Práctica 2

Problemas de muestras desbalanzeadas

Carbonera Francesco
Giada Cesaro

21/11/2019

Índice

1 Introducción y análisis explorativo de los datos

Para esta práctica se utilizará el dominio "oil", que consiste en la detección de escapes de petróleo ("oil spill") en el mar a partir de imágenes de radar desde satélite, para distinguirlas de otras imágenes que parecen escapes pero no lo son (sombras, etc.).

Se carga el fichero de datos y se ve y su estructura.

```
dati <- readRDS("oil.rds")
dim(dati)
```

```
[1] 937  50
```

```
a<- introduce(dati)[,3:6]
a<-t(a)
knitr::kable(a, caption = "Resumen de los datos")
```

Table 1: Resumen de los datos

discrete_columns	1
continuous_columns	49
all_missing_columns	0
total_missing_values	0

Se ve que en conjunto de datos tiene 49 columnas numéricas y una columna discreta. Los atributos no tienen nombres significativos porque los valores son resúmenes de mediciones que describen una imagen (cada registro es la representación de una imagen, resumida de forma sencilla en 50 atributos). En total los registros son 937.

```
table(dati$class)
```

```
notoil    oil
   896     41
```

```
a<- sum(dati$class=="oil")/nrow(dati)
b<- sum(dati$class=="oil")/sum(dati$class=="notoil")
```

Se acaba de verificar que éste es un problema con clases desbalanzeadas, ya que el porcentaje de datos de clase positiva ("oil") es 0.0437567, y el cociente entre el número de datos de clase positiva y el de clase negativa es 0.0457589.

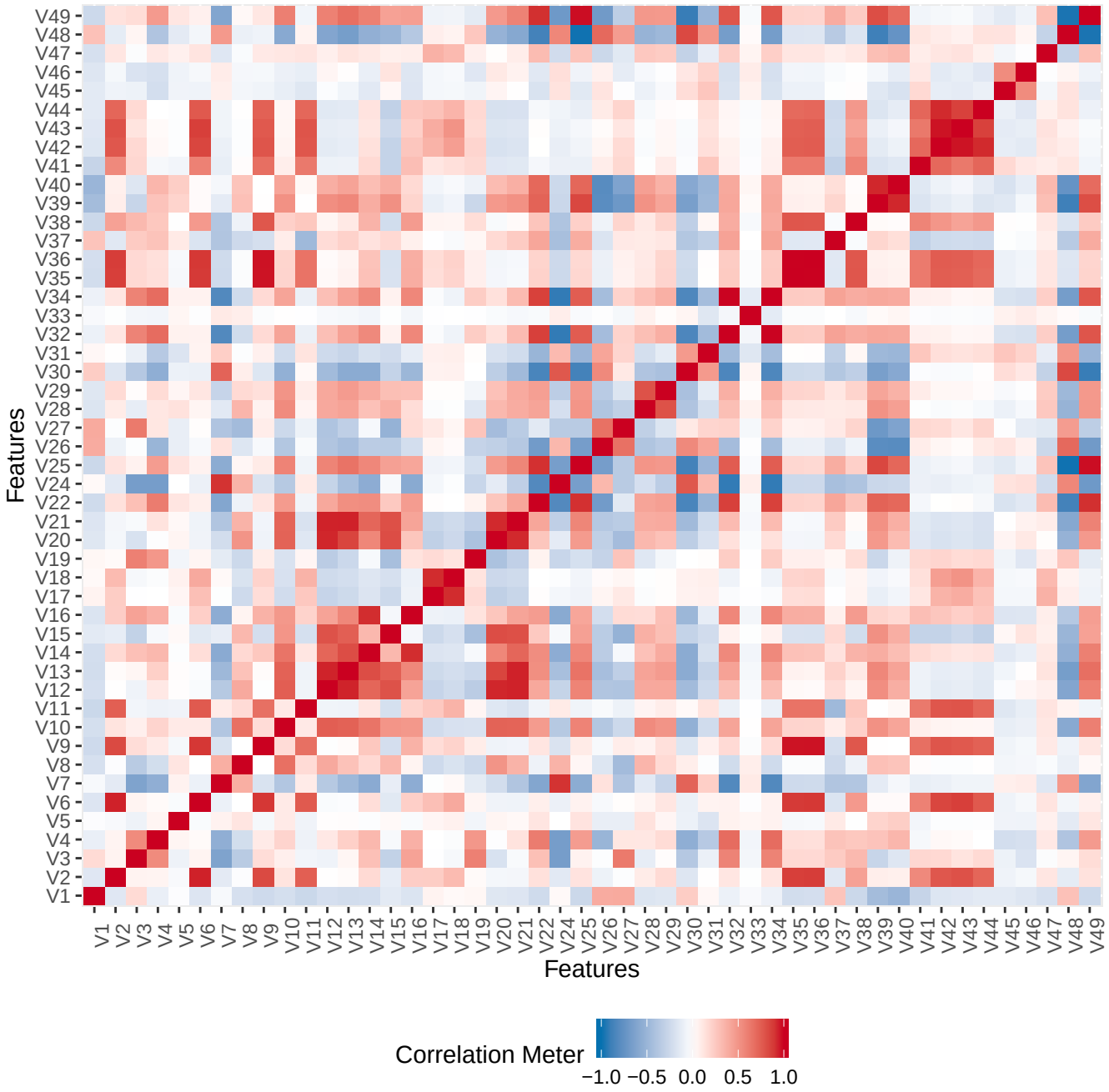
Se ve que el comando `removeConstantFeatures()` identifica un atributo constante, que se elimina.

```
dati<- removeConstantFeatures(dati)
ncol(dati)
```

```
[1] 49
```

