# Evaluación MLII temas 1, 2, 6 y 7: Ejercicio 3

Clasificación no balanceada

*Inmaculada Perea Fernández*

*mayo 2017*

Completar el tratamiento de los datos de Insolvencia mediante técnicas apropiadas para Clasificación No Balanceada (datos en el material de dicho tema).

## 1. Carga e instalación de librerías necesarias

```
if (!require('caret')) install.packages('caret'); library('caret')
if (!require('pROC')) install.packages('pROC'); library('pROC')
if (!require('DMwR')) install.packages('DMwR'); library('DMwR')
```

## 2. Carga, inspección y preparación de los datos

### 2.1 Carga de datos

```
load("Insolvencia.RData")
dim(datos)
```

```
## [1] 2877   16
```

```
str(datos)
```

```
## 'data.frame':    2877 obs. of  16 variables:
##  $ CETL            : num  0.645 0.758 0 0.281 0 ...
##  $ STLTA           : num  0.8 0.234 0 0 0 ...
##  $ TLCA            : num  31 0.767 1 0.78 1.333 ...
##  $ NWTA            : num  -0.476 0.298 0 0.22 0 ...
##  $ QACA            : num  1 0.977 1 0.293 0.583 ...
##  $ NCNW            : num  0 1.57 0 1.33 12 ...
##  $ CRATIO          : num  1 1.95 1 1.28 0.75 ...
##  $ CASHTA          : num  0.0476 0.4681 0.5556 0.2927 0 ...
##  $ PRTA            : num  -0.476 0.298 0 0.22 0 ...
##  $ TCTD            : num  57 0.667 1 16 1 ...
##  $ TCTL            : num  0.0323 0.6667 1 0.0625 1 ...
##  $ TDTA            : num  0 0.426 0.444 0 0.438 ...
##  $ ln_assets       : num  9.95 10.76 9.1 11.31 10.37 ...
##  $ CHNW_new        : num  0 1.333 -7.3 0.636 -1 ...
##  $ CHNWTA_new      : num  0.0476 0.3901 -10.6 0.6763 -1 ...
##  $ failed_insolvent: Factor w/ 2 levels "No","Yes": 1 2 1 1 1 2 1 1 1 1 ...
##  - attr(*, "na.action")=Class 'omit'  Named int [1:123] 5 15 66 75 85 89 98 105 113 195 ...
##   .. ..- attr(*, "names")= chr [1:123] "4326488" "4392417" "4386613" "4323350" ...
```

```
summary(datos)
```

```
##       CETL             STLTA             TLCA
##  Min.   :-0.9480  Min.   :0.00000  Min.   :  0.0000
##  1st Qu.: 0.0000  1st Qu.:0.00000  1st Qu.:  0.5454
##  Median : 0.3333  Median :0.00000  Median :  0.9873
##  Mean   : 2.2197  Mean   :0.06748  Mean   :  2.7228
##  3rd Qu.: 1.2353  3rd Qu.:0.00000  3rd Qu.:  1.6046
##  Max.   :80.5000  Max.   :0.80000  Max.   :159.0000
##       NWTA               QACA            NCNW            CRATIO
##  Min.   :-34.50000  Min.   :0.0000  Min.   :-4.700  Min.   : 0.0000
##  1st Qu.: -0.03448  1st Qu.:0.9231  1st Qu.: 0.000  1st Qu.: 0.7143
##  Median :  0.13750  Median :1.0000  Median : 0.750  Median : 1.0814
##  Mean   : -0.03672  Mean   :0.8758  Mean   : 3.318  Mean   : 2.6059
##  3rd Qu.:  0.52863  3rd Qu.:1.0000  3rd Qu.: 2.800  3rd Qu.: 1.9714
##  Max.   :  1.00000  Max.   :1.0000  Max.   :34.000  Max.   :51.0000
##      CASHTA              PRTA                TCTD
##  Min.   :0.00000  Min.   :-18.250000  Min.   :  0.0000
##  1st Qu.:0.01333  1st Qu.:  0.000000  1st Qu.:  0.9643
##  Median :0.21569  Median :  0.131783  Median :  1.0000
##  Mean   :0.34290  Mean   :  0.008251  Mean   : 12.1243
##  3rd Qu.:0.63636  3rd Qu.:  0.500000  3rd Qu.:  1.0000
##  Max.   :1.00000  Max.   :  1.000000  Max.   :185.0000
##       TCTL             TDTA           ln_assets         CHNW_new
##  Min.   :0.0000  Min.   :0.00000  Min.   : 8.987  Min.   :-7.3000
##  1st Qu.:0.9167  1st Qu.:0.02857  1st Qu.: 9.680  1st Qu.:-0.2857
##  Median :1.0000  Median :0.23256  Median :10.491  Median : 0.0000
##  Mean   :0.8457  Mean   :0.33327  Mean   :10.538  Mean   : 0.1763
##  3rd Qu.:1.0000  3rd Qu.:0.57895  3rd Qu.:11.350  3rd Qu.: 0.3077
##  Max.   :1.0000  Max.   :1.00000  Max.   :12.333  Max.   :18.8000
##    CHNWTA_new       failed_insolvent
##  Min.   :-66.5000   No :2740
##  1st Qu.: -0.2889   Yes: 137
##  Median :  0.0000
##  Mean   : -0.2265
##  3rd Qu.:  0.2533
##  Max.   : 11.9000
```

La variable dependiente es *failed_insolvent* (16) factor con dos niveles relativos a la insolvencia de empresas (*No* y *Yes*)

## 2.2 Inspección del número de casos disponible para cada clase

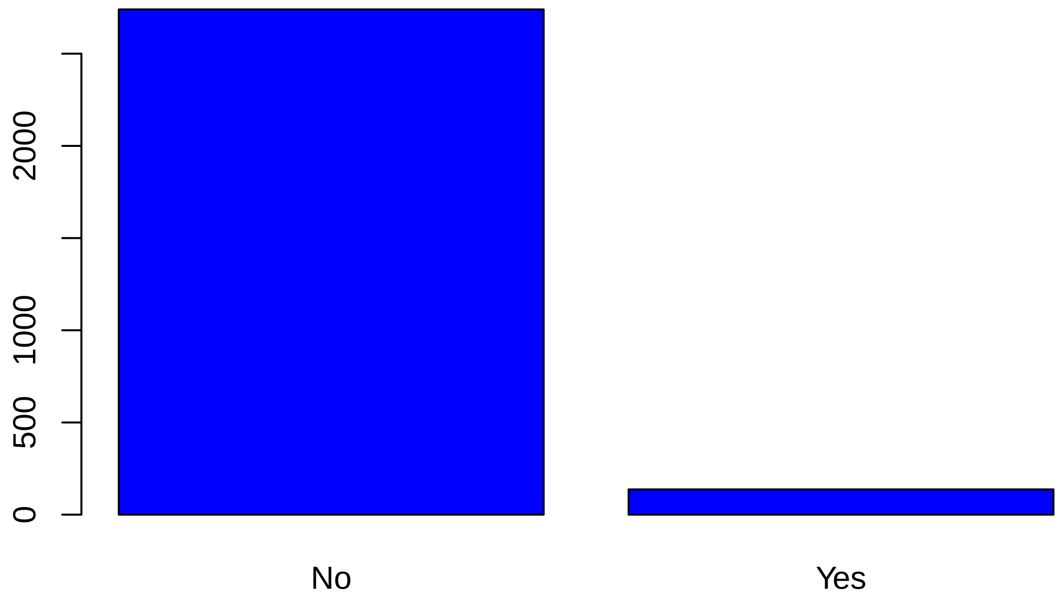
```r
table(datos$failed_insolvent)
```

```
##
##   No  Yes
## 2740  137
```

```r
prop.table(table(datos$failed_insolvent))
```

```
##
##         No        Yes
## 0.95238095 0.04761905
```

```r
barplot(table(datos$failed_insolvent), col = "blue")
```




Se observa que los datos están no balanceados, porque la clase *Yes* de la variable respuesta se presenta en el conjunto de entrenamiento en proporciones muy inferiores a la de la categoría *No*. En concreto un 95% son *No*, frente a un 0.05% que son *Yes*

A continuación realizaremos transformaciones para que la primera clase corresponda a la clase minoritaria (*Yes*) y de este modo poder usar Sens en los ajustes, la sensitividad se referiará a ella.

```
datos$failed_insolvent = factor(as.character(datos$failed_insolvent),
                                levels = rev(levels(datos$failed_insolvent)))
table(datos$failed_insolvent)
```

```
##
##  Yes   No
##  137 2740
```

## 2.3 División en entrenamiento, test y validación

Se va a dividir el cojunto de datos en 3 partes:

- Entrenamiento (60%)
- Validación (15%)
- Test (25%)

El subconjunto de validación solo se utilizará para configurar el punto de corte (una de las estrategias para datos no balanceados), por eso y porque solo hay 2877 casos solo se va a reservar un 15% para el conjunto de validación.

```r
set.seed(271)
n=nrow(datos)
indices=1:n
ient=sample(indices,floor(n*0.6))
ival=sample(setdiff(indices,ient),floor(n*0.15))
itest=setdiff(indices,union(ient,ival))

training  = datos[ient,]
validation  = datos[ival,]
testing     = datos[itest,]
training_valid=rbind(training, validation)

dim(training)
```

```
## [1] 1726   16
```

```r
dim(validation)
```

```
## [1] 431  16
```

```r
dim(testing)
```

```
## [1] 720  16
```

```r
dim(training_valid)
```

```
## [1] 2157   16
```

## 2.4 Variables, funciones y configuración auxiliar

Variable *Index* para usarlo con *trainControl*

```r
Index= 1:nrow(training)
```

Obtenemos los nombres de las variables predictoras en la variable *predictors*

```r
predictors = names(training)[names(training) != "failed_insolvent"]
predictors
```

```
##  [1] "CETL"       "STLTA"      "TLCA"       "NWTA"       "QACA"
##  [6] "NCNW"       "CRATIO"     "CASHTA"     "PRTA"       "TCTD"
## [11] "TCTL"       "TDTA"       "ln_assets"  "CHNW_new"   "CHNWTA_new"
```

En los objetos *testResults* y *validResults* se van a guardar las predicciones del conjunto test y validación respectivamente

```r
testResults = data.frame(failed_insolvent = testing$failed_insolvent)
validResults = data.frame(failed_insolvent = validation$failed_insolvent)
```

## 2.5 Funciones para medir el rendimiento

La función *fiveStats* devuelve las medidas de *twoClassSummary* y *defaultSummary* (Accuracy, Kappa, AUC ROC, Sensitivity y Specificity). La función *fourStats* devuelve todo lo aterior menos AUC

```r
fiveStats = function(...)
  c(twoClassSummary(...), defaultSummary(...))
```

```
fourStats = function (data, lev = levels(data$obs),
                      model = NULL)
{

  accKapp = postResample(data[, "pred"], data[, "obs"])
  out = c(accKapp,
          sensitivity(data[, "pred"], data[, "obs"], lev[1]),
          specificity(data[, "pred"], data[, "obs"], lev[2]))
  names(out)[3:4] = c("Sens", "Spec")
  out
}
```

Opciones de control para el entrenamiento mediante el paquete *caret*. Se usará validación cruzada ya que el conjunto de validación es muy reducido. Notar que se utilizan 3 pliegues y no 10 porque el conjunto de validación es reducido y no es necesario 10 pliegues, de este modo conseguimos que tarde menos en calcular.

```
ctrlcv = trainControl(method = "cv",
                      number=3,      # número de pliegues
                      classProbs = TRUE,
                      summaryFunction = fiveStats,
                      verboseIter=TRUE)
```

