```
#########################################################
#CLASIFICACION_DETERMINACION_ AUTORÍA_IMPOSTERS

#https://computationalstylistics.github.io/resources/

#########################################################


#.  IMPOSTERS es una implementación de 'stylo' con la función imposters(), creada
#para Elegir entre varios documentos indubitados  aquel que presenta mayores
#similitudes con el documento dubitado que nos ocupa.


#########################################################
#La función de forma iterativa contrasta el texto dubitado con algunos textos
#representativos de los posibles candidatos a la autoría.
#Para ello debemos
# a) Preparar el corpus en una capeta que reúna el texto dubitado y varios otros textos
#indubitados de los candidatos y algún otro autor i"impostro" que en modo alguno
#pudiera haber escrito el texto.
 # Aplicamos la función imposters() que asignará a los textos indubitados una
#puntuación entre 0 y 1, donde el 0 significa exclusión total del candidato y 1
#asignación absolutamente segura del candidato.


#########################################################
# 1. En términos teóricos, cualquier puntaje superior a 0.5 sugeriría que la verificación
#de autoría para un candidato dado fue exitosa. Sin embargo, los puntajes
#entre  0.39 y 0.63 deben considerarse sospechosos: parecen sugerir que el
#clasificador tuvo problemas para tomar decisiones claras.

# 2. La función complementaria de  imposters.optimize() está destinada a evaluar,
#mediante una búsqueda en cuadrícula las puntuaciones que definen el área gris de
#indeterminadión confiable.


#########################################################
#PASOS PARA LA EJECUCIÓN:


# 0. Activamos stylo
library(stylo)

#  1. Preparar los textos en un corpus con (1) el texto dubitado + 2. Varios textos de
#impostores.

# 2. Establecemos el directorio que contiene los textos de nuestro corpus de análisis
```

```
setwd("~/Dropbox/4. Ling_forense/cuentapalabras/analisis_textos/datos/textos_5")

# 3. Reclamamos los textos del corpus y este lo descomponemos en las palabras que lo
#componen
tokenized.texts = load.corpus.and.parse(files = "all")

# 4. Contabiizamos X número de palabras más frecuentes como base del análisis que
#queremos realizar. Así, en nuestro ejemplo, le diremos a la máquina que trabaje con
#las 2000 palabras más frecuentes. Este número variará en función de la extensión
#mayor o menor d elos textos analizados
features = make.frequency.list(tokenized.texts, head = 2000)

# 5. Creamos las tablas de frecuencia relativa:
data = make.table.of.frequencies(tokenized.texts, features, relative = TRUE)

# 6. Comprobamos qué fila de la tabla de entrada de datos contiene el texto en
#disputa.
rownames(data)

# 7. Le decimos al algoritmo qué fila contine las frecuencias de nuestro texto dubitado
my_text_to_be_tested = data[1,]

# 8. Excluimos esa fila de la tabal de frecuencias
my_frequency_table = data[-c(1),]

# 9. Hacemos corer la función imposters():
imposters(reference.set = my_frequency_table, test = my_text_to_be_tested)

# 10. Optimizamos el resultado, para establecer la cifra por debajo de la cual un texto
#dubitado deberá ser excluido de la lista de candidatos (por ejemplo 0.43) y la cifra por
#encima de la cual la atribución es absolutamente fiable (por ejemplo 0.55)
imposters.optimize(data)

# 11. Para mayor fiabilidad probaremos otros algoritmos(el mejor  "wurzburg")
# Classic Delta distance
imposters(data, distance = "delta")

# Eder's Delta distance
imposters(data, distance = "eder")

# Cosine Delta (aka Wurzburg Distance)
imposters(data, distance = "wurzburg", imposters = 0.8)

# Ruzicka Distance (aka Minmax Distance)
imposters(data, distance = "minmax")
```