Interrupciones de la Red de Telecomunicaciones de TELSTRA

Trabajo Final del Programa de Experto en Data Science U-TAD

Rafael Enríquez-Herrador (rafael.enriquez@live-utad.com)

17 de marzo de 2016

Contents

1 Introducción

Tesltra Corporation Limited (TELSTRA) es una de las mayores empresas operadoras de telecomunicaciones de Australia, ofreciendo servicios de telefonía, móvil, acceso a Internet, televisión por cable y otros productos de entretenimiento. Siendo poseedora de una de las mayores redes de telecomunicaciones a nivel global, almacenan gran cantidad de *logs* sobre el funcionamiento y los fallos de ésta. Uno de los principales retos de las operadoras de telecomunicaciones es ofrecer la mejor calidad de servicio a sus clientes, previendo posibles fallos y optimizando la asignación de recursos para el mantenimiento de sus redes de telecomunicaciones. Basándose en los datos aportados por TELSTRA, el objetivo de este proyecto es predecir si una interrupción del servicio será un fallo momentáneo o una interrupción total de la conectividad, según los grados de severidad aportados por la compañía en los *datasets*.

El problema planteado pertenece a una competición organizada por TELSTRA en plataforma *Kaggle*[1] y para su resolución se deberán subir distintas soluciones generadas por los modelos desarrollados, en un formato concreto, que serán evaluadas aplicando la *pérdida logarítmica multi clase*, definida como:

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij} log(p_{ij})$$

2 Estructura del proyecto en R

Para el desarrollo del proyecto, se utilizará de forma exclusiva el lenguaje de programación *R* y las herramientas integradas en el entorno de desarrollo *RStudio*[2].

Toda la información sobre la estructura del proyecto estará contenida en el fichero *TelstraNetworkDisruptions.Rproj* que puede ser abierto fácilmente utilizando *RStudio*. La jerarquía de carpetas se detalla a continuación:

- **data**: Contiene todos los *datasets* que se pueden descargar desde la web de la competición en *Kaggle* en formato `.zip`, así como los ficheros originales descomprimidos en formato `.csv`.
  - **features**: Aquí encontraremos tanto ficheros `.csv` como ficheros `.Rdata` (contenedor de datos utilizado por *R*) con las *features* generadas durante el desarrollo del proyecto y que serán descritos a lo largo de este documento.
- **doc**: Contiene documentos `.pdf` con información relevante sobre el proyecto, como, por ejemplo, este documento.
- **model**: Aquí se podrán encontrar los distintos modelos predictivos generados durante el transcurso del proyecto, en formato válido para ser utilizados en *R*.
- **res**: Esta carpeta contiene tanto ficheros `.csv` con resultados de predicciones generadas por los modelos desarrollados que se han subido a la plataforma *Kaggle* para su puntuación como ficheros `.png` con gráficas que nos ayudarán a entender los pasos seguidos en el análisis de datos y la evaluación de los modelos.
- **src**: Aquí se podrá encontrar todo el código fuente en ficheros `.R` que se ha creado para el desarrollo de este proyecto y es mencionado en este documento.

3 Análisis de los datos

Una vez descargados los *datasets* desde la web de la competición y almacenados en el directorio **data**, pasamos a descomprimirlos. Para ello podemos utilizar el siguiente código que automatiza el proceso, o podemos optar por descomprimirlos manualmente.

[1] https://www.kaggle.com/c/telstra-recruiting-network
[2] https://www.rstudio.com/

```
zipFiles <- dir(path = "data", pattern = "\\.zip")
zipFiles

unzipDataFiles <- function(zipFiles) {
  for (file in zipFiles) {
    unzip(paste(c("data/"), file, sep = ""), exdir = "data")
  }
}

unzipDataFiles(zipFiles)
```

Una vez descomprimidos, nos encontramos con 5 ficheros `.csv` distintos:

- data/log_feature.csv
- data/event_type.csv
- data/resource_type.csv
- data/severity_type.csv
- data/test.csv
- data/train.csv
- data/sample_submission.csv

que pasaremos a estudiar a continuación.

3.1 data/log_feature.csv

Cargamos el *dataset* en memoria:

```
logFeatureData <- read.table(
  file = "data/log_feature.csv",
  header = T,
  sep = ",",
  stringsAsFactors = F)
```

```
head(logFeatureData)
```

```
#     id log_feature volume
# 1 6597  feature 68      6
# 2 8011  feature 68      7
# 3 2597  feature 68      1
# 4 5022 feature 172      2
# 5 5022  feature 56      1
# 6 5022 feature 193      4
```

```
tail(logFeatureData)
```

```
#         id log_feature volume
# 58666 8720  feature 54      1
# 58667 8720 feature 209      1
# 58668 6488  feature 54      3
# 58669  878  feature 62      1
# 58670 4464 feature 209      1
# 58671 4464  feature 87      2
```

```
summary(logFeatureData)
```

```
#        id        log_feature            volume
#  Min.   :    1   Length:58671       Min.   :   1.000
#  1st Qu.: 4658   Class :character   1st Qu.:   1.000
#  Median : 9275   Mode  :character   Median :   2.000
#  Mean   : 9271                      Mean   :   9.685
#  3rd Qu.:13903                      3rd Qu.:   7.000
#  Max.   :18552                      Max.   :1310.000
```

```
str(logFeatureData)
```

```
# 'data.frame': 58671 obs. of  3 variables:
#  $ id         : int  6597 8011 2597 5022 5022 5022 5022 6852 6852 6852 ...
#  $ log_feature: chr  "feature 68" "feature 68" "feature 68" "feature 172" ...
#  $ volume     : int  6 7 1 2 1 4 3 2 1 2 ...
```

```
dim(logFeatureData)
```

```
# [1] 58671     3
```

```
length(unique(logFeatureData$id))
```

```
# [1] 18552
```

```
length(unique(as.factor(logFeatureData$log_feature)))
```

```
# [1] 386
```

```
length(unique(logFeatureData$volume))
```

```
# [1] 341
```

Observamos que el *dataset* se compone de 58671 filas y 3 columnas:

- **id**: con 18552 entradas distintas de tipo numérico.
- **log_feature**: con 386 *features* distintas de tipo carácter cuya parte descriptiva es un número.
- **volume**: con 341 entradas distintas de tipo numérico.

3.2 data/event_type.csv

Cargamos el *dataset* en memoria:

```
eventTypeData <- read.table(
  file = "data/event_type.csv",
  header = T,
  sep = ",",
  stringsAsFactors = F)
```

```r
head(eventTypeData)
```

```
#     id    event_type
# 1 6597 event_type 11
# 2 8011 event_type 15
# 3 2597 event_type 15
# 4 5022 event_type 15
# 5 5022 event_type 11
# 6 6852 event_type 11
```

```r
tail(eventTypeData)
```

```
#         id    event_type
# 31165 8955 event_type 11
# 31166 3761 event_type 11
# 31167 8720 event_type 11
# 31168 6488 event_type 11
# 31169  878 event_type 11
# 31170 4464 event_type 11
```

```r
summary(eventTypeData)
```

```
#        id         event_type
#  Min.   :    1   Length:31170
#  1st Qu.: 4625   Class :character
#  Median : 9288   Mode  :character
#  Mean   : 9268
#  3rd Qu.:13915
#  Max.   :18552
```

```r
str(eventTypeData)
```

```
# 'data.frame': 31170 obs. of  2 variables:
#  $ id        : int  6597 8011 2597 5022 5022 6852 6852 5611 14838 14838 ...
#  $ event_type: chr  "event_type 11" "event_type 15" "event_type 15" "event_type 15" ...
```

```r
dim(eventTypeData)
```

```
# [1] 31170     2
```

```r
length(unique(eventTypeData$id))
```

```
# [1] 18552
```

```r
length(unique(as.factor(eventTypeData$event_type)))
```

```
# [1] 53
```

Observamos que el *dataset* se compone de 31170 filas y 2 columnas:

- **id**: con 18552 entradas distintas de tipo numérico.
- **event_type**: con 53 *features* distintas de tipo carácter cuya parte descriptiva es un número.

3.3 data/resource_type.csv

Cargamos el *dataset* en memoria:

```
resourceTypeData <- read.table(
  file = "data/resource_type.csv",
  header = T,
  sep = ",",
  stringsAsFactors = F)
```

```
head(resourceTypeData)
```

```
#     id   resource_type
# 1 6597 resource_type 8
# 2 8011 resource_type 8
# 3 2597 resource_type 8
# 4 5022 resource_type 8
# 5 6852 resource_type 8
# 6 5611 resource_type 8
```

```
tail(resourceTypeData)
```

```
#         id   resource_type
# 21071 8955 resource_type 8
# 21072 3761 resource_type 8
# 21073 8720 resource_type 8
# 21074 6488 resource_type 8
# 21075  878 resource_type 8
# 21076 4464 resource_type 8
```

```
summary(resourceTypeData)
```

```
#        id        resource_type
#  Min.   :    1   Length:21076
#  1st Qu.: 4600   Class :character
#  Median : 9256   Mode  :character
#  Mean   : 9256
#  3rd Qu.:13907
#  Max.   :18552
```

```
str(resourceTypeData)
```

```
# 'data.frame': 21076 obs. of  2 variables:
#  $ id           : int  6597 8011 2597 5022 6852 5611 14838 2588 4848 6914 ...
#  $ resource_type: chr  "resource_type 8" "resource_type 8" "resource_type 8" "resource_type 8" ...
```

```
dim(resourceTypeData)
```

```
# [1] 21076     2
```

```
length(unique(resourceTypeData$id))
```

```
# [1] 18552
```

```
length(unique(as.factor(resourceTypeData$resource_type)))
```

```
# [1] 10
```

Observamos que el *dataset* se compone de 21076 filas y 2 columnas:

- **id**: con 18552 entradas distintas de tipo numérico.
- **resource_type**: con 10 *features* distintas de tipo carácter cuya parte descriptiva es un número.

3.4 data/severity_type.csv

Cargamos el *dataset* en memoria:

```
severityTypeData <- read.table(
  file = "data/severity_type.csv",
  header = T,
  sep = ",",
  stringsAsFactors = F)
```

```
head(severityTypeData)
```

```
#     id   severity_type
# 1 6597 severity_type 2
# 2 8011 severity_type 2
# 3 2597 severity_type 2
# 4 5022 severity_type 1
# 5 6852 severity_type 1
# 6 5611 severity_type 2
```

```
tail(severityTypeData)
```

```
#         id   severity_type
# 18547 8955 severity_type 1
# 18548 3761 severity_type 1
# 18549 8720 severity_type 1
# 18550 6488 severity_type 2
# 18551  878 severity_type 2
# 18552 4464 severity_type 1
```

```
summary(severityTypeData)
```

```
#        id        severity_type
#  Min.   :    1   Length:18552
#  1st Qu.: 4639   Class :character
#  Median : 9276   Mode  :character
#  Mean   : 9276
#  3rd Qu.:13914
#  Max.   :18552
```

```r
str(severityTypeData)
```

```
# 'data.frame': 18552 obs. of  2 variables:
#  $ id           : int  6597 8011 2597 5022 6852 5611 14838 2588 4848 6914 ...
#  $ severity_type: chr  "severity_type 2" "severity_type 2" "severity_type 2" "severity_type 1" ...
```

```r
dim(severityTypeData)
```

```
# [1] 18552     2
```

```r
length(unique(severityTypeData$id))
```

```
# [1] 18552
```

```r
length(unique(as.factor(severityTypeData$severity_type)))
```

```
# [1] 5
```

Observamos que el *dataset* se compone de 18552 filas y 2 columnas:

- **id**: con 18552 entradas distintas de tipo numérico.
- **severity_type**: con 5 *features* distintas de tipo carácter cuya parte descriptiva es un número.

En este punto nos podemos hacer a la idea de que el conjunto de **datasets** nos presenta información para 18552 eventos distintos registrados en los *logs* de TELSTRA.

3.5 data/test.csv

Cargamos el *dataset* en memoria:

```r
testData <- read.table(
  file = "data/test.csv",
  header = T,
  sep = ",",
  stringsAsFactors = F)
```

```r
head(testData)
```

```
#      id     location
# 1 11066 location 481
# 2 18000 location 962
# 3 16964 location 491
# 4  4795 location 532
# 5  3392 location 600
# 6  3795 location 794
```

```
tail(testData)
```

```
#          id      location
# 11166  2356  location 806
# 11167 14806 location 1073
# 11168  1825   location 11
# 11169  2374  location 917
# 11170  7277  location 208
# 11171  9886  location 438
```

```
summary(testData)
```

```
#        id          location
#  Min.   :    2   Length:11171
#  1st Qu.: 4630   Class :character
#  Median : 9316   Mode  :character
#  Mean   : 9284
#  3rd Qu.:13926
#  Max.   :18552
```

```
str(testData)
```

```
# 'data.frame': 11171 obs. of  2 variables:
#  $ id      : int  11066 18000 16964 4795 3392 3795 2881 1903 5245 6726 ...
#  $ location: chr  "location 481" "location 962" "location 491" "location 532" ...
```

```
dim(testData)
```

```
# [1] 11171     2
```

```
length(unique(testData$id))
```

```
# [1] 11171
```

```
length(unique(as.factor(testData$location)))
```

```
# [1] 1039
```

Observamos que el *dataset* de *test* se compone de 11171 filas y 2 columnas:

- **id**: con 11171 entradas distintas de tipo numérico.
- **location**: con 1039 localizaciones distintas de tipo carácter que, al igual que los *datasets* anteriores se muestran con la etiqueta *location* y un número descriptivo.