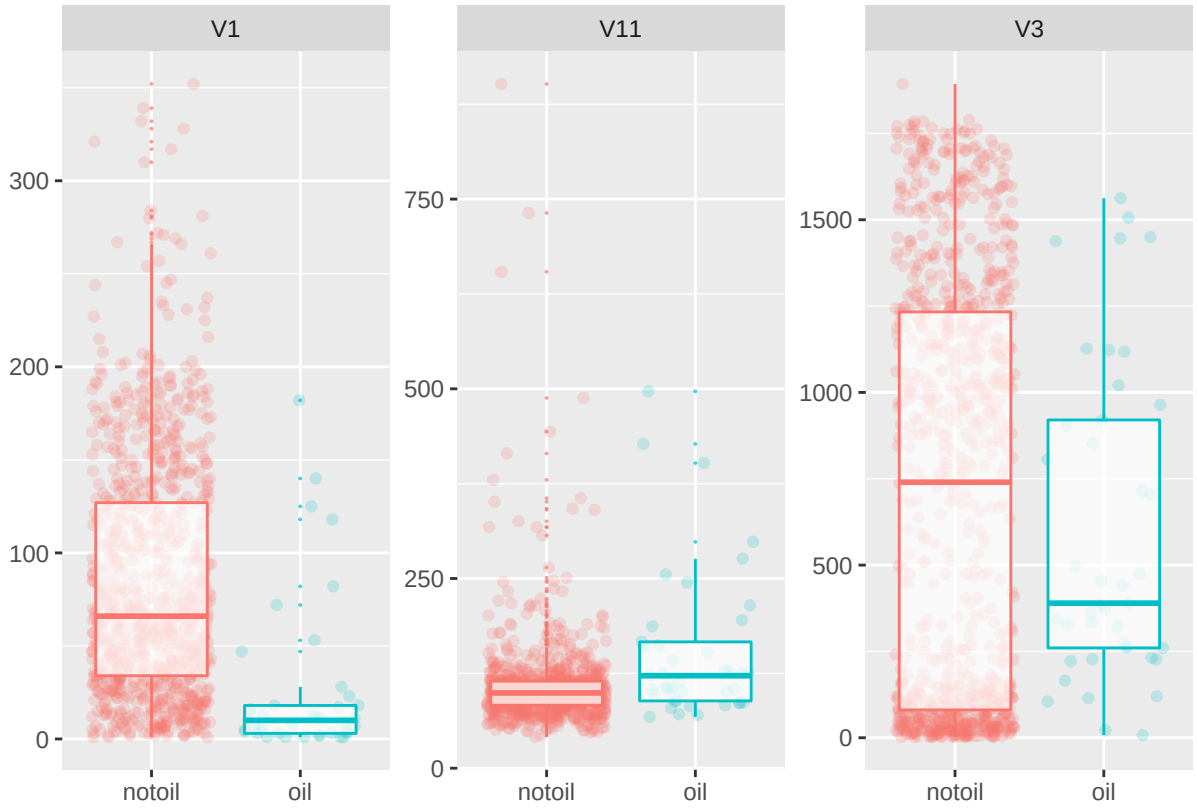
De la matriz de correlación podemos ver que hay variables muy positivamente correlacionadas (`V11`, `V12`, `V13`, `V14`, `V15`), y otras que tienen correlación negativa muy alta (`V48` y `V25`, `V48` y `V49`).

```
dati2<-dati %>% select(-class)
plot_correlation(dati2,maxcat = 5L)
```

