Universidad de las Ciencias Informáticas

Facultad 1

Subsistema para la detección de similitud entre noticias para la Plataforma C.U.B.A

Trabajo de diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autor:

Williams Celestino Arrieta Pérez

Tutores:

Ing. Ruben Reynaldo Bonachea

MSc. Delly Lien González Hernández

La Habana, septiembre de 2020

"Año 62 de la Revolución"

Declaración jurada de autoría

Declaro por este medio que yo Williams Celestino Arrieta Pérez, con carné de identidad 96011408607 soy el autor principal del trabajo de diploma titulado “Subsistema para la detección de similitud entre noticias para la plataforma C.U.B.A” y autorizo a la Universidad de las Ciencias Informáticas a hacer uso de la misma en su beneficio, así como los derechos patrimoniales con carácter exclusivo.

Para que así conste se firma la presente, a los ___ días del mes de _____ del año 2020.

_____

Autor

Williams Celestino Arrieta Pérez

_____ _____

Tutor **Tutor**

Ing. Ruben Reynaldo Bonachea MSc. Delly Lien González Hernández

Resumen

La presente investigación tiene como objetivo proponer un subsistema de detección de similitud entre noticias, que permita detectar las noticias similares, para contribuir así, al proceso de recuperación de información en la red cubana. Durante la investigación se analizaron sistemas homólogos existentes, identificando su funcionamiento y los principales algoritmos para la detección de similitud entre textos, así como sus aspectos positivos y negativos. Para darle cumplimiento al objetivo planteado, en la investigación se realizó el estudio del estado del arte sobre el proceso de detección de similitud entre textos, funciones, algoritmos de similitud y la recuperación de información. El proceso de desarrollo estuvo guiado por la metodología de desarrollo de *software* AUP-UCI. Se seleccionaron como principales tecnologías un entorno de desarrollo integrado (IDE) Netbeans 8.1, para la programación, el lenguaje Java 8, como indexador la herramienta Apache Solr 6.3 y Visual Paradigm 8.0, como herramienta para el modelado. A partir de la aplicación de las pruebas de software de integración y funcionalidad se verificó un correcto funcionamiento del subsistema implementado para contribuir a elevar la efectividad de las peticiones realizadas por los usuarios en la sección de noticias de la plataforma C.U.B.A, durante el proceso de recuperación de la información en la red cubana a partir de la detección de similitud entre noticias.

Palabras clave: algoritmos de similitud, detección de similitud, documentos, similitud entre textos, noticias, recuperación de información.

Abstract

The present research aims to propose a similarity detection subsystem between news items, which allows detecting similar news, thus contributing to the information retrieval process on the Cuban network. During the investigation, existing homologous systems were analyzed, identifying their operation and the main algorithms for detecting similarity between texts, as well as their positive and negative aspects. In order to fulfill the proposed objective, the research carried out the study of the state of the art on the process of detecting similarity between texts, functions, similarity algorithms, and information retrieval. The development process was guided by the AUP-UCI software development methodology. Select as main technologies in an integrated development environment (IDE) Netbeans 8.1, for programming, the Java 8 language, as indexer the tool Apache Solr 6.3 and Visual Paradigm 8.0, as a tool for modeling. From the application of the integration and functionality software tests, a correct operation of the implemented subsystem was verified to contribute to increasing the effectiveness of the requests made by users in the news section of the CUBA Platform, during the recovery process of information in the Cuban network from the detection of similarity between news.

Keywords*: similarity algorithms, similarity detection, documents, texts similarity, news, information retrieval.*

Índice

Índice de imágenes

Índice de tablas

Introducción

Desde hace muchos años, ya en la era moderna, cuando se necesitaba información de carácter científico, comercial o de entretenimiento solía encaminarse la búsqueda hacia una biblioteca pública, especializada o académica, en la que un bibliotecario o referencista lo orientaba. Se podía también consultar los tradicionales catálogos de autor, título, materia u otro que describiera los documentos existentes. En el peor de los casos, el problema se resolvía cuando se remitía el usuario a otra biblioteca.

Inevitablemente, con el desarrollo se produjo un crecimiento exponencial de la literatura, sobre todo científica, que aun cuando coloca a disposición de la comunidad académica, una gran variedad de recursos, requiere de una inversión importante de tiempo y esfuerzo para su consulta, evaluación y asimilación.

Con el desarrollo científico y tecnológico, y su crecimiento gigantesco, se ha producido, entre otros fenómenos, el incremento y mejora acelerada de las Tecnologías de la Información y las Comunicaciones (TIC), para lograr un mejor registro, procesamiento, búsqueda y diseminación de la información, donde la Internet ha jugado un papel fundamental debido a la cantidad de información contenida en la Web (Torres, 2003; Berners-Lee, 2001).

Uno de los avances que más impacto social tiene en la actualidad, como resultado de esta rápida evolución de las TIC y la Internet, son los Sistemas de Recuperación de Información (SRI). Entre ellos, se tienen los buscadores, los cuales tributan a la navegación y el hallazgo de la información necesaria de forma rápida y automática. Estos son sistemas que, dado un criterio de búsqueda introducido por los usuarios, obtienen un subconjunto de aquellos documentos que mayor relevancia tengan para dicho criterio (Torres, 2003).

A pesar de existir herramientas disponibles en Internet para la búsqueda y recuperación de información, resulta contradictorio que en Cuba no existiera una manera de buscar información que no fuera en buscadores extranjeros que responden a otros intereses. Estos buscadores en muchos casos limitan muchos de sus servicios para Cuba, por lo que la necesidad de contribuir a la búsqueda y recuperación de información en los contenidos publicados en la red cubana con el fin de lograr un mayor desarrollo

científico y social, se convierte en eje estratégico del proceso de informatización cubano y de la soberanía tecnológica que se impone alcanzar ante el hostil contexto internacional.

Con tal objetivo se crea la plataforma C.U.B.A (Contenidos Unificados para la Búsqueda Avanzada), creada en la Universidad de las Ciencias Informáticas en el año 2015 (Viltres y otros, 2018; Alvarez, 2018), como respuesta a la necesidad de mostrar a los usuarios la información existente en el dominio cubano que brinda sitios de interés cultural, informativo e investigativo. C.U.B.A se basó en las tecnologías del buscador cubano Orión desarrollado en la misma Universidad en el año 2010 (Martínez, 2016).

Como plataforma aglutinadora de contenidos, posee una serie de categorías predefinidas para facilitar las búsquedas. Entre ellas, destacan en su página de inicio accesos directos a sitios de deportes y recreación, entretenimiento, noticias, salud, arte y humanidades, entre otros. Actualmente la plataforma C.U.B.A en la sección de noticias no está preparada para detectar similitud entre estas, por lo que a los usuarios no se les proporciona una amplia diversidad de referencias de una noticia alojada en varias fuentes.

Es por ello que se requieren mecanismos que permitan la detección de similitud entre noticias, que dado un resultado de una consulta realizada por el usuario en la sección de noticias le brinde la opción en cada resultado de optar por una diversa información relacionada con esa noticia (en este trabajo las noticias se refieren únicamente a textos). De esta forma, el usuario emplearía menos tiempo en encontrar lo que necesita, lo haría de una forma más rápida y cómoda, y posiblemente encontraría otros productos y/o servicios interesantes o similares para él.

Teniendo en cuenta la situación problemática descrita anteriormente se enuncia el siguiente **problema de investigación**: ¿Cómo elevar la efectividad de las peticiones realizadas por los usuarios en la sección de noticias de la plataforma C.U.B.A.?

Se enmarca el **objeto de estudio** en el proceso de detección de similitud entre textos.

Como **campo de acción** el procesamiento de la detección de similitud entre noticias en la sección de noticias de la plataforma C.U.B.A.

Objetivo general: Desarrollar un subsistema para la detección de similitud entre noticias para elevar la efectividad de las peticiones realizadas por los usuarios en la sección de noticias de la plataforma

C.U.B.A.

Objetivos específicos:

1. Construir el marco teórico conceptual respecto a las tecnologías y funcionalidades de los algoritmos de similitud de textos.

2. Diseñar el subsistema para la detección de similitud entre noticias.

3. Implementar un subsistema que permita detectar noticias similares.

4. Validar el correcto funcionamiento del subsistema desarrollado.

Para guiar la investigación se plantea la siguiente **hipótesis de investigación**: El desarrollo de un subsistema para la detección de similitud entre noticias que utilice técnicas para procesar el lenguaje natural en la plataforma C.U.B.A. contribuirá a elevar la efectividad de las peticiones realizadas por los usuarios en la sección de noticias de dicha Plataforma.

Teniendo en cuenta la hipótesis anteriormente planteada, se define como **variable independiente**: subsistema de detección de similitud entre noticias, el cual consiste en un subsistema para detectar la similitud entre las noticias en la plataforma C.U.B.A. (Tabla 1).

Como **variable dependiente**: efectividad de las peticiones realizadas por los usuarios en la plataforma C.U.B.A. Esta variable hace alusión a la efectividad con que se obtiene la información relevante de acuerdo al criterio de búsqueda ingresado por los usuarios (Tabla 1).

Tabla 1: Operacionalización de variables.

Variable independiente	**Dimensión**	**Definición conceptual**	**Indicadores**	**Unidades de medida**

Subsistema para la detección de similitud entre noticias	Detección de similitud entre noticias	Subsistema para la detección de similitud entre noticias para la plataforma C.U.B.A que analiza y obtiene noticias similares a las consultadas por el usuario usando técnicas para procesar el lenguaje natural	Capacidad de análisis y obtención de noticias similares a las consultadas por los usuarios	Porcentual
Variable dependiente	**Dimensión**	**Definición conceptual**	**Indicadores**	**Unidades de medida**
Efectividad de las peticiones realizadas por los usuarios en la plataforma C.U.B.A.	Efectividad	Obtener resultados relevantes al criterio de búsqueda de acuerdo al contexto en que se realiza la consulta	Cantidad de documentos similares recuperados de los resultados obtenidos de consultas realizadas por los usuarios	Cantidad

Posibles resultados

Subsistema para la detección de similitud entre noticias.

Métodos de investigación científica

Métodos Teóricos

Histórico-Lógico: utilizado para estudiar el desarrollo evolutivo de los sistemas de detección de similitud entre textos, así como de los algoritmos más utilizados para mejorar la comprensión de su funcionamiento, surgimiento y desarrollo de los sistemas de recuperación de la información, en especial los buscadores web.

Analítico-Sintético: empleado para la descomposición del objeto de estudio, permitiendo combinar los elementos en una propuesta de solución.

Modelado: utilizado para realizar una reproducción simplificada de la realidad, mostrando las relaciones internas y características del subsistema a través de diagramas.

Métodos Empíricos

Entrevista: a través de un conjunto de preguntas realizadas a los principales especialistas del proyecto, permitiendo conocer las características de la plataforma C.U.B.A. y problemas que surgen por falta de un proceso de detección de similitud entre noticias.

El presente trabajo de diploma consta de la siguiente estructura: Introducción, tres Capítulos, Conclusiones, Recomendaciones, Bibliografía y Anexos.

Capítulo 1. "Fundamentación teórica del procesamiento de detección de similitud entre noticias en la plataforma C.U.B.A"

Contiene los fundamentos teóricos que aborda el basamento del subsistema para la detección de similitud entre noticias. Además, se hace un estudio de sistemas homólogos, obteniéndose: objetivos de su uso, funcionamiento y tecnologías que utilizan.

Capítulo 2. "Análisis y diseño del subsistema para la detección de similitud entre noticias"

En este capítulo se explica cómo se desarrolla el flujo actual de los procesos, y se describe la propuesta de solución para resolver el problema planteado. Por otra parte, se especifican los requisitos funcionales y no funcionales, y los elementos fundamentales del diseño y arquitectura.

Capítulo 3. "Implementación y validación del subsistema para la detección de similitud entre noticias"

En este capítulo se incluye la programación realizada a partir de los requisitos, así como las pruebas realizadas para su validación. Además, se establecen los estándares de codificación utilizados para el desarrollo del subsistema.

Capítulo 1: Fundamentación teórica del procesamiento de la detección de similitud entre noticias en la plataforma C.U.B.A.

1.1 Introducción

En este capítulo se precisan los elementos teóricos necesarios para el desarrollo del subsistema. Se abordan conceptos fundamentales referentes al seguimiento de la base tecnológica de la plataforma C.U.B.A. a partir de las entrevistas realizadas a desarrolladores del proyecto (ver Tabla 2 en Anexos) y la revisión bibliográfica asociada a dicha plataforma (Martínez, 2016; Caraballo, 2017; Alvarez, 2018). Se realiza, además, la selección de tecnologías, metodología y herramientas necesarias para lograr la solución deseada en la presente investigación, a partir de lo referido sobre la plataforma C.U.B.A.

