Grado en Estadística

TRABAJO FIN DE ESTUDIOS

Desarrollo de una herramienta para la detección de plagio en código R

Manuel Isaac Gandulo Lara

Sevilla, Junio de 2021

Índice general

Prólogo

La motivación de este Trabajo Fin de Grado vino dada por la necesidad del personal docente de la Universidad de Sevilla para detectar plagio en entregas realizadas en código fuente. Si bien la herramienta Blackboard dispone de una aplicación para la detección de plagio, dicha aplicación es poco configurable y no puede ser usada fuera de línea. Por lo tanto, el desarrollo de un software propio atendiendo a las necesidades particulares de los docentes sería de gran ayuda.

En particular, este trabajo se centra en las necesidades de los docentes del Departamento de Ciencias de la Computación e Inteligencia Artificial de dicha Universidad. De esta manera, en el análisis del estado del arte se estudian las herramientas existentes para la detección de plagio en los principales lenguajes de programación utilizados por las asignaturas del Departamento: Python, Haskell, R y RMarkdown. Tras dicho análisis, se ven posibilidades de mejora en las herramientas existentes para la detección de plagio en R y RMarkdown, lo cual tendría aplicación directa en asignaturas tales como Informática (Grado en Bioquímica), Informática (Grado en Estadística), Biotecnología (Grado en Ingeniería de la Salud), Biología Molecular de Sistemas (Grado en Bioquímica), Técnicas Ómicas y Bioinformática (Grado en Biomedicina Básica y Experimental), Diseño de Experimentos y Análisis de Datos (M.U. en Biología Avanzada) y Técnicas Inteligentes en Bioinformática (M.U. en Lógica, Computación e Inteligencia Artificial), entre otras.

El lenguaje de programación seleccionado para el desarrollo de la herramienta presentada es, a su vez, el lenguaje R. Las causas principales de esta decisión son: un mejor dominio de este lenguaje por parte del autor de este trabajo con respecto a otros lenguajes y, fundamentalmente, la posibilidad de utilizar y extender la biblioteca SimilaR, cuyas ventajas y desventajas se detallan minuciosamente en el estado del arte, encontrando diversos puntos de mejora que serán conducidos en los siguientes capítulos. Es por ello que el nombre seleccionado para la nueva herramienta haya sido SimilaR2.

Se ha puesto un gran empeño en documentar tanto la nueva herramienta como su proceso de elaboración, para que así el lector pueda entender cada una de sus funcionalidades y las decisiones tomadas. Para facilitar el uso de la herramienta, se ha creado un paquete R (repositorio) y una aplicación web mediante Shiny (app) que muestra los resultados obtenidos por las funciones del paquete R de forma interactiva. Cabe decir finalmente que la instalación del paquete se puede realizar de forma sencilla tras instalar la biblioteca de R **devtools** y utilizar el comando `devtools::install_gitlab('igandullo88/similar2_package')`.

Agradecimientos.

Este trabajo está dedicado a mi familia, en especial a mi esposa, ya que me ha apoyado durante todo el trayecto, ayudándome a librar las batallas internas surgidas en el desarrollo de este trabajo.

Quiero dar las gracias a D. Luis Valencia Cabrera por su colaboración al proporcionar el dataset, su aporte ha sido fundamental para la elaboración del análisis estadístico.

Por otro lado, he de agradecer la cooperación de mis compañeros del Grado en Estadística (Ángel Caro, Manuel Buendía, Fco D. Orduña, entre otros) por contribuir en el proceso de plagiado de documentos.

Por último y no menos importante, debo un reconocimiento especial a mi tutor D. Ignacio Pérez Hurtado De Mendoza por secundar toda la elaboración de este Trabajo Fin de Grado. Gracias a su paciencia, comprensión y entrega, ha sido posible finalizar este proyecto.

Información sobre el autor:

Manuel Isaac Gandullo Lara

- Estudiante de cuarto curso del Grado en Estadística de la Universidad de Sevilla.
- Email: igandullo91088@gmail.com

Para **citar este trabajo** en publicaciones, utilizar el siguiente formato:

- Gandullo-Lara, I.G. 2021. Desarrollo de una herramienta para la detección de plagio en código R [10].

Para insertar esta referencia en un fichero bibliográfico BibTeX, añadir el siguiente código:

```
@Manual{IsaacGandullo2021,
title = {Desarrollo de una herramienta para la detección de plagio en código R},
author = {Isaac G. Gandullo-Lara},
year = {2021}
}
```

Resumen

Este trabajo se basa en el desarrollo de una herramienta de detección de plagio en código R y RMarkdownn. Con el fin de ofrecer al personal docente un software de código abierto y On Premise capaz de detectar plagio en entregas con esta extensión.

Con la finalidad de ofrecer un software de código abierto capaz de detectar plagio entre documentos con esta codificación, brindando la posibilidad al personal docente de detectar plagio en entregas con extensión .R y .Rmd.

El contenido de este trabajo se ha dividido en 7 capítulos de la siguiente forma:

En el **capítulo 1** se realiza una introducción sobre el plagio, el plagio en código fuente y como se combate el plagio dentro de las universidades.

A lo largo del **capítulo 2** se describen los aspectos generales del estado del arte, se analizan diferentes programas de detección de plagio en código fuente, destacando sus principales cualidades y analizando sus carencias. Finalmente, se realiza un análisis de dicho estado del arte, sirviendo como argumentación fundamental para el desarrollo de la herramienta de software que se presenta en este TFG.

El **capítulo 3** se centra en el paquete SimilaR, ya presentado en el capítulo anterior. Este paquete es quizás el más utilizado actualmente para la detección de plagio en código R. En este capítulo se estudian minuciosamente sus prestaciones y limitaciones, con la finalidad de proponer mejoras.

El **capítulo 4** está dedicado a explicar los detalles de la nueva implementación. Se consigue que la nueva herramienta, que bautizamos como SimilaR2, sea capaz de detectar plagio en secciones de comentarios gracias a la implementación de algoritmos fingerprints, así como ampliar las zonas de código R en dónde se puede detectar plagio (SimilaR tan solo puede detectar plagio en el cuerpo de funciones, pero no en el código escrito fuera de ellas). Adicionalmente, se explica la salida de SimilaR2, la cual aporta información detallada del análisis de detección de plagio, análisis de clústering y nubes de palabras, entre otras posibilidades.

Durante el **capítulo 5**, se detalla el proceso de construcción del paquete R SimilaR2, así como de una aplicación web interactiva desarrollada con Shiny, esta última es capaz de ejecutar SimilaR2 y mostrar los resultados en un interfaz web.

En el **capítulo 6** se realiza un análisis estadístico de los resultados obtenidos por la herramienta SimilaR2. Para ello, se construye un dataset de archivos R categorizados en originales / plagios para así obtener una variable respuesta, sobre el dataset se aplica la herramienta SimilaR2 y se realiza una serie de transformaciones para obtener las variables predictoras. El marco de datos (variable respuesta y predictora) es modelizado mediante Naive Bayes aplicando validación cruzada. Asimismo, se obtienen varios conjuntos de resultados, ejecutando SimilaR2 sobre el dataset con diferentes parámetros. Finalmente, se efectúa un análisis de clústering para determinar los grupos de plagio / no plagio en el dataset.

Por último, en el **capítulo 7** se enumeran las conclusiones sobre el trabajo realizado y se plantean algunas líneas de trabajo futuro.

Abstract

This eassay based on the development of a plagiarism detection tool in R code and RMarkdownn. The aim is to provide teaching staff with an open source and On Premise software capable of detecting plagiarism in deliveries with this extension.

The content of this work has been divided into 7 chapters as follows:

Chapter 1 provides an introduction to plagiarism, plagiarism in source code and how plagiarism is combated within universities.

Chapter 2 describes the general aspects of the state of the art analyses different programmes for detecting plagiarism in source code, highlighting their main qualities and analysing their shortcomings. Finally, an analysis of the state of the art is carried out, serving as a fundamental argumentation for the development of the software tool presented in this dissertation.

Chapter 3 focuses on the SimilaR package, already presented in the previous chapter. This package is perhaps the most widely used at present for the detection of plagiarism in R code. In this chapter, its features and limitations are studied in detail, with the aim of proposing improvements.

Chapter 4 is dedicated to explain the details of the new implementation. The new tool, which we are going to call SimilaR2, is able to detect plagiarism in comment sections thanks to the implementation of fingerprinting algorithms, as well as to extend the areas of R code where plagiarism can be detected (SimilaR can only detect plagiarism in the body of functions, but not in the code written outside them). Additionally, the output of SimilaR2 is explained, which provides detailed information on plagiarism detection analysis, clustering and word cloud analysis, among other possibilities.

During **chapter 5**, the process of building the SimilaR2 R package is detailed, as well as an interactive web application developed with Shiny, the latter being able to run SimilaR2 and display the results in a web interface.

In **chapter 6**, a statistical analysis of the results obtained by the SimilaR2 tool is carried out. To do so, a dataset of R files categorised into originals/plagiarism is constructed to obtain a response variable, the SimilaR2 tool is applied on the dataset and a series of transformations are carried out to obtain the predictor variables. The data framework (response and predictor variable) is modelled by Naive Bayes applying cross-validation. Several sets of results are obtained by running SimilaR2 on the dataset with different parameters. Finally, a clustering analysis is performed to determine the plagiarism/non-plagiarism groups in the dataset.

Finally, **chapter 7** lists the conclusions of the work carried out and suggests some lines of future work.

Aportes

Los principales aportes de este TFG son los siguientes:

- Análisis del estado del arte existente en detección de plagio, tanto en documentos de texto plano, como en código fuente. Análisis de las ventajas y desventajas de cada solución.
- Análisis detallado de las funcionalidades y limitaciones del paquete SimilaR, que es un paquete ampliamente utilizado para la detección de plagio en código fuente R. Dicho paquete sirve de base para el desarrollo de software realizado en este TFG.
- Desarrollo de un nuevo paquete denominado SimilarR2 que extiende y mejora las funcionalidades de SimilaR. En particular, el paquete SimilaR2 incluye:
  - Detección de plagio en los comentarios en código R.
  - Detección de plagio en todas las zonas del código R, pues el anterior paquete SimilaR se limita a la detección de plagio en el cuerpo de funciones.
  - Generación de resultados estadísticos, incluyendo análisis de clustering y nubes de palabras entre otros.
- Desarrollo de un interfaz web interactivo para SimilarR2, utilizando la herramienta Shiny.
- Análisis estadístico de resultados utilizando un dataset de archivos R categorizados en originales / plagio.
- Elaboración de un repositorio en Gitlab con todo el código del proyecto, documentación y pruebas TFG-plagio..
- Subida del paquete SimilaR2 al repositorio gitlab, accesible con la instalación de la librería devtools y ejecutando la instrucción `devtools::install_gitlab('igandullo88/similar2_package')`).

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Plagio

¿Qué es el plagio? Al hecho de manifestar la autoría de un trabajo, unas ideas o de las palabras de otra persona se denomina plagio.

Siempre que se utilice información de un libro, una web, etcétera y no se cite la fuente de donde proviene se considera plagio. Puesto que la información es de otra persona, esas palabras le corresponden y no pueden utilizarse sin su permiso.

Según el Real Decreto Legislativo 1/1996, de 12 de abril, de la Ley de Propiedad Intelectual (P.I.), el plagio es una infracción del derecho de autor sobre una obra de cualquier tipo, este ocurre cuando se copia la misma, sin que la persona que es dueña lo autorice, y se atribuye la autoría. Según los Art. 6.1 y 138 de la P.I. el autor original puede reclamar la paternidad de su obra ante la justicia.

Si existe ánimo de lucro y prejuicio de un tercero, es decir, si violenta los derechos morales y patrimoniales del autor original, el Código Penal art. 270.1 lo considera delito. Puesto que usurpa la autoría del autor y se defraudan sus intereses económicos.

1.1.1. Plagio dentro de las universidades

Los estudiantes de colegios e institutos cada vez tienen más normalizado el uso de internet para buscar información y realizar tareas y trabajos. También, son cada vez más los centros docentes de primaria y secundaria los que animan a sus alumnos a utilizar la web como única fuente de información y de trabajo. Esto hace que cuando los estudiantes llegan a la universidad, tengan completamente normalizado el hecho de proveerse de información digital para realizar sus tareas. En otras palabras, el hecho de hacer una tarea o trabajo a base de copiar y pegar textos de internet es un mal hábito cada vez más extendido entre los universitarios, esto se intensifica para aquellos alumnos que no tienen interés académico.

¿Cómo intentan combatir el plagio los docentes?

La mayoría de las universidades tienen programas informáticos como Turnitin [19] o Urkund [20] capaces de detectar plagio; se basan en comparar el trabajo con indicios de plagio, con una base de datos bastante extensa, devolviendo el porcentaje de similitud que se ha encontrado. Estos presentan las siguientes características:

- Comparan párrafos completos, es decir, son sensibles a cualquier cambio en un párrafo (sinónimos, intercambios de lugar de palabras, entre otros).
- Admiten archivos de tipo común (.txt, .pdf, .ppt, .docx).

Los resultados proporcionados por cualquier herramienta antiplagio se deben interpretar de forma orientativa, recayendo la categorización definitiva en el docente (factor humano). En la detección de plagio manual se han catalogado dos tipos de supuestos:

1. **Caso 1**. El individuo no realiza cambios del texto plagiado. Si la copia se ha realizado mediante búsquedas en web públicas, basta con rastrear del texto sospechoso en el motor de búsqueda *Google* para detectar el plagio.
   Por otro lado, los programas Turnutin o Urkund detectarían con facilidad el plagio y sus resultados serían muy fiables.
2. **Caso 2**. El individuo realiza pequeños cambios en las líneas de los párrafos o emplea sinónimos. En este caso, la detección de plagio se vuelve una tarea complicada y los softwares anteriormente citados no serían 100 % eficaz, ya que son sensibles a dicho cambios.

Para detectar cambios complejos, caso 2, se requiere de un tiempo y esfuerzo considerable, ya que se han de realizar una cantidad importante de búsquedas y comparaciones en cada documento.
El personal docente tiene numerosos frentes abiertos (tareas, tutorías, proyectos de investigación, entre otros) durante el curso, por lo que no dispone de tiempo natural para realizar labores complejas de detección.

Cuando en las universidades se **detecta un plagio**, se castiga al impostor con la suspensión de la asignatura. Puesto que el reglamento en vigor es el *Reglamento de Disciplina Académica de 1954*, una ley obsoleta y sumamente ambigua de aplicar.

1.2. El plagio en código fuente

El plagio en código fuente es copiar/reproducir el código de otra persona sin mencionar la autoría del programador del código original.

Si el autor del código original reclama la paternidad del código ante la justicia, y de demostrarse la autoría del denunciante, el impostor es sancionado según la ley de Propiedad Intelectual; si existiera ánimo de lucro con prejuicio al autor del código original, el denunciado recae ante un delito de usurpación de autoría y defraudación de los intereses económicos del denunciante (art. 270.1 del Código Penal).

Plagio en código fuente dentro del ámbito académico y profesional

La programación se ha convertido en un lenguaje más, por lo que ha aumentado el número de estudiantes que optan por enseñanzas relacionadas con este sector. A su vez, ha incrementado el número de alumnos con acusaciones de plagio. Debido al mal hábito de utilizar el código programable de otra persona sin mencionar su autoría.
El pensamiento de muchos de estos alumnos es: " *Si un código está situado en un repositorio público se puede copiar sin mencionar su autoría.*". Sin embargo, al igual que cualquier trabajo de investigación hay que mencionar al programador del código fuente original.

En el ámbito laboral existe un crecimiento exponencial en la búsqueda de programadores, encontrándose a veces con un déficit de demanda para cubrir tanta oferta. Lo que conlleva a las empresas a contratar personal no cualificado, por tanto, obtienen resultados no satisfactorios, ya que este tipo de perfiles suelen recaer en malas prácticas y no programan los códigos de forma adecuada.

En ambos ámbitos, plagiar el código de otra persona puede tener graves consecuencias. Desde la cancelación de un artículo, rechazo de un programa o suspensión de una materia, a problemas más serios como infracción de derechos de autor o una demanda. Estas malas prácticas pueden afectar seriamente a la carrera del programador, así como a su reputación.

Dado que la programación es un lenguaje cada vez más extendido, siempre es mejor asegurarse de escribir los códigos adecuados. Si bien esto se puede detectar manualmente de manera similar al plagio en lenguaje natural, la mejor solución al problema es usar herramientas automáticas de detección de plagio y posteriormente comprobar los resultados manualmente.

Detección de plagio en la Universidad de Sevilla

La Universidad de Sevilla tiene dos herramientas para la detección de plagio:

1. **Programa antiplagio Turnitin** [19]. Adquirido por el Vicerrectorado de Investigación y coordinado y gestionado por la Biblioteca de la Universidad de Sevilla, es un software eficiente en la detección de plagio. Ha sido creado para los docentes responsables de trabajos académicos (Tesis doctorales, TFG y TFM) de la Universidad de Sevilla.
   Turnitin permite detectar plagio comprobando las coincidencias de un documento con múltiples fuentes de información (internet, artículos científicos y con su base de datos interna). También reconoce el contenido no original traducido del inglés. La herramienta facilita al docente un informe de originalidad resaltando el porcentaje de similitud del documento y mostrando las fuentes originales. El programa permite además hacer comentarios (retroalimentación) y evaluar los trabajos académicos.
2. **SafeAssign** [8]. Los docentes pueden utilizar esta herramienta para la detección de plagio en las entregas realizadas mediante Enseñanza Virtual. SafeAssign es una herramienta de control antiplagio basada en un algoritmo de coincidencia de texto único capaz de detectar coincidencias exactas e inexactas entre un documento y el material de referencia. Los profesores pueden usar el servicio SafeAssign para comprobar la originalidad en las tareas enviadas. SafeAssign compara las tareas enviadas con respecto a un conjunto de documentos académicos para identificar áreas que coinciden entre las tareas enviadas y los trabajos existentes. El docente de la asignatura decide si utiliza o no esta herramienta antiplagio, en el caso de utilizarla, puede decidir también si quiere que sus alumnos vean el informe de originalidad de Safeassign de sus intentos.

Turnitin y SafeAssign no están especializadas en la detección de plagio en código fuente, solo admiten archivos de tipo común (.txt, .docx, .pdf, .ppt). Esta premisa ha sido unos de las motivaciones del tema y objetivos de este trabajo; **construir una herramienta de detección de plagio en código fuente** que pueda ser usada por los docentes de la Universidad de Sevilla.

Capítulo 2

Estado del arte

En este capítulo se realiza un estudio exhaustivo de las herramientas existentes sobre análisis de plagio en código fuente. Hoy en día, es complicado encontrar un tema de investigación que no se haya documentado de forma adecuada. Por tanto, antes de comenzar cualquier tipo de estudio, hay que analizar detalladamente los trabajos existentes en la temática. Sería irrazonable no hacerlo, pues se podrían dar resultados que ya han sido presentados antes o, coloquialmente hablando, *reinventar la rueda*. Por otra parte, el análisis del estado del arte permite detectar los puntos débiles de los estudios existentes y encontrar puntos de mejora.

2.1. Análisis del estado del arte

En el análisis del estado del arte se documentan las herramientas de detección de plagio en código fuente más relevantes para este Trabajo Fin de Grado. El estudio se centra en dos grupos de herramientas: **Programas multilenguaje** (herramientas capaces de detectar plagio en varios lenguajes de programación) y **programas especializados en un lenguaje de análisis de datos** (detectan plagio en un lenguaje de programación orientado al análisis de datos). Todos los programas, indiferentemente del grupo, se analizan como sigue:

- **Autor y origen de la herramienta**, repositorios disponibles y finalidad.
- **Utilización**. Se detalla, si es posible, la instalación (o registro) de la herramienta, la interfaz utilizada y los inconvenientes de instalación o interfaz.
- **Funcionamiento**. Se profundiza en el funcionamiento de la herramienta y se detalla los tipos de salidas que ofrece.
- **Técnicas aplicadas en el algoritmo**. Si es posible, se detallan las técnicas que aplica la herramienta.

2.1.1. Programas multilenguaje

Las herramientas multilenguaje de detección de plagio tienen varias características con respecto a las programadas para un lenguaje de computación concreto:

- Detectan plagio en varios lenguajes de programación. Normalmente, cinco o más idiomas.

- Los programas suelen estar diseñados por empresas o universidades. Por tanto, muchos de estos programas no son gratuitos. Por otra parte, su instalación o registro puede ser complejo y suelen requerir de otros softwares para su utilización.
- Las universidades utilizan estas herramientas para detectar copia en los trabajos prácticos de los estudiantes y las empresas para detectar códigos plagiados de softwares.
- Muchas de estas herramientas utilizan como interfaz una web, a la cual solo se tiene acceso mediante un registro. Normalmente la suscripción requiere de un coste monetario.

Efectuar un buen análisis documental de estos programas es difícil, puesto que existen inconvenientes como: Documentación pública escasa, programas de pago, instalación o registro complejo, etcétera. A pesar de las dificultades, se ha conseguido documentar los siguientes programas: MOSS [3] y Jplag [16]. Las herramientas elegidas tienen las siguientes características: Son gratuitas, la dificultad de su instalación o registro es media y no admiten archivos de código R. Por último, la interfaz y la documentación de las técnicas de detección de plagio utilizadas en estas herramientas resultan interesantes para los fines de este Trabajo Fin de Grado.

2.1.1.1. MOSS

Origen y finalidad

MOSS son las siglas de *Measure Of Software Similarity*, fue desarrollado en 1994 por Berkeley. Es una herramienta de detección de plagio capaz de analizar código en multitud de lenguajes de programación, tales como: Haskell, Python, C, C#, Java y VisualBasic, entre otros. Otra característica importante, en comparación a la competencia (CodeGrade, copyleaks, ...), es que es una herramienta gratuita a través de internet desde 1997.

Utilización

Para poder usar esta herramienta hay que registrarse mediante el envío de un email a la dirección de correo electrónico moss@moss.stanford.edu. El cuerpo del mensaje debe ser el siguiente:

registeruser

mail username@domain

El registro en la base de datos de MOOS no está automatizado. Hay que esperar que reenvíen un ID al email proporcionado.

Funcionamiento

Se procede a descargar de la web MOOS el *script de envío* (disponible solo para Linux), hay que modificar el ID por defecto por el recibido, al salvar el documento se desbloquean las funcionalidades de MOOS.
La interfaz web de la herramienta se inicia al ejecutar el script de envío modificado, esta permite cargar los documentos a comparar y efectúa el análisis. Tras la comparación, se muestra una tabla con el porcentaje de similitud de cada par de archivo. Para ver las partes de código plagiadas entre dos archivos, se clicka en el porcentaje que indica el total de plagio de ese par de documentos.