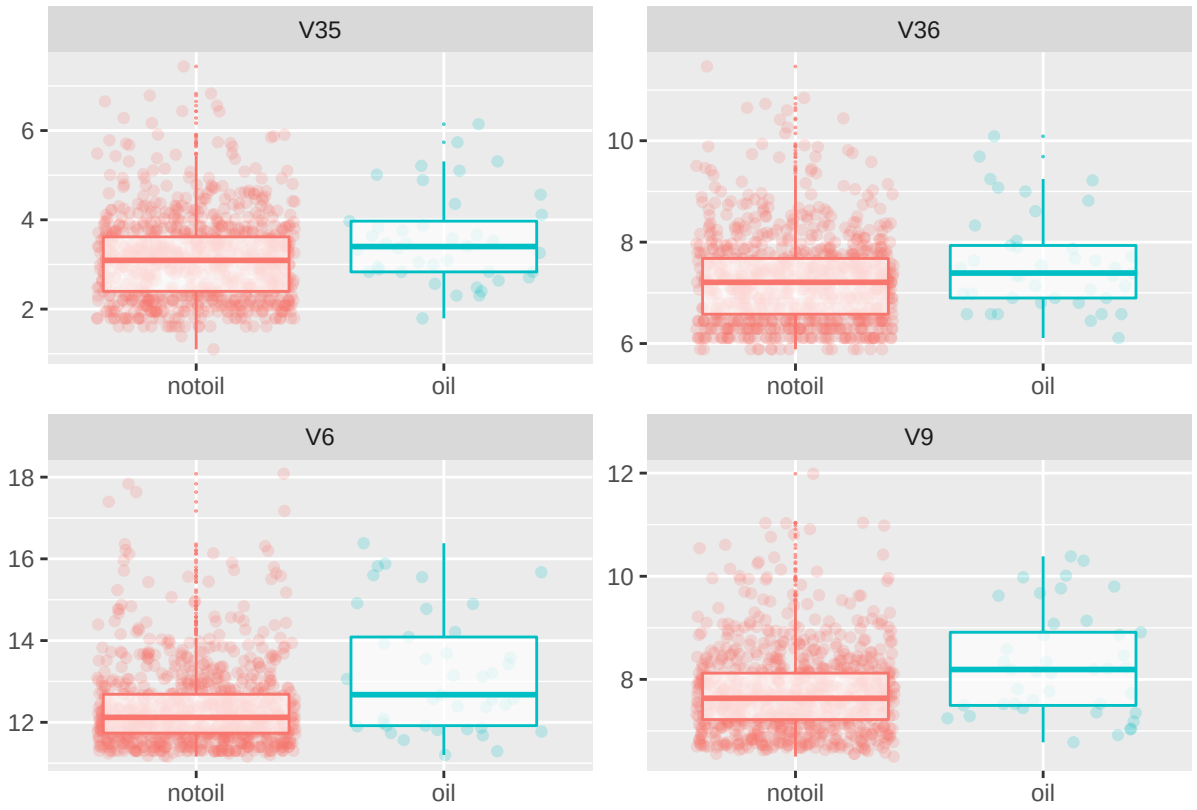


Se muestran algunas de las variables que parecen (tomadas individualmente) tener mayor poder discriminatorio para identificar la clase. Se ha usado el comando `jitter()` para dibujar los puntos para que no se superpongan todos. Los últimos cuatro boxplot se han construido utilizando el logaritmo de "V6", "V9", "V35"y "V36" respectivamente, de modo que los puntos en el eje y no estén demasiado cerca y no se puedan distinguir.

```
g
```


V1
300
200
100
0
notoil
oil
V11
750
500
250
0
notoil
oil
V3
1500
1000
500
0
notoil
oil

g


V35
6
4
2
notoil
oil
V36
10
8
6
notoil
oil
V6
18
16
14
12
notoil
oil
V9
12
10
8
notoil
oil

2 Resultados iniciales

Vamos a comprobar ahora si el desbalanceo de las clases lleva aparejado un mal resultado en la clase minoritaria. Para asegurarnos de que esto es así, vamos a probar con tres métodos distintos: rpart, svm-lineal, y svm-radial, sin ajuste de hiper-parámetros. En la creación de la partición de entrenamiento se utiliza una partición estratificada, de modo que tengamos la misma proporción de observaciones positivas y negativas en ambos conjuntos. Esto se debe a que, especialmente en problemas desbalanzeados, podría ser que por puro azar en la partición de entrenamiento aparezcan muy pocos datos de la clase minoritaria (en el caso extremo ninguno) y, por lo tanto, queremos asegurarnos de que esto no suceda.

```
dati<- normalizeFeatures(dati, target= "class") #problema del data leakage

dati$class <- as.factor(dati$class)
levels(dati$class)=(c(1,2)) # el primer nivel es "notoil"
task_class <- makeClassifTask(data= dati, target="class", positive = 2)

learner_rpart <- makeLearner("classif.rpart")
learner_SVMlineal<- makeLearner("classif.svm", kernel="linear")
learner_SVMradial<- makeLearner("classif.svm")

desc <- makeResampleDesc("Holdout", stratify = T) #default 2/3 para entrenamiento
set.seed(100390180)
particion <- makeResampleInstance(desc, task_class)
```

```
#trivial (para usarlo como benchmark)
#se calcula la performance del class trivial en el conjunto de test
TPR=0
TNR=1
ind<- particion$test.inds[[1]]
datiTrain<- dati[ind, ]
ACC=sum(datiTrain$class==1)/nrow(datiTrain) # medida de error no adecuada para evaluar el modelo
BAC=(TPR+TNR)/2  # medida de error adecuada
trivial <- c( ACC,BAC,TPR,TNR)


#rpart
set.seed(100390180)
errores_rpart <- resample(learner_rpart, task_class, particion, measures = list(acc, bac, tpr, tnr))

calculateConfusionMatrix(errores_rpart$pred, relative = TRUE)$result[1:2, 1:2]
```

```
    predicted
true   1 2
   1 295 4
   2   8 6
```

```
#svmlineal
set.seed(100390180)
errores_SVMlineal <- resample(learner_SVMlineal, task_class, particion, measures = list(acc, bac,
                                                                                        tpr, tnr))

calculateConfusionMatrix(errores_SVMlineal$pred, relative = TRUE)$result[1:2, 1:2]
```

```
    predicted
true   1 2
   1 292 7
   2   9 5
```

```r
#svmradial
set.seed(100390180)
errores_SVMradial <- resample(learner_SVMradial, task_class, particion, measures = list(acc, bac,
                                                                                        tpr, tnr))

calculateConfusionMatrix(errores_SVMradial$pred, relative = TRUE)$result[1:2, 1:2]
```

```
    predicted
true   1 2
   1 298 1
   2  11 3
```

```r
performance_table <- data.frame(Trivial= trivial,Rpart=errores_rpart$aggr,
                                SVM_lineal=errores_SVMlineal$aggr,
                                SVM_radial=errores_SVMradial$aggr,
                                row.names = c("ACC","BAC","TPR","TNR"))
knitr::kable(performance_table, caption = "Comparación de resultados")
```