1.2.1 Conceptos fundamentales

Con el objetivo de facilitar la comprensión de la investigación, se analizan los principales conceptos asociados al objeto de estudio y al campo de acción. A saber:

Similitud

1. Semejanza, parecido, analogía. Relación entre cosas o personas que tienen características en común (Real Academia Española).
2. La **similitud** semántica en el área de procesamiento de lenguajes naturales, es la medida de la interrelación existente entre dos palabras cualesquiera en un texto. Dos palabras o términos por el hecho de tener su existencia en un mismo documento poseen un contexto similar (Rada et al, 1989).

Noticia

La noticia es un relato de un suceso de actualidad, que provoca el interés del público. Quien la publica tiene la responsabilidad de relatar con la mayor objetividad y veracidad posibles cómo se ha producido ese acontecimiento (Alvarez, 2018).

En esta investigación las noticias se refieren únicamente a textos y siempre que se aborde la similitud entre noticias sería equivalente a similitud entre textos.

Recuperación de información

Para analizar el funcionamiento genérico del proceso de detección de similitud, vale aclarar que la parte más difícil del desarrollo, es la extracción del texto del mismo para su análisis o comparación, ya que es la parte más complicada del proceso. En este punto la recuperación de información juega un papel decisivo.

La recuperación de información es el proceso mediante el cual, partiendo de una colección de textos, tales como resúmenes de libros o una necesidad de información, se devuelven los documentos que mejor satisfacen esa necesidad. Se trata de un proceso con aplicaciones prácticas, como en los buscadores web o bibliotecas digitales (Martínez, 2006; Baeza-Yates y Ribeiro-Neto, 2011; Viltres et al, 2018).

Generalmente el proceso sigue estos pasos:

1. Se analizan los documentos y se transforman a una representación interna (más adelante se explica un método de representación) de cada uno.
2. Se analiza la consulta y se transforma a una representación interna.
3. A partir de las representaciones obtenidas en los pasos anteriores se calcula el grado de similitud entre cada documento y la consulta.
4. Se recuperan los documentos que guardan mayor similitud con la consulta del usuario.

1.2.2 Funciones de similitud

Función de similitud

La similitud semántica en el área de procesamiento de lenguajes naturales, es la medida de relación existente entre dos palabras, este concepto es fundamentado en la idea que se tiene de la lingüística sobre la coexistencia de palabras y del discurso coherente. Dos palabras o términos por el hecho de tener su existencia en un mismo documento poseen un contexto similar, por lo que se puede deducir su distancia semántica (Rada et al, 1989 citado por Alvarez, 2018; Sidorov et al, 2014; Torres y Arco, 2016; Raju et al, 2017; Chinniyan et al, 2017; Wadhawan y Mittal, 2017).

Estas funciones de similitud consideran cada cadena de caracteres como una secuencia ininterrumpida de caracteres. Al examinar los resultados de las investigaciones de Amón y Jimenez (2010), Morato et al (2014), Sidorov et al (2014), Valero (2017) y Selvi et al (2017) fueron identificadas las siguientes funciones de similitud:

Distancia de edición

La distancia de edición entre dos cadenas de texto *A* y *B* se basa en el conjunto mínimo de operaciones de edición necesarias para trasformar *A* en *B* (o viceversa). Las operaciones de edición permitidas son eliminación, inserción y sustitución de un carácter.

En el modelo original, cada operación de edición tiene costo unitario, siendo referido como distancia de Levenshtein. Needleman y Wunsch (1970) lo modificaron para permitir operaciones de edición con distinto costo, permitiendo modelar errores ortográficos y tipográficos comunes. Por ejemplo, en el idioma español es frecuente encontrar la letra “n” en lugar de la “m”, entonces, tiene sentido asignar un costo de sustitución menor a este par de caracteres que a otros dos sin relación alguna (Levenshtein, 1966).

Función Coseno

La similitud entre dos documentos representados en dos vectores de términos se puede calcular mediante el ángulo que forman, en concreto, con la utilización del coseno del ángulo (considerando que cuanto más próximos están dos vectores, mayor es la similitud entre ellos). Cuando el ángulo entre los vectores es menor, la similitud es mayor y en consecuencia el coseno del ángulo es mayor (Seijo et al., 2011; Sidorov et al, 2014). La ecuación (1) representa la fórmula del coseno:

$$sim(d_i, d_j) = cos(\alpha) = \frac{\sum_{k=1}^{m} d_{ik} \cdot d_{jk}}{\sqrt{\sum_{k=1}^{m} d_{ik}^2} \sqrt{\sum_{k=1}^{m} d_{jk}^2}} \qquad \textbf{(1)}$$

Descripción: En la fórmula d_i y d_j son los documentos y d_{ik} representa el peso del rasgo *k* en el documento d_i.

Distancia de brecha afín

La distancia de edición y otras funciones de similitud tienden a fallar al identificar cadenas equivalentes que han sido demasiado truncadas, ya sea mediante el uso de abreviaturas o la omisión de *tokens* ("Jorge Eduardo Rodríguez López" vs "Jorge E Rodríguez"). La distancia de brecha afín ofrece una solución al penalizar la inserción/eliminación de *k* caracteres consecutivos (brecha) con bajo costo, mediante una función afín representada en la ecuación (2), donde *g* es el costo de iniciar una brecha, *h* el costo de extenderla un carácter, y *h* << *g*. Bilenko y Mooney (2003) describen un modelo para entrenar automáticamente esta función de similitud a partir de un conjunto de datos. La distancia de brecha afín no normalizada es calculada con un orden computacional O(*n*) mediante el algoritmo de programación dinámica propuesto por Gotoh (1982).

$$\mathrm{P(k)} = \mathrm{g} + \mathrm{h} * (\mathrm{k} - 1) \qquad (2)$$

Descripción: En la fórmula *k* son los caracteres consecutivos, g representa el costo de iniciar la brecha y *h* el costo de extenderla un carácter.

Similitud Smith-Waterman

La similitud Smith-Waterman (SW) entre dos cadenas *A* y *B* es la máxima similitud entre una pareja denotadas (*A* ', *B* '), sobre todas las posibles, tal que *A*' es subcadena de *A* y *B*' es subcadena de *B*, este problema es conocido como alineamiento local. El modelo original de Smith y Waterman define las mismas operaciones de la distancia de edición, y además permite omitir cualquier número de caracteres al principio o final de ambas cadenas. El análisis del planteamiento anterior demuestra que la similitud SW es adecuado para identificar cadenas equivalentes con prefijos/sufijos que, al no tener valor semántico, pueden ser descartados (Smith y Waterman, 1981).

Es posible normalizar la similitud de Smith-Waterman con base en la longitud de la cadena de mayor longitud, la de menor longitud o la longitud media de ambas cadenas, que corresponden a los coeficientes de Jaccar, Overlap y Dice respectivamente. La similitud Smith-Waterman puede ser calculada en 0 (*nm*)

mediante el algoritmo de Smith-Waterman (1981). Baeza-Yates y Gonnet (1992) presentan un algoritmo que toma 0 (*n*) para verificar si la similitud Smith-Waterman entre dos cadenas es menor que cierta constante *k*.

Similitud de Jaro

Jaro (1980) desarrolló una función de similitud que define la trasposición de dos caracteres como la única operación de edición permitida. Los caracteres no necesitan ser adyacentes, sino que pueden estar alejados cierta distancia *d* que depende de la longitud de ambas cadenas.

Winkler (2000) propone una variante que asigna puntajes de similitud mayores a cadenas que comparten algún prefijo, basándose en un estudio realizado por Pollock y Zamora (1984). En Cohen y otros (2003) se propone un modelo basado en distribuciones Gaussianas. Yancey (2006) compara la eficacia de la similitud de Jaro y algunas de sus variantes. La similitud de Jaro puede ser calculada con un orden de complejidad de 0 (*n*).

Selección de la función de similitud

El análisis de las funciones de similitud identificadas en la revisión bibliográfica permite la selección de la Función Coseno por presentar facilidades de mapeo a dimensiones diferentes en el modelo espacio-vectorial, permitiendo la similitud entre palabras relacionadas semánticamente. La complejidad de esta función es cuadrática, permitiéndole ser completamente aplicable a problemas del mundo real. El estudio de las funciones también demostró que para el desarrollo de la investigación se puede poner en práctica cualquiera de las anteriormente descritas. Luego de aplicar la función se utiliza un algoritmo de detección de similitud de textos para obtener los textos similares.

1.4 Algoritmos utilizados

1.4.1 Características generales de un algoritmo de detección de similitud de textos

Un algoritmo de detección de similitud de textos debe contar con tres propiedades fundamentales, según considera este autor, a partir de la revisión bibliográfica (Rodríguez y Martín, 2010; Aguirre, 2012; Sardiñas, 2016; Gadri y Moussaoui, 2017):

- No debe ser sensible a los espacios en blanco: cuando se hace una comparación entre dos fragmentos de textos, esta no debe afectarse por la existencia de espacios en blanco extras, capitalización, signos de puntuación.
- Eliminación de ruido: se deben eliminar aquellas palabras y caracteres que no contengan información relevante (eliminación de palabras como “en”, “de”, “a”).
- Independencia de posición: la permutación de palabras dentro del documento no debe afectar el proceso de detección.

Sánchez-Vega (2016), hace una comparación entre los diferentes métodos o algoritmos utilizados para la detección de plagio y las características enunciadas por Rodríguez y Martín (2010). De esta manera se logra identificar cuáles de esos parámetros son tenidos en cuenta por cada algoritmo para el proceso de detección.

En esta comparación de Sánchez-Vega (2016) sobresalen tres algoritmos:

• ***Greedy String Tiling***: basa su funcionamiento en el cálculo de subsecuencias comunes entre documentos.

• ***FingerPrint***: su funcionamiento se sustenta en la determinación de un patrón que va a representar el contenido del documento, para la búsqueda del mismo en una base de datos de patrones y detectar posibles similitudes.

• **Modelo Vectorial**: funciona sobre la base de la representación del contenido de los documentos en términos de vectores y posteriormente mediante fórmulas matemáticas se arrojan los resultados de las similitudes.

El primero de estos es un algoritmo *Greedy* (Glotón), por este motivo su funcionamiento es lento, requiere de gran espacio en memoria para almacenar las estructuras de datos usadas para la detección, y se ve afectado más aún cuando el corpus de documentos es grande. Por otro lado este solo detecta copias exactas, por lo que un simple cambio en el orden de las palabras o estas ser sustituidas por sinónimos, afectaría el proceso de detección. Los restantes algoritmos se caracterizan por ser rápidos y requerir poco espacio de almacenamiento por las estructura de datos usadas por cada uno. A continuación se describen.

2. **Algoritmo *Document FingerPrint*: *Winnowing***

   Uno de los algoritmos más usados para la detección de plagio o similitudes de texto, es el que consiste en la obtención de una firma o *fingerprint* de un documento. Utilizando instancia específica del algoritmo *Document Fingerprinting*, llamada *Winnowing* (Aventar), para obtener un mejor resultado en la comparación entre documentos. La *fingerprint* (huella digital o firma) de un documento se refiere al conjunto de valores numéricos que representan a varias porciones del mismo (Aguirre, 2012).

3. **Algoritmo basado en la medida de similitud *Word Chunking Overlap* (WCO)**

   Respecto a la representación interna de un documento, el método más comúnmente utilizado en sistemas de RI es el VSP (*Vector Space Model* o Modelo de Espacio Vectorial), donde la medida de similitud entre dos documentos es calculada mediante la fórmula del coseno. Dando como resultado el valor del ángulo entre los vectores que representan los mismos, mientras más pequeño sea el ángulo más similares serán estos documentos, esta representación es llamada “bolsa de palabras” pues el orden de aparición de estas en el documento no se mantiene, perdiéndose la relación entre estas y como consecuencia su significado. Por lo que tal representación no es la adecuada para sistemas de detección de plagio o similitud que trabajan con texto, además la fórmula del coseno tiene dificultades cuando los vectores de los documentos difieren en tamaño (Aguirre, 2012).

1.5 Análisis de sistemas homólogos

Programas de detección de similitud entre textos, han sido identificados, a nivel internacional y nacional cuya descripción y análisis final se explican seguidamente.

1.5.1 *Wcopyfind*

Es una alternativa de código abierto basada en Windows (http://wcopyfind.findmysoft.com/) que compara documentos e informes similares para buscar parecidos en su texto. Por lo tanto, no compara los textos con una base de datos online, sino que busca plagios en textos que ya se tienen y se añaden de forma manual a la aplicación.

A cambio de esta limitación, se tiene una aplicación totalmente gratuita. Entre las cosas que puede hacer, aunque la configuración es algo compleja, están el escribir fuentes de webs de noticias para buscar similitudes, crear listas de palabras que se repiten o analizar hábitos de escritura para resaltar frases sobrantes.

1.5.2 *Copyleaks*

Es un servicio de análisis de textos (https://copyleaks.com/es/) que incluye función de encontrar plagios. Las 10 primeras páginas escaneadas son gratuitas, pero existen planes de suscripción flexibles, adaptados a empresas y docentes. A los efectos de este programa, se considera que diez páginas pueden ser poco para una maestría, pero útil para trabajos más cortos.

