Detección de plagio en texto

Ángel Guevara, Arturo Sirvent , Miguel Hortelano, Francisco Olayo González

2021-11-15

Contents

1 Introducción y motivación

En los últimos años, el plagio y su detección se han posicionado como un problema en crecimiento dentro del ámbito académico. Según la Real Academia Española (RAE), plagiar es la acción de copiar en lo sustancial obras ajenas dándolas como propias. En el caso de textos, plagiar es la acción de atribuirse la autoría de fragmentos o ideas cuya propiedad intelectual no le corresponde al autor (Dario Funez 2012).

Debido al aumento de interés en la detección de plagio, han surgido diferentes soluciones por parte del mundo empresarial y del educativo. Entre otras, el desarrollo de sistemas para ayudar a los educadores en la ardua tarea de identificar trabajos plagiados, o incluso el desarrollo de sistemas que son capaces de detectar el plagio automáticamente.

Uno de los problemas de estos sistemas de detección de plagio es la falta de datos para entrenar los modelos ya que no hay muchos repositorios con textos que estén admitidos como plagiados. Esto se debe a la ilegalidad de publicar textos plagiados. Por eso, en este trabajo usaremos un corpus de texto plagiado que ha sido simulado por los investigadores y que está disponible para uso libre.

A lo largo de este trabajo, explicaremos como funcionan algunos de estos detectores de plagio, incluyendo todas las etapas del proceso:

- Desarrollo de un corpus de textos.
- Preprocesado del texto previo al análisis.
- Comparación de textos a través de una métrica.
- Evaluación de resultados.

2 Teoria

En este apartado desarrollamos los conceptos y técnicas necesarias para llevar a cabo más tarde en un caso práctico la construcción de un detector de plagio. Esta teoría esta subdividida en los diferentes pasos a seguir en la práctica para el tratado de los datos y la construcción y evaluación del modelo.

Atendiendo a los diferentes tipos de plagio y los métodos usados para su detección, podemos realizar distintas clasificaciones. A continuación se exponen algunas de ellas (Hussain Chowdhury 2016) (Vani Kanjirangat 2016).

Nota: También es muy importante diferenciar respecto al objeto de estudio. Si se está analizando plagio en código, las técnicas no serán igual que si estamos tratando con un artículo científico. Nuestro interés se centra en texto plano y no en código.

La clasificación menos relevante para nuestro trabajo sería la que atiende a la intención de quien escribe el texto. Dicha clasificación podría ser (Shivani 2016):

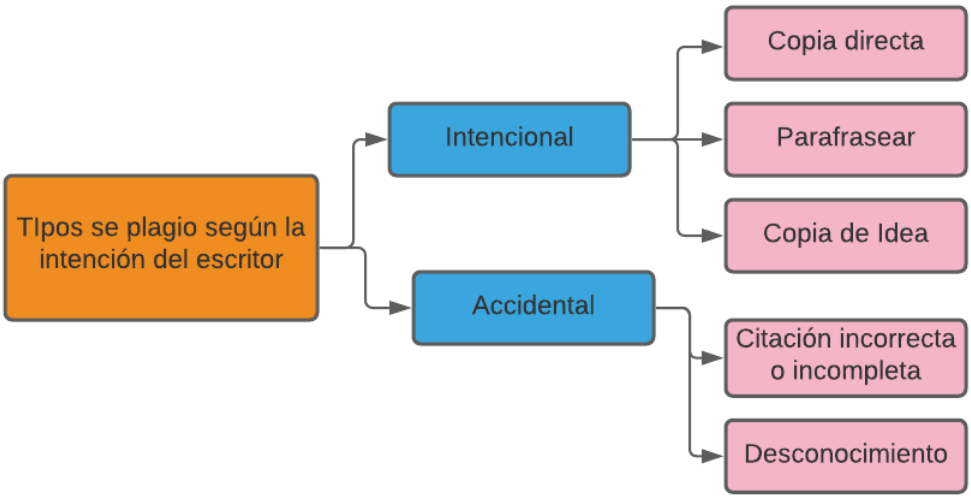

Figure 1: Clasificación del pragio según la intención del escritor

Una de las clasificaciones que se hacen más comúnmente sobre el plagio, son las de plagio intrínseco y el plagio extrínseco.

