Clasificación de texto legal

Victor Gallego y Roi Naveiro

09/04/2019

```
library(text2vec)
library(tidyverse)
library(plyr)
library(caret)
```

Introducción

En este ejercicio, entrenaremos varios clasificadores sobre texto legal. Cada documento se corresponde con un párrafo que puede pertenecer a una de cinco porsibles clases: ...

El objetivo es, dado un nuevo texto nunca visto, predecir la clase a la que pertenece.

Primero descargamos y leemos los datos

```
# Descargamos y leemos los datos
download.file('https://github.com/vicgalle/neural-classifier/blob/master/data/Quantum1.xlsx?raw=true',
data <- readxl::read_xlsx('data.xlsx')
```

Explora los datos

```
# Echamos un vistazo
count(data, 'Grupo')
```

```
##   Grupo freq
## 1     A  500
## 2     B  500
## 3     C  500
## 4     D  490
## 5     E  500
```

```
data$Fallos[1]
```

```
## [1] "Inadmitir el recurso de apelación interpuesto por VERDELENA S.A. contra el Auto 50/2016 de 2 de
```

Divide el conjunto de datos en train y test con proporciones 0.8 y 0.2 respectivamente

```
# Creamos split train - test
ind_train <- sample(1:nrow(data), 0.8*nrow(data))
data_train <- data[ind_train,]
data_test <- data[-ind_train,]
```

Preprocesado de textos

Aplica a cada texto del train el siguiente preprocesado:

1. Reducir a minúsculas.
2. Separar por palabras
3. Recortar el vocabulario para solo seleccionar las palabras que al menos aparecen diez veces en algún documento y que además aparecen en al menos en el 0.1 % de los documentos.
4. Vectorizar estas palabras, usando representación bag of words.

```
# Definimos el preprocesado y tokenizado
it_train = itoken(data_train$Fallos,
                  preprocessor = tolower,
                  tokenizer = word_tokenizer,
                  ids = data_train$idSentidosFallos,
                  progressbar = TRUE)
vocab = create_vocabulary(it_train)
```

```
##
  |
  |=======                                                  |  10%
  |
  |=============                                            |  20%
  |
  |====================                                     |  30%
  |
  |==========================                               |  40%
  |
  |================================                         |  50%
  |
  |=======================================                  |  60%
  |
  |==============================================           |  70%
  |
  |======================================================   |  80%
  |
  |=============================================================    |  90%
  |
  |=====================================================================| 100%
```

```
# nos quedams con palabras que al menos aparezcan 10 veces.
# Cada palabra deberá estar al menos en el 0.1% de documentos
pruned_vocab = prune_vocabulary(vocab,
                                term_count_min = 10,
                                doc_proportion_min = 0.001)
vectorizer = vocab_vectorizer(pruned_vocab)

#dtm: document term matrix
dtm_train = create_dtm(it_train, vectorizer)
```

```
##
  |
  |=======                                                  |  10%
  |
  |=============                                            |  20%
  |
  |====================                                     |  30%
  |
  |==========================                               |  40%
  |
  |================================                         |  50%
  |
  |=======================================                  |  60%
  |
  |==============================================           |  70%
```

```
  |
  |=================================================                 |  80%
  |
  |=======================================================           |  90%
  |
  |==================================================================| 100%
```

¿Cuál es la dimensión del train tras el preprocesado? ¿Cuántas palabras tiene el vocabulario?

```
dim(dtm_train)
```

```
## [1] 1992 2115
```