1.5.3 *Plagius*

Esta no es una página web (https://www.plagius.com/es) en la que subir y escanear documentos, *Plagius* es otra aplicación de escritorio que debe descargarse para analizarlo todo de forma local. Examina los documentos de varios formatos como Word, PDF, OpenOffice, HTML o texto plano, y luego muestra los informes detallados con los posibles plagios o documentos similares encontrados. En estos informes se muestran las referencias que se han encontrado en el documento, la frecuencia con las que aparecen en Internet, y un porcentaje que cuanto más alto sea más posibilidades tiene de que provenga de un plagio o documento similar. Además de Internet, también busca coincidencias en archivos locales por si se hubiera copiado alguien a sí mismo sin querer.

1.5.4 QtNLP-Diff

En el contexto nacional, a los efectos de la presente investigación, sobresale este módulo desarrollado en la Universidad Central “Marta Abreu” de Las Villas para la herramienta QtNLP. QtNLP-Diff analiza aspectos del procesamiento del lenguaje natural, las herramientas gráficas y de consola para la

comparación de textos, así como las bibliotecas de procesamientos XML y las características de la herramienta QtNLP. Este trabajo (Sardiñas, 2016) adiciona a QtNLP la posibilidad de “comparar casos de corpus lingüísticos con casos de resultados de algoritmos de detección de similitud textual”. El lenguaje empleado no se corresponde con lo requerido.

El análisis de sistemas homólogos de detección de similitud permitió observar que en Internet existen diversos sistemas de este tipo, pero no responden a las necesidades de la aplicación que se desea realizar. Se detectó como deficiencia que el tipo de licencia es privativa generalmente en el caso del ámbito internacional. En la esfera nacional existe un sistema para la detección automática de similitud de textos, pero no presenta todos los elementos que forman parte de lo que requiere la propuesta de solución. Porque la utilización de Python y Qt como lenguajes de programación no se corresponden con los requisitos tecnológicos de la plataforma C.U.B.A.

1.6 Metodologías de desarrollo

Se utilizó la metodología AUP en una versión adaptada por la UCI. Esta metodología es recomendada para proyectos no muy extensos y permite ser adaptada a las peculiaridades de cada proyecto, lográndose así que el proceso sea configurable.

Las fases de AUP-UCI son: inicio, ejecución y cierre. Durante el inicio del proyecto se elaboran las actividades relacionadas con la planeación. En la segunda fase se ejecutan las actividades requeridas para desarrollar el software, incluyendo el ajuste de los planes del proyecto, considerándose los requisitos y la arquitectura. En la fase de cierre se analizan los resultados del proyecto como su ejecución y se realizan las actividades formales de cierre de proyecto.

AUP-UCI define cuatro escenarios para modelar el sistema en dependencia de la complejidad y características del proceso de desarrollo. El escenario número cuatro de esta metodología se adapta al desarrollo del subsistema propuesto: Aplica a los proyectos que hayan evaluado el negocio a informatizar y como resultado obtengan un negocio bien definido. El cliente se encontrará siempre acompañando al equipo de desarrollo para convenir los detalles de los requisitos y así poder implementarlos, probarlos y validarlos (Universidad de las Ciencias Informáticas, 2015).

1.7 Lenguajes, tecnologías y herramientas a utilizar en la implementación del subsistema de detección de similitud entre noticias

Para desarrollar el subsistema informático que se propone, se tuvieron en cuenta los lenguajes, tecnologías y herramientas a utilizar en correspondencia con la base tecnológica de la plataforma C.U.B.A (Martínez, 2016; Caraballo, 2017; Alvarez, 2018). A continuación, se procede con el estudio y análisis de las mismas.

1.7.1 Rastreador

Un rastreador, es un programa que sigue o rastrea enlaces a través de Internet, recopilando contenido de los sitios web y añadiéndolo a las bases de datos de los motores de búsqueda. Seguidamente se presenta el rastreador seleccionado (Estrada, 2012).

Nutch 1.9

Es un programa distribuido bajo la licencia Apache, desarrollado con el lenguaje de programación Java, altamente escalable y modular. Provee interfaces extensibles de parsing, indexación y filtros de selección por puntajes de gran ayuda para implementaciones personalizadas. Nutch se puede ejecutar en una sola máquina, pero gana mucho de su fuerza en un clúster Hadoop. Además, de los sistemas de rastreo es el que más se aproxima a la búsqueda semántica, debido a uno de sus plugins para ontologías (Estrada, 2012).

1.7.2 Indexador

En la actualidad existen disímiles herramientas catalogadas como indexadores de documentos, estas registran ordenadamente datos e información, para elaborar su índice, con la finalidad de obtener resultados de forma sustancialmente más rápida y relevante al momento de realizar una búsqueda (Estrada, 2012). A continuación, se presenta el sistema de indexación seleccionado:

Solr 4.10.3

Solr está escrito en Java y se ejecuta como un servidor de búsqueda de texto completo independiente dentro de un contenedor de *servlets*. Apache Solr es una plataforma de búsquedas basada en Apache Lucene, que funciona como un servidor de búsquedas. Sus principales características incluyen búsquedas de texto completo, resaltado de resultados, *clustering* dinámico, y manejo de documentos ricos (como Word y PDF). Es escalable, permitiendo realizar búsquedas distribuidas y replicación de índices. La principal característica de Solr es su API estilo REST, ya que en vez de usar drivers o APIs programáticas para la comunicación con Solr se pueden hacer peticiones HTTP y obtener resultados en XML (en español, Lenguaje de Marcado Extensible) o JSON (en español, Notación de Objetos de JavaScript). Posee un esquema de datos configurable y utiliza varios cachés para agilizar las búsquedas, como también proporciona diferentes funcionalidades como búsqueda y navegación por facetas, es decir, explorar la información desde diferentes perspectivas (Estrada, 2012).

Este indexador permite la configuración de la indexación y recuperación de documentos mediante ficheros de configuración XML. Añade una librería de analizadores textuales a los que provee por defecto Lucene e introduce el concepto de campo tipado, lo que permite introducir fechas y mejorar la ordenación.

1.7.3 Framework

Un marco de trabajo (*framework*) es un conjunto de componentes físicos y lógicos estructurados de manera que permiten ser reutilizados en el diseño y desarrollo de nuevos sistemas de información (Caraballo, 2017).

***Spring*:** es un marco de desarrollo o *framework* de código abierto creado para abordar la complejidad de desarrollo de las aplicaciones empresariales. Cualquier aplicación Java se puede beneficiar con *Spring* en términos de simplicidad, la capacidad de prueba, y bajo acoplamiento. Una ventaja de *Spring* es su modularidad, pudiendo usar algunos de los módulos sin comprometerse con el uso del resto (Caraballo, 2017).

***Spring Boot*:** es una herramienta que nace con la finalidad de simplificar aún más el desarrollo de aplicaciones basadas en el popular *framework Spring Core* (Caraballo, 2017).

Spring Boot centra su éxito en las siguientes características que lo hacen extremadamente fácil de utilizar:

- **Configuración:** *Spring Boot* cuenta con un complejo módulo que autoconfigura todos los aspectos de la aplicación para poder simplemente ejecutar la aplicación, sin tener que definir absolutamente nada.
- **Resolución de dependencias:** con *Spring Boot* solo hay que determinar qué tipo de proyecto utilizar y se encarga de resolver todas las librerías/dependencias para que la aplicación funcione.
- **Despliegue:** *Spring Boot* se puede ejecutar como una aplicación *Stand-alone (independiente)*, pero también es posible ejecutar aplicaciones web, ya que es posible desplegar las aplicaciones mediante un servidor web integrado, como es el caso de *Tomcat*, *Jetty* o *Undertow*.
- **Métricas:** por defecto, *Spring Boot* cuenta con servicios que permiten consultar el estado de salud de la aplicación, permitiendo saber si la aplicación está prendida o apagada, memoria utilizada y disponible, número y detalle de los *Bean's (objetos que manejan el contenedor de spring)* creados por la aplicación, controles para el prendido y apagado, etc.
- **Extensible:** *Spring Boot* permite la creación de complementos, los cuales ayudan a que la comunidad de Software Libre cree nuevos módulos que faciliten aún más el desarrollo.

1.7.4 Lenguajes de programación

Los lenguajes de programación pueden ser utilizados para crear programas que controlen el comportamiento físico y lógico de una máquina, para expresar algoritmos con precisión, o como modo de comunicación humana. Estos están conformados por reglas semánticas y sintácticas, que definen su estructura y el significado de sus expresiones. Permiten, además, especificar los datos que se deben procesar, almacenar o transmitir, y las acciones que se deben realizar bajo determinadas circunstancias (Caraballo, 2017).

El estudio de los lenguajes a utilizar en la confección de la propuesta de solución estará conformado en dos grupos: el primero enfocado al lenguaje de programación para el desarrollo de la propuesta de solución y un segundo grupo dedicado a los demás lenguajes a utilizar.

Java 1.8.0

Este lenguaje ofrece un entorno de aplicaciones avanzado, con un alto nivel de seguridad que es idóneo para las aplicaciones de red. Java proporciona portabilidad en una amplia gama de procesadores y

sistemas operativos integrados, y alcanza un alto rendimiento nativo. Este funciona con las principales plataformas de hardware y sistemas operativos, siendo uno de los entornos de programación más rápidos que incluye optimizaciones integradas para entornos multiproceso (Caraballo, 2017).

El modelo de Java para la gestión de la memoria, los procesos múltiples y la gestión de excepciones lo convierte en un lenguaje eficaz para los desarrolladores nuevos y para los más experimentados. Es una de las plataformas de aplicaciones más populares que existen y proporciona un interesante ecosistema de desarrolladores impulsado por herramientas eficaces, libros, bibliotecas y muestras de código.

1.7.5 Formato de intercambio de información

JSON 2.1

JSON es un formato ligero de intercambio de datos. Fácil de leer y escribir, como también es simple interpretarlo y generarlo. Está basado en un subconjunto del lenguaje de programación JavaScript. Es un formato de texto que es completamente independiente del lenguaje, pero utiliza convenciones que son ampliamente conocidos por los programadores de lenguajes C, C++, C#, Java, JavaScript, Perl, Python, y muchos otros. Estas propiedades hacen que JSON sea un lenguaje ideal para el intercambio de datos. JSON está constituido por dos estructuras: una colección de pares de nombre/valor. En varios lenguajes esto es conocido como un objeto, registro, estructura, diccionario, tabla hash, lista de claves o un arreglo asociativo. Y otra, una lista ordenada de valores. En la mayoría de los lenguajes, esto se implementa como arreglos, vectores, listas o secuencias (Caraballo, 2017).

1.7.6 Lenguaje de modelado

Lenguaje Unificado de Modelado 2.4.1

El Lenguaje Unificado de Modelado (UML, por sus siglas en inglés*, Unified Modeling Language*) está compuesto por una notación muy específica y por las reglas semánticas relacionadas para la construcción de sistemas de software (Caraballo, 2017).

Describe la notación para clases, componentes, nodos, actividades, flujos de trabajo, casos de uso, objetos, estados y cómo modelar la relación entre esos elementos, es decir, es utilizado para visualizar,

especificar, construir y documentar los artefactos de un sistema y, además, sirve para el modelado del negocio y sistemas de software. También soporta la idea de extensiones personalizadas a través de elementos estereotipados. Provee beneficios significativos al facilitar la construcción de modelos rigurosos, trazables y mantenibles, que soporten el ciclo de vida de desarrollo de software completo.

Se escoge esta versión porque facilita la comunicación, es fácil de entender y utilizar, su uso óptimo se consigue en procesos dirigidos por casos de uso y cubre las diferentes vistas de la arquitectura del subsistema.

1.8 Entorno de desarrollo

1.8.1 Entorno de Desarrollo Integrado Netbeans

Es un entorno de desarrollo integrado (IDE por sus siglas en inglés *Integrated Development Environment*) y una plataforma de desarrollo. Aunque inicialmente, *NetBeans IDE* sólo podía ser utilizado para desarrollar aplicaciones *Java*, a partir de la versión 6, *NetBeans* soporta varios lenguajes de programación, ya sea a través de una función de apoyo, o mediante la instalación de complementos adicionales, algunos de estos lenguajes son *Java*, C, C++, PHP, HTML, *JavaScript*, y *Scala* (Heffelfinger, 2015). *NetBeans* proporciona funcionalidades adicionales como la compilación en tiempo real, la comprobación de tipos, refactorización, navegadores de clase y soluciones rápidas para los errores en tiempo de compilación (Caraballo, 2017).

1.8.2 Gestor de base de datos

Debido a que el sistema cuenta con una serie de datos que deben ser bien administrados y guardados, un sistema gestor de base de datos sirve para lograr un mejor manejo de los datos y su definición, así como mantener la integridad de los datos dentro de la base de datos, controlar la seguridad, la privacidad y el manejo de los datos.