- Plagio intrínseco: Se habla de plagio intrínseco cuando en un mismo documento o texto, cambia el estilo de redacción y se distinguen diferentes autores. Su nombre viene del hecho que no se requieren de bases de datos ni fuentes externas para la comparación. Como consecuencia de dicho método, solo podremos sacar una sospecha del plagio, pues no contaremos con la fuente del plagio.

- Plagio extrínseco: Donde los recursos originales o fuente del plagio son usados y comparados con el texto a analizar. El plagio no solo puede venir de una copia textual entre documento original y ejemplar (llamado *plagio literal* o *copia-pega*), si se usan sinónimos y se reescriben las oraciones de manera similar, deberemos atender a la similitud entre ambos documentos (este tipo de plagio es llamado *plagio inteligente*). Esta similitud puede separarse en:
  - Similitud Semántica: Considerando el uso de sinónimos, se busca la similitud de la estructura por **palabras**.
  - Similitud Sintáctica: Se busca una similitud por oraciones, donde se le da relevancia a similitud entre oraciones a nivel sintáctico.

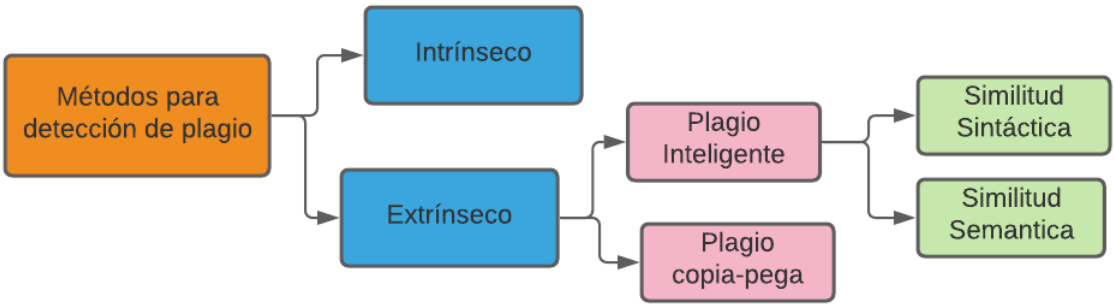

Figure 2: Clasificación del plagio

2.1 Corpus

Un corpus lingüístico es un conjunto de textos variados que cumple la función de representar el uso real de la lengua que se quiere examinar. En el caso del procesamiento de texto es necesario contar con un corpus adecuado al campo y características que se buscan extraer. En nuestro caso hacemos uso del corpus proporcionado por miembros de la universidad de Sheffield (P Clough 2009). Consta de un set de 5 cuestiones que veinte alumnos tuvieron que responder. Los alumnos se dividieron en cuatro grupos que podían responder las preguntas con materiales externos o consultando páginas de Wikipedia, permitiéndose niveles diferentes de copia y referencia. De esta forma tenemos diferentes grados de plagio con el documento original, que van del corta y pega al no uso de la fuente, y una fácil comparación. Además de todas estas respuestas, también se proporciona un índice con el alumno que escribió cada texto, cual era la pregunta original y el nivel de copia (near copy, light revision, heavy revision, non- plagiarism) debidamente indexado. En resumidas cuentas, nuestro corpus cuenta con 100 textos breves (entre 200 y 300 palabras) con diferentes niveles de plagio, de manera que son lo suficientemente largos para realizar un análisis didáctico sin convertirse en una carga difícil de procesar con nuestros medios.

2.2 Preprocesado

El preprocesado es el primer contacto con los datos, las primeras tareas de limpieza y formateo de los datos suelen ser por lo general sencillas y en su mayoría genéricas.

Primero exponemos las funciones básicas para este propósito:

```
#librerías para cargar los datos y preprocesarlos
```

```
texto<-" ¡ Ésto és ún  TexTo De  çñ(Prúebá)"
#pasarlo a minúsculas
texto<-tolower(texto)
texto
```

```
## [1] " ¡ ésto és ún  texto de  çñ(prúebá)"
```

```
#eliminar signos de puntuación, carácteres problemáticos
texto<-gsub("[[:punct:]]"," ",texto)
texto<-gsub("(\\n)|(\\t)"," ",texto)
texto
```

```
## [1] "   ésto és ún  texto de  çñ prúebá "
```

```
#y comprobar que no hay espacios repetidos y al principio y final del string (str_trim())
texto<-gsub("\\s+"," ",str_trim(texto))
#la función str_squish() de la librería stringr hace exactamente
#elimilar los espacios en blanco al comienzo y al final del string
texto
```

```
## [1] "ésto és ún texto de çñ prúebá"
```