# 3. Ajuste de dos modelos: RF y regresión logística

## 3.1 Random Forest

Con *tuneLength* = total de valores de *mtry* a explorar. Como tarda algo de tiempo, tomamos *ntree* = 100

```
rfFit = train(failed_insolvent ~ .,
              data = training,
              method = "rf",
              trControl = ctrlcv,
              ntree = 100,
              do.trace=TRUE,
              tuneLength=3,
              metric = "Sens", #Sensitividad
              trace= FALSE)
```

```
rfFit
```

```
## Random Forest
##
## 1726 samples
##   15 predictor
##    2 classes: 'Yes', 'No'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold)
## Summary of sample sizes: 1150, 1151, 1151
## Resampling results across tuning parameters:
##
##   mtry  ROC        Sens        Spec       Accuracy   Kappa
##    2    0.7422538  0.00000000  1.0000000  0.9472776  0.00000000
```

```
##    8    0.7552326  0.02222222  0.9975535  0.9461202  0.03537502
##   15    0.7412525  0.02222222  0.9957187  0.9443810  0.03088483
##
## Sens was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 8.
```

```r
rfFit$results #Cada medida, con su desv. tip.(NA EN ESTE CASO)
```

```
##   mtry       ROC       Sens      Spec  Accuracy      Kappa      ROCSD
## 1    2 0.7422538 0.00000000 1.0000000 0.9472776 0.00000000 0.03251892
## 2    8 0.7552326 0.02222222 0.9975535 0.9461202 0.03537502 0.03179783
## 3   15 0.7412525 0.02222222 0.9957187 0.9443810 0.03088483 0.02315780
##       SensSD      SpecSD  AccuracySD    KappaSD
## 1 0.00000000 0.000000000 0.000950048 0.00000000
## 2 0.01924501 0.001059358 0.002954736 0.03632234
## 3 0.01924501 0.002802799 0.002983915 0.03312567
```

Probabilidades estimadas de la categoría *Yes*

```r
validResults$RF = predict(rfFit, validation,
                          type = "prob")[,1]
testResults$RF = predict(rfFit, testing,
                         type = "prob")[,1]
```

Vamos a calcular las medidas de rendimiento en el conjunto test

```r
rfTestROC = roc(testResults$failed_insolvent, testResults$RF,
                levels = rev(levels(testResults$failed_insolvent)))
rfTestROC
```

```
##
## Call:
## roc.default(response = testResults$failed_insolvent, predictor = testResults$RF,     levels = rev(le
##
## Data: testResults$RF in 689 controls (testResults$failed_insolvent No) < 31 cases (testResults$faile
## Area under the curve: 0.6917
```

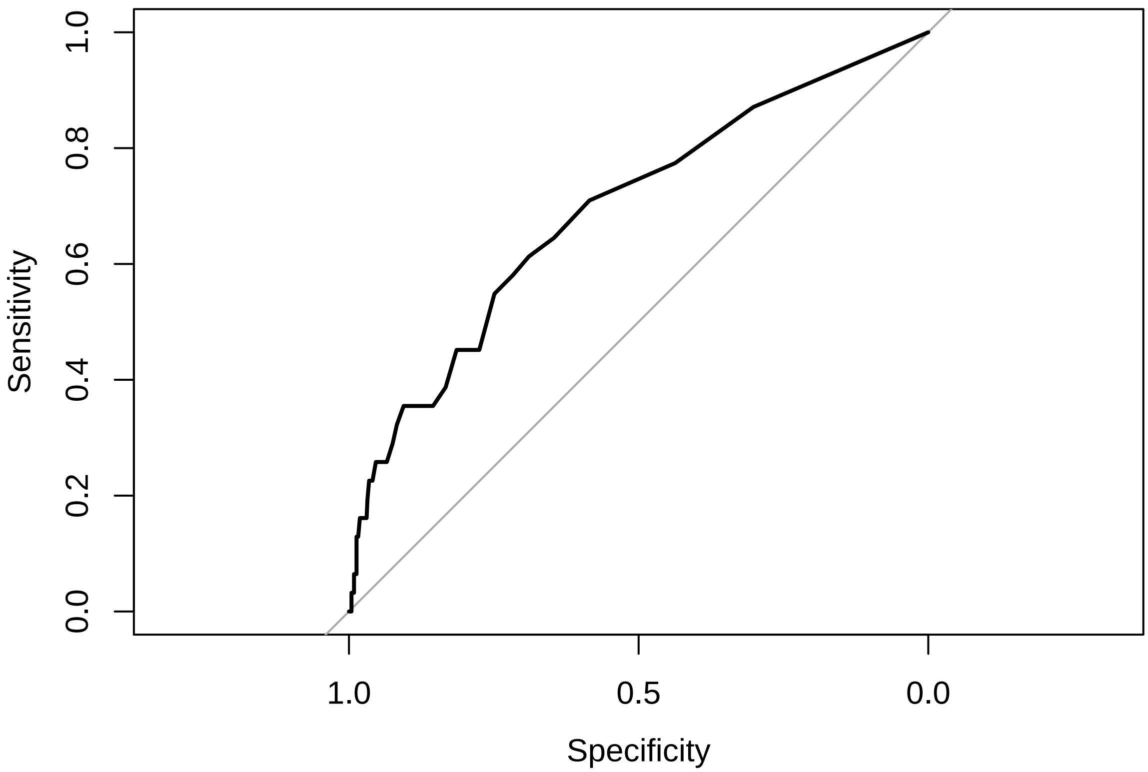
```r
rfTestCM = confusionMatrix(predict(rfFit, testing),
                           testResults$failed_insolvent)
rfTestCM
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Yes  No
##        Yes   0   3
##        No   31 686
##
##                Accuracy : 0.9528
##                  95% CI : (0.9346, 0.9671)
##     No Information Rate : 0.9569
##     P-Value [Acc > NIR] : 0.7451
##
##                   Kappa : -0.0077
##  Mcnemar's Test P-Value : 3.649e-06
##
##             Sensitivity : 0.000000
##             Specificity : 0.995646
```

```
##          Pos Pred Value : 0.000000
##          Neg Pred Value : 0.956764
##              Prevalence : 0.043056
##          Detection Rate : 0.000000
##    Detection Prevalence : 0.004167
##       Balanced Accuracy : 0.497823
##
##        'Positive' Class : Yes
##
```

Obtenemos una especificidad alta (0.96), pero una sensitividad muy baja (0) por lo que el modelo no es bueno para clasificar la clase minoritaria *Yes*.

```r
plot(rfTestROC)
```




### 3.2 Modelo de regresión logística

En este modelo no hay parámetros que configurar, aplicaremos directamente el modelo a los datos de entrenamiento

```r
ctrlrlog = trainControl(method = "none",
                        classProbs = TRUE,
                        summaryFunction = fiveStats)



lrFit = train(failed_insolvent ~ .,
              data = training,
```

```r
              method = "glm",
              trControl = ctrlrlog)
lrFit
```

```
## Generalized Linear Model
##
## 1726 samples
##   15 predictor
##    2 classes: 'Yes', 'No'
##
## No pre-processing
## Resampling: None
```

```r
summary(lrFit)
```

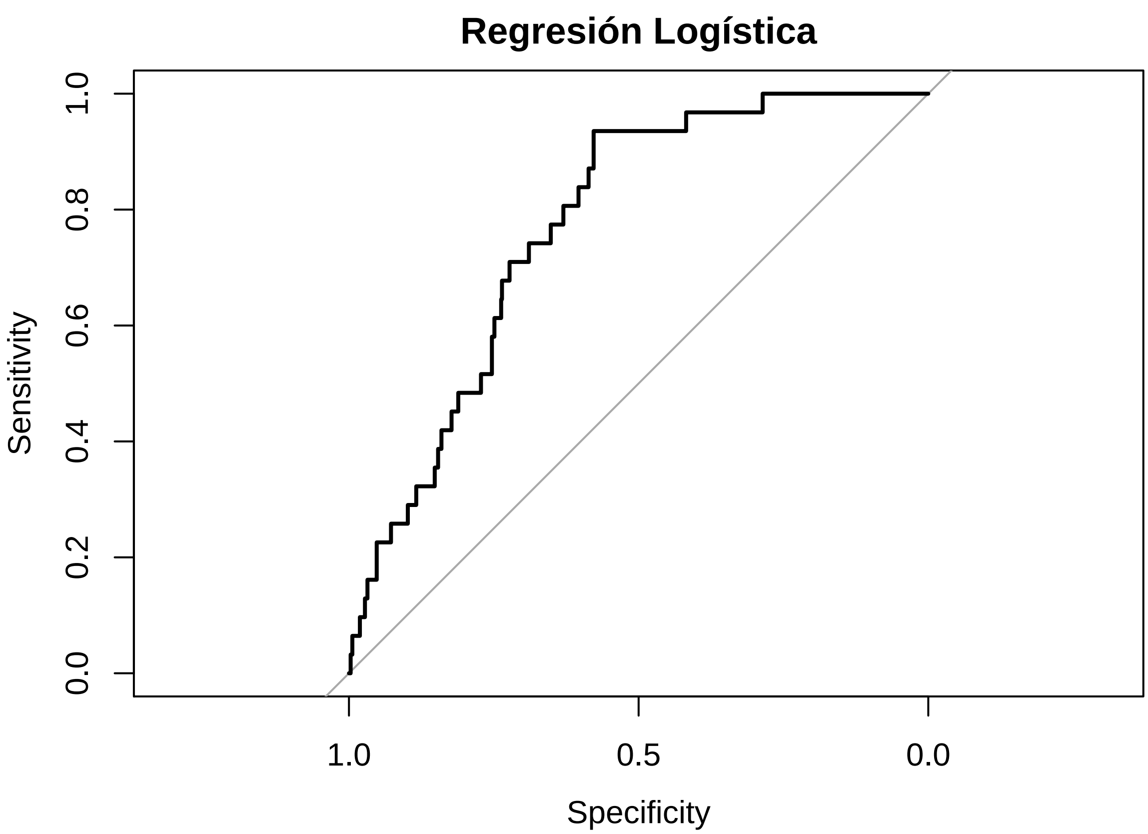
```
##
## Call:
## NULL
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.3482   0.1232   0.2249   0.3547   1.5429
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) 10.776852   1.697475   6.349 2.17e-10 ***
## CETL         0.124379   0.079126   1.572  0.11597
## STLTA       -0.809959   0.723133  -1.120  0.26268
## TLCA         0.101578   0.054844   1.852  0.06400 .
## NWTA         0.058674   0.109681   0.535  0.59269
## QACA        -0.850929   0.634945  -1.340  0.18019
## NCNW        -0.016428   0.016297  -1.008  0.31345
## CRATIO      -0.025089   0.058032  -0.432  0.66551
## CASHTA       2.066667   0.730118   2.831  0.00465 **
## PRTA         0.439689   0.126125   3.486  0.00049 ***
## TCTD         0.003717   0.007291   0.510  0.61022
## TCTL         0.067798   0.589633   0.115  0.90846
## TDTA        -0.308486   0.591633  -0.521  0.60208
## ln_assets   -0.707362   0.137794  -5.133 2.84e-07 ***
## CHNW_new     0.102038   0.093978   1.086  0.27758
## CHNWTA_new  -0.096063   0.084473  -1.137  0.25545
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 712.69  on 1725  degrees of freedom
## Residual deviance: 599.32  on 1710  degrees of freedom
## AIC: 631.32
##
## Number of Fisher Scoring iterations: 8
```

Probabilidades estimadas para la clase *Yes*

```
validResults$LogReg = predict(lrFit,
                              validation,
                              type = "prob")[,1]

testResults$LogReg = predict(lrFit,
                             testing,
                             type = "prob")[,1]

lrTestROC = roc(testResults$failed_insolvent, testResults$LogReg,
                levels = rev(levels(testResults$failed_insolvent)))
lrTestROC
```

```
##
## Call:
## roc.default(response = testResults$failed_insolvent, predictor = testResults$LogReg,     levels = re
##
## Data: testResults$LogReg in 689 controls (testResults$failed_insolvent No) < 31 cases (testResults$fa
## Area under the curve: 0.7724
```

```
plot(lrTestROC, main="Regresión Logística")
```




```
lrTestCM = confusionMatrix(predict(lrFit, testing),
                           testResults$failed_insolvent)
lrTestCM
```

```
## Confusion Matrix and Statistics
##
```

```
##            Reference
## Prediction Yes  No
##        Yes   1   2
##        No   30 687
##
##                Accuracy : 0.9556
##                  95% CI : (0.9378, 0.9694)
##     No Information Rate : 0.9569
##     P-Value [Acc > NIR] : 0.6183
##
##                   Kappa : 0.0516
##  Mcnemar's Test P-Value : 1.815e-06
##
##             Sensitivity : 0.032258
##             Specificity : 0.997097
##          Pos Pred Value : 0.333333
##          Neg Pred Value : 0.958159
##              Prevalence : 0.043056
##          Detection Rate : 0.001389
##    Detection Prevalence : 0.004167
##       Balanced Accuracy : 0.514678
##
##        'Positive' Class : Yes
##
```

Obtenemos de nuevo una sensitividad muy baja, el modelo no clasifica bien la clase *Yes*. Esto pone de manifiesto que los datos no están balanceados y los modelos aplicados directamente sobre ellos no dan buenos resultados.