3.6 data/train.csv

Cargamos el *dataset* en memoria:

```r
trainData <- read.table(
  file = "data/train.csv",
  header = T,
  sep = ",",
  stringsAsFactors = F)
```

```r
head(trainData)
```

```
#      id     location fault_severity
# 1 14121 location 118              1
# 2  9320  location 91              0
# 3 14394 location 152              1
# 4  8218 location 931              1
# 5 14804 location 120              0
# 6  1080 location 664              0
```

```r
tail(trainData)
```

```
#         id      location fault_severity
# 7376 10455 location 1075              2
# 7377   870  location 167              0
# 7378 18068  location 106              0
# 7379 14111 location 1086              2
# 7380 15189    location 7              0
# 7381 17067  location 885              0
```

```r
summary(trainData)
```

```
#        id          location         fault_severity
#  Min.   :    1   Length:7381        Min.   :0.0000
#  1st Qu.: 4647   Class :character   1st Qu.:0.0000
#  Median : 9222   Mode  :character   Median :0.0000
#  Mean   : 9265                      Mean   :0.4502
#  3rd Qu.:13885                      3rd Qu.:1.0000
#  Max.   :18550                      Max.   :2.0000
```

```r
str(trainData)
```

```
# 'data.frame': 7381 obs. of  3 variables:
#  $ id            : int  14121 9320 14394 8218 14804 1080 9731 15505 3443 13300 ...
#  $ location      : chr  "location 118" "location 91" "location 152" "location 931" ...
#  $ fault_severity: int  1 0 1 1 0 0 0 0 1 1 ...
```

```r
dim(trainData)
```

```
# [1] 7381    3
```

```r
length(unique(trainData$id))
```

```
# [1] 7381
```

```
length(unique(as.factor(trainData$location)))
```

```
# [1] 929
```

```
length(unique(trainData$fault_severity))
```

```
# [1] 3
```

```
length(unique(c(trainData$id, testData$id)))
```

```
# [1] 18552
```

```
length(unique(c(trainData$location, testData$location)))
```

```
# [1] 1126
```

Observamos que el *dataset* de *test* se compone de 7381 filas y 3 columnas:

- **id**: con 7381 entradas distintas de tipo numérico.
- **location**: con 929 localizaciones distintas con el mismo formato que el *dataset* de *test*.
- **fault_severity**: con 3 grados de gravedad de fallo, numerados del 0 al 2.

Si combinamos los *datasets* de *training* y *test* tenemos de nuevo 18552 **id** y hasta 1126 **location** distintas entre los dos.

3.7 data/sample_submission.csv

Cargamos el *dataset* en memoria:

```
sampleSubmissionData <- read.table(
  file = "data/sample_submission.csv",
  header = T,
  sep = ",",
  stringsAsFactors = F)
```

```
head(sampleSubmissionData)
```

```
#      id predict_0 predict_1 predict_2
# 1 11066         0         1         0
# 2 18000         0         1         0
# 3 16964         0         1         0
# 4  4795         0         1         0
# 5  3392         0         1         0
# 6  3795         0         1         0
```

```
tail(sampleSubmissionData)
```

```
#          id predict_0 predict_1 predict_2
# 11166  2356         0         1         0
# 11167 14806         0         1         0
# 11168  1825         0         1         0
# 11169  2374         0         1         0
# 11170  7277         0         1         0
# 11171  9886         0         1         0
```

```
summary(sampleSubmissionData)
```

```
#        id          predict_0   predict_1   predict_2
#  Min.   :    2   Min.   :0   Min.   :1   Min.   :0
#  1st Qu.: 4630   1st Qu.:0   1st Qu.:1   1st Qu.:0
#  Median : 9316   Median :0   Median :1   Median :0
#  Mean   : 9284   Mean   :0   Mean   :1   Mean   :0
#  3rd Qu.:13926   3rd Qu.:0   3rd Qu.:1   3rd Qu.:0
#  Max.   :18552   Max.   :0   Max.   :1   Max.   :0
```

```
str(sampleSubmissionData)
```

```
# 'data.frame': 11171 obs. of  4 variables:
#  $ id       : int  11066 18000 16964 4795 3392 3795 2881 1903 5245 6726 ...
#  $ predict_0: int  0 0 0 0 0 0 0 0 0 0 ...
#  $ predict_1: int  1 1 1 1 1 1 1 1 1 1 ...
#  $ predict_2: int  0 0 0 0 0 0 0 0 0 0 ...
```

```
dim(sampleSubmissionData)
```

```
# [1] 11171     4
```

```
length(unique(sampleSubmissionData$id))
```

```
# [1] 11171
```

En este ejemplo de *submission* se nos indica que los resultados deberán ir agrupados en cuatro columnas:

- **id**: entrada para la que se realiza la predicción.
- **predict_0**: la probabilidad que nuestro modelo calcule para el grado 0 de fallo para la entrada predicha.
- **predict_1**: la probabilidad que nuestro modelo calcule para el grado 1 de fallo para la entrada predicha.
- **predict_2**: la probabilidad que nuestro modelo calcule para el grado 2 de fallo para la entrada predicha.

4 Creando variables

El formato que presenta nuestros ficheros no es manejable para aplicar técnicas predictivas. Cada fichero tiene una longitud distinta y los tipos de datos no aportan linearidad. Por esta razón, el primer paso será crear conjuntos de datos que describan de forma más eficiente y numérica los *logs* aportados, agrupando esta nueva información por **id**.

4.1 Variables estadísticas

En primer lugar vamos a crear **estadísticos** que nos permitan describir los datos de forma más eficiente para cada una de las *features* que aparecen en nuestros *datasets*.

4.1.1 Contador de apariciones de *features*

En primer lugar vamos a generar un nuevo *dataset* con el número de apariciones de *features* distintas definidas en cada *dataset* para cada **id**.

```
require(data.table)

et <- fread(input = "data/event_type.csv")
st <- fread(input = "data/severity_type.csv")
rt <- fread(input = "data/resource_type.csv")
lf <- fread(input = "data/log_feature.csv")

count_et <- et[, .(count_event_type = length(unique(event_type))), by = id]
count_st <- st[, .(count_severity_type = length(unique(severity_type))), by = id]
count_rt <- rt[, .(count_resource_type = length(unique(resource_type))), by = id]
count_lf <- lf[, .(count_log_feat = length(unique(log_feature))), by = id]
count_vol <- lf[, .(count_log_vol = length(unique(volume))), by = id]

tmp <- merge(count_et, count_st, all = T, by = "id")
tmp <- merge(tmp, count_rt, all = T, by = "id")
tmp <- merge(tmp, count_lf, all = T, by = "id")
tmp <- merge(tmp, count_vol, all = T, by = "id")

colnames(tmp) <- c("id", "count_et", "count_st", "count_rt", "count_lf", "count_vol")

write.csv(tmp, file = "data/features/count_feats.csv", quote = F, row.names = F)
```

Este es el nuevo *dataset* generado:

```
#           id count_et count_st count_rt count_lf count_vol
#     1:     1        2        1        2        3         2
#     2:     2        2        1        1        5         1
#     3:     3        1        1        1        1         1
#     4:     4        1        1        1        1         1
#     5:     5        2        1        1        2         2
#    ---
# 18548: 18548        2        1        1        6         2
# 18549: 18549        3        1        1        6         2
# 18550: 18550        1        1        1        2         1
# 18551: 18551        1        1        1        2         2
# 18552: 18552        2        1        1        2         2
```

Observamos que para cada **id**, ahora tenemos:

- Número de **event_type** distintos.
- Número de **severity_type** distintos.
- Número de **resource_type** distintos.
- Número de **log_feature** distintos.
- Número de **volume** distintos.

4.1.2 Suma de *features*

A continuación, sumamos todos los valores de las distintas *features* que tenemos para cada **id**. Parece que el número que describe a las columnas de tipo carácter podría ser importante, por lo tanto, y para diferenciarlo del conteo anterior, separamos el número del resto del texto y lo asignamos como valor de dicha característica para su posterior suma.

```
require(data.table)

splitAndSum <- function(x) {
  sum(as.numeric(unlist(lapply(strsplit(x, " "), function(x) x[2]))))
}

test_train <- rbind(train, test)

et <- fread(input = "data/event_type.csv")
st <- fread(input = "data/severity_type.csv")
rt <- fread(input = "data/resource_type.csv")
lf <- fread(input = "data/log_feature.csv")

sum_et <- et[, .(sum_event_type = splitAndSum(event_type)), by = id]
sum_st <- st[, .(sum_severity_type = splitAndSum(severity_type)), by = id]
sum_rt <- rt[, .(sum_resource_type = splitAndSum(resource_type)), by = id]
sum_lf <- lf[, .(sum_log_feat = splitAndSum(log_feature)), by = id]
sum_vol <- lf[, .(sum_log_vol = sum(volume)), by = id]

tmp <- merge(sum_et, sum_st, all = T, by = "id")
tmp <- merge(tmp, sum_rt, all = T, by = "id")
tmp <- merge(tmp, sum_lf, all = T, by = "id")
tmp <- merge(tmp, sum_vol, all = T, by = "id")

colnames(tmp) <- c("id", "sum_et", "sum_st", "sum_rt", "sum_lf", "sum_vol")

write.csv(tmp, file = "data/features/sum_feats.csv", quote = F, row.names = F)
```

Este es el nuevo *dataset* generado:

```
#           id sum_et sum_st sum_rt sum_lf sum_vol
#     1:     1     24      1     14    592       5
#     2:     2     69      2      2   1408       5
#     3:     3     11      1      8    171       2
#     4:     4     47      4      2    370       3
#     5:     5     69      2      2    544      17
#    ---
# 18548: 18548     69      2      2   1640      22
# 18549: 18549     89      1      2   1419       8
# 18550: 18550     11      1      8    270       6
# 18551: 18551     11      1      8    268       7
# 18552: 18552     69      2      2    544      48
```

Para cada **id**, ahora tenemos:

- Suma de **event_type**.

- Suma de **severity_type**.
- Suma de **resource_type**.
- Suma de **log_feature**.
- Suma de **volume**.

4.1.3 Media de *features*

Siguiendo el principio de particionado de las columnas de tipo carácter y el uso de la parte numérica como valor de la *feature* que utilizamos en el apartado anterior, generamos la media de valores de *features* que tenemos para cada **id**.

```
require(data.table)

splitAndMean <- function(x) {
  mean(as.numeric(unlist(lapply(strsplit(x, " "), function(x) x[2]))))
}

test_train <- rbind(train, test)

et <- fread(input = "data/event_type.csv")
st <- fread(input = "data/severity_type.csv")
rt <- fread(input = "data/resource_type.csv")
lf <- fread(input = "data/log_feature.csv")

mean_et <- et[, .(mean_event_type = splitAndMean(event_type)), by = id]
mean_st <- st[, .(mean_severity_type = splitAndMean(severity_type)), by = id]
mean_rt <- rt[, .(mean_resource_type = splitAndMean(resource_type)), by = id]
mean_lf <- lf[, .(mean_log_feat = splitAndMean(log_feature)), by = id]
mean_vol <- lf[, .(mean_log_vol = mean(volume)), by = id]

tmp <- merge(mean_et, mean_st, all = T, by = "id")
tmp <- merge(tmp, mean_rt, all = T, by = "id")
tmp <- merge(tmp, mean_lf, all = T, by = "id")
tmp <- merge(tmp, mean_vol, all = T, by = "id")

colnames(tmp) <- c("id", "mean_et", "mean_st", "mean_rt", "mean_lf", "mean_vol")

write.csv(tmp, file = "data/features/mean_feats.csv", quote = F, row.names = F)
```

A continuación el *dataset* generado.

```
#           id  mean_et mean_st mean_rt  mean_lf  mean_vol
#     1:     1 12.00000       1       7 197.3333  1.666667
#     2:     2 34.50000       2       2 281.6000  1.000000
#     3:     3 11.00000       1       8 171.0000  2.000000
#     4:     4 47.00000       4       2 370.0000  3.000000
#     5:     5 34.50000       2       2 272.0000  8.500000
#    ---
# 18548: 18548 34.50000       2       2 273.3333  3.666667
# 18549: 18549 29.66667       1       2 236.5000  1.333333
# 18550: 18550 11.00000       1       8 135.0000  3.000000
# 18551: 18551 11.00000       1       8 134.0000  3.500000
# 18552: 18552 34.50000       2       2 272.0000 24.000000
```

No hemos tenido errores al generar el *dataset* (divisiones por cero), por lo que podemos asegurar que cada **id** tiene, al menos una *feature* de cada tipo presente. De esta manera ahora tenemos, para cada **id**:

- Media de **event_type**.
- Media de **severity_type**.
- Media de **resource_type**.
- Media de **log_feature**.
- Media de **volume**.

4.1.4 Desviación estándar de *features*

Aplicamos el mismo procedimiento que en el apartado anterior, pero para obtener la desviación estándar por *feature* e **id**, nos aseguramos de no utilizar la *corrección de Bessel*[3], ya que en algunos casos tenemos que un **id** sólo tiene una *feature* y eso generaría una indeterminación.

```
require(data.table)

SD <- function(x) {
  x_mean <- mean(x)
  sqrt(sum(((x - x_mean)^2)/length(x)))
}

splitAndSD <- function(x) {
  SD(as.numeric(unlist(lapply(strsplit(x, " "), function(x) x[2]))))
}

test_train <- rbind(train, test)

et <- fread(input = "data/event_type.csv")
st <- fread(input = "data/severity_type.csv")
rt <- fread(input = "data/resource_type.csv")
lf <- fread(input = "data/log_feature.csv")

sd_et <- et[, .(sd_event_type = splitAndSD(event_type)), by = id]
sd_st <- st[, .(sd_severity_type = splitAndSD(severity_type)), by = id]
sd_rt <- rt[, .(sd_resource_type = splitAndSD(resource_type)), by = id]
sd_lf <- lf[, .(sd_log_feat = splitAndSD(log_feature)), by = id]
sd_vol <- lf[, .(sd_log_vol = SD(volume)), by = id]

tmp <- merge(sd_et, sd_st, all = T, by = "id")
tmp <- merge(tmp, sd_rt, all = T, by = "id")
tmp <- merge(tmp, sd_lf, all = T, by = "id")
tmp <- merge(tmp, sd_vol, all = T, by = "id")

colnames(tmp) <- c("id", "sd_et", "sd_st", "sd_rt", "sd_lf", "sd_vol")

write.csv(tmp, file = "data/features/sd_feats.csv", quote = F, row.names = F)
```

Aquí el nuevo **dataset** generado:

```
#           id    sd_et sd_st sd_rt     sd_lf    sd_vol
```

[3] https://en.wikipedia.org/wiki/Bessel%27s_correction

```
#     1:     1 1.000000     0     1 113.82540 0.4714045
#     2:     2 0.500000     0     0  38.88239 0.0000000
#     3:     3 0.000000     0     0   0.00000 0.0000000
#     4:     4 0.000000     0     0   0.00000 0.0000000
#     5:     5 0.500000     0     0  40.00000 2.5000000
#    ---
# 18548: 18548 0.500000     0     0  40.01944 0.4714045
# 18549: 18549 6.847546     0     0  59.06705 0.4714045
# 18550: 18550 0.000000     0     0  61.00000 0.0000000
# 18551: 18551 0.000000     0     0  61.00000 2.5000000
# 18552: 18552 0.500000     0     0  40.00000 1.0000000
```

Éste nos proporciona, para cada **id**:

- Desviación estándar de **event_type**.
- Desviación estándar de **severity_type**.
- Desviación estándar de **resource_type**.
- Desviación estándar de **log_feature**.
- Desviación estándar de **volume**.

