RANDOM FOREST

Introducción

Uno de los problemas que tienen los árboles de decisión, es lo afectados que se ven en los casos en los que los conjuntos de datos estén sesgados (aspecto realmente fácil de encontrar, y que suele ocurrir la mayor parte de las veces). Para ello, se proponen técnicas más avanzadas, como *Bagging* o *Random Forest*, que intentan controlar este tipo de problemática.

En esta sección, vamos a tratar *Random Forest*. El método de Random Forest es una modificación del método Bagging, utiliza una serie de árboles de decisión, con el fin de mejorar la tasa de clasificación. Se usa como clasificador de clases preestablecidas.

En general, lo que busca esta técnica es descorrelacionar los diversos árboles que son generados por las diferentes muestras de los datos de entrenamiento, y luego simplemente reducir la varianza en los árboles promediándolos.

La idea es construir muchos árboles de tal manera que la correlación entre los árboles sea más pequeña. En nuestro caso, vamos a realizar una predicción acerca de la efectividad del medicamento utilizando esta técnica.

Para ello, primero utilizaremos la técnica, usando como variables para el entrenamiento aquellas que sean etiquetas numéricas. Por tanto, tendremos en cuenta los distintos atributos numéricos que nos dan información acerca de las opiniones del fármaco, y los utilizaremos para realizar dicha clasificación. Estos atributos son *rating*, *effectivenessNumber* y *sideEffectsInverse*. Posteriormente, utilizaremos la técnica para realizar predicciones en cuanto al texto, concretamente, a los comentarios acerca de los beneficios del fármaco.

Uso de Random Forest con las valoraciones

Para ello, lo primero es realizar la lectura de datos, con la posterior selección de las variables que nos interesan. Como ya se comentó en la introducción de este capítulo, estas serán *rating*, *effectivenessNumber* y *sideEffectsInverse*.

Por ello, tras leer los datos, lo primero que vamos a hacer, es crear un nuevo data-frame de la forma específica que necesitamos que estén los datos para esta técnica en concreto. Si consultamos el data-frame original, las variables que necesitamos se corresponde con las columnas número 2, 12 y 19 respectivamente, y estas serán por tanto, las que filtraremos para el nuevo conjunto de datos con el que vamos a trabajar.

```
# Establecemos una semilla para obtener siempre los mismos resultados
set.seed(3)

# Cargamos los datos
datos_train <- read.table("datos_train_preprocesado.csv", sep=",",
                          comment.char="",quote = "\"", header=TRUE)

datos_test <- read.table("datos_test_preprocesado.csv", sep=",",
                         comment.char="",quote = "\"", header=TRUE)

# Nos quedamos con las columnas que nos interesan
# Las columnas que tenemos son: RATING, SIDE_EFFECTS_INVERSE, EFFECTIVENESS_NUMBER
datos_train2 = datos_train[c(2,12,9)]
datos_test2 = datos_test[c(2,12,9)]
```

De nuevo, al igual que hemos realizado en apartados anteriores, debemos adaptar el dataset a la forma que consideremos más conveniente para poder aplicar la técnica. En este caso, realizaremos el mismo preprocesamiento que se llevó a cabo en el apartado *6.2 Agrupaciones en base a puntuaciones*. En resumen, lo que llevamos a cabo en la siguiente sección de código es adecuar el dataset de forma que las columnas y