MySQL

Es la base de datos de código abierto más popular del mundo. Con su rendimiento, confiabilidad y facilidad de uso comprobados, MySQL se ha convertido en la opción de base de datos líder para

aplicaciones basadas en web, utilizadas por propiedades web de alto perfil como Facebook, Twitter, 40 YouTube, Yahoo! Y muchos más. Oracle impulsa la innovación de MySQL y ofrece nuevas capacidades para potenciar la próxima generación de aplicaciones web, en la nube, móviles e integradas (MySQL, 2018).

Visual Paradigm para UML 8.0

Se usa Visual Paradigm for UML teniendo en cuenta que soporta el modelado mediante UML y proporciona asistencia a ingenieros de software y desarrolladores, durante todos los pasos del ciclo de vida de desarrollo de un software. Permite dibujar todo tipo de diagrama de clases, generar código fuente a partir de diagramas y generar documentación. Permite integrarse con otras aplicaciones, como herramientas ofimáticas, lo cual aumenta la productividad (Visual Paradigm, 2014).

Considerando todas las facilidades brindadas por esta herramienta, que se puede adquirir mediante licencia gratuita y comercial, que es fácil de instalar y actualizar, además de que existe una amplia experiencia en la Universidad en el uso de esta herramienta en los proyectos productivos, se puede concluir que constituye la mejor opción para realizar el modelado de diagramas de este trabajo según la metodología propuesta.

Postman

Programa gratuito que permite probar servicios web fácilmente, basta con indicar la URL, el método HTTP (*POST, GET*, etc.) y los parámetros de la petición (Caraballo, 2017).

1.9 Conclusiones parciales

Del estudio y análisis realizado de la plataforma C.U.B.A., de las soluciones informáticas precedentes y de la literatura científica, sobre la detección de similitud entre textos, se obtuvieron las ventajas y desventajas de este tipo de herramientas, así como las principales características de los algoritmos empleados. Se selecciona la función coseno junto a un algoritmo de detección de similitud de textos para obtener los textos similares, asociado a la propia función.

Con el uso de la metodología AUP en una versión adaptada por la UCI, el lenguaje de modelado UML, la herramienta de modelado Visual Paradigm, el lenguaje de programación Java, el marco de trabajo Spring MVC, el gestor de base de datos MySQL y la herramienta de desarrollo Netbeans con la que se pretende desarrollar la propuesta de solución.

El subsistema se desarrollará con herramientas que están distribuidas bajo licencias de software libre en correspondencia con el principio de independencia tecnológica en el cual está basado actualmente el desarrollo de software en Cuba, lo que contribuye a que el subsistema esté libre de costo alguno.

Capítulo 2: Análisis y diseño del subsistema de detección de similitud entre noticias

2.1 Introducción

En el capítulo se detallan las características del Subsistema de detección de similitud entre noticias según plantea la metodología AUP-UCI en su escenario 4, se presenta la propuesta de solución a desarrollar. Se especifican las funcionalidades del subsistema con la descripción de los requisitos funcionales, no funcionales y las historias de usuario como artefacto principal generado. Se describe la arquitectura a utilizar, el diagrama de clases con los patrones de diseño utilizados y el diagrama de despliegue del subsistema.

3.2.1 Descripción de la propuesta de solución

El proceso del subsistema de detección de similitud comienza con la extracción de los documentos (noticias) del servidor de indexación Apache Solr, el subsistema prosigue con un algoritmo de similitud que retorna la relevancia de las noticias similares obtenidas. Se crean dos nuevos campos con el nombre "***News***" y "***NewsSimilarity***", en los que se guardarán las noticias y las noticias relevantes a estas respectivamente.

3.2.2 Modelado de negocio

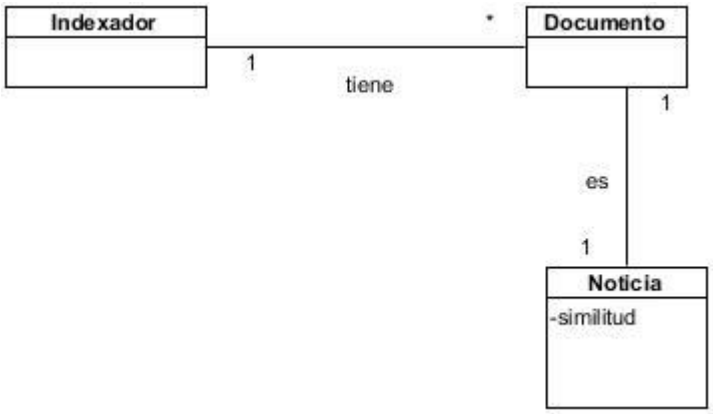

Ilustración 1:Modelo de dominio. Elaboración propia.

Descripción de los conceptos del dominio (se muestran en la Figura 1).

Indexador: es la herramienta que se encarga de indexar los documentos recuperados de la Web.

Documento: documento recuperado de la Web.

Noticia: documento evaluado**.**

2.3 Requisitos

Los requisitos del software permiten establecer lo que el sistema debe hacer, sus características fundamentales y las restricciones en el funcionamiento del sistema y los procesos de desarrollo del software. De manera general, estos requisitos expresan las necesidades objetivas que presentan los usuarios, ante un sistema que resuelve un problema en particular de un determinado dominio (Sommerville, 2005). Estas cualidades se clasifican en:

2.3.1 Requisitos funcionales

Describen lo que el sistema debe realizar (Sommerville, 2005).

La siguiente Tabla (2) muestra un listado de los requisitos funcionales (RF) y sus correspondientes prioridad y complejidad (alta, media y baja), con el objetivo de esclarecer la comunicación entre usuario y software:

Tabla 2: Listado de requisitos funcionales. Elaboración propia.

No.	Nombre	Descripción	Prioridad	Complejidad
RF1	Desarrollar una API REST para insertar noticias.	Permite mediante API que nutch inserte noticias en el sistema	Alta	Alta
RF2	Obtener noticias de la API REST de Solr.	Permite consultar los documentos indexados en el servidor obteniéndose los valores de los	Media	Media

		campos URL, contenido y título.		
RF3	Almacenar las noticias.	Guarda las noticias en la base de datos del subsistema en la tabla news.	Media	Media
RF4	Obtener las noticias no evaluadas.	Obtiene todas aquellas noticias de la base de datos del subsistema en la tabla noticias que no hayan sido evaluadas.	Media	Media
RF5	Calcular la similitud de las noticias no evaluadas.	Se aplica el algoritmo de detección de similitud seleccionado, para calcular el por ciento de similitud entre las noticias.	Alta	Alta
RF6	Almacenar similitudes calculadas.	Guarda las similitudes calculadas en la base de datos del subsistema en la tabla similitudes	Media	Media
RF7	Obtener noticias similares.	Dado una noticia se obtienen las noticias similares a ella.	Media	Media

Véase en Anexos, Figuras 9 y 10 que ofrecen capturas de pantallas que demuestran la aplicación del algoritmo de detección de similitud seleccionado y los resultados de las similitudes entre noticias luego de calculadas, en correspondencia con los RF descritos.

2.3.2 Requisitos no funcionales

Son aquellos que no se refieren a las funcionalidades específicas que proporciona el sistema, sino a las propiedades emergentes de éste como fiabilidad, el tiempo de respuesta y la capacidad de almacenamiento, entre otros (Sommerville, 2005). Se pueden apreciar en la Tabla 3, con sus nombres y atributos.

Tabla 3: Listado de requisitos no funcionales. (RNF). Elaboración propia.

No. RNF	Nombre	Atributo
1	Emplear Java como lenguaje de programación.	Software
2	Los servidores donde se instalarán cada uno de los componentes del subsistema del sistema seguirán una distribución GNU/Linux.	Software
3	La aplicación se ejecutará en cualquier Sistema Operativo (SO) que soporte las librerías de Java 8.	Funcionalidad
4	Requisitos mínimos en el servidor: 2 Gb de memoria RAM y CPU con 2 *cores* de procesamiento a 2.5 GHz de velocidad.	Hardware
5	El subsistema no debe requerir de información confidencial.	Seguridad
6	La información administrada por el subsistema debe de estar protegida de accesos no autorizados.	Seguridad
7	El subsistema debe ser capaz de responder con efectividad, en dependencia del tipo de petición y los datos que son manejados.	Rendimiento
8	Debe permitir su actualización y mantenimiento.	Soporte
9	Un usuario autorizado y con experiencia podrá interactuar con el componente, por lo que deberá ser intuitivo.	Usabilidad

2.4 Historias de usuario

Las Historias de Usuario (HU) son un enfoque de requerimiento ágil que se focaliza en establecer conversaciones sobre las necesidades de los clientes. Son descripciones cortas y simples de las funcionalidades del sistema, narradas desde la perspectiva de la persona que desea implementar cada funcionalidad (Izaurralde, 2013). Las Tablas de la 4 a la 10 muestran las HU correspondientes a cada RF definido para el susbsistema.

Tabla 4: HU Desarrollar un API REST para insertar noticias. Elaboración propia.

Número: 1	**Nombre del requisito:** Desarrollar un API REST para insertar noticias
Programador: Williams Celestino Arrieta Perez	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo:** 48 horas
Riesgo: Alto	**Tiempo real:** 40 horas
Descripción: Permite mediante API que nutch inserte noticias en el sistema	
Observaciones:	
Prototipo de Interfaz: No aplica.	

Tabla 5: HU Obtener noticias de la API REST de Solr. Elaboración propia.

Número: 2	**Nombre de requisito:** Obtener noticias de la API REST de Solr
Programador: Williams Celestino Arrieta Perez	**Iteración Asignada:** 1

Prioridad: Media	**Tiempo:** 24 horas
Riesgo: Alto	**Tiempo real:** 16 horas
Descripción: Permite consultar los documentos indexados en el servidor obteniéndose los valores de los campos URL, contenido y título.	
Observaciones:	
Prototipo de Interfaz: No aplica.	

Tabla 6: HU Almacenar las noticias. Elaboración propia.

Número: 3	**Nombre de requisito:** Almacenar las noticias
Programador: Williams Celestino Arrieta Perez	**Iteración Asignada:** 1
Prioridad: Media	**Tiempo:** 32 horas
Riesgo: Alto	**Tiempo real:** 28 horas
Descripción: Guarda las noticias en la base de datos del subsistema en la tabla noticias.	
Observaciones:	
Prototipo de Interfaz: No aplica.	

Tabla 7: Obtener las noticias no evaluadas. Elaboración propia.

Número: 4	**Nombre de requisito**: Obtener las noticias no evaluadas
Programador: Williams Celestino Arrieta Perez	**Iteración Asignada:** 1
Prioridad: Media	**Tiempo:** 24 horas
Riesgo: Alto	**Tiempo real:** 16 horas
Descripción: Obtiene todas aquellas noticias de la base de datos del subsistema en la tabla noticias que no hayan sido evaluadas.	
Observaciones:	
Prototipo de Interfaz: No aplica.	

Tabla 8: HU Calcular la similitud de las noticias no evaluadas. Elaboración propia.

Número: 5	**Nombre de requisito:** Calcular la similitud de las noticias no evaluadas
Programador: Williams Celestino Arrieta Perez	**Iteración Asignada:** 2
Prioridad: Alta	**Tiempo:** 96 horas

Riesgo: Alto	**Tiempo real:** 96 horas
Descripción: Se aplica el algoritmo de detección de similitud seleccionado para calcular el por ciento de similitud entre las noticias.	
Observaciones:	
Prototipo de Interfaz: No aplica.	

Tabla 9: HU Almacenar similitudes calculadas. Elaboración propia.

Número: 6	**Nombre de requisito:** Almacenar similitudes calculadas
Programador: Williams Celestino Arrieta Perez	**Iteración Asignada:** 3
Prioridad: Media	**Tiempo:** 32 horas
Riesgo: Alto	**Tiempo real:** 28 horas
Descripción: Guarda las similitudes calculadas en la base de datos del subsistema en la tabla similitudes.	
Observaciones:	
Prototipo de Interfaz: No aplica.	

Tabla 10: HU Obtener noticias similares. Elaboración propia.

Número: 7	**Nombre de requisito:** Obtener noticias similares
Programador Williams Celestino Arrieta Perez	**Iteración Asignada:** 3
Prioridad: Media	**Tiempo:** 24 horas
Riesgo: Alto	**Tiempo real:** 16 horas
Descripción: Dado una noticia se obtienen las noticias similares a ella.	
Observaciones:	
Prototipo de Interfaz: No aplica.	

2.5 Modelado del diseño

Un diagrama de clases del diseño describe gráficamente las especificaciones de las clases de software y de las interfaces en una aplicación. Un diagrama de este tipo contiene las definiciones de las entidades del software (Larman, 2004). Por tal motivo, se decide utilizar este tipo de diagrama para la representación gráfica de las clases y sus relaciones. Se muestra a continuación en la Figura 2.