## 3.3. Curvas COR y LIFT en el conjunto test

```r
labs = c(RF = "Random Forest", LogReg = "Reg.Log.")

lift1 = lift(failed_insolvent ~ RF + LogReg , data = testResults,
             labels = labs)
str(lift1)
```

```
## List of 5
##  $ data     :'data.frame':   764 obs. of  11 variables:
##   ..$ liftModelVar: Factor w/ 2 levels "Random Forest",..: 1 1 1 1 1 1 1 1 1 1 ...
##   ..$ cuts        : num [1:764] 1 0.61 0.55 0.53 0.45 0.44 0.42 0.4 0.39 0.37 ...
##   ..$ events      : int [1:764] 0 0 0 0 2 2 2 4 4 4 ...
##   ..$ n           : int [1:764] 0 2 4 6 8 12 14 16 18 20 ...
##   ..$ Sn          : num [1:764] 0 0 0 0 0.0323 ...
##   ..$ Sp          : num [1:764] 1 0.999 0.997 0.996 0.996 ...
##   ..$ EventPct    : num [1:764] 0 0 0 0 25 ...
##   ..$ CumEventPct : num [1:764] 0 0 0 0 3.23 ...
##   ..$ lift        : num [1:764] NaN 0 0 0 5.81 ...
##   ..$ CumTestedPct: num [1:764] 0 0.139 0.278 0.417 0.556 ...
##   ..$ originalName: Factor w/ 2 levels "RF","LogReg": 1 1 1 1 1 1 1 1 1 1 ...
##  $ class    : chr "Yes"
##  $ probNames: chr [1:2] "RF" "LogReg"
##  $ pct      : num 4.31
```

```
##  $ call     : language lift.formula(x = failed_insolvent ~ RF + LogReg, data = testResults,      labe
##  - attr(*, "class")= chr "lift"
```

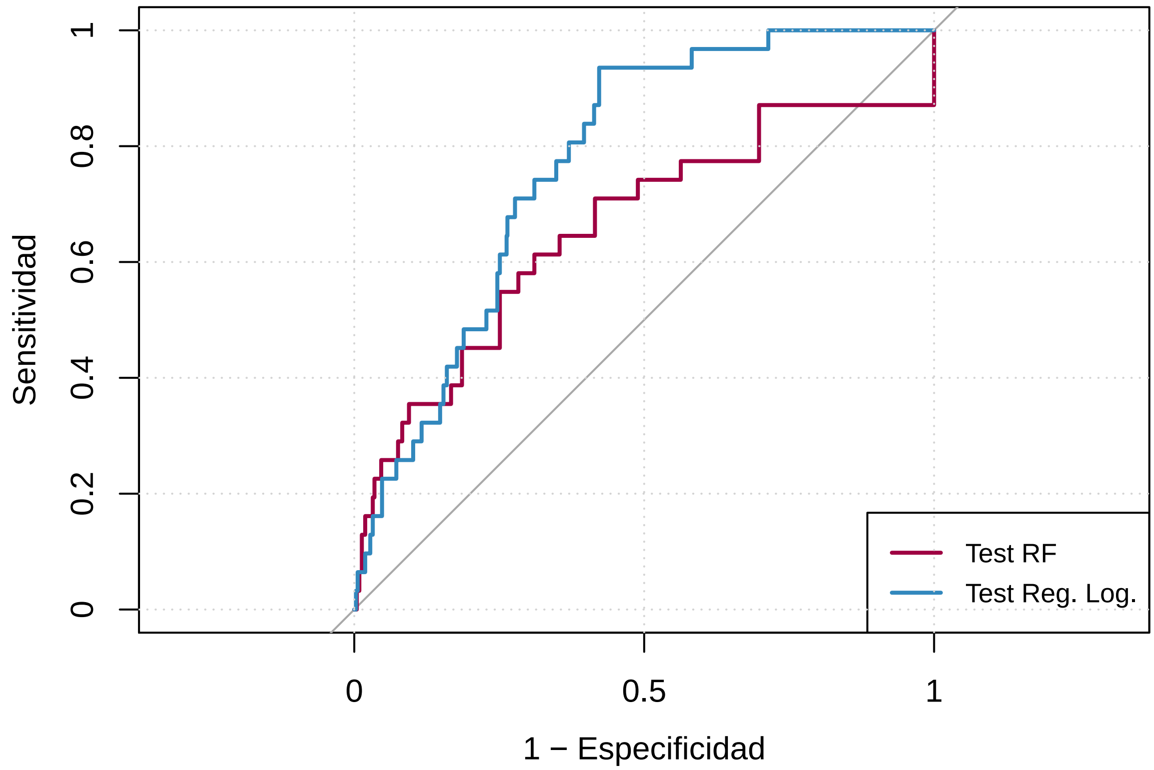
```r
prop.table(table(testResults$failed_insolvent))
```

```
##
##        Yes         No
## 0.04305556 0.95694444
```

```r
lift1$pct
```

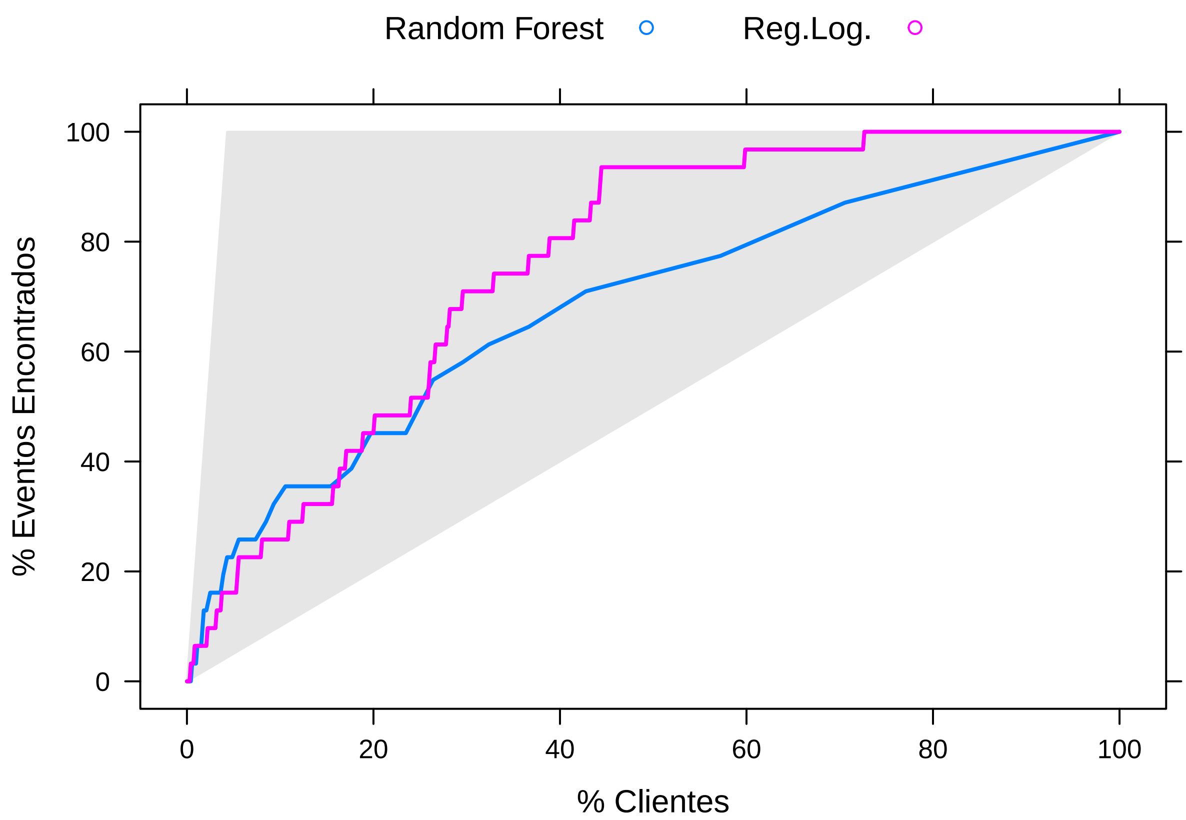
```
## [1] 4.305556
```

```r
plotTheme = caretTheme()  #CONFIGURACION DE COLORES
plot(rfTestROC, type = "S", col = plotTheme$superpose.line$col[1],
     legacy.axes = TRUE, xlab="1 - Especificidad",ylab="Sensitividad")
plot(lrTestROC, type = "S", col = plotTheme$superpose.line$col[2],
     add = TRUE, legacy.axes = TRUE)
legend("bottomright",
       c("Test RF", "Test Reg. Log."),
       cex = .85,
       col = plotTheme$superpose.line$col[1:2],
       lwd = rep(2, 2),
       lty = rep(1, 2))
grid()
```




```r
xyplot(lift1,
       ylab = "% Eventos Encontrados",
```

```
        xlab =  "% Clientes",
        lwd = 2,
        type = "l", auto.key = list(columns = 2))
```




Los resultados obtenidos con ambos clasificadores para la clase minoritaria *Yes* no son buenos. La construcción directa de modelos de clasificación sobre datos no balanceado suele conllevar bajas tasas de acierto sobre las clases minoritorias, e incluso valores bajos para el coeficiente AUC en problemas de clasificación binaria.

A continuación aplicaremos algunas de las principales estrategias para construir modelos más eficientes cuando los datos no están balanceados.

- Puntos de corte alternativos:
- Costes de Clasificación Incorrecta
- Métodos de muestreo
- Método SMOTE

# 4 Muestreo en la clase mayoritaria (*Downsampling*)

## 4.1 Balanceo con la técnica *Downsamplig*

Sean $n$ y $N$, los totales de casos en las clases minoritarias y mayoritarias (suponemos clasificación binaria) en el conjunto de entrenamiento.

Se genera un conjunto de datos balanceado de tamaño $2n$ formado por:

- Los $n$ casos de la clase minoritaria.