4.1.5 Mediana de *features*

Aplicamos el mismo procedimiento de *splitting* y obtención de la parte numérica para las *features* necesarias y obtenemos la mediana para cada una de ellas, agrupadas por **id**.

```
require(data.table)

splitAndMedian <- function(x) {
  median(as.numeric(unlist(lapply(strsplit(x, " "), function(x) x[2]))))
}

test_train <- rbind(train, test)

et <- fread(input = "data/event_type.csv")
st <- fread(input = "data/severity_type.csv")
rt <- fread(input = "data/resource_type.csv")
lf <- fread(input = "data/log_feature.csv")

median_et <- et[, .(median_event_type = splitAndMedian(event_type)), by = id]
median_st <- st[, .(median_severity_type = splitAndMedian(severity_type)), by = id]
median_rt <- rt[, .(median_resource_type = splitAndMedian(resource_type)), by = id]
median_lf <- lf[, .(median_log_feat = splitAndMedian(log_feature)), by = id]
median_vol <- lf[, .(median_log_vol = as.double(median(volume))), by = id]

tmp <- merge(median_et, median_st, all = T, by = "id")
tmp <- merge(tmp, median_rt, all = T, by = "id")
tmp <- merge(tmp, median_lf, all = T, by = "id")
tmp <- merge(tmp, median_vol, all = T, by = "id")

colnames(tmp) <- c("id", "median_et", "median_st", "median_rt", "median_lf", "median_vol")

write.csv(tmp, file = "data/features/median_feats.csv", quote = F, row.names = F)
```

El nuevo *dataset* quedaría como:

```
#           id median_et median_st median_rt median_lf median_vol
#     1:     1      12.0         1         7     179.0        2.0
#     2:     2      34.5         2         2     312.0        1.0
#     3:     3      11.0         1         8     171.0        2.0
#     4:     4      47.0         4         2     370.0        3.0
#     5:     5      34.5         2         2     272.0        8.5
#    ---
# 18548: 18548      34.5         2         2     273.5        4.0
# 18549: 18549      34.0         1         2     226.5        1.0
# 18550: 18550      11.0         1         8     135.0        3.0
# 18551: 18551      11.0         1         8     134.0        3.5
# 18552: 18552      34.5         2         2     272.0       24.0
```

Para cada **id** tenemos:

- Mediana de **event_type**.
- Mediana de **severity_type**.
- Mediana de **resource_type**.
- Mediana de **log_feature**.
- Mediana de **volume**.

4.1.6 Máximo de *features*

El siguiente paso es obtener el valor numérico máximo para cada *feature* descrita en los *datasets* para cada **id**. Para obtener este valor, aplicamos el mismo procedimiento que en los apartados anteriores.

```
require(data.table)

splitAndMax <- function(x) {
  max(as.numeric(unlist(lapply(strsplit(x, " "), function(x) x[2]))))
}

test_train <- rbind(train, test)

et <- fread(input = "data/event_type.csv")
st <- fread(input = "data/severity_type.csv")
rt <- fread(input = "data/resource_type.csv")
lf <- fread(input = "data/log_feature.csv")

max_et <- et[, .(max_event_type = splitAndMax(event_type)), by = id]
max_st <- st[, .(max_severity_type = splitAndMax(severity_type)), by = id]
max_rt <- rt[, .(max_resource_type = splitAndMax(resource_type)), by = id]
max_lf <- lf[, .(max_log_feat = splitAndMax(log_feature)), by = id]
max_vol <- lf[, .(max_log_vol = max(volume)), by = id]

tmp <- merge(max_et, max_st, all = T, by = "id")
tmp <- merge(tmp, max_rt, all = T, by = "id")
tmp <- merge(tmp, max_lf, all = T, by = "id")
tmp <- merge(tmp, max_vol, all = T, by = "id")
```

```
colnames(tmp) <- c("id", "max_et", "max_st", "max_rt", "max_lf", "max_vol")

write.csv(tmp, file = "data/features/max_feats.csv", quote = F, row.names = F)
```

Nuestro nuevo *dataset* sería:

```
#           id max_et max_st max_rt max_lf max_vol
#     1:     1     13      1      8    345       2
#     2:     2     35      2      2    315       1
#     3:     3     11      1      8    171       2
#     4:     4     47      4      2    370       3
#     5:     5     35      2      2    312      11
#    ---
# 18548: 18548     35      2      2    315       4
# 18549: 18549     35      1      2    312       2
# 18550: 18550     11      1      8    196       3
# 18551: 18551     11      1      8    195       6
# 18552: 18552     35      2      2    312      25
```

Donde tenemos, para cada **id**:

- Máximo valor de **event_type**.
- Máximo valor de **severity_type**.
- Máximo valor de **resource_type**.
- Máximo valor de **log_feature**.
- Máximo valor de **volume**.

4.1.7 Mínimo de *features*

Finalmente, aplicamos el mismo procedimiento para calcular el mínimo valor de cada *feature* por **id**. Cabe destacar, que, aunque sería aceptable, nos encontraremos con que los resultados nunca serán 0, ya que, como hemos visto anteriormente, cada **id** tiene, al menos una *feature* de cada tipo (*volume*, *event type*, *severity type*, *resource type* y *log feature*)

```
require(data.table)

splitAndMin <- function(x) {
  min(as.numeric(unlist(lapply(strsplit(x, " "), function(x) x[2]))))
}

test_train <- rbind(train, test)

et <- fread(input = "data/event_type.csv")
st <- fread(input = "data/severity_type.csv")
rt <- fread(input = "data/resource_type.csv")
lf <- fread(input = "data/log_feature.csv")

min_et <- et[, .(min_event_type = splitAndMin(event_type)), by = id]
min_st <- st[, .(min_severity_type = splitAndMin(severity_type)), by = id]
min_rt <- rt[, .(min_resource_type = splitAndMin(resource_type)), by = id]
min_lf <- lf[, .(min_log_feat = splitAndMin(log_feature)), by = id]
```

```
min_vol <- lf[, .(min_log_vol = min(volume)), by = id]

tmp <- merge(min_et, min_st, all = T, by = "id")
tmp <- merge(tmp, min_rt, all = T, by = "id")
tmp <- merge(tmp, min_lf, all = T, by = "id")
tmp <- merge(tmp, min_vol, all = T, by = "id")

colnames(tmp) <- c("id", "min_et", "min_st", "min_rt", "min_lf", "min_vol")

write.csv(tmp, file = "data/features/min_feats.csv", quote = F, row.names = F)
```

Al igual que para el máximo, nuestro *dataset* será:

```
#           id min_et min_st min_rt min_lf min_vol
#     1:     1     11      1      6     68       1
#     2:     2     34      2      2    233       1
#     3:     3     11      1      8    171       2
#     4:     4     47      4      2    370       3
#     5:     5     34      2      2    232       6
#    ---
# 18548: 18548     34      2      2    232       3
# 18549: 18549     20      1      2    134       1
# 18550: 18550     11      1      8     74       3
# 18551: 18551     11      1      8     73       1
# 18552: 18552     34      2      2    232      23
```

Para cada **id** tendremos:

- Mínimo valor de **event_type**.
- Mínimo valor de **severity_type**.
- Mínimo valor de **resource_type**.
- Mínimo valor de **log_feature**.
- Mínimo valor de **volume**.

4.2 Componentes principales

El *dataset* más importante en extensión, de todos los aportados es, sin duda, `log_feature.csv`, con sus más de 58000 filas y las columnas *log_feature* y *volume*, de las que, la primera de ellas contiene más de 380 tipos diferentes de valores categóricos posibles.

El *análisis de componentes principales*[4] nos permitirá obtener el valor informativo de estas variables, pero reduciendo su cantidad. Para ello nuestra idea se basa en convertir el *dataset* original en una nueva tabla que tenga por filas cada uno de los **id** y, como columnas, cada una de los valores posibles de *log_feature*. En cada celda de esta nueva tabla tendremos el valor *volume* para dicha intersección.

Una vez calculada la matriz de rotación del *PCA*, multiplicamos (matricialmente) cada fila con los valores de *log_feature* para cada **id**, obteniendo así los valores para cada componente principal.

```
# install.packages(devtools)
library(devtools)
# Visualizacion PCA/clustering
```

[4] https://en.wikipedia.org/wiki/Principal_component_analysis

```r
# install_github("sinhrks/ggfortify")
# install_github("ggbiplot", "vqv")
library(ggbiplot)
# install.packages(corrplot)
library(corrplot)
library(ggplot2)
library(plyr)
require(reshape2)
##library(data.table)
library(dplyr)
require(stringr)
require(data.table)

stringToValue <- function(x) {
  for (i in 1:length(x)) {
    if (is.na(x[i]))
      x[i] <- 0
  }

  return(x)
}

lf.df <- read.table(
  file = "data/log_feature.csv",
  header = T,
  sep = ",",
  stringsAsFactors = F)


lf.df <- dcast(
  data = lf.df,
  formula = id ~ log_feature,
  value.var = "volume")

lf.df[, 2:ncol(lf.df)] <- as.numeric(
  apply(
    lf.df[, 2:ncol(lf.df)],
    2,
    stringToValue))

lf.dt <- as.data.table(lf.df)

lf.df.pca <- prcomp(lf.df[, 2:ncol(lf.df)])
lf.df.pca.var <- lf.df.pca$sdev^2
lf.df.pca.pve <- lf.df.pca.var/sum(lf.df.pca.var)

png(filename = "res/accPCA.png")
qplot(seq_along(lf.df.pca.pve),
      cumsum(lf.df.pca.pve),
      geom = c("point", "path"),
      xlab = "Componentes principales",
      ylab = "Proporción acumulada de Varianzas Explicadas")
dev.off()
```

```r
tmp <- as.matrix(lf.df[, 2:ncol(lf.df)]) %*% lf.df.pca$rotation

tmp <- cbind(lf.df$id, tmp, 1)

colnames(tmp)[1] <- "id"

lf.dt.pca <- as.data.table(tmp)

save(lf.dt.pca, file = "data/features/log_features_pca.Rdata")
```

La nueva tabla de componentes principales por **id** quedará de la siguiente forma:

```
#     id           PC1           PC2           PC3           PC4
#  1:  1 -0.0138270479   0.005395006 -0.0004346765  1.034051e-03
#  2:  2  0.1004979156  -1.286733206  1.2011605274 -3.230296e-02
#  3:  3  0.0004268579   0.002764046 -0.0001958216  8.031253e-04
#  4:  4  0.0002734774   0.001774283 -0.0001264311  5.190394e-04
#  5:  5  0.7886779861 -10.563433502 -2.3906555315 -2.362874e-01
#  6:  6  0.0493769339  -0.665681067 -0.1504726667 -1.527688e-02
#  7:  7  0.0033076106   0.021785777 -0.0015778320  6.818965e-03
#  8:  8  0.3694997462  -4.901206601 -1.1827695663 -1.011987e-01
#  9:  9  0.9273492949   8.269300554 -0.9151839305 -2.987720e+01
# 10: 10  0.0122310881   0.084338547 -0.0064448215  3.438213e-02
#               PC5
#  1:  -0.008786748
#  2:   0.189324028
#  3:  -0.005091055
#  4:  -0.003363982
#  5:   1.391159604
#  6:   0.090200236
#  7:  -0.044586824
#  8:   0.587237785
#  9: 126.052918314
# 10:  -0.233781930
```

Además, como se puede observar en la *Figura 1*, en la que mostramos la proporción acumulada de varianza explicada por componente principal, que las 10 primeras componentes principales describen más del 80 por ciento de la información de nuestro conjunto de datos.

Comprobamos que nuestra nueva tabla tenga las dimensiones adecuadas:

```r
dim(lf.dt.pca)
```

```
# [1] 18552   387
```

Después de aplicar *PCA*, tenemos que la matriz de rotación nos indica la importancia de cada *log_feature* en cada una de las componentes de principales de manera normalizada $[-1, 1]$, siendo aquellas que tengán un valor más próximo a los extremos, las que más información están proporcionando. Si a esto le añadimos que `prcomp` ordena las componentes principales por valor descriptivo, podemos tomar las 15 primeras componentes principales, sumar el valor absoluto del valor de cada una de las *log_feature* en cada una de ellas y ordenar el resultado de mayor a menor. Las *log_feature* mayores serán aquellas que tienen mayor importancia descriptiva en `log_feature.csv` (al menos para las 15 primeras componentes principales).

