Aprendizaje Automático: Proyecto Final

Óscar López Arcos. 75571640

Problema a resolver

El problema a resolver consiste en predecir si un determinado individuo fallará en el pago del mes próximo. Para generar una respuesta disponemos de un conjunto de datos ("default_of_credit_card_clients") con múltiples casos de diferentes individuos. Estos individuos vienen representados por una serie de características, tales como la cantidad de dinero a pagar, el género, la edad, el estado civil o incluso un historial de los pagos en los últimos meses. Nuestro objetivo será aprender de estos datos de forma que, conociendo únicamente las características previamente explicadas, tengamos la capacidad de predecir una respuesta fiable, acerca de si un individuo fallará o no en el pago del próximo mes.

A continuación, procederemos a estudiar más a fondo el problema y, de acuerdo a nuestras conclusiones, aplicaremos diferentes modelos y técnicas de aprendizaje hasta encontrar alguna satisfactoria.

Primer modelo: RandomForest

Observando las distintas variables de nuestro conjunto de datos, podemos apreciar dos características que nos sugieren aplicar este modelo. En primer lugar, el número de muestras es elevado, requisito indespensable para el correcto funcionamiento del algoritmo. Por otro lado, la existencia de diferentes tipos de variables, tanto categóricas como continuas sin normalizar, hacen que usar RandomForest sea más cómodo, ya que por su propia construcción no necesita un preprocesado de los datos.

Lectura de datos y particionado

Leemos los datos y eliminamos las columnas y filas que definen los nombres de las variables e individuos. Asignamos el nombre "label" a la clase de cada muestra.

```
library("leaps")
library("caret")
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
library("randomForest")
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
datos <- read.table("./datos/default_of_credit_card_clients.csv", quote="\"", sep = ";",
                    comment.char="", stringsAsFactors=FALSE)
datos <- datos[c(-1,-2),-1]
```

```
names(datos) = paste("X",1:ncol(datos), sep="")
names(datos)[ncol(datos)] = "label"
```

Creamos una partición *training* y una *test* con el 70%/30% de los datos respectivamente.

```
set.seed(1)
train_index = createDataPartition(datos$label, p = .70, list = FALSE)
data_train = datos[train_index,]

data_test = datos[-train_index,]
test_labels = data_test$label
data_test = data_test[,-ncol(data_test)]
```

Aplicación del modelo

Aplicamos el modelo del paquete RandomForest, estableciendo un máximo de 100 árboles.

```
rf = randomForest( as.factor(label) ~ ., data = data_train, ntree = 100)
```

Obtengo las predicciones y el Error

Predecimos los valores que nuestro clasificador predeciría para la propia muestra *train*, obteniendo así el E_{in}:

```
train_labels_predicted_rf = predict(rf, data_train[-ncol(data_train)])
errorTrainModelo_rf = mean(data_train$label != train_labels_predicted_rf)
errorTrainModelo_rf
```

```
## [1] 0.007285367
```

La clasificación dentro del *train* ha sido casi total. Veamos cómo actúa nuestro clasificador para el *test*

```
test_labels_predicted_rf = predict(rf, data_test)
errorTestModelo_rf = mean(test_labels != test_labels_predicted_rf)
errorTestModelo_rf
```

```
## [1] 0.1831315
```

Esta diferencia puede deberse a la facilidad de sobreajuste de los árboles cuando hay ruido en los datos.