- Una selección aleatoria de $n$ casos entre los $N$ de la clase mayoritaria

```
dim(training)
```

```
## [1] 1726   16
```

```
downSampled = downSample(training[, -ncol(training)],
                         training$failed_insolvent)
dim(downSampled)
```

```
## [1] 182  16
```

```
table(downSampled$Class)
```

```
##
## Yes  No
##  91  91
```

```
downSampled_valid = downSample(validation[, -ncol(validation)],
                         validation$failed_insolvent)
dim(downSampled_valid)
```

```
## [1] 30 16
```

```
table(downSampled_valid$Class)
```

```
##
## Yes  No
##  15  15
```

```
downSampled_train_valid=rbind(downSampled, downSampled_valid )
dim(downSampled_train_valid)
```

```
## [1] 212  16
```

## 4.2 Modelo Random Forest con datos Downsampling

A continuación se va a construir el modelo Random Forest sobre el conjunto resultante del resmuestreo downsampling

```
rfDown = train(Class ~ .,
              data = downSampled_train_valid,
              method = "rf",
              trControl = ctrlcv,
              ntree = 100,
              do.trace=TRUE,
              tuneLength=3,
              metric = "Sens")
```

```
rfDown
```

```
## Random Forest
##
## 212 samples
##  15 predictor
##   2 classes: 'Yes', 'No'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold)
```

```
## Summary of sample sizes: 142, 140, 142
## Resampling results across tuning parameters:
##
##   mtry  ROC        Sens       Spec       Accuracy   Kappa
##    2    0.7220358  0.7066138  0.6809524  0.6937831  0.3875661
##    8    0.7412885  0.7076720  0.6997354  0.7037037  0.4074074
##   15    0.7418384  0.7074074  0.7560847  0.7317460  0.4634921
##
## Sens was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 8.
```

```r
rfDown$results
```

```
##   mtry       ROC      Sens      Spec  Accuracy     Kappa      ROCSD
## 1    2 0.7220358 0.7066138 0.6809524 0.6937831 0.3875661 0.06422537
## 2    8 0.7412885 0.7076720 0.6997354 0.7037037 0.4074074 0.05359232
## 3   15 0.7418384 0.7074074 0.7560847 0.7317460 0.4634921 0.03724253
##       SensSD    SpecSD AccuracySD    KappaSD
## 1 0.09032406 0.1567094 0.03827737 0.07655474
## 2 0.03078233 0.1496034 0.08050504 0.16101008
## 3 0.01920132 0.1255508 0.05681121 0.11362241
```

Probabilidades estimadas de la categoría *Yes*

```r
validResults$rfDown = predict(rfDown, validation, type = "prob")[,1]
testResults$rfDown = predict(rfDown, testing, type = "prob")[,1]
```

Vamos a calcular las medidas de rendimiento en el conjunto test

```r
rfDownTestROC = roc(testResults$failed_insolvent, testResults$rfDown,
                levels = rev(levels(testResults$failed_insolvent)))
rfDownTestROC
```

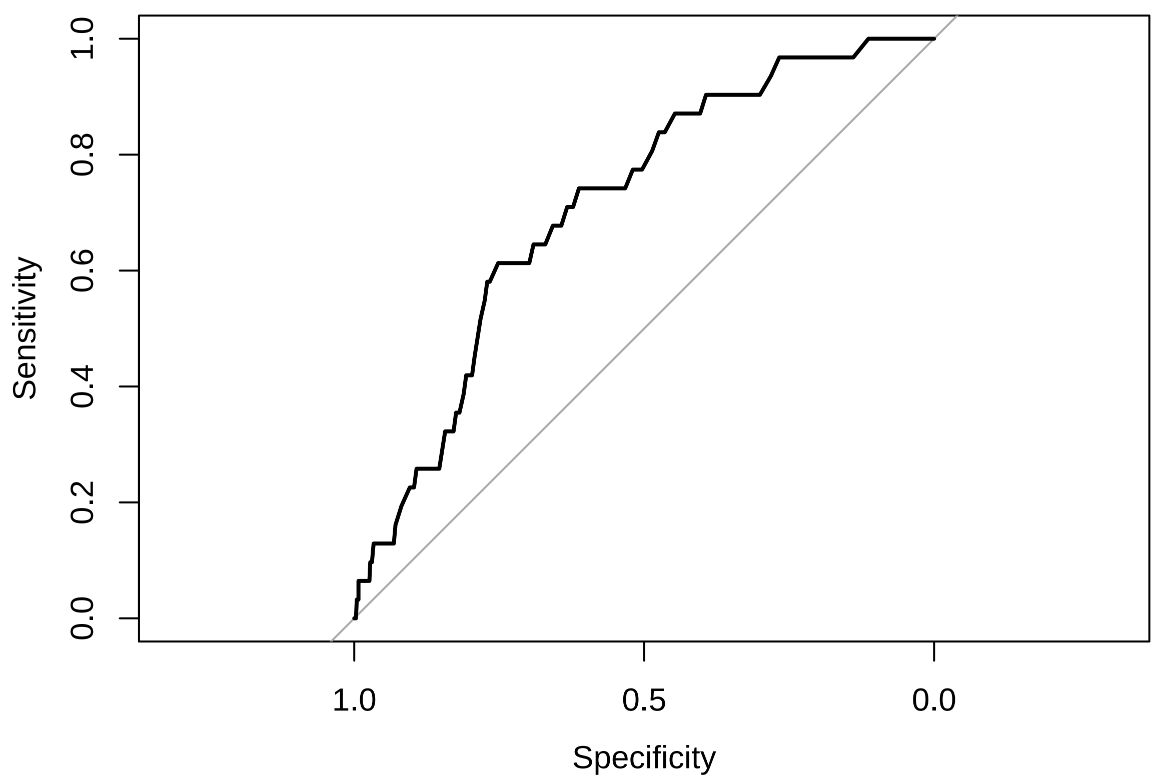
```
##
## Call:
## roc.default(response = testResults$failed_insolvent, predictor = testResults$rfDown,     levels = re
##
## Data: testResults$rfDown in 689 controls (testResults$failed_insolvent No) < 31 cases (testResults$f
## Area under the curve: 0.7122
```

```r
rfDownTestCM = confusionMatrix(predict(rfDown, testing),
                           testResults$failed_insolvent)
rfDownTestCM
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Yes  No
##        Yes  19 176
##        No   12 513
##
##                Accuracy : 0.7389
##                  95% CI : (0.7052, 0.7706)
##     No Information Rate : 0.9569
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1014
##  Mcnemar's Test P-Value : <2e-16
```

```
##
##             Sensitivity : 0.61290
##             Specificity : 0.74456
##          Pos Pred Value : 0.09744
##          Neg Pred Value : 0.97714
##              Prevalence : 0.04306
##          Detection Rate : 0.02639
##    Detection Prevalence : 0.27083
##       Balanced Accuracy : 0.67873
##
##        'Positive' Class : Yes
##
```

```
plot(rfDownTestROC)
```




### 4.3 Modelo Regresión logística con datos Downsampling

A continuación se va a construir el modelo Regresión logística sobre el conjunto resultante del resmuestreo downsampling

```
lrDown = train(Class ~ .,
              data = downSampled_train_valid,
              method = "glm",
              trControl = ctrlrlog)
lrDown
```

```
## Generalized Linear Model
```

```
##
## 212 samples
##  15 predictor
##   2 classes: 'Yes', 'No'
##
## No pre-processing
## Resampling: None
```

```r
summary(lrDown)
```

```
##
## Call:
## NULL
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.43228  -0.78365  -0.02948   0.78183   2.19953
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  7.750e+00  2.390e+00   3.243 0.001184 **
## CETL         3.246e-01  2.596e-01   1.251 0.211072
## STLTA        6.477e-01  1.283e+00   0.505 0.613706
## TLCA         3.180e-01  1.743e-01   1.824 0.068096 .
## NWTA        -2.564e-01  7.806e-01  -0.328 0.742545
## QACA        -1.879e+00  1.235e+00  -1.521 0.128187
## NCNW         1.141e-02  2.911e-02   0.392 0.695072
## CRATIO      -2.616e-01  2.990e-01  -0.875 0.381495
## CASHTA       4.064e+00  1.384e+00   2.936 0.003329 **
## PRTA         1.416e+00  7.430e-01   1.906 0.056691 .
## TCTD         7.836e-05  8.277e-03   0.009 0.992446
## TCTL         3.771e-01  9.100e-01   0.414 0.678616
## TDTA         1.491e+00  1.274e+00   1.170 0.241871
## ln_assets   -7.891e-01  2.073e-01  -3.807 0.000141 ***
## CHNW_new     5.295e-02  9.439e-02   0.561 0.574823
## CHNWTA_new  -4.865e-02  1.034e-01  -0.471 0.637996
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 293.89  on 211  degrees of freedom
## Residual deviance: 208.06  on 196  degrees of freedom
## AIC: 240.06
##
## Number of Fisher Scoring iterations: 6
```

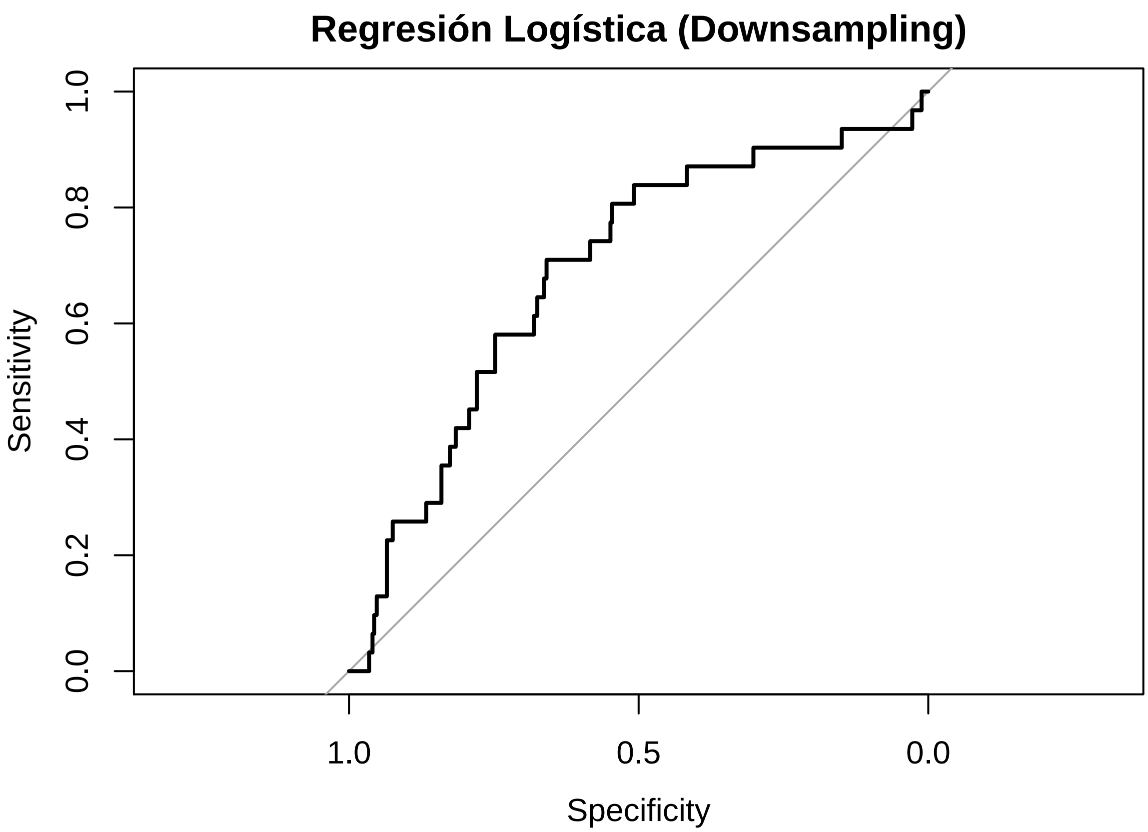
Probabilidades estimadas para la clase *Yes*

```r
validResults$lrDown = predict(lrDown,
                              validation,
                              type = "prob")[,1]

testResults$lrDown = predict(lrDown,
                             testing,
                             type = "prob")[,1]
```

```r
lrDownTestROC = roc(testResults$failed_insolvent, testResults$lrDown,
                    levels = rev(levels(testResults$failed_insolvent)))
lrDownTestROC
```

```
##
## Call:
## roc.default(response = testResults$failed_insolvent, predictor = testResults$lrDown,     levels = re
##
## Data: testResults$lrDown in 689 controls (testResults$failed_insolvent No) < 31 cases (testResults$fa
## Area under the curve: 0.6893
```

```r
plot(lrDownTestROC, main="Regresión Logística (Downsampling)")
```




```r
lrDownTestCM = confusionMatrix(predict(lrDown, testing),
                               testResults$failed_insolvent)
lrDownTestCM
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Yes  No
##        Yes  19 222
##        No   12 467
##
##                Accuracy : 0.675
##                  95% CI : (0.6394, 0.7091)
##     No Information Rate : 0.9569
```

```
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0686
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.61290
##             Specificity : 0.67779
##          Pos Pred Value : 0.07884
##          Neg Pred Value : 0.97495
##              Prevalence : 0.04306
##          Detection Rate : 0.02639
##    Detection Prevalence : 0.33472
##       Balanced Accuracy : 0.64535
##
##        'Positive' Class : Yes
##
```

