UNIVERSIDAD CENTRAL DE VENEZUELA
FACULTAD DE CIENCIAS
POSTGRADO EN CIENCIAS DE LA COMPUTACIÓN

RECUPERACIÓN, EXTRACCIÓN Y CLASIFICACIÓN DE INFORMACIÓN DE SABER UCV

Trabajo de grado de Maestría presentado ante la
ilustre Universidad Central de Venezuela por el
Econ. José Miguel Avendaño Infante para optar al título de
Magister Scientiarum en Ciencias de la Computación

Tutor: Dr. Andres Sanoja

Caracas - Venezuela
Octubre 2023

Resumen:

Se presenta una propuesta de aplicación web distribuida para implementar un Sistema de Recuperación de Información (*information retrieval*) mediante técnicas de Procesamiento de Lenguaje Natural (NLP), de indexación en base de datos y visualizaciones con gráficos y gráfos de coocurrencias de palabras para los trabajos especiales de pre y postgrado realizados por los estudiantes de la Universidad Central de Venezuela que se encuentran alojados en el repositorio www.saber.ucv.ve.

Esta propuesta se basa principalmente en que mediante la búsqueda de palabras o frases se genere un (*query*), y mediante la técnica conocida como "*full text search*" se puedan extraer los trabajos en que se encuentran contenidas tales palabras y a partir de ahí enriquecer la experiencia del usuario con la presentación de la información recuperada.

La aplicación se diseña como un sistema distribuido en distintos contenedores soportando cada uno un proceso para el funcionamiento, teniendo entre los principales el de base de datos, el servidor de la aplicación y otro con los distintos procesamientos que son efectuados sobre los textos.

Dedicatoria:

A mi hijo Cassiel y

mi esposa Waleska.

Agradecimientos:

- A mi madre, obvio, sino no habría ni una sola palabra acá.

- A mi padre Fernando por negarme el Atari e insistir en el Oddysey 2.

- A mi tía Mercedes Infante.

- A Cesar Alejandro García por todas las ayudas.

- A mi hermano David por su soporte.

- Dr. Andres Sanoja primero por aceptar la tutoría y enseñarme qué es la investigación dentro de una comunidad científica.

- Dr. José Mirabal por siempre andar con alguna idea a desarrollar y el tiempo dedicado a escuchar las propuestas y complementar esta Investigación.

- Dra. Concettina Di Vasta por las imponentes sesiones de 2 horas 15 minutos llenas de coherencia y conocimiento.

- Dra. Haydemar Nuñez por la rigurosidad al impartir los conocimientos.

- Dra. Vanessa Leguizamo por tomarse el tiempo de revisar la solicitud de estudio de un oxidado economista y por ser mi Prof.ª.

- Dra. Nuri Hurtado Villasana por tomarse el tiempo de escucharme y brindarme sugerencias en la elaboración de este trabajo.

- A todo el personal del Postgrado: sus buenos días, por tener a mano la llave, por ayudar a mantener viva la Academia.

- Mauricio Sáez Toro del equipo Saber UCV por mantener activo el Sistema Saber UCV y tener el tiempo de haber colaborado con esta investigación.

- A toda la comunidad de creadores de software libre y open sourcem en especial a los #useRs por motivarme a adentrarme al mundo de las ciencias de la computación.

- A Alexandra Asanovna Elbakyan, sin ella serían mínimas las citas bibliográficas de esta Investigación.

Every important aspect of programming arises somewhere in the context of sorting or searching.

— Donald Knuth, *The Art of Computer Programming*, Volume 3

Índice

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

Capítulo 2

Planteamiento del Problema:

2.1 Propuesta Técnica:

En este Capítulo se detalla la propuesta técnica de la Solución en la sección 2.2. Posteriormente se indica el Objetivo General 2.3 y los Objetivos Específicos 2.4 que se aspiran cubrir en el desarrollo. A continuación se muestra el esquema general de la aplicación 2.5, junto con el esquema de funcionamiento en los contenedores 3.9.1 y se describe el funcionamiento de cada contenedor.

Se realiza un análisis de factibilidad en la sección 2.6 para la implementación y se muestra un cronograma **??** de actividades.

2.2 Propuesta Técnica:

Crear una Solución que funcione como una aplicación web distribuida bajo la arquitectura cliente-servidor. Del lado del servidor se encontrarán contenedores que alojarán los distintos servicios necesarios para el funcionamiento.

Del lado del cliente se podrán formular los *querys* con múltiples atributos (la frase a buscar, la jerarquía académica, intervalo de fechas, facultad y/o escuela de adscripción).

La Solución permitirá generar procesos de extracción de información (*information retrieval*) y los resultados se mostrarán en tablas interactivas, gráficos y grafos de coocurrencia de palabras. Adicionalmente se indicarán recomendaciones de textos basados en la similitud que presente un documento con los otros.

Los textos del resumen de cada tesis con los cuales se conformará el *Corpus*, serán obtenidos y actualizados mediante técnicas de *web crawling* realizadas al repositorio Saber UCV.

Se opta por la estrategia de hacer el *web crawling* a los documentos contenidos en Saber UCV al no ser posible, al momento de realizar esta propuesta, la obtención de la base de datos de los documentos ahí alojados.

La Solución propuesta está diseñada para irse actualizando periódicamente incorporando los nuevos documentos que sean alojados en el repositorio Saber UCV.

Contará con un procedimiento que permite clasificar los documentos (tesis) por área académica, según donde haya efectuado estudios el autor del trabajo, al ser la tesis el requisito necesario para obtener el título de grado. Actualmente el repositorio Saber UCV no dispone de esta clasificación.

2.3 Objetivo:

Crear una Solución que permita realizar la clasificación de documentos y la búsqueda de información sobre los trabajos de investigación que se encuentran en el Repositorio SABER UCV usando técnicas de extracción de información (*information retrieval*).

Para el procesamiento de las búsquedas será usado un sistema distribuido con distintos componentes (contenedores) que permitan la ingesta, procesamiento y transformación de los datos, al igual que el alojamiento de los mismos en una base de datos.

2.4 Objetivos Específicos

- *Querys* multi atributo en distintas dimensiones: tiempo, jerarquías (pregrado, especializaciones, maestría y/o doctorado), facultad ,carreras o postgrados.
- A cada resultado que arroje la búsqueda se le debe asignar un nivel de relevancia. El conjunto de los textos recuperados se debe mostrar ordenadamente de mayor a menor relevancia. La asignación de la relevancia será hecha mediante una función que cuente con distintos parámetros.
- Los datos: textos, fechas, clasificaciones, y demás que sean necesarios para el funcionamiento de la Solución, deben registrarse en una base de datos estructurada y la ingesta y consulta se efectuará mediante un manejador de base de datos.
- Generar recomendaciones de textos basados en la similitud coseno que presente un documento con cada uno de otros que conforman el *Corpus*.
- En los documentos extraídos se debe contar con el enlace a los documentos que reposan en Saber UCV.
- Permitir la concurrencia de accesos al Sistema.
- La tolerancia a fallas en los contenedores.
- Contar con el certificado SSL para acceso seguro por parte de los visitantes.
- Procesar las coocurrencias de las palabras más comunes las cuales se deberán visualizar mediante grafos.
- Disponer de una aplicación web para el acceso al Sistema por parte del cliente.
- Disponer de ventanas emergentes de ayuda en la interface gráfica de la aplicación web.

- En la representación de coocurrencias de palabras mediante grafos, se debe poder filtrar documentos interactivamente al hacer *click* con el *mouse* sobre los arcos que unen un par de nodos, donde cada uno de estos representa la coocurrencia de una dupla de palabras. El *click* generará el evento para el query sobre los documentos que contienen esa determinada coocurrencia.