filas tengan los nombres adecuados, además de mostrar únicamente un valor promedio por fármaco en los casos que tengamos más de alguna opinión de alguno de ellos (Para conocer a más detalle esta parte del procesamiento, consultar la sección mencionada).

```
# Arreglamos el conjunto de datos para poder trabajar con lo que nos interesa

# Obtenemos en una variable todos los nombres de
# fármacos que existen en el conjunto de train
nombres_farmacos_train <- unique(datos_train[,1])

# Hacemos lo mismo para test
nombres_farmacos_test <- unique(datos_test[,1])

# Creamos una matriz (vacía) con tantas filas como fármacos haya, y tantas columnas como
#atributos queramos utilizar. En este caso son 3 columnas porque necesitamos guardar la
#info de "rating", "sideEffectsInverse" y "effectivenessNumber".
datos_procesados_train <- matrix(ncol=3, nrow=length(nombres_farmacos_train))
datos_procesados_test <- matrix(ncol=3, nrow=length(nombres_farmacos_test))


# Primero procesamos TRAIN

# Convertimos la estructura que tiene todos los nombres de los fármacos, en un data-frame,
# para poder trabajar con esta información de manera más cómoda
df_farmacos_train = as.data.frame(nombres_farmacos_train)

# Para cada uno de los fármacos existentes, vamos a guardar su información asociada en
# la matriz creada anteriormente
for(i in 0:length(nombres_farmacos_train)){

  # Obtenemos todas las filas del dataset que se correspondan con el fármaco número i.
  filas_farmaco_train <- datos_train2[which(datos_train$urlDrugName == nombres_farmacos_train[i]),]

  # Como pueden ser más de una fila, resumimos dicha información.
  mean_rating_train <- mean(filas_farmaco_train$rating)
  mean_side_effect_train <- mean(filas_farmaco_train$sideEffectsInverse)
  mean_effectiveness_train <- mean(filas_farmaco_train$effectivenessNumber)

  # La información resumida es la que asociamos a dicho fármaco, montando la fila de la
  # manera adecuada. Nótese que redondeamos el número medio obtenido para sideEffects y
  # effectivenessNumber. Esto es porque necesitamos que sea un número entero (recordemos
  # que estos dos valores se corresponden con etiquetas).
  datos_procesados_train[i,] <- c(mean_rating_train, round(mean_side_effect_train),
                                  round(mean_effectiveness_train))
}

# Procesamos TEST de forma análoga a TRAIN.
df_farmacos_test = as.data.frame(nombres_farmacos_test)

for(i in 0:length(nombres_farmacos_test)){

  filas_farmaco_test <- datos_test2[which(datos_test$urlDrugName == nombres_farmacos_test[i]),]

  mean_rating_test <- mean(filas_farmaco_test$rating)
```

```
  mean_side_effect_test <- mean(filas_farmaco_test$sideEffectsInverse)
  mean_effectiveness_test <- mean(filas_farmaco_test$effectivenessNumber)

  datos_procesados_test[i,] <- c(mean_rating_test, round(mean_side_effect_test), round(mean_effectivenes

}

# Por último, convertimos las matrices procesadas anteriormente en data-frame
data_train_procesado <- data.frame(datos_procesados_train)
data_test_procesado <- data.frame(datos_procesados_test)

# Una vez creados los data-frame, les asignamos nombres a las filas y columnas de
# los nuevos data-frames.
rownames(data_train_procesado) <- nombres_farmacos_train
rownames(data_test_procesado) <- nombres_farmacos_test
colnames(data_train_procesado) <- c("rating", "sideEffectsInverse", "effectivenessNumber")
colnames(data_test_procesado) <- c("rating", "sideEffectsInverse", "effectivenessNumber")
```