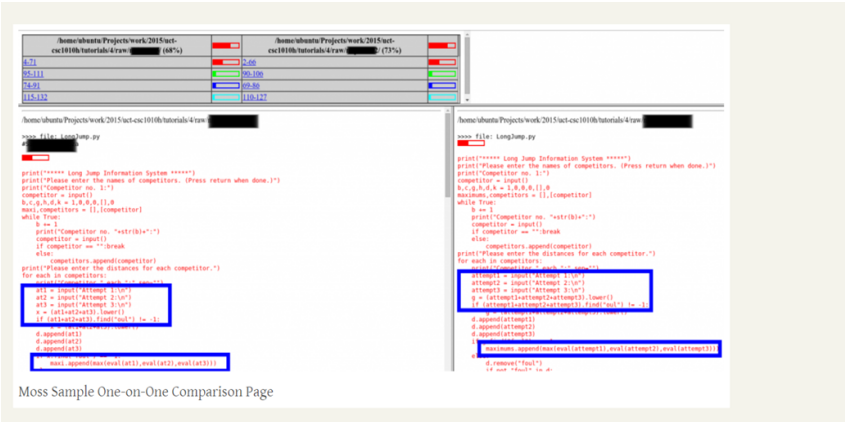


Figura 2.1: Interfaz MOSS

Técnicas aplicadas en el algoritmo

La herramienta utiliza el algoritmo Winnowing [7], una técnica local de fingerprinting de documentos. Esta técnica utiliza la noción de k-grams: un k-gram es una subcadena de longitud k tomada de un texto. Se calcula una secuencia de hashes para todos los posibles k-grams en un texto, luego se toma un subconjunto de estos hashes como las fingerprints del documento.

2.1.1.2. JPLAG

Origen y finalidad

Es una herramienta programada en lenguaje Java por la Universidad de Karlsruhe de Alemania en 2001. Es compatible con Java, C, C#, C++, Scheme y texto en lenguaje natural. Jplag tiene una interfaz web muy dinámica y fácil de entender, aunque esta opción solo la tienen los usuarios que pudieron registrarse. En la actualidad, no hay opción de disfrutar del servicio web de jplag.

Utilización

Para poder usar esta herramienta, se necesita tener instalado Java 7 y descargar jplag del repositorio de github [22]. Tiene una arquitectura de trabajo cliente/servidor. La interfaz funciona a través de línea de comando, o bien, permite la implementación de un cliente propio.

Funcionamiento

No se ha conseguido instalar este programa, puesto que tiene dependencia de JAVA y requiere de unos conocimientos de instalación por encima de los del autor de este trabajo. También, Al no tener la funcionalidad de detectar plagio en ningún lenguaje centrado en el análisis de datos, tales como R software o Haskell, se ha considerado que el esfuerzo y tiempo requeridos para su instalación eran mayores a la información que iba a proporcionar.

Técnicas aplicadas en el algoritmo

JPlag opera en dos fases:

1. Todos los programas que se van a comparar se analizan (o escanean, según el lenguaje de entrada) y se convierten en cadenas de tokens.

2. Estas cadenas de tokens se comparan en pares para determinar la similitud de cada par. El método utilizado es básicamente "*Greedy String Tiling*" [23]: Durante cada una de estas comparaciones, JPlag intenta cubrir una cadena de tokens con subcadenas ("tiles") tomadas de la otra lo mejor posible. El porcentaje de cadenas de tokens que se pueden cubrir es el valor de similitud.

En la tesis "*Herramientas para la detección de plagio de software- F. Javier Diaz*" [5] se realiza un estudio sobre los programas multiarchivo de detección de plagio. Donde se analiza la eficacia de cada programa, mediante una batería de scripts (plagiados y no plagiados), tras el análisis se concluye que el programa que mejor detecta plagio es **Jplag**. Por ello, podemos decir que la técnica de **tokenización** utilizada por el algoritmo jplag es bastante eficiente.

2.1.2. Programas especializados en el análisis de datos

Las características de este grupo de programas son:

- Están programados para un solo lenguaje de programación. Normalmente, se programan en el mismo lenguaje para el que se diseña.
- La mayoría de estas herramientas son gratuitas.
- La interfaz se muestra a través de línea de comandos. Las salidas suelen ser simples y poco detalladas.

En este apartado se documentan los lenguajes de programación Python [15], Haskell [4] y R [13], ya que son lenguajes que tienen muchísimas funcionalidades en el análisis de datos. Para cada lenguaje se analiza la herramienta o herramientas de detección de plagio existentes.

2.1.2.1. Python

Python es uno de los programas más utilizados en el análisis de datos. Existen paquetes en el repositorio de PyPI, como pandas, con multitud de funciones diseñadas para el análisis de datos.

La herramienta analizada en este lenguaje es **pycode_similar** [21]. Software diseñado para cuantificar la similitud entre dos archivos .py.

Origen y finalidad

Se encuentra en el repositorio PyPI de python, en la versión 1.4 publicada el 18 de agosto de 2020, por los programadores con usuarios *frystone* y *thekulu* en GitHub. La herramienta está programada íntegramente en python.

Utilización

Se debe tener instalado python, no requiere ninguna versión en concreto. Se puede usar **pycode_similar** de tres maneras distintas:

- Si se tiene instalado pip, se puede instalar el módulo con la instrucción pip install pycode_similar.
- Se puede cargar como una biblioteca de python mediante la instrucción import pycode_similar.

- Al no tener dependencia de terceros, se puede cargar el archivo pycode_similar.py.

Funcionamiento

Cuadro 1

Para entender las expresiones de plagio que realizan los autores de pycode_similar en la documentación, hay que resaltar de ésta las siguientes definiciones:

- Dado dos scripts de python, uno se denominará código de referencia y el otro código candidato.
- Plagio es la cantidad de código referenciado plagiado por el código candidato.

Para ver el funcionamiento de la herramienta, hay que cargar el módulo y proporcionar el directorio donde se encuentran los dos archivos a comparar. Por último, se ejecuta el siguiente comando:

pycode_similar .detect([archivo_referencia.py, archivo_candidato.py])

La función devuelve el porcentaje de similitud que existe entre el código referencia y el código candidato.

La interfaz de esta herramienta es a través de línea de comando. Por tanto, no es posible ver las partes de códigos plagiadas.
Por otro lado, los autores no especifican si el algoritmo tiene en cuenta los comentarios en la detección de plagio. Para solventar esta cuestión, se ejecuta pycose_similar sobre dos scripts y se obtiene el porcentaje de similitud, a su vez, se añaden comentarios (con partes plagiadas) a ambos scripts y se ejecuta pycode_simlar obteniéndose el mismo resultado. Por tanto, se deduce que esta herramienta ignora los comentarios.

Técnicas aplicadas en el algoritmo

El algoritmo normaliza la representación AST [14] (árboles de sintaxis abstracta) de Python y usa difflib para obtener la modificación del código referenciado al código candidato.

En el algoritmo se han implementado dos métodos de diferencia (diff): diff basado en líneas (UnifiedDiff) y diff basado en la distancia de edición de árbol (TreeDiff), ambos se ejecutan en el procedimiento AST.

1. **UnifiedDiff**: Función normalizada de diferencia de líneas de cadena AST.
2. **TreeDiff**: Función diff AST, lento y el resultado no es bueno para funciones pequeñas (depende del paquete zss).

2.1.2.2. Haskell

En el lenguaje de programación Haskell existen dos tesis referentes a la creación de una herramienta para la detección de plagio: *Plagiarism Detection for Haskell with Holmes* [11] y *Plagiarism detection in Haskell programs using call graph matching* [12]. La primera tesis es una implementación de la herramienta MOSS en Haskell, efectuando algunas mejoras. La segunda tesis es perfeccionamiento de la herramienta Holmes, perfeccionando el algoritmo mediante grafos de llamada.

En el análisis documental de las herramientas de este lenguaje se han encontrado varios inconvenientes: La limitación del autor del presente Trabajo Fin de Grado en la programación del lenguaje Haskell y la dificultad de obtener los módulos de las herramientas mencionadas en las tesis [11] y [12]. A consecuencia de estos contratiempos, solo se documentará las técnicas aplicadas en el algoritmo de la tesis [11].

Técnicas aplicadas en el algoritmo

La herramienta Holmes consta de dos partes: un preprocesador (Sherlock) y una parte de comparación (Holmes).

La primera parte, Sherlock, está construida sobre el analizador Hellium [6] que es un subprograma del lenguaje Haskell y un compilador diseñado específicamente para enseñar Haskell, desarrollado por la Universidad de Utrecht. Hellium puede proporcionar mensajes de error mucho más útiles y específicos para que los estudiantes aprendan más rápidamente. Sherlock usa el analizador Hellium para extraer abstracciones de envíos de programas: versiones normalizadas del código fuente con características irrelevantes eliminadas, como comentarios, funciones no utilizadas (código muerto) y funciones de plantilla. Estas características se filtran para reducir la complejidad del problema y mejorar la búsqueda de similitudes en la segunda fase.

La segunda parte, Holmes, toma toda la información almacenada por el preprocesador y realiza diferentes tipos de comparaciones entre envíos. Se realizan cuatro tipos de análisis:

- **Análisis de cadenas de texto**: mediante coincidencia aproximada de cadenas, se analizan todas las cadenas de texto literales y todas las secciones de comentarios que aparecen en los scripts comparados.
- **Análisis de flujo de tokens** donde el código fuente se transforma en una secuencia de tokens normalizada en la que se eliminan los espacios en blanco, las cadenas de texto literales y los identificadores se reemplazan por símbolos y los símbolos específicos del idioma, como los corchetes, se conservan. Tanto los flujos de tokens en los que los nombres de las funciones estaban ordenados como los flujos en los que las funciones no estaban ordenadas se comparan entre los scripts utilizando Haskell diff library.
- **Métricas del gráfico de llamadas**: los grafos de llamada estáticos se construyen solo para las funciones de nivel superior, donde cada nodo del gráfico representa una función y cada arista representa una función que llama a otra función. Para cada tipo de grado de los nodos (grado de entrada, grado de salida y grado total) se calcula el mínimo, el máximo y la media, se calculan y se comparan los scripts. También se calcula y compara el diámetro de cada grafo, que es la longitud del camino más largo entre todos los caminos más cortos entre nodos.
- **Coincidencia de fingerprints de documentos**, se implementa winnowing de la misma manera que en MOSS, utilizando 25 grams para el hash y un tamaño de ventana de 5.

La investigación y la experimentación de esta tesis concluye que, de todas las técnicas empleadas por Holmes, solo el análisis de **flujo de tokens** y la **comparación de fingerprints** tienen un éxito razonable cuando se aplica a un testeo.

2.1.2.3. R software

Origen y finalidad

El lenguaje de computación R tiene implementada una biblioteca para la detección de plagio en código fuente R, que se encuentra en el repositorio CRAN con el nombre de **SimilaR** [1]. La versión disponible es la 1.0.8, esta actualización se publicó en CRAN, el 26 de junio de 2020, por el autor Maciej Bartoszuk. Requiere de una versión de R, 3.1.0 o superior.

La librería SimilaR utiliza un algoritmo capaz de cuantificar la similitud de las funciones R, basándose en grafos de dependencia. Su uso está diseñado para la detección de clones de código para mejorar la calidad del software, así como para detectar plagio en las tareas de los estudiantes.

Utilización

Para poder utilizar esta herramienta, se debe disponer del programa R en una versión 3.1.0 o superior. El proceso de instalación de SimilaR es: Abrir R y en línea de comando, ejecutar la instrucción '*install.packages("SimilaR")*', siendo necesario disponer de conexión a internet para descargar la librería SimilaR del repositorio CRAN. Tras la instalación, se utiliza la instrucción '*library(SimilaR)*' para hacer uso de la librería.

Funcionamiento

La herramienta está escuetamente documentada, siendo necesario realizar varios ejemplos y analizar sus resultados para entender su funcionamiento. Para ello, se analizan las funciones de la librería.

En la ayuda de la librería, '*?SimilaR_fromDirectory*', podemos ver que dispone de dos funciones *SimilaR_fromTwoFunctions* y *SimilaR_fromDirectory*:

- **SimilaR_fromTwoFunctions**. Está programada para detectar plagio en dos funciones de R. En su salida se muestra un data.frame con cuatro columnas, las columnas `name1` y `name2` corresponde con los nombres de las funciones comparadas. En `SimilaR` se muestra la proporción hallada por el algoritmo SimilaR. En la columna `decision` representa la decisión tomada por la herramienta en función del coeficiente SimilaR, si se obtiene un coeficiente superior a 0.7 el algoritmo considera existencia de plagio, 1, si se obtiene un valor inferior retorna 0.

```
a <- function(a, b){
  j = a + b
  g = j * 2
  g
}

b <- function(c,d){
  c * d
}

SimilaR::SimilaR_fromTwoFunctions(a, b)
```

name1	name2	SimilaR	decision
a	b	0.9	1

Figura 2.2: SimilaR_TwoFunctions (salida)

- **SimilaR_fromDirectory**. Su funcionalidad es detectar plagio entre funciones definidas en archivos R. Se debe proporcionar un directorio donde se hallen los scripts de R a comparar. Esta función compara dos a dos cada una de las funciones de los archivos R del directorio, para cada comparación la salida es igual que la mostrada por **SimilaR_fromTwoFunctions**.

Como se aprecia en 2.2, la interfaz es mediante línea de comandos. Por tanto, no hay posibilidad de conocer las partes de códigos plagiadas, prestación que sí ofrecen herramientas como MOSS. Otras limitaciones encotradas para esta herramienta son:

- No detecta plagio en secciones de comentarios, ni en partes de código que no sean funciones.
- No calcula la proporción de copia total entre documentos, es decir, si se comparan varios scripts, p.e. 1,2,3, no devuelve la proporción de plagio global entre 1-2, 1-3 y 2-3 (este prodría ser el promedio de coeficientes SimilaR entre cada par de archivos), habría que calcularlo manualmente.
- Si existen funciones no hace distinción de nombres, es decir, las funciones que se han reescrito con el mismo nombre se muestran en la salida con el mismo identificador. Esto dificulta la comprensión del análisis.

Técnicas aplicadas en el algoritmo

Utiliza grafos de dependencia para cuantificar la similitud de las funciones.

No se dispone de más información en la documentación de la herramienta.

2.2. Conclusiones

El estado del arte ha tenido como objetivo documentar las herramientas existentes de detección de plagio en código fuente, con la finalidad de recabar información para construir o mejorar una herramienta de detección de plagio en código R.

Tras efectuar el análisis, se ha podido comprobar que la herramienta de detección de plagio en código R presenta algunas limitaciones funcionales, en comparación con otras herramientas de detección de plagio en código fuente. Entre las limitaciones, se puede destacar: Una interfaz simple a través de líneas de comando, imposibilidad de detectar plagio en secciones de comentarios y en código de programación que no es función. También presenta carencias en la documentación, donde no profundiza en las técnicas de detección de plagio que realiza el algoritmo.

La ausencia de documentación sobre las técnicas de detección de plagio que realiza el algoritmo SimilaR, dificulta la construcción de una nueva herramienta. Por tanto, en este trabajo se aportan mejoras complementarias para dicha herramienta.

2.3. Objetivos

El objetivo general de este trabajo es mejorar las prestaciones funcionales de la herramienta para detección de plagio en código R, SimilaR. Para la implementación de estas mejoras se ha construido una librería R, la cual tiene una función capaz de mostrar cada una de las siguientes mejoras:

- **Capacidad de detectar plagio en secciones de comentarios**. Se implementan en R algunos de los algoritmos documentados en el estado del arte para detectar plagio en los comentarios.
- **Detección de plagio en todo el código R**. Se realiza una modificación del contenido de cada script, para que SimilaR consiga detectar plagio en todo el código de programación.
- Se añade la funcionalidad de **detectar plagio en archivo Rmd**.
- **Salida más clara e informativa**, la cual muestra información relevante, tal como la proporción total de plagio en cada par de scripts comparados, así como información sobre la detección de plagio en secciones de comentarios y en todo el código de programación (funciones y no funciones).
- **Una interfaz vistosa, clara y detallada**. Se crea una interfaz mediante Shiny, que muestra con detalles todos los puntos anteriores.
- **Análisis estadístico**. Se comprueba la fiabilidad de la nueva herramienta.

Capítulo 3

La herramienta SimilaR para detección de plagio en código R

En este capítulo se realiza un análisis exhaustivo de la herramienta SimilaR, 2.1.2.3. Este se divide en dos bloques, el primero basado en el estudio de las funcionalidades y el segundo en las limitaciones vistas en el apartado, ya citado, del estado del arte:

- No detecta plagio en bloques de comentarios, ni en las partes de código que no son funciones.
- Solo devuelve las proporciones de plagio existente entre las funciones de dos documentos. Ejemplo, dado dos documentos $A(f_1, f_2)$ y $B(f_3)$, donde f son las funciones de cada archivo. Las comparaciones realizadas por SimilaR son f_1, f_3; f_2, f_3. La proporción de plagio entre ambos documentos no la calcula.
- Para funciones con el mismo nombre, ejemplo $A(f_1, f_1)$, no hace distinción de nombres en la salida (aparecen con el mismo identificador). Veremos que esto resulta muy confuso a la hora de interpretar el data.frame de salida.

Finalmente, a lo largo del capítulo y especialmente en la última sección del mismo, se proponen algunas soluciones a las limitaciones existentes en SimilaR, que servirán de base para la implementación explicada en el siguiente capítulo.

3.1. Funcionalidades y limitaciones de SimilaR

En el repositorio principal pruebas_SimilaR se han creado diferentes scripts de R para analizar las funcionalidades y limitaciones que ofrece SimilaR. En este capítulo, cualquier directorio adicional se encuentra dentro del repositorio principal.

3.1.1. Funcionalidades

El objetivo de estudiar las funcionalidades de SimilaR es comprender cada una de las prestaciones de la herramienta. Esto ayuda a conocer las funciones y parámetros más adecuados en la comparación de documentos.

SimilaR tiene varias prestaciones que no se analizaron en el estado del arte, como son el ofrecimiento de dos tipos de salidas, dos formas de comparar funciones y varias formas de calcular los coeficientes.

- **Salidas**. El parámetro **returnType** puede recibir dos valores ("data.frame", "matrix"). Por defecto, recibe `data.frame`. En la figura(3.1) se puede ver las salidas para cada caso. Para `data.frame` se muestra un marco de datos que da la información sobre la similitud de los pares de funciones inspeccionadas, en el caso de `matrix` devuelve una matriz cuadrada, donde el elemento en el índice (i,j) equivale al grado de similitud entre la función i-ésima y la j-ésima.

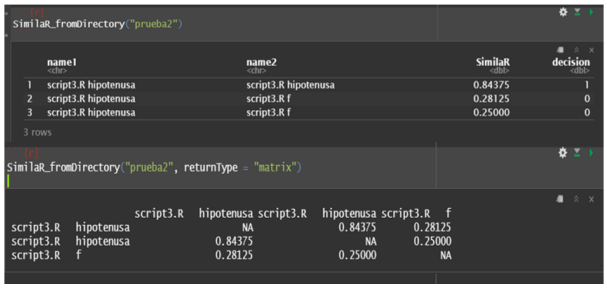


Figura 3.1: Salidas SimilaR.returnType

- **Tipo de análisis**. El parámetro **fileTypes** admite dos opciones ("function", "file"). `function` compara cada función con cualquier otra función; `file` compara solo las funciones definidas en diferentes archivos fuente.
  Para `fileTypes = "function"` admite analizar solo un archivo, mientras que para `file` debe ser un directorio con al menos dos archivos. Para entender las comparaciones que realiza el algoritmo Similar, véase figura 3.2. Para ambos casos, se puede calcular el número total de comparaciones de la siguiente forma:
  - Si denotamos n = nº de funciones de FILEA y m = nº de funciones de FILEB.
  - Para `file` se consigue n x m comparaciones.
  - En el caso de `function` se obtiene n x $m + \sum_{i=1}^{n-1} n-i + \sum_{j=1}^{m-1} m-j$ comparaciones.

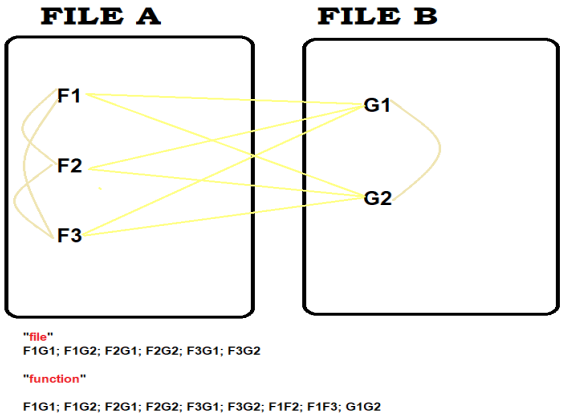


Figura 3.2: Comparaciones SimilaR. typeFile

- **Modelos de asimetría**. El parámetro **aggregation** sirve para especificar el grado de asimetría que se va a utilizar (`sym`, `tnorm`, `both`). `sym` significa que se calcula

un grado de similitud (global); `both` evalúa y devuelve el grado en que la primera función, de un par de funciones, es similar a la segunda y, por separado, el grado en que la segunda función es similar a la primera; `tnorm` calcula dos valores de similitud y los agrega a un único número, es decir, calcula la media de los valores obtenidos con el modelo de asimetría `both`.
En la figura 3.3, se puede observar los resultados de aplicar SimilaR en el directorio `prueba1` con cada modelo de asimetría.

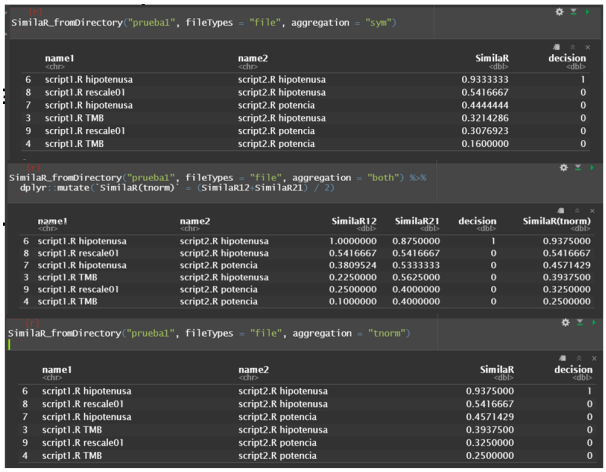


Figura 3.3: SimilaR. Modelos de asimetría

Elección de la función y los parámetros de SimilaR para la comparación de archivos

Uno de los objetivos de este trabajo es proporcionar una medida global para la detección de plagio entre dos archivos, por tanto, solo es de utilidad la función **SimilaR_fromDirectory**. Esto también implica que `typeFile="file"`. Para el parámetro `returnType`, se utiliza `data.frame`, ya que proporciona más información y la obtención de resultados globales es menos compleja. En cuanto a la elección del modelo asimétrico, se usa `both`, porque proporciona dos valores de similitud para cada par funciones.