Curva ROC

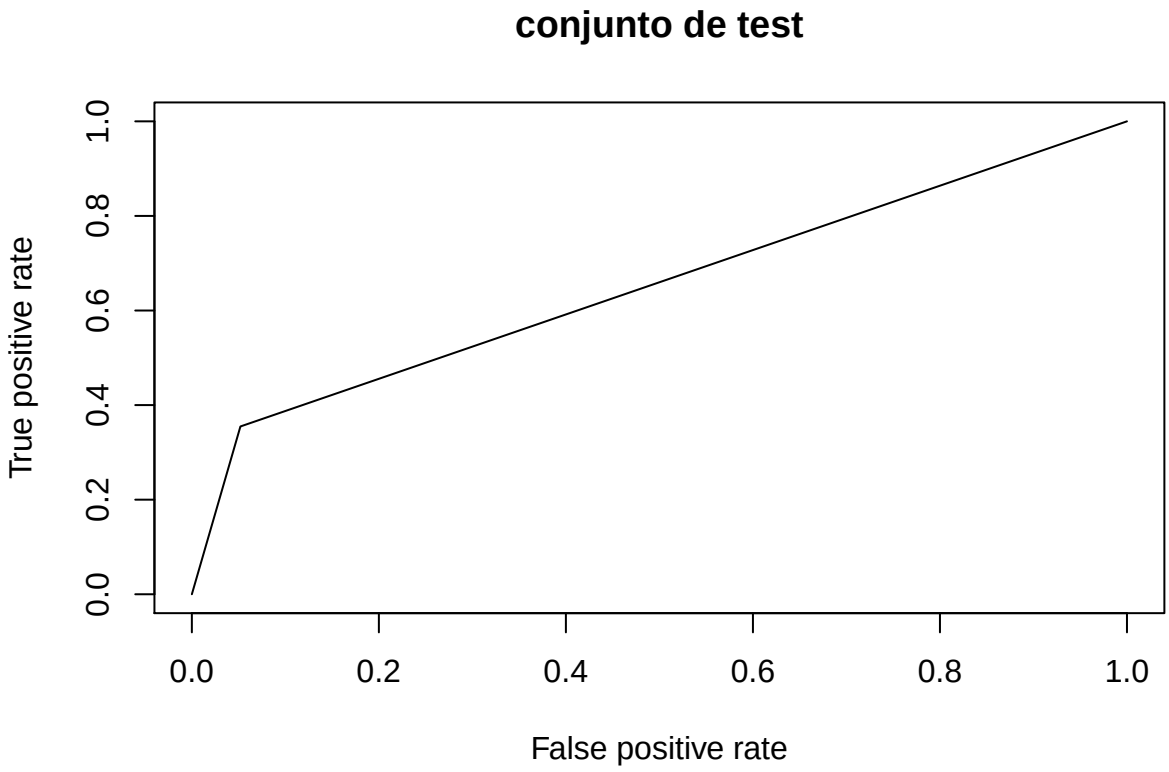
```
library("ROCR")
```

```
## Loading required package: gplots
```

```
##
## Attaching package: 'gplots'
```

```
## The following object is masked from 'package:stats':
##
##     lowess
```

```
pred1 = prediction(as.numeric(test_labels_predicted_rf), as.numeric(test_labels))
perf1 = performance(pred1,"tpr","fpr")
plot(perf1, main="conjunto de test")#,add=TRUE) # pinta la curva
```



Support Vector Machine

Para la aplicación de este modelo basado en kernels, sí necesitamos que los datos hayan sido preprocesados para poder trabajar con ellos, a diferencia de ens RandomForest.

Preprocesado

Lo primero de todo, debemos especeficar aquellas características que actuan como factor (categóricas) y las que actuan como número (numéricas)

```
library("kernlab")
```

```
##
## Attaching package: 'kernlab'
```

```
## The following object is masked from 'package:ggplot2':
##
##     alpha
```

```
# Variables categóricas
for(i in 2:11){
  datos[,i] = as.factor(datos[,i])
}
datos[,24] = as.factor(datos[,24])

# Variables numéricas
datos[,1] = as.integer(datos[,1])
for(i in 12:23){
```

```
  datos[,i] = as.integer(datos[,i])
}
```

Una vez hecho esto, volvemos a separar los datos en *training* y *test*

```
set.seed(1)
train_index = createDataPartition(datos$label, p = .70, list = FALSE)
data_train = datos[train_index,]

data_test = datos[-train_index,]
test_labels = data_test$label
data_test = data_test[,-ncol(data_test)]
```

A continuación, comenzamos eliminando aquellos atributos (columnas) cuya varianza sea 0 o próxima a este número. Estos datos no son relevantes ya que, al ser bastante similares para todos los casos, no aportan información o ésta es ínfima respecto a la magnitud del problema.

```
eliminarCaracteristicas <- function(conjunto,columnasABorrar){
  conjunto = conjunto[-columnasABorrar]
  conjunto
}
caracInutiles = nearZeroVar(data_train)
if(length(caracInutiles) > 0){
  training = eliminarCaracteristicas(data_train, caracInutiles)
  test = eliminarCaracteristicas(data_test, caracInutiles)
}
```

Posteriormente, aplicando el método *preProcess* ajustamos los datos para poder trabajar con ellos.

```
ObjetoTrans =  preProcess(data_train, method = c("YeoJohnson", "center", "scale", "pca"),
                          thres=0.8)
data_train = predict(ObjetoTrans,data_train)
data_test = predict(ObjetoTrans,data_test)
```

Los cuatro métodos utilizados para preprocesar son:

-YeoJohnson: Extensión del método Box-Cox que funciona mejor cuando hay variables negativas o de valor 0 (como en nuestro caso). Su función consiste en corregir sesgos en la distribución de errores, corregir varianzas desiguales (para diferentes valores de la variable predictora) y principalmente para corregir la no linealidad en la relación (mejorar correlación entre las variables).
-center: Resta a cada variable predictora la media de éstas, haciendo que la media de los predictores sea igual a 0 y el valor de la intersección tenga sentido puesto que el 0 está en el rango.
-scale: Divide cada variable predictora por la desviación típica de éstas. Junto al método anterior, se encargan de normalizar el conjunto ($z = \frac{x-\mu}{\sigma}$)
-pca: Preprocesado con Análisis Principal de Componentes. Reduce el número de predictores para explicar los datos, reduce la media de ruido.