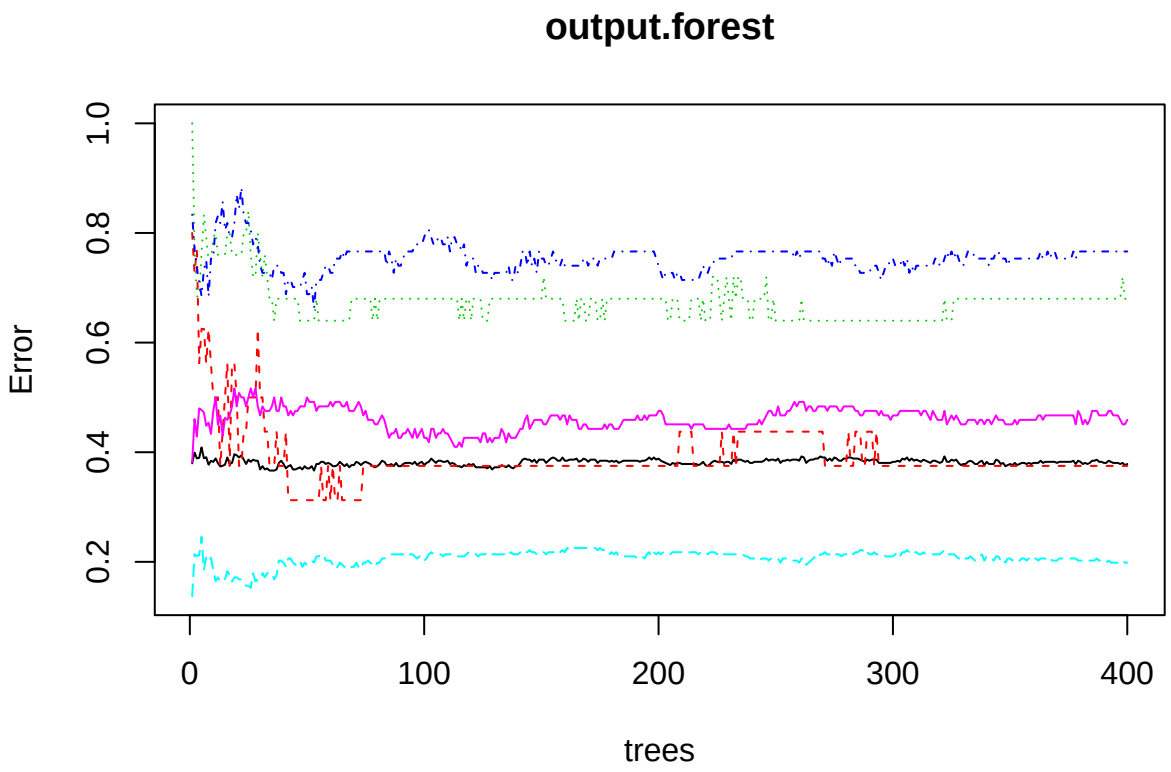
Como primera aproximación, vamos a coger un número alto de árboles (hemos decidido coger 400), y ejecutaremos el algoritmo para ver cómo funciona. Una de las características de la función que hemos utilizado, es que nos proporciona una gráfica, en la cuál podemos consultar las distintas tasas de error para el valor de las etiquetas que intentamos predecir (así como para la tasa de error global), para distintos valores de árboles. Esta información es muy útil, puesto que nos permite ver, de una manera gráfica, a partir de qué número de árboles no compensa seguir entrenando el algoritmo. Sin embargo, viendo la gráfica, los buenos resultados se mantienen al utilizar entre 250 y 400 árboles, sujetos a mayor o menor variación.

```
# Generamos 400 árboles para predecir la efectividad del medicamento
output.forest <- randomForest(as.factor(effectivenessNumber) ~ . ,
              data = data_train_procesado, replace = T,ntree=400)

# Mostramos la matriz de confusión
output.forest
```

```
##
## Call:
##  randomForest(formula = as.factor(effectivenessNumber) ~ ., data = data_train_procesado,      replac
##                Type of random forest: classification
##                      Number of trees: 400
## No. of variables tried at each split: 1
##
##         OOB estimate of  error rate: 37.85%
## Confusion matrix:
##    1 2  3   4  5 class.error
## 1 10 0  4   2  0   0.3750000
## 2  2 8 10   5  0   0.6800000
## 3  2 1 18  55  1   0.7662338
## 4  0 2 23 210 27   0.1984733
## 5  0 0  2  54 66   0.4590164
```

```
# Generamos el gráfico donde se observa la evolución del error en función del número de árboles
plot(output.forest)
```




```
# Generamos las predicciones
mod_rf = predict(output.forest, newdata = data_test_procesado[-3], type="response")

# Calculamos la tasa de error
y.falladas = sum(mod_rf!=data_test_procesado$effectivenessNumber)/length(data_test_procesado$effectiven
print("Error en test:")
```

```
## [1] "Error en test:"
```

```
print(y.falladas)
```

```
## [1] 0.3578275
```

Según esta observación, podemos hacer una funció que nos permita conocer, dentro del rango que queramos, el mejor resultado posible.