2.5 Esquema del SSCSU:

Se representa el Sistema y sus interacciones mediante un esquema con un elevado grado de abstracción. Ver figura 2.1.

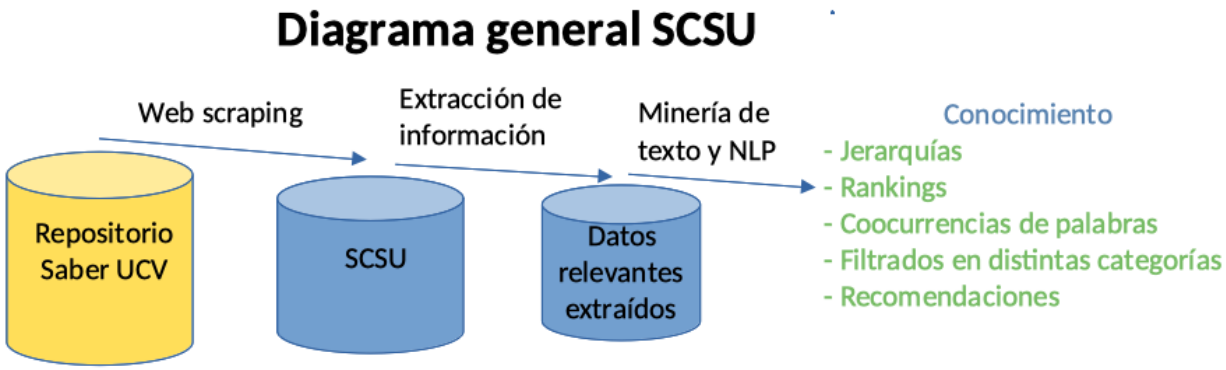

Figura 2.1: Esquema General

En la figura 2.1 se muestra como el SSCSU está diseñado para que mediante técnicas de *web crawling* se pueda extraer la información del repositorio Saber UCV. Posteriormente el sistema ante la consulta del usuario va a recuperar los documentos que sean relevantes, representando el conocimiento con la generación de jerarquías, rankings, coocurrencias de las palabras más mencionadas y recomendaciones de texto según similitudes.

El SSCSU se implementará mediante el uso de Docker. En la figura 2.2 se muestra el esquema con los contenedores.

2.5.1 Cliente - Servidor:

La arquitectura **cliente-servidor** en esta Propuesta se basa en:

2.5.1.1 1 - Cliente - Navegador web:

El acceso del cliente a la aplicación web se hace en el navegador web. Este realiza la petición al servidor que aloja la Solución. Actualmente se cuenta con un prototipo al que se puede acceder en la dirección https://proyecta.me/.

El sistema no está diseñado para usarse desde dispositivos móviles, aunque igualmente es viable el acceso no estando optimizada la interfaz del usuario ni las visualizaciones de los resultados de los *querys*.

DIAGRAMA COMPONENTES DOCKER

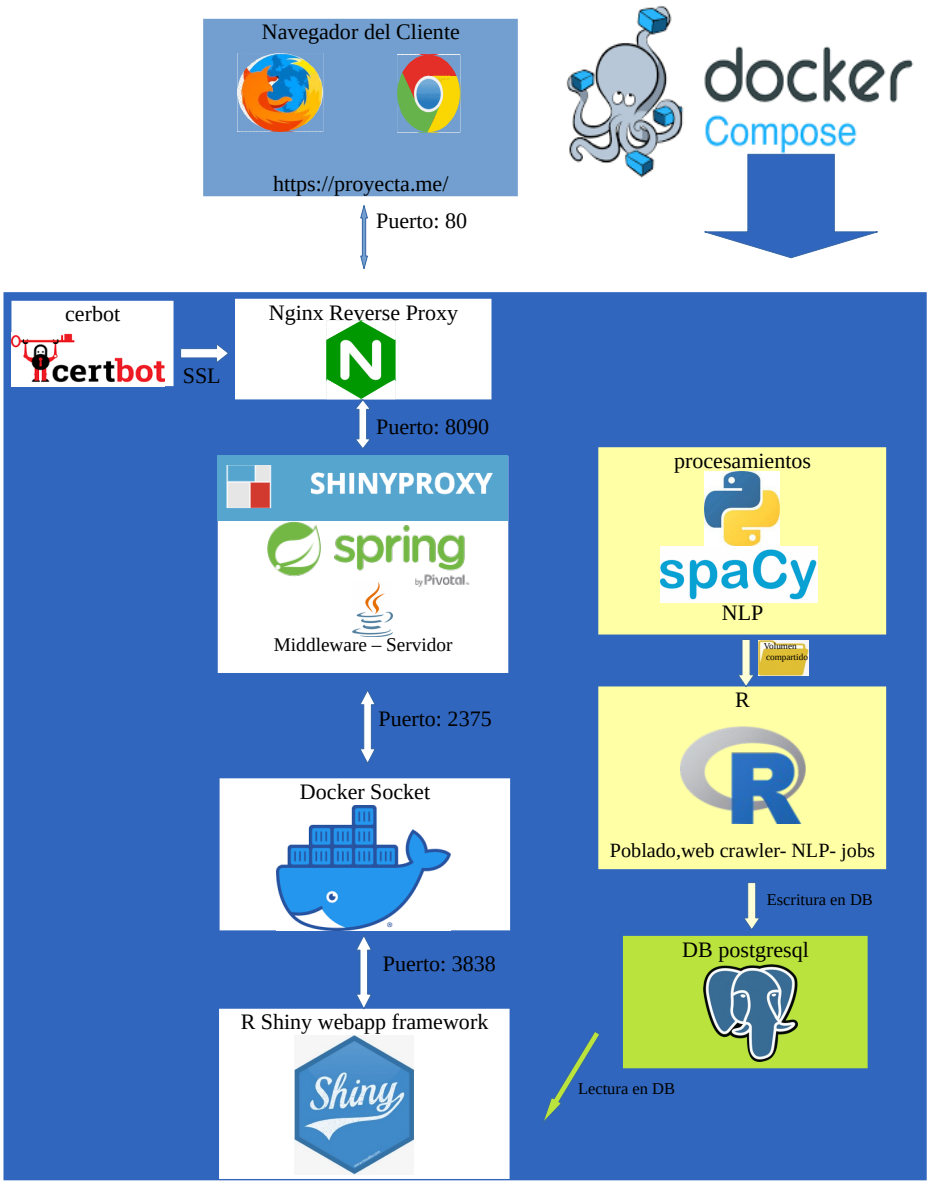

Figura 2.2: Diagrama de Componentes

2.5.1.2 2 - Servidor:

El prototipo del sistema requiere para funcionar al menos estos recursos:

- 2 CPU virtual
- 2 GB de memoria RAM
- 50 GB de disco duro

En el servidor se instala el software Docker sobre el cual se despliegan los siguientes contenedores:

2.5.2 Contenedores:

2.5.2.1 Docker Compose:

Funciona como un orquestador para correr aplicaciones distribuidas en múltiples contenedores usando un archivo en formato yml donde se establecen las imágenes, los puertos y los volúmenes que serán usados y compartidos por cada uno de los contenedores.

2.5.2.2 Contenedor con Nginx:

Es el servidor web/proxy inverso de código abierto que sirve para redireccionar las peticiones del puerto 80 al puerto 8090. Mediante este contenedor también se define el certificado SSL para permitir conexiones por el protocolo HTTPS.

Este contenedor fue generado desde una imagen oficial de NGINX que se encuentra en el ***docker hub*** [1] sin añadir ninguna capa (layer) adicional.