Aplico el modelo

```
svm = ksvm( label ~ ., data = data_train, type="C-svc")
```

Obtengo las predicciones y el Error

Calculo las clases predecidas para el *training* y el E_{in}

```r
train_labels_predicted_svm = predict(svm, data_train)
errorTrainModelo_svm = mean(data_train$label != train_labels_predicted_svm)
errorTrainModelo_svm
```

```
## [1] 0.1727061
```

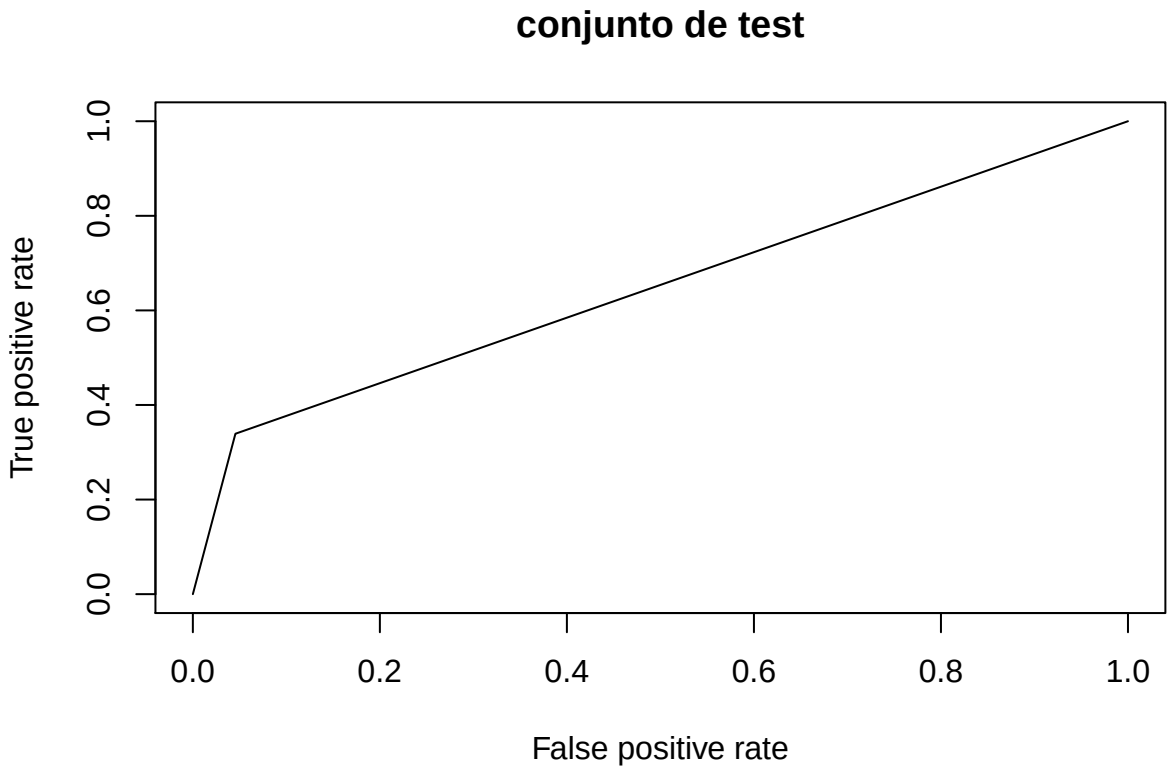
Calculo el E_{out}

```r
test_labels_predicted_svm = predict(svm, data_test)
errorTestModelo_svm = mean(test_labels != test_labels_predicted_svm)
errorTestModelo_svm
```

```
## [1] 0.1815757
```

Curva ROC

```r
pred2 = prediction(as.numeric(test_labels_predicted_svm), as.numeric(test_labels))
perf2 = performance(pred2,"tpr","fpr")
plot(perf2, main="conjunto de test")#,add=TRUE) # pinta la curva
```



Último modelo: Regresión Logística

Por último, veamos el resultado que ofrece un modelo lineal en comparación a los utilizados anteriormente. Aplicaremos una regresión logística y utilizaremos el mismo preprocesado que para SVM.