A continuación se va a generar una función para calcular la tasa de error en la predicción de la efectividad, teniendo en cuenta el número de árboles, y realizaremos llamadas a dicha función para diferentes rangos de números de árboles.

```
obtener_arboles_optimo = function(x){
  arboles_optimo = 1;
  error_min = 100.0
  for(i in x){

    set.seed(3)
    output.forest <- randomForest(as.factor(effectivenessNumber) ~ . , data = data_train_procesado, repl

    # Predicciones
    mod_rf = predict(output.forest, newdata = data_test_procesado[-3], type="response")
    # Fallo
    y.falladas = sum(mod_rf!=data_test_procesado$effectivenessNumber)/length(data_test_procesado$effecti
```

```
    if(y.falladas < error_min){
      error_min = y.falladas
      arboles_optimo = i
    }
  }

  cat("Numero de árboles óptimo",arboles_optimo, "\n")
  cat("Error minimo conseguido",error_min, "\n")

}
```

Ahora vamos a calcular la tasa de error para un número de árboles total de tamaño 500, y con ayuda de la función, determinaremos el valor con el que conseguimos un resultado óptimo.

```
valores_arboles_500 = seq(from=1, to=500, by=1)

# Ahora se va a realizar las predicciones empleando una serie de tamaños de árboles
# El error mÍnimo con 500 árboles:
obtener_arboles_optimo(x = valores_arboles_500)
```

```
## Numero de árboles óptimo 109
## Error minimo conseguido 0.3514377
```

Como se ha podido observar, con 109 aŕboles llegamos al mejor resultado en el rango que estamos contemplando (alcanzamos un error del 35%), por lo que no compensa añadir mayor esfuerzo computacional. Vemos que si aumentamos el número de árboles el resultado no mejora considerablemente.

Podemos concluir que la **precisión del modelo es del 65%**, siendo un mejor resultado que el obtenido en los árboles de decisión (tal y como era de esperar). Tenemos que tener en cuenta que, a pesar de estar utilizando una técnica potente, nuestro conjunto de datos está sujeto a muchísima subjetividad. Para unas personas, poca efectividad tendrá asociado rating 1 y para otras puede ser rating 3. Esto nos afecta mucho los resultados, y por tanto, aumenta considerablemente el error alcanzado.

Uso de Random Forest con los comentarios de BenefitsReview

De nuevo, lo primero que tenemos que hacer es adecuar el conjunto de datos a nuestras necesidades. Para ello, hemos tenido que hacer lo siguiente:

1. **Procesar el contenido del conjunto de train**, para quitar aquellas columnas con algún caracter extraño, y sobretodo, para eliminar números. También se eliminarán aquellas filas que combinen caracteres alfanuméricos, como es el caso de las columnas 17 y 2585 del conjunto de train. En ambas columnas, se tiene la palabra "800mg", que impide el procesamiento de la técnica. Como seguimos teniendo un conjunto lo suficientemente grande, podemos eliminar estas dos líneas sin que afecte de forma visible a nuestros resultados. Estos cambios, se aplican también a test, para que sea consistente.

```
# Lectura de datos
datos_train = read.csv("datos_train_preprocesado.csv", stringsAsFactors = F)
datos_train <- datos_train[-c(17,2585),]
datos_train$benefits_sin_numeros <-gsub("[0-9]+", "", datos_train$benefits_preprocesado)
datos_train$benefits_sin_rombos <-gsub("\uFFFD", "", datos_train$benefits_sin_numeros)


datos_test = read.csv("datos_test_preprocesado.csv", stringsAsFactors = F)
datos_test <- datos_test[-c(17,2585),]
datos_test$benefits_sin_numeros <-gsub("[0-9]+", "", datos_test$benefits_preprocesado)
```

```
datos_test$benefits_sin_rombos <-gsub("\uFFFD", "", datos_test$benefits_sin_numeros)
datos_test$benefits_sin_rombos <-gsub("alon", "", datos_test$benefits_sin_rombos)
```