# 5 Remuestreo en la clase minoritaria (*Upsampling*)

## 5.1 Balanceo con la técnica *Upsamplig*

```r
upSampled = upSample(training[, -ncol(training)],
                     training$failed_insolvent)
dim(upSampled)
```

```
## [1] 3270   16
```

```r
table(upSampled$Class)
```

```
##
##  Yes   No
## 1635 1635
```

```r
upSampled_valid = upSample(validation[, -ncol(validation)],
                     validation$failed_insolvent)
dim(upSampled_valid)
```

```
## [1] 832  16
```

```r
table(upSampled_valid$Class)
```

```
##
## Yes  No
## 416 416
```

```r
upSampled_train_valid=rbind(upSampled, upSampled_valid )
dim(upSampled_train_valid)
```

```
## [1] 4102   16
```

## 5.2 Modelo Random Forest con datos Upsampling

A continuación se va a construir el modelo Random Forest sobre el conjunto resultante del resmuestreo upsampling

```r
rfUp = train(Class ~ .,
              data = upSampled_train_valid,
              method = "rf",
              trControl = ctrlcv,
              ntree = 100,
              do.trace=TRUE,
              tuneLength=3,
              metric = "Sens")
```

```r
rfUp
```

```
## Random Forest
##
## 4102 samples
##   15 predictor
##    2 classes: 'Yes', 'No'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold)
## Summary of sample sizes: 2735, 2735, 2734
## Resampling results across tuning parameters:
##
##   mtry  ROC  Sens  Spec       Accuracy   Kappa
##    2    1    1     0.9805025  0.9902490  0.9804981
##    8    1    1     0.9717256  0.9858609  0.9717219
##   15    1    1     0.9649023  0.9824487  0.9648976
##
## Sens was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 2.
```

```r
rfUp$results
```

```
##   mtry ROC Sens      Spec  Accuracy     Kappa ROCSD SensSD      SpecSD
## 1    2   1    1 0.9805025 0.9902490 0.9804981     0      0 0.009727325
## 2    8   1    1 0.9717256 0.9858609 0.9717219     0      0 0.008297722
## 3   15   1    1 0.9649023 0.9824487 0.9648976     0      0 0.012728131
##    AccuracySD     KappaSD
## 1 0.004869758 0.009739247
## 2 0.004158008 0.008315501
## 3 0.006371805 0.012743150
```

Probabilidades estimadas de la categoría *Yes*

```r
validResults$rfUp = predict(rfUp, validation,
                          type = "prob")[,1]
testResults$rfUp = predict(rfUp, testing,
                         type = "prob")[,1]
```

Vamos a calcular las medidas de rendimiento en el conjunto test

```r
rfUpTestROC = roc(testResults$failed_insolvent, testResults$rfUp,
                levels = rev(levels(testResults$failed_insolvent)))
rfUpTestROC
```

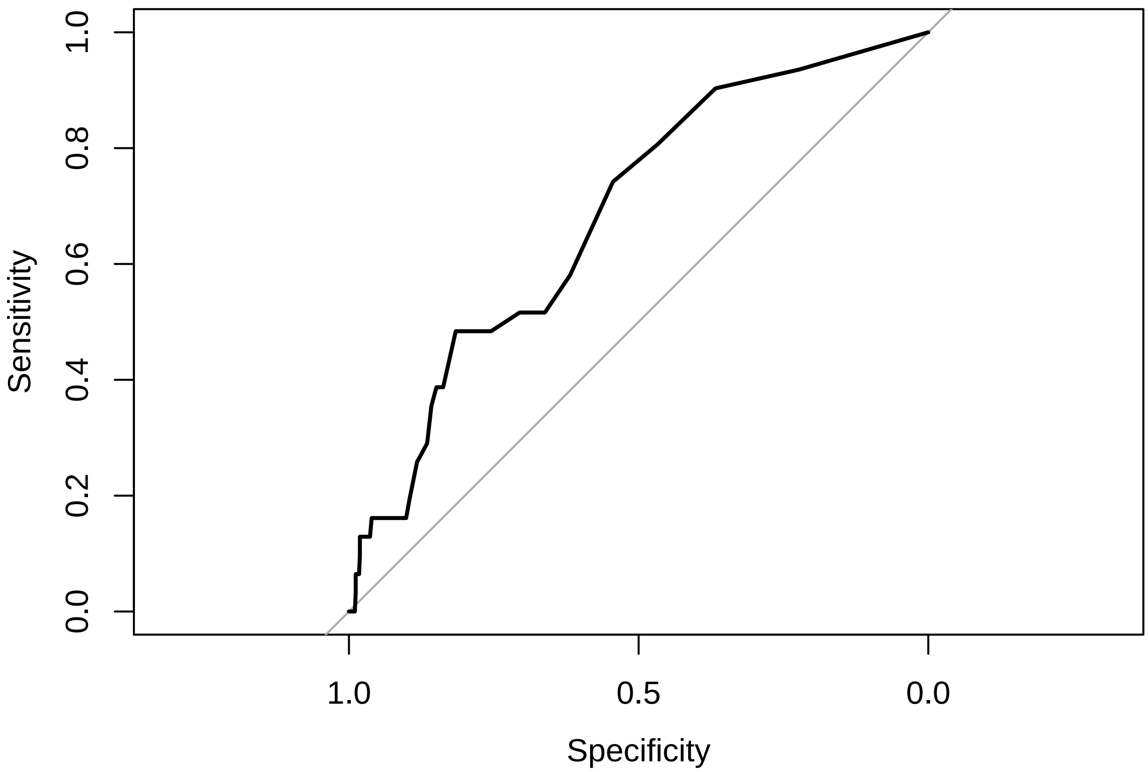
```
##
## Call:
## roc.default(response = testResults$failed_insolvent, predictor = testResults$rfUp,     levels = rev(l
```

```
##
## Data: testResults$rfUp in 689 controls (testResults$failed_insolvent No) < 31 cases (testResults$fai
## Area under the curve: 0.6841
```

```r
rfUpTestCM = confusionMatrix(predict(rfUp, testing),
                             testResults$failed_insolvent)
rfUpTestCM
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Yes  No
##        Yes   0   5
##        No   31 684
##
##                Accuracy : 0.95
##                  95% CI : (0.9314, 0.9647)
##     No Information Rate : 0.9569
##     P-Value [Acc > NIR] : 0.8439
##
##                   Kappa : -0.0121
##  Mcnemar's Test P-Value : 3.091e-05
##
##             Sensitivity : 0.000000
##             Specificity : 0.992743
##          Pos Pred Value : 0.000000
##          Neg Pred Value : 0.956643
##              Prevalence : 0.043056
##          Detection Rate : 0.000000
##    Detection Prevalence : 0.006944
##       Balanced Accuracy : 0.496372
##
##        'Positive' Class : Yes
##
```

```r
plot(rfUpTestROC)
```




### 5.3 Modelo Regresión logística con datos Upsampling

A continuación se va a construir el modelo Regresión logística sobre el conjunto resultante del resmuestreo upsampling

```
lrUp = train(Class ~ .,
             data = upSampled_train_valid,
             method = "glm",
             trControl = ctrlrlog)
lrUp
```

```
## Generalized Linear Model
##
## 4102 samples
##   15 predictor
##    2 classes: 'Yes', 'No'
##
## No pre-processing
## Resampling: None
```

```
summary(lrUp)
```

```
##
## Call:
## NULL
##
```

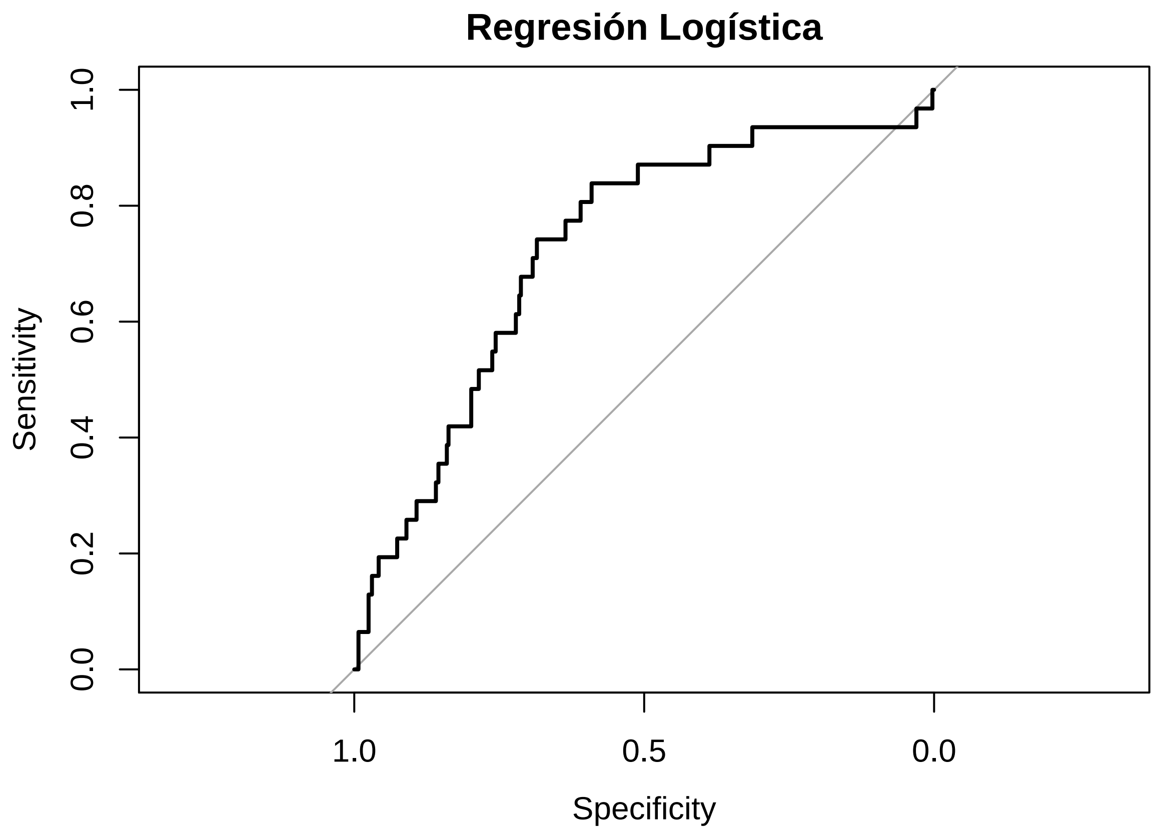
```
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.3716  -0.8592  -0.0970   0.8719   3.4317
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  7.771430   0.545621  14.243  < 2e-16 ***
## CETL         0.123901   0.024140   5.133 2.86e-07 ***
## STLTA       -0.873557   0.265670  -3.288 0.001009 **
## TLCA         0.156810   0.022067   7.106 1.19e-12 ***
## NWTA        -0.262743   0.091057  -2.885 0.003908 **
## QACA        -0.754398   0.227324  -3.319 0.000905 ***
## NCNW        -0.010955   0.006407  -1.710 0.087283 .
## CRATIO      -0.024713   0.023142  -1.068 0.285565
## CASHTA       1.921214   0.238248   8.064 7.39e-16 ***
## PRTA         0.961279   0.092518  10.390  < 2e-16 ***
## TCTD         0.001867   0.002152   0.867 0.385672
## TCTL        -0.071345   0.187888  -0.380 0.704153
## TDTA        -0.195675   0.219796  -0.890 0.373328
## ln_assets   -0.696370   0.044859 -15.524  < 2e-16 ***
## CHNW_new     0.058426   0.023114   2.528 0.011481 *
## CHNWTA_new  -0.022948   0.022645  -1.013 0.310871
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 5686.6  on 4101  degrees of freedom
## Residual deviance: 4328.8  on 4086  degrees of freedom
## AIC: 4360.8
##
## Number of Fisher Scoring iterations: 6
```