2.5.2.3 Contenedor con Cerbot:

Cerbot es una herramienta de código abierto que permite habilitar las conexiones mediante el protocolo HTTPS con el uso de un certificado "Let´s Encrypt". El uso de este certificado está asociado al uso de un dominio en el *deploy* de la aplicación.

Este contenedor fue generado desde una imagen de CERBOT del ***docker hub*** sin ninguna modificación posterior.

2.5.2.4 Contenedor con Shinyproxy:

Es una implementación del servidor "*Spring boo*t" que dará servicio a las aplicación web desarrolladas en *shiny* [2] 2.5.2.5. Con el uso de este *middleware* se obtienen las siguientes ventajas:

- Ante cada petición de acceso al servidor se despliega un *workspace* completamente aislado, es decir, un contenedor distinto. Las aplicaciones desarrolladas en *shiny* son *single threaded* y adoptar esta estrategia representa una ventaja, motivado a que se pueden controlar los recursos de memoria y cpu asignados a cada contenedor que se despliega.

[1] repositorio de imágenes de contenedores de docker.
[2] Shiny un framework para crear aplicaciones web en el lenguaje de programación R.

- Permite establecer *login* en el uso de la aplicación y grupos de usuarios. También da soporte a distintos métodos de autenticación. Si bien en estos momentos la aplicación es de libre acceso, en algún momento se pudiese restringir y no sería necesaria ninguna modificación en la arquitectura, más allá de cambiar el archivo de configuración.
- Uso de una tecnología estable y probada.

Es necesario destacar que originalmente *Shiny* como *framework* cuenta con su propio software que actúa como servidor, pero para tener acceso a ciertas funcionalidades es necesario pagar por el servicio directamente a la Fundación RStudio Software y usar el alojamiento que ellos proveen, no siendo todos los componentes de código abierto. Por ejemplo, el acceso mediante *login* no está disponible en la versión libre del *Shiny Server* sino en la *Shiny Server Professional*.

Ciertas configuraciones de librerías e incluso la propia contenerización de la aplicación, no es posible usando el servicio pago, así que la propuesta acá adoptada, si bien representa un mayor esfuerzo en la configuración, claramente implica que se obtienen una serie de ventajas, por todas las adaptaciones y control que es posible realizar al y sobre el sistema.

Este contenedor funciona en el puerto 2375 y fue generado desde la imagen del docker hub *Shinyproxy* sin ninguna modificación.

2.5.2.5 Contenedor con "*R Shiny Web App framework*":

En este contenedor es donde reposa la aplicación web con todas las librerías necesarias para generar las visualizaciones y remitir los *querys* al contenedor del manejador de la base de datos 2.5.2.6 . Como comentamos anteriormente, cada vez que ocurre desde el navegador del cliente una petición de acceso, desde el contenedor *shinyproxy* 2.5.2.4, se crea una replica de este contenedor con todos los elementos necesarios para que la app funcione correctamente.

En caso de presentar alguna falla, el sistema sería tolerante, porque se pueden seguir recibiendo peticiones que replicarían una imagen nueva del contenedor sin afectar al que presentase el fallo, o viceversa.

Desde este contenedor se realiza el acceso de lectura al contenedor que contiene *PostgreSQL* 2.5.2.6 donde reposa la base de datos que contiene los textos ya procesados.

Por los momentos no hay escritura de datos en las tablas, pero está contemplado que se registren los querys formulados en alguna tabla, junto con los documentos que el usuario revisa mediante las interacciones, para así generar métricas de calidad del sistema y del uso que se le da.

La imagen que se usa en este servidor fue definida a medida con todos los recursos necesarios.

En un posterior Capítulo a desarrollar en donde se mostrará el proceso de desarrollo de la Solución, serán descritas todas las librerías que se encuentran incluidas en este contenedor y se expondrán las razones para seleccionar cada una de ellas.

Varios de los procesos que se ejecutan en este contenedor ocurren al momento de recibir un *query,* no obstante todos los procesos que puedan ser pre computados, se trata de ejecutarlos previamente en el contenedor "R Servicios" 2.5.2.7, para lograr así la disminución de los tiempos.

Funcionalidades principales: **ENTRADAS**

- Contiene un campo para la entrada de texto que generará el query.
- Contiene un selector para indicar si se quiere generar la coocurrencia de palabras
- Contiene tablas para seleccionar:

  1) Nivel académico del trabajo. Opciones (pregrado, especialización, maestría, doctorado).
  2) Facultad o Centro de adscripción. Opciones: 11 Facultades más un centro (CENDES).
  3) Nombre del pregrado o postgrado. En total son 412 las opciones.

  Cada una de las tablas anteriores se actualiza según se vayan seleccionando las relaciones y la disponibilidades. Por ejemplo, al seleccionar pregrado solo se mostrarán los nombres de las carreras de pregrado, pero si se selecciona también el nombre de la Facultad, sólo se mostrarán las carreras de pregrado dentro de la Facultad seleccionada. Para una determinada tabla también se permiten selecciones múltiples dando una total flexibilidad al momento de ejecutar los *querys*.

SALIDAS - Ante el *query* se genera:

- Una tabla creada con la librería *reactable*. En la misma se muestra cada uno de los documentos recuperados con los distintos atributos disponibles: autor, fecha, palabras clave, texto resumen. El orden en que aparece está generado con una función de ranking. Adicionalmente se muestra un enlace al repositorio Saber UCV que es donde se encuentra alojado el respectivo trabajo (el documento en PDF). Igualmente se presentan los textos que tienen mayor similitud con el texto seleccionado.
- Un gráfico con la frecuencia por año de los trabajos extraídos mediante el *query*. El gráfico se generá con la librería *apexcharter* que es un wraper para Javascript, por lo cual el gráfico tiene ciertas interactividades. Con el *hoover* muestra el valor de cada columna.
- Gráfico de coocurrencia interactivo de palabras: se genera mediante la librería de *VisNetwork* que también es un *wrapper* de javascript. Este gráfico permite seleccionar un arco de unión entre dos palabras coocurrentes. Al realizar la selección se filtran un subconjunto de los documentos que contienen ambas palabras representadas por nodos. Los documentos filtrados se mostrarán en una tabla contigua, también generada en *reactable*, donde sólo se incluye el texto resumen de cada trabajo. En un próximo Capítulo a desarrollar será mencionado el diseño y funcionamiento a detalle de esta visualización.
- Gráfico de coocurrencia estático de palabras: mediante la librería *ggraph* son generados un par de gráficos con distintas granularidades. El primero exhibe la misma coocurrencia de palabras expuesta en el punto anterior pero esta vez es generada en una visualización estática. En cuandoa la granularidad se muestran las palabras que coocurren dentro de todo el resumen, El segundo gráfico también muestra la coocurrencia, pero solo de palabras que coocurren una seguida de otra dentro del texto resumen, es decir que se muestran los resultados con una menor granularidad.

Con la librería *UdPipe* se generan las estructuras de datos necesarias para generar los grafos (arcos y nodos).

2.5.2.6 Contenedor con PostgreSQL:

En este contenedor se tiene una imagen de *PostgreSQL* versión 13.3. No fue realizada ninguna otra modificación distinta a la definición de usuarios y poblado desde base de datos incluyendo un volumen compartido para garantizar que tengamos "datos persistentes. Este contenedor recibe consultas del contenedor"R Shiny Web App framework" 2.5.2.5 y escritura desde el contenedor "R imagen Servicios" 2.5.2.7.

En este contenedor ocurre la indexación de la base de datos y la generación del ranking al procesar el resultado con la función de PosgreSQL llamada tsrank [3].