Aplico el modelo con validación cruzada

```
ctrol = trainControl(method="cv", number=5)
lg = train( as.factor(label) ~ . , data=data_train, method ="glm", trControl = ctrol)
```

Obtengo las predicciones y el Error

Calculo E_{in}

```
fit = predict(lg, data_train, type = "prob")
etiquetasPred = rep(0, length(data_train$label))
etiquetasPred[fit[,1]<fit[,2]] = 1
errorTrainModelo = mean(data_train$label != etiquetasPred);errorTrainModelo
```

```
## [1] 0.1768011
```

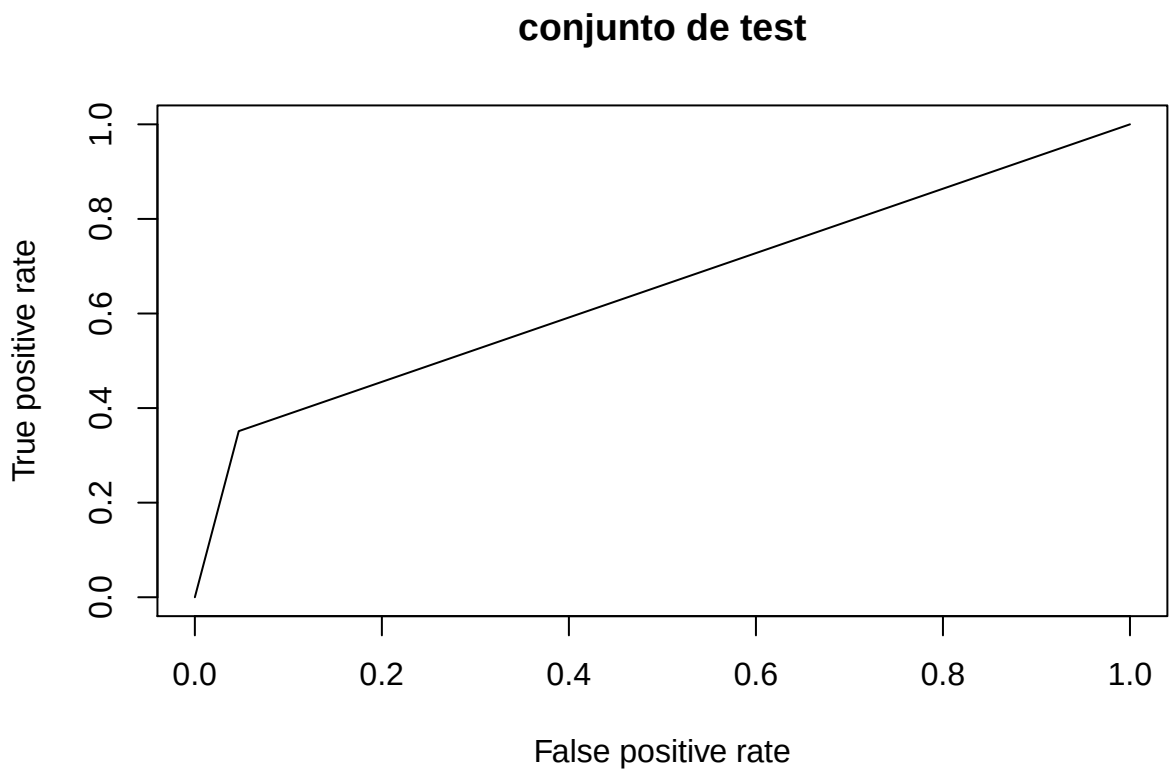
Calculo E_{out}

```
fit = predict(lg, data_test, type = "prob")
etiquetasPred = rep(0, length(test_labels))
etiquetasPred[fit[,1]<fit[,2]] = 1
errorTestModelo = mean(test_labels != etiquetasPred);errorTestModelo
```

```
## [1] 0.18002
```

Curva ROC

```
pred3 = prediction(etiquetasPred, as.numeric(test_labels))
plot(performance(pred3,"tpr","fpr"), main="conjunto de test")
```



Conclusión

Finalmente, como se puede observar en las curvas ROC y en los errores E_{in}/E_{out}, los tres modelos generan una predicción muy similar. Con un poco más de experimentación, es cierto que los modelos no-lineales tienen mayor capacidad de aprendizaje y por tanto una probabilidad más elevada de acierto.
Por ejemplo, si en RF limitásemos el número de nodos o la profundidad máxima conseguiríamos reducir el sobreajuste. En SVM, con un mejor estudio de las características, podríamos elegir un kernel distinto o ajustar los parámetros e hiperparámetros hasta que nuestro modelo de predicción presente unos resultados bastante mejores.
Sin embargo, hemos podido comprobar que un modelo lineal genera de igual forma un predictor bastante decente, lo que nos demuestra una vez más que la complejidad del modelo no necesariamente produce mejores resultados, ya que esta característica depende únicamente del problema a abordar (NFL-Theorem).