Probabilidades estimadas para la clase *Yes*

```
validResults$lrUp = predict(lrUp,
                            validation,
                            type = "prob")[,1]

testResults$lrUp = predict(lrUp,
                           testing,
                           type = "prob")[,1]

lrUpTestROC = roc(testResults$failed_insolvent, testResults$lrUp,
                levels = rev(levels(testResults$failed_insolvent)))
lrUpTestROC
```

```
##
## Call:
## roc.default(response = testResults$failed_insolvent, predictor = testResults$lrUp,     levels = rev(l
##
## Data: testResults$lrUp in 689 controls (testResults$failed_insolvent No) < 31 cases (testResults$fail
## Area under the curve: 0.7255
```

```
plot(lrUpTestROC, main="Regresión Logística")
```




```
lrUpTestCM = confusionMatrix(predict(lrUp, testing),
                             testResults$failed_insolvent)
lrUpTestCM
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Yes  No
##        Yes  19 196
##        No   12 493
##
##                Accuracy : 0.7111
##                  95% CI : (0.6765, 0.744)
##     No Information Rate : 0.9569
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0857
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.61290
##             Specificity : 0.71553
##          Pos Pred Value : 0.08837
##          Neg Pred Value : 0.97624
##              Prevalence : 0.04306
##          Detection Rate : 0.02639
##    Detection Prevalence : 0.29861
```

```
##       Balanced Accuracy : 0.66422
##
##        'Positive' Class : Yes
##
```

# 6 Método *SMOTE* (Synthetic Minority Over-Sampling Technique)

## 6.1 Balanceo con la técnica *SMOTE*

```r
smoted = SMOTE(failed_insolvent ~ ., data = training)
dim(smoted)
```

```
## [1] 637  16
```

```r
table(smoted$failed_insolvent)
```

```
##
## Yes  No
## 273 364
```

```r
smoted_valid = SMOTE(failed_insolvent ~ ., data = validation)
smoted_train_valid=rbind(smoted, smoted_valid)
dim(smoted_train_valid)
```

```
## [1] 742  16
```

```r
table(smoted_train_valid$failed_insolvent)
```

```
##
## Yes  No
## 318 424
```

## 6.2 Random Forest con datos SMOTE

A continuación se va a construir el modelo Random Forest sobre el conjunto resultante de aplicar la técnica SMOTE

```r
rfSmote = train(failed_insolvent ~ .,
              data = smoted_train_valid,
              method = "rf",
              trControl = ctrlcv,
              ntree = 100,
              do.trace=TRUE,
              tuneLength=3,
              metric = "Sens")
```

```r
rfSmote
```

```
## Random Forest
##
## 742 samples
##  15 predictor
##   2 classes: 'Yes', 'No'
##
## No pre-processing
```

```
## Resampling: Cross-Validated (3 fold)
## Summary of sample sizes: 494, 495, 495
## Resampling results across tuning parameters:
##
##   mtry  ROC        Sens       Spec       Accuracy   Kappa
##    2    0.9200173  0.8301887  0.8631172  0.8490379  0.6923657
##    8    0.9164109  0.8018868  0.8702094  0.8409571  0.6742938
##   15    0.9215873  0.8081761  0.8843772  0.8517370  0.6959091
##
## Sens was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 2.
```

```
rfSmote$results
```

```
##   mtry       ROC      Sens      Spec  Accuracy     Kappa      ROCSD
## 1    2 0.9200173 0.8301887 0.8631172 0.8490379 0.6923657 0.01665530
## 2    8 0.9164109 0.8018868 0.8702094 0.8409571 0.6742938 0.01157448
## 3   15 0.9215873 0.8081761 0.8843772 0.8517370 0.6959091 0.01014680
##       SensSD     SpecSD  AccuracySD    KappaSD
## 1 0.02495992 0.03606744 0.012007709 0.02214637
## 2 0.02495992 0.02725085 0.013236372 0.02612703
## 3 0.01089340 0.02202158 0.009670259 0.01845227
```

Probabilidades estimadas de la categoría *Yes*

```
validResults$rfSmote = predict(rfSmote, validation,
                          type = "prob")[,1]

testResults$rfSmote = predict(rfSmote, testing,
                         type = "prob")[,1]
```

Vamos a calcular las medidas de rendimiento en el conjunto test

```
rfSmoteTestROC = roc(testResults$failed_insolvent, testResults$rfSmote,
                levels = rev(levels(testResults$failed_insolvent)))
rfSmoteTestROC
```

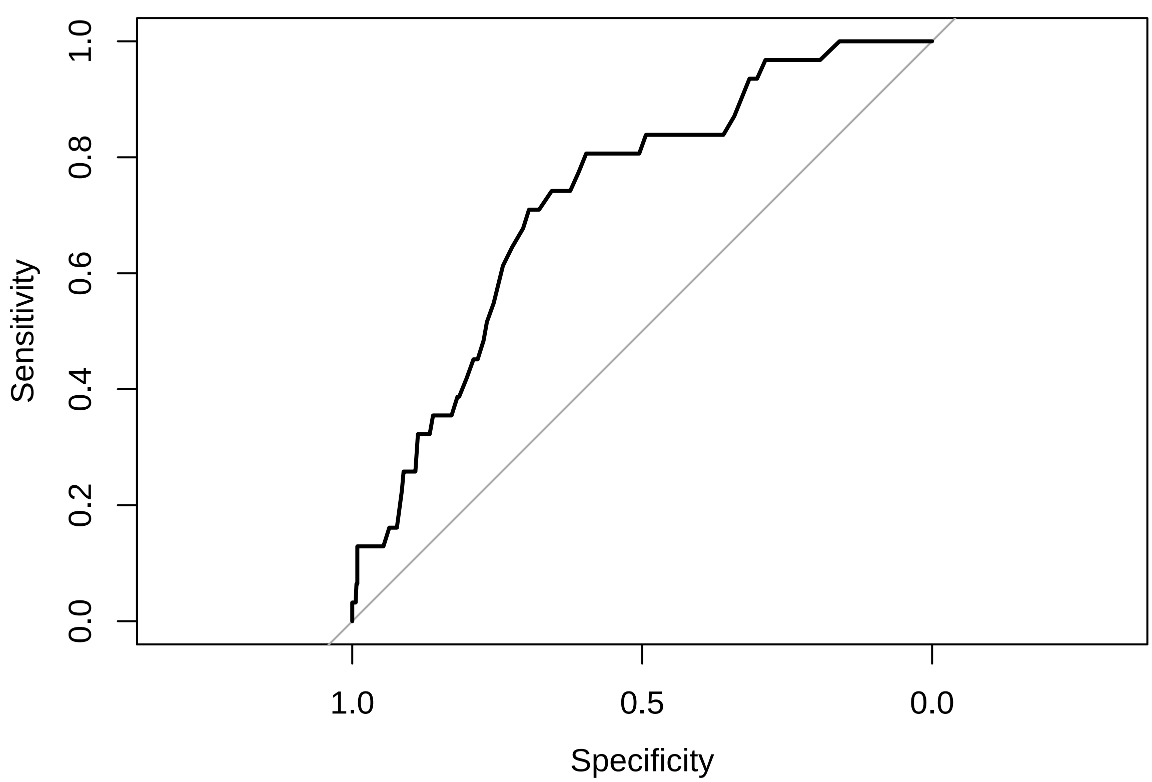
```
##
## Call:
## roc.default(response = testResults$failed_insolvent, predictor = testResults$rfSmote,     levels = re
##
## Data: testResults$rfSmote in 689 controls (testResults$failed_insolvent No) < 31 cases (testResults$f
## Area under the curve: 0.7277
```

```
rfSmoteTestCM = confusionMatrix(predict(rfSmote, testing),
                           testResults$failed_insolvent)
rfSmoteTestCM
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Yes  No
##        Yes  10  80
##        No   21 609
##
##                Accuracy : 0.8597
##                  95% CI : (0.8322, 0.8843)
##     No Information Rate : 0.9569
```

```
##      P-Value [Acc > NIR] : 1
##
##                    Kappa : 0.1082
##   Mcnemar's Test P-Value : 7.87e-09
##
##              Sensitivity : 0.32258
##              Specificity : 0.88389
##           Pos Pred Value : 0.11111
##           Neg Pred Value : 0.96667
##               Prevalence : 0.04306
##           Detection Rate : 0.01389
##     Detection Prevalence : 0.12500
##        Balanced Accuracy : 0.60324
##
##         'Positive' Class : Yes
##
```

```
plot(rfSmoteTestROC)
```




### 6.3 Modelo Regresión logística con datos SMOTE

A continuación se va a construir el modelo Regresión logística sobre el conjunto resultante de aplicar la técnica SMOTE

```
lrSmote = train(failed_insolvent ~ .,
                data = smoted_train_valid,
```

```r
                 method = "glm",
                 trControl = ctrlrlog)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```r
lrSmote
```

```
## Generalized Linear Model
##
## 742 samples
##  15 predictor
##   2 classes: 'Yes', 'No'
##
## No pre-processing
## Resampling: None
```

```r
summary(lrSmote)
```