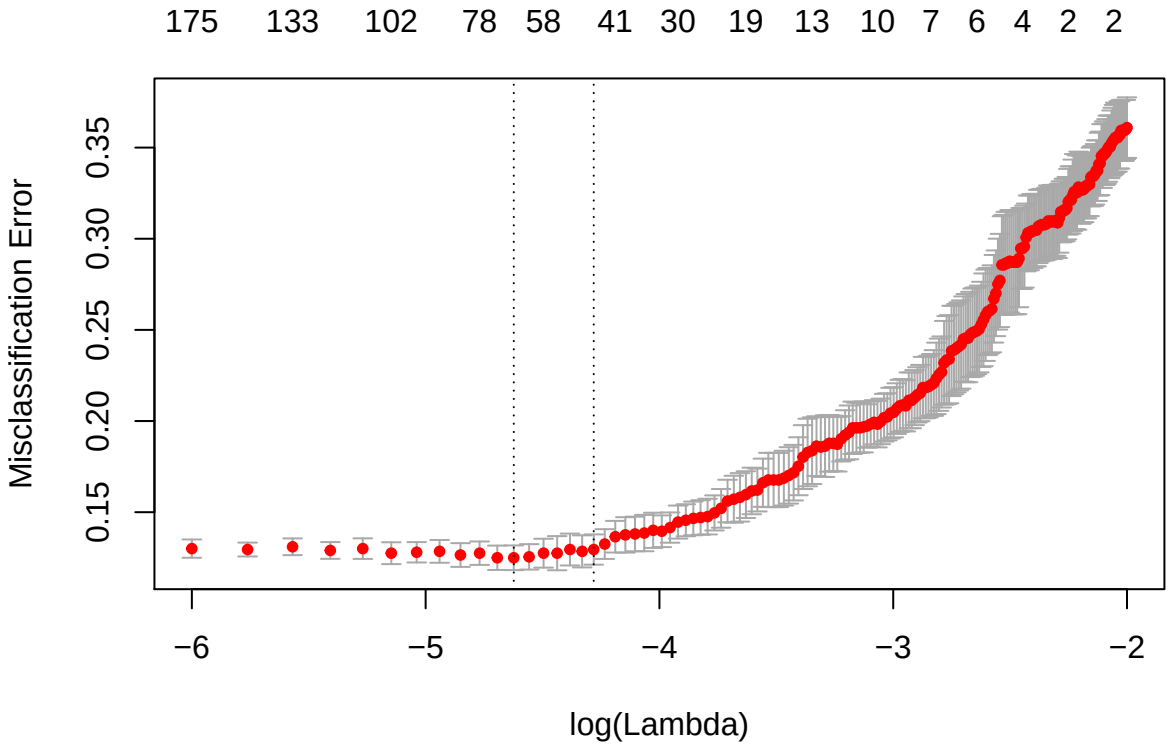
Crea el conjunto de test (OJO, usa el vectorizer generado por el train, sino es trampa... ¿por qué?).

```
it_test = data_test$Fallos %>%
  tolower %>%
  word_tokenizer %>%
  itoken(ids = data_test$idSentidosFallos,
         progressbar = FALSE)
dtm_test = create_dtm(it_test, vectorizer)
```

Regresión Logística

Entrena un modelo de regresión logística con regularización L1 usando la librería glmnet. Haz validación cruzada con 4 folds, del coeficiente de regularización λ. Pinta la curva de error de clasificación frente a λ. ¿Cuál es el valor óptimo de este hiperparámetro?

```
library(glmnet)
NFOLDS = 4
glmnet_classifier = cv.glmnet(x = dtm_train, y = data_train$Grupo,
                              family = 'multinomial',
                              # L1 penalty
                              alpha = 1,
                              lambda = seq(exp(-6), exp(-2), length.out = 200),
                              type.measure = "class",
                              # 4-fold cross-validation
                              nfolds = NFOLDS,
                              # high value is less accurate, but has faster training
                              thresh = 1e-3,
                              # again lower number of iterations for faster training
                              maxit = 1e3)
plot(glmnet_classifier)
```



Predice sobre el test. ¿Qué precisión obtienes?

```
preds = predict(glmnet_classifier, dtm_test, type = 'class')
mean(preds == data_test$Grupo)
```

```
## [1] 0.8654618
```

Naive Bayes

Entrena un modelo NB sobre el texto legal. Para ello primero tendrás que crear dataframes de train y test. No olvides convertir cada variable predictora a un factor!! ¿Qué pasaría si no lo haces?

```
train_df = data.frame(as.matrix(dtm_train))
train_df = lapply(train_df, as.factor)
#train_df$label = data_train$Grupo

test_df = data.frame(as.matrix(dtm_test))
test_df = lapply(test_df, as.factor)
#train_df$label = data_train$Grupo
```

Entrena el modelo sin hacer validación.

```
library(e1071)
nb = naiveBayes(x = train_df, y = factor(data_train$Grupo) )
```

Predice sobre el test. ¿Qué precisión obtienes?

```
preds = predict(nb, test_df, type = "class")
mean(preds == data_test$Grupo)
```

```
## [1] 0.6546185
```