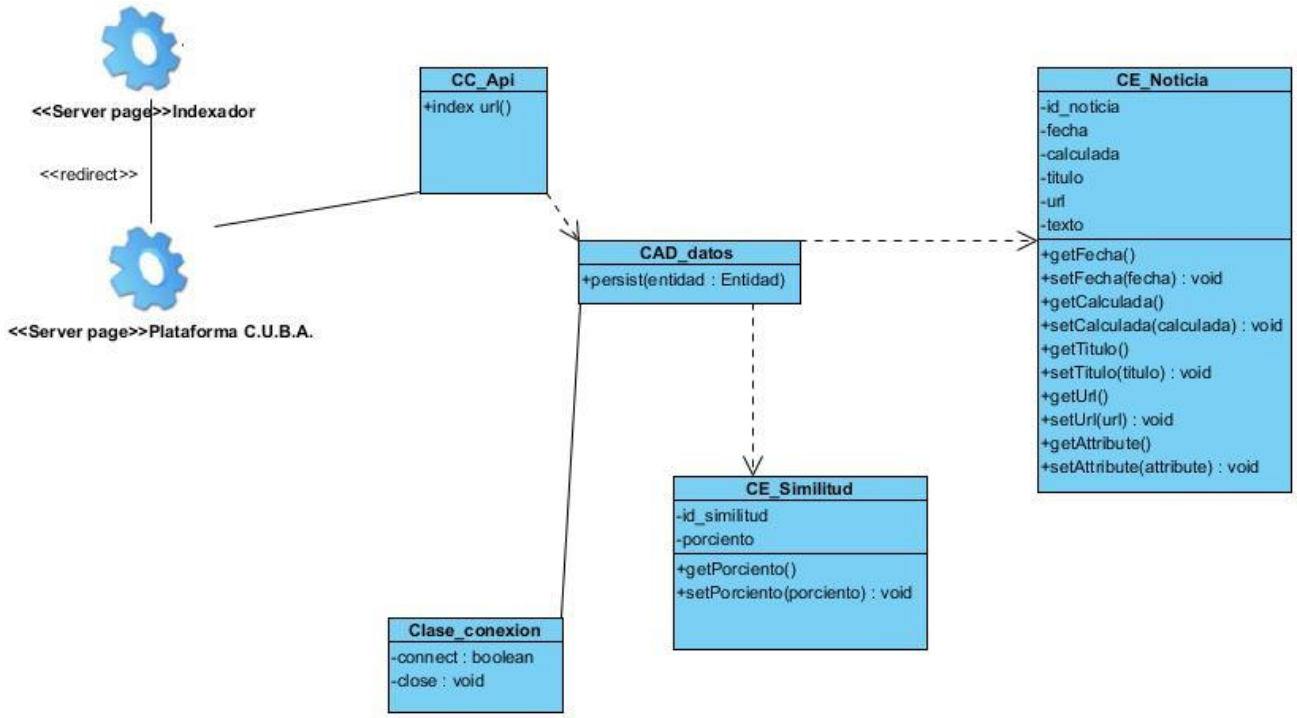

Ilustración 2: Diagrama de clases del diseño con estereotipos web. Elaboración propia.

2.6 Patrones

Según Larman (2004), un patrón es una descripción de un problema y su solución, que recibe un nombre y que puede emplearse en otros contextos. También puede ser considerado como una pareja de problema/solución con una sugerencia sobre la manera de utilizarlo en situaciones nuevas.

Experto en información

La solución que propone este patrón es la de asignar una responsabilidad a la clase (experto) que cuenta con la información necesaria para cumplirla. Ofrece una analogía con el mundo real ya que da origen al diseño donde el objeto de software realiza las operaciones que normalmente se aplican al elemento real que representa (Larman, 2004). La clase ***similarityScheduled*** es un ejemplo de Experto, encargada del pre-procesamiento del contenido de cada documento y es experta en esa función.

Creador

Guía la asignación de responsabilidades relacionadas con la creación de objetos. Una de las consecuencias de usar este patrón es la visibilidad entre la clase creada y la clase creador. Una ventaja es el bajo acoplamiento, lo cual supone facilidad de mantenimiento y reutilización. La creación de instancias es una de las actividades más comunes en un sistema orientado a objetos. En consecuencia, es útil contar con un principio general para la asignación de las responsabilidades de creación. Si se asignan bien, el diseño puede soportar un bajo acoplamiento, mayor claridad, encapsulación y reutilización (Larman, 2004). En el subsistema de detección de similitud entre textos se encuentra la clase ***newNews***, responsable de crear y cargar los objetos necesarios para el resto de las clases existentes.

Alta Cohesión

Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme. Significa que las clases del sistema tienen asignadas solo las responsabilidades que les corresponde y mantienen una estrecha relación con el resto de las clases (Larman, 2004). En el subsistema es utilizado este patrón en las clases ***newNews***, ***getLastNews*** y ***similarityScheduled***.

Bajo Acoplamiento

Determina el nivel de dependencia de una clase con respecto a otras. Una clase con bajo acoplamiento no depende de muchas otras. En los lenguajes orientados a objetos como C++, Java y Smalltalk una de las formas más comunes de acoplamiento de TipoX a TipoY es cuando TipoY es una interfaz y TipoX la implementa (Larman, 2004). Las clases ***newNews*** *y* ***getLastNews*** no tienen relaciones entre ellas, solo se comunican con la clase ***similarityScheduled***, y pueden ser reutilizadas como subsistemas para otros sistemas sin que ocurran grandes impactos por los cambios.

2.7 Modelado de datos

Orientado a presentar los elementos que procesan el sistema, la composición y atributos de los mismos, muestra dónde se encontrarán almacenados dichos objetos, la relación entre ellos y los procesos que los transforman (Pressman, 2010). A continuación, se muestra el modelo de datos con que será implementado el subsistema para la detección de similitudes entre noticias en la Figura 3.

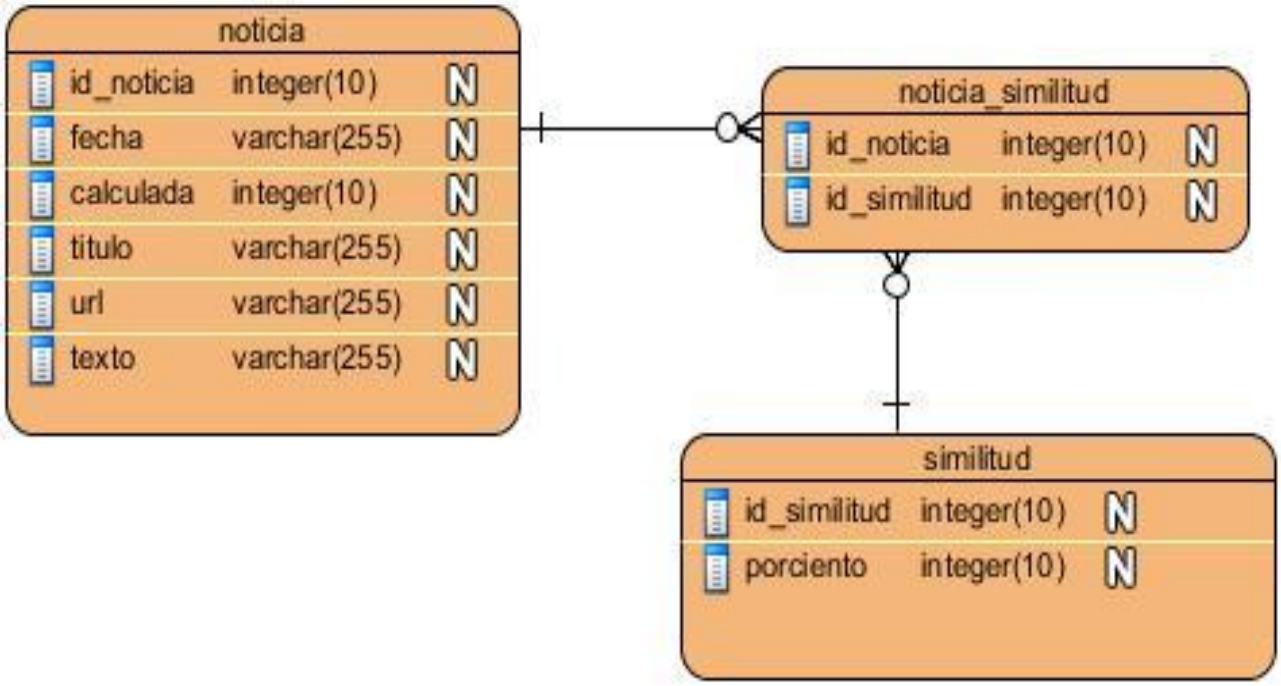

Ilustración 3: Diagrama Entidad-Relación. Elaboración propia.

2.8 Diseño arquitectónico

La programación por capas es un estilo de programación en el que el objetivo primordial es la separación de la lógica de negocios de la lógica de diseño. La ventaja principal de este estilo es que el desarrollo se puede llevar a cabo en varios niveles y, en caso de que sobrevenga algún cambio, sólo se atacará al nivel requerido sin tener que revisar entre código mezclado.

Capa de negocio: es donde residen los programas que se ejecutan, se reciben las peticiones del usuario y se envían las respuestas tras el proceso. Se denomina capa de negocio (e incluso de lógica del negocio) porque es aquí donde se establecen todas las reglas que deben cumplirse. Esta capa se comunica con la capa de presentación, para recibir las solicitudes y presentar los resultados, y con la capa de datos, para solicitar al gestor de base de datos almacenar o recuperar datos de él. También se consideran aquí los programas de aplicación.

Capa de datos: es donde residen los datos y es la encargada de acceder a los mismos. Está formada por uno o más gestores de bases de datos que realizan todo el almacenamiento de datos, reciben solicitudes de almacenamiento o recuperación de información desde la capa de negocio.

La **arquitectura de microservicios** (en inglés, *Micro Services Architecture*, MSA) es una aproximación para el desarrollo de *software* que consiste en construir una aplicación como un conjunto de pequeños servicios, los cuales se ejecutan en su propio proceso y se comunican con mecanismos ligeros (normalmente una API de recursos HTTP). Cada servicio se encarga de implementar una funcionalidad completa del negocio. Cada servicio es desplegado de forma independiente y puede estar programado en distintos lenguajes y usar diferentes tecnologías de almacenamiento de datos.

2.9 Conclusiones parciales

Con el estudio de los temas referentes al análisis y diseño de la propuesta del subsistema para la detección de similitud entre noticias se pudo arribar a las siguientes conclusiones:

- Los requisitos funcionales y no funcionales obtenidos a partir del levantamiento de requisitos responden a que la propuesta de solución satisface las necesidades del cliente.
- Las descripciones de las historias de usuario y la elaboración de los diagramas de clases del diseño posibilitaron una mejor comprensión del funcionamiento de la propuesta de solución.
- Adoptar la arquitectura de software n-capas permitió una propicia organización del sistema a implementar. Los artefactos generados según la metodología AUP-UCI, la arquitectura y los patrones de diseño crearon las bases necesarias para la construcción de la propuesta de solución.

Capítulo 3: Implementación, pruebas y validación del subsistema para la detección de similitud entre noticias para la plataforma C.U.B.A.

3.1 Introducción

En el presente capítulo se caracterizan los elementos definidos en el proceso de desarrollo, para obtener un producto de calidad. Es presentado y descrito el diagrama de componentes para una mayor comprensión de los elementos físicos del sistema y sus relaciones. Se incluye la especificación del estándar de codificación utilizado para la implementación del subsistema. Se ejecutan las pruebas del software, con el objetivo de verificar el correcto funcionamiento del subsistema y así validar la propuesta de solución.

3.2 Implementación

3.2.1 Estándares de implementación (o codificación)

Los estándares de codificación son un conjunto de reglas o patrones de codificación a seguir por los desarrolladores con el objetivo de establecer un orden y un formato común en el código fuente del sistema en desarrollo. Estos estándares cumplen con el principio de legibilidad y mantenibilidad, correspondientes al objetivo de que el programador entienda el código y sea capaz de modificarlo (Microsoft, 2016; Microsoft, 2020). En la implementación del componente es utilizada la siguiente estandarización:

- ✓ Se utiliza el margen adicional de cuatro espacios.
- ✓ Las variables son declaradas de forma independiente, en líneas nuevas y no en las mismas sentencias.

Pueden apreciarse fragmentos del código fuente implementado en las Figuras de la 4 a la 6.

```
@RequestMapping(value = "/getLastNews", method = RequestMethod.GET)
public ResponseEntity<List<NewsDto>> getLastNews() {
    List<News> topById = newsRepository.findTop10ByOrderByDate();
    List<NewsDto> newsDtos = new ArrayList<>();
    topById.forEach(news -> {
        NewsDto newsDto = new NewsDto();
        newsDto.setBody(news.getBody());
        newsDto.setTitle(news.getTitle());
        newsDto.setUrl(news.getUrl());
        newsDtos.add(newsDto);
    });
    return ResponseEntity.ok(newsDtos);
}
```

Ilustración 4: Fragmento del código fuente de la clase similarityScheduled.