Table 2: Comparación de resultados

	Trivial	Rpart	SVM_lineal	SVM_radial
ACC	0.9552716	0.9616613	0.9488818	0.9616613
BAC	0.5000000	0.7075968	0.6668657	0.6054706
TPR	0.0000000	0.4285714	0.3571429	0.2142857
TNR	1.0000000	0.9866221	0.9765886	0.9966555

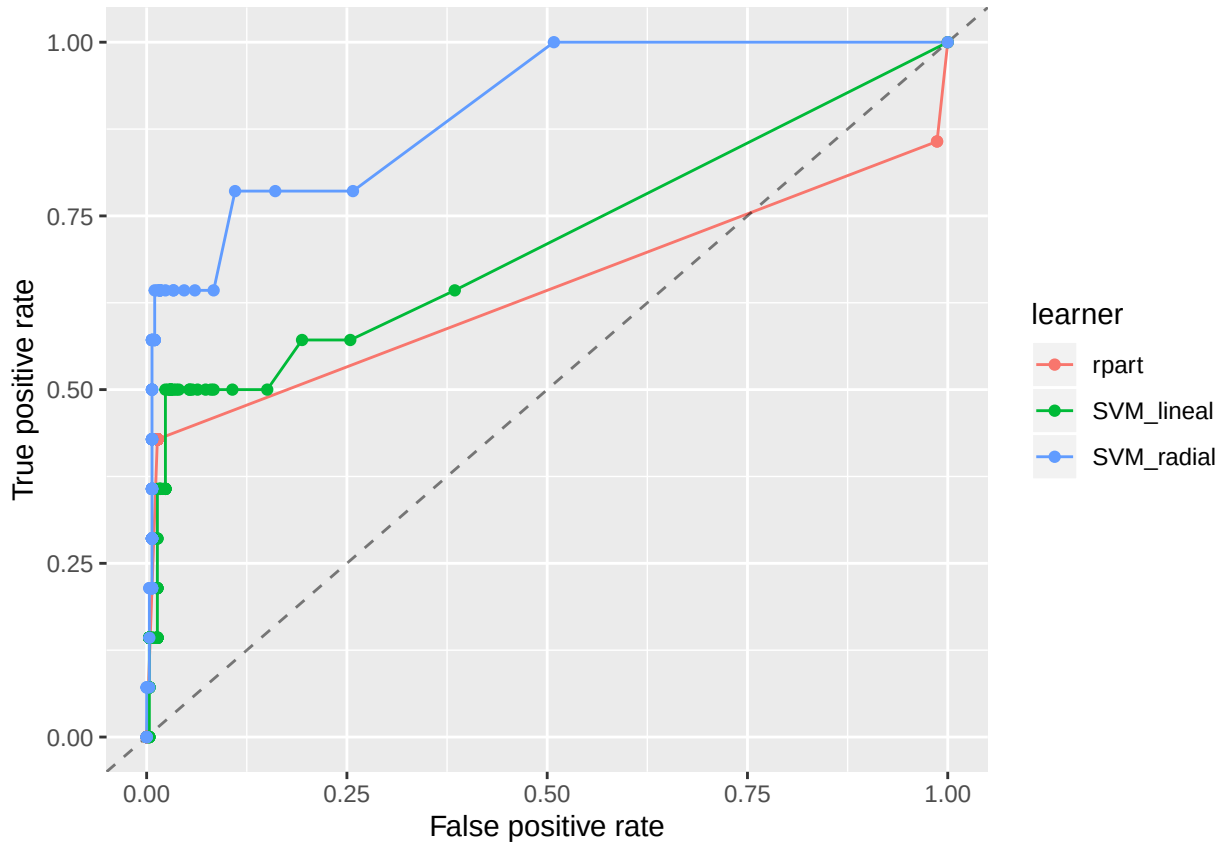
A partir de los resultados de la **Tabla 2**, se puede ver que la *accuracy* no es una buena medida para evaluar un clasificador binario porque se pueden obtener valores muy altos al usar un clasificador trivial que clasifica cada instancia como "nonoil" (clase mayoritaria). En este caso los modelos construidos no consiguen mejorar mucho la