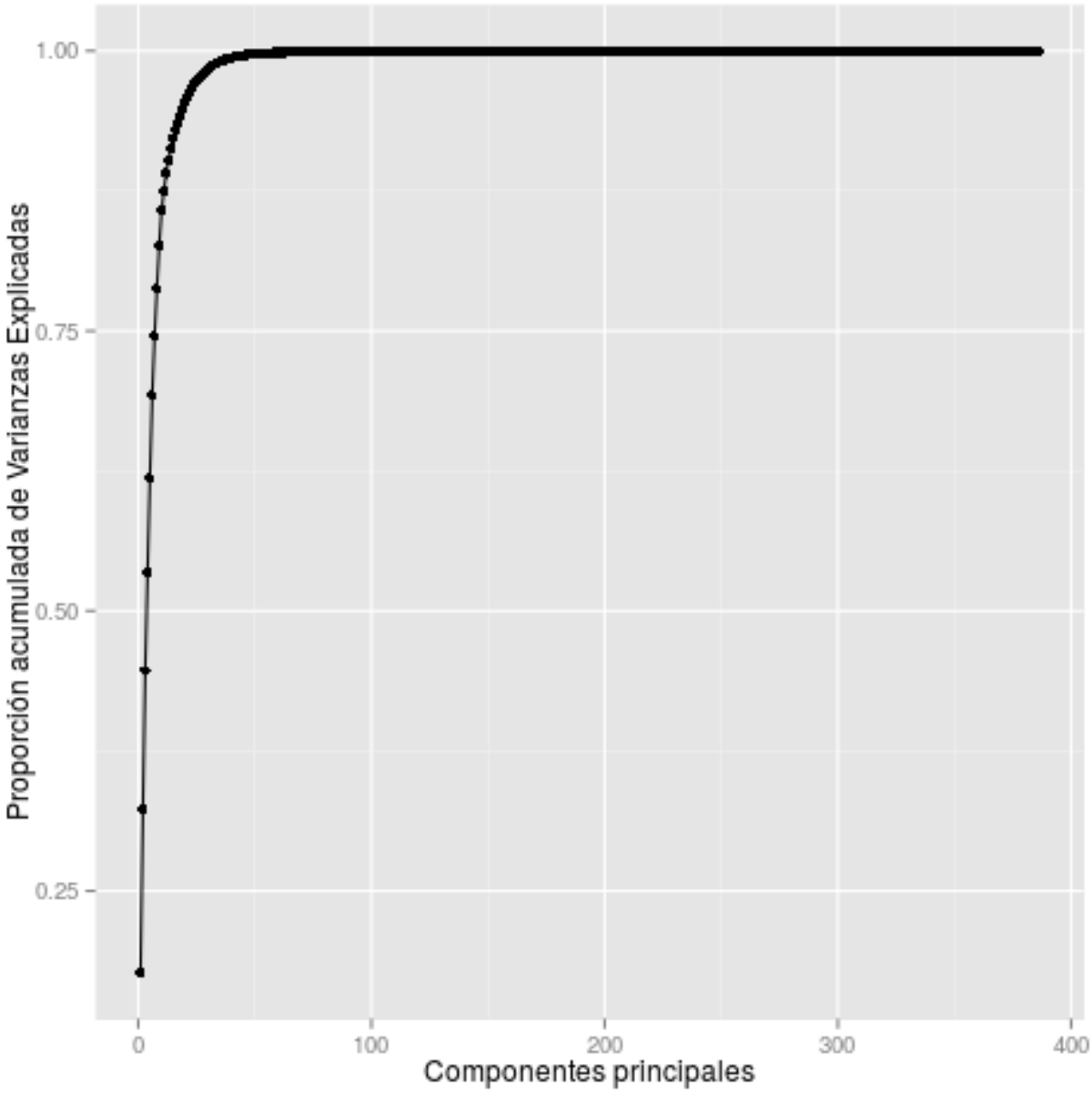

Figure 1: Varianzas acumuladas PCA.

```
tmp <- lf.df.pca$rotation

rownames(tmp) <- as.numeric(
  unlist(lapply(strsplit(rownames(tmp), " "), function(x) x[2])))

tmp <- cbind(as.numeric(rownames(tmp)), tmp)
colnames(tmp)[1] <- "id"

tmp.dt <- as.data.table(tmp)

tmp.dt.sum <- abs(tmp.dt)
tmp.dt.sum$sum <- rowSums(tmp.dt.sum[, 2:6, with = F])
tmp.dt.sum[order(-sum)][1:15, c(1,388), with = F]
```

```
#      id       sum
#  1:  82 1.2127724
#  2:  51 1.0560950
#  3: 315 0.9755371
#  4:  54 0.9677375
#  5: 232 0.9662923
#  6: 312 0.8702236
#  7: 235 0.7803970
#  8: 101 0.6732174
#  9:  56 0.4924128
# 10: 103 0.4922023
# 11: 203 0.3464853
# 12: 313 0.2681106
# 13: 233 0.2611363
# 14: 172 0.2452009
# 15: 170 0.1921970
```

El resultado de ésta operación se muestra en la tabla anterior y, en este proyecto sólo lo utilizaremos como información descriptiva, que nos permitirá decidir que *log_feature* son más descriptivas en nuestro *dataset*.

4.3 Buscando la variable mágica

En los foros de la competición, los participantes hablan de una o varias variables mágicas que se pueden obtener, basadas en el orden inherente a los *datasets* aportados por TELSTRA. Es lógico pensar que si los *datasets* describen los fallos de una red telecomunicaciones basada en localizaciones, también estén estos datos ordenados en el tiempo.

En primer lugar, utilizando los *datasets* de *training* y *test* vamos a intentar comparar cada entrada en el *log* (por **id**) respecto a su localización.

```
require(tm)
require(data.table)
require(ggplot2)
require(plyr)

train <- fread(input = "data/train.csv")
test <- fread(input = "data/test.csv")
```

```
splitLocation <- function(x) {
  as.numeric(unlist(lapply(strsplit(x, " "), function(x) x[2])))
}

test$fault_severity <- -1

test_train <- rbind(train, test)

locations <- test_train[,
                        .(location = splitLocation(location),
                          fault_severity),
                        by = id]
locations <- locations[,
                       .(location,
                         fault_severity,
                         fs_factor = as.factor(fault_severity)),
                       by = id]

png(filename = "res/location_vs_id.png")
ggplot(data = locations, aes(x = location, y = id)) +
  geom_point(aes(color = fs_factor, alpha = fs_factor, size = fs_factor)) +
  scale_color_manual(
    name = "fault_severity",
    values = c("-1"="black", "0"="green", "1"="blue", "2"="red")) +
  scale_alpha_manual(
    name = "",
    values = c("-1"=0.8, "0"=0.7, "1"=0.7, "2"=0.7)) +
  scale_size_manual(values = c("-1"=0.5, "0"=2, "1"=2, "2"=2))
dev.off()
```

Como se puede observar en la *Figura 2*, parece que existe una cierta agrupación de los datos en columnas. La variable *location*, por tanto, tendrá un peso predictivo muy importante en nuestro modelo(s).

Añadimos al conjunto de datos el *dataset* `severity_type.csv` ya que contiene un registro por **id** y no tenemos que prepararlo. Lo importante es no variar el orden de los *datasets* al hacer el *merge*.

A continuación agregamos un contador a cada aparición que aparezca agrupada por *location*. Es decir, con el orden original de los datos, para cada *location* vamos a establecer el orden en el que aparecen los datos originalmente de tres formas distintas:

- **sort**: orden incremental
- **revsort**: orden decremental
- **cumsort**: proporción acumulada de entradas que han aparecido hasta el momento agrupdas por *location*.

```
st <- fread(input = "data/severity_type.csv")
st <- st[, .(severity_type = splitLocation(severity_type)), by = id]

st <- merge(st, locations, all = T, by = "id", sort = F)
head(st, 100)
tail(st, 100)

st[, sort := c(1:.N), by = location]
st[, revsort := c(.N:1), by = location]
```

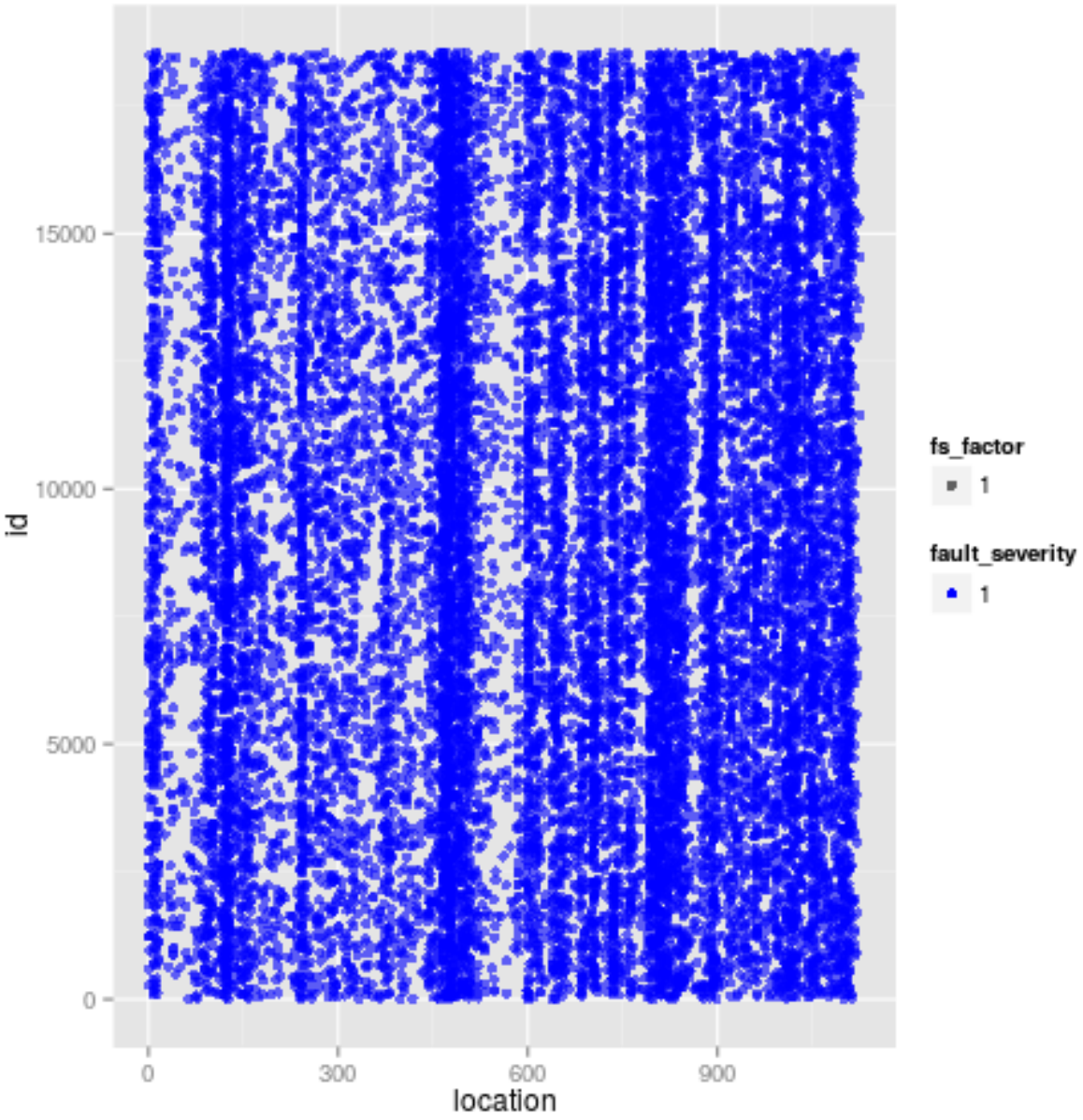

Figure 2: *location* vs *ID*

```
st[, cumsort := (sort/(.N + 1)), by = location]

png(filename = "res/hist_sort.png")
ggplot(data = st, aes(x = sort)) + geom_histogram()
dev.off()
```

En la *Figura 3* mostramos el histograma de las apariciones de cada *sort*. Se puede observar que los valores más bajos de *sort* tienen más apariciones en la tabla y, conforme éste crece, el número de apariciones tiende a 0. Esto se puede interpretar como que muchas *location* presentan registros de fallos al principio, pero conforme avanzamos en el tiempo, estos fallos desaparecen y finalmente, son pocas las que presentan fallos con valor de orden alto.

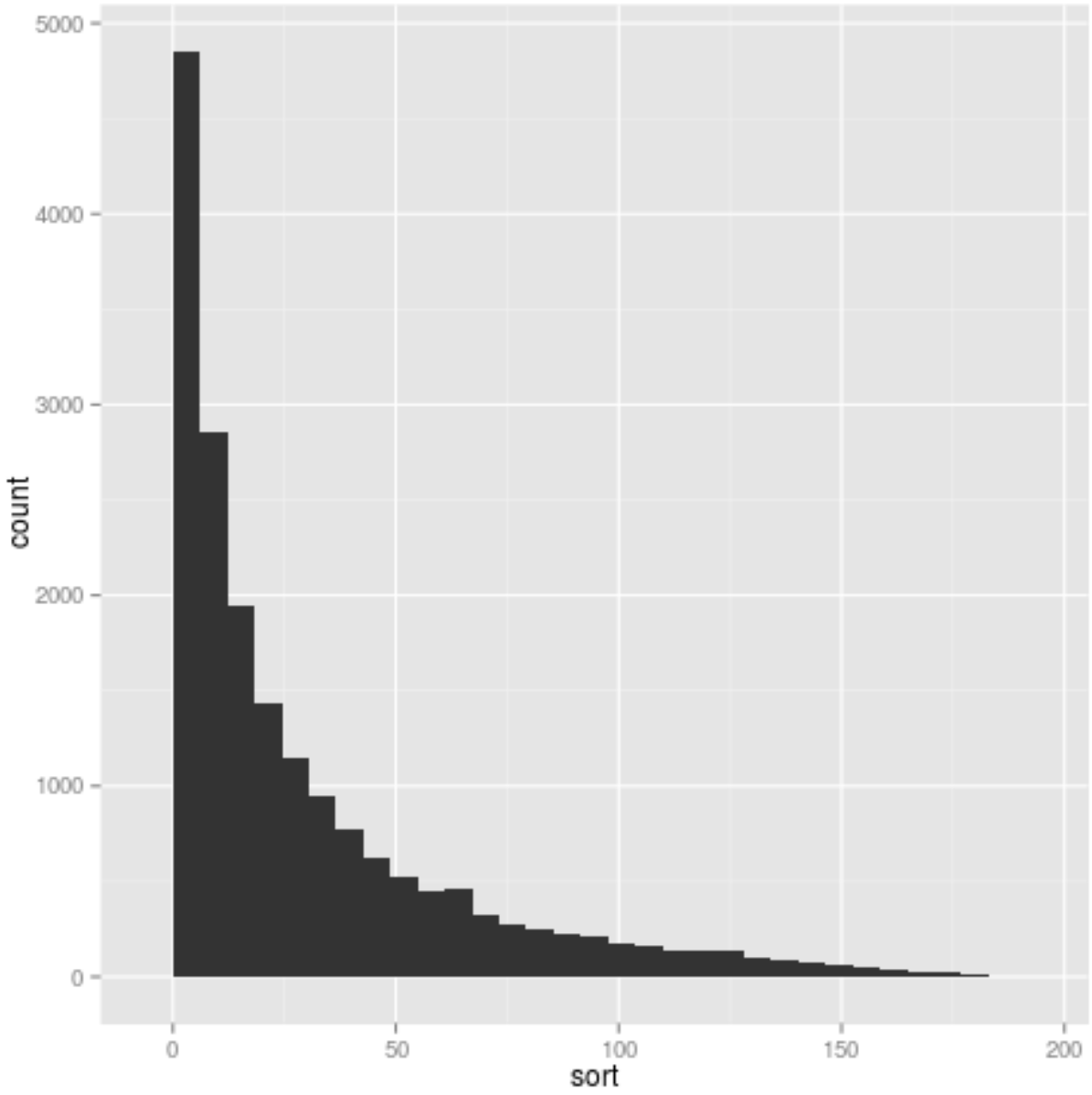

Figure 3: Histograma de la variable *sort*

Realizamos el mismo procedmiento con *revsort* (orden inverso).

```
png(filename = "res/hist_revsort.png")
ggplot(data = st, aes(x = revsort)) + geom_histogram()
dev.off()
```

Aunque los valores más altos están al comienzo de la secuencia, al no haber más de 200 fallos por *location* y debido a la escala, la *Figura 4* se ve igual que la *Figura 3* por la misma explicación anterior.