Nota: A partir de este punto, cuando se haga mención a SimilaR, se referirá al uso de función SimilaR_fromDirectory.

3.1.2. Limitaciones

Se analiza las carencias de SimilaR en la detección de plagio entre pares de archivos.

No analiza ni comentarios ni código que no es función

En el directorio prueba1 existen dos archivos `("script1.R", "script2.R")`, sus contenidos son funciones, comentarios y código que no es función, dentro de este directorio existe otro directorio, `modificación`, donde se encuentran los archivos `("script1_mod.R", "script2_mod.R")`, sus contenidos son las mismas funciones de los "script1.R" y "script2.R" respectivamente.
Al aplicar SimilaR en cada directorio, se obtienen los data.frames que se muestran en la figura 3.4. En la imagen se puede observar que los data.frames son idénticos. Por tanto, se puede afirmar que SimilaR solo analiza funciones.

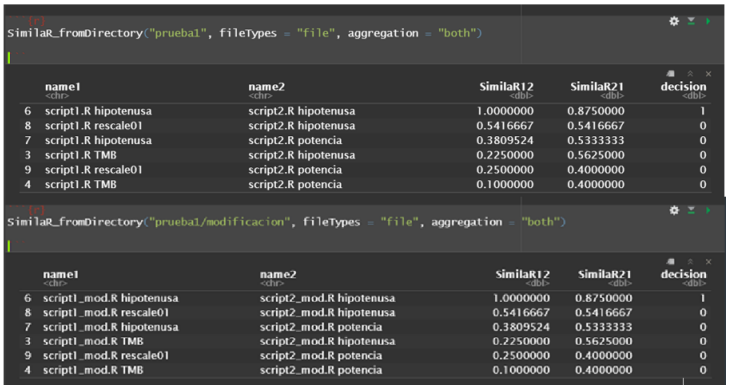


Figura 3.4: Solo analiza funciones (prueba1 / modificacion)

No se obtiene una proporción para la comparación global de los pares de archivos

Como ya se ha mostrado en diferentes salidas de SimilaR, p.e. figura 3.4, no muestra ninguna proporción global para la comparación de archivos. Solo se muestran los coeficientes SimilaR para cada par de funciones.

Propuesta

Una propuesta para obtener una proporción de similitud de plagio entre cada par de archivos es:

$$
\begin{aligned}
&\varsigma_l^1;\ l = 1, ..., nxm \\
&\varsigma_l^2;\ l = 1, \cdots, nxm \\
&\varsigma = E[\varsigma_1^1, \varsigma_1^2], \cdots, E[\varsigma_{nxm}^1, \varsigma_{nxm}^2] \\
&\overline{\varsigma} = E[\varsigma] = E[\varsigma_1, \varsigma_2, \cdots, \varsigma_{nxm-1}, \varsigma_{nxm}] = \frac{\sum_{l=1}^{nxm} \varsigma_l}{n \cdot m}
\end{aligned}
\tag{3.1}
$$

Siendo ς_l^1 los coeficientes SimilaR12, ς_l^2 los coeficientes SimilaR21 y ς son los coeficientes SimilaR que se obtienen por el modelo asimétrico `tnorm`.
La medida $\overline{\varsigma}$ es simétrica, es decir, se obtiene el mismo valor comparando los archivos A-B que B-A.
No obstante, se va a demostrar que es una mala medida para la comparación de dos archivos. Para ello, se ha creado un directorio, `prueba3`, donde se encuentran el archivo `script1.R` y una copia de este `script1-copia.R`. Al ser los mismos archivos $\overline{\varsigma}$ debería ser igual a 1. En cambio, como se observa en la figura 3.5 se obtiene un valor de 0.6625. Este resultado se debe a que SimilaR compara cada par de funciones; en el único caso que se obtiene 1, en la comparación de dos archivos iguales, es si solo existe una función en cada script.

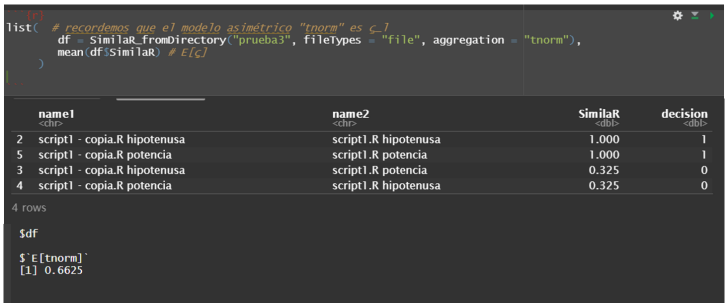


Figura 3.5: Cálculo $\bar{\varsigma}$

No tiene en cuenta las funciones renombradas

En el directorio `prueba4` se han creado dos scritps (`script1.R`, `script2.R`), ambos tienen dos funciones redefinidas, es decir, dos funciones con el mismo nombre. Hay que resaltar que SimilaR ordena las filas del data.frame por los coeficientes SimilaR en orden decreciente, esto hace más complicado entender las salidas con funciones redefinidas.

Por otro lado, se ha creado el directorio `prueba4/modificacion`, donde se encuentran los scripts (`script1_mod.R`, `script2_mod.R`), sus contenidos son las mismas funciones de (`script1.R`, `script2.R`), pero se han añadido subíndices a las funciones redefinidas. Para entenderlo mejor, véase la siguiente figura:

Figura 3.6: Funciones renombradas. (prueba4 / modificacion)

Se puede observar que la salida de SimilaR para `prueba4` es muy compleja de entender, ya que no se entiende que función redefinida se están comparando. En cambio la salida para `prueba4/modificacion` es clara y entendible.

3.2. Soluciones propuestas para superar las limitaciones de SimilaR.

Se enumeran a continuación algunas propuestas para superar las limitaciones de SimilaR.

- Implementar los algoritmos fingerprint [18, pág 1] y winnowing [7] para la detección de plagio en bloques de comentarios.
- Convertir artificialmente y de forma automática las partes de código que no son funciones en funciones. De esta forma, SimilaR sería capaz de analizar todo el código.

- Renombrar las funciones redefinidas utilizando subíndices. A modo y manera que en la figura 3.6.
- Generar los datos más relevantes de la tabla SimilaR original, proporcionando los coeficientes óptimos para calcular una medida para comparar archivos.
- Generar una tabla en donde se resuma la información resultante de la comparación de cada par de archivos.
- Realizar análisis de clústering para determinar grupos de archivos similares.

Capítulo 4

La nueva herramienta SimilaR2

En este capítulo se detallan los procedimientos utilizados en cada implementación de la nueva herramienta. Esta se ha bautizado bajo el nombre SimilaR2, debido a que la librería SimilaR tiene un papel importante en la construcción de la nueva aplicación.

En el repositorio principal R_SimilaR2 se han creado varios scripts de R para mostrar las funcionalidades de la implementación de SimilaR2. En este capítulo, cualquier directorio adicional se encuentra dentro del repositorio principal.

4.1. Detección de plagio en comentarios

El procedimiento implementado para detectar plagio en los comentarios de cada par de documentos R o RMarkdown de un directorio es el siguiente: se aíslan los **comentarios** (1) de los archivos. En segundo lugar, se aplican los algoritmos **fingerprint** y **winnowing** para obtener las marcas características de cada archivo. Por último, mediante el **coeficiente Sorensen** se obtiene una proporción de similitud para cada par de archivos.

Definición 1 La definición de comentario varía según el tipo de archivo.

- **R**. En una línea de código es todo aquello (palabras, números, símbolos, ...) que este situado a la derecha del símbolo #.
- **Rmd**. (Rmd $\rightarrow$ R)[1]. Varía según la disposición del comentario en el archivo Rmd. Si se encuentra **dentro de chunk**, en el archivo de conversión, la definición es igual que para archivos R. Si se localiza **fuera de chunk**, en el script transformado, es igual que en R cambiando el símbolo # por los símbolos #'.

(Rmd $\rightarrow$ R)[1]. *Conversión del archivo Rmd a R con la función purl del paquete knitr.*

El procedimiento comienza aislando los comentarios mediante funciones de cadena de texto (paquete `stringr`). En la siguiente figura se muestra cómo se han separado los comentarios del archivo `prueba1/sript1.R`.

```
# A tibble: 11 x 1
   value
   <chr>
 1 " Esta funcion devuevle el cuadrado de a. Elevando el valor 'a'"
 2 " a la potencia 2"
 3 " funcionA multiplica el valor a por si mismo, y retorna el valor calculado"
 4 " En esta funcion se suma 'a' tantas veces como 'a' sea. Es decir,"
 5 " si a = 5.  t = 5 + 5 + 5 + 5 + 5"
 6 " y por ultimo, se retorna t"
 7 " Se guarda la variable s = de j a 1. Es decir, si j = 5"
 8 " s = 5, 4, 3, 2, 1"
 9 " Y se retorna 's' elevado a potencia 2"
10 " Se calcula 'w', siendo w = raiz cuadrada de y"
11 " Por último, retorna el valor de w redondeado a 0 decimales"
```

Figura 4.1: Comentarios aislados

Los comentarios aislados son sometidos de forma progresiva a los siguientes procedimientos:

$$\text{parsing} \Rightarrow \text{tokenización} \Rightarrow \text{hashing}$$

4.1.1. Parsing

Definición 2 Un parser, en este caso, es aplicar una cadena de instrucciones a una entrada (comentarios), para descomponerse en sus *componentes principales*.

Componentes principales: Son las palabras resultantes de aplicar las cadenas de instrucciones a los comentarios.

Los comentarios aislados son sometidos al siguiente proceso parsing:

- **Conversión a ASCII**. Todo el texto se convierte a codificación *ASCII*. Transformando las acentuaciones de idioma a lenguaje universal. Por ejemplo, `áéíóúñ` → `aeioun`.
- **Plegado de mayúsculas y minúsculas**.Todas las letras deben ser insensibles a las mayúsculas y minúsculas. En este estudio todas las letras se cambian a minúsculas.
- **Filtrado**. También, se denomina eliminación de palabras vacías. Este proceso se divide en dos fases:
  1. Se eliminan signos de puntuación, símbolos, tabulación y saltos de líneas.
  2. Se eliminan las palabras más frecuentes del idioma (preposiciones, artículos, ...).
- **Stemming**. En este paso se reduce las palabras a la raíz. Por ejemplo, "inventado", "inventor", "invención" se reducen y se convierten a su raíz, invent.

Después de aplicar el proceso parsing, se aplica el paso de **pre-tokenización** que es eliminar los espacios en blanco de cada línea del texto.

A continuación se muestra el resultado de aplicar parsing al ejemplo anterior:

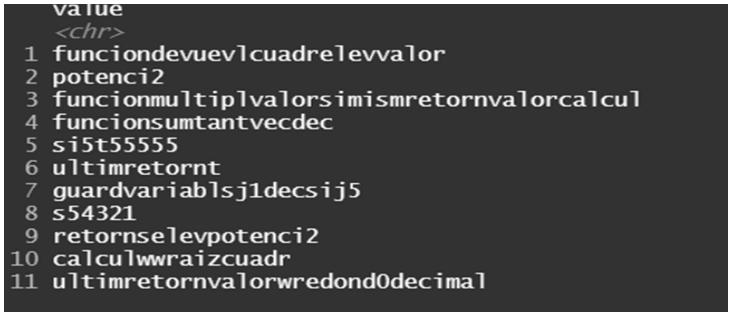


Figura 4.2: parsing en prueba1/script1.R

4.1.2. Tokenización

Definición 3 La tokenización es dividir un texto en pequeños trozos. Por ejemplo: "Está lloviendo", se puede dividir en dos tokens `"Está", "lloviendo"`.

El proceso de obtención de tokens es:

- Se agrupa todas las líneas del proceso anterior en una sola línea (`'texto'`).
- En este paso se define la longitud de cada token y el desplazamiento en cada letra del texto para elegir el siguiente token:
  - Los tokens tiene una longitud constante de tamaño `k`. Formalmente, estas subcadenas de tamaño `k`, se denominan kgrams. El valor de `k` ha sido parametrizado, ya que cada texto es distinto; un texto largo requerirá un `k` más grande y uno más pequeño exigirá un kgram inferior.
  - La elección del desplazamiento en cada letra del texto para adquirir el siguiente kgram si está definida, según los autores del algoritmo Winnowing [17], para que se den condiciones de uniformidad este desplazamiento debe tener una longitud de `1`. Ejemplo: si tenemos el texto Estalloviendo; `k = 4`. Los kgrams son `"Esta" "stal" "tall" "allo" "llov" "lovi" "ovie" "vien" "iend" "endo"`.
    De forma generalizada, el número de kgrams de un texto es `n - k + 1` (n = nº de letras del texto, k = kgram). En el ejemplo anterior `n = 13`, `k = 4`, `nkgrams = 10`.

4.1.3. Hashing

Definición 4 **Función hash**. Convierte un elemento u elementos de entrada a una función en otro elemento. Por ejemplo,

Zorro → Función hash → **DFCD3454**

El procedimiento de hashing se emplea a cada uno de los kgrams de la siguiente forma:

- Se les aplica función hash **md5** con una semilla de `16L`.

- Los valores hash obtenido se convierten a valores hexadecimales.
- Se divide cada valor hex por `257`. Obteniéndose `n -k + 1` hashes en un rango de `0` a `256`.

En el documento [2, pág 130], se detalla que utilizan la función **hash md5** en la implementación de la técnica hashing para el algoritmo Winnowing. También, precisan que el valor hexadecimal obtenido se divide por un cierto número primo (no detallando cual).

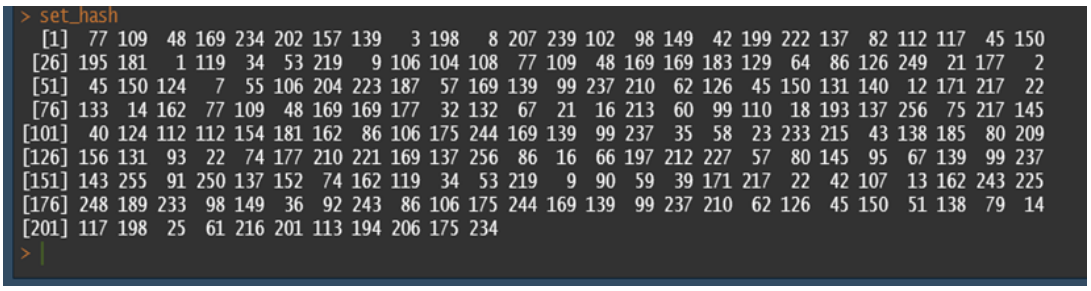


Figura 4.3: Aplicación de hashing en prueba1/script1.R

4.1.4. Algorítmos

Los procesos de parsing, tokenización y hashing forman parte de los algoritmos fingerprint y winnowing, es decir, el preprocesamiento hasta obtener los hashes es el mismo en los dos algoritmos. La diferencia es el método de obtención de las fingerprints finales, se puede decir que las fingerprints son una muestra aleatoria de los hashes. A continuación se explica cómo se obtienen en cada algoritmo:

- **Fingerprint**. Se seleccionan aquellos hashes múltiplos de 4, es decir, las fingerprints son aquellos hashes que al dividirlos por 4 su resto es `0`.
- **Winnowing**. Sigue el método de desplazamiento de ventana, `w = t - k + 1`, donde `k` es la longitud del k-gram y `t` es la longitud mínima de palabra para la detección (`t` *está parametrizado*). Se obtienen ventanas de hashes de tamaño `w` y con desplazamiento `1`. En cada ventana se selecciona el valor mínimo del hash. Si hay más de un hash con el valor mínimo, se selecciona la ocurrencia más a la derecha. Para entender mejor este proceso, observar el ejemplo 1.

Ejemplo 1 Supongamos que $k = 4, t = 7$ y que los hashes, h, son:

$$h = [77, 109, 48, 169, 234, 202, 157, 139, 3]; \quad n_h = 9;$$

Por tanto, tenemos que $w = t - k + 1 = 4 \Rightarrow n_w = n_h - w + 1 = 6$

Las w_i; $i = 1, \cdots, n_w$ ventanas que obtenemos son:

$w_i =$ `[77, 109, 48, 169]; [109, 48, 169 , 234]; [48, 169, 234, 202];`

`[169, 234, 202 , 157]; [234, 202, 157, 139]; [202, 157, 139, 3]`

En forma matricial:

$$W = \begin{bmatrix} 77 & 109 & 48 & 169 \\ 109 & 48 & 169 & 234 \\ 48 & 169 & 234 & 202 \\ 169 & 234 & 202 & 157 \\ 234 & 202 & 157 & 139 \\ 202 & 157 & 139 & 3 \end{bmatrix}$$

Los fingerprints, fp, se obtienen de la siguiente forma:

$$fp = min(W_i) = W_{i,j} \Leftrightarrow W_{i,j} \neq W_{i-1,j+1}$$

Observaciones:

- $W_{i,j} = W_{i,w} = fp$
- $fp = min(W_i) = min(W_{i-1}) \Leftrightarrow W_{i,j} \neq W_{i-1,j+1}$

Las fingerprints de este ejemplo son:

$$W = \begin{bmatrix} 77 & 109 & \mathbf{48} & 169 \\ 109 & 48 & 169 & 234 \\ 48 & 169 & 234 & 202 \\ 169 & 234 & 202 & \mathbf{157} \\ 234 & 202 & 157 & \mathbf{139} \\ 202 & 157 & 139 & \mathbf{3} \end{bmatrix}$$

$$fp = [48, 157, 139, 3]$$

4.1.5. Comparación fingerprints

El último paso, es obtener una medida que indique *la similitud de los comentarios de dos archivos*. Sean FP_A y FP_B las fingerprints de los scripts A y B respectivamente, se consigue una proporción de similitud entre las fingerprints de ambos documentos con el índice Sorensen (coeficiente de similitud-cuantitativo) [9].

Para este trabajo, se utiliza la versión cuantitativa del índice Sorensen, también llamado *Porcentaje de similitud*. Este índice está basado en datos de abundancia y es calculado como:

$$\varphi_{(A,B)} = \frac{(2 \cdot W)}{A + B}$$

Donde W es la sumatoria del valor mínimo de la abundancia entre los documentos comparados para cada fingerprint. A y B es la suma de las abundancias de todas las fingerprints en cada archivo.

Ejemplo 2 Supongamos que para en el archivo A, se han obtenido n-FP$_A$, para el archivo B, n-FP$_B$ y que el número de categorías entre los dos conjuntos de fingerprints es 4 (FP_1, FP_2, FP_3, FP_4). Se contabiliza para cada archivo y categoría cuantas fingerpritns existen. Y se hallan los parámetros A,B y W como se muestra en la siguiente tabla:

Tabla 4.1: Tabla abundancia: Obtención parámetros coeficiente Sorensen

	FP_1	FP_2	FP_3	FP_4	**Medidas**	**Tipo**
A	1	21	11	16	49	A
B	1	8	3	0	12	B
min(FP)	**1**	**8**	**3**	**0**	**12**	**W**

min(FP): Se calcula el mínimo de cada FP
[a] Medidas$_1$: n-FP_A
[b] Medidas$_2$: n-FP_B
[c] Medidas$_3$: n-min(FP)
[*] Tipo: parámetros

Obtenemos el coeficiente Sorensen:

$$\varphi_{(A,B)} = \frac{(2 \cdot W)}{A + B} = \frac{12}{49 + 12} = 0.3934$$

Esto significa que los documentos A y B tienen un porcentaje de similitud en los comentarios del 39.34 %.

Matriz Sorensen

De forma generalizada se obtiene la matriz cuadrada de Sorensen (n x n). Donde n es el número de archivos de un directorio. En esta matriz se representan todos los $\varphi_{i,j}$; $\quad i = 1, \cdots, n$; $\quad j = 1, \cdots, n$, es decir, el coeficiente Sorensen obtenido de la comparación de los archivos i-ésimo y j-ésimo.

$$S = \begin{bmatrix} \varphi_{11} & \varphi_{12} & \cdots & \varphi_{1n} \\ \varphi_{21} & \varphi_{22} & \cdots & \varphi_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \varphi_{n1} & \varphi_{n2} & \cdots & \varphi_{nn} \end{bmatrix}$$

Se realiza la siguiente transformación de S para obtener una matriz de distancia simétricas:

$$d_{i,j} = 1 - \varphi_{i,j}; \quad i, j = 1, \cdots, n$$

Por tanto:

$$\Phi = 1_{nxm} - S = \begin{bmatrix} d_{11} & d_{12} & \cdots & d_{1n} \\ d_{21} & d_{22} & \cdots & d_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ d_{n1} & d_{n2} & \cdots & d_{nn} \end{bmatrix} \tag{4.1}$$

4.2. Detección de plagio en todo el código ejecutable

El procedimiento implementado para detectar plagio en los códigos de cada par de documentos R o RMarkdown de un directorio es el siguiente: Se aplica la función write_func_SimilaR a cada documento para convertir todas las partes de código que no son funciones en funciones, con ello SimilaR puede analizar todo el código. En segundo lugar, se realizan una serie de modificaciones de la salida proporcionada por SimilaR, hallándose la tabla *SimilaR_mod*, mediante la cual se obtienen las medidas τ yβ. Por último, se calcula la medida $\kappa = E[\bar{\tau} + \bar{\beta}]$ utilizada para cuantificar la similitud de plagio entre los códigos de dos documentos.

4.2.1. Implementación de la función write_func_SimilaR

En la función write_func_SimilaR (*wfS*) se ha implementado dos funcionalidades, la primera convierte todo el código que no es función en función y la otra añade subíndices a los nombres de funciones redefinidas. La función wfS recibe como parámetro principal un archivo (R o Rmd) y genera el documento transformado.