accuracy del clasificador trivial (incluso el metodo *SVM_lineal* lo hace peor que el clasificador trivial). Además, estos tres modelos no consiguen aprender correctamente cómo clasificar elementos de clase minoritarios: de hecho se ve que la clase negativa tiene una tasa de aciertos muy elevada (*TNR*) mientras que la tasa de aciertos en los positivos es muy baja (*TPR*). En el mejor caso en terminos de aciertos de los positivos (*Rpart*) se clasifican correctamente menos que la mitad de los positivos. Se ve tambien que la medida más adecuada para este problema es el *Balanced accuracy (BAC)* porque tiene en cuenta dos valores diferentes que miden el rendimiento del clasificador: el *TPR* y el *TNR*. En este caso el mejor método sería *rpart*. En lugar de mirar directamente al *BAC* (medida de resumen), pueden considerarse conjuntamente ambos *TPR* y *TNR*: en el *rpart* se ve como de hecho se consigue mejorar bastante el *TPR* sim que el *TNR* empeore mucho (con respecto al *SVM_radial* el *TPR* mejora de 0.21 mientras que el *TNR* empeora solo de 0.01).

3 Thresholding

Primero se genera una curva ROC para cada uno de los tres métodos. Se representan en el mismo gráfico para compararlas.

```
#rpart
learner_rpart_prob <- makeLearner("classif.rpart", predict.type = "prob")
errores_rpart_prob <- resample(learner_rpart_prob, task_class, particion,
                               measures = list(acc, bac, tpr, tnr, auc))

set.seed(100390180)
#SVM_lineal
learner_SVM_lineal_prob <- makeLearner("classif.svm",kernel="linear", predict.type = "prob")

set.seed(100390180)

errores_SVM_lineal_prob <- resample(learner_SVM_lineal_prob, task_class, particion,
                                    measures = list(acc, bac, tpr, tnr, auc))

#SVM_radial
set.seed(100390180)
learner_SVM_radial_prob <- makeLearner("classif.svm", predict.type = "prob")

errores_SVM_radial_prob <- resample(learner_SVM_radial_prob, task_class, particion,
                                    measures = list(acc, bac, tpr, tnr, auc))

#comparacion
set.seed(100390180)
df_juntas <- generateThreshVsPerfData(list(rpart=errores_rpart_prob,
          SVM_lineal=errores_SVM_lineal_prob, SVM_radial=errores_SVM_radial_prob),
          measures = list(fpr, tpr, tnr, bac))

plotROCCurves(df_juntas, measures=list(fpr, tpr)) + geom_point()
```



A la vista de las curvas ROC de los tres modelos se puede concluir que a través de un ajuste apropiado del parámetro t es posible mejorar el TPR sin empeorar mucho el FPR en el caso del *SVM_radial* y del *SVM_lineal*. De hecho se puede ver, por ejemplo, que en el caso del *SVM_radial* existe un valor del parametro t con el que se consigue un *FPR* de aproximadamente 12% y un *TPR* proximo al 80%. Este resultado es muy diferente a los valores correspondientes de la tabla anterior (el *TPR* mejora mucho mientras que el *TNR*= 1-*FPR* no es mucho peor). Lo mismo puede decirse de la *SVM_lineal* , aunque si en este caso, para no empeorar demasiado la tasa de aciertos en los negativos, es posible alcanzar como máximo un TPR de aproximadamente 60%. En el caso de *rpart*, sin embargo, cuando el parámetro t cambia, solo se pueden lograr dos resultados (excluyendo el caso de los clasificadores triviales). El primero está representado por el punto naranja más a la izquierda, en correspondencia con un valor de FPR cercano a 0 (se sabe de la tabla anterior que el valor exacto es 0.013378). Este es precisamente el mismo valor que se obtiene usando el valor t que viene por defecto. Para otros valores de t, en cambio, es posible alcanzar un *TPR* de aproximadamente el 80%, pero el modelo clasificaría correctamente un porcentaje muy bajo de negativos. El mejor método en general es *SVM_radial* porque es la curva que tiene el área subyacente más grande: esto significa que para (prácticamente) todos los valores de t el modelo puede alcanzar una tasa de aciertos en los positivos y en los negativos que son mayores que las de los otros metodos. Dicho de otra forma, una vez que se fija un valor para *FPR* en el eje x, el modelo *SVM_radial* es el que logra conseguir la mejor tasa de aciertos para aquel porcentaje de *FPR*.