2. En esta técnica, pasa igual que con otras técnicas ya comentadas (como Naive Bayes), donde se utilizan las palabras obtenidas en el conjunto de datos, y, a la hora de predecir, todas esas palabras deben existir en el modelo con el que se ha realizado dicha predicción. Esto se debe a que se deben conocer la frecuencia de todas las palabras existentes en nuestro problema para poder llevar a cabo las predicciones, lo que nos ha obligado a fusionar ambos conjuntos, calcular la matriz de frecuencias, y posteriormente separarlos, para poder entrenar y testear el modelo de manera adecuada.

3. Por último, como ya hemos visto en este problema aplicado para árboles de decisión, vamos a coger no un conjunto demasiado grande de palabras (porque como tenemos tantas, al final obtener un camino coherente para las distintas etiquetas no es posible, y termina discriminando para una parte de ellas únicamente), por lo que se ha modificado los tamaños utilizados para una mejor precisión. Como al final, internamente, hay árboles de decisión, se ha optado por utilizar el mismo particionamiento de este conjunto total, ya que vimos que funcionaba relativamente bien con un único árbol de decisión.

```
datos_total <- bind_rows(datos_train,datos_test)

# Quitamos las columnas de texto que no queremos
datos_total <- datos_total[-c(6,7)]
corpus = Corpus(VectorSource(datos_total$benefits_sin_rombos))

# Creamos matriz de términos con las palabras de los comentarios.
# Entonces cada fila, va a estar formada por los comentarios, donde cada palabra es una columna
tdm <- tm::DocumentTermMatrix(corpus)
tdm.tfidf <- tm::weightTfIdf(tdm)
reviews = as.data.frame(cbind(datos_total$effectivenessNumber, as.matrix(tdm.tfidf)))
preparado_train <- reviews[1:1500,]
preparado_test <- reviews[-(1:1500),]
```

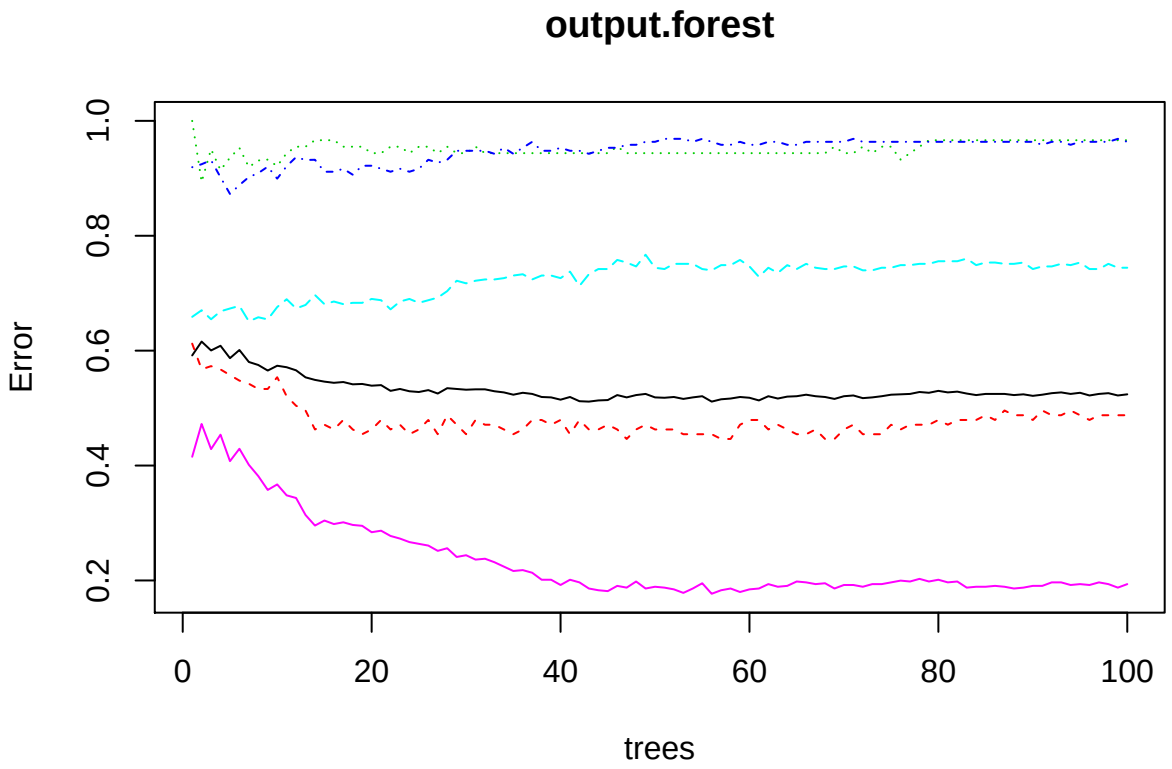
Una vez que hemos tomado estas decisiones, únicamente llamamos al algoritmo con un número de árboles concreto. Por alguna razón, no se cogen bien las variables si no se añade "_c" a todas, por lo que se ha tenido que llevar a cabo esta modificación también.

```
library("randomForest")

colnames(preparado_train) <- paste(colnames(preparado_train), "_c", sep = "")
output.forest <- randomForest(as.factor(V1_c) ~ . , data = preparado_train, replace = T,ntree=100)
output.forest
```

```
##
## Call:
##  randomForest(formula = as.factor(V1_c) ~ ., data = preparado_train,      replace = T, ntree = 100)
##                Type of random forest: classification
##                      Number of trees: 100
## No. of variables tried at each split: 80
##
##         OOB estimate of  error rate: 52.4%
## Confusion matrix:
##    1 2  3   4   5 class.error
## 1 62 0  1  15  43   0.4876033
## 2  9 3  2  26  49   0.9662921
## 3  5 0  7  59 121   0.9635417
## 4  7 0 10 113 312   0.7443439
## 5  6 0  3 118 529   0.1935976
```

```r
plot(output.forest)
```