4.2.1.1. Conversión full_function (todo el código a función)

wfS lee cada línea del archivo y lo guarda en la variable `text`. A esta se le aplica el siguiente procedimiento:

- Se aplican funciones de cadena de texto (`strignr`) a `text` para separar el contenido en dos bloques (código y comentarios).
- El bloque código se separa en subbloques (Las partes de código que son funciones y las que no lo son).
- Las partes de código que no son funciones se convierten en funciones. Para ello se añade "f_i **= function(){**"; $(i = 1, 2, \cdots)$ al principio de la parte analizada y "**}**" al final.
- En el bloque código se ordenan funciones originales y nuevas, es decir, las funciones que existen en el documento de partida y las que se crean, f_i.
- Por último, se concatenan los bloques código y comentarios.

Se obtiene un código con el mismo contenido que el original, diferenciándose en la conversión a funciones de las partes de código que no son funciones. Con esto se logra que SimilaR pueda analizar todo el código ejecutable de un documento. Antes de salvar el código obtenido en un script .R, se define este código en la variable `wr_scp` y se le aplica la función `change_function` 4.2.1.2.

4.2.1.2. Subíndices para la redefinición de funciones

Esta función recibe el nombre de **change_function**, recibe la variable `wr_scp`, compara los nombres de las funciones y si son iguales, añade subíndices a los nombres.

Ejemplo 3 Transformación de funciones:

Original	FuncionA, FuncionA, FuncionA
Implementación	FuncionA, FuncionA_2, *FuncionA_3*

Tras aplicarse esta segunda función, se guarda el documento con el mismo nombre que el archivo original en el directorio binder (se crea tras aplicar la función wfS al primer documento).

4.2.2. Tabla SimilaR_mod y medida κ

La tabla SimlaR_mod es una modificación de la tabla SimilaR basada en filtrar las funciones de cada par de archivos con mayor coeficiente SimilaR, para las funciones del archivo A se filtran los máximos coeficientes SimilaR12 y para las de B los máximos coeficientes SimilaR21. Se obtiene una tabla con $n+m$ filas, n funciones en el archivo A y m en el B. A continuación se detalla el proceso:

Sea F_i; ($i = 1, \cdots$, n) las n funciones del archivo A. Y sea G_j; ($j = 1, \cdots$, m) las m funciones del archivo B.

En 3.2 se vio las comparaciones que realiza SimilaR (`typeFile = "file"`). De forma generalizada las comparaciones de funciones para ambos archivos son:

$$\begin{array}{cccc} F_1 = \varsigma^1_{1,1}, \varsigma^2_{1,1}; & \varsigma^1_{1,2}, \varsigma^2_{1,2}; & \cdots & ; \varsigma^1_{1,m}, \varsigma^2_{1,m} \\ F_2 = \varsigma^1_{2,1}, \varsigma^2_{2,1}; & \varsigma^1_{2,2}, \varsigma^2_{2,2}; & \cdots & ; \varsigma^1_{2,m}, \varsigma^2_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ F_n = \varsigma^1_{n,1}, \varsigma^2_{n,1}; & \varsigma^1_{n,2}, \varsigma^2_{n,2}; & \cdots & ; \varsigma^1_{n,m}, \varsigma^2_{n,m} \end{array}$$

Siendo $\varsigma^1_{i,j}$ = El coeficiente SimilaR12 resultante de comparar la función i-ésima de F con la j-ésima de G.

Análogamente, $\varsigma^2_{i,j}$ = El coeficiente SimilaR21 resultante de comparar la función i-ésima de F con la j-ésima de G.

Modificaciones. Obtención de tabla SimilaR_mod

1) Se escogen los coeficientes SimilaR12 mayores para las comparaciones F_i, lo denotamos como τ_i; $i = 1, \cdots, n$. Se hallan los $max(\varsigma^1_i)$:

$$\begin{array}{c} \tau_1 = max(\varsigma^1_1) = max(\varsigma^1_{1,1}, \varsigma^1_{1,2}, \cdots, \varsigma^1_{1,m}) \\ \vdots \\ \tau_n = max(\varsigma^1_n) = max(\varsigma^1_{n,1}, \varsigma^1_{n,2}, \cdots, \varsigma^1_{n,m}) \end{array}$$

2) Se seleccionan los coeficientes SimilaR21 mayores para las comparaciones G_j, lo denotamos como β_j; $j = 1, \cdots, m$. Se hallan los $max(\varsigma_j^2)$:

$$\begin{aligned} \beta_1 = max(\varsigma_1^2) &= max(\varsigma_{1,1}^2, \varsigma_{2,1}^2, \varsigma_{3,1}^2, \cdots, \varsigma_{n,1}^2) \\ &\vdots \\ \beta_m = max(\varsigma_m^2) &= max(\varsigma_{1,m}^2, \varsigma_{2,m}^2, \cdots, \varsigma_{n,m}^2) \end{aligned}$$

La tabla SimilaR_mod presenta la siguiente composición:

File1	**Function1**	**File2**	**Function2**	τ	β	**decision**
A	F_1	B	G_α	τ_1	$\varsigma_{1,\alpha}^2$	d
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
A	F_n	B	G_α	τ_n	$\varsigma_{n,\alpha}^2$	d
A	F_r	B	G_1	$\varsigma_{r,1}^1$	β_1	d
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
A	F_r	B	G_m	$\varsigma_{r,m}^1$	β_m	d

$d \in [0, 1]$, $\alpha \in \mathbb{N}[1, \cdots, m]$ y $r \in \mathbb{N}[1, \cdots, n]$.

Nótese, que los $\varsigma_{i,\alpha}^2$ $(i = 1, \cdots, n)$ y $\varsigma_{r,j}^1$ $(j = 1, \cdots, m)$ son comparaciones de las funciones F_i-G_α y F_r-G_j respectivamente. Y son coeficientes que no se utilizan en la obtención de κ.

Se ha elegido está salida de **SimilaR_mod** para ayudar al usuario a identificar las funciones con mayor índice de plagio en ambos documentos.

Obtención κ

La fórmula que se utiliza para hallar κ es:

$$\kappa = E[U, V]$$

Siendo

$$\begin{aligned} U = E[\tau] &= E[\tau_1, \cdots, \tau_n] \\ V = E[\beta] &= E[\beta_1, \cdots, \beta_m] \end{aligned}$$

La medida κ es una medida simétrica. En el apartado 3.1.2 se demostró que al comparar dos archivos iguales se obtiene que $\bar{\varsigma} \neq 1 \Leftrightarrow n_F + m_G > 2$. Por el contrario, con la medida κ se obtiene un valor de `1` (observar tabla 4.2). Con esto se concluye que la medida κ es óptima para detectar plagio en el código de cada par de archivos.

file1	function1	file2	function2	SimilaR12	SimilaR21	decision
τ						
script1.R	funcionA	script2.R	funcionA	1	1	1
script1.R	funcionE	script2.R	funcionE	1	1	1
script1.R	funcionD	script2.R	funcionD	1	1	1
script1.R	funcionB	script2.R	funcionB	1	1	1
script1.R	funcionC	script2.R	funcionC	1	1	1
β						
script1.R	funcionA	script2.R	funcionA	1	1	1
script1.R	funcionE	script2.R	funcionE	1	1	1
script1.R	funcionD	script2.R	funcionD	1	1	1
script1.R	funcionB	script2.R	funcionB	1	1	1
script1.R	funcionC	script2.R	funcionC	1	1	1

[a] U = 1
[b] V = 1
[c] $\kappa = 1$

Tabla 4.2: Tabla SimilaR_mod. Ejemplo scripts iguales

Matriz Kappa

Obtenemos la matriz K de la misma forma que se obtuvo la matriz de diferencia de coeficientes φ, 4.1.

$$k = 1 - L = 1_{nxn} - \begin{bmatrix} \kappa_{11} & \kappa_{12} & \cdots & \kappa_{1n} \\ \kappa_{21} & \kappa_{22} & \cdots & \kappa_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \kappa_{n1} & \kappa_{n2} & \cdots & \kappa_{nn} \end{bmatrix} = \begin{bmatrix} l_{11} & l_{12} & \cdots & l_{1n} \\ l_{21} & l_{22} & \cdots & l_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{bmatrix} \tag{4.2}$$

4.3. Medida para cuantificar la similitud de documentos

Para hallar una medida que proporcione cuan símil son dos documentos, hay que tener en cuenta que:

- φ que cuantifica la similitud entre pares de documentos.
- κ mide la proporción de plagio entre los códigos de dos archivos.
- $p \in [0, 1]$ es un parámetro que indica la importancia que tiene los comentarios en el análisis.

A partir de estos puntos, la medida utilizada para cuantificar la similitud de plagio entre dos documentos es:

$$\theta = p \cdot \varphi + p^c \cdot \kappa; \quad \theta \in [0, 1]$$

θ proporciona una medida global de similitud entre dos documentos, ya que tiene en cuenta comentarios y código ejecutable, estableciendo en el parámetro `p` la importancia de estos. Por defecto p = 0.5, por lo que φ y κ tienen la misma importancia.

4.4. Salidas de la función SimilaR2

4.4.1. Salida principal

La salida principal muestra los resultados más relevantes para la detección de plagio entre pares de archivos R o Rmd. Ésta se representa en un data.frame con las siguientes 8 columnas:

- **File1** y **File2** referencian los nombres de los archivos que se comparan.
- **kappa,** (κ). Medida que cuantifica la similitud de los códigos de cada par de archivos.
- **pi_1,** (π_1). 0 o 1. El valor 1 indica que los códigos de ambos archivos son muy similares y 0 que son disímiles. Esta variable se forma con el condicional $\kappa > \alpha_1 = 1;\ 0$ `en caso contrario` (siendo α_1 un parámetro).
- **Varphi,** (φ). Medida que cuantifica la similitud de los códigos de cada par de archivos.
- **pi_2,** (π_2). 0 o 1. El valor 1 indica que los comentarios de ambos archivos son muy similares y 0 que son disímiles. Esta variable se forma con el condicional $\varphi > \alpha_2 = 1;\ 0$ `en caso contrario` (siendo α_2 un parámetro).
- **theta,** (θ). Medida que cuantifica la similitud de cada par de documentos.
- **Pi,** (Π). 0 o 1. El valor 1 indica que los documentos son muy similares y 0 que son disímiles. Esta variable se forma con el condicional $\theta > \alpha_3 = 1;\ 0$ `en caso contrario` (siendo α_3 un parámetro).

  No obstante, es importante resaltar que los valores obtenidos por la herramienta son informativos, ya que hay muchos factores que pueden alterar los resultados,. Por tanto, es fundamental el factor humano para contrastar estos la información.

En la siguiente imagen se puede ver la salida principal obtenida de aplicar la nueva herramienta al directorio prueba 1.

File1	File2	K	π_1	φ	π_2	θ	Π
prueba1.R	prueba2.R	1	1	0.79	1	0.89	1

Figura 4.4: Salida principal. (prueba1)

4.4.2. Otras salidas

Son salidas complementarias para reforzar las conclusiones principales. Ofrecen resultados para el conjunto de datos, **clustering** y también salidas más individualizadas o que sirven para filtrar comparaciones entre dos archivos.

4.4.2.1. Clustering

La herramienta ofrece la posibilidad de mostrar varios resultados de los análisis clústering (tipo jerárquico) realizados a partir de las matrices simétricas (K, Φ). Para la matriz K se obtiene la existencia de grupos de plagio en los códigos del conjuntos de datos analizados; análogamente, para Φ se mide la existencia de grupos de plagio en los comentarios.

Los función ofrece la posibilidad de escoger el tipo de clasificación jerárquica entre aglomeratvia y disocitiva. También el número a formar.

Salidas

- Si el tipo de clasificación escogida es disociativa, se muestra una tabla con dos medidas obtenidas del análisis clústering: coeficiente de división y media de los valores silueta.
- Si el tipo de clasificación seleccionada es aglomerativa se calculan los coeficientes aglomerativos por los métodos media, vecino más cercano, vecino más lejano y Ward. Se muestra una tabla con la siguiente información: El método que ha proporcionado mayor coeficiente aglomerativo, el valor de ese coeficiente y la media de los valores silueta.
- Para cada matriz simétrica (K, Φ) se muestra sus gráficos de dispersión, silueta y dendrograma.
- Tabla que muestra los valores silueta medios para cada clúster.

Propósito:

El análisis clustering efectuado por SimilaR2 a un directorio de archivos R o Rmd (válido desde 2, se recomienda al menos 10), tiene la finalidad de clasificar estos docuemtnos en grupos de `plagio / no plagio`.

Ejemplo 4

Se aplica SimilaR2 con los parámetros para el análisis clúster $n^K = n^\Phi = 3$ (nº clúster) a un directorio con 15 archivos R. Donde se obtienen buenos resultados del análisis (media de valores siluetas y coeficiente aglomerativo o de división) y se obtiene la siguiente configuración de agrupación (1, 1, 1, 1, 1, 2, 1, 1, 1, 2, 3, 1, 1, 3, 1).

Se podría deducir que existen 11 archivos donde no hay copia (1) y otros dos grupos donde hay plagio (2) y (3). No obstante, es importante insistir que aunque las medidas obtenidas sean muy buenas, los resultados son orientativos y deben ser verificados por el factor humano.

4.4.2.2. Frecuencia de palabras

Para cada archivo analizado, se muestra una tabla de dos columnas, la primera contiene las palabras (después del proceso parsing) usadas en los comentarios y la segunda la frecuencia con las que aparecen.

La tabla de **frecuencia de palabras** se utiliza para construir una nube de palabras con librerías como 'wordcloud2'. En la aplicación construida se muestran las nubes de palabras

de los dos archivos que se seleccionen en las opciones de los imputs. Con esto se puede visualizar de forma más vistosa y clara si las palabras de ambos archivos son iguales.

4.4.2.3. Salida Tabla SimilaR_mod

La tabla SimilaR_mod 4.2.2 ofrece al usuario la posibilidad de observar las funciones de cada par de archivos que han obtenido los mayores coeficientes SimilaR.

Capítulo 5

SimilaR2. Package R

Se documenta el proceso de construcción de la librería **SimilaR2**, a su vez, se analizan las prestaciones y uso de la aplicación Sihiny, la cual es capaz de ejecutar SimilaR2 y mostrar los resultados de forma interactiva.

5.1. Paquete SimilaR2

Crear un proyecto para construir un paquete R y ensamblarlo para producir la librería (de forma local) es muy sencillo. La dificultad recae en programar las funciones del paquete. Debido a esto, es recomendable construir todas las funciones del paquete y comprobar sus funcionalidades antes de crear el proyecto.
En este apartado se documenta el proceso de elaboración del paquete R, pero no de las funciones. Estas se encuentran documentadas en el Apéndice A.2.

El paquete R **SimilaR2** se ha construido de la siguiente forma:

1. Abrir RStudio/ crear nuevo proyecto/ New Directory / R package. En este paso se introduce el nombre del paquete (SimilaR2), la ruta de instalación y se marcan las casillas `crear repositorio git` y usar `renv`. Finalmente, se pulsa **Create Project**.
2. Se abre el proyecto y se modifica el archivo **DISCRIPTION**, donde se editan los parámetros Title, Autor, Description, etcétera. Si el paquete depende de otras librerías es importante añadir `Imports: 'nombre librería'`.

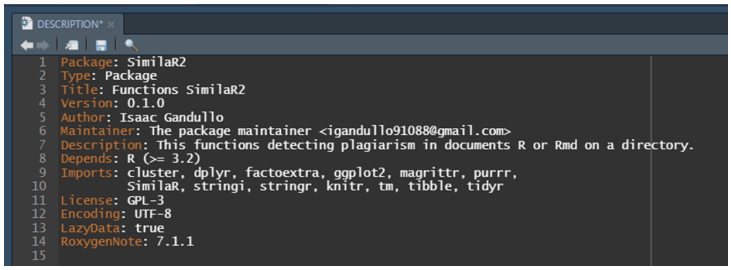


Figura 5.1: Documento DESCRIPTION del paquete SimilaR2

3. En la carpeta **R** del proyecto se copian todas las funciones del paquete (internas y externas).
   Las funciones de la carpeta **R** que quedan disponibles para el usuario son:

- **plagiarism_code**. Dado un directorio de archivos R o Rmd cuantifica la similitud de los códigos de cada par de archivos.
- **plagiarism_comments**. Dado un directorio de archivos R o Rmd cuantifica la similitud de los comentarios de cada par de archivos.
- **SimilaR2**. Dado un directorio de archivos R o Rmd cuantifica la similitud de cada par de archivos.
- **clustering**. Dada una matriz de distanccia cuadrada realiza un análisis clústering de tipo jerárquico.

La información de las citadas funciones (descripción, uso, argumentos, detalles, evaluación y ejemplos) se encuentra en el Apéndice A.2.

4. Para **documentar** las **funciones exportadas** se ha seguido la siguiente estructura:
   - **@title** título.
   - **@description**: descripción.
   - **@param** : nombre del parámetro y su funcionalidad.
   - **@example**: ejemplos.
   - Antes de empezar una función hay que añadir **@export 'nombre función'**, esto sirve para indicar a la librería **roxygen2** que es una función utilizable del paquete.

   Estos apartados para documentar funciones son los más generalizados. Se pueden utilizar otros apartados más específicos, tales como los detallados en este artículo sobre *roxygen2*.

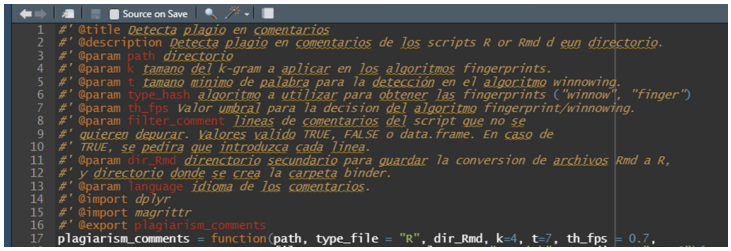


Figura 5.2: Proceso de documentación de funciones

5. **Instalación del paquete SimilaR2**. Primero se instalan todos los paquetes de dependencias de SimilaR2 (cluster, dplyr, factorextra, ...), estos se instalan en la carpeta renv del proyecto. Por último, se instalan las librerías **devtools** y **roxygen2**.
6. Se ejecutan los siguientes comandos:
   - **devtools::document()**. Genera la documentación del paquete.
   - **devtools::build()**. Genera un archivo `.tar.gz`, con el cual se puede instalar el paquete con la instrucción `"install.packages('ruta/nombrearchivo.tar.gz', repo = NULL, type = 'source')"`.
   - **devtools::install()**. Se instala el paquete, con esta instrucción se obvia el paso anterior. No obstante, este comando no siempre funciona y de a de optar por otra opción.

7. El proyecto se sube al repositorio SimilaR2 package. Con este paso se puede instalar SimilaR2 mediante el comando `devtools::install_gitlab('igandullo88/similar2_package')`. Si no puede instalar SimilaR2 mediante esa instrucción, puede descargar el archivo .tar.gz en este enlace e instalarlo con la instrucción `install.packages('ruta/nombrearchivo.tar.gz', repo = NULL, type = 'source')`.

Para finalizar, se informa que la herramienta sigue en construcción y mantenimiento; con la finalidad de conseguir la instalación de SimilaR2 en el repositorio oficial de R CRAN, así como mejorar sus prestaciones y rendimiento.

5.2. Aplicación Shiny

La aplicación shiny es capaz de ejecutar SimilaR2 mediante una interfaz con opciones interactivas, así como de mostrar en otra web los resultados obtenidos (con opciones interactivas) de forma vistosa, clara y detallada.

La aplicación se puede descargar en SimilaR2 Shiny app. Requiere tener instalados los paquetes de R markdown, flexdashboard, ggplot2, kableExtra, shiny, shinyjs y SimilaR2.

5.2.1. Problemas y uso de la app

Durante la construcción de la aplicación se ha encontrado un problema que el autor de este TFG desconocía, siendo este la imposibildad de cargar una aplicación Shiny dentro de otra aplicación del mismo tipo. El framework shiny server es capaz de solventar este problema, pero solo funciona en Linux. Dado que este trabajo se ha realizado en arquitectura Windows, no ha podido ser utilizado.
Debido a este incoveniente se debe ejecutar la aplicación de la siguiente forma:

1. Se ejecuta el archivo **Interfaz.R**. En esta aplicación el usuario introduce los parámetros de la función SimilaR2, y la ejecuta a través de un evento de botón. Al terminar de compilarse la función, se genera un archivo **.RData** para ser usado por la segunda aplicación, a su vez, aparece un mensaje informando al usuario que se han generado los datos, que proceda a cargar la app Resultados.
2. Se ejecuta el archivo **Resultados.Rmd**. Se muestran los resultados principales y desagregados generados por SimilaR2.

5.2.2. Interfaz y prestaciones.

App interfaz

Al ejecutarse **Interfaz.R** se puede vez la siguiente interfaz:

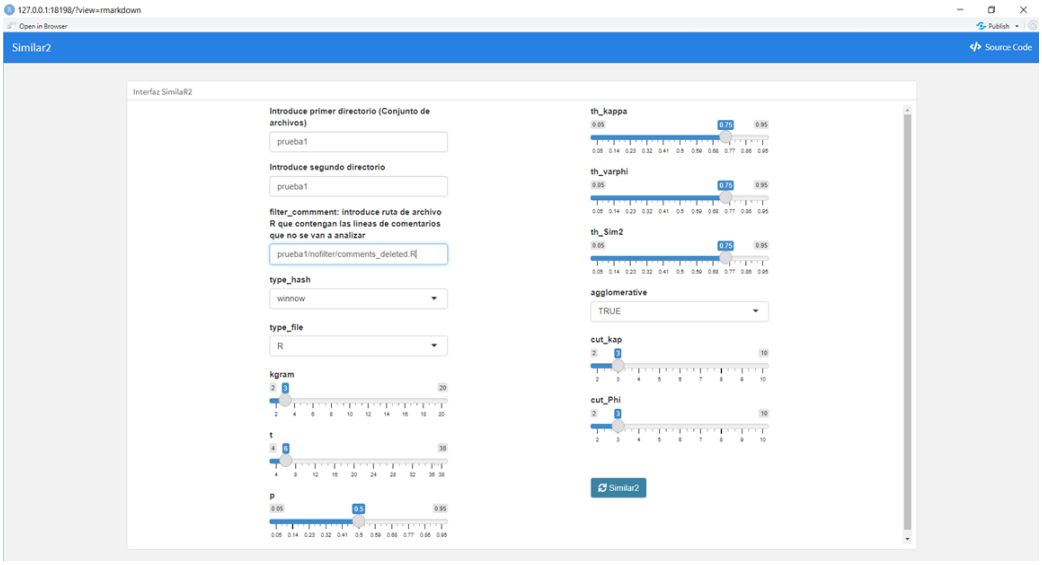


Figura 5.3: Interfaz SimilaR2

El usuario dispone de opciones modificables para todos los parámetros de SimilaR2. Pero se debe tener en cuenta lo siguiente:

- Los parámetros **path** y **dir_Rmd** son las rutas de los directorios principal y secundario. Estas rutas han de introducirse desde la raíz de la aplicación.
  Ejemplo: Si la app está en `"C:/App"` y el directorio de archivos está en `"C:/Archivos/"`; el parámetro path = `"../Archivos"`.
  Consejo: Para evitar problemas con las rutas, descargue la carpeta APP, en la carpeta prueba1 borre los archivos .R y pegue los archivos que quiera analizar. Cargue la app Interfaz y pulse el botón SimilaR2.
- Para el parámetro **filter_comment** se ha habilitado la opción de introducir la ruta de un archivo R. Este archivo debe contener cada línea de comentarios que no se quiera analizar. La aplicación lee cada línea del documento, lo transforma a data.frame y lo pasa al parámetro filter_comment. La opción por defecto de filter_comment en la app es **FALSE**.