3.1 Ajustando el *threshold*

Para los casos en los que se ha observado que se pueden mejorar los resultados cambiando el threshold, éste se ajusta automáticamente.

```
#svm lineal
a<- learner_SVM_lineal_prob$par.set$pars$cost$default
ps = makeParamSet(makeDiscreteParam("cost", values= a))

control_grid <- makeTuneControlGrid(tune.threshold = TRUE, tune.threshold.args =
                                    list(measure=list(bac)))
evaluacion_grid <- makeResampleDesc("Holdout", stratify = TRUE)
```

```
learner_ajuste_SVM_lineal <- makeTuneWrapper(learner_SVM_lineal_prob,
                              resampling = evaluacion_grid, par.set = ps,
                              control = control_grid, measures = list(bac))

set.seed(100390180)
errores_ajuste_SVM_lineal <- resample(learner_ajuste_SVM_lineal, task_class,
                           particion, measures = list(acc, bac, tpr, tnr),
                           extract = getTuneResult, models = TRUE)

t1<- errores_ajuste_SVM_lineal$models[[1]]$learner.model$opt.result$threshold



#svm radial
ps = makeParamSet(
  makeDiscreteParam("cost", values=a)
)

control_grid <- makeTuneControlGrid(tune.threshold = TRUE, tune.threshold.args =
                                      list(measure=list(bac)))

evaluacion_grid <- makeResampleDesc("Holdout", stratify = TRUE)

learner_ajuste_SVM_radial <- makeTuneWrapper(learner_SVM_radial_prob, resampling =
                evaluacion_grid, par.set = ps, control = control_grid, measures = list(bac))

set.seed(100390180)
errores_ajuste_SVM_radial <- resample(learner_ajuste_SVM_radial, task_class, particion,
            measures = list(acc, bac, tpr, tnr), extract = getTuneResult, models = TRUE)

t2<- errores_ajuste_SVM_radial$models[[1]]$learner.model$opt.result$threshold
```

Resulta que en el caso del *SVM_lineal* el valor resultante para t es 0.0725429 mientras que en el segundo caso es 0.0541677.
Se resumen los resultados de los modelos con ajuste del *threshold* y de los anteriores en la siguiente tabla.

```
performance_table <- data.frame(Trivial= trivial,Rpart=errores_rpart$aggr,
        SVM_lineal=errores_SVMlineal$aggr,SVM_lineal_ajuste =
          errores_ajuste_SVM_lineal$aggr, SVM_radial=errores_SVMradial$aggr,
        SVM_radial_ajuste= errores_ajuste_SVM_radial$aggr,
        row.names = c("ACC","BAC","TPR","TNR"))

knitr::kable(performance_table)
```

	Trivial	Rpart	SVM_lineal	SVM_lineal_ajuste	SVM_radial	SVM_radial_ajuste
ACC	0.9552716	0.9616613	0.9488818	0.9009585	0.9616613	0.9201278
BAC	0.5000000	0.7075968	0.6668657	0.7098662	0.6054706	0.7879838
TPR	0.0000000	0.4285714	0.3571429	0.5000000	0.2142857	0.6428571
TNR	1.0000000	0.9866221	0.9765886	0.9197324	0.9966555	0.9331104

Se consigue mejorar los *TPR* y *BAC* del apartado anterior, sin empeorar demasiado el *TNR*, simplemente ajustando el threshold. En el caso de 'rpart, tambien se ha hecho un ajuste y, como ya se habia comentado al ver la curva ROC, el resultado con el t que viene por defecto es el mejor posible (se dijo que el segundo punto posible de la curva, el naranja en el extremo derecho, corresponde a un clasificador que falla con casi todos los negativos).

4 Usando *SMOTE*

Se usa el método SMOTE para el entrenamiento del modelo, para corregir el problema de las clases desbalanzeadas. El hiper-parámetro más importante de SMOTE es el ratio que queremos usar para sobre muestrear la clase minoritaria (`sw.rate`). Se empieza usando un valor razonable de rate, por ejemplo rate = número datos not-oil/número datos oil.

```
rate <- ((sum(dati$class==1)/sum(dati$class==2)))

learner_smote_rpart = makeSMOTEWrapper(learner_rpart_prob, sw.rate = rate)
learner_smote_SVM_lineal = makeSMOTEWrapper(learner_SVM_lineal_prob, sw.rate = rate)
learner_smote_SVM_radial = makeSMOTEWrapper(learner_SVM_radial_prob, sw.rate = rate)

set.seed(100390180)
errores_smote_rpart <- resample(learner_smote_rpart, task_class, particion,
                                measures = list(acc, bac, tpr, tnr), models = TRUE)
errores_smote_SVM_lineal <- resample(learner_smote_SVM_lineal, task_class, particion,
                                     measures = list(acc, bac, tpr, tnr), models = TRUE)
errores_smote_SVM_radial <- resample(learner_smote_SVM_radial, task_class, particion,
                                     measures = list(acc, bac, tpr, tnr), models = TRUE)
```

