.10, vjust = 0, size = 4.2,

label = paste("AUC =", round(100\*ci[c("AUC"),],2))) +

annotate("text", x = 0.25, y = 0.05, vjust = 0, size = 2.8,

label = paste0("95% CI: ",

round(100\*ci[c("min"),],2),"-",

round(100\*ci[c("max"),],2)))

if(![is.na](http://is.na)(subtitle)) {

p <- p + labs(subtitle = subtitle)

}

if(![is.na](http://is.na)(model\_name)) {

p <- p + labs(caption = model\_name)

}

if (plotly == TRUE) {

require(plotly)

p <- ggplotly(p)

}

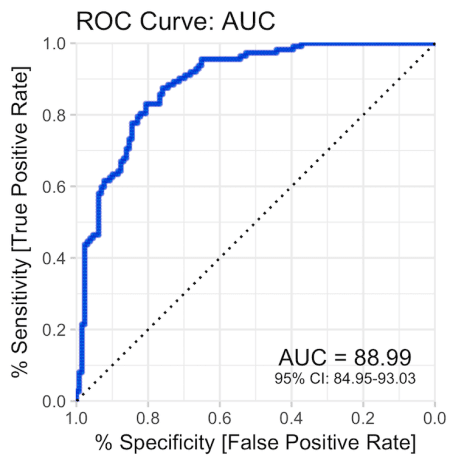
if (save == TRUE) {

p <- p + ggsave(file\_name, width = 6, height = 6)

}

return(p)

}

Gives this plot:  
[](https://i1.wp.com/datascienceplus.com/wp-content/uploads/2018/07/Captura-de-pantalla-2018-07-17-a-las-4.10.41-p.-m..png?ssl=1)

**Cuts by quantile**

If we’d have to cut the score in n equal-sized buckets, what would the score cuts be? Is the result a ladder (as it should), or a huge wall, or a valley? Is our score distribution lineal and easy to split?

mplot\_cuts <- function(score, splits = 10, subtitle = NA, model\_name = NA,

save = FALSE, file\_name = "viz\_ncuts.png") {

require(ggplot2)

if (splits > 25) {

stop("You should try with less splits!")

}

deciles <- quantile(score,

probs = seq((1/splits), 1, length = splits),

names = TRUE)

deciles <- data.frame(cbind(Deciles=row.names(as.data.frame(deciles)),

Threshold=as.data.frame(deciles)))

p <- ggplot(deciles,

aes(x = reorder(Deciles, deciles), y = deciles \* 100,

label = round(100 \* deciles, 1))) +

geom\_col(fill="deepskyblue") +

xlab('') + theme\_minimal() + ylab('Score') +

geom\_text(vjust = 1.5, size = 3, inherit.aes = TRUE, colour = "white", check\_overlap = TRUE) +

labs(title = paste("Cuts by score: using", splits, "equal-sized buckets"))

if(![is.na](http://is.na)(subtitle)) {

p <- p + labs(subtitle = subtitle)

}

if(![is.na](http://is.na)(model\_name)) {

p <- p + labs(caption = model\_name)

}

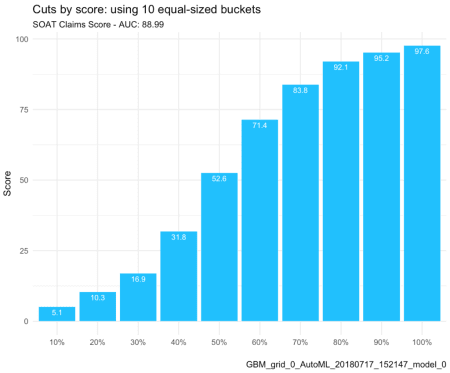
if (save == TRUE) {

p <- p + ggsave(file\_name, width = 6, height = 6)

}

return(p)

}

Gives this plot:  
[](https://i1.wp.com/datascienceplus.com/wp-content/uploads/2018/07/Captura-de-pantalla-2018-07-17-a-las-4.12.22-p.-m..png?ssl=1)

**Split and compare quantiles**

This parameter is the easiest to sell to the C-level guys. “Did you know that with this model, if we chop the worst 20% of leads we would have avoided 60% of the frauds and only lose 8% of our sales?” That’s what this plot will give you:

mplot\_splits <- function(tag, score, splits = 5, subtitle = NA, model\_name = NA, facet = NA,

save = FALSE, file\_name = "viz\_splits.png") {

require(ggplot2)

require(dplyr)

require(RColorBrewer)

if (length(tag) != length(score)) {

message("The tag and score vectors should be the same length.")

stop(message(paste("Currently, tag has",length(tag),"rows and score has",length(score))))

}

if (splits > 10) {

stop("You should try with less splits!")