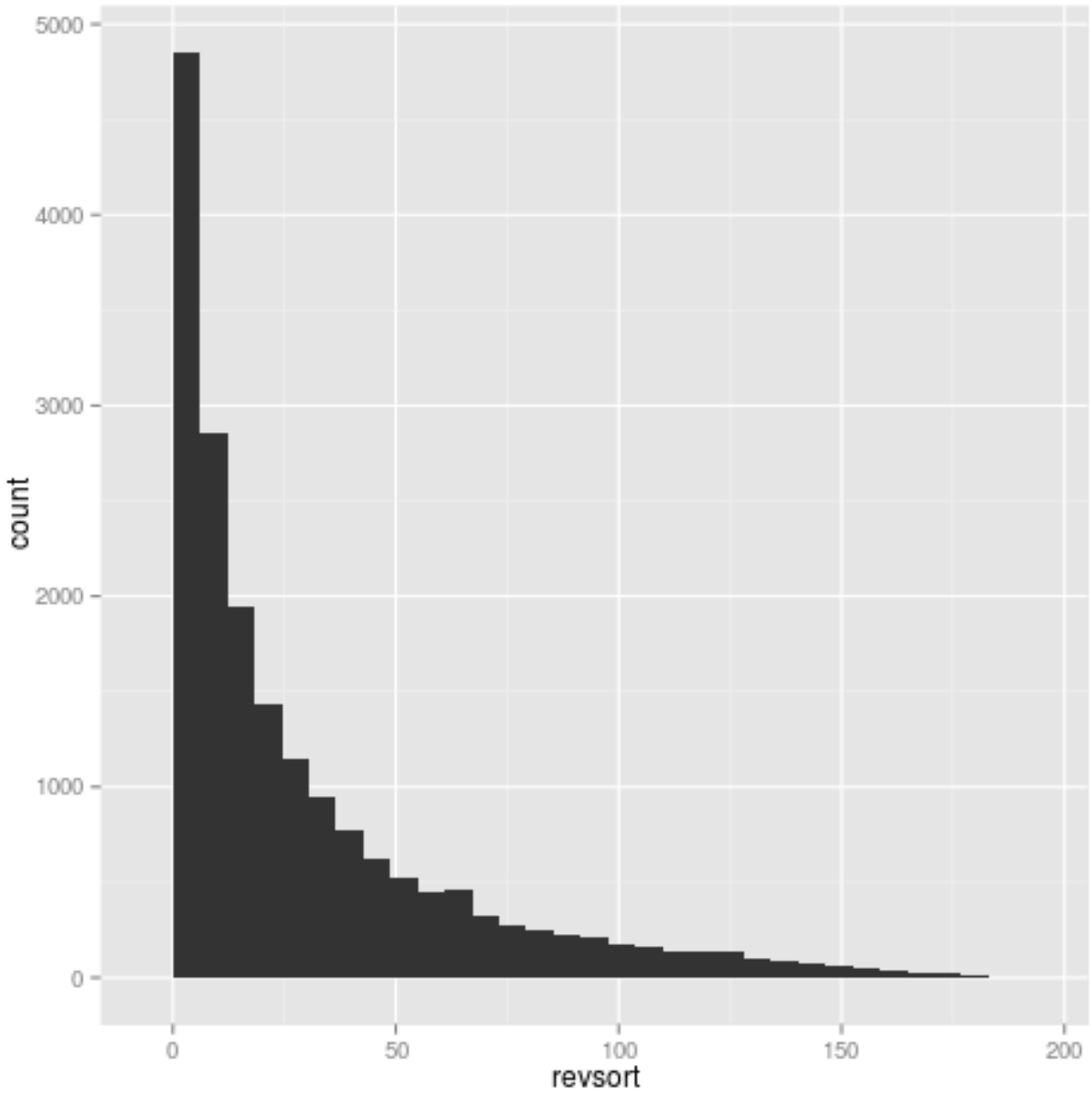

Figure 4: Histograma de la variable *revsort*

Finalmente, mostramos el histograma de *cumsort*.

```
png(filename = "res/hist_cumsort.png")
ggplot(data = st, aes(x = cumsort)) + geom_histogram()
dev.off()
```

Podemos ver en la *Figura 5* que el caso de *cumsort* no es tan explicativo.

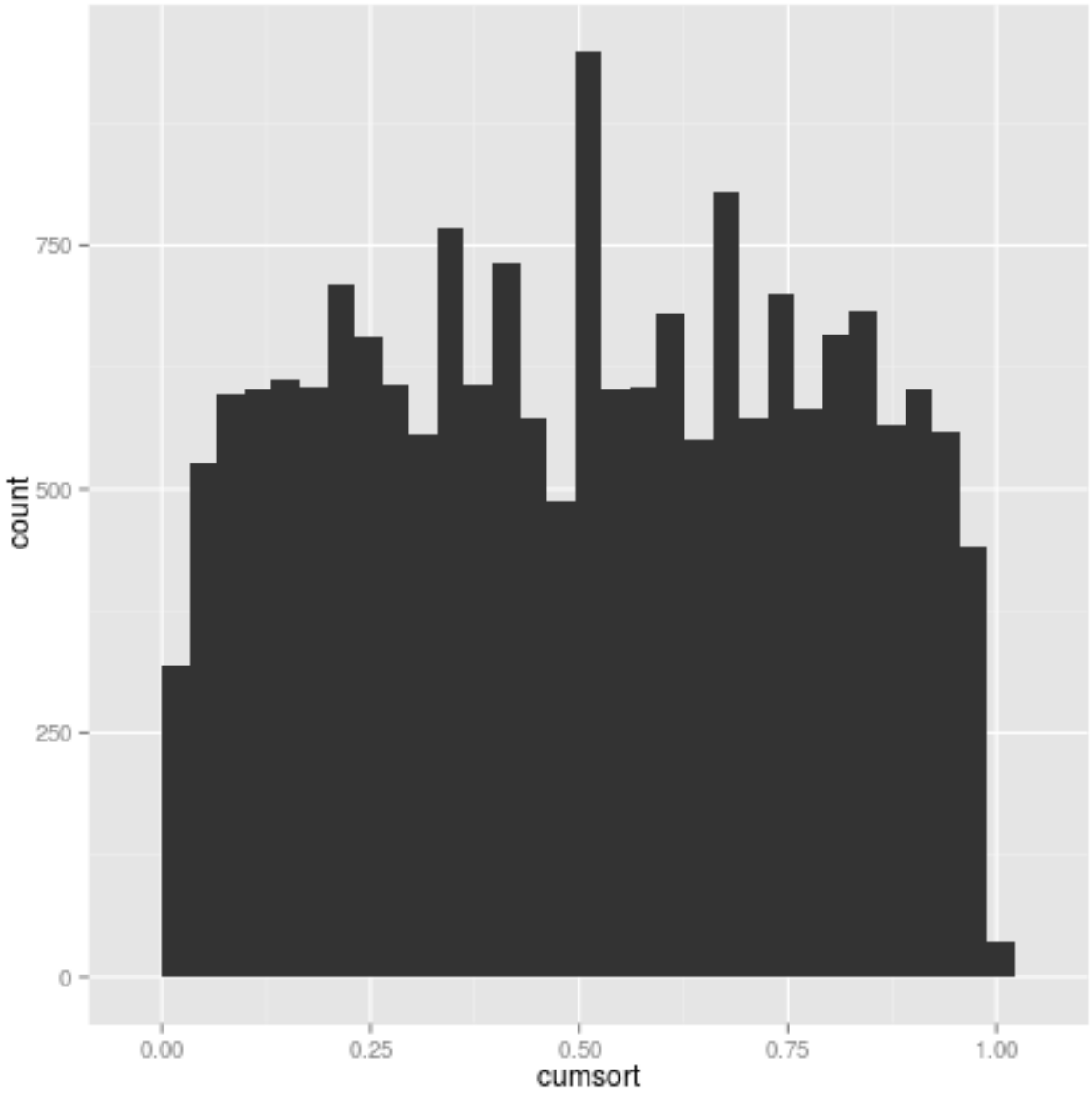

Figure 5: Histograma de la variable *cumsort*

Veamos que sucede si representamos el orden (*sort*) frente a *location*, con la componente de severidad del fallo.

```
st$fs_factor <- as.factor(st$fault_severity)
```

```
png(filename = "res/location_vs_sort.png")
ggplot(data = st, aes(x = location, y = sort)) +
  geom_point(aes(color = fs_factor, alpha = fs_factor, size = fs_factor)) +
  scale_color_manual(
    name = "fault_severity",
    values = c("-1"="black", "0"="green", "1"="blue", "2"="red")) +
  scale_alpha_manual(
    legend = NULL,
    values = c("-1"=0.8, "0"=0.6, "1"=0.6, "2"=0.6)) +
  scale_size_manual(
    name = "fault_severity",
    legend = NULL,
    values = c("-1"=0.5, "0"=2, "1"=2, "2"=2))
dev.off()
```

En la *Figura 6* podemos observar que los fallos de más severidad se producen en las *location* más altas, y que un *sort* pequeño podría acotar los fallos de baja gravedad.

Hacemos lo mismo con el orden inverso (*revsort*).

```
png(filename = "res/location_vs_revsort.png")
ggplot(data = st, aes(x = location, y = revsort)) +
  geom_point(aes(color = fs_factor, alpha = fs_factor, size = fs_factor)) +
  scale_color_manual(
    name = "fault_severity",
    values = c("-1"="black", "0"="green", "1"="blue", "2"="red")) +
  scale_alpha_manual(
    legend = NULL,
    values = c("-1"=0.8, "0"=0.6, "1"=0.6, "2"=0.6)) +
  scale_size_manual(
    name = "fault_severity",
    legend = NULL,
    values = c("-1"=0.5, "0"=2, "1"=2, "2"=2))
dev.off()
```

En la *Figura 7* vemos que ocurre lo mismo, pero con la salvedad de que separa mejor los fallos de gravedad 1 y 2.

Por último veamos que ocurre con el orden acumulado (*cumsort*).

```
png(filename = "res/location_vs_cumsort.png")
ggplot(data = st, aes(x = location, y = cumsort)) +
  geom_point(aes(color = fs_factor, alpha = fs_factor, size = fs_factor)) +
  scale_color_manual(
    name = "fault_severity",
    values = c("-1"="black", "0"="green", "1"="blue", "2"="red")) +
  scale_alpha_manual(
    legend = NULL,
    values = c("-1"=0.8, "0"=0.6, "1"=0.6, "2"=0.6)) +
  scale_size_manual(
    name = "fault_severity",
    legend = NULL,
    values = c("-1"=0.5, "0"=2, "1"=2, "2"=2))
dev.off()
```