El resto de los parámetros vienen con opciones por defectos, tales como: **type_file: R**, **kgram: 3**, **p: 0.5**, entre otros.
Cuando se hallan introducidos los parámetros se debe pulsar el botón **SimilaR2**. Acto seguido, aparece una alerta informando del procesamiento del análisis (5.4 captura izquierda), al terminar el análisis, debajo del botón SimilaR2, aparece el mensaje: *Datos generados. Proceda a ejecutar la app Resultados* (5.4 captura derecha).

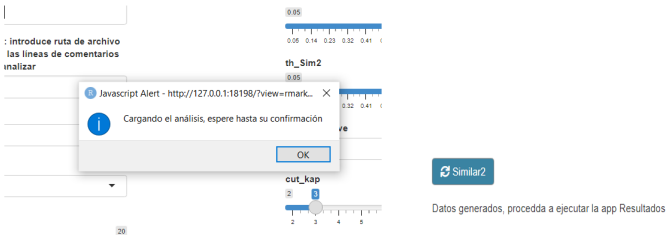


Figura 5.4: Interfaz SimilaR2. Mensajes

App Resultados

En el apartado 5.5.1 se argumento que para un uso completo de la app se deben ejecutar las dos aplicaciones. No obstante, se puede realizar una modificación en el archivo Resultados.Rmd y no sería necesario ejecutar Interfaz.R. A continuación se explican los dos procedimientos para cargar la segunda aplicación:

1. **Solo app Resultados**. Abrimos la app Resultados.Rmd y descomentamos las líneas 27-31, y comentamos la 32. E introducimos los parámetros SimilaR2 de forma manual. Una vez introducidos ejecutamos la aplicación.

```
library(SimilaR2)
plagio = SimilaR2(path =  "prueba1", dir_Rmd = "prueba1",
                  type_hash = "winnow", k = 4, t = 6,
                  type_file = "R", filter_comment = FALSE,
                  cut_Kap = 3, cut_Phi = 3, cluster = TRUE,
                  breakdown_comments = TRUE,
                  breakdown_code = TRUE, View_wr_scp = FALSE,
                  th_kap = 0.75, th_fps = 0.75)
#load("save_data/plagio.RData")
```

2. **Ambas aplicaciones**. En el caso de obtener el archivo `plagio.RData` de la app Interfaz.R se procede a ejecutar Resultados, en caso contrario recurrir al punto anterior.

Compilándose de una forma u otra se obtiene la siguiente web:

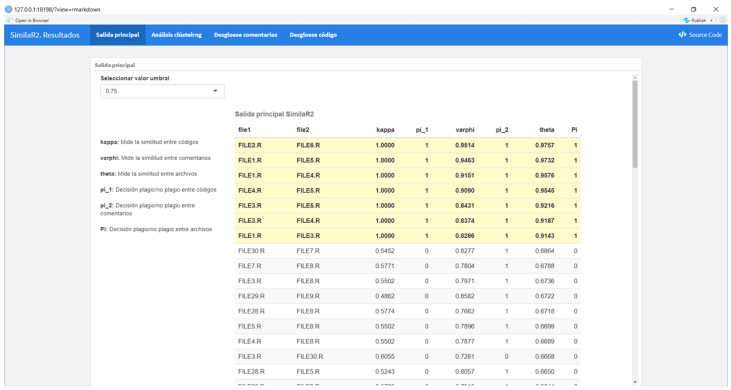


Figura 5.5: Interfaz SimilaR2. Salida principal

En la imagen observamos que la web se divide en 4 tabset (pestañas): **Salida principal**, **Análisis clustéring**, **Desglose comentarios** y **Desglose código**:

1. **Salida principal**. Muestra la salida principal de SimilaR2, a la izquierda de la tabla hay una breve descripción de las variables.
   Las variables dicotómicas de decisión (pi_1, pi_2, Pi) están creadas según los valores umbrales que se introdujeron en la app Interfaz. Si se quiere comprobar los resultados de estas variables con otro valor umbral, basta con seleccionar otro valor umbral en el input tipo select situado en la parte superior izquierda de esta página. También, se han resaltado, mediante negrita y background (`"#FFFCCC"`), las filas cuyo valor de theta es mayor al valor umbral. (Figura 5.5).

2. **Análisis clústering**. Muestra el análisis clústering de las matrices Φ y K. Donde se puede ver las tablas de coeficientes y silueta, así como, un cuadro con diferentes tabset para los gráficos de dispersión, silueta y dendrograma de ambas matrices.

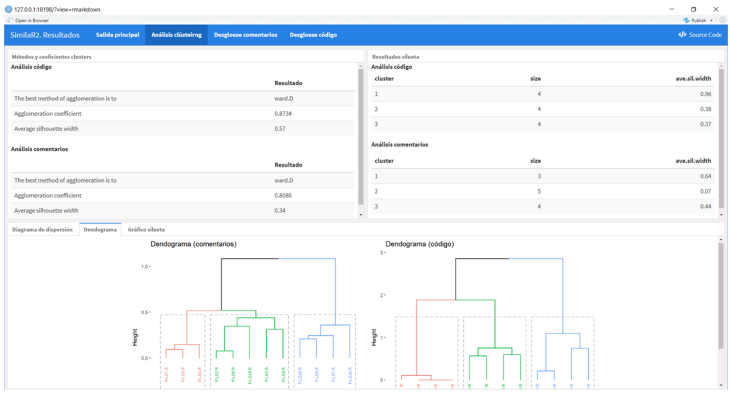


Figura 5.6: Interfaz SimilaR2. Análisis clústering

3. **Desglose comentarios**. En el recuadro inferior se muestra la tabla Sorensen ordenada por filas de mayor a menor por φ_{win}, con la finalidad de que el usuario observe los pares de archivos que han sido plagiados en comentarios. El proceso de resaltado de filas es igual al de la tabla salida principal. Una vez que el cliente sabe que pares de archivos es interesante desglosar, los puede seleccionar en los inputs tipo select situados en el sidebar, para así mostrar las nubes palabras de cada archivo y comprobar si las palabras más utilizada en ambos scripts son equitativas.

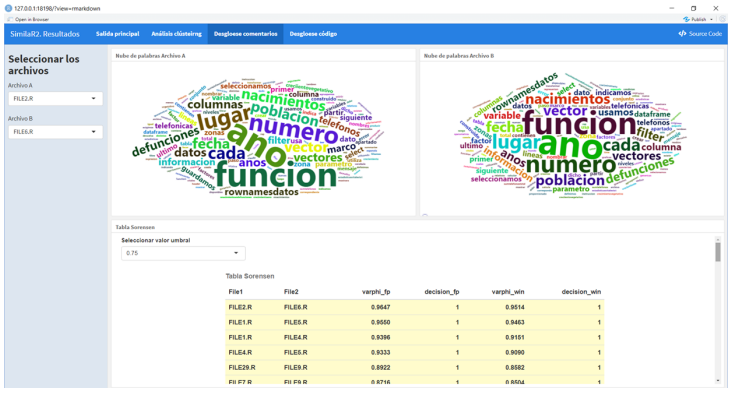


Figura 5.7: Interfaz SimilaR2. Desglose comentarios

4. **Desglose código**. El contenido es similar al anterior pero con la tabla kappa. En este caso, al seleccionar un par de archivos se muestra la tabla SimilaR_mod para ese par de archivos. Con ello se comprueba como de parecidas son las funciones de cada par de archivos.

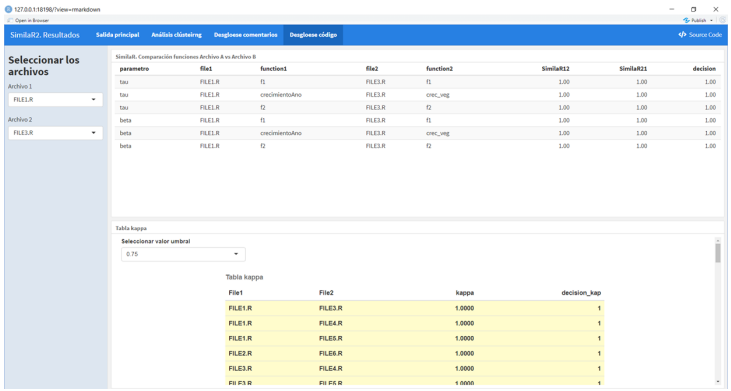


Figura 5.8: Interfaz SimilaR2. Desglose código

5.2.3. Mejoras propuestas.

La aplicación requiere de una serie de mejora como por ejemplo:

- Conseguir unificar ambas aplicaciones en un entorno por ejemplo shiny / javascript.
- Subir la aplicación a Shiny web con uso público, para que cualquier usuario pueda utilizarla.
- Optimización. Depuración de código, para un mayor rendimiento de la aplicación.

Se trabajará en estas mejoras, y posteriormente en su mantenimiento.

Capítulo 6

Análisis de resultados

Para el desarrollo del análisis estadístico se ha efectuado el siguiente procedimiento:

- **Obtención del dataset**. Conjunto de archivos R o Rmd para efectuar las pruebas de la herramienta.
- **Marco de datos generalizado**. Se obtienen las variables que componen el conjunto de datos con el que se efectúan las diferentes pruebas.
- **Elección del algoritmo fingerpritnts**. Para la detección de plagio en comentarios, se analiza que algoritmo fingerprints es más adecuado.
- **Fiabilidad de SimilaR2**. Para la detección de plagio en archivos, y con una importancia equitativa entre los comentarios y el código, p = 0.5, se analiza la fiabilidad de la herramienta.
- **Análisis clúster**. Para la detección de plagio en archivos, y usando la matriz de distancia Θ, se efectúa un análisis clustering para comprobar la existencia de grupos de archivos (plagiados / no plagiados).

No obstante, hay que insistir que los resultados no tienen una validez irrefutable y se debe contrastar la información manualmente.

6.1. Dataset

La recopilación de una batería de archivos R o Rmd para realizar un análisis estadístico de SimilaR2 ha sido una tarea ardua, destacando los siguientes inconvenientes:

1. Las premisas de un buen análisis estadístico es obtener una muestra de al menos tamaño 30 (archivos R o Rmd).
2. Cada archivo debe tener la solución en código R e interpretada con palabras (comentarios).
3. Cada script debe ser programado por una única persona, con esto se evita sesgo en la sintaxis.
4. Cada archivo debe catalogarse como original o plagio y debe existir entre un 25-40 % de plagios.
   La persona encargada de plagiar el documento original no puede ser la misma que haya programado el script original.

Obtención de los archivos originales

Para obtener la participación del máximo número de personas en la construcción del dataset, se partió de unas tareas de introducción a R que se realizaron en la asignatura Inteligencia Artificial Estadística en el curso 2019/2020. La unión de las tareas contiene un total de 9 ejercicios, que cualquier alumno de cuarto curso del Grado en Estadística debe poder cumplimentar en pocos minutos. El contenido de estas tareas puede verse en el anexo A.1.

En un primer momento, antes de encontrar los inconvenientes mencionados al principio del capítulo, se intentó construir el dataset con la ayuda de tres compañeros del grado en estadística. Para ello, se cumplimentaron 9 archivos y de estos se plagiaron 3, consiguiendo un total de 6 archivos originales y 6 plagiados. Esto presentaba problemas de insuficiencia de archivos, así como posible sesgo debido a una sintaxis similar.

Para poder obtener un dataset de archivos originales/plagiados, el cual no presentara estos incovenientes se decide que cada archivo debe cumplimentarse por una única persona. Para intentar lograr este objetivo se pidió ayuda a los compañeros del Grado en Estadística, obteniéndose una negativa mayoritaria se tuvo que buscar otra alternativa.

Se efectúa un ruego al profesor de la asignatura IAE del curso 2019/2020, D. Luis Valencia Cabrera, donde se le solicita el envío de las tareas de introducción a R entregadas por los alumnos de ese curso. Después de consultar a la Delegada de Protección de Datos de la US, se concluyó que los archivos se podrían utilizar en el estudio siempre y cuando estuviesen anonimizados y se hubiesen eliminado referencias personales.

Gracias al profesor de IAE, se obtuvo 44 scripts de R (22 tarea repaso1.1 y 22 de la tarea repaso1.2) completamente anonimizados. Los nombres de los scripts recibidos presentaban el subfijo i para catalogar las coincidencias de las tareas, tarea repaso1.1_i.R, repaso1.2_i.R; i = 1, $\cdots$, 22. Para obtener los scripts originales definitivos del dataset, se programó en R un código de plegado de texto, donde se unieron las tareas repaso1.1_i.R - repaso1.2_i.R sii i = i. Obteniéndose 22 archivos originales del dataset.

Proceso de plagio

Se parte de 22 archivos R (dataset) que se tratan como originales, se plagian 4 de los archivos originales, cada uno por una persona distinta, consiguiendo un total de 26 archivos (18 originales y 8 plagiados). Para conseguir 30 archivos, se construyeron 4 scripts plagiados entre sí, cada copia construida por una persona. Con un resultado final, de 18 archivos originales y 12 archivos considerados como plagios.
Nota: *El archivo original plagiado pasa a considerarse como plagio. Por tanto, por cada archivo original plagiado se obtienen dos archivos plagiados.*

6.2. Marco de datos generalizado

La construcción del marco de datos que se explica a continuación se ha realizado con unos parámetros fijos en **k** y **w**, los cuales se han elegido a partir del análisis de elección del algoritmo fingerprints 6.4.

El conjunto de datos consta de una variable respuesta y de tres variables predictoras (κ^*, φ^* y θ^*):

Variable respuesta

La variable respuesta se introduce manualmente a partir del proceso de plagio definido en el apartado construcción del dataset. Se denota como Y, $Y = y_1, y_2, \cdots, y_{30}$.
Donde $Y = 1$ o 0.

$$Y_i = \begin{cases} 1 \texttt{ el documento i-ésimo ha sido plagiado por algún documento} \\ 0 \texttt{ el documento i-ésimo no ha sido plagiado} \end{cases}$$

FILE1.R	FILE10.R	FILE11.R	FILE12.R	FILE13.R	FILE14.R	FILE15.R	FILE16.R	FILE17.R	FILE18.R
1	0	1	0	0	0	1	0	1	0
FILE19.R	**FILE2.R**	**FILE20.R**	**FILE21.R**	**FILE22.R**	**FILE23.R**	**FILE24.R**	**FILE25.R**	**FILE26.R**	**FILE27.R**
0	1	0	1	0	0	0	1	0	0
FILE28.R	**FILE29.R**	**FILE3.R**	**FILE30.R**	**FILE4.R**	**FILE5.R**	**FILE6.R**	**FILE7.R**	**FILE8.R**	**FILE9.R**
0	0	1	0	1	1	1	0	1	0

Tabla 6.1: Variable respuesta

Variables predictoras

Las variables predictoras se obtienen tras realizar unas pequeñas modificaciones de la salida principal de SimilaR2.
En resumen, se obtienen los coeficientes κ, φ y θ máximos de cada archivo del dataset, es decir, fijado un archivo se comparan con el resto de archivos obteniéndose tres vectores (κ, φ y θ) de dimensión 29 y se hallan sus máximos.

Ejemplo

FILE1.R se comparan con el resto de archivos FILE2.R, ..., FILE30.R.

kappa* = máx$(\kappa_{1,2}, \cdots, \kappa_{1,30})$.

varphi* = máx$(\varphi_{1,2}, \cdots, \varphi_{1,30})$.

theta* = máx$(\theta_{1,2}, \cdots, \theta_{1,30})$.

```
          kappa*   varphi*  theta*
FILE1.R   0.5462   0.4120   0.4671
```

Obtención de las variables predictoras en R

Se aplica la herramienta SimilaR2 sobre el directorio analysis_SimilaR2 con los siguientes parámetros:

- filter_comment = remove_comment. Donde `remove_comment` es un data.frame, resultado de leer cada línea del script R con el contenido del anexo I (A.1) y convertir esas líneas de texto en data.frame.

```
remove_comment = readLines("TEST/comments_deleted.R") %>%
                 dplyr::as_data_frame()
```

- `k = 3` y `t = 6`, obteniendo `w = t - k + 1 = 4`. Parámetros fijos obtenidos de la elección del algoritmo fingerprints.

Se guardan los resultados en la variable `res_SimilaR2`

```
Res_SimilaR2 = SimilaR2("dataset", filter_comment = remove_comment)
```

De la lista `Res_SimilaR2` se utiliza solo el marco de datos `final_score` (salida principal). A continuación se muestra un head de este data.frame.

file1	**file2**	κ	π_1	φ	π_2	θ	Π
FILE2.R	FILE6.R	1	1	0.9203	1	0.96015	1
FILE1.R	FILE5.R	1	1	0.8936	1	0.94680	1
FILE1.R	FILE4.R	1	1	0.8471	1	0.92355	1
FILE4.R	FILE5.R	1	1	0.8449	1	0.92245	1
FILE3.R	FILE5.R	1	1	0.6651	0	0.83255	1
FILE3.R	FILE4.R	1	1	0.6573	0	0.82865	1

Tabla 6.2: Salida principal de SimilaR2 sobre dataset

A partir del marco de datos `final_score` se construyen las matrices M1, M2 y M3, donde se guardan los coeficientes $\kappa_{i,j}, \varphi_{i,j}$ y $\theta_{i,j}$ respectivamente, siendo i,j la comparación del i-ésimo archivo A con el j-ésimo B (i,j = 1, $\cdots$, 30). Las dimensiones de cada matriz son (30, 30).

Mediante las matrices M1, M2 y M3 obtenemos las variables $\kappa^*, \varphi^*, \theta^*$ de la siguiente forma:

$$\begin{aligned} \kappa^* &= max(A); A = M1_i \Leftrightarrow i \neq j \\ \varphi^* &= max(B); B = M2_i \Leftrightarrow i \neq j \\ \theta^* &= max(C); C = M3_i \Leftrightarrow i \neq j \end{aligned} \tag{6.1}$$

Se calculan los máximos por filas de cada matriz y se obtienen las variables predictoras.

	κ^*	φ^*	θ^*		κ^*	φ^*	θ^*
FILE2.R	1.0000	0.9203	0.9602	**FILE8.R**	0.8225	0.7741	0.6730
FILE6.R	1.0000	0.9203	0.9602	**FILE19.R**	0.6999	0.6517	0.6613
FILE1.R	1.0000	0.8936	0.9468	**FILE18.R**	0.6551	0.6517	0.6441
FILE5.R	1.0000	0.8936	0.9468	**FILE16.R**	0.6641	0.6599	0.6395
FILE4.R	1.0000	0.8471	0.9236	**FILE28.R**	0.6325	0.6769	0.6335
FILE3.R	1.0000	0.6651	0.8326	**FILE23.R**	0.8225	0.5280	0.6254
FILE17.R	0.7163	0.8022	0.7491	**FILE15.R**	0.6641	0.7091	0.6249
FILE21.R	0.7149	0.8022	0.7491	**FILE24.R**	0.5721	0.7222	0.6219
FILE11.R	0.7163	0.7468	0.7316	**FILE12.R**	0.5789	0.6409	0.5954
FILE13.R	0.7789	0.6730	0.7259	**FILE20.R**	0.5584	0.6003	0.5685
FILE26.R	0.7789	0.6730	0.7259	**FILE27.R**	0.6115	0.5769	0.5651
FILE14.R	0.7063	0.7143	0.7103	**FILE29.R**	0.5769	0.5856	0.5433
FILE22.R	0.7149	0.6707	0.6928	**FILE7.R**	0.6023	0.5613	0.5399
FILE10.R	0.6982	0.7232	0.6864	**FILE9.R**	0.4862	0.6369	0.5359
FILE25.R	0.6257	0.7741	0.6730	**FILE30.R**	0.6143	0.4865	0.4863

Tabla 6.3: Variables predictoras

6.3. Modelización y evaluación

Para conseguir evaluar la variable respuesta, se tomó la decisión de modelizar cada conjunto de datos mediante el modelo de Naive Bayes con corrección Laplace y mediante validación cruzada. Después de la modelización, se efectúan las predicciones y se evalúa el modelo.

Modelo Naive Bayes

Naive bayes es un modelo de aprendizaje automático que utiliza el principio de probabilidad bayesiana:

$$P(A_i|B) = \frac{P(B|A_i) \cdot P(A_i)}{P(B)}$$

donde

- $P(A_i)$ son las probabilidades a priori.
- $P(B|A_i)$ es la probabilidad de B en la hipótesis A_i.
- $P(A_i|B)$ son las probabilidades a posteriori.

Para validar este modelo se utiliza la librería de R **tidymodels** aplicando los siguientes pasos:

- Se divide el conjunto de datos en entrenamiento y prueba.
- Sobre el conjunto de entrenamiento se aplica una receta:
  - Se define las variables predictoras y respuesta.
- Definimos el modelo:
  - Se aplica el modelo **naive bayes**, se fija el parámetro **Laplace = 1** y el parámetro **smoothness** (parámetro que controla la suavidad relativa del límite de clase) se define como **tune()** (con esta instrucción al realizar validación cruzada se obtendrá el valor de smoothness que mejor accuracy proporcione).
- Se crea el workflow de la receta y el modelo.
- Se crea `fold` que contienen las instrucciones para la aplicación de validación cruzada, 8 particiones, y 10 repeticiones, (n = 800).
- Se guardan los resultados de aplicar validación cruzada sobre el workflow. Se obtiene una lista de tibbles, cada uno guarda los resultados de una iteración del proceso.
- Se halla el workflow cuya evaluación ha proporciona un mayor accuracy. Finalmente, se calculan las métricas de este (accuracy, sensibilidad, sensitividad, roc_auc, acierto).