}

df <- data.frame(tag, score, facet)

npersplit <- round(nrow(df)/splits)

names %

mutate(quantile = ntile(score, splits)) %>% group\_by(quantile) %>%

summarise(n = n(),

max\_score = round(100 \* max(score), 1),

min\_score = round(100 \* min(score), 1)) %>%

mutate(quantile\_tag = paste0(quantile," (",min\_score,"-",max\_score,")"))

p %

mutate(quantile = ntile(score, splits)) %>%

group\_by(quantile, facet, tag) %>% tally() %>%

ungroup() %>% group\_by(facet, tag) %>%

arrange(desc(quantile)) %>%

mutate(p = round(100\*n/sum(n),2),

cum = cumsum(100\*n/sum(n))) %>%

left\_join(names, by = c("quantile")) %>%

ggplot(aes(x = as.character(tag), y = p, label = as.character(p),

fill = as.character(quantile\_tag))) + theme\_minimal() +

geom\_col(position = "stack") +

geom\_text(size = 3, position = position\_stack(vjust = 0.5), check\_overlap = TRUE) +

xlab("Tag") + ylab("Total Percentage by Tag") +

guides(fill = guide\_legend(title=paste0("~",npersplit," p/split"))) +

labs(title = "Tag vs Score Splits Comparison") +

scale\_fill\_brewer(palette = "Spectral")

if(![is.na](http://is.na)(subtitle)) {

p <- p + labs(subtitle = subtitle)

}

if(![is.na](http://is.na)(model\_name)) {

p <- p + labs(caption = model\_name)

}

if(![is.na](http://is.na)(facet)) {

p <- p + facet\_grid(. ~ facet, scales = "free")

}

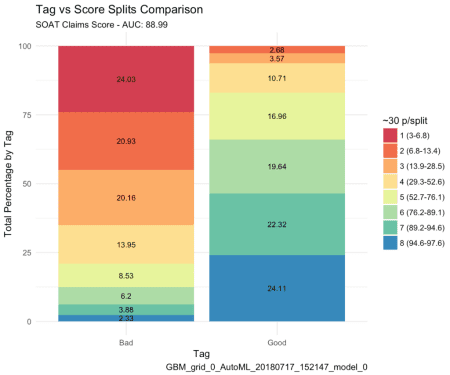
if (save == TRUE) {

p <- p + ggsave(file\_name, width = 6, height = 6)

}

return(p)

}

Gives this plot:  
[](https://i2.wp.com/datascienceplus.com/wp-content/uploads/2018/07/Captura-de-pantalla-2018-07-17-a-las-4.13.23-p.-m..png?ssl=1)

**Finally, let’s plot our results**

Once we have defined these functions above, we can create a new one that will bring everything together into one single plot. If you pay attention to the variables needed to create this dashboard you would notice it actually only needs two: the label or tag, and the score. You can customize the splits for the upper right plot, set a subtitle, define the model’s name, save it in a new folder, change the image’s name.

mplot\_full <- function(tag, score, splits = 8, subtitle = NA, model\_name = NA,

save = FALSE, file\_name = "viz\_full.png", subdir = NA) {

require(ggplot2)

require(gridExtra)

options(warn=-1)

if (length(tag) != length(score)) {

message("The tag and score vectors should be the same length.")

stop(message(paste("Currently, tag has",length(tag),"rows and score has",length(score))))

}

p1 <- mplot\_density(tag = tag, score = score, subtitle = subtitle, model\_name = model\_name)

p2 <- mplot\_splits(tag = tag, score = score, splits = splits)

p3 <- mplot\_roc(tag = tag, score = score)

p4 <- mplot\_cuts(score = score)

if(save == TRUE) {

if (![is.na](http://is.na)(subdir)) {

dir.create(file.path(getwd(), subdir))

file\_name <- paste(subdir, file\_name, sep="/")