Las implementaciones de las clases tienen la siguiente estructura:

- ✓ Atributos de la clase.
- ✓ Constructores.
- ✓ Métodos de acceso.
- ✓ Métodos.

```java
@Component
public class Schedules {

    @Autowired
    NewsRepository newsRepository;

    @Autowired
    NewsSimilarityRepository newsSimilarityRepository;

    private static final Logger logger = Logger.getLogger(Application.class.getName());

    @Scheduled(fixedDelay = 3000)
    public void similarityScheduled() {

        logger.info( msg: "UPDATE SIMILARITY NEWS..");
        //Se obtienen las noticias que no se le han calculado la similitud
        List<News> byEvaluatedIsFalse = newsRepository.findByEvaluatedIsFalse();
        Calendar calendar = Calendar.getInstance();
        calendar.add(Calendar.DAY_OF_MONTH, amount: -10);
        byEvaluatedIsFalse.forEach(news -> {
```

Ilustración 5: Fragmento del código fuente de la clase similarityScheduled

- ✓ Los atributos relacionados se declaran en una misma línea. El resto se declara en líneas separadas.
- ✓ Los comentarios se declaran con una gramática correcta y contienen los signos de puntuación apropiados.

```
logger.info( msg: "UPDATE SIMILARITY NEWS..");
//Se obtienen las noticias que no se le han calculado la similitud
List<News> byEvaluatedIsFalse = newsRepository.findByEvaluatedIsFalse();
Calendar calendar = Calendar.getInstance();
calendar.add(Calendar.DAY_OF_MONTH,  amount: -10);
byEvaluatedIsFalse.forEach(news -> {

    //Se obtienen las x últimas noticias
    List<News> newsByDateAfter = newsRepository.findNewsByDateAfter(calendar.getTime());
    //Se calcula la similitud de la noticia inicial con la nueva noticia
    newsByDateAfter.forEach(news1 -> {
```

Ilustración 6: Fragmento del código fuente de la clase *similarityScheduled*

La utilización de los estándares de codificación garantizó que el código resultante fuera de sencilla comprensión, independientemente del desarrollador del producto. Permitió asegurar calidad y legibilidad.

3.3 Diagrama de componentes

El diagrama de componentes permite visualizar la estructura de un sistema informático mediante el análisis de las partes que lo conforman. Los componentes son utilizados como una unidad de composición independiente e indispensable dentro de un sistema, donde se identifica las dependencias que existen entre cada uno de los elementos. Algunos ejemplos de componentes físicos lo constituyen los archivos, módulos, librerías, ejecutables y binarios. (Sommerville, 2005).

Véase a continuación el diagrama de componentes en la Figura 7.

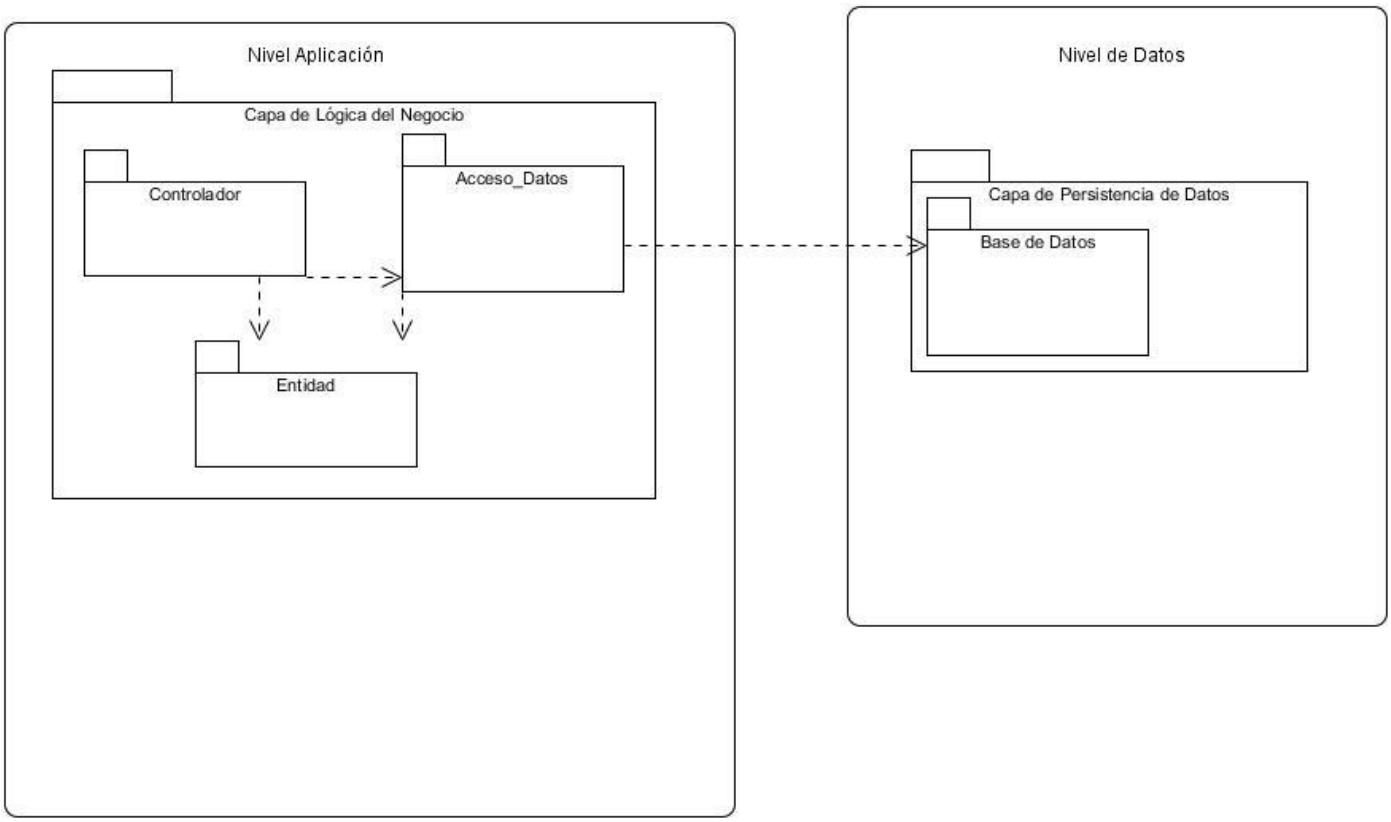

Ilustración 7: Diagrama de componentes. Elaboración propia

3.4 Diagrama de despliegue

Un modelo de despliegue representa la relación física que se establece entre los distintos componentes o nodos que describen la topología de un sistema. Estos definen la configuración de los elementos de hardware, y detallan cómo los elementos y artefactos del software se relacionan en esos nodos (SparxSystems, 2018).

Se muestra seguidamente el diagrama de despliegue en la Figura 8.

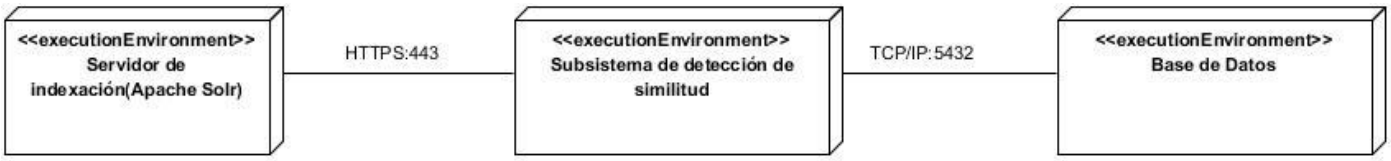

Ilustración 8: Diagrama de despliegue. Elaboración propia.

La Figura muestra el nodo **Apache Solr** como servidor de indexación; esta herramienta recibe consultas del nodo **subsistema para la detección de similitud de textos** mediante el protocolo HTTP por el puerto 443, la aplicación a su vez intercambia información con el **servidor de base de datos** por HTTP en el puerto 5432.

3.5 Pruebas de software

Según Navarro y Pérez (2014) las pruebas de software comprenden una fase del proceso de desarrollo que se centra en asegurar la calidad, fiabilidad y robustez de un software, dentro de un contexto o escenario donde está previsto que este sea utilizado. Se encuentra encaminado a medir el cumplimiento de las funcionalidades establecidas por el cliente, reduciendo de esta manera el número de errores no detectados. Esta fase es una de las más costosas del ciclo de vida del software.

Tipos de pruebas

Pruebas de integración

Las pruebas de integración se centran en comprobar que los módulos probados por separado funcionen en conjunto, con el objetivo de verificar que interactúan correctamente a través de sus interfaces y cubren las funcionalidades establecidas en los requisitos (Pressman, 2010). En la Tabla 12 se muestran los resultados de la prueba de integración.

Tabla 11: Resultados de la prueba de integración.

Componente	Funcionalidad	Funcionalidad Integridad	Resultado de la prueba
Servidor	Obtener noticias de	Se realiza la extracción de	Fueron encontradas en la

Apache Solr	tipo texto almacenadas en Apache Solr.	documentos desde el servidor de indexación hacia el componente.	primera iteración dos no conformidades, la primera, relacionada a una mala referencia entre las clases *similarityScheduled* y Conectar_Solr y la segunda, a falta de librerías, ambas fueron corregidas. Fue realizada una segunda iteración donde no fueron encontradas deficiencias.
	Agregar y actualizar la información en Apache Solr.	Permite agregar y actualizar las noticias con las noticias similares a estas en el servidor de indexación.	El subsistema inserta y actualiza, de forma satisfactoria, la información en Apache Solr.

Las ejecuciones de las pruebas de integración permitieron verificar el trabajo en conjunto del componente con los servidores. Se realizó especial énfasis en la comunicación entre el componente y Apache Solr, con el objetivo de detectar incoherencias en el funcionamiento de la aplicación. El análisis de los resultados obtenidos de las pruebas concluye con la solución de las incoherencias encontradas y la correcta integración con los servidores, necesarios para la detección de similitud.

Pruebas funcionales

Las pruebas funcionales se centran en comprobar que los sistemas desarrollados funcionan acorde a las especificaciones funcionales y requisitos del cliente. Este servicio ayuda a detectar los posibles defectos derivados de errores en la fase de programación (Globe, 2017).

En las Tablas que se muestran a continuación, de la 13 a la 18, aparecen los casos de prueba para las HU del subsistema implementado.

Tabla 12: Caso de prueba a la HU "*Desarrollar una API REST para insertar noticias*"

Escenario	Descripción	V1	V2	R/ del subsistema	Flujo Central
EC 1.1 Desarrollar una API REST para insertar noticias	Permite mediante una API que nutch inserte noticias en el servidor de indexación Apache Solr.	V Noticia 1	V Noticia 2	nutch almacena las noticias en el servidor de indexación.	nutch analiza e identifica la noticia en un documento mediante la API y lo almacena en el campo *"Noticias"* del servidor de indexación Apache Solr.

Tabla 13: Caso de prueba a la HU "Obtener noticias de la API REST de Solr"

Escenario	Descripción	V1	V2	R/ del subsistema	Flujo Central
EC 2.1 Obtener noticias de la API REST de Solr	El subsistema de detección de similitud consulta las noticias indexadas en el servidor obteniéndose los valores de los campos URL, contenido y título.	V Noticia 1	V Noticia 2	El subsistema consulta las noticias indexadas en el servidor.	El subsistema de detección de similitud consulta las noticias indexadas de la API REST de Solr obteniéndose los valores de los campos URL, contenido y título.

Tabla 14: Caso de prueba a la HU "Almacenar las noticias"

Escenario	Descripción	V1	R/ del subsistema	Flujo Central

EC 3.1 Almacenar las noticias.	El subsistema de detección de similitud almacena las noticias en la base de datos del subsistema.	V Noticias	El subsistema almacena las noticias en la base de datos.	El subsistema almacena en el campo *"News" de la base de datos del mismo* las noticias correspondientes.

Tabla 15: Caso de prueba a la HU "Obtener las noticias no evaluadas"

Escenario	Descripción	V1	R/ del subsistema	Flujo Central
EC 4.1 Obtener las noticias no evaluadas	Obtiene todas aquellas noticias de la base de datos del subsistema en la tabla noticias que no hayan sido evaluadas.	V Noticias	El subsistema consulta las noticias almacenadas en la base de datos del subsistema que no han sido evaluadas.	El subsistema de detección de similitud consulta las noticias no evaluadas de la base de datos almacenadas en la tabla "news".

Tabla 16: Caso de prueba a la HU "Calcular la similitud de las noticias no evaluadas"

Escenario	Descripción	V1	V2	R/ del subsistema	Flujo Central
EC 5.1	El subsistema de	V	V	El subsistema	El subsistema calcula la

Calcular la similitud de las noticias no evaluadas.	detección de similitud entre noticias analiza la similitud entre las noticias no evaluadas y las noticias presentes en la base de datos del mismo a través de la utilización de una función de similitud.	Noticia 1	Noticias	retorna la mayor similitud entre las noticias no evaluadas y las noticias presentes en la base de datos del mismo mediante la utilización de una función de similitud.	relación entre las noticias no evaluadas y las noticias presentes en la base de datos del mismo mediante el uso de una función de similitud. Se realizó dos iteraciones, la primera con la función de Levenshtein y la segunda con el Coseno, se obtuvo como resultado 70% y 95% de concordancia para cada caso, respectivamente. La prueba demostró la utilización de función Coseno para la similitud de términos en la aplicación.