Evaluación

La evaluación del modelo se realiza mediante la matriz de confusión. Esta es una tabla que muestra cuatro categorías diferentes: Verdadero Positivo, Verdadero Negativo, Falso Positivo y Falso Negativo.

- Verdadero Positivo: Documento plagiado catalogado por el modelo como plagio.

- Verdadero Negativo: Documento no plagiado catalogado por el modelo como no plagio.
- Falso Positivo: Documento no plagiado catalogado por el modelo como plagio.
- Falso Negativo: Documento plagiado catalogado por el modelo como no plagio.

		Prediccion	
		plagio	**no plagio**
Real	**plagio**	VP	FN
	no plagio	FP	VN

El rendimiento del modelo se mide mediante Acurracy (Exactitud), Sensibilidad, Especificidad, Precisión y Área bajo la curva Roc.

- **Accuracy** mide cuántos de nuestros datos se predice correctamente.
- **Sensibilidad** mide de todos los resultados positivos, cuántos se predicen correctamente.
- **Especificidad** mide cuántos resultados negativos se predicen correctamente.
- **Precisión** mide cuántas de nuestras predicciones positivas son correctas.
- Área bajo la curca ROC, **roc_auc**, el AUC proporciona una medición agregada del rendimiento en todos los umbrales de clasificación posibles. Una forma de interpretar el AUC es como la probabilidad de que el modelo clasifique un ejemplo positivo aleatorio más alto que un ejemplo negativo aleatorio.

$$
\begin{aligned}
Acurracy &= \frac{VP + VN}{VP + VN + FP + FN} \\
Sensitividad &= \frac{VP}{VP + FN} \\
Especificidad &= \frac{VN}{VN + FP} \\
Precision &= \frac{VP}{VP + FP}
\end{aligned}
\tag{6.2}
$$

6.4. Elección de algoritmo fingerprints

Para comprobar qué algoritmo fingerprints y bajo qué parámetros k y w se obtienen mejores resultados, se han construido los siguientes marcos de datos:

- **Fingerprint**. Se obtiene un marco de datos para cada **kgram** 2, 3, 4 y 5.

	φ_{fp}	respuesta		φ_{fp}	respuesta
FILE1.R	0.9259	1	**FILE23.R**	0.6329	0
FILE10.R	0.7767	0	**FILE24.R**	0.7611	0
FILE11.R	0.8000	1	**FILE25.R**	0.8184	1
FILE12.R	0.7352	0	**FILE26.R**	0.7584	0
FILE13.R	0.7642	0	**FILE27.R**	0.5907	0
FILE14.R	0.7880	0	**FILE28.R**	0.7672	0
FILE15.R	0.7984	1	**FILE29.R**	0.6889	0
FILE16.R	0.7551	0	**FILE3.R**	0.7921	1
FILE17.R	0.8744	1	**FILE30.R**	0.6243	0
FILE18.R	0.7484	0	**FILE4.R**	0.9010	1
FILE19.R	0.7339	0	**FILE5.R**	0.9259	1
FILE2.R	0.9401	1	**FILE6.R**	0.9401	1
FILE20.R	0.6850	0	**FILE7.R**	0.6161	0
FILE21.R	0.8744	1	**FILE8.R**	0.8184	1
FILE22.R	0.7484	0	**FILE9.R**	0.6889	0

Tabla 6.4: Marco de datos fingerprint (kgram 3)

- **Winnowing**. Se obtiene un marco de datos para cada combinación de **k** = {2, 3, 4 y 5} y **w** = {4, 8, 10, 12}. Un total de 20 conjuntos.

	φ_{win}	respuesta		φ_{win}	respuesta
FILE1.R	0.9102	1	**FILE23.R**	0.5635	0
FILE10.R	0.7105	0	**FILE24.R**	0.6946	0
FILE11.R	0.7861	1	**FILE25.R**	0.7555	1
FILE12.R	0.6513	0	**FILE26.R**	0.6515	0
FILE13.R	0.6805	0	**FILE27.R**	0.5734	0
FILE14.R	0.6756	0	**FILE28.R**	0.7049	0
FILE15.R	0.7674	1	**FILE29.R**	0.6469	0
FILE16.R	0.6544	0	**FILE3.R**	0.7145	1
FILE17.R	0.8357	1	**FILE30.R**	0.4916	0
FILE18.R	0.6583	0	**FILE4.R**	0.8597	1
FILE19.R	0.6683	0	**FILE5.R**	0.9102	1
FILE2.R	0.9213	1	**FILE6.R**	0.9213	1
FILE20.R	0.6099	0	**FILE7.R**	0.5863	0
FILE21.R	0.8357	1	**FILE8.R**	0.7555	1
FILE22.R	0.6427	0	**FILE9.R**	0.6555	0

Tabla 6.5: Marco de datos winnowing (kgram 3, w = 4)

Siguiendo los criterios de modelización y evaluación expuestos en el apartado anterior, se modeliza y evalúa cada marco de datos. Obteniendo los siguientes resultados:

Fingerprint

	kgram	accuracy	roc_auc	sens	spec
k2	2	**0.845**	**0.959**	**0.85**	**0.84**
k3	3	**0.866**	**0.999**	**0.998**	**0.657**
k4	4	**0.865**	**0.924**	**0.992**	**0.664**
k5	5	**0.819**	**0.977**	**0.902**	**0.685**

Tabla 6.6: Métricas fingerprint (según kgram)

Según los resultados obtenidos se puede decir que el test realizado con el marco de datos 3 kgram predice correctamente 86.6 %, de los documentos considerados plagios ha clasificado correctamente el 99.8 %, pero de los no plagios solo el 65.7 %, aunque el ajuste del test es excelente, AUC = 0.99, es considerable tener en cuenta los resultados obtenidos con el marco de datos 2 kgram, ya que el acierto y AUC es similar, si bien presenta una sensibilidad menor, su especificidad es bastante mayor, es decir, el porcentaje de no plagio catalogados correctamente es un 18.7 % mejor.

Para este tipo de análisis es importante tener un equilibrio entre la sensibilidad y la especificidad, ya que es grave tanto detectar un no plagio como plagio, pero es peor detectar un no plagio como plagio. Por tanto, se concluye que los mejores resultados para el algoritmo finngerprint se consiguen con el marco de datos **2 kgram**.

Winnowing

kgram	window size	accuracy	roc_auc	sens	spec
2	4	0.938	0.95	0.979	0.873
	6	0.872	0.96	0.994	0.679
	8	0.954	1	1	0.88
	10	0.934	0.955	0.999	0.83
	12	0.912	0.991	0.928	0.886
3	4	0.957	1	1	0.888
	6	0.943	1	1	0.852
	8	0.914	0.956	1	0.777
	10	0.957	1	1	0.888
	12	0.911	0.979	0.991	0.783
4	4	0.875	0.917	0.989	0.692
	6	0.856	0.918	0.983	0.655
	8	0.813	0.886	0.97	0.563
	10	0.819	0.913	0.976	0.571
	12	0.875	0.95	0.979	0.709
5	4	0.831	0.947	0.943	0.651
	6	0.885	0.967	0.915	0.838
	8	0.915	0.985	0.99	0.795
	10	0.857	0.977	0.99	0.646
	12	0.851	0.957	0.986	0.635

Tabla 6.7: Métricas winnowing (según kgram y tamaño de ventana)

Los resultados obtenidos con los marcos de datos del algoritmo winnowing son más destacables que los de fingerprint. No obstante, ambos proporcionan los mejores resultados

con los kgrams 2 y 3. En el caso de winnowing los resultados son más concluyentes, pues las mejores métricas se logran con 3 kgram y con un desplazamiento de ventana 4 o 10. Prediciendo correctamente el 95.9 %, con un AUC y sensibilidad perfectos, 1, y catalogando correctamente el 88.8 % de los no plagio. Se concluye que el algoritmo winnowing con los parámetros k = 3 y w = 4 o w = 10 proporciona los mejores resultados.

Para concluir, se ha calculado para el algoritmo winnowing las medias de las métricas (accuracy , roc_auc) según kgram y se ha comparado con las de fingerprint.

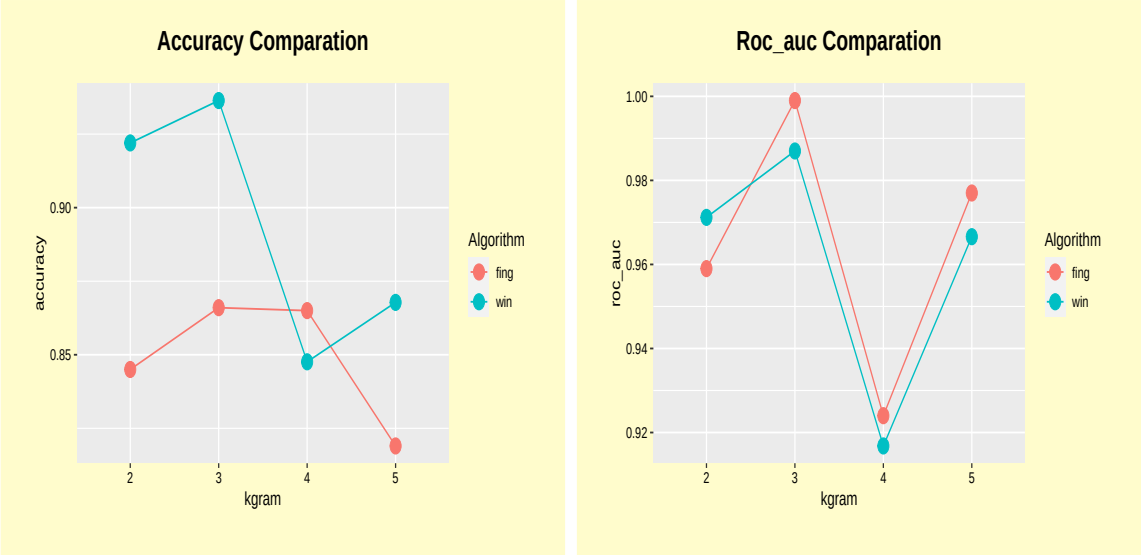


Figura 6.1: Fingerprint and winnowing Comparation

En los gráficos no aportan nuevas conclusiones. Pero sirven para acentuar la importancia del algoritmo winnowing y sobre todo de su parámetro `w`. Puesto que al haber calculado las medias de las métricas se obtienen peores resultados e incluso ligeramente inferiores a los de fingerprint.

En la elección de algoritmo fingerprints se concluye que el algoritmo winnowing con los parámetros **kgram = 3** y un **tamaño de ventana** de 4 o 10 proporciona el mejor ajuste.

window size	accuracy	roc_auc	sens	spec
4	0.9568	1.0000	1.0000	0.888
6	0.9425	1.0000	1.0000	0.852
8	0.9140	0.9557	1.0000	0.777
10	0.9568	1.0000	1.0000	0.888
12	0.9105	0.9787	0.9907	0.783

Tabla 6.8: Metricas winnowing (3 k-gram)

6.5. Fiabilidad de la función SimilaR2

Para comprobar la fiabilidad en la detección de plagio en archivos, es necesario comprobar bajo que transformación dicotómica de las variables predictoras se obtienen mejores

resultados, es decir, partiendo del marco de datos generalizado (variable respuesta y predictoras), 6.2, se transforman las variables en eventos (plagio o no plagio), según el condicional: si el valor de un coeficiente es mayor que un cierto valor umbral se etiqueta como **plagio**, si es menor que el valor umbral se clasifica como **no plagio**. Por ejemplo,

Fijado umbral 0.7

$\theta^*_{1,6} = 0.79 \Rightarrow \text{des\_the}_{1,6} = \text{plagio}$

$\varphi^*_{2,7} = 0.43 \Rightarrow \text{des\_var}_{2,7} = \text{no plagio}$

Los valores umbral utilizados para entrenar el modelo son 0.65, 0.70, 0.75 y 0.80. Para cada valor umbral se obtiene un marco de datos y se entrena el modelo.

En la siguiente tabla se muestra el marco de datos obtenido para el valor umbral 0.7.

	des_kap	**des_var**	**des_the**	**respuesta**		**des_kap**	**des_var**	**des_the**	**respuesta**
FILE2.R	plagio	plagio	plagio	plagio	**FILE8.R**	plagio	plagio	noplagio	plagio
FILE6.R	plagio	plagio	plagio	plagio	**FILE19.R**	noplagio	noplagio	noplagio	noplagio
FILE1.R	plagio	plagio	plagio	plagio	**FILE18.R**	noplagio	noplagio	noplagio	noplagio
FILE5.R	plagio	plagio	plagio	plagio	**FILE16.R**	noplagio	noplagio	noplagio	noplagio
FILE4.R	plagio	plagio	plagio	plagio	**FILE28.R**	noplagio	noplagio	noplagio	noplagio
FILE3.R	plagio	noplagio	plagio	plagio	**FILE23.R**	plagio	noplagio	noplagio	noplagio
FILE17.R	plagio	plagio	plagio	plagio	**FILE15.R**	noplagio	plagio	noplagio	plagio
FILE21.R	plagio	plagio	plagio	plagio	**FILE24.R**	noplagio	plagio	noplagio	noplagio
FILE11.R	plagio	plagio	plagio	plagio	**FILE12.R**	noplagio	noplagio	noplagio	noplagio
FILE13.R	plagio	noplagio	plagio	noplagio	**FILE20.R**	noplagio	noplagio	noplagio	noplagio
FILE26.R	plagio	noplagio	plagio	noplagio	**FILE27.R**	noplagio	noplagio	noplagio	noplagio
FILE14.R	plagio	plagio	plagio	noplagio	**FILE29.R**	noplagio	noplagio	noplagio	noplagio
FILE22.R	plagio	noplagio	noplagio	noplagio	**FILE7.R**	noplagio	noplagio	noplagio	noplagio
FILE10.R	noplagio	plagio	noplagio	noplagio	**FILE9.R**	noplagio	noplagio	noplagio	noplagio
FILE25.R	noplagio	plagio	noplagio	plagio	**FILE30.R**	noplagio	noplagio	noplagio	noplagio

Tabla 6.9: Variables predictoras transformadas (umbral 0.7)

En la tabla 6.10 se muestra el rendimiento alcanzado para los distintos marcos de datos $\{th_i;\ \text{i}= 0.65, 0.7, 0.75, 0.8\}$.

	smoothness	**precision**	**sens**	**spec**	**accuracy**	**roc_auc**
th065	0.718	0.75	0.667	0.75	0.7	0.792
th070	1.176	1.00	0.833	1.00	0.9	0.917
th075	1.000	1.00	1.000	1.00	1.0	1.000
th080	1.350	0.50	1.000	0.50	0.8	0.750

Tabla 6.10: Métricas obtenidas con los distintos valores umbral

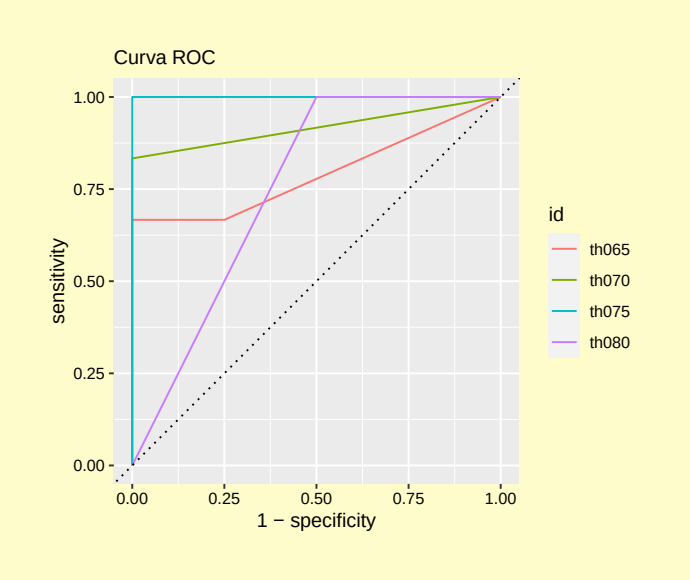


Figura 6.2: Curva ROC para distintos valores umbral

Conclusiones

En la tabla 6.10 se muestran las mejores métricas obtenidas del proceso de validación cruzada de cada marco de datos, es decir, para cada conjunto de datos se representan las métricas de la iteracción que ha proporcionado mejor accuracy. Para el marco de datos **th075** se obtienen unos ajustes perfectos, clasificando correctamente el 100 % de los archivos, es trivial que el valor de la especificidad y sensibilidad es 1.

Se puede concluir que las variables predictoras halladas de la transformación ($\theta^*, \varphi^*, \kappa^* > 0.75$) clasifican de forma muy correcta la detección de plagio / no plagio.

Para comprobar que los resultados obtenidos para el marco de datos **th075** no son debidos a una única mejor iteracción, se realiza una validación cruzada más compleja y se obtienen las medias y desviaciones típicas de las métricas.

.metric	mean	n	std_err
accuracy	0.950	800	0.005
roc_auc	0.927	800	0.007
sens	1.000	800	0.000
spec	0.881	800	0.011

Tabla 6.11: Métricas validación cruzada para el valor umbral 0.75

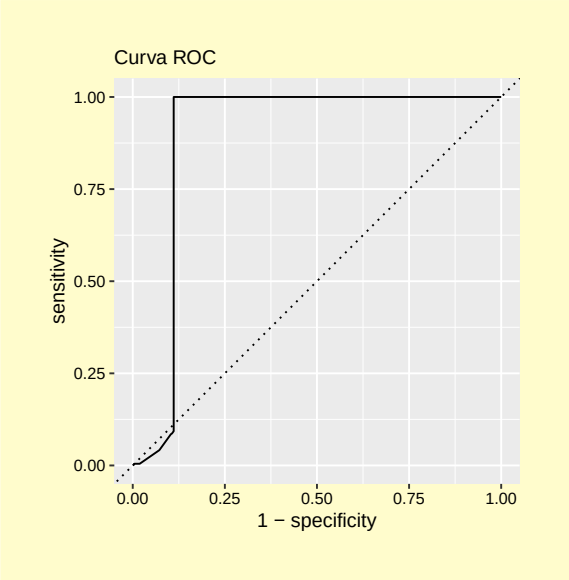


Figura 6.3: Curva ROC (valor umbral 0.75)

Observamos que los resultados son muy buenos, el 94.9 % de las clasificaciones se han predicho de forma correcta, catalogando el 100 % de los plagios y un 88 % de los no plagios correctamente. El ajuste del test, AUC = 0.926, es muy bueno. Se puede decir que las variables predictoras clasifican correctamente la existencia o no de plagio.

Se concluye que la fiabilidad de SimilaR2 bajo las condiciones del análisis es bastante buena. Pero se debe recalcar que los resultados son orientativos para este conjunto de datos y se deben comprobados manualmente.

6.6. Análisis de clústering

El objetivo de efectuar un análisis clústering es comprobar si dentro del conjunto de archivos dataset existen diferentes grupos homogéneos, es decir, de los 30 archivos del dataset, al menos 18 archivos originales (no plagiados) deben encontrarse en un solo grupo, y el resto deben formar otros clústeres (2, 3, . . .).

La matriz de distancia utilizada para efectuar el análisis clúster es:
Siendo

$$\Theta = \begin{bmatrix} \theta_{1,1} & \theta_{1,2} & \cdots & \theta_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ \theta_{n,1} & \theta_{n,2} & \cdots & \theta_{n,n} \end{bmatrix} \tag{6.3}$$

Θ es una matriz simétrica, donde cada coeficiente proporciona información sobre la similitud de plagio entre cada par de archivos. Para convertir Θ en matriz distancia se efectúa la siguiente transformación:

$$d_\Theta = 1_n - \Theta$$

Se aplica la función **clustering** del paquete SimilaR2 a la matriz Θ con los parámetros `cut_k =3` y `agglomerative = TRUE`. Las opciones y resultados que proporciona están detalladas en el Apéndice A.2, pág 13.

Elección de parámetor:

- **Clúster jerárquico** aglomerativo. Este método es bueno para identificar clústeres grandes.
- **Número de clúster**. Se selecciona un tamaño de clúster de 3. El proceso de plagiado se ha efectuado de dos formas disntintas, por tanto, deben existir al menos tres grupos (2 plagios y 1 original).

6.6.1. Resultados

Definición 5 El valor de la silueta es una medida que indica cómo de bueno es el agrupamiento de cada clúster. Este valor está comprendido en un intervalo de [-1, 1], donde los valores cercanos a -1 es un mal agrupamiento, a 0 indiferentes y a 1 buen agrupamiento. Para calcular el valor silueta se tiene en cuenta la cohesión y la separación de cada punto x del conjunto de datos, donde la cohesión a(x) es la distancia promedio de x a todos los demás puntos en el mismo clúster; y la separación b(x) es la distancia promedio de x a todos los demás puntos en el clúster más cercano.

El coeficiente de silueta para el punto x está definido como:

$$s(x) = \frac{b(x) - a(x)}{max\{a(x), b(x)\}}$$

El coeficiente de Silhouette para todo el agrupamiento es:

$$SC = \frac{1}{N} \sum_{i=1}^{N} s(x)$$

Al aplicar la función clustering sobre d_Θ con k = 3, se obtienen los siguientes resultados:

	Resultado
The best method of agglomeration is to	ward.D
Agglomeration coefficient	0.8633
Average silhouette width	0.34

Tabla 6.12: Resultados clústering

El método que mayor coeficiente aglomerativo proporciona es **Ward**, con un valor de 0.863, esto representa la existencia de una estructura de agrupación fuerte. El valor medio de silueta, 0.34, indica una cohesión entre los grupos no muy buena.

cluster	size	ave.sil.width
1	4	0.94
2	22	0.20
3	4	0.51

Tabla 6.13: Valores silueta por clúster

La coesión del primer clúster es muy apropiada 0.94, el clúster 3 se alcanza una cohesión bastante aceptable, 0.51. Por último, el clúster 2, con un valor silueta de 0.22, puede sugerir la existencia de un nuevo grupo.
Las pruebas realizadas con cuatro clústeres dan peores resultados de cohesión.