}

png(file\_name, height = 2000, width = 3200, res = 300)

grid.arrange(

p1, p2, p3, p4,

widths = c(1.3,1),

layout\_matrix = rbind(c(1,2), c(1,2), c(1,3), c(4,3)))

dev.off()

}

return(

grid.arrange(

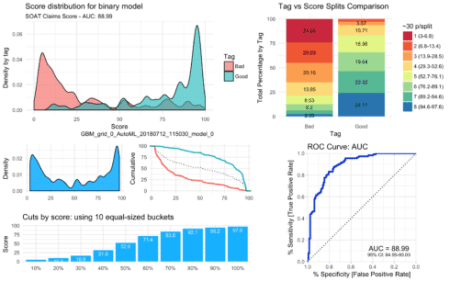
p1, p2, p3, p4,

widths = c(1.3,1),

layout\_matrix = rbind(c(1,2), c(1,2), c(1,3), c(4,3)))

)

}

That’s it. This dashboard will give us almost everything we need to visually evaluate our model’s performance into the test set.  
[](https://i1.wp.com/datascienceplus.com/wp-content/uploads/2018/07/viz_full.png?ssl=1)

One bonus tip for these plots: you can set the subtitle and subdirectory before you plot everything so you don’t have to change it whenever you are trying a new model.

subtitle <- paste(results$project, "- AUC:", round(100 \* results$auc\_test, 2))

subdir <- paste0("Models/", round(100\*results$auc\_test, 2), "-", results$model\_name)

**Bonus: Variables Importance**

If you are working with a ML algorithm that let’s you see the importance of each variable, you can use the following function to see the results:

mplot\_importance <- function(var, imp, colours = NA, limit = 15, model\_name = NA, subtitle = NA,

save = FALSE, file\_name = "viz\_importance.png", subdir = NA) {

require(ggplot2)

require(gridExtra)

options(warn=-1)

if (length(var) != length(imp)) {

message("The variables and importance values vectors should be the same length.")

stop(message(paste("Currently, there are",length(var),"variables and",length(imp),"importance values!")))

}

if ([is.na](http://is.na)(colours)) {

colours <- "deepskyblue"

}

out <- data.frame(var = var, imp = imp, Type = colours)

if (length(var) < limit) {

limit <- length(var)

}

output <- out[1:limit,]

p <- ggplot(output,

aes(x = reorder(var, imp), y = imp \* 100,

label = round(100 \* imp, 1))) +

geom\_col(aes(fill = Type), width = 0.1) +

geom\_point(aes(colour = Type), size = 6) +

coord\_flip() + xlab('') + theme\_minimal() +

ylab('Importance') +

geom\_text(hjust = 0.5, size = 2, inherit.aes = TRUE, colour = "white") +

labs(title = paste0("Variables Importances. (", limit, " / ", length(var), " plotted)"))

if (length(unique(output$Type)) == 1) {

p <- p + geom\_col(fill = colours, width = 0.2) +

geom\_point(colour = colours, size = 6) +

guides(fill = FALSE, colour = FALSE) +

geom\_text(hjust = 0.5, size = 2, inherit.aes = TRUE, colour = "white")

}

if(![is.na](http://is.na)(model\_name)) {

p <- p + labs(caption = model\_name)

}

if(![is.na](http://is.na)(subtitle)) {

p <- p + labs(subtitle = subtitle)

}

if(save == TRUE) {

if (![is.na](http://is.na)(subdir)) {

dir.create(file.path(getwd(), subdir))

file\_name <- paste(subdir, file\_name, sep="/")

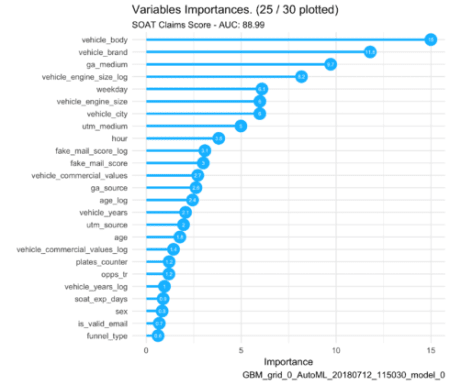
}

p <- p + ggsave(file\_name, width=7, height=6)

}

return(p)

}

Gives this plot:  
[](https://i0.wp.com/datascienceplus.com/wp-content/uploads/2018/07/viz_importance.png?ssl=1)