Tabla 17: Caso de prueba a la HU "Almacenar similitudes calculadas"

Escenario	**Descripción**	**V1**	**R/ del subsistema**	**Flujo Central**
EC 6.1 Almacenar similitudes calculadas.	El subsistema de detección de similitud almacena las noticias evaluadas en su base de datos.	V Noticias	El subsistema almacena las noticias en la base de datos.	El subsistema almacena en la tabla *"NewsSimilarity"* de la base de datos del mismo las noticias evaluadas.

En Anexos (Figuras 9 y 10) se pueden apreciar capturas de pantalla del principal proceso de la solución, cálculo de similitud aplicando el algoritmo seleccionado durante el desarrollo de la investigación, asociado a la función coseno. En la primera captura se muestran un grupo de noticias, ya recuperadas de Solr o adicionadas mediante el API REST de la aplicación y en la segunda se puede apreciar la ejecución del algoritmo donde detectó que la noticia con identificador 1 y la noticia con identificador 2 son similares y poseen un índice de similitud de 0.92, así como la noticia con identificadores 5 y 4 con un índice de similitud, 1.0.

Como se aprecia en dichas Figuras, el subsistema implementado recibe un grupo de noticias y una tarea procesa estas noticias en segundo plano calculando el índice de similitud de las nuevas noticias, con las ya existentes en la base de datos. Esto permite que dentro del buscador se puedan utilizar estos índices de similitud, para la búsqueda o sugerencias de noticias a los usuarios que utilicen la plataforma, al mostrarle los resultados de la recuperación de información relacionados con la temática de la noticia que busca. La aplicación desarrollada cuenta con API REST la cual podrá ser consultada desde el buscador.

3.6 Evaluación de la hipótesis de la investigación

Sobre la validación de la hipótesis, no fue posible llegar al final de este importante aspecto por las afectaciones de presencialidad y accesibilidad causadas por la COVID-19, se propone realizarla en las recomendaciones de esta investigación.

Para la validación de la hipótesis científica se propone utilizar el método de consulta a expertos en su variante Delphi (Sánchez, 2015) siguiendo los puntos siguientes:

- Identificación de los posibles expertos.
- Selección de los expertos.
- Realización de consultas a expertos, procesamiento y valoración de la información obtenida.

3.7 Conclusiones parciales

En este capítulo se abordaron aspectos correspondientes a la implementación y validación del subsistema de detección de similitud entre noticias, arribándose a las siguientes conclusiones:

- El diagrama de componentes facilitó la comprensión de la estructura general de la aplicación.

- Al aplicar los estándares de codificación se logró adoptar una estructura homogénea que facilita la comunicación y asegura la calidad, menos errores y fácil mantenimiento.
- La aplicación de pruebas funcionales e integración permitieron identificar las principales deficiencias en el desarrollo del subsistema en etapas tempranas, así como solucionar los errores detectados y obtener un producto con un alto valor, pertinencia y utilidad.
- Se recomienda aplicar método de expertos para la validación de la hipótesis de la investigación.

Conclusiones

Finalizada la presente investigación, se arriba a las siguientes conclusiones:

- Con la valoración del estado del arte de los sistemas de detección de similitud de documentos, se garantizó una mayor compresión del objeto de estudio, con énfasis en la implementación de la función de similitud coseno para mejorar la recuperación de la información.
- El enfoque propuesto por la metodología AUP en su variación para la universidad y la utilización de Java como lenguaje de programación, Netbeans como IDE de programación, Visual Paradigm para la modelación y Apache Solr para la indexación de los documentos, permitieron analizar los procesos que debían implementarse, estableciéndose una correcta correspondencia entre las especificaciones del cliente y las características que debía presentar el subsistema.
- El diseño e implementación de las clases del subsistema permitieron cumplir con los requisitos funcionales. Los requisitos no funcionales evidenciaron claramente las cualidades del producto. La modelación de los artefactos permitió obtener una arquitectura sólida en la aplicación y garantizó la base en la organización lógica del código fuente.
- Las pruebas funcionales y de integración aplicadas al subsistema erradicaron las insuficiencias detectadas en la aplicación, demostraron que cumple con las necesidades de los usuarios finales y es una solución de calidad.
- El subsistema desarrollado para la detección de similitud entre noticias mostró en las pruebas de software un correcto funcionamiento y nivel de integración, lo que constituye un paso importante en el propósito de elevar la efectividad de las peticiones realizadas por los usuarios en la sección de noticias de la plataforma C.U.B.A.

Recomendaciones

Una vez concluida la investigación y el desarrollo de la propuesta de solución, el autor del presente trabajo recomienda:

- Añadir más funcionalidades con el objetivo de mejorar la recuperación de la información.
- Validar la hipótesis de la investigación con el método de consulta a expertos en su variante Delphi.

Bibliografía

AGUIRRE, Yurisel. *Sistema para la detección de plagio y validación de estructura en los documentos científicos emitidos en el Centro de Información Científico Técnica (CICT) del ISMMM*. Revista Caribeña de Ciencias Sociales, Servicios Académicos Intercontinentales SL, 2012. No.12, December.

ALVAREZ, David. *Componente para la categorización semántica de documentos*. Trabajo de diploma en opción al título de Ingeniero en Ciencias Informáticas, Universidad de las Ciencias Informáticas, La Habana, Cuba, 2018.

AMÓN, Iván y JIMÉNEZ, Claudia. *Funciones de Similitud sobre Cadenas de Texto: Una Comparación Basada en la Naturaleza de los Datos*. In: *International Conference on Information Resources Management*. 2010. p. 13.

BAEZA-YATES, Ricardo A. y GONNET, Gaston H. *A New Approach to Text Searching*. *Communications of the ACM* [en línea]. 1992. Vol. 35, p. 74-82. Disponible en: http://dl.acm.org/citation.cfm?id=135243

BAEZA-YATES, Ricardo y RIBEIRO-NETO, Berthier. *Modern Information Retrieval: The Concepts and Technology behind Search*. *Information Retrieval*. 2011. Vol. 82, p. 944.

BERNERS-LEE, Tim, HENDLER, James y LASSILA, Ora. *The semantic web*. 2001. ISBN 9783540762973.

BILENKO, Mikhail y MOONEY, Raymond J*. Learning to Combine Trained Distance Metrics for Duplicate Detection in Databases*. *Artificial Intelligence*. 2002, p. 19.

BORDIGNON, Fernando y TOLOSA, Gabriel. *Recuperación de información: un área de investigación en crecimiento*. *Ciencias de la Información* [en línea]. 2007. Vol. 6, no. 1, p. 53-76. Disponible en: http://www.redalyc.org/articulo.oa?id=181414865002.

CACHEDA, Fidel, FERNÁNDEZ, Luna y HUETE, Guadix. *Recuperación de Información. Un enfoque práctico y multidisciplinar*. Ra-Ma, 2011.

CANÓS, José H y LETELIER, MªCarmen Penadés Patricio. *Metodologías ágiles en el desarrollo de software*. In: *Metodologías Ágiles en el Desarrollo de Software*. 2012.

CARABALLO, Lázaro. *Sistema de procesamiento de consultas para el motor de búsqueda Orión.* Trabajo de Diploma para optar por el Título de Ingeniero en Ciencias Informáticas, Universidad de las Ciencias Informáticas, La Habana, Cuba, 2017.

CASTRO, L. *Guía de gestión del riesgo tecnológico para el tratamiento de la seguridad durante el proceso de desarrollo de software*. 2014.

CHINNIYAN, Kavitha, GANGADHARAN, Sudha y SABANAIKAM, Kiruthika. *Semantic Similarity based Web Document Classification Using Support Vector Machine*. *The International Arab Journal of Information Technology*. 2017. Vol. 14, no. 3.

COHEN, William W, RAVIKUMAR, Pradeep y FIENBERG, Stephen E. *A comparison of string metrics for matching names and records*. *KDD Workshop on Data Cleaning and Object Consolidation*. 2003. Vol. 3, p. 73-78.

Cubanic. [En línea] [Cosultado el: 25 octubre de 2019.] http://www.nic.cu/.

EKAPUTRA, Fajar J, SABOU, Marta, SERRAL, Estefanía, KIESLING, Elmar y BIFFL, Stefan. *Ontology-Based Data Integration in Multi-Disciplinary Engineering Environments: A Review*. *Open Journal of Information Systms*. 2017. Vol. 4, no. 1, p. 1-26.

ESTRADA RAMOS, Luis Miguel. *Apache Solr, un motor de búsqueda de código abierto*. *Revista Digital Universitaria*. 2012. Vol. 13, no. 11, p. 1-9.

FALLIS, A.G. Metodología Actual Metodología XP. *Journal of Chemical Information and Modeling* [en línea]. 2013. Vol. 53, no. 9, p. 1689-1699. [Consultado el: 18 octubre 2019] Disponible en: http://blogs.unellez.edu.ve/dsilva/files/2014/07/Metodologia-XP.pdf

FATIMA, Shugufta y SRINIVASU, B. *Text Document categorization using support vector machine*. 2017.

FERNÁNDEZ, Miriam, CANTADOR, Iván, LÓPEZ, Vanesa, VALLET, David, CASTELLS, Pablo y MOTTA, Enrico. *Semantically enhanced Information Retrieval: An ontology-based approach*. *Journal of Web Semantics*. 2011. Vol. 9, no. 4, p. 434-452.

GADRI, S. y MOUSSAOUI, A. *Application of a new set of pseudo-distances in documents categorization*. *Neural Network World*. 2017. Vol. 27, no. 2, p. 231-245.

GLOBE. *Pruebas de rendimiento* [en línea]. 2016. [Consultado el: 22 marzo 2020]. Disponible en: https://www.globetesting.com/pruebas-de-rendimiento/

GOTOH, Osamu. *An improved algorithm for matching biological sequences.* s.l. : Journal of molecular biology, 1982. págs. 705-708. Vol. 162.

GUARINO, N. *Formal ontology, conceptual analysis and knowledge representation*. *Formal Ontology in Conceptual Analysis*. 1993. P. 1-21.

GUTIÉRREZ FARAONI, Federico Julián. *Desarrollo de una aplicación web con Spring Framework para un gestor de un recetario*. Universidad Politécnica de Madrid, 2015.

INTERNET LIVE STATS. *Real Time Statistics*. [en línea]. 2017. [Consultado el: 12 octubre 2019] Disponible en: http://www.internetlivestats.com/

IZAURRALDE, María Paula. 2013. Caracterización de Especificación de Requerimientos en entornos Ágiles: Historias de Usuario. 2013.

JARO, Matthew**.** *UNIMATCH, a Record Linkage System: Users Manual.* s.l. : Bureau of the Census, 1980.

JOHNSON, Rie y ZHANG, Tong*. Effective Use of Word Order for Text Categorization with Convolutional Neural Networks*. 2015. No. 2011, p. 103-112.

KORFHAGE, R. R. *Information storage and retrieval*. 2008.

LANTZ, Brett. *Classification using Naive Bayes*. In: *Machine Learning with R*. 2013. p. 396. ISBN 1782162151.

LARMAN, Craig. *Applying UML and Patterns*. 2004. ISBN 0131489062.

LEVENSHTEIN, Vladimir I. *Binary codes capable of correcting deletions, insertions, and reversals.* s.l.: Soviet physics doklady, 1966. págs. 707-710.

LEVENSHTEIN, Vladimir I. *Binary codes capable of correcting deletions, insertions, and reversals*. *Soviet Physics Doklady*. 1966. Vol. 10, no. 8, p. 707-710.

NAVARRO, P. L. y PÉREZ, G. M. *Open HMI Tester: un Framework Open-source para herramientas de Pruebas de Software.* España: Universidad de Murcia, 2014.

MAEDCHE, Alexander y STAAB, Steffen. *Learning Ontologies for the Semantic Web*. 2001.

MARTÍNEZ COMECHE, Juan Antonio. *Los modelos clásicos de Recuperación de información y su vigencia*. In: *Memoria del 3º seminario Hispano-mexicano de investigación en Bibliotecología y Documentación: Tendencias de la investigación en Bibliotecología y Documentación en México y España* [en línea]. 2006. p. 187-206. [Consultado el: 12 diciembre 2019]. ISBN 970-32-3961-7. Disponible en: http://eprints.ucm.es/5979/1/Modelos_RI_preprint.pdf

MARTÍNEZ, J. I. *Sistema de categorización de documentos para el buscador cubano Orión*. Universidad de las Ciencias Informáticas, 2016.

MATTHEWS, B. *Semantic web technologies*. *E-learning*. 2005. Vol. 6, p. 8.