En una tabla se encuentra la identificación del documento, la fecha de creación y propiamente los textos que mediante el Tsvector [4] almacena el título, el resumen, el autor y las palabras clave.

En otra tabla se encuentra el almacenamiento del procesamiento que se le hace a los textos, clasificando cada una de las palabras mediante el *part of speach* , y registrando el identificador del documento al que está asociada y el lema [5]de cada una.

El TSvector es la estructura de datos que permite la búsqueda de texto completa (*Full Text Seach*) mediante la función de *PostgreSQL* denominada *tsquery*. En un futuro capítulo a desarrollar será mostrado el diseño de las tablas con sus campos.

2.5.2.7 Contenedor con "R Imagen Servicios":

En este contenedor se creó una imagen con todos los servicios necesarios para realizar el web crawling, el procesamientos de textos y la descarga de los archivos desde Saber UCV para realizar la clasificación de las Tesis. Al iniciar el Sistema también contiene las funcionalidades que permiten realizar la creación de la base de datos, de las tablas y el poblado de estas.

Periodicamente es invocado este contenedor para realizar los procesos de incorporación de aquellos documentos nuevos que se detecte que están disponibles en Saber UCV.

En una primera fase se uso la utilidad del sistema operativo linux para la programación de actividades, sin embargo se está implementando el uso de la tecnología *Apache Airflow* para la ejecución de la programación de los flujos de trabajo.

La imagen base usada es la del proyecto ***Rocker*** [@RJ-2017-065:2017] ,la cual es una versión ampliamente probada y optimizada en, y por, la comunidad de usuarios de R.

Posteriormente serán descritas todas las librerías que fueron añadidas mediante una capa (layer) a este contenedor. Por los momentos se especifican que se encuentran agregadas las siguientes:

2.5.2.7.1 Text Mining y NLP: Generalmente las distintas librerías que permiten realizar procesos de *Natural Language Processing* también hacen procesos de *Text Mining* parcial o totalmente. En

[3] esta función mide la relevancia de los documentos para una consulta en particular, de modo que cuando haya muchas coincidencias, las más relevantes puedan mostrarse primero tomando en cuenta la información léxica, de proximidad y estructural del documento (título, cuerpo del documento, etc).

[4] El Tsvector es una función de postgresql que crea una estructura de datos que es una lista ordenada de distintos lexemas, que son palabras que se han normalizado para fusionar diferentes variantes de la misma palabra mediante el algoritmo de Porter.

[5] El lema es la forma que por convenio se acepta como representante de todas las formas flexionadas de una misma palabra.

la investigación fueron evaluadas múltiples librerías como *Spacy, Quanteda, OPENLP, CoreNLP, Freeling y Udpipe*.

En una futura entrega se hará una breve mención a cada una y la razón por la cual se adoptó Spacy para varios de los procesos de NLP.

Para ejecutar *Spacy* es necesario usar los archivos que se encuentran en el contenedor *"Python Spacy"* , 2.5.2.8 donde está instalada la libraría *Spacy* que corre en *Python*.

Procesos que se ejecutan llamando al contenedor "Python Spacy": Tokenización, POS y Lematización.

También en este contenedor se realizan los siguientes procesos:

1. Poblado base de datos: se hace el *web crawling* para el poblado inicial y adicionales de la base de dato, con los textos de los Resúmenes.
2. Clasificación de los documentos: mediante una rutina compuesta por varios algoritmos serán clasificados los distintos trabajos según lo antes expuesto.
3. Cálculo de la Similitud de los documentos: cuando se ha realizado la lematización de las palabras se procede a generar una matriz de tipo Td-Idf (term document- inverse document frecuency), que sirve de insumo para el cálculo de similitud entre los documentos. Este cálculo de similitud se realiza con la librería Quanteda.texstats y se usa la medida de similitud coseno, ya que varios autores la sitúan como una de las mejores formas de comparar la similitud entre un documento y otro.

2.5.2.8 Contenedor con "Python Spacy":

Se creó una imagen que contiene un ubuntu con python, spacy y el modelo de Spacy es_core_news_sm . Su función es que mediante un volumen compartido pueda ser invocado desde el contenedor "R Imagen Servicios" 2.5.2.7 para así realizar los procesamientos de NLP antes descritos.

2.6 Factibilidad:

Para la propuesta del SCSU se hizo una evaluación de la factibilidad del desarrollo del proyecto que consistió en hacer pruebas de ***web crawling*** sobre el repositorio Saber UCV al no ser viable la obtención del conjunto de datos por otro medio. Igualmente se realizaron pruebas sobre el hardware disponible con arquitecturas similares a la que se propondrá más adelante en nuestro desarrollo.

Estas pruebas fueron exitosas por lo cual no se estima que exista algún factor que impida la implementación del Sistema acá expuesto.

Capítulo 3

Marco teórico-referencial:

En este Capítulo se muestra el marco teórico en que se sustentan los aspectos de mayor relevancia para el desarrollo de la Solución. Principalmente se enuncian una serie de conceptos que involucran algoritmos de búsqueda, la recuperación de información, la minería de texto, el procesamiento del lenguaje natural, la estructuración de la base de datos y lo referente a la arquitectura distribuida en que se soporta la SSCSU.

3.1 Reseña histórica:

El profesor Donald Knuth señala, dentro del campo de las ciencias de la computación, que la **búsqueda** *es el proceso de recolectar información que se encuentra en la memoria del computador de la forma más rápida posible, esto cuando tenemos una cantidad N de registros y nuestro problema es encontrar el registro apropiado de acuerdo a un criterio de búsqueda* (Knuth, 1997) (p. 392) .

Iniciamos con esta cita porque la recuperación de información gira en torno a un problema central de las ciencias de la computación que es la **búsqueda**. A continuación se mencionarán una serie de algoritmos que abordan este problema, no necesariamente resultando óptimos para dar solución a lo planteado en **??**.

En la década de 1940 cuando aparecieron las computadoras, las búsquedas no representaban mayor problema debido a que estas máquinas disponían de poca memoria *RAM* pudiendo almacenar sólo moderadas cantidades de datos. Ellas estaban diseñadas para realizar cómputos y arrojar los resultados más no para tenerlos almacenados en memoria.

No obstante con el desarrollo del almacenamiento en memoria *RAM* o en dispositivos de almacenamiento permanentemente, ya en la década de 1950 empezaron a aparecer los problemas de **búsqueda** y los primeras investigaciones para afrontarla. En la década de 1960 se adoptan por ejemplo estrategias basadas en arboles.

Los primeros algoritmos que sirvieron para localizar la aparición de una frase dentro de un texto, o expresado de forma más abstracta, como la detección de una subcadena *P* dentro de otra cadena *T*, fueron los algoritmos de *Pattern-Matching* (Goodrich et al., 2013) (p. 584).

Así nos encontramos en la literatura con el algoritmo *Fuerza Bruta* donde dado un texto T y una subcadena P, se va recorriendo cada elemento de la cadena T para detectar la aparición de la subcadena P. Si bien este algoritmo no presentaba el mejor desempeño, por contar con ciclos anestados en su ejecución, creó una forma válida de enfrentar el problema de la búsqueda de subcadenas de texto.

El algoritmo *Knuth-Morris-Pratt* que se introdujo en 1976 tenía como novedad que se agregó una función que iba almacenando "previas coincidencias parciales" en lo que eran fallos previos y así al realizar un desplazamiento tomaba en cuenta cuántos caracteres se podían reusar. De esta forma se logró considerablemente mejorar el rendimiento en los tiempos de ejecución de $O(n+m)$ que son asintóticamente óptimos.