En el caso de que el idioma tenga acentuación, sería recomendable eliminarla para no tener problemas de formato ni considerar diferentes palabras iguales solo por estar acentuada o no. Esto lo podemos hacer de forma sencilla usando `stringi::stri_trans_general()`.

```
texto<-stringi::stri_trans_general(texto,"Latin-ASCII")
# este proceso será muy dependiente del idioma del texto
texto
```

```
## [1] "esto es un texto de cn prueba"
```

Una limpieza más intensiva podría seguir con la eliminación de palabras unión `stop-words` y hasta la reducción de palabras hasta su raíz (término *stemming* en inglés). Hay librerías que implementan funciones para dicho proposito en R, algunas de ellas son `tm` y `tidytext`.
Sin embargo esta sobre-simplificación puede no ser lo más recomendado en todos los casos. La elección será muy dependiente del algoritmo y del tipo de plagio buscado (ya hemos comentado previamente que hay diferentes tipos de plagio a detectar).

```
#eliminar stop-words con la librería tm

texto2<-removeWords(texto,stopwords("es"))
texto2<-str_squish(texto2)
texto2
```

```
## [1] "texto cn prueba"
```

```
#stemming palabras con la libreria tm
texto2<-unlist(str_split(texto2," "))
texto3<-stemDocument(texto2,language = "es")
texto3
```

```
## [1] "text"  "cn"    "prueb"
```

2.3 Tokenizado

Una vez importado el corpus y depurado los datos, el siguiente paso es tokenizarlos. Esto consiste en separar en elementos más pequeños el conjunto de datos que tenemos. Hablando de datos de tipo texto, tokenizar podría ser separar en palabras o en frases por ejemplo. La función `strsplit()` nos permitiría hacer dichas acciones variando el argumento de separación, `split`, según nos convenga.

- `split=" "` un espacio podría servir para separar en palabras.
- `split="."` un punto podría servir para separar en frases.

Sin embargo, en este caso nos va a interesar usar N-gramas, que son subconjuntos del conjunto original que van cogiendo combinaciones de n elementos y dónde el elemento siguiente está formado por los (n-1) elementos del anterior N-grama más un elemento nuevo. Por ejemplo, si tenemos el texto: "Hoy hemos quedado en la rusa a las seis." Su división en 3-gramas (o trigramas) sería:

[Hoy hemos quedado], [hemos quedado en], [quedado en la], [en la rusa],[la rusa a], [rusa a las],[a las seis].

En R, se puede hacer fácilmente usando el paquete `ngram`, y más concretamente la función `unnest_tokens()`, que toma como argumentos principales los siguientes:

- `tbl`: un data frame
- `output`: el nombre de la columna que se va crear
- `input`: los datos que queremos usar.
- `token`: a qué token queremos convertirlos (en este caso el que más nos interesa es "ngrams")
- `n`: argumento adicional para n-gramas para indicar el tamaño de éstos.

Los N-gramas son muy utilizados en aplicaciones de análisis de texto, como por ejemplo en la detección de plagio. La similitud entre dos textos se puede calcular contando el número de N-gramas que tienen en común.

2.4 Métricas y modelos

Las métricas y modelos utilizados a la hora de evaluar los niveles de plagio entre textos, ya sean del mismo autor, o de varios, se deben dividir en varios grupos, dependiendo de la previa tokenización, agrupación de las palabras o letras del corpus.("Comparison of the Text Distance Metrics," n.d.)

2.4.1 Métricas basadas en similitudes de tokenes

Una de las formas más utilizadas a la hora de comparar bloques de texto es la de comparaciones vectoriales, en las que, por ejemplo, cada palabra se convierte en el índice de un vector, y la cantidad de veces que aparece es su escalar. Utilizando este proceso, también se puede extrapolar de manera que en vez de contar las palabras, se cuente los pares o trios de palabras, lo que vendría a ser separación en n-gramas(bigramas o trigramas).("3 Text Distances That Every Data Scientist Should Know," n.d.)

Son bastante eficientes en general, y funcionan bien para textos largos a diferencia de los otros tipos de métricas.