Se comparan los resultados con los mejores obtenidos hasta el momento.

```
performance_table <- data.frame(Rpart=errores_rpart$aggr, Rpart_smote= errores_smote_rpart$aggr,
                                SVM_lineal_ajuste=errores_ajuste_SVM_lineal$aggr, SVM_lineal_smote =
                                errores_smote_SVM_lineal$aggr,
                                SVM_radial_ajuste=
                                errores_ajuste_SVM_radial$aggr,
                                SVM_radial_smote=errores_smote_SVM_radial$aggr,
                                row.names = c("ACC","BAC","TPR","TNR"))
knitr::kable(performance_table)
```

	Rpart	Rpart_smote	SVM_lineal_ajuste	SVM_lineal_smote	SVM_radial_ajuste	SVM_radial_smote
ACC	0.9616613	0.9041534	0.9009585	0.9201278	0.9201278	0.9520767
BAC	0.7075968	0.6434544	0.7098662	0.6858576	0.7879838	0.6344959
TPR	0.4285714	0.3571429	0.5000000	0.4285714	0.6428571	0.2857143
TNR	0.9866221	0.9297659	0.9197324	0.9431438	0.9331104	0.9832776

Para el valor que se ha usado de `sw.rate`, tenemos que los modelos con ajuste del *threshold* funcionan mejor para resolver el problema de muestras desbalanzeadas (se pueden comparar o bien mirando el *BAC* o bien razonando en términos de cuánto mejora el TPR, manteniendo bajo control la tasa de empeoramiento de el *TNR*).

Se usa ahora el *MBO* para ajustar automáticamente `sw.rate`.

```
ps = makeParamSet(
  makeDiscreteParam("sw.rate", values = seq(15, 30, 1))
)

evaluacion_grid <- makeResampleDesc("Holdout",  stratify = TRUE)

control_grid <- makeTuneControlMBO(budget =3) #MBO

#rpart
learner_smote_rpart_aj = makeSMOTEWrapper(learner_rpart_prob)
learner_ajuste_smote_rpart <- makeTuneWrapper(learner_smote_rpart_aj, resampling =
                              evaluacion_grid, par.set = ps, control = control_grid,
                              measures = list(bac, tpr, acc, tnr), show.info = FALSE)
```

```
learner_ajuste_smote_rpart <- makeTuneWrapper(learner_smote_rpart_aj, resampling =
                              evaluacion_grid, par.set = ps, control = control_grid,
                              measures = list(bac, tpr, acc, tnr), show.info = FALSE)

set.seed(100390180)

errores_ajuste_smote_rpart <- resample(learner_ajuste_smote_rpart, task_class,
                  particion, measures = list(acc, bac, tpr, tnr),
                  extract = getTuneResult, models=TRUE)

errores_ajuste_smote_rpart$extract
```

```
[[1]]
Tune result:
Op. pars: sw.rate=16
bac.test.mean=0.8512004
```

```
#svm_lineal
learner_smote_SVM_lineal_aj = makeSMOTEWrapper(learner_SVM_lineal_prob)
learner_ajuste_smote_SVM_lineal <- makeTuneWrapper(learner_smote_SVM_lineal_aj,
                            resampling = evaluacion_grid, par.set = ps, control = control_grid,
                           measures = list(bac, tpr, acc, tnr), show.info = FALSE)
set.seed(100390180)
errores_ajuste_smote_SVM_lineal <- resample(learner_ajuste_smote_SVM_lineal, task_class,
          particion, measures = list(acc, bac, tpr, tnr), extract = getTuneResult, models=TRUE)

errores_ajuste_smote_SVM_lineal$extract
```

```
[[1]]
Tune result:
Op. pars: sw.rate=28
bac.test.mean=0.9042434
```

```
#svm_radial

learner_smote_SVM_radial_aj = makeSMOTEWrapper(learner_SVM_radial_prob )
learner_ajuste_smote_SVM_radial <- makeTuneWrapper(learner_smote_SVM_radial_aj,
                            resampling = evaluacion_grid, par.set = ps, control = control_grid,
                           measures = list(bac, tpr, acc, tnr), show.info = FALSE)
set.seed(100390180)
errores_ajuste_smote_SVM_radial <- resample(learner_ajuste_smote_SVM_radial, task_class,
              particion, measures = list(acc, bac, tpr, tnr), extract = getTuneResult, models=TRUE)

errores_ajuste_smote_SVM_radial$extract
```

```
[[1]]
Tune result:
Op. pars: sw.rate=16
bac.test.mean=0.6490787
```

```
performance_table <- data.frame(Rpart_AJsmote=errores_ajuste_smote_rpart$aggr,
                              SVM_lineal_AJsmote=errores_ajuste_smote_SVM_lineal$aggr ,
                     SVM_radial_AJsmote= errores_ajuste_smote_SVM_radial$aggr,
                     row.names = c("ACC","BAC","TPR","TNR"))
knitr::kable(performance_table)
```