Posteriormente en 1977 el problema se enfrenta con un nuevo algoritmo que es el de *Boyer-Moore* en el cual se implementan dos heurísticas (*looking-glass y character-Jump)* que permiten ir realizando algunos saltos en la búsqueda, ante la no coincidencia de la subcadena con la cadena y adicionalmente, el orden en el que se va realizando la comparación se invierte. Estas modificaciones permitieron obtener un mejor desempeño.

Sobre una modificación al algoritmo *Boyer-Moore* se sustenta la utilidad *grep* de la línea de comandos UNIX que igualmente le da soporte a diversos lenguajes que la usan para ejecutar búsquedas de texto con un proceso que comúnmente es conocido como *grepping*. Esta utilidad fue ampliamente usada para resolver parcialmente lo expuesto en **??**.

Los algoritmos mencionados anteriormente pueden ser usados en procesos de recuperación de información en conjunto con técnicas que pueden mejorar considerablemente los tiempos en la ejecución de las rutinas siendo una de estas el preprocesamiento de los textos, *eg*. remover determinados caracteres, aplicar el algoritmo de porter, entre otras más.

Una estrategia que surgió para enfrentar las búsquedas de texto, fue el uso de la programación lineal donde bajo la premisa *divida et impera* los problemas que requieren tiempo exponencial para ser resueltos son descompuestos en polinomios y por lo tanto se disminuye la complejidad en tiempo para ser resueltos. Entre este tipo de algoritmos se puede mencionar los de *alineación del ADN* . Originalmente el algoritmo se desarrolló para resolver problemas de alineación de cadenas de ADN de forma parcial o total dentro de una cadena mayor. Posteriormente se identificó que este tipo de procedimiento era extrapolable a los subcadenas de texto. Una de las versiones de estos algoritmos es la denominada ***Smith-Waterman*** que resultó de gran utilidad para resolver el problema planteado en **??** ya que la estrategia de usar la utilidad *grep* fue infructuosa en algunos casos.

Un gran paso para aproximarnos a la aparición de los Sistemas de Recuperación de Información lo representó el enfoque que presentan los algoritmos *Tries*. Este nombre proviene del proceso de *Information Retrieval* y principalmente se basa en hacer una serie de preprocesamientos a los textos para que al momento de ejecutar la búsqueda de texto, es decir, de la subcadena dentro de la cadena, ya tengamos una parte del trabajo realizado previamente y no tener que ejecutarlo todo *"on the fly"*, es decir, sobre la marcha.

Un *Trie* (Fredkin, 1960) es una estructura de datos que se crea para almacenar textos para así poder ejecutar más rápido la coincidencia en la búsqueda. En la propuesta de la SSCSU todos los textos van siendo procesados con distintas técnicas a medida que son insertados en la base de datos. Esto claramente representa una mejora en el desempeño con la disminución en los tiempos de búsqueda.

3.2 Recuperación de Información:

El eje central sobre el cual gira el proceso de recuperación de información (RI) es satisfacer las necesidades de información relevante que sean expresadas por un usuario mediante una consulta (***query***) de texto. El investigador Charu Aggarwal en su libro sobre Minería de Texto (Aggarwal and Zhai, 2012) (p.14) menciona que el objetivo del proceso de RI es conectar la información correcta, con los usuarios correctos en el momento correcto, mientras que otro de los autores con mayor dominio sobre el tema, Christopher Manning en su libro *Information Retrieval* indica que "es el proceso de encontrar materiales (generalmente documentos) de una naturaleza no estructurada (generalmente texto) que satisface una necesidad de información dentro de grandes colecciones (normalmente almacenada en computadores)" (Manning et al., 2008) (p. 25).

Satisfacer una necesidad de recuperación de información no sólo se circunscribe a un problema **búsqueda** de un texto dentro de un *corpus*. En la mayoría de los casos se deberá cumplir con ciertos criterios, o restricciones, como por ejemplo que el *query* esté dentro de un período de fechas, o que se encuentre comprendido en un subconjunto del corpus que es a lo que se denomina **búsqueda múlti atributo**.

La información que se recolecte en una búsqueda tendrá distintos aspectos que aportarán peso en el orden en que sea presentada al usuario y no sólo vendrá dado por la aparición de las palabras sino también por otros elementos como lo puede ser la aparición de la frase del ***query*** dentro del título, la proximidad (la cercanía entre dos palabras) que tengan los términos que conforman el *query* o por otra parte la frecuencia que una palabra, o varias, se repitan dentro un determinado documento que compone el *corpus*. Igual puede aportar un peso mayor a la recuperación de un documento las referencias (citas) que contengan otros documentos a ese determinado documento. El fin último será la extracción de los documentos que resulten de mayor relevancia para el usuario.

Incluso es válido incorporar documentos, en los resultados que arroje la búsqueda, que propiamente no coincidan con los términos buscados sino que contengan términos que sean sinónimos o también añadiendo a los resultados documentos que presenten alguna similitud con el texto del ***query***. Lo que acabamos de mencionar incorporará formalmente dentro del proceso de extracción de información algo de imprecisión con la intención última de enriquecer el proceso de ***Information Retrieval*** (Kraft and Colvin, 2017).

Evaluando el proceso con cierto nivel de abstracción se tiene que el proceso de recuperación de información está compuesto principalmente por: un *query*, por un corpus y por una función de *ranking* para ordenar los documentos recuperados de mayor importancia a menor.

El desarrollo de los algoritmos expuestos en 3.1 sumado a la necesidad de resolver los problemas asociados a la búsqueda de un texto dentro de un *corpus* con múltiples atributos en tiempos aceptables y la abundante cantidad de información digital, potenciada por el uso generalizado de los computadores, abonó las condiciones para la creación de los **Sistemas de Recuperación de Información**.

3.3 Sistemas de Recuperación de Información (SRI) :

Los Sistemas de Recuperación de Información (*Information Retrieval Systems-IRS*) son los dispositivo (software y/o hardware) que median entre un potencial usuario que requiere información

y la colección de documentos que puede contener la información solicitada (Kraft and Colvin, 2017) 1. El SRI se encargará de la representación, el almacenamiento y el acceso a los datos que estén estructurados y se tendrá presente que las búsquedas que sobre él recaigan tendrán distintos costos siendo uno de estos el tiempo que tarde en efectuarse.

Es de nuestro conocimiento que generalmente los datos estructurados son gestionados mediante un sistema de base de datos pero en el caso de los textos estos se gestionan por medio de un motor de búsqueda motivado a que los textos en un estado crudo carecen de estructura (Aggarwal and Zhai, 2012) (p. 2). Son los motores de búsqueda (*search engines*) los que permiten que un usuario pueda encontrar fácilmente la información que resulte de utilidad mediante un *query*.

El SSCSU está diseñado como un IRS donde se pueden ejecutar querys que son procesados y los resultados que se obtienen son sometidos a una función de ranking que será expuesta en una fase posterior del desarrollo de esta investigación.

3.4 Ejemplos de IRS:

Profundizando en el tema de esta Investigación se mencionan un par de páginas de internet que funcionan coomo IRS.

1. Una es el proyecto denominado Arvix alojado en https://arxiv.org/, que es un repositorio de trabajos de investigación. Al momento del usuario hacer un requerimiento de información, adicional al texto de la búsqueda, se pueden indicar distintos filtros a aplicar como puede ser el área del conocimiento (física, matemática, computación, etc.). Igualmente se puede indicar si se quiere buscar sólo dentro del título de una investigación, o el autor, en el *abstract, o* en las referencias, por ejemplo.

   Al ejecutar una búsqueda pueden ser recuperados miles de documentos y la interacción con el sistema permite ver que se genera un *ranking* en la exhibición de los resultados obtenidos. El primero de estos *rankings* se ordena con base en la fecha de publicación, pero es viable que se ordenaran los documentos por la relevancia que presentan.