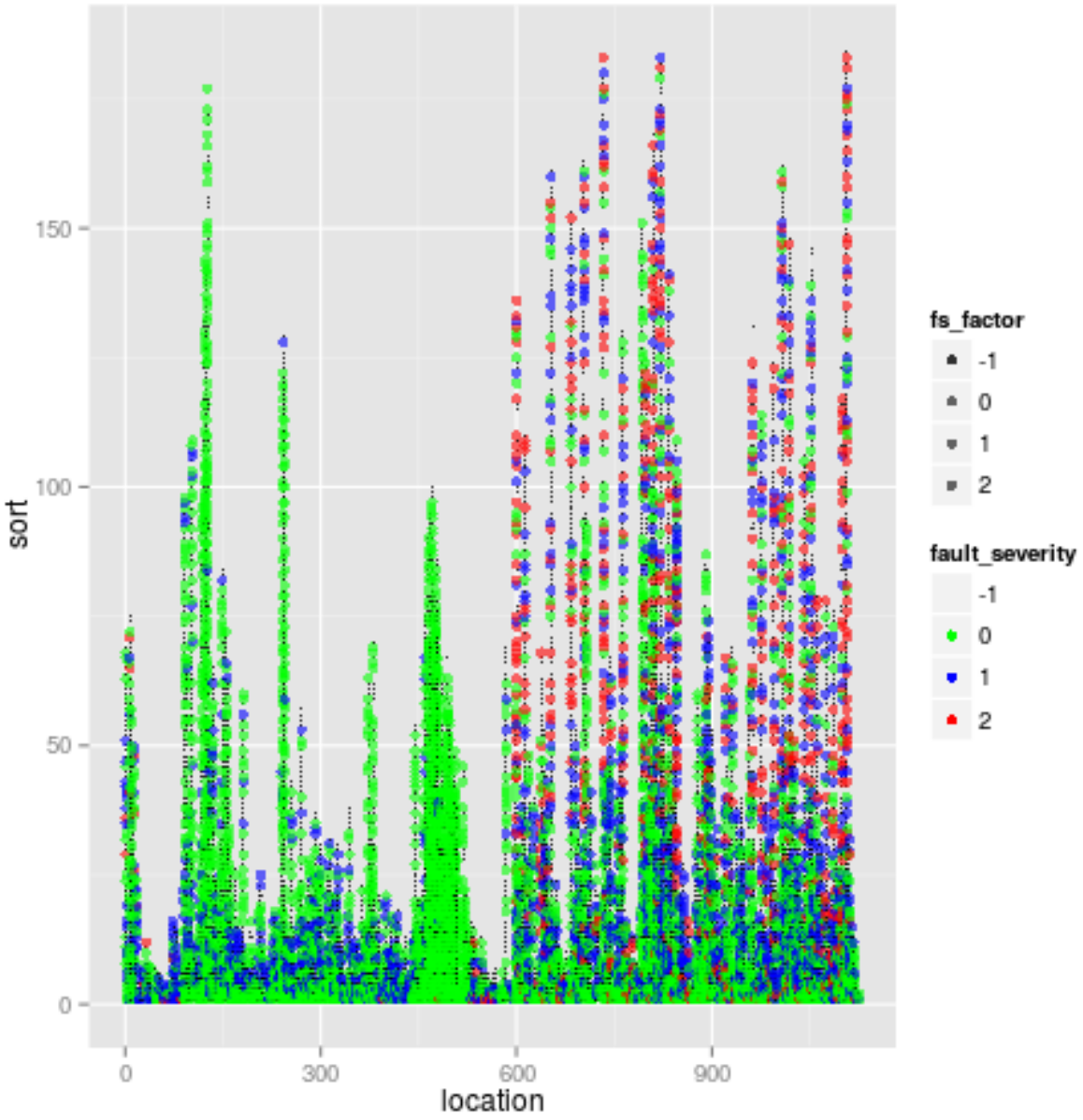

Figure 6: *location* vs *sort*

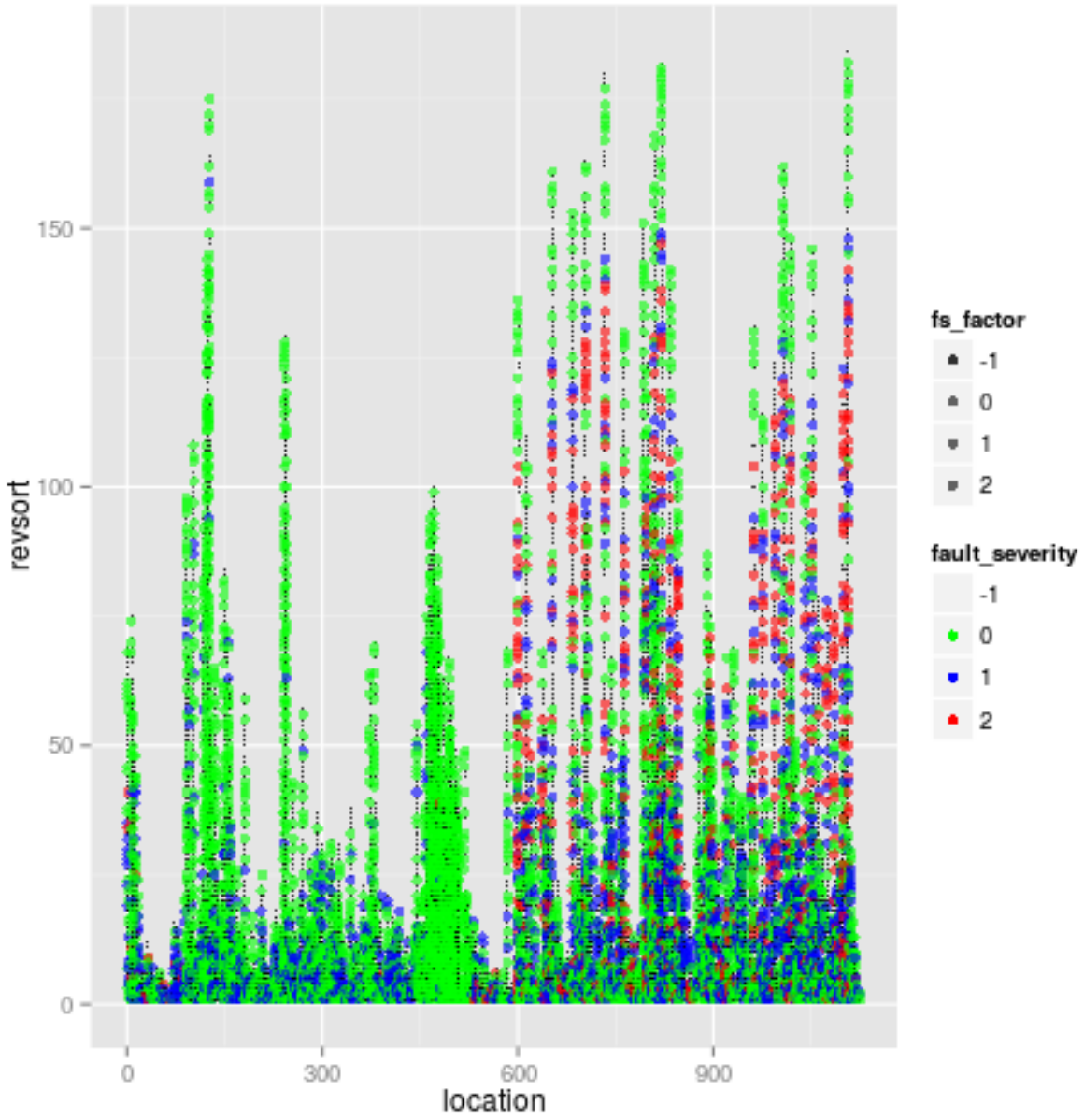

Figure 7: *location* vs *revsort*

Vemos en la *Figura 8* que esta nueva variable, al normalizar la aparición del fallo, es capaz de separarnos mucho mejor los datos por gravedad, especialmente los fallos de gravedad 0, de los de gravedad 1 y 2 (estos últimos concentrados en la el área rectangular que parte de la esquina superior derecha.

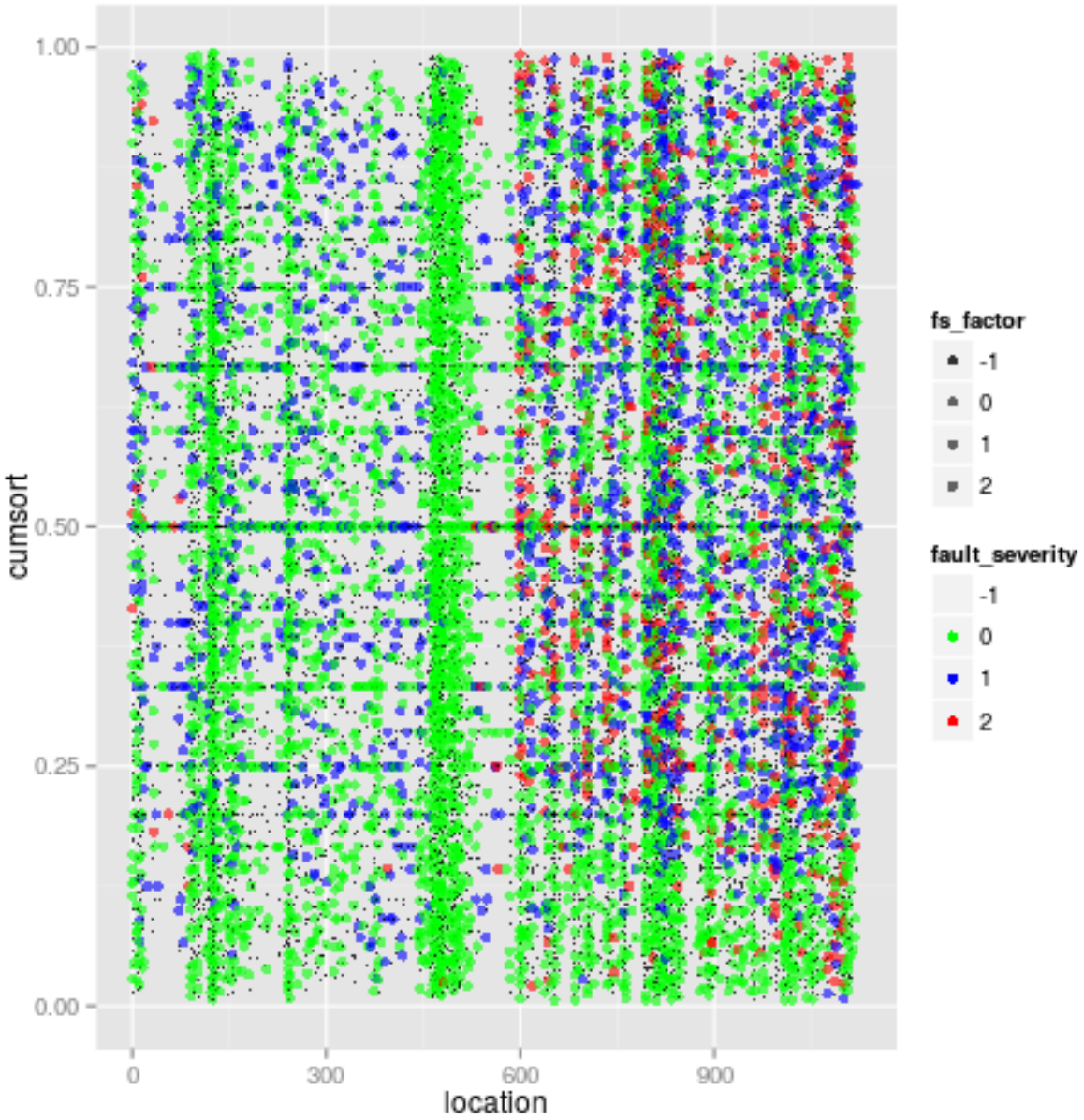

Figure 8: *location* vs *cumsort*

Podemos asegurar que estas nuevas variables, junto con *location* serán muy importante a la hora de construir nuestros modelos. Por lo tanto almacenamos los nuevos *datasets* en disco.

```
st[, fs_factor := NULL]
st[, location := NULL]
st[, fault_severity := NULL]

write.csv(locations,
```

```
          file = "data/features/locations_feats.csv",
          quote = F,
          row.names = F)
write.csv(st,
          file = "data/features/sev_type_sort_feats.csv",
          quote = F,
          row.names = F)
```

```
#           id location fault_severity
#     1: 14121      118              1
#     2:  9320       91              0
#     3: 14394      152              1
#     4:  8218      931              1
#     5: 14804      120              0
#    ---
# 18548: 14806     1073             -1
# 18549:  1825       11             -1
# 18550:  2374      917             -1
# 18551:  7277      208             -1
# 18552:  9886      438             -1
```

```
#          id severity_type sort revsort    cumsort
#     1: 6597             2    1      69 0.01428571
#     2: 8011             2    2      68 0.02857143
#     3: 2597             2    3      67 0.04285714
#     4: 5022             1    4      66 0.05714286
#     5: 6852             1    5      65 0.07142857
#    ---
# 18548: 3761             1   12       5 0.70588235
# 18549: 8720             1   13       4 0.76470588
# 18550: 6488             2   14       3 0.82352941
# 18551:  878             2   15       2 0.88235294
# 18552: 4464             1   16       1 0.94117647
```

5 Entrenando modelos

Finalmente entrenamos varios modelos, basados en distintas técnicas, para realizar nuestras predicciones y evaluar nuestros resultados mediante *submissions* a la platforma *Kaggle*.

5.1 Random Forests

En primer lugar elegimos entrenar varios *Random forests* por tratarse de un modelo fácil de utilizar y con resultados muy aceptables en problemas de clasificación.

5.1.1 Random Forest simple

En primer lugar elegimos un *Random Forest* simple, con los valores por defecto que encontramos en `randomForest`, con la única salvedad de que en este primer intento no incluiremos las variables generadas por el *PCA* y utilizaremos 100 árboles para generar nuestro modelo.

```
library(data.table)
library(randomForest)
library(corrplot)
library(gmodels)

## Train Random Forest
trainRFModel <- function(train.data, train.class, ntrees) {
  rf.model <- randomForest(train.data, y = train.class, method = "class",
                           ntree = ntrees, do.trace = T, na.action = na.omit)
  return(rf.model)
}

locations_feats <- fread(input = "data/features/locations_feats.csv") # This contains train and test
count_feats <- fread(input = "data/features/count_feats.csv")
min_feats <- fread(input = "data/features/min_feats.csv")
max_feats <- fread(input = "data/features/max_feats.csv")
sum_feats <- fread(input = "data/features/sum_feats.csv")
median_feats <- fread(input = "data/features/median_feats.csv")
mean_feats <- fread(input = "data/features/mean_feats.csv")
sev_type_sort_feats <- fread(input = "data/features/sev_type_sort_feats.csv")
sd_feats <- fread(input = "data/features/sd_feats.csv")
load(file = "data/features/log_features_pca.Rdata")
log_features_pca <- lf.dt.pca
rm(lf.dt.pca)

full <- merge(locations_feats, count_feats, all = T, by = "id")
full <- merge(full, min_feats, all = T, by = "id")
full <- merge(full, max_feats, all = T, by = "id")
full <- merge(full, sum_feats, all = T, by = "id")
full <- merge(full, median_feats, all = T, by = "id")
full <- merge(full, mean_feats, all = T, by = "id")
full <- merge(full, sev_type_sort_feats, all = T, by = "id")
full <- merge(full, sd_feats, all = T, by = "id")

class(full)

train <- full[full$fault_severity != -1]
test <- full[full$fault_severity == -1]

table(train$fault_severity)
train.class <- train[["fault_severity"]]
train.id <- train[["id"]]
test.id <- test[["id"]]
train[, fault_severity := NULL]
test[, fault_severity := NULL]

rf.model.000 <- trainRFModel(train, as.factor(train.class), 100)

png(filename = "res/RFImportance.000.png")
varImpPlot(rf.model.000)
dev.off()

save(rf.model.000, file = "model/rf.model.000")
```

```
pred.rf.prob <- predict(rf.model.000, test, type = "prob")
pred.rf.resp <- predict(rf.model.000, test, type = "response")

#CrossTable(test.data$y, pred.rf.resp, prop.chisq = F, prop.c = T, prop.r = F)

submission <- fread(input = "data/sample_submission.csv")
submission <- submission[order(id)]
submission <- as.data.frame(submission)
submission[, 2:4] <- pred.rf.prob

write.csv(submission, file = "res/rf.submission.000.csv", quote = F, row.names = F)
```

Estamos en la posición **452** con un *scoring* de **0.53791**.

En la *Figura 9*, observamos que las variables de orden y *location* son las más importantes y que, a continuación los estadísticos de `log_feature.csv` son los más descriptivos.

5.1.2 Random Forest optimizado

A continuación, haciendo uso de `tuneRF` del paquete `randomForest` pasamos a aumentar el rendimiento de nuestro bosque. Para ello generamos *random forests* hasta que el error en *training* sea menor del 5 por ciento, con un máximo de 100 árboles.

```
rf.model.001 <- tuneRF(
  train,
  as.factor(train.class),
  mtryStart = 1,
  ntreeTry = 100,
  improve = 0.05,
  trace = T,
  plot = T,
  doBest = T)

png(filename = "res/RFImportance.001.png")
varImpPlot(rf.model.001)
dev.off()

save(rf.model.001, file = "model/rf.model.001")

pred.rf.prob <- predict(rf.model.001, test, type = "prob")
pred.rf.resp <- predict(rf.model.001, test, type = "response")

#CrossTable(test.data$y, pred.rf.resp, prop.chisq = F, prop.c = T, prop.r = F)

submission <- fread(input = "data/sample_submission.csv")
submission <- submission[order(id)]
submission <- as.data.frame(submission)
submission[, 2:4] <- pred.rf.prob

write.csv(submission, file = "res/rf.submission.001.csv", quote = F, row.names = F)
```

Estamos en la posición **67** con un *scoring* de **0.44174**.