	Rpart_AJsmote	SVM_lineal_AJsmote	SVM_radial_AJsmote
ACC	0.9009585	0.9201278	0.9552716

	Rpart_AJsmote	SVM_lineal_AJsmote	SVM_radial_AJsmote
BAC	0.6417821	0.6858576	0.6702102
TPR	0.3571429	0.4285714	0.3571429
TNR	0.9264214	0.9431438	0.9832776

En comparación con el caso en el que se usaba un `sw.rate` fijo, si éste se ajusta se obtienen mejores resultados solo en el caso del *SVM_radial* (de 0.6345 a 0.6702), mientras que para los otros dos métodos el BAC permanece casi sin cambios. De todas formas, el *BAC* que consigue el *SVM_radial* sigue siendo peor que el *BAC* de ambos el *SVM_radial* y el *SVM_lineal* cuando se usaba el *thresholding* (respectivamente 0.7098 y 0.7880). Los valores de `sw.rate` que se eligen en cada particion son, en orden, 16, 28, 16.

5 Ajuste de hiperparámetros para el peor método

Por último, además de usar *thresholding* y SMOTE, existe la posibilidad de que se puedan mejorar los resultados en una muestra desbalanceada simplemente ajustando los hiper-parámetros, de la manera habitual. Usando únicamente el peor método que haya funcionado hasta el momento, se hace un ajuste de hiper-parámetros con MBO para optimizar el *BAC*. El peor método entre los tres (versión básica) es el *SVM_radial*.

```
ps = makeParamSet(makeNumericParam("cost", lower = -3, upper = 1,
trafo = function(x) 2^x))

control_grid <- makeTuneControlMBO(budget = 10)
desc_inner <- makeResampleDesc("Holdout", stratify = TRUE)
set.seed(100390180)
learner_ajustePAR_SVMradial <- makeTuneWrapper(learner_SVMlineal,
resampling = desc_inner,
par.set = ps,
control = control_grid,
measures = list(bac))

set.seed(100390180)

errores_ajustePAR_SVM_radial <- resample(learner_ajustePAR_SVMradial,
task_class,
particion,
measures = list(acc, bac, tpr, tnr),
extract = getTuneResult)
errores_ajustePAR_SVM_radial$extract
```

```
[[1]]
Tune result:
Op. pars: cost=0.125
bac.test.mean=0.8308208
```

Ajustar "*cost*" en la fase de aprendizaje del modelo es otra forma de mejorar el clasificador en términos de *BAC*, ya que en este caso el BAC mejora (de en el primer 0.6055 modelo sin ajuste a 0.7043). A continuación se presentan las cuatro medidas que representan una estimación de error de un modelo que, para resolver el problema de clases desbalanzeadas, utiliza un ajuste de hiperparámetros.

```
a<- t(errores_ajustePAR_SVM_radial$measures.test)[2:5,]
knitr::kable(a)
```

	x
acc	0.9552716
bac	0.7042523
tpr	0.4285714
tnr	0.9799331

6 Conclusión

Los resultados de todos los métodos utilizados se presentan en sucesión (la medida de error elegida es el BAC)

```
performance_table <- data.frame(Base= c(errores_rpart$measures.test$bac,
                                        errores_SVMlineal$measures.test$bac,
                                        errores_SVMradial$measures.test$bac),
                                Threshold = c(errores_rpart$measures.test$bac,
                                        errores_ajuste_SVM_lineal$measures.test$bac,
                                        errores_ajuste_SVM_radial$measures.test$bac),
                                SMOTE= c(errores_smote_rpart$measures.test$bac,
                                        errores_smote_SVM_lineal$measures.test$bac,
                                        errores_smote_SVM_radial$measures.test$bac),
                                AJ_SMOTE= c(errores_ajuste_smote_rpart$measures.test$bac,
                                        errores_ajuste_smote_SVM_lineal$measures.test$bac,
                                        errores_ajuste_smote_SVM_radial$measures.test$bac),
                                row.names = c("Rpart","Svm_lineal","Svm_radial"))
knitr::kable(performance_table, caption="BAC para los metodos utilizados")
```

Table 7: BAC para los metodos utilizados

	Base	Threshold	SMOTE	AJ_SMOTE
Rpart	0.7075968	0.7075968	0.6434544	0.6417821
Svm_lineal	0.6668657	0.7098662	0.6858576	0.6858576
Svm_radial	0.6054706	0.7879838	0.6344959	0.6702102