Una vez hemos entrenado el modelo, falta probarlo en el conjunto de test, que ya habiamos preparado anteriormente. A continuación se lleva a cabo la prueba en el conjunto de test, así como el cálculo del error cometido.

```r
# Predicciones
colnames(preparado_test) <- paste(colnames(preparado_test), "_c", sep = "")
mod_rf = predict(output.forest, newdata = preparado_test[-1], type="response")
# Fallo
datos_test_usados = datos_total[-(1:1500),]
y.falladas = sum(mod_rf!=datos_test_usados$effectivenessNumber)/length(datos_test_usados$effectivenessNu
print("Error en test:")
```

```
## [1] "Error en test:"
```

```r
print(y.falladas)
```

```
## [1] 0.5237192
```

De nuevo, podemos ver que, con un error cometido de 52.3% seguimos superando los resultados que se tienen con árboles de decisión. Este hecho era totalmente previsible, ya que se trata de una técnica potente, y con consolidadas pruebas de que el comportamiento sobre el árbol de decisión clásico es superior.

Aun así, y con todas las mejoras que se llevan a cabo esta técnica, **pensamos que este tipo de algoritmos de clasificación no resultan adecuados en nuestro problema**, debido a dos cuestiones:

1. Tenemos una cantidad muy amplia de datos, y nuestra información se sesga hacia unas etiquetas concretas. Al utilizar todo el conjunto de train, no somos capaces de obtener un árbol con cinco nodos hoja (uno por etiqueta).
2. El hecho de que todas las palabras utilizadas en el conjunto de test tengan que formar parte del conjunto con el que entrenamos (aunque al final no se haga uso de ellas), hace que la técnica sea muy limitada y

de difícil generalización. Por tanto, no vemos adecuado, en nuestro caso, este tipo de técnica, porque ya no es sólo las palabras que no existan, sino las faltas ortográficas y topográficas que puedan tener los usuarios al redactar (y que serían tenidas en cuenta como palabras diferentes en algunos casos).