MCCALLUM, Andres y NIGAM, Kamal*. A Comparison of Event Models for Naive Bayes Text Classification*. *AAAI/ICML-98 Workshop on Learning for Text Categorization* [en línea]. 1998. [Consultado el: 9 octubre 2019] P. 41-48. Disponible en: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.65.9324&rep=rep1&type=pdf

MICROSOFT. Algoritmo de clasificación de *Azure Cognitive Search* [en línea]. 2020. [Consultado el: 10 junio 2020] Disponible en: https://docs.microsoft.com/es-es/azure/search/index-ranking-similarity

MICROSOFT. *Revisiones de código y estándares de codificación* [en línea]. 2016. [Consultado el: 15 marzo 2020]. Disponible en: https://msdn.microsoft.com/es-es/library/aa291591(v=vs.71).aspx

MORATO, J., SÁNCHEZ-CUADRADO, S., RUIZ-ROBLES, A. y MOREIRO-GONZÁLEZ, J.-A. *Visualización y recuperación de información en la web semántica*. *Profesional de la Información*. 2014. Vol. 23, no. 3.

NEEDLEMAN, Saul B. y WUNSCH, Christian D. *A general method applicable to the search for similarities in the amino acid sequence of two proteins*. *Journal of Molecular Biology*. 1970. Vol. 48, no. 3, p. 443-453.

OCLC. [en línea]. 2015. [Consultado el: 1 diciembre 2019]. Disponible en: http://www.oclc.org/research/activities/scorpion.html

PAVONE, P. *Un Método de Text Mining para la categorización Fuzzy de documentos*. 2015.

POLLOCK, Joseph J. y ZAMORA, Antonio. *Automatic spelling correction in scientific and scholarly text*. *Communications of the ACM*. 1984. Vol. 27, no. 4, p. 358-368.

PRESSMAN, Roger S. *Ingeniería del software. Un enfoque práctico. 7ma Edición*. 2010.

PURI, S. *VECTOR*, *Support y CLASSIFIER, Machine. A Fuzzy Similarity Based Concept Mining Model for Text Classification*. *International Journal of Advanced Computer Science and Applications*. 2011. Vol. 2, no. 11, p. 115-121.

RADA, Roy, MILI, Hafedh, BICKNELL, Ellen y BLETTNER, Maria. *Development and Application of a Metric on Semantic Nets*. *IEEE Transactions on Systems, Man and Cybernetics*. 1989. Vol. 19, no. 1, p. 17-30.

RAJU, Miji K., SUBRAHMANIAN, Sneha T. y SIVAKUMAR, T. *A Comparative Survey on Different Text Categorization Techniques. Journal of Computer Science and Engineering*. 2017. Vol. 5, p. 1612-1618.

RAMÍREZ BUSTAMANTE, Flora y LÓPEZ DÍAZ, Enrique. *Spelling error patterns in Spanish for word processing applications*. In: *LREC 2006. Proceedings of the 5th International Conference on Language Resources and Evaluation*. 2006, p. 93-98.

RODRÍGUEZ, Diego y MARTÍN, José. *Detección de plagio en documentos. Sistema externo monolingüe de altas prestaciones basado en n-gramas contextuales*. Procesamiento de Lenguaje Natural, Revista nº 45, septiembre 2010, pp 49-57.

SAMPIERI, H. *Medición y operacionalización. Variables. Indicadores* [en línea]. 2010. [Consultado el: 22 marzo 2020] Disponible en: https://es.slideshare.net/Eulaliaperalta/operacionalizacin-variables-sampieri.

SANCHEZ, S*. Estrategia de soporte técnico para el proceso de migración a código abierto en los Organismos de la Administración Central del Estado*. *Universidad de las Ciencias Informáticas*, 2015.

SANCHEZ-VEGA, Fernando. Identificación de plagio parafraseado incorporando estructura, sentido y estilo de los textos. [en línea]. 2016. [Consultado el: 20 junio 2020] Disponible en: https://www.repositorionacionalcti.mx/autor/JOSE+FERNANDO+SANCHEZ+VEGA

SARDIÑAS, Javier. QtNLP-Diff: *Comparador de casos de corpus lingüísticos y resultados de algoritmo de similitud.* Trabajo de diploma en opción al título de Ingeniero Informático, Universidad Central "Marta Abreu" de las Villas, Villa Clara, Cuba, 2016.

SEARCH TECHNOLOGIES. *Arquitectura Basada en Componentes* [en línea]. 2017. [Consultado el: 10 diciembre 2019]. Disponible en: http://www.scribd.com/doc/14704374/Arquitectura-Basada-en-Componentes

SELVI, S. Thamarai, KARTHIKEYAN, P., VINCENT, A., ABINAYA, V., NEERAJA, G. y DEEPIKA, R. *Text categorization using Rocchio algorithm and random forest algorithm*. In: *2016 8th International Conference on Advanced Computing, ICoAC 2016*. 2017. p. 7-12. ISBN 9781509058884.

SIDOROV, Grigori, GELBUKH, Alexander, GÓMEZ-ADORNO, Helena y PINTO, David*. Soft similarity and soft cosine measure: Similarity of features in vector space model. Computación y Sistemas*. 2014. Vol. 18, no. 3, p. 491-504.

SMITH, T. y WATERMAN, M. *Identification of Common Molecular Subsequences.* s.l. : Molecular Biology, 1981. págs. 195-197. Vol. 147.

SOMMERVILLE, Ian. *Ingeniería del software* [en línea]. 2005. [Consultado el: 17 enero 2020] ISBN 8478290745. Disponible en: http://danielr.obolog.es/ingenieria-software-355416%5Cnhttp://fondoeditorial.uneg.edu.ve/citeg/numeros/c02/c02_art10.pdf

SPARXSYSTEMS. *Diagrama de despliegue UML 2* [en línea]. 2018. [Consultado el: 25 enero 2020]. Disponible en: http://www.sparxsystems.com.ar/resources/tutorial/uml2_deploymentdiagram.html

TORRES, Annia. *El uso de los buscadores en Internet.* ACIMED Vol.11, no.3. Ciudad de La Habana Mayo-jun. 2003. Disponible en: http://scielo.sld.cu/scielo.php?script=sci_arttext&pid=S1024-94352003000300004

TORRES, Carmen y ARCO, Leticia. *Representación textual en espacios vectoriales semánticos*. *Revista Cubana de Ciencias Informáticas*. 2016. Vol. 10, no. 2, p. 148-180.

UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS. *Metodología de desarrollo para la Actividad productiva de la UCI*. 2015.

VALERO M., Ana Isabel. *Técnicas estadísticas en minería de textos*. 2017.

VILTRES, H. S., LEYVA, P. R., FEBLES, J. P. y SENTÍ, V. E. *Procesamiento Semántico de información en Sistemas de Recuperación de Información*. *Revista Cubana de Ciencias Informáticas*. 2018. Vol. 12, p. 102-116.

VISUAL PARADIGM. *Software design tools for agile software development* [en línea]. 2014. [Consultado el: 10 junio 2020]. Disponible en: http://www.visual-paradigm.com/product/vpuml

WADHAWAN, Rimpy y MITTAL, Saurabh. *Improved Nearest Neighbour Approach for Document Categorization*. 2017.

WINKLER, William E. *Frequency-based matching in Fellegi-Sunter model of record linkage*. *Bureau of the Census Statistical Research Division*. 2000. Vol. 14.

YANCEY, William. *Evaluating string comparator performance for record linkage.* Statistical Research Division. 2005. P. 3905-3912.

Anexos

Tabla 18: Plan de preguntas. Entrevista realizada a los principals especialistas del proyecto de la plataforma C.U.B.A

No.	Preguntas
1	¿Qué estructura tiene la plataforma C.U.B.A.?
2	¿Cómo almacena las noticias?
3	¿Cómo detecta la similitud entre noticias?
4	¿Qué herramientas tecnológicas se utilizan en su implementación?
5	¿Qué metodología de desarrollo empleó el proyecto?

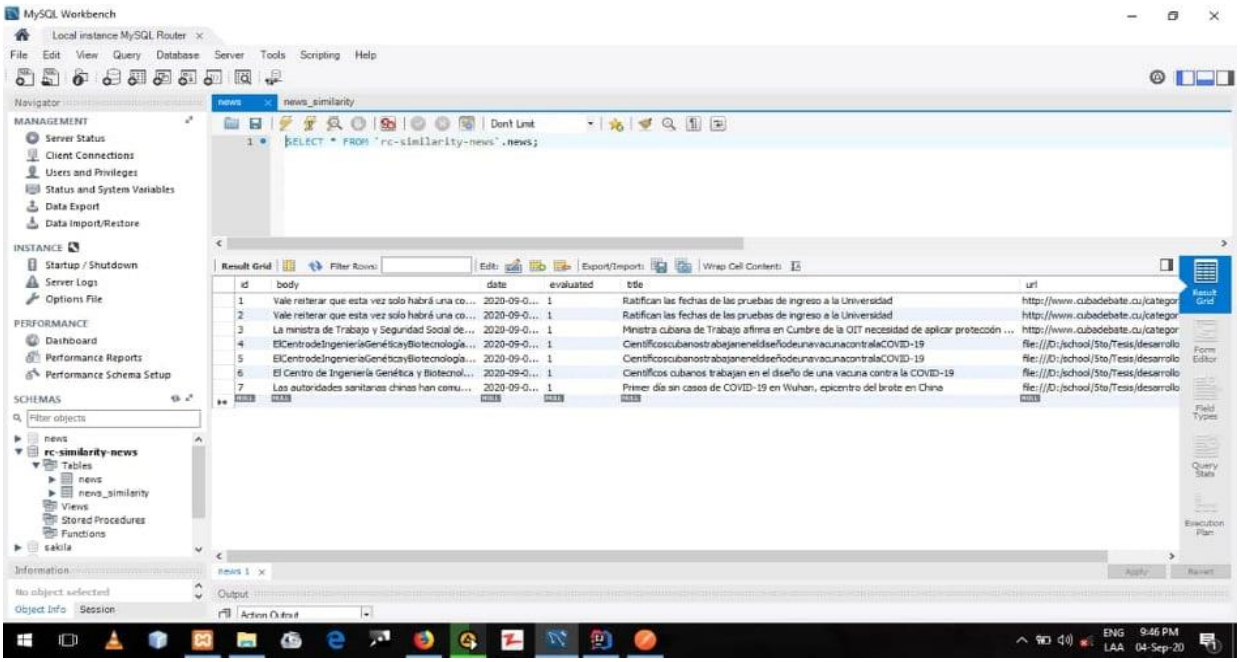

Ilustración 9: Cálculo de similitud aplicando algoritmo seleccionado (captura de pantalla)

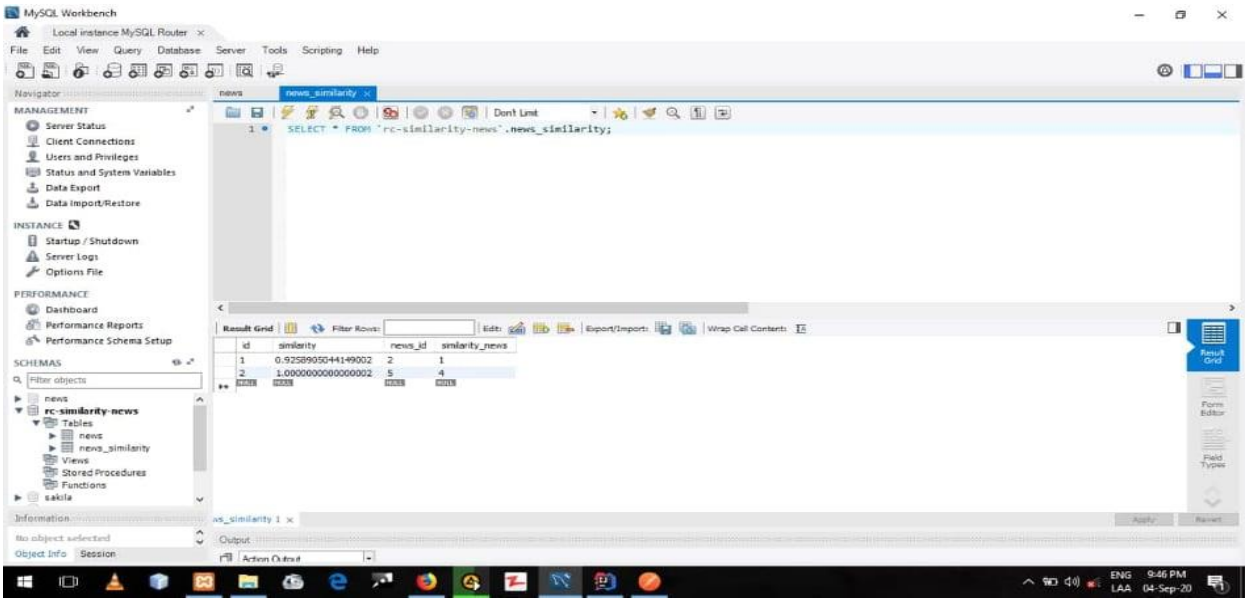

Ilustración 10: Ejecución del algoritmo (captura de pantalla, noticia con identificador 1 y noticia con identificador 2 son similares y poseen un índice de similitud de 0.92; noticia con identificadores 5 y 4 con un índice de similitud, 1.0)