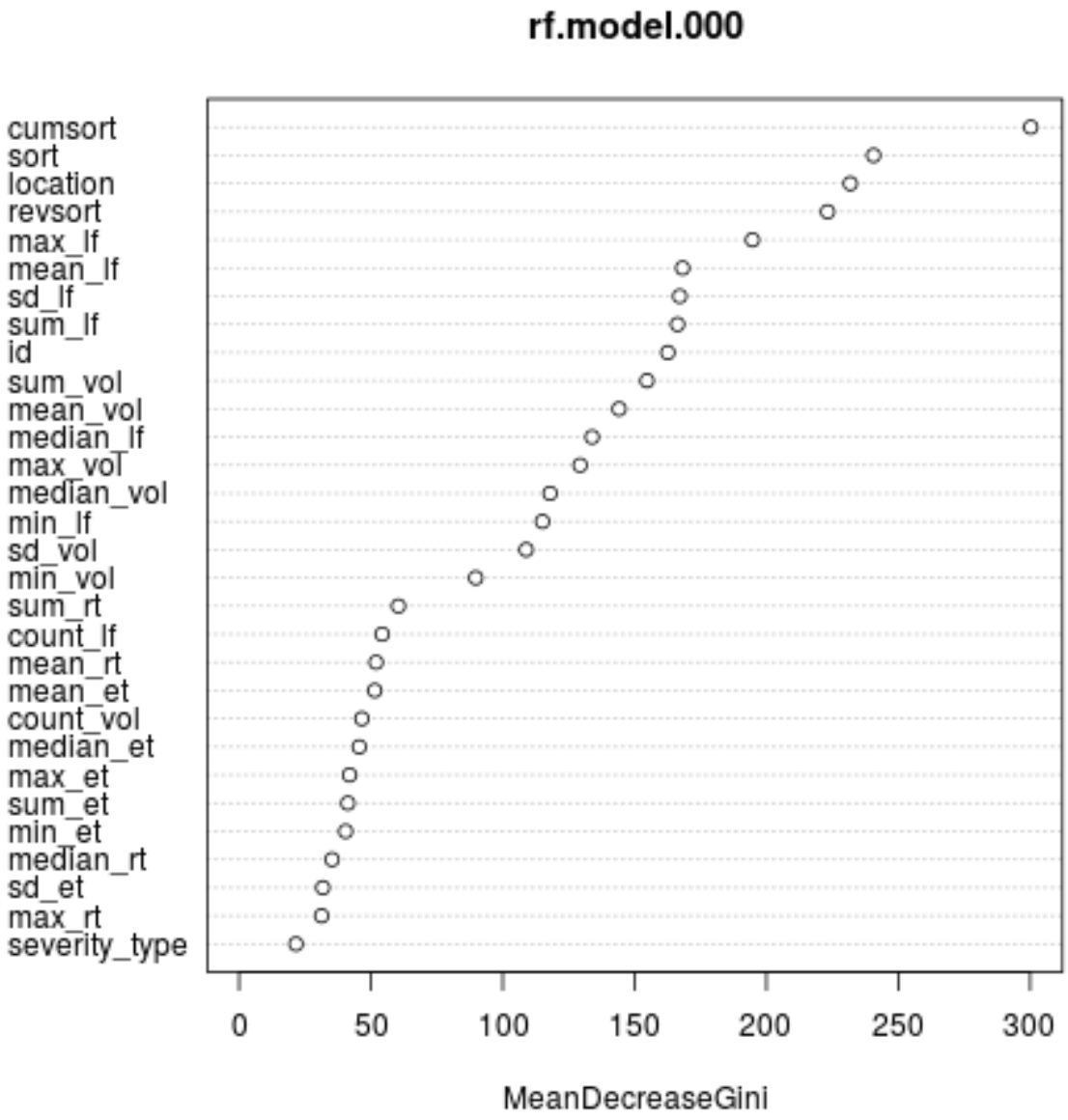

Figure 9: *varimplot* del modelo *rf.model.000*

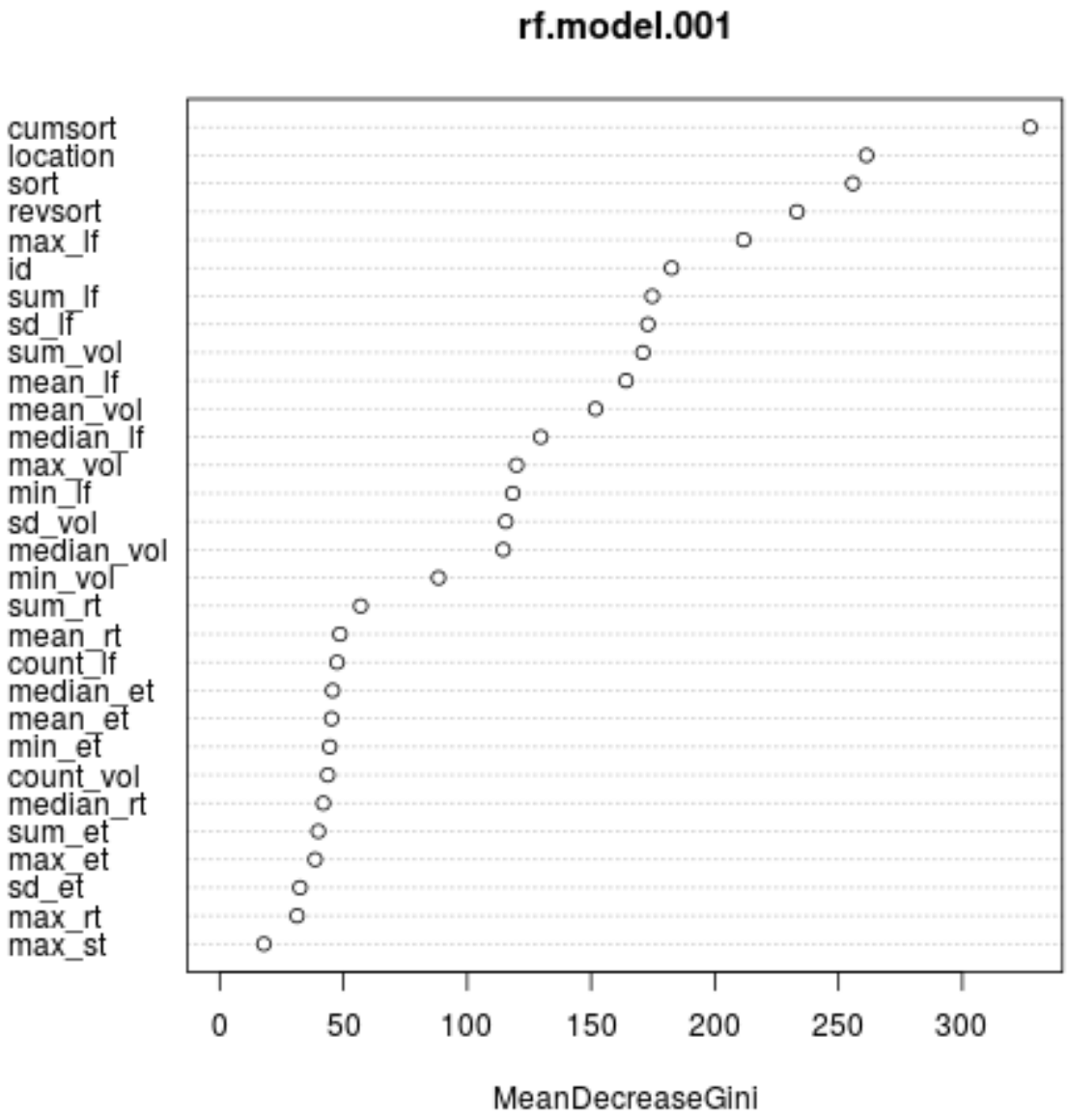

Figure 10: *varimplot* del modelo *rf.model.001*

En la *Figura 10* vemos que las variables con más importancia siguen siendo las de orden y *location*, pero en este caso parece que *id* ha desplazado a algunos estadísticos de `log_feature.csv`.

5.1.3 Random Forest optimizado y PCAs

El siguiente *random forest* tendrá las mismas características que el anterior, pero en este caso añadiremos las 5 primeras componentes principales al *dataset* que estamos utilizando para el entrenamiento y la predicción.

```
best_5_pcs <- log_features_pca[, 1:6, with = F]

full <- merge(locations_feats, count_feats, all = T, by = "id")
full <- merge(full, min_feats, all = T, by = "id")
full <- merge(full, max_feats, all = T, by = "id")
full <- merge(full, sum_feats, all = T, by = "id")
full <- merge(full, median_feats, all = T, by = "id")
full <- merge(full, mean_feats, all = T, by = "id")
full <- merge(full, sev_type_sort_feats, all = T, by = "id")
full <- merge(full, sd_feats, all = T, by = "id")
full <- merge(full, best_5_pcs, all = T, by = "id")

train <- full[full$fault_severity != -1]
test <- full[full$fault_severity == -1]

train.class <- train[["fault_severity"]]
train.id <- train[["id"]]
test.id <- test[["id"]]
train[, fault_severity := NULL]
test[, fault_severity := NULL]

rf.model.002 <- tuneRF(
  train,
  as.factor(train.class),
  mtryStart = 1,
  ntreeTry = 100,
  improve = 0.05,
  trace = T,
  plot = T,
  doBest = T)

png(filename = "res/RFImportance.002.png")
varImpPlot(rf.model.002)
dev.off()

save(rf.model.002, file = "model/rf.model.002")

pred.rf.prob <- predict(rf.model.002, test, type = "prob")
pred.rf.resp <- predict(rf.model.002, test, type = "response")

#CrossTable(test.data$y, pred.rf.resp, prop.chisq = F, prop.c = T, prop.r = F)

submission <- fread(input = "data/sample_submission.csv")
submission <- submission[order(id)]
submission <- as.data.frame(submission)
```

```
submission[, 2:4] <- pred.rf.prob

write.csv(submission, file = "res/rf.submission.002.csv", quote = F, row.names = F)
```

Estamos en la posición **67** con un *scoring* de **0.44258**. Aunque nos mantenemos en la misma posición, nuestra puntuación ha descendido, por lo que la introducción de las variables obtenidas por el método del **PCA** no parecen que le aporten mayor información a nuestro modelo. Realmente, el *scoring* es peor que el anterior.

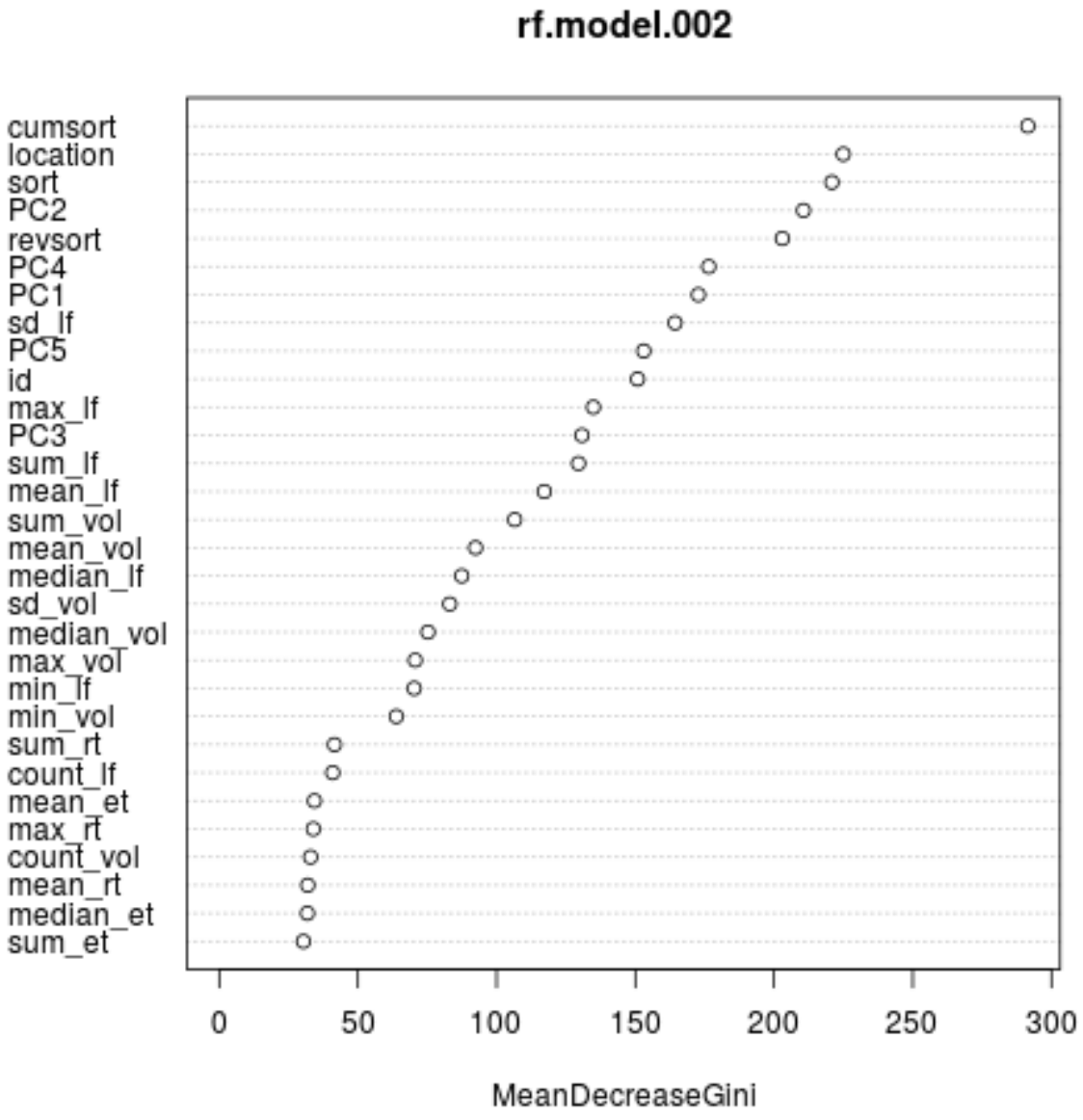

Figure 11: *varimplot* del modelo *rf.model.002*

Las variables que más importancia tienen en nuestro modelo son las usuales hasta ahora, pero en este caso parece que los estadísticos han sido desplazados por las componentes principales, como se puede observar en la *Figura 11*.

5.1.4 Forzando el Random Forest

Por último, forzamos la generación de un *random forest* con 1000 árboles y un error menor del 5 por 1000 (lo que generará) errores con respecto al *dataset* de *training*.

```
best_5_pcs <- log_features_pca[, 1:6, with = F]

full <- merge(locations_feats, count_feats, all = T, by = "id")
full <- merge(full, min_feats, all = T, by = "id")
full <- merge(full, max_feats, all = T, by = "id")
full <- merge(full, sum_feats, all = T, by = "id")
full <- merge(full, median_feats, all = T, by = "id")
full <- merge(full, mean_feats, all = T, by = "id")
full <- merge(full, sev_type_sort_feats, all = T, by = "id")
full <- merge(full, sd_feats, all = T, by = "id")
full <- merge(full, best_5_pcs, all = T, by = "id")

train <- full[full$fault_severity != -1]
test <- full[full$fault_severity == -1]

train.class <- train[["fault_severity"]]
train.id <- train[["id"]]
test.id <- test[["id"]]
train[, fault_severity := NULL]
test[, fault_severity := NULL]

rf.model.003 <- tuneRF(
  train,
  as.factor(train.class),
  mtryStart = 1,
  ntreeTry = 1000,
  improve = 0.005,
  trace = T,
  plot = T,
  doBest = T)

png(filename = "res/RFImportance.003.png")
varImpPlot(rf.model.003)
dev.off()

save(rf.model.003, file = "model/rf.model.003")

pred.rf.prob <- predict(rf.model.003, test, type = "prob")
pred.rf.resp <- predict(rf.model.003, test, type = "response")

#CrossTable(test.data$y, pred.rf.resp, prop.chisq = F, prop.c = T, prop.r = F)

submission <- fread(input = "data/sample_submission.csv")
submission <- submission[order(id)]
submission <- as.data.frame(submission)
submission[, 2:4] <- pred.rf.prob

write.csv(submission, file = "res/rf.submission.003.csv", quote = F, row.names = F)
```

Estamos en la posición **70** con un *scoring* de **0.44474**. Aunque cabría esperar una mejor puntuación, vemos que hemos descendido en el *leaderboard*. Esto puede deberse a que a partir de cierto punto, nuestro modelo deja de ser útil para predecir variables que no se encuentren en nuestro *dataset* de *training*. Podríamos estar entrando en el terreno del *overfitting*.