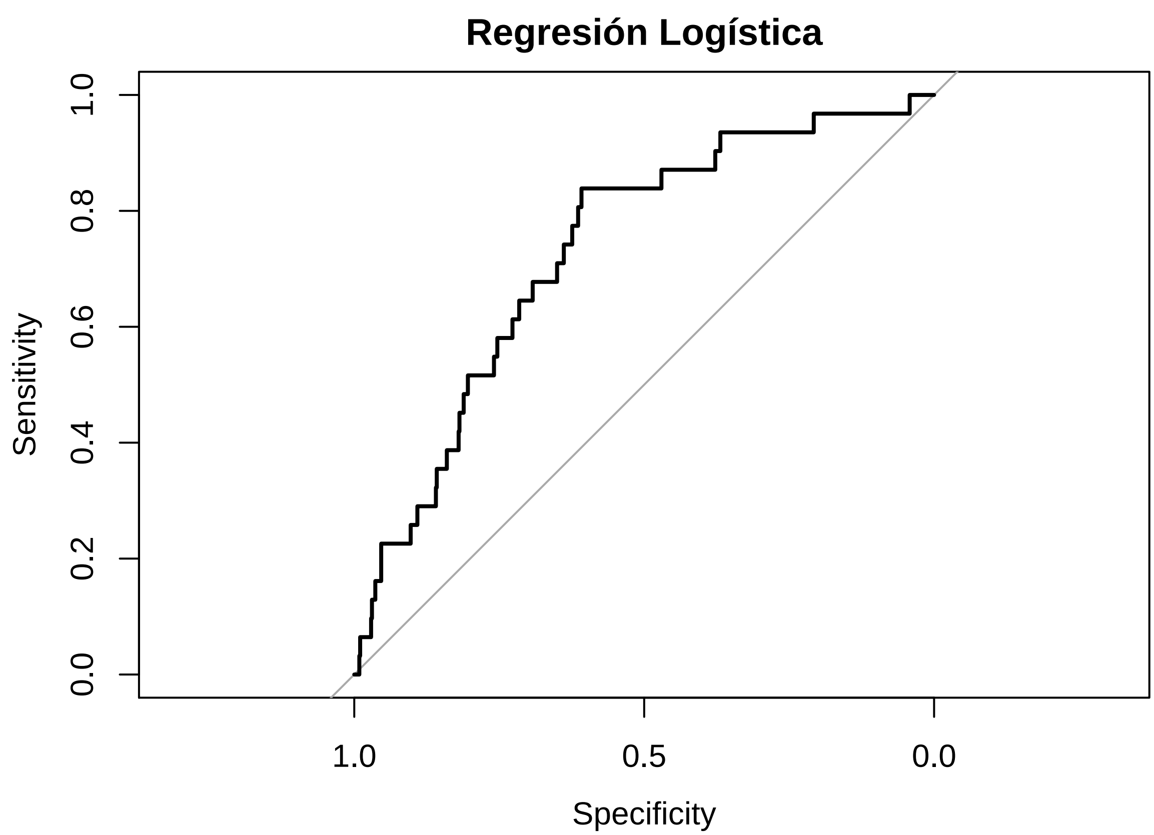
```
##
## Call:
## NULL
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.7277  -0.8880   0.3152   0.8220   3.5288
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  8.245261   1.333455   6.183 6.27e-10 ***
## CETL         0.134346   0.069946   1.921 0.054769 .
## STLTA       -1.868821   0.729645  -2.561 0.010429 *
## TLCA         0.207584   0.061962   3.350 0.000808 ***
## NWTA        -0.092583   0.216614  -0.427 0.669079
## QACA        -0.784894   0.533394  -1.472 0.141153
## NCNW         0.018824   0.013670   1.377 0.168518
## CRATIO      -0.028736   0.063501  -0.453 0.650891
## CASHTA       2.160226   0.580077   3.724 0.000196 ***
## PRTA         1.031870   0.254373   4.057 4.98e-05 ***
## TCTD         0.007794   0.005155   1.512 0.130543
## TCTL        -0.805809   0.482811  -1.669 0.095119 .
## TDTA        -0.058988   0.525161  -0.112 0.910566
## ln_assets   -0.683329   0.111748  -6.115 9.66e-10 ***
## CHNW_new     0.113559   0.071993   1.577 0.114709
## CHNWTA_new  -0.081461   0.066734  -1.221 0.222209
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1013.44  on 741  degrees of freedom
## Residual deviance:  755.92  on 726  degrees of freedom
## AIC: 787.92
##
## Number of Fisher Scoring iterations: 6
```

Probabilidades estimadas para la clase *Yes*

```r
validResults$lrSmote = predict(lrSmote,
                               validation,
                               type = "prob")[,1]

testResults$lrSmote = predict(lrSmote,
                              testing,
                              type = "prob")[,1]

lrSmoteTestROC = roc(testResults$failed_insolvent,
                     testResults$lrSmote,
                     levels = rev(levels(testResults$failed_insolvent)))

lrSmoteTestROC
```

```
##
## Call:
## roc.default(response = testResults$failed_insolvent, predictor = testResults$lrSmote,     levels = re
##
## Data: testResults$lrSmote in 689 controls (testResults$failed_insolvent No) < 31 cases (testResults$f
## Area under the curve: 0.7305
```

```r
plot(lrSmoteTestROC, main="Regresión Logística")
```




```r
lrSmoteTestCM = confusionMatrix(predict(lrSmote, testing),
                                testResults$failed_insolvent)
lrSmoteTestCM
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Yes  No
##        Yes  16 142
##        No   15 547
##
##                Accuracy : 0.7819
##                  95% CI : (0.75, 0.8116)
##     No Information Rate : 0.9569
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1049
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.51613
##             Specificity : 0.79390
##          Pos Pred Value : 0.10127
##          Neg Pred Value : 0.97331
##              Prevalence : 0.04306
##          Detection Rate : 0.02222
##    Detection Prevalence : 0.21944
##       Balanced Accuracy : 0.65502
##
##        'Positive' Class : Yes
##
```

# 7. Conclusiones

## 7.1 Tabla comparativa

Las siguiente función obtiene un resumen de los distintos modelos construidos.

Parámetros:

- x: Modelo
- evl: conjunto de validación
- tst: conjunto test

La función determina el mejor umbral según:

- best.method="closest.topleft" en validación
- SALIDA: valROC, testROC, testSens, testSpec

```
samplingSummary = function(x, evl, tst)
  {
    lvl = rev(levels(tst$failed_insolvent))
    evlROC = roc(evl$failed_insolvent,
                 predict(x, evl, type = "prob")[,1],
                 levels = lvl)
    tstROC= roc(tst$failed_insolvent,
               predict(x, tst, type = "prob")[,1],
               levels = lvl)
    rocs = c(auc(evlROC),auc(tstROC))
    cut = coords(evlROC, x = "best", ret="threshold",
```

```
                  best.method="closest.topleft")
    # coords=punto de corte, el punto de corte lo calcula en el conjunto test
    bestVals = coords(tstROC, cut, ret=c("sensitivity", "specificity"))
    out = c(rocs, bestVals*100)
    names(out) = c("valROC", "testROC", "testSens", "testSpec")
    out
  }
```

Esta fucncion además de evaluar los resultados introduce una nueva estrategia para construir modelos más eficientes para datos no balanceados. Esta estrategia se conoce como *Puntos de corte alternativo*. En clasificación binaria como el problema que nos ocupa (el conjunto de datos analizado clasifica entre 2 clases: *Yes*, *No*) se admite una expresión donde se compara la probabilidad estimada de pertenecer a la clase de interés con un umbral. Este umbral por defecto es 0.5, que es el valor con el que se ha calculado la sensitividad y especificidad en cada uno de los modelos de los apartados anteriores. La idea de esta estrategia es utilizar otros puntos de corte que conduzcan a mayores valores para la sensitividad.

```
results = rbind(samplingSummary(rfFit, validation, testing),
                samplingSummary(rfDown, validation, testing),
                samplingSummary(rfUp, validation, testing),
                samplingSummary(rfSmote, validation, testing),
                samplingSummary(lrFit, validation, testing),
                samplingSummary(lrDown, validation, testing),
                samplingSummary(lrUp, validation, testing),
                samplingSummary(lrSmote, validation, testing))

rownames(results) = c("RF (Original)", "RF (Downsampling)",
                      "RF (Upsampling)", "RF (SMOTE)", "LR (Original)",
                      "LR (Downsampling)", "LR (Upsampling)", "LR (SMOTE)")


print(knitr::kable(round(results,4), format = "pandoc", align='c'))
```

```
##
##
##                        valROC    testROC    testSens    testSpec
## -------------------  --------  ---------  ----------  ----------
## RF (Original)         0.8031    0.6917     54.8387     74.8911
## RF (Downsampling)     0.9573    0.7122     22.5806     89.6952
## RF (Upsampling)       1.0000    0.6841      0.0000     99.8549
## RF (SMOTE)            0.9788    0.7277     12.9032     94.9202
## LR (Original)         0.8018    0.7724     48.3871     80.9869
## LR (Downsampling)     0.8338    0.6893     29.0323     86.6473
## LR (Upsampling)       0.8446    0.7255     48.3871     79.6807
## LR (SMOTE)            0.8186    0.7305     51.6129     76.9231
```

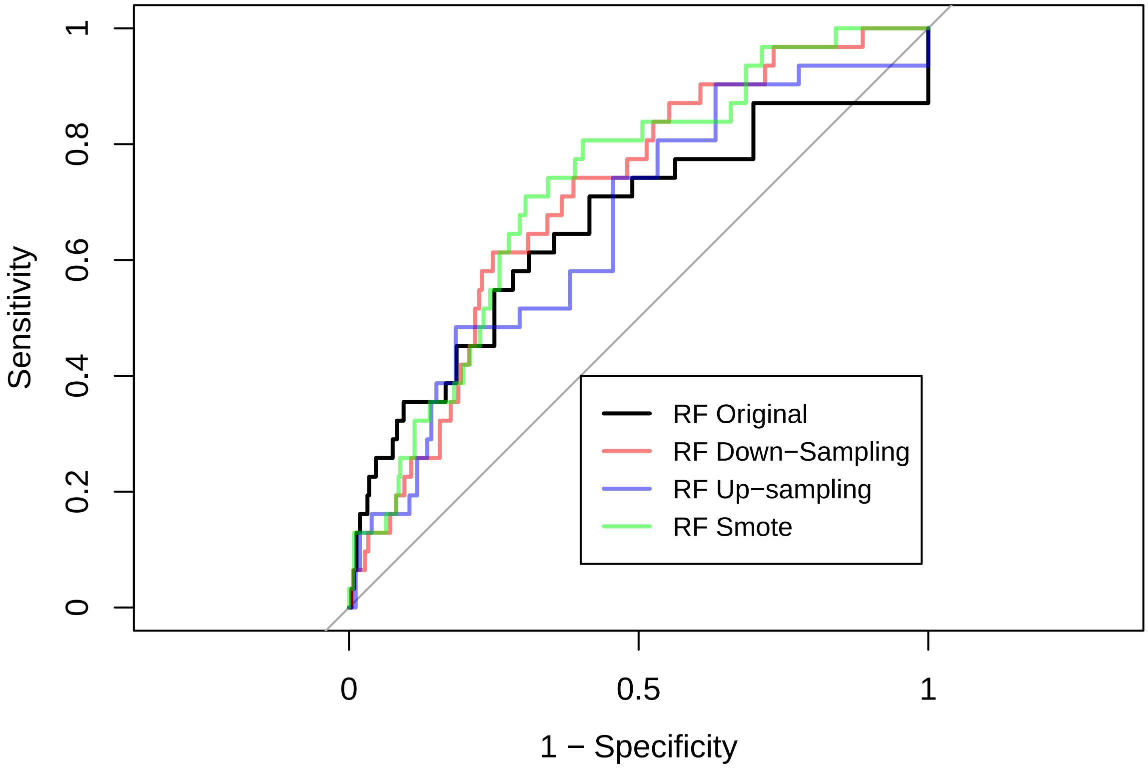
A la vista de los resultados podemos concluir que el modelo con el que mejores resultados se obtienen es con el de Regresión Logística, y aplicando la técnica de puntos de corte alternativos para datos no balanceados.

Observamos que realmente la técnica que ha mejorado los valores de sensitividad es la del punto de corte alternativo. El resto de estrategias aplicadas no mejoran de forma tan apreciable. No existe mucha diferencia con los resultados obtenidos de aplicar ese mismo a datos Smote, downSamplig o Upsampling.