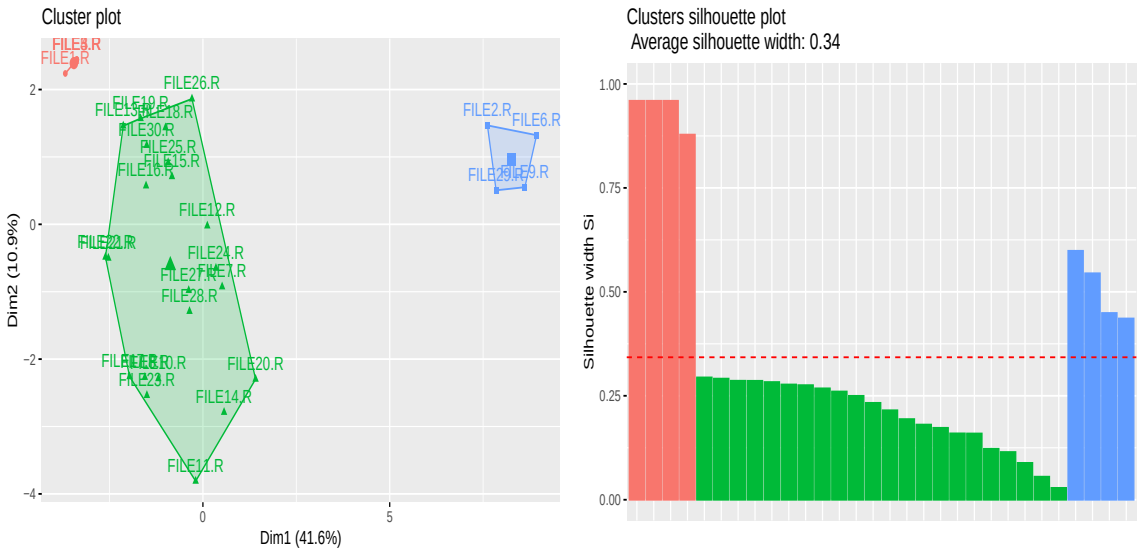
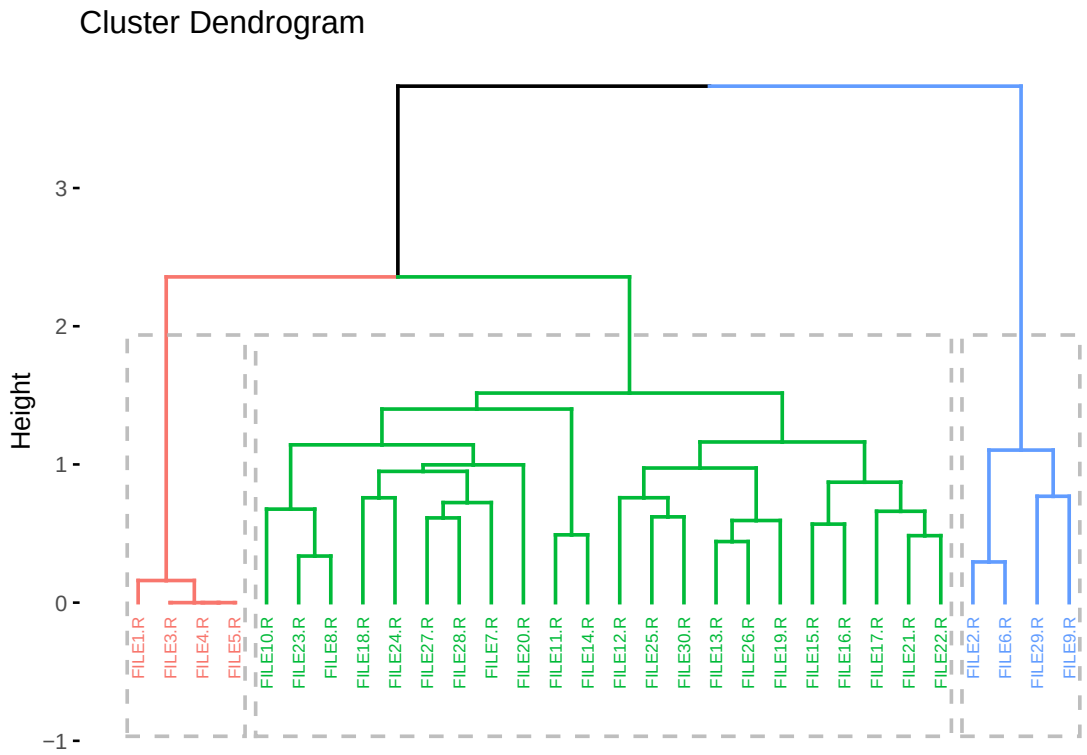


Figura 6.4: Gráficos clustering

En el gráfico de dispersión se observa que los elementos están agrupados correctamente y que no existen elementos de solapamientos. El gráfico de silueta no se resaltan nuevas conclusiones.



En dendrograma se puede observar la agrupación de los elementos en cada clúster, se concluye que:

- Los archivos 1, 3, 4 y 5 podrían formar un grupo de plagio.
- Los archivos 2, 6, 29 y 9 podrían formar otro grupo de plagio.
- El resto de archivos forma el grupo de archivos originales.

El dataset de partida está compuesto de 12 archivos plagiados y 18 originales. Por otro lado, en el dendrograma de ha detectado un grupo original de 22 archivos y dos posibles grupos de pllagio (con un total de 8 archivos). Por tanto, se clasifica correctamente el 88m6 % de los documentos.

6.7. Comparativa de detección de plagio utilizando SimilaR y SimilaR2

En este apartado se comparan los resultados obtenidos de aplicar las herramientas SimilaR y SimilaR2 sobre el dataset. Para ello, hay que tener en cuenta las siguientes particularidades:

- SimilaR solo analiza funciones.
  En el documento anexo I (A.1) existe solo una función (ejercicion5.2), este el esqueleto de cada archivo del dataset. Por tanto, SimilaR únicamente analiza esa parte del documento.
- SimilaR2 analiza todo el documento. Secciones de comentarios y todo el código ejecutable.

- Como SimilaR no retorna un valor de similitud entre pares de documentos se ha decidido utilizar la medida $\overline{\varsigma}$, explicada en 3.1, para comparar cada par de documentos.
- Por lo general, se obtiene una mayor similitud con SimilaR. Puesto que solo compara una pequeña parte del script y además es una función bastante sencilla de programar. Por tanto, es muy probable obtener un falso positivo.

Estos contratiempos dificultan el análisis estadístico. Dado que al no poder comparar las mismas partes de códigos no se pueden obtener resultados concluyentes.
Por las razones mencionadas se ha decidido aplicar estadística descriptiva para tener una visión generalizada de esta comparación.

Marco de datos y evaluación

El marco de datos que se ha construido consta de 5 variables:

- Para la herramienta SimilaR, $\overline{\varsigma}^*$ medida que proporciona el valor máximo de similitud ($\overline{\varsigma}_{i,j}$) entre el archivo i-ésimo y j-ésimo.
- des_varsigma. Valor dicotómico (0, 1), siendo 1 si $\overline{\varsigma}^* > 0.75$, 0 en c.c.
- Para la herramienta SimilaR2, θ^* medida que proporciona el valor máximo de similitud ($\theta_{i,j}$) entre el archivo i-ésimo y j-ésimo.
- des_theta. Valor dicotómico (0, 1), siendo 1 si $\theta^* > 0.75$, 0 en c.c.
- respuesta. Veáse 6.1.

File	$\overline{\varsigma}^*$	des_varsigma	θ^*	des_theta	respuesta
FILE2.R	1.0000	1	0.9606	1	1
FILE6.R	1.0000	1	0.9606	1	1
FILE1.R	1.0000	1	0.9551	1	1
FILE5.R	1.0000	1	0.9551	1	1
FILE4.R	1.0000	1	0.9299	1	1
FILE3.R	1.0000	1	0.8573	1	1
FILE11.R	1.0000	1	0.7460	0	1
FILE13.R	1.0000	1	0.7152	0	0
FILE26.R	1.0000	1	0.7152	0	0
FILE14.R	1.0000	1	0.6909	0	0
FILE8.R	1.0000	1	0.6637	0	1
FILE23.R	1.0000	1	0.6360	0	0
FILE10.R	0.8714	1	0.6800	0	0
FILE21.R	0.8333	1	0.7658	1	1
FILE16.R	0.8333	1	0.6367	0	0

Tabla 6.14: head(15) Marco de datos SimilaR vs SimilaR2

Para comparar ambas herramientas vamos a obtener el accuracy, especificidad y sensitividad para des_varsigma-respuesta y des_theta-respuesta. En 6.3, pár evaluación, se puede ver las definiciones de las medidas de evaluación.

A continuación se muestran los gráficos de calor de las matrices de confusión obtenidas:

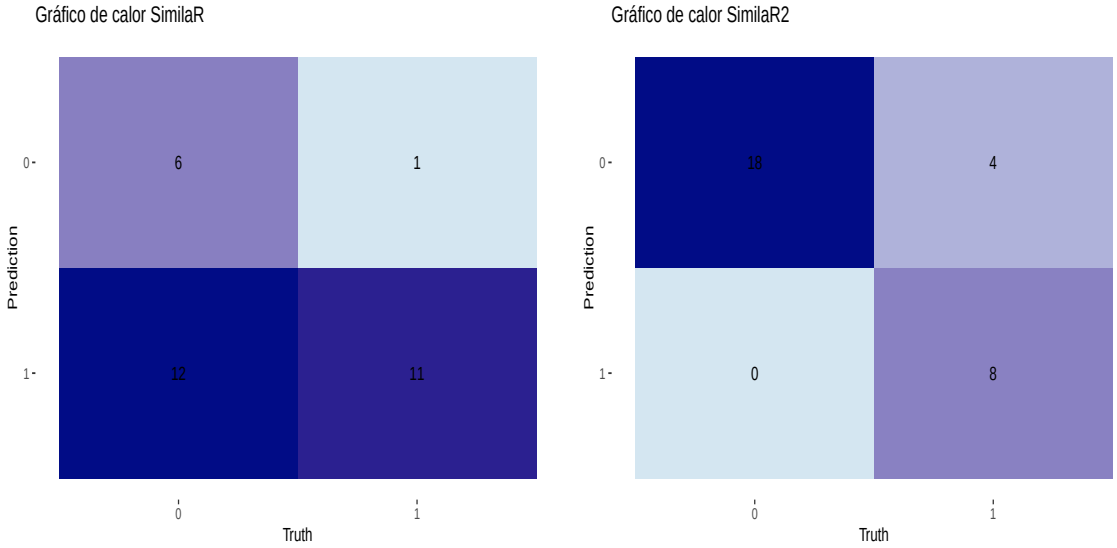


Figura 6.5: Gráficos de calor

SimilaR2 obtiene bastante mayor acierto con respecto a la otra herramienta, clasifica correctamente todos los `no plagios` y falla el 33.33 % de los `plagios`. SimilaR falla el 66.6 % en la clasificación de los `no plagios` y un 8.33 % de los `plagios`.

Es normal obtener una proporción alta de falsos positivos en SimilaR, ya que solo analiza una función y es muy sencilla de programar. Siendo muy probable la programación de un código muy similar sin existir plagio real.

tool	.metric	.estimate
SimilaR	accuracy	0.5667
SimilaR2	**accuracy**	**0.8667**
SimilaR	sens	0.3333
SimilaR2	**sens**	**1.0000**
SimilaR	**spec**	**0.9167**
SimilaR2	spec	0.6667

Tabla 6.15: Metricas SimilaR vs SimilaR2

El porcentaje de acierto de SimilaR2 es del 86.67 %, mientras que SimilaR solo acierta el 56.67 %. La nueva herramienta ha clasificado correctamente un 30 % mejor a su antecesora.

Para finalizar hay que destacar dos casos de este análisis:

- **Caso 1**: En el dataset analizado SimilaR2 analiza todo el documento y SimilaR únicamente analiza la función. La nueva herramienta proporciona una medida de similitud entre pares de documentos y su antecesora no. Por tanto, se puede concluir que SimilaR2 ofrece mayores prestaciones a SimilaR.

- **Caso 2**: El análisis comparativo de las herramientas es poco exhaustivo debido a las limitaciones de SimilaR. Por tano, no se puede concluir que una herramienta sea más eficaz que otra. Pero se puede destacar la siguiente conclusión:
    - En 4.2, pár anterior, se demostró que en la comparación de dos documentos la medida κ era mejor o igual que $\overline{\varsigma}$. Será igual si solo existe una función y más precisa en el resto de los casos. Por tanto, Si la función SimilaR2 utiliza κ para cuantificar la similitud de los códigos de cada par de documentos, es trivial que los resultados obtenidos por SimilaR2 serán mejores o iguales a los de SimilaR.

Capítulo 7

Conclusiones finales

En este capítulo se enumeran las conclusiones sobre el trabajo realizado y se plantean algunas líneas de trabajo futuro.

7.1. Conclusiones sobre el trabajo

El objetivo principal de este Trabajo Fin de Grado ha sido *desarrollar una herramienta para la detección de plagio en código R y RMarkdown*. Para alcanzar este objetivo se ha implementado el paquete SimilaR2, por medio de su función homónima proporciona una medida cuantitativa de la similitud de plagio entre cada par de documentos R o RMarkdown, a su vez, se aplica machine learning para evaluar la fiabilidad de la herramienta, hallándose unos resultados muy satisfactorios. Por estas razones se considera superado el objeto de este trabajo.

Asimismo, el paquete SimilaR2 y la app shiny han solventado los restantes objetivos de este TFG de la siguiente forma:

- Con la implementación de los algoritmos fingerprints en la función *plagiarism_comments* se consigue detectar plagio en secciones de comentarios.
- La función interna write_func_SimilaR genera un nuevo script donde convierte las partes de código que no son funciones en funciones y añade subíndices a las funciones renombradas.
  Al aplicar SimilaR al nuevo script se consiguen cumplir los siguientes objetivos:
  - Se analiza todo el código ejecutable. Al estar todas las partes de código ensambladas en funciones, SimilaR consigue analizar todo el código.
  - Se añaden subíndices a las funciones redefinidas. Debido a esta implementación, SimilaR consigue proporcionar una salida más limpia.

  Todo el proceso se ejecuta dentro de la función *plagiarism_code*.
- Se añade la funcionalidad de detectar plagio en archivos RMarkdown. Para ello, se programa una función interna que convierte un archivo Rmd a R y elimina las partes afuncionales para el análisis (yaml, warnings, messages, entre otros); el procesamiento del nuevo script es análogo al de archivos R, p.e. sí se quiere hallar la similitud en comentarios de dos archivos RMarkdownn, se convierten a archivos R y se les aplican las funciones de *plagiarism_comments*.

- En la salida principal de la función SimilaR2, **final_score**, se consigue ofrecer una salida clara e informativa. Proporcionando los resultados más relevantes de la detección de plagio entre cada par de documentos, tales como:
  - φ. Valor que cuantifica la similitud de los comentarios. Y π_1 (0 o 1), si $\varphi >$ α_1`(parámetro)` se considera que los comentarios son muy parecidos. En c.c. son disímiles.
  - κ. Valor que cuantifica la similitud de los códigos ejecutables. Y π_2 (0 o 1), si $\kappa > \alpha_2$`(parámetro)` se considera que los códigos son muy parecidos. En c.c. son disímiles.
  - θ. Valor que cuantifica la similitud global. Y Π (0 o 1), si $\theta > \alpha_3$`(parámetro)` se considera que los documentos son muy parecidos. En c.c. son disímiles.
- Tras la programación de la aplicación shiny se consigue una interfaz vistosa, clara y detallada. La cual ofrece información de forma interactiva de la salida principal, así como de otras salidas.
- En el análisis de resultados se obtienen varios resultados concluyentes:
  - Se evalúan diferentes marcos de datos mediante Naive Bayes utilizando validación cruzada y se concluye que los parámetros `k = 3` y `w = 4` para los algoritmos fingerprints ofrecen los mejores resultados.
  - Al evaluar qué valor umbral aporta las mejores métricas, se obtiene que el marco de datos resultado de la transformación θ^*, φ^*, $\kappa^* > 0.75$ alcanza resultados óptimos (accuracy de `0.95`, sensibilidad de `1` y especificidad de `0.88`). A su vez, se concluye que la fiabilidad de SimilaR2 es buena.
  - El análisis de clustering realizado con la función de SimilaR2 sobre el dataset ha proporcionado unos resultados bastante aceptables. Con 3 clústeres ha clasificado correctamente el 86.8 % de los documentos, proporcionando un coeficiente aglomerativo de 0.863 y un valor medio de silueta de 0.34.
    También, con los resultados obtenidos se puede concluir que la función clustering del paquete SimilaR2 realiza correctamente su propósito.
  - En la comparación entre SimilaR y SimilaR2 se concluye:
    - SimilaR2 ofrece prestaciones más completas a SimilaR:
      - ◇ SimilaR2 analiza la similitud entre pares de documentos proporcionando medidas totales y desagregadas.
      - ◇ SimilaR analiza las funciones de cada par de documentos y ofrece únicamente la similitud de pares de funciones. No proporciona una medida de similitud entre documentos.
    - Se demuestra que con la variable κ se obtienen iguales o mejores resultados que con la medida $\overline{\varsigma}$ (obtenida para medir la similitud entre documentos en SimilaR). Si la función SimilaR2 utiliza κ para cuantificar la similitud de los códigos de cada par de documentos, es trivial que los resultados obtenidos por SimilaR2 serán mejores o iguales a los de SimilaR.

7.2. Limitaciones e ideas de mejora de SimilaR2

A continuación, se exponen una serie de mejoras para perfeccionar las funcionalidades de la herramienta SimilaR2 y su aplicación:

- **Los algoritmos fingerprints** pueden ser mejorados con la programación de una función capaz de efectuar un análisis sintáctico, esta se aplicaría a continuación del proceso parsing perfeccionando la obtención de tokens y hashes; lo cual implica una mayor optimización de los algoritmos.
- Las mejoras propuestas para **el paquete R SimilaR2** son:
  - Disminuir las dependencias de otras librerías, para facilitar al usuario la instalación del paquete. Para solventar este problema se pueden sustituir las funciones de otras librerías por funciones del paquete base, o bien utilizar importaciones de estas funciones, a través de la librería `import`.
  - Subir el paquete al repositorio oficial de R ,**CRAN**, a fin de ofrecer mayor facilidad en la instalación.
    Nota: CRAN retorna varios warnings y messages al intentar subir el paquete SimilaR2. Seguramente al disminuir las dependencias de otras librerías se subsanarían estos errores.
- Existen redundancias entre funciones del paquete SimilaR2. Se puede optimizar el paquete conectando estas funciones para eliminar las partes redundantes.
- Se puede mejorar la app Shiny concatenando ambas aplicaciones (Interfaz y Resultados). Por ejemplo, realizando la aplicación Resultados en javascript, puesto que sí se puede ejecutar un archivo .js desde shiny.

Finalmente, se podría crear una base de datos capaz de recopilar (a petición del usuario) los documentos de un análisis efectuado por SimilaR2. A su vez, se implementaría una función capaz de comparar una batería de documentos con otros especificados (mediante un parámetro) de la base de datos.

Apéndice A

Apéndice: Anexos

A.1. Anexo I: Plantilla dataset

Plantilla de ejercicios de introducción a R

Ejercicios de vectores, factores y gráficos.

Los datos necesarios para realizar los siguientes ejercicios se encuentran en el fichero repaso1.RData.

Ejercicio 1

El vector telefonos contiene el número de lineas telefonicas en distintas zonas del mundo y distintos años. Los datos están organizados por zonas, y para cada zona, por años.

Los vectores lugar y fecha contienen la información relativa a las zonas y años de los que se tiene información.

(1) Crear un factor con la zona asociada a cada dato del vector telefonos, y otro factor con la fecha.

(2) Calcular un vector telefonos1961 con el número de líneas telefónicas en cada zona en el año 1961; nombrar cada dato con la zona correspondiente.

(3) Calcular el vector porcentaje1961 con el porcentaje que representan las líneas telefónicas de cada zona en el año 1961.

Ejercicio 5

El fichero estadisticas_vitales.txt contiene una tabla de datos relativos a las estadísticas vitales genéricas para la población española. La tabla consta de 49 filas, cada una de las cuales se corresponde con un año, desde 1948 hasta 1996. Para cada fila existen 3 columnas que contienen la siguiente información:

Población total en ese año.
Número de nacimientos.
Número de defunciones.

Se pide construir un marco de datos con el contenido del fichero anterior y escribir expresiones de R que seleccionen del mismo la siguiente información:

1. La población en el año 1992.

2. Definir una función crecimientoAño que, dado un año, devuelva el creciemiento vegetativo correspondiente a dicho año. En caso de no existir el dato en la tabla, la función debe producir un error y mostrar el siguiente mensaje: “Sólo disponibles para los años entre 1948 y 1996”.

3. El número de nacimientos desde el primer hasta el último año.

4.La población, el número de nacimientos y el número de defunciones del año 1960.

5. La población, el número de nacimientos y el número de defunciones desde el año 1950 hasta el año 1989.

6.El número de nacimientos y defunciones de los años 1950, 1960, 1970, 1980 y 1990.

A.2. Anexo II: Documentación de la nueva herramienta

Documentación del paquete SimilaR2

Package 'SimilaR2'

Isaac Gandullo

27/8/2021

R topics documented:

Documentación del paquete SimilaR2

Package: SimilaR2

Type: Package

Title: Package 'SimilaR2'

Version: 0.1.0

Author: Isaac Gandullo

Maintainer: El mantenedor del paquete igandullo91088@gmail.com

Description: Implementación de mejoras complementarias del paquete SimilaR. Añadiendo la capacidad de detectar plagio en secciones de comentarios y todo el código ejecutable de cada par de documentos. También se proporciona una medida para el plagio total entre estos. Dichas mejoras se logran gracias a la implementación de los algoritmos fingerpritns y a una serie de transformaciones en la ejecución y salida de SimilaR.

Depends: R (>= 3.2)

Imports: cluster, dplyr, factoextra, ggplot2, purrr, SimilaR, stringr, knitr, tm, tibble, tidyr

License: GPL-3

Encoding: UTF-8

RoxygenNote: 7.1.1

NeedsCompilation: no

Packaged: 2021-08-27 18:08:56 UTC; igand

Documentación del paquete SimilaR2

SimilaR2-package

`SimilaR2-package` *El paquete SimilaR2*

Descripción

El paquete SimilaR2 consta de cuatro funciones:

1. plagiarism_code. Ver plagiarism_code() para más detalles.
2. plagiarism_comments. Ver plagiarism_comments() para más detalles.
3. SimilaR2. Ver función SimilaR2() para más detalles.
4. clustering. Ver clustering() para más detalles.

Autor

Isaac Gandullo

plagiarism_code

`plagiarism_code.` *Cuantifica la similitud del código ejecutable de pares de documentos R o Rmd.*

Descripción

La función **plagiarism_code** está programada para detectar plagio en los códigos programados de cada par de archivos R o Rmd de un directorio. Es recomendable utilizar esta función en casos donde tenga poco sentido o no se necesite analizar comentarios, como por ejemplo; tareas guiadas en código R, exámenes en código R donde no tengan importancia las explicaciones, entre otros.