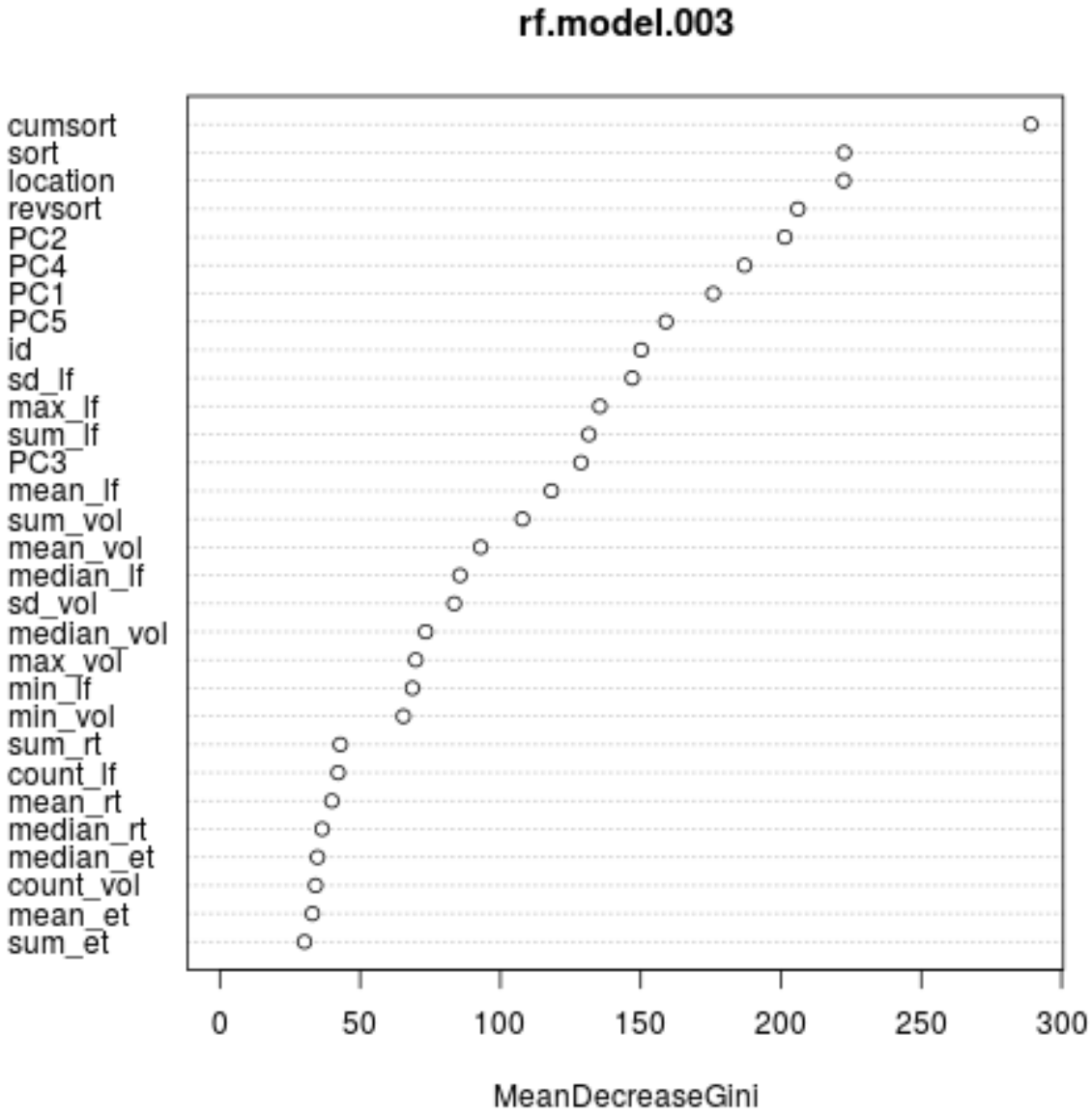

Figure 12: *varimplot* del modelo *rf.model.003*

En este caso, las variables que más importancia tienen en nuestro modelo son las que se muestran en la *Figura 12*. Como se puede observar, el orden, *location* y las componentes principales han tomado la mayor importancia.

5.2 Support Vector Machines

Cambiando el enfoque, vamos a probar con la técnica de *Support Vector Machines*. Aunque más costosa en recursos, bien ajustada, puede proporcionar muy buenos resultados en problemas de clasificación.

Entrenamos este modelo utilizando k-5 *cross validation* (en los *random forest* se aplica por defecto) con un *kernel* Gaussiano.

```
require(kernlab)

best_5_pcs <- log_features_pca[, 1:6, with = F]

full <- merge(locations_feats, count_feats, all = T, by = "id")
full <- merge(full, min_feats, all = T, by = "id")
full <- merge(full, max_feats, all = T, by = "id")
full <- merge(full, sum_feats, all = T, by = "id")
full <- merge(full, median_feats, all = T, by = "id")
full <- merge(full, mean_feats, all = T, by = "id")
full <- merge(full, sev_type_sort_feats, all = T, by = "id")
full <- merge(full, sd_feats, all = T, by = "id")
full <- merge(full, best_5_pcs, all = T, by = "id")

train <- full[full$fault_severity != -1]
test <- full[full$fault_severity == -1]

train[, id := NULL]
test[, id := NULL]

svm.model.000 <- ksvm(
  as.factor(fault_severity) ~ .,
  data = train,
  cross = 5,
  kernel = "rbfdot",
  prob.model = T)

save(svm.model.000, file = "model/svm.model.000")

pred.svm.prob <- predict(svm.model.000, test, type = "prob")
pred.svm.resp <- predict(svm.model.000, test, type = "response")

submission <- fread(input = "data/sample_submission.csv")
submission <- submission[order(id)]
submission <- as.data.frame(submission)
submission[, 2:4] <- pred.svm.prob

write.csv(submission, file = "res/svm.submission.000.csv", quote = F, row.names = F)
```

Estamos en la posición **684** con un *scoring* de **0.68356**. Podemos aceptar que en este caso **SVM** no es una técnica que se adapte bien a nuestro caso, ya que, aunque no aparezca en este documento, se probó con distintos *kernels*, tales como *vanilladot* (lineal) y *polydot* (polinomial) y ambos se quedaron atascados buscando nuevas variables, después de una ejecución de varias horas.

5.3 Gradient Boosting

Las técnicas de *gradient boosting*, tales como *xgboost* o *gamboost* suelen proporcionar buenos resultados en modelos predictivos para funciones de pérdida (*loss*), tales como la función de evaluación aplicada en esta competición para medir la bondad del modelo. Por esa razón hemos optado por crear un modelo en base a la técnica *xgboost* con *cross-validation* mediante el paquete `xgboost`. Los parámetros que hemos decidido para nuestro modelo son:

- Evaluación *mlogloss* por ser la utilizada para evaluar la predicción en *Kaggle*.
- Una tasa de aprendizaje de 0.05, ya que los valores pequeños previenen el *overfitting*.
- Una proporción de muestreo del 70 por ciento.
- Una proporción de columnas al construir los árboles del 85 por ciento.
- Como objetivo utilizamos `multi:softprob` que devuelve una clasificación multiclase con resultado de probabilidades.
- Por último realizaremos 1000 pasadas con k-10 *cross validation*.

```
require(xgboost)

best_5_pcs <- log_features_pca[, 1:6, with = F]

full <- merge(locations_feats, count_feats, all = T, by = "id")
full <- merge(full, min_feats, all = T, by = "id")
full <- merge(full, max_feats, all = T, by = "id")
full <- merge(full, sum_feats, all = T, by = "id")
full <- merge(full, median_feats, all = T, by = "id")
full <- merge(full, mean_feats, all = T, by = "id")
full <- merge(full, sev_type_sort_feats, all = T, by = "id")
full <- merge(full, sd_feats, all = T, by = "id")
full <- merge(full, best_15_pcs, all = T, by = "id")

train <- full[full$fault_severity != -1]
test <- full[full$fault_severity == -1]

train.class <- train[["fault_severity"]]
train.id <- train[["id"]]
test.id <- test[["id"]]
train[, fault_severity := NULL]
test[, fault_severity := NULL]

params <- list(
  eval_metric = 'mlogloss',
  num_class = max(train.class) + 1,
  eta = 0.05,
  subsample = 0.7,
  colsample_bytree = 0.85,
  objective = 'multi:softprob',
  gamma = 0.1,
  lambda=0.9)

best.xgbcv <- xgb.cv(
  params = params,
  data = data.matrix(train),
  label = train.class,
  nfold = 10,
```

```
  nrounds = 1000,
  print.every.n = 10,
  early.stop.round=100)

NROUNDS <- which.min(best.xgbcv$train.mlogloss.mean)

xgb.model.000 <- xgboost(
  params = params,
  data = data.matrix(train),
  label = train.class,
  nrounds=NROUNDS,
  print.every.n = 100)

xgb.save(xgb.model.000, fname = "model/xgb.model.000")
# xgb.model.000 <- xgb.load(modelfile = "model/xgb.model.000")

pred.xgb.prob <- predict(xgb.model.000, data.matrix(test))

pred.xgb.prob <- matrix(pred.xgb.prob, ncol = 3, byrow = T)

submission <- fread(input = "data/sample_submission.csv")
submission <- submission[order(id)]
submission <- as.data.frame(submission)
submission[, 2:4] <- pred.xgb.prob

write.csv(submission, file = "res/xgb.submission.000.csv", quote = F, row.names = F)
```

Estamos en la posición **70** con un *scoring* de **0.44586**.

5.4 Ensembles

Finalmente, vamos a combinar los dos modelos que mejores resultados han obtenido: `xgb.model.000` con un *scoring* de **0.44586** y `rf.model.001` con un *scoring* de **0.44174**. Para ello balanceramos las predicciones aunque sin llevar a ninguna de las dos a los extremos, ya que las puntuaciones obtenidas con ambos modelos son muy parecidas.

```
xgb.submission.000 <- fread("res/xgb.submission.000.csv", data.table = F)
rf.submission.001 <- fread("res/rf.submission.001.csv", data.table = F)

submission <- xgb.submission.000

submission[, 2:4] <- 0.5 * xgb.submission.000[, 2:4] + 0.5 * rf.submission.001[, 2:4]
write.csv(submission, file = "res/ens.submission.000.csv", quote = F, row.names = F)
```

Estamos en la posición **67** con un *scoring* de **0.44238**.

```
submission[, 2:4] <- 0.4 * xgb.submission.000[, 2:4] + 0.6 * rf.submission.001[, 2:4]
write.csv(submission, file = "res/ens.submission.001.csv", quote = F, row.names = F)
```

Estamos en la posición **70** con un *scoring* de **0.44474**.

```
submission[, 2:4] <- 0.3 * xgb.submission.000[, 2:4] + 0.7 * rf.submission.001[, 2:4]
write.csv(submission, file = "res/ens.submission.002.csv", quote = F, row.names = F)
```

Estamos en la posición **71** con un *scoring* de **0.44793**.

```
submission[, 2:4] <- 0.6 * xgb.submission.000[, 2:4] + 0.4 * rf.submission.003[, 2:4]
write.csv(submission, file = "res/ens.submission.003.csv", quote = F, row.names = F)
```

Estamos en la posición **67** con un *scoring* de **0.44071**.

```
submission[, 2:4] <- 0.7 * xgb.submission.000[, 2:4] + 0.3 * rf.submission.003[, 2:4]
write.csv(submission, file = "res/ens.submission.004.csv", quote = F, row.names = F)
```

Estamos en la posición **67** con un *scoring* de **0.44016**.

```
submission[, 2:4] <- 0.8 * xgb.submission.000[, 2:4] + 0.2 * rf.submission.003[, 2:4]
write.csv(submission, file = "res/ens.submission.005.csv", quote = F, row.names = F)
```

Estamos en la posición **67** con un *scoring* de **0.44061**.

Como cabía esperar, la combinación de ambos modelos, aunque modestamente, nos proporciona mejores resultados que los dos por separado. Aunque, inesperadamente, parece que al ofrecer mayor peso en el *ensemble* al modelo que obtuvo peor *scoring* obtenemos mejores resultados.

6 Conclusiones

Como conclusión final al proyecto realizado, cabe destacar que el puesto **67** de **974** participantes con un *scoring* de **0.44061** es buen lugar para una primera aproximación al problema. Más aún, teniendo en cuenta que la competición había finalizado cuando se realizarón las *submissions*, además de que muchos participantes han publicado sus soluciones, en las que, en algunos casos, han sido de mucha utilidad para la realización de este ejercicio.

El mayor grado de complejidad en el desarrollo del proyecto está en la elaboración de *datasets* adecuados para su tratamiento, generando variables útiles que permitan aumentar la tasa de clasificación, ya que los *datasets* originales son poco descriptivos y no demasiado útiles.

Finalmente, como mejoras posibles al proyecto, me gustaría añadir que se podría continuar por dos vías distintas para mejorar el *scoring* obtenido:

1. Creación de nuevas variables más descriptivas, como por ejemplo buscar patrones en la aparición de *features* por **id** y otorgar un valor a las combinaciones de *features* que más se repitan para los distintos grados de fallo.
2. Entrenamiento de un mayor número de instancias del mismo modelo y obtención de los resultados de predicción basados en la media de los resultados obtenidos por cada una de las instancias: por ejemplo, entrenar 1000 *random forests* con las mismas características y obtener como resultado de la predicción la media de los obtenidos por las 1000 instancias, cruzando estos resultados con el resultado obtenido aplicando el mismo método, por ejemplo, usando *xgboost* y realizar la *submission* del resultado final.