2. También está el portal de la *Asociation Computery Machine* (ACM) que incorpora motores de búsqueda con particulares características facilitando la labor de investigación y extracción de información ante una determinada necesidad. Esto lo decimos porque los resultados de una búsqueda son acompañados de distintas representaciones gráficas que le dan un valor adicional a la representación de los resultados. En la figura 3.1 se ve una de estas representaciones que incluye la frecuencia de aparición del *query* en el tiempo.

3.5 Modelos de Recuperación de Información:

3.5.1 Recuperación boleana:

Ante una búsqueda de información se recorre linealmente todo el documento para retornar un valor boleano indicando la presencia o no del término buscado. Es uno de los primeros modelos que se

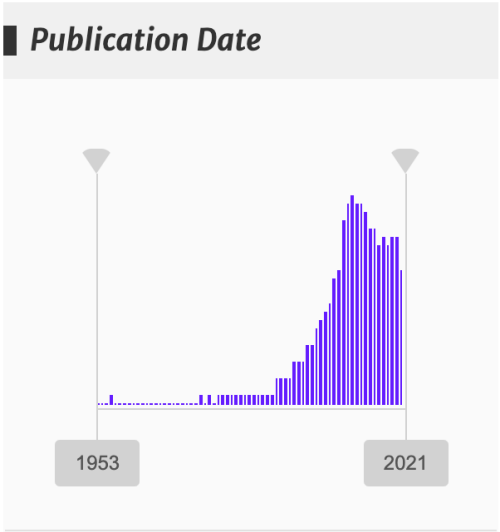

Figura 3.1: Gráfico que acompaña resultados de búsqueda de un término en la biblioteca digital de la Association for Computing Machinery (https://dl.acm.org/)

uso y está asociado a técnicas de *grepping* (Manning et al., 2008) (p.3). El desarrollo de este modelo apareció entre 1960 y 1970.

El usuario final obtendrá como respuesta a su *query* sólo aquellos textos que contengan el término. Es un modelo muy cercano a los típicos *querys* de bases de datos con el uso de operadores "AND", "OR" y "NOT". En el procesamiento de los textos se genera una matriz de incidencia binaria término-documento, donde cada término que conforma el vocabulario, ocupa una fila *i* de la matriz mientras que cada columna *j* se asocia a un documento. La presencia de el término *i* en el documento *j* se denotará con un valor verdadero o un "**1**".

La recuperación boleana si bien representa una buena aproximación a la generación de *querys* más rápidos, presenta una gran desventaja y es que al crecer la cantidad de documentos y el vocabulario, se obtiene una matriz dispersa de una alta dimensionalidad que hace poco efectiva su implementación.

Los documentos y los *querys* son vistos como conjuntos de términos indexados, que luego fueron llamados "bolsa de términos" *(bag of terms)*. Las deficiencias de este modelo recaen en que los resultados, no tienen ningún ranking. Si por ejemplo el término sobre el cual se realiza el *query* aparece 100 veces en un documento y en otro aparece sólo una vez, en la presentación de los resultados ambos documentos aparecerán al mismo nivel, no pudiendo mostrar preferencia del uno sobre el otro.

Otra de las desventajas es que no se registra el contexto semántico de las palabras, incluso se pierde el orden en que aparecen las palabras en cada texto.

Este modelo se presume que en el cual se basa la implementación de Saber UCV y por eso es que en general se termina presentando el problema de que al usar el operador *AND* en las búsquedas exactas mencionadas en **??** se obtiene una alta ***precision*** [1] en los resultados pero un bajo ***recall*** [2] mientras que al usar el operador *OR* da una baja ***precisión*** y un gran ***recall***.

[1] Precision: la fracción, o porcentaje, de los documentos recuperados que son relevantes en la búsqueda efectuada.
[2] Recall: la fracción, o porcentaje, de los documentos relevantes en la colección que fueron recuperados por el sistema.

Con la propuesta del SSBSU se quiere una versión de recuperación de información que aplica métodos de mayor eficiencia, que permiten mejorar el desempeño (tiempo) y la calidad de los resultados.

3.5.2 Índices Invertidos:

Se denominan índices invertidos porque en vez de guardar los documentos con las palabras que en ellos aparecen, en estos se procede a guardar cada palabra y se indica los documentos en los cuales se encuentra y adicionalmente se puede registrar la posición en que aparece cada palabra con distintas granularidades, pudiendo ser estas: dentro del documento, del capítulo, del párrafo o de la oración. También pueden contener la frecuencia con que se presenta determinada palabra. Toda esta información nos permite mejorar los tiempos de búsqueda pero con ciertos costos.

El primero es el espacio en disco que implica guardar estos datos adicionales, que puede oscilar del 5% al 100% del valor inicial de almacenamiento, mientras que el segundo costo lo representa el esfuerzo computacional de actualizarlos una vez que se incorporan nuevos documentos (Mahapatra and Biswas, 2011).

Existen diversos tipos de **Índices Invertidos** y constantemente se están realizando investigaciones que permitan mejorar su desempeño motivado en que sobre ellos recae gran parte de la efectividad que podamos obtener ejecutando los *querys*. Algunos ejemplos de estos índices son el *Generalized Inverted Index* (GIN), también está el RUM [3] o el VODKA [4]que es otra implementación con menos literatura sobre posibles usos pero con métodos disponibles para su uso en manejadores de base de datos como PostgreSQL.

El espacio que ocupa la implementación de estos índices se puede ver afectado, por un lado se tiene que se puede reducir mediante el preprocesamiento que hagamos a las palabras buscando su raiz con el stemming {#steaming} o removiendo las stop words (las palabras que no generan mayor valor semántico como: la, el, tu).

Por otra parte el peso total se puede incrementar a medida que decidamos tener una granularidad más fina en el registro de las palabras y su ubicación dentro de los documentos. En el transcurso del desarrollo de nuestra investigación se indicará en cuánto se incremento el espacio de almacenamiento en disco con la aplicación de este índice y la granularidad que se adoptó junto con el valor del costo en espacio de almacenamiento.

Continuando con los índices inversos hay estrategias que significan la adopción de generar dos índices inversos para un sistema, conteniendo uno de estos la lista de documentos y la frecuencia de la palabra, mientras que el otro registra la lista con las posiciones de la palabra.

El uso de los índices invertidos permite la denominada “búsqueda de texto completa” (*full text search*) que es uno de los pilares que sustenta a los motores de búsqueda y se entiende por este tipo de búsqueda aquella que permite encontrar documentos que contienen las palabras clave o frases determinadas en el texto del *query*. Adicionalmente se puede introducir el criterio de búsqueda de texto aproximado *(approximate text searching)*, que permite flexibilizar la coincidencia entre el texto requerido y el resultado.

[3] En el vínculo https://github.com/postgrespro/rum se tiene acceso a la explicación e implementación de este índice para PostgreSQL.

[4] este índice fue presentado en la Postgres Conference en el año 2014 https://www.pgcon.org/2014/schedule/attachments/318_pgcon-2014-vodka.pdf

En la Solución que se propone, la optimización en la generación de este índice quedará bajo la administración del propio manejador de base de datos que es *postgreSQL*.

Cuando la base de datos que registra el índice invertido crece y no es viable almacenarla en un único computador, es necesario acudir al uso de técnicas que permitan distribuir la base de datos con el uso de tecnologías como *Spark, Hadoop, Apache Storm* entre otras.