Para esta tokenización, las métricas más utilizadas, son las de medida de distancia entre un texto y otro, comparando vector a vector.

-Una primera métrica seria la de **Jaccard**.

Esta métrica trata de medir los elementos que comparten ambos vectores.

$$D_{jaccard}(X,Y) = \frac{\sum x_i \cap y_i}{\sum x_i \cup y_j}$$

En términos generales, consiste en dividir el vocabulario común de los documentos entre el vocabulario de la unión de ambos documentos, por lo que, cuanto más cerca están de 1 más parecidos son y lejanos cuando se acercan a 0.

-La siguiente métrica es muy parecida ya que es la de **Jaccard pesada**. En esta métrica se tiene en cuenta también la cantidad potencial de coincidencias.

$D_{wjaccard}(X,Y) = \frac{\sum min(x,y)}{\sum max(x,y)}$

-Métrica de **Sorensen**.

$D_{sorensen}(X,Y) = \frac{2*\sum x_i \cap y_i}{\sum x_j \cap 1 + \sum y_k \cap 1}$

Consiste en dividir la cantidad de palabras comunes multiplicadas por 2, ya que se repiten al ser comunes. Esto se divide por todas las palabras tipo de ambos documentos.

-Con **Jaccard** y **Sorensen**, aparece un problema y es que si uno de los dos textos a comparar tiene un tamaño mucho mayor que otro, las distancias tienden siempre a cero, por ello **Overlap** trata de disminuir este problema al dividirlo por el vocabulario más pequeño que corresponde a uno de los documentos.

$D_{overlap}(X,Y) = \frac{2*\sum x_i \cap y_i}{min(\sum x_j \cap 1 + \sum y_k \cap 1)}$

-La métrica **Masi** (Measuring Agreement on Set-valued Items) sigue un proceso parecido al overlap pero tiene en cuenta el mayor.

$D_{masi}(X,Y) = \frac{2*\sum x_i \cap y_i}{max(\sum x_j \cap 1 + \sum y_k \cap 1)}$

-Distancia de **containment**.

Se basa en realizar un conteo de la cantidad de de n-gramas unicos que hay en A y en B y se divide entre el número de n-gramas unicos que hay en A.

$C_n(A,B) = \frac{|S(A,n) \cap S(B,n)|}{|S(A,n)|}$

Podemos usar esta medida de distancia, siempre que el texto fuente sea más grande que los textos posiblemente plagiados a evaluar.

2.4.2 Métricas basadas en similitudes de edición

Estas métricas en general se basan en comparar palabras mediante el número de trasnformaciones que hay que realizar para llegar de una a otra.("Distancias En Texto," n.d.) Se suelen utilizar para comparar distancias linguisticas entre diferentes idiomas, duplicación de textos y correción de léxico. Las más utlizadas son:

-Distancia de **Hamming**.

Compara cada letra de dos palabras basadas por su posición, su ventaja es su velocidad y simplicidad. Por el contrario, es muy estricto ya que necesita que ambas palabras sean del mismo tamaño.

-Distancia de **Levenshtein**.

Compara el número de transformaciones necesarias para transformar una palabra en otra, estas transformaciones se resumen en:

```
-Insertar caracteres.
-Eliminar caracteres.
-Sustituir un caracter por otro.
```

Esta distancia en general es más completa y compleja que la distancia de Hamming.

2.4.3 Métricas basadas en las secuencias de palabras

Compara las diferencias entre dos palabras, teniendo en cuenta las sub-palabras o sub-cadenas de palabras más largas dentro de las palabras iniciales. (Longest common subsequence and longest common substring).

La diferencia básica entre las sub-palabras y las sub-cadenas de palabras, reside en que el primero utiliza letras dentro de la palabra sin necesidad de que sean contiguas, y el segundo si implica que esté una contigua a la otra. Por ejemplo entre las palabras 'palabras' y 'paralelas,' con el método de sub-palabras tendrían en común : 'palas,' mientras que con el método de sub-cadenas sería: 'pala' y 'para.'("Longest Common Subsequence Problem," n.d.)

3 Casos práticos

En este apartado, vamos a aplicar los procedimientos descritos a lo largo del trabajo a un caso real de detección de plagio, descrito en el apartado del Corpus. En primer lugar veremos un caso de similitud semántica, y después veremos otro caso en el que se hace un tokenizado por N-gramas.