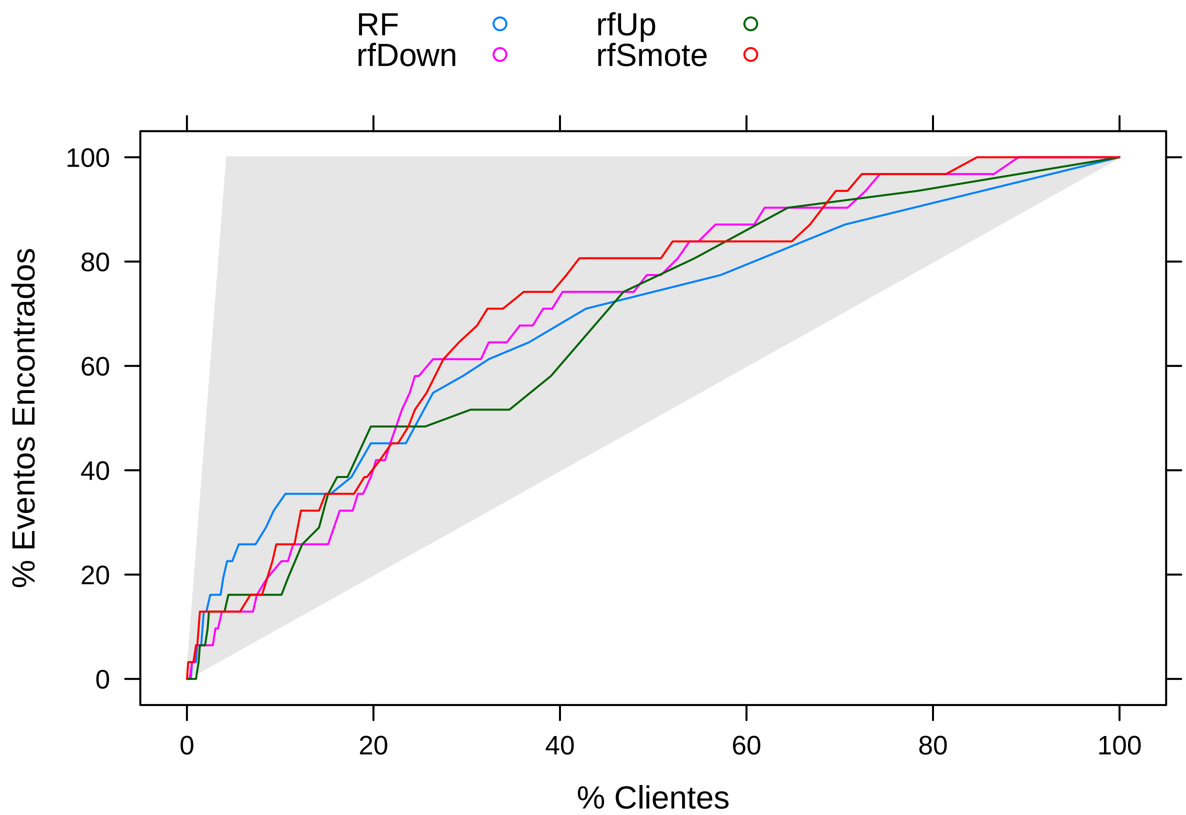
## 7.2 Representación gráfica

A continuación se representarán gráficamente los resultados de todos los modelos calculados con las técnicas mencionadas.

```
rocCols = c("black", rgb(1, 0, 0, .5), rgb(0, 0, 1, .5), rgb(0, 1, 0, .5))


plot(roc(testResults$failed_insolvent, testResults$RF,
         levels = rev(levels(testResults$failed_insolvent))),
         type = "S", col = rocCols[1], legacy.axes = TRUE)

plot(roc(testResults$failed_insolvent, testResults$rfDown,
         levels = rev(levels(testResults$failed_insolvent))),
         type = "S", col = rocCols[2],add = TRUE, legacy.axes = TRUE)

plot(roc(testResults$failed_insolvent, testResults$rfUp, levels =
         rev(levels(testResults$failed_insolvent))),
         type = "S", col = rocCols[3], add = TRUE, legacy.axes = TRUE)

plot(roc(testResults$failed_insolvent, testResults$rfSmote, levels =
         rev(levels(testResults$failed_insolvent))),
         type = "S", col = rocCols[4], add = TRUE, legacy.axes = TRUE)

legend(.6, .4,
       c("RF Original", "RF Down-Sampling", "RF Up-sampling", "RF Smote"),
       lty = rep(1, 3),
       lwd = rep(2, 3),
       cex = .8,
       col = rocCols)
```



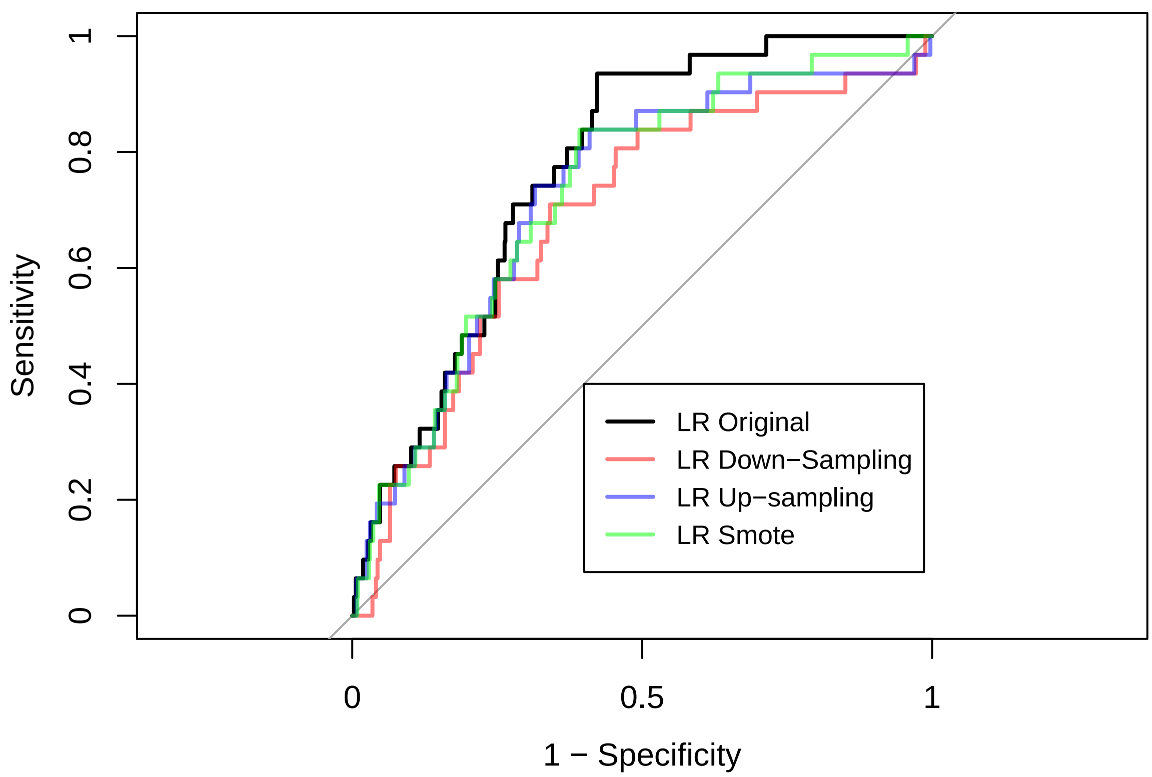

```
xyplot(lift(failed_insolvent ~ RF + rfDown + rfUp + rfSmote, data = testResults),
       type = "l",
       ylab = "% Eventos Encontrados",
       xlab =  "% Clientes",
       auto.key=list(columns = 3))
```



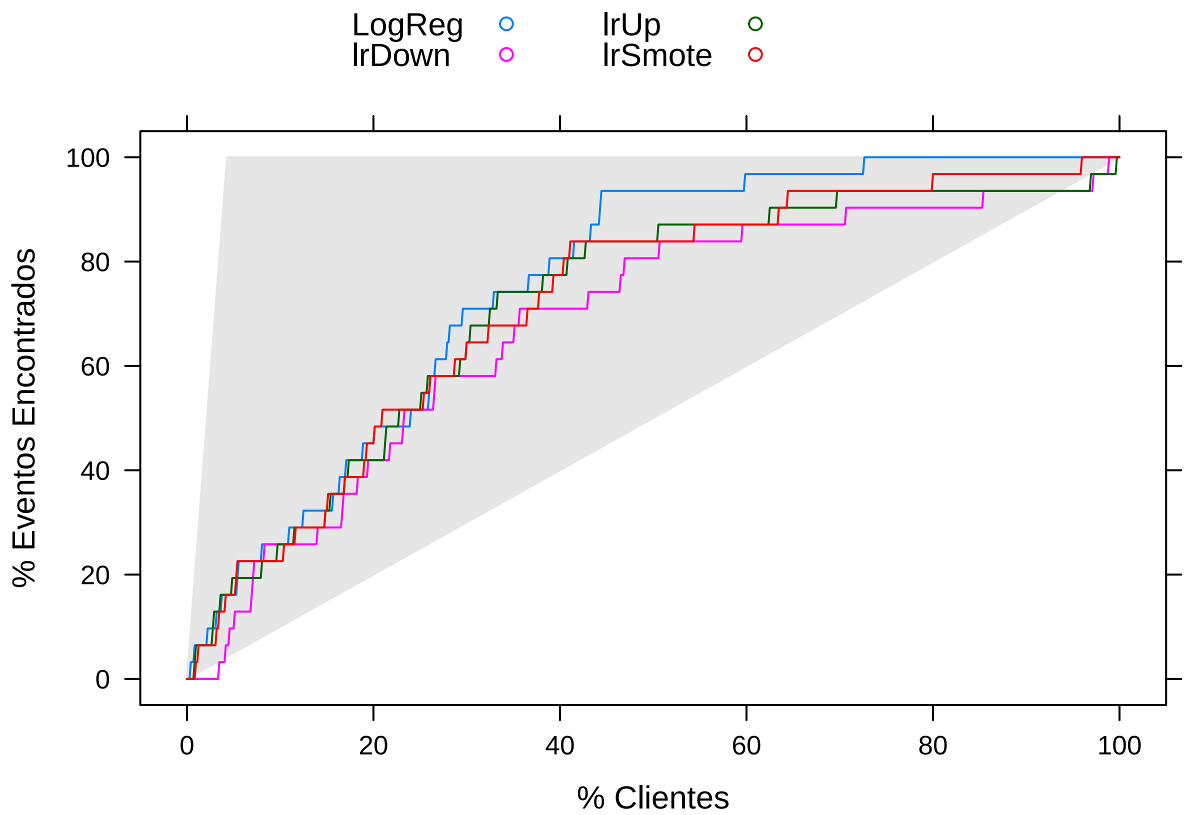

No existe muchas mejora al aplicar DownSampling, UpSampling o Smote al modelo Random Forest con puntos de corte alternativos.

```
rocCols = c("black", rgb(1, 0, 0, .5), rgb(0, 0, 1, .5), rgb(0, 1, 0, .5))


plot(roc(testResults$failed_insolvent, testResults$LogReg,
         levels = rev(levels(testResults$failed_insolvent))),
         type = "S", col = rocCols[1], legacy.axes = TRUE)

plot(roc(testResults$failed_insolvent, testResults$lrDown,
         levels = rev(levels(testResults$failed_insolvent))),
         type = "S", col = rocCols[2],add = TRUE, legacy.axes = TRUE)

plot(roc(testResults$failed_insolvent, testResults$lrUp, levels =
         rev(levels(testResults$failed_insolvent))),
         type = "S", col = rocCols[3], add = TRUE, legacy.axes = TRUE)

plot(roc(testResults$failed_insolvent, testResults$lrSmote, levels =
         rev(levels(testResults$failed_insolvent))),
         type = "S", col = rocCols[4], add = TRUE, legacy.axes = TRUE)

legend(.6, .4,
       c("LR Original", "LR Down-Sampling", "LR Up-sampling", "LR Smote"),
       lty = rep(1, 3),
       lwd = rep(2, 3),
       cex = .8,
```

```
        col = rocCols)
```




```
xyplot(lift(failed_insolvent ~ LogReg + lrDown + lrUp + lrSmote, data = testResults),
       type = "l",
       ylab = "% Eventos Encontrados",
       xlab =  "% Clientes",
       auto.key=list(columns = 3))
```




Se observa que el modelo de Regresión Logística original, con la técnica de puntos de corte alternativo es el que más se aproxima a la forma de triángulo y el que está por encima del resto de su familia con el que estamos comparando.