Documentación del paquete SimilaR2

Uso

```
plagiarism_code(
  path,
  dir_Rmd,
  th_kap,
  type_file = c("R", "Rmd"),
  View_wr_scp = c(TRUE,FALSE),
  encoding
)
```

Argumentos

- **path**. Ruta de un directorio con archivos de tipo R o Rmd.
- **type_file**: Tipo de archivo ("R" o "Rmd").
- **dir_Rmd**. Ruta del directorio secundario. En esta ruta se crea la carpeta `binder`. Esta contiene los archivos R modificados para que SimilaR pueda analizar todo el código ejecutable.
- **th_kap**. Valor en el intervalo [0,1]. Se utiliza como valor umbral para la variable `kappa`.
- **View_wr_scp**. Recibe (TRUE o FALSE). TRUE, se muestra la carpeta binder en dir_Rmd. FALSE, se elimina la carpeta binder.
- **encoding**. Codificación de los documentos que se analizan.

Detalles

La agregación que se utiliza para ejecutar SimilaR es both, a partir de la salida obtenida se calculan las medidas `tau` y `beta` y se halla `kappa = $\dfrac{\texttt{tau + beta}}{2}$`.

Evaluación

La función retorna los siguientes tres resultados:

- **table_SimilaR_mod**, es una modificación de la salida que ofrece SimilaR:
  - `parametro`.
    - `tau`. tau = coeficiente SimilaR12, es decir, las filas donde `parámetro = tau` sirven para identificar que el coeficiente SimilaR12 es tau.
    - `beta`. beta = coeficiente SimilaR21, es decir, las filas donde `parámetro = beta` sirven para identificar que el coeficiente SimilaR21 es beta.

Documentación del paquete SimilaR2

- `file1`. El nombre del primer archivo de un par.
- `function1`. El nombre de la primera función de un par.
- `file2`. El nombre del segundo archivo de un par.
- `function2`. El nombre de la segunda función de un par.
- `SimilaR12`. Valores en el intervalo [0,1]. Cuantifica el grado en que la primera función es un subconjunto de la segunda. Nótese que para el cálulo de kappa en cada par de archivos, se tiene en cuenta los valores de SimilaR12 de cada par de archivos donde parámetro = tau.
- `SimilaR21`. Valores en el intervalo [0,1]. Cuantifica el grado en que la segunda función es un subconjunto de la primera. Nótese que para el cálulo de kappa en cada par de archivos, se tiene en cuenta los valores de SimilaR21 de cada par de archivos donde parámetro = beta.
- `decision`. 0 o 1; 1 dos funciones son muy parecidas y 0 en caso contrario. Nótese que esta es la decision del algortimo SimilaR.

Esta salida sirve para ayudar al usuario a identificar las funciones con mayor índice de plagio en ambos documentos.

- **Matriz_Kappa**. Se muestra la matriz de distancia K. Es una matriz cuadrada y simétrica, la cual contiene los valores $d_\kappa = 1 - \kappa$ de la comparación de cada par de archivos. Esta matriz es utilizable para efectuar análisis clústering y comprobar la existencia de grupos de plagio.
- **table_Kappa**. Tabla que muestra los resultados del análisis de similitud en los códigos de cada par de archivos. Es un marco de datos con las siguientes 4 columnas:
  - `File1`. El nombre del primer archivo de un par.
  - `File2`. El nombre del segundo archivo de un par.
  - `kappa`. El valor de kappa se obtiene de los valores tau y beta de cada par de archivos: `kappa = E[beta]` $\cdot$ `0.5 + E[tau]` $\cdot$ `0.5`.
  - `decision_kap`. 0 o 1. La decisión tomada según el valor kappa y el valor umbral establecido en el parámetro `th_kap`. 1 indica que los códigos de ambos archivos son muy similares y 0 que son disímiles.

Documentación del paquete SimilaR2

Ejemplos

```
# Comprueba el código plagiado de cada par de documentos
# con extensión .R situados en el directorio "TEST/Ejemplos"

plagiarism_code(
  path = "TEST/Ejemplos", dir_Rmd = "TEST/Ejemplos",
  th_kap = 0.75,
  type_file = "R",
  View_wr_scp = FALSE,
  encoding = "UTF-8"
)


# Comprueba el código plagiado de cada par de documentos
# con extensión .Rmd situados en el directorio "TEST/EjemplosRmd"

plagiarism_code(
  path = "TEST/EjemplosRmd", dir_Rmd = "TEST/EjemplosRmd",
  th_kap = 0.75,
  type_file = "Rmd",
  View_wr_scp = FALSE,
  encoding = "UTF-8"
)
```

plagiarism_comments

---

plagiarism_comments. *Cuantifica la similitud de las secciones de comentarios de pares de documentos R o Rmd.*

Descripción

La función **plagiarism_comments** está diseñada para detectar plagio en los comentarios de cada par de archivos R o Rmd de un directorio. Es aconsejable usar esta función en casos donde la importancia de la detección de plagio en código R o Rmd recaiga en la interpretación de resultados. Por ejemplo, una tarea o examen en R donde la solución del código programable es muy homogénea.

Documentación del paquete SimilaR2

Uso

```
plagiarism_comments(path,
  dir_Rmd,
  type_file = c("R","Rmd"),
  k,
  t,
  th_fps,
  filter_comment= c(TRUE, FALSE, data.frame),
  language,
  encoding
)
```

Argumentos

- **path, type_file, dir_Rmd, encoding*‘. **Ver** plagiarism_code. Argumentos**.
- **k**, longitud de kgram, con este parámetro se calculan los hashes de los algoritmos fingerprints.
- **t**, longitud mínima de palabra a detectar. Se utiliza para calcular `w = t - k + 1`, siendo `w` el tamaño de desplazamiento de ventana, con este se hallan los hashes finales del algoritmo winnowing.
- **th_fps**. Valor en el intervalo [0,1]. Se utiliza como valor umbral para la variable `varphi`.
- **filter_comment**. Líneas de comentarios que no se van a analizar. Admite (TRUE, FALSE, data.frame):
  - `TRUE`. En línea de comandos aparece en mensaje `enter non-comparable paragraphs (enter 'ok' to end):` donde se deben introducir las líneas a descartar hasta finalizar con `ok`.
  - `data.frame`. Recibe un data.frame donde cada fila debe contener una línea de comentarios a eliminar.
  - `FALSE`. No se descartan comentarios.
- **language**. Lenguaje en el que se analizan las secciones de comentarios en el proceso parsing.

Detalles

En el parámetro `filter_comment` se deben introducir las líneas a descartar tal y como aperecen en el script R o Rmd. No se eliminará la línea si al introducirla se erra en una letra, palabra o signo de puntuación.

Documentación del paquete SimilaR2

Evaluación

La función retorna los siguientes resultados:

- **Matrix_Phi_fp**. Se muestra la matriz de distancia Φ_{fp}. Es una matriz cuadrada y simétrica que contiene los valores $d_{\varphi_{fp}} = 1 - \varphi_{fp}$ de la comparación de cada par de archivos.
- **Matrix_Phi_win**. Se muestra la matriz de distancia Φ_{win}. Es una matriz cuadrada y simétrica que contiene los valores $d_{\varphi_{win}} = 1 - \varphi_{win}$ de la comparación de cada par de archivos
- **tab_res_Sorensen**. Tabla que muestra los resultados del análisis de similitud en los comentarios de cada par de archivos. Es un marco de datos con las siguientes 6 columnas:
  - `File1`. El nombre del primer archivo de un par.
  - `File2`. El nombre del segundo archivo de un par.
  - `varphi_fp`. Valor del coeficiente Sorensen, obtenido de cuantificar la similitud de los hashes (aplicando algoritmo fingerprint) de un par de archivos.
  - `decision_fp`. 0 o 1. La decisión tomadasegún el valor `varphi_fp` y el valor umbral establecido en el parámetro `th_fps`. 1 indica que los comentarios de ambos archivos son muy similares, y 0 que son disímiles.
  - `varphi_win`. Valor del coeficiente Sorensen, obtenido de cuantificar la similitud de los hashes (aplicando algoritmo winnowing) de un par de archivos.
  - `decision_win`. 0 o 1. La decisión tomada según el valor `varphi_win` y el valor umbral establecido en el parámetro `th_fps`. 1 indica que los comentarios de ambos archivos son muy similares, y 0 que son disímiles.
- **freq_letter**. Es una lista que contiene tantos elementos como archivos se analizan. Correspondiendo el elemento i-ésimo de la lista con el archivo i-ésimo analizado, cada elemento contiene la tabla de frecuencias de palabras de ese archivo. La tabla de frecuencia de palabras consta de dos columnas:
  - `word`: palabra
  - `freq`: frecuencia con la que aparece la palabra.

Documentación del paquete SimilaR2

Ejemplos

```
# Comprueba los comentarios plagiados de cada par de documentos
# con extensión .R situados en el directorio "TEST/Ejemplos"

plagiarism_comments(path = "TEST/Ejemplos",
  dir_Rmd = "TEST/Ejemplos",
  type_file = "R",
  k = 3, t = 6,
  th_fps = 0.75,
  filter_comment= FALSE,
  language = "spanish",
  encoding = "UTF-8
)
```

```
# Comprueba los comentarios plagiados de cada par de documentos
# con extensión .Rmd situados en el directorio "TEST/EjemplosRmd"

plagiarism_comments(path = "TEST/EjemplosRmd",
  dir_Rmd = "TEST/EjemplosRmd",
  type_file = "Rmd",
  k = 3, t = 6,
  th_fps = 0.75,
  filter_comment= FALSE,
  language = "spanish",
  encoding = "UTF-8
)
```

SimilaR2. Función

`SimilaR2-función` *Cuantifica la similitud de pares de documentos*

Descripción

La función SimilaR2 cuantifica ´la similitud de cada par de archivos R o Rmd de un directorio. Este valor se calcula con la fórmula, theta = $\mathtt{p} \cdot$ varphi + $\mathtt{p}^{c} \cdot$ kappa, es decir, a partir de la importancia de los comentarios, $\mathtt{p} \in [0, 1]$, se calcula una proporción de similitud global de cada par de documentos; por defecto `p = 0.5`.
Opcionalmente, la función es capaz de proporcionar resultados desagregados de los comentarios y códigos, así como realizar un análisis clústering.

Documentación del paquete SimilaR2

Uso

```
SimilaR2(path,
  dir_Rmd,
  type_file = c("R","Rmd"),
  View_wr_scp = c(TRUE, FALSE),
  th_kap,
  th_fps,
  k,
  t,
  filter_comment = c(TRUE, FALSE, data.frame),
  language,
  type_hash = c("finger", "winnow"),
  p,
  th_Sim2,
  cut_Kap,
  cut_Phi,
  cluster = c(TRUE, FALSE),
  agglomerative = c(TRUE, FALSE),
  breakdown_comments= c(TRUE, FALSE),
  breakdown_code = c(TRUE, FALSE)
)
```

Argumentos

- **type_file, th_kap, View_wr_scp, dir_Rmd, encoding**. Ver **plagiarism_code. Argumentos**.
- **k, t, th_fps, filter_comment, language**. Ver **plagiarism_comments. Argumentos**.
- **type_hash**. Admite los valores ("finger", "winnow"). Si **finger** retorna los coeficientes Sorensen, `varphi_fp`, obtenidos mediante el algoritmo fingerprint. Si **winnow** retorna los coeficientes Sorensen, `varphi_win`, calculados a partir del algoritmo winnowing.
- **p**. Valor en el intervalo [0,1]. Indica la importancia de los comentarios en el proceso de detección de plagio. Siendo el complementario de p, la importancia del código en el análisis. A partir de p y p^c se obtiene el valor `theta`= p · `varphi` + `kappa` ·p^c,
- **th_Sim2**. Valor en el intervalo [0,1]. Se utiliza como valor umbral para la variable `theta`.
- **clúster**. Recibe (TRUE o FALSE). TRUE, se muestran los análisis clústering efectuados sobre las matrices Φ y K. FALSE, no se muestran estos resultados.
- **cut_Kap**. Determina el número de clústeres que se forman en el análisis clústering de la matriz K.

Documentación del paquete SimilaR2

- **cut_Phi**. Determina el número de clústeres que se forman en el análisis clústering de la matriz Φ.
- **agglomerative**. Recibe (TRUE o FALSE). TRUE, realiza un análisis clústering jerárquico por el método de aglomeración. FALSE, se aplica método de división.
- **breakdown_comments**. Recibe (TRUE o FALSE). TRUE, retorna una salida desagregada de los comentarios obtenida de plagiarism_comments `(tabla_Sorensen, freq_letter)`. FALSE, no se muestran los resultados.
- **breakdown_code**. Recibe (TRUE o FALSE). TRUE, retorna una salida desagregada del código obtenida de plagiarism_code `(tabla_mod_SimilaR, tabla_Kappa)`. FALSE, no se muestran los resultados.

Detalles

La salida de la funicón SimilaR2 esta compuesta por `final_score` y `breakdown`. Donde `breakdown` es una lista compuesta entre 0 y 3 elementos, dependiendo del número de parámetros (`cluster, breakdown_code, breakdown_comments`) que obtengan el valor TRUE.

Evaluación

- **final_score**. Es la salida principal de SimilaR2. Compuesta por una tabla que muestra los resultados del análisis de similitud total de cada par de archivos. Se compone de las siguientes 8 columnas:
  - `File1`. El nombre del primer archivo de un par.
  - `File2`. El nombre del segundo archivo de un par.
  - `kappa`. Valor en [0,1]. Cuantifica la similitud de los códigos de un par de archivos.
  - `pi_1`. 0 o 1. La decisión tomada según el valor kappa y el valor umbral establecido en el parámetro `th_kap`. 1 indica que los códigos de ambos archivos son muy similares, y 0 que son disímiles.
  - `varphi`. Valor en [0,1]. Cuantifica la similitud de los comentarios de un par de archivos.
  - `pi_2`. 0 o 1. La decisión tomada según el valor `varphi` y el valor umbral establecido en el parámetro `th_fps`. 1 indica que los comentarios de ambos archivos son muy similares, y 0 que son disímiles.
  - `theta`. Valor en [0,1]. Cuantifica la similitud de cada par de archivos R o Rmd de un directorio. Este valor se calcula con la fórmula, theta = $\mathtt{p} \cdot$ varphi + $\mathtt{p}^{c} \cdot$ kappa
  - `Pi`. 0 o 1. La decisión tomada según el valor `theta` y el valor umbral establecido en el parámetro `th_Sim2`. 1 indica que ambos archivos son muy similares, y 0 que son disímiles.
- **breakdown**. Se componen de otras listas con salidas complementarias:

Documentación del paquete SimilaR2

- **breakdown_comments**. Devuelve la lista de elementos `tab_res_Sorensen, freq_letter` de la salida plagiarism_comments. Renombrándolos con los nombres `table_Sorensen y freq_letter`. Para más detalles, ver **plagiarism_comments. Evaluación**.
- **breakdown_code**. Devuelve la lista de elementos `table_SimilaR_mod, table_Kappa` de la salida plagiarism_code. Renombrándolos con los nombres `SimilaR_mod, table_Kappa`. Para más detalles, ver **plagiarism_comments. Evaluación**.
- **clústering**. Retrona las listas de los análisis clústering de las matrices Φ y K. Estos se nombran mediante `cluster_Phi, cluster_Kappa`. Para más detalles, ver **Clústering. Evaluación**.

Ejemplos

```
# Comprueba la existencia de plagio en cada par de documentos
# con extensión .R situados en el directorio "TEST/Ejemplos"


SimilaR2(path = "TEST/Ejemplos", dir_Rmd ="TEST/Ejemplos",
  type_file = "R", View_wr_scp = FALSE,
  th_kap = 0.75, th_fps = 0.75, k = 3, t = 6,
  filter_comment = FALSE, language = "spanish",
  type_hash = "winnow",
  p = 0.5, th_Sim2 = 0.75,
  cut_Kap = 3, cut_Phi = 3, cluster = TRUE,
  agglomerative = TRUE,
  breakdown_comments= TRUE, breakdown_code = TRUE
)

# Comprueba la existencia de plagio en cada par de documentos
# con extensión .Rmd situados en el directorio "TEST/EjemplosRmd"

SimilaR2(path = "TEST/EjemplosRmd", dir_Rmd ="TEST/EjemplosRmd",
  type_file = "Rmd", View_wr_scp = FALSE,
  th_kap = 0.75, th_fps = 0.75, k = 3, t = 6,
  filter_comment = FALSE, language = "spanish",
  type_hash = "winnow",
  p = 0.5, th_Sim2 = 0.75,
  cut_Kap = 3, cut_Phi = 3, cluster = TRUE,
  agglomerative = TRUE,
  breakdown_comments= TRUE, breakdown_code = TRUE
)
```

Documentación del paquete SimilaR2

Clustering

`Clústering` *Análisis clústering de una matriz de distancia*

Descripción

La función **clústering** realiza un analisis clustering jeráquico. Su finalidad es comprobar si en la batería de documentos analizados existen grupos de `plagio` / `no plagio`.

Uso

```
clustering(
  M,
  cut_k,
  agglomerative = c(TRUE, FALSE)
)
```

Argumentos

- **M**. Recibe una matriz de distancia, simétrica y cuadrada.
- **cut_k**. Determina el número de clústeres que se forma en el análisis clústering
- **agglomerative**. Admite (TRUE o FALSE). TRUE, realiza un análisis clústering jerárquico mediante un tipo de clasificación aglomerativa. FALSE, se aplica un tipo de clasificación disociativa.

Evaluación

- **coef**. Depende del valor del parámetro `agglomerative`:
  - FALSE. Se muestra una tabla con dos medidas obtenidas del análisis clúster: `Division coefficient` y `Average silhouette width`.
  - TRUE. Se calculan los coeficientes aglomerativos por los métodos media, vecino más cercano, vecino más lejano y Ward. Se muestra una tabla con la siguiente información: `The best method of agglomeration`, `Agglomeration coefficient` y `Average silhouette width`.
- **Res_Silhouette**. data.frame de dos columnas: tamaño cluster: `size` y valor silueta: `ave.sil.width`.
- **dispersion_diagram**. Gráfico de diagrama obtenido del análisis clústering.
- **Dendogram**. Dendrograma obtenido del análisis clústering.
- **Silhouette_graph**. Gráfico de silueta obtendio del análisis clústering.

Documentación del paquete SimilaR2

Ejemplos

```
# Análisis clústering jerarquico por el método de aglomeración.

clustering(M1,
  agglomerative = FALSE,
  cut_k =3
)

# Análisis clústering jerarquico por el método de división.

clustering(M1,
  agglomerative = TRUE,
  cut_k = 4
)
```

Bibliografía

[1] *Package SimilaR*, 2020. URL https://cran.r-project.org/web/packages/SimilaR/SimilaR.pdf.

[2] Alex Ari M. Barmawi Agung Toto Wibowo, Kadek W. Sudarmadi. *Comparison between fingerprint and winnowing algorithm to detect plagiarism fraud on Bahasa Indonesia documents.* URL https://www.researchgate.net/publication/261078881_Comparison_between_fingerprint_and_winnowing_algorithm_to_detect_plagiarism_fraud_on_Bahasa_Indonesia_documents.

[3] Alex Aiken. *MOSS. A System for Detecting Software Similarity*, 1994. URL http://theory.stanford.edu/~aiken/moss/.

[4] Anónimo. *Haskell el lenguaje de programación funcional*, 2020. URL https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/que-es-haskell/.

[5] Claudia Mariana Banchoff Tzancoff, Anahí Soledad Rodríguez, and Francisco Javier Díaz. Herramientas para la detección de plagio de software. In *XIV Congreso Argentino de Ciencias de la Computación*, 2008.

[6] Jurriaan Hage Bastiaan Heeren. *helium: The Helium Compiler.*, 2014-2015. URL https://hackage.haskell.org/package/helium.

[7] Abdul Sabor Bostan. *Winnowing Algorithm for Program Code*, 2017. URL https://www.sts.tu-harburg.de/pw-and-m-theses/2017/abdul17.pdf.

[8] Universidad de Almería. *SafeAssign, ¿Que és?*, 2019. URL http://www2.ual.es/aulavirtual/docs/estudiante/bblearn/herramientas/safe-assign/.

[9] Carlos Iván Espinosa. *Similitud de comunidades biológicas*, 2019. URL https://ciespinosa.github.io/Similitud/indices-de-similitud.html#indices-cuantitativos.

[10] Isaac G. Gandullo-Lara. *Desarrollo de una herramienta para la detección de plagio en código R*, 2021.

[11] Jurriaan Hage, Brian Vermeer, and Gerben Verburg. Plagiarism detection for haskell with holmes. In *CSERC*, pages 19–30, 2013.

[12] ML Kammer, HL Bodlaender, and J Hage. Plagiarism detection in haskell programs using call graph matching. *Utrcht University, Master's thesis*, pages 1–87, 2011.

[13] Juan López. *QUÉ ES R SOFTWARE*, 2018. URL https://www.maximaformacion.es/blog-dat/que-es-r-software/.

[14] Various python programmers. *ast — Abstract Syntax Trees*, 2021. URL https://docs.python.org/3/library/ast.html.

[15] Angel Robledano. *Qué es Python: Características, evolución y futuro*, 2019. URL https://openwebinars.net/blog/que-es-python/.

[16] Timur Saglam. *JPlag - Detecting Software Plagiarism*, 2013. URL http://jplag.ipd.kit.edu/.

[17] Alex Aiken Saul Schleimer, Daniel S. Wilkerson. *Winnowing: Local Algorithms for Document Fingerprinting*. URL https://theory.stanford.edu/~aiken/publications/papers/sigmod03.pdf.

[18] Saul Schleimer, Daniel S Wilkerson, and Alex Aiken. Winnowing: local algorithms for document fingerprinting. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 76–85, 2003.

[19] LLC Turnitin. Turnitin, 2017. URL https://www.turnitin.com/es.

[20] Urkund. Urkund, 2018. URL https://www.urkund.com/es/.

[21] user:thektulu github user: fyrestone github. *pycode-similar 1.4*, 2020. URL https://pypi.org/project/pycode-similar/.

[22] user weigelt github. *Legacy version of JPlag*, 2019. URL https://github.com/jplag/jplag/releases/tag/v2.12.1-SNAPSHOT.

[23] Michael J Wise. String similarity via greedy string tiling and running karp-rabin matching. *Online Preprint, Dec*, 119(1):1–17, 1993.