3.1 Caso práctico 1 (similitud semántica):

Identificación de plagio mediante una sencilla comparación semántica.
La estrategia para esta técnica será la tokenización del texto en sus raíces semánticas, para luego realizar una comparación con las raíces semánticas del texto original.

El proceso seguido ha sido:
Pre-procesado del texto para dejarlo limpio, eliminación de las stop-words, y tokenizado según sus raíces. Finalmente realizamos una sencilla comparación elemento a elemento entre el texto original y el que se está analizando. De esta comparación obtedremos una lista con las raices coincidentes. Por último, para realizar un análisis de los resultados, se obtuvieron porcentajes para cada texto, dividiendo las raices comunes entre todas las palabras del texto a analizar.

Una vez tenemos el porcentaje de palabras plagiadas para cada texto, podemos establecer un umbral para clasificarlo como **plagio** y así determinar la precisión. En nuestro caso escogimos un umbral de **50%**, pero esto es un parámetro que se puede *tunear* para obtener el mejor rendimiento del modelo.

A continuación mostramos una matriz de confusión con los resultados obtenidos:

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  NO PLAGIO PLAGIO
##   NO PLAGIO        24      2
##   PLAGIO           14     55
##
##                Accuracy : 0.8316
##                  95% CI : (0.741, 0.9006)
##     No Information Rate : 0.6
##     P-Value [Acc > NIR] : 9.961e-07
##
##                   Kappa : 0.6296
##
##  Mcnemar's Test P-Value : 0.00596
##
##             Sensitivity : 0.6316
```

```
##             Specificity : 0.9649
##          Pos Pred Value : 0.9231
##          Neg Pred Value : 0.7971
##              Prevalence : 0.4000
##          Detection Rate : 0.2526
##    Detection Prevalence : 0.2737
##       Balanced Accuracy : 0.7982
##
##        'Positive' Class : NO PLAGIO
##
```

De la matriz anterior sacamos las siguientes observaciones:

- Se ha confundido un 7.6% de no plagiados como sí plagiados.
- El 63 % de los no plagio están bien clasificados.
- El 96 % de los plagios si se clasifican.
- Solo un 20% se los que no debería clasificarse como plagio se clasifican como plagio.

3.2 Caso práctico 2 (N-gramas)

Para el análisis, nosotros lo que tendremos será un data frame con 100 muestras, de las cuales las 95 primeras muestras son la respuesta a una de las preguntas por parte de los alumnos, y las 5 últimas muestras son los enlaces a Wikipedia que podían consultar para responder a cada una de las 5 preguntas. Algunas de las columnas de ese data frame serán: el tipo de plagio con el que se pedía responder, la pregunta que se responde, el individuo que la responde y la respuesta en forma de cadena de texto.

Una vez que tenemos el data frame, lo primero que hacemos es depurar la columna que contiene la respuesta tal y como se ha explicado en el apartado de preprocesado de los datos. En particular, desarrollamos una función `agrupa()` que elimina los signos de puntuación y convierte todo a minúsculas. Una vez hecho esto, dividimos cada una de las respuestas en N-gramas, en concreto hemos cogido N=3 ya que se ha visto que es la que mejor funciona.

```
agrupa <-function(text){
  temp <-unlist(text)
  temptemp <- temp[temp != ""]

  temp2 <- temp[1]
  for(i in 2:length(temp)){
    temp2 <- paste(temp2,temp[i])
  }
  temp2 <- gsub('[[:punct:] ]+',' ',temp2)%>%
    tolower()

  temp <- tibble(word = unlist(temp2))%>%
    unnest_tokens(trigram, word,token = "ngrams",n=3)#cambiar el número para tamaño del ngrama
}
```

Una vez tenemos cada respuesta dividida en trigramas, podemos proceder a aplicar algunas de las métricas descritas en el trabajo para ver si hacen buenas predicciones. Hemos escogido las métricas de Jaccard y la métrica de 'containment.' Definimos las siguientes funciones que nos calculan cada una de las métricas respectivamente.