3.5.3 Scoring Model:

Es un modelo aplicado en amplias colecciones de documentos donde es necesario exclusivamente mostrar al usuario que realizó la búsqueda, una fracción de los documentos encontrados, pero es necesario que estos sean los de mayor puntuación (*score*), de acuerdo a distintos criterios que permitan determinar cuales son los que tienen mayor relevancia, como lo puede ser la cantidad de veces que se repite una palabra aparecida en el *query* dentro del documento, o la distancia de aparición de cada una de las palabras que conforman el query (cantidad de palabras que median entre una y otra).

También se puede asignar un peso mayor en la generación del ranking, si una, o varias, de las palabras que generan el *query* aparece dentro del título, o en otros campos que se registrasen en la base de datos.

3.6 Procesamientos de texto:

En esta sección mostramos métodos de trabajo con los textos. Para el idioma español no son de abundante literatura, a diferencia de aquellos que están en el idioma inglés. Incluso hasta hace unos pocos años las herramientas computacionales para el procesamiento de los textos **??**nlproc) tampoco eran abundantes y sabiendo que son justamente los textos, el insumo que recibe de nuestro sistema de recuperación de información, la calidad en los procesamientos que sobre ellos hagamos, marcarán en gran medida la propia calidad del sistema que tengamos.

Como decíamos la literatura y herramientas disponibles para el NLP en el idioma español, fueron escasas durante un considerable período. Se tenían disponibles algunas como el coreNLP de la Universidad de Stanford pero no incluía todas las utilidades, tales como la identificación de parte del discurso *(Part of Speech Tagging),* ni el análisis morfológico (*Morphological Analysis)* o el reconocimiento de entidades nombradas (*Named Entity Recognigtion)*, sino algunas pocas como el tokenizador 3.6.1.1 y el separador de oraciones (Sentences Splitting).

Casos similares se presentaban con otras herramientas, siendo un caso aparte el esfuerzo del CLiC-Centre de Llenguatge i Computación quienes hicieron la anotación del Corpus AnCora [5] . También la Universidad Politécnica de Cataluña creó la herramienta FreeLing [6] que permitió realizar algunas de las funcionalidades mencionadas en el párrafo anterior. No obstante su integración en cadenas

[5] **AnCora** es un corpus del **catalán (AnCora-CA)** y del **español (AnCora-ES)** con diferentes niveles de anotación como lema y categoría morfológica, constituyentes y funciones sintácticas, estructura argumental y papeles temáticos, clase semántica verbal, tipo denotativo de los nombres deverbales, sentidos de WordNet nominales, entidades nombradas (NER), relaciones de correferencia (http://clic.ub.edu/corpus/es/ancora)

$$< http://clic.ub.edu/corpus/es/ancora >$$

[6] https://nlp.lsi.upc.edu/freeling/node/1

de trabajo y la actualización de sus modelos de entrenamiento, presentan rezagos en comparación a otros modelos que actualmente se están usando, los cuales serán evaluados en el transcurso de esta investigación y serán señalados en un capítulo que aún está por desarrollar.

3.6.1 Procesamiento del Lenguaje Natural (natural language processing- NLP):

Son las técnicas computacionales desarrolladas para permitir al computador "comprender" el significado de los textos de lenguaje natural. En esta Investigación son aplicadas una serie de estos métodos sobre los textos que componen el Corpus y se pueden mencionar las siguientes:

3.6.1.1 Tokenizador:

Básicamente es separar el documento en palabras, o unidades semánticas que tengan algún signficado a las cuales se le llaman *tokens*. Para el idioma español no representa un mayor reto, ya que se puede usar el espacio como delimitador de palabras, no así en otros idiomas como el chino donde el problema se aborda de manera distinta.

Al obtener las palabras como entidades separadas de un texto nos permite calcular la frecuencia de uso de las mismas.

Es común que las librerías de procesamiento de lenguaje natural contengan tokenizadores que presentan un 100% como métrica de precisión en el idioma español.

3.6.1.2 Entidades Nombradas *(named entity reconigtion-NER)*:

Son los procesos que permiten la extracción de las distintas entidades contenidas dentro de los textos. Las entidades son: nombres, lugares, organizaciones. También permite detectar relaciones entre entidades. Las medidas de precisión en los módulos de NER alcanzan una medida cercana al 89% en modelos entrenados con *machine learning*, tal es el caso de spacy, que es una de las librerías propuestas para realizar estos procesamientos en nuestro desarrollo.

3.6.1.3 Etiquetado de Partes del Discurso *(Part of speech tagging-POS)*:

Una de las técnicas usadas en el Procesamiento del Lenguaje Natural es el part-of-speech (POS) y consiste en asignar un rol sintáctico a cada palabra dentro de una frase (Eisenstein, 2019) siendo necesario para ello evaluar cómo cada palabra se relaciona con las otras que están contenidas en una oración y así se revela la estructura sintáctica.

Los roles sintácticos principales de interés en la elaboración de esta Investigación son los sustantivos, adjetivos y verbos.

- Los sustantivos tienden a describir entidades y conceptos.
- Los verbos generalmente señalan eventos y acciones.
- Los adjetivos describen propiedades de las entidades

Igualmente dentro del POS se identifican otros roles sintácticos como los adverbios, nombres propios, interjecciones entre otros.

El POS es un procesamiento que sirve de insumo para la coocurrencia de palabras, que es una de las formas en que se representan los resultados de los *querys* en la SSCSU.

En el estado del arte este etiquetado alcanza un 98% de precisión.

3.6.1.4 Stemming:

Stemming es un algoritmo que persigue encontrar la raíz de una palabra, teniendo como el de mayor uso el Algoritmo de Porter. Al ser usado se puede reducir considerablemente el número de palabras que conforman el vocabulario del *Corpus* y se pueden mejorar los tiempos en que se ejecuta la búsqueda de un texto ya que se disminuye el espacio de búsqueda. La aplicación de este tipo de algoritmos no toma en consideración el contexto en el que aparece la palabra a la que se le extrae la raíz. Como ejemplo se muestra que "yo canto, tú cantas, ella canta, nosotros cantamos, ellos cantan" tendrá como raíz **cant**.

Es necesario considerar que al crear el **índice invertido** 3.6.1.4 son las raíces las que se guardarán y no propiamente la palabra que aparece en el texto

3.6.1.5 Lematización:

Es el proceso en que se consigue el lema de una palabra, entendiendo que el lema es la forma que por convenio se acepta como representante de todas las formas flexionadas de una misma palabra.

Al buscar el lema se tiene presente la función sintáctica que tiene la palabra, es decir que se evalúa el contexto en el que ocurre. Una de las ventajas de aplicar esta técnica es que se reduce el vocabulario del Corpus y eso conlleva a que también se reduce el espacio de búsqueda en los documentos.

En el estado del arte este etiquetado alcanza un 96% de precisión.

3.7 Minería de Texto:

La extracción de ideas útiles derivadas de textos mediante la aplicación de algoritmos estadísticos y computacionales, se conoce con el nombre de minería de texto, analítica de texto o aprendizaje automático para textos (text mining, text analytics, machine learning from text). Se quiere con ella representar el conocimiento en una forma más abstracta y así poder detectar relaciones en los textos.

El uso de las técnicas de minería de texto ha ganado atención en recientes años motivado a las grandes cantidades de textos digitales que están disponibles. La minería de texto surge para dar respuesta a la necesidad de tener métodos y algoritmos que permitan procesar estos datos no estructurados (Aggarwal and Zhai, 2012) . Una vez que mediante el proceso de recuperación de información se tiene un cúmulo de textos, es posible que se necesite dar un paso más allá y usar un conjunto de técnicas que nos permitan analizar y digerir estos textos, mediante la detección de patrones.

Uno de los desafíos al trabajar con textos es darles estructura para que resulte viable trabajar con ellos desde la perspectiva de procesos computacionales.

Una de las primeras fases consiste en agrupar todos los textos en un Corpus. Posteriormente se procederá a conformar una matriz dispersa de una alta dimensionalidad que se denominará *"Sparce Term-Document Matrix)* de tamaño *n X d,* donde *n* es el número total de documentos y *d* es la cantidad de términos o vocabulario (palabras distintas) presentes entre todos los documentos. Formalmente se sabe que la entrada *(i,j)* de nuestra matriz es la frecuencia (cantidad de veces que aparece) de la palabra *j* en el documento *i* .

El problema de la alta dimensionalidad de la matriz mencionada motiva ir aplicando otras técnicas que en principio puedan colaborar a reducirla por medio de simplificar los atributos, es decir, disminuyendo el vocabulario por ejemplo aplicando el algoritmo de Porter (stemming).

Igualmente a nivel de minería de texto se hace deseable poder contar con la identificación semántica de cada una de las palabras que conforman nuestro vocabulario, para así obtener representaciones que aportan un mayor significado. Los procesamientos inherentes al NLP mencionados anteriormente son insumo para la minería de texto.

3.7.1 Coocurrencia de Palabras:

En esta investigación se usará una técnica denominada "Coocurrencia de Palabras" para la detección de patrones en los textos. Esto consiste en evaluar las palabras que coocurren dentro de los documentos que conforman el *corpus* y se puede hacer con distintas granularidades. Por ejemplo: las palabras que coocurren una seguida de otra o las que coocurren dentro de la misma oración, o dentro de un párrafo y así sucesivamente.

Para la representación visual se usan los grafos representando cada palabra un nodo y la coocurrencia de una palabra con otra implica que se extienda un arco entre ellas. Las palabras dispuestas para representarse en el grafo serán exclusivamente las que tengan la función dentro del discurso (POS) 3.6.1.3 de adjetivos y sustantivos, es decir que cada coocurrencia será un sustantivo con el adjetivo que la acompaña, donde es posible tener una relación de un sustantivo con {0,1,…,n} adjetivos.

La selección de las funciones gramaticales propuestas se hace para disminuir el espacio de representación y se considera que los sustantivos, al contar con el adjetivo que las acompaña, logran hacer una representación que muestra proximidad semántica y se representan los temas (*topics*) más relevantes (Segev, 2021).

En el método que se usará en este Sistema se filtrarán las *n* (*n* igual a 100), palabras que presenten mayor coocurrencia dentro de los resúmenes filtrados en el ***query***, siendo posible seleccionar la granularidad (todo el texto o en un párrafo).

En la figura 3.2 se visualiza lo expuesto de una manera gráfica al ver la representación en un grafo de la coocurrencia de palabras sobre los textos de los resúmenes de las Tesis y TEG de la Escuela de Física de la U.C.V.

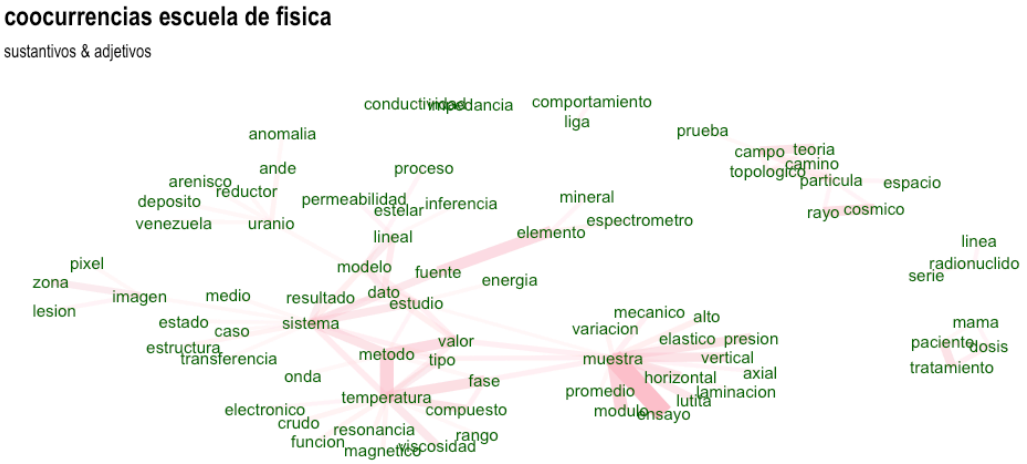

Figura 3.2: Coocurrencia de Palabras

3.8 Similitud de documentos:

Para poder realizar la recomendación de documentos, una de las técnicas que se usa es medir la similitud que presenta un documento con los otros contenidos en el corpus. Un ejemplo de esta técnica es el uso de la similitud coseno que se explica con esta fórmula.

$$\cos(\mathbf{t}, \mathbf{e}) = \frac{\mathbf{te}}{\|\mathbf{t}\|\|\mathbf{e}\|} = \frac{\sum_{i=1}^{n} \mathbf{t}_i \mathbf{e}_i}{\sqrt{\sum_{i=1}^{n} (\mathbf{t}_i)^2} \sqrt{\sum_{i=1}^{n} (\mathbf{e}_i)^2}} \tag{3.1}$$

En la fórmula *t* representa un documento y *e* representa otro documento. Ambos documentos se asumen que están en un espacio con *i* atributos, o dimensiones, y la intención es calcular un índice de similitud entre ambos documentos.

Este es uno de los métodos más usados para detectar similitudes en los textos, aunque existen otras fórmulas para el cálculo de la similitud como es el índice de jaccard.

El otro elemento de gran importancia en obtención de esta medición, es la representación que se haga del documento. Son distintas las técnicas que existen estando entre ellas la representación mediante “bolsas de palabras” o *bag of words*. Recientemente se han creado formas más complejas para la representación como lo son los *words embeddings* que son obtenidos mediante el entrenamiento de redes neuronales de aprendizaje profundo.

Estas formas de representación de textos serán investigadas más a fondo en futuras fases de la elaboración de este trabajo.

3.9 Sistemas Distribuidos:

Los distintos procesos y componentes de la Solución propuesta han sido diseñados e implementados como un sistema distribuido y por eso se hace la mención a este tema.

Una definición formal que se le puede dar a los sistemas distribuidos es “cuando los componentes de hardware y/o sofware se encuentran localizados en una red de computadores y estos coordinan sus acciones sólo mediante el pase de mensajes” (Coulouris, 2012).

Algunas de las principales características que tienen los sistemas distribuidos es la tolerancia a fallos, compartir recursos, concurrencia, ser escalables (Czaja, 2018) entre otras. Mencionamos estas en particular al ser propiedades que están presentes en la propuesta acá descrita.

1. Fiabilidad o la llamada tolerancia a fallos: en caso de fallar un componente del sistema los otros se deben mantener en funcionamiento.
2. Compartir recursos: un conjunto de usuarios pueden compartir recursos como archivos o base de datos.
3. Concurrencia: poder ejecutar varios trabajos en simultáneo.
4. Escalable: al ser incrementada la escala del sistema se debe mantener en funcionamiento el sistema sin mayores contratiempos.

3.9.1 Contenedores:

Un contenedor es una abstracción de una aplicación que se crea en un ambiente virtual, en el cual se encuentran “empaquetados” todos los componentes (sistema operativo, librerías, dependencias, etc.), que una aplicación necesita para poder ejecutarse. En su diseño se tiene presente que sean ligeros y que con otros contenedores pueden compartir el *kernel*, usando un sistema de múltiples capas, que también pueden ser compartidas entre diversos contenedores, ahorrando espacio en disco del *host* donde se alojan los contenedores.

El uso de los contenedores permite crear