```
compara <- function(original, sospechoso){

  coinc <- c()
  coinc2 <- c()

  or <- tibble(trigram = unlist(original))
  sus <- tibble(trigram = unlist(unlist(sospechoso)))

  sus <- separate(sus,trigram,c("P1","P2","P3"),sep=" ")
  or <- separate(or,trigram,c("P1","P2","P3"),sep=" ")

  for( i in 1:length(sus$P1)){
    count <- 0;
    count2 <- 0
    sus_str <- c(sus[i,]$P1,sus[i,]$P2,sus[i,]$P3)

    for( j in 1:length(or$P1)){
      or_str <- c(or[j,]$P1,or[j,]$P2,or[j,]$P3)
      count <- count + all(or_str == sus_str)
      count2 <- count2 + sum(or_str==sus_str)
    }

    coinc <- append(coinc,count)
    coinc2 <- append(coinc2,count2)
    score <- sum(coinc)/length(sus$P1)
    salida <- c(score,coinc2)
  }
  return(salida)
}
compara_djabbard <- function(original, sospechoso){

  or <- unlist(original)
  sus <- unlist(sospechoso)

  dj <- length(intersect(sus,or))/length(union(sus,or))
  return(dj)
}
```

A continuación, mostramos una pequeña tabla dónde podemos ver el porcentaje de plagio que nos brinda cada una de las métricas, al comprobar el nivel de plagio entre algunas de las respuestas de la primera y la última pregunta (incluida la fuente de Wikipedia).

Task	Category	djaccard	containment
a	non	0.005882353	0.0138249
a	non	0.007312614	0.0147601
a	heavy	0.05139186	0.1354167
a	cut	0.3908046	0.7653631
a	light	0.8945687	0.9843478
a	non	0.005952381	0.0243309

Task	Category	djaccard	containment
e	light	0.09289617	0.5204082
e	cut	0.5510204	0.9415205
e	light	0.358047	0.8379447
e	heavy	0.1476923	0.4048583
e	non	0.02781641	0.0966387
e	original	1	1.0466926

En las tablas podemos ver el porcentaje de plagio que se le asigna a la respuesta en función de la métrica usada. En general, vemos que todas funcionan bastante bien y nos dan un resultado acorde con la categoría, que recordemos que hace referencia al tipo de plagio que habían usado los alumnos para contestar a la pregunta.

Veamos lo bueno que es nuestro modelo. Ahora vamos a suponer que tenemos solo dos categorías: PLAGIADO y NO PLAGIADO. Dentro de la categoría PLAGIADO están incluidas las categorías "cut," "light" y "heavy." Vamos a evaluar entonces los aciertos del modelo en función de la métrica "containment." Para ello, consideramos que una métrica de "containment" superior a 0.50 se considera dentro de la categoría PLAGIADO, y si obtenemos un "containment" por debajo de 0.5 consideraremos que estamos en la categoría NO PLAGIADO.

Si realizamos la matriz de confusión, obtenemos el siguiente resultado.

```
confusionMatrix(T_abcde$Prediccion,T_abcde$Realidad)
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  NO PLAGIO PLAGIO
##   NO PLAGIO        38     29
##   PLAGIO            0     33
##
##                Accuracy : 0.71
##                  95% CI : (0.6107, 0.7964)
##     No Information Rate : 0.62
##     P-Value [Acc > NIR] : 0.03816
##
##                   Kappa : 0.4638
##
##  Mcnemar's Test P-Value : 1.999e-07
##
##             Sensitivity : 1.0000
##             Specificity : 0.5323
##          Pos Pred Value : 0.5672
##          Neg Pred Value : 1.0000
##              Prevalence : 0.3800
##          Detection Rate : 0.3800
##    Detection Prevalence : 0.6700
##       Balanced Accuracy : 0.7661
##
##        'Positive' Class : NO PLAGIO
##
```

Observamos que tiene un accuracy del 71%, es decir, acierta el 71% de las veces. Es una medida bastante aceptable, por lo que podemos concluir que nuestro pequeño modelo nos sirve para detectar plagio con cierta fiabilidad en respuestas cortas escritas por alumnos.

También podemos visualizar la similitud de los textos gráficamente con esta métrica. En la siguiente figura cada casilla es un ngrama, dispuesto de la manera en que estaría en un texto escrito, y el color representa la similitud encontrada de este ngrama en el texto original. Se aprecian las diferencias entre un texto con fragmentos copiados y pegados (izquierda), y otro sin plagio (derecha).

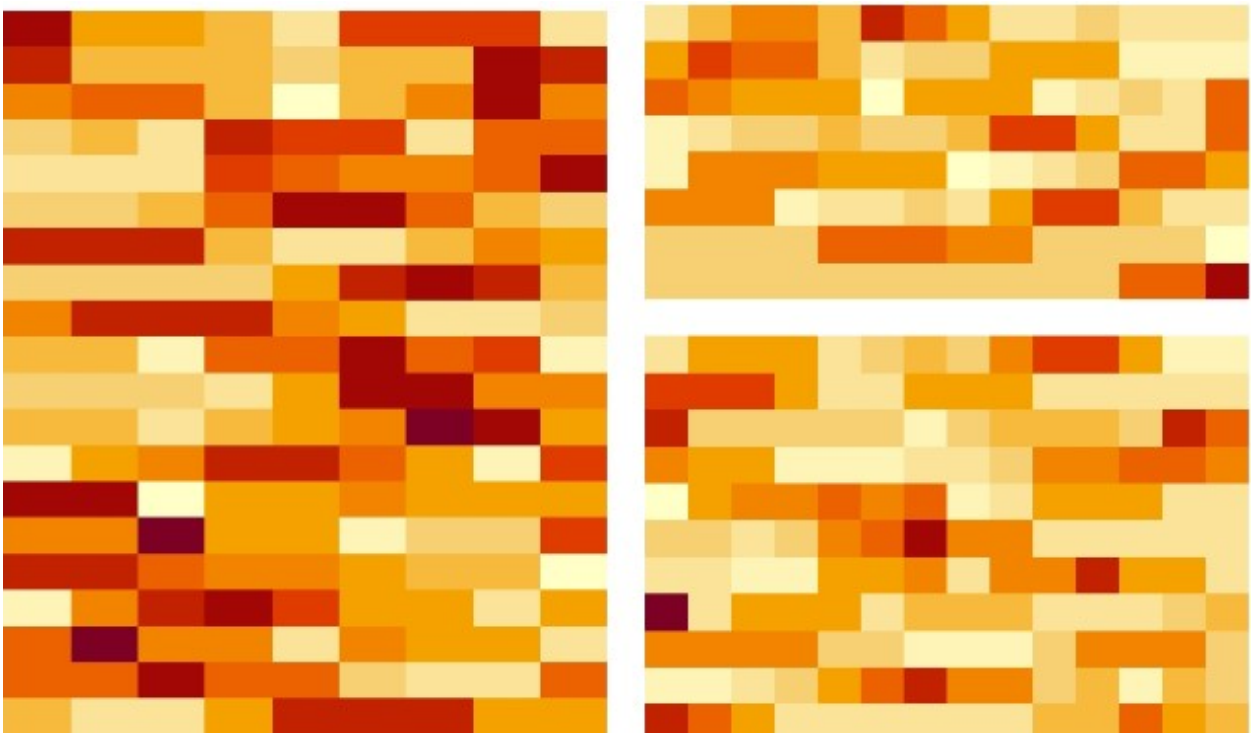

Figure 3: Mapa de calor coincidencia trigramas(Izquierda plagio, Derecha sin plagio)

Bibliografía

"3 Text Distances That Every Data Scientist Should Know." n.d. https://towardsdatascience.com/3-text-distances-that-every-data-scientist-should-know-7fcdf850e510.

"Comparison of the Text Distance Metrics." n.d. https://www.kdnuggets.com/2019/01/comparison-text-distance-metrics.html.

Dario Funez, Marcelo Errecalde. 2012. "Detección de Plagio Intínseco Basada En Histogramas."

"Distancias En Texto." n.d.

Hussain Chowdhury, Dhruba Bhattacharyya. 2016. "Plagiarism: Taxonomy, Tools and Detection Techniques." *Knowledge, Library and Information Networking.*

"Longest Common Subsequence Problem." n.d. https://www.techiedelight.com/longest-common-subsequence.

P Clough, M Stevenson. 2009. "Developing a Corpus of Plagiarised Short Answers, Language Resources and Evaluation: Special Issue on Plagiarism and Authorship Analysis, in Press."

Shivani, Vishal Goyal. 2016. "A Novel Approach for Plagiarism Detection in English Text." *International Journal of Computer Applications.*

Vani Kanjirangat, Deepa Gupta. 2016. "A Study on Extrinsic Text Plagiarism Detection Techniques and Tools." *Journal of Engineering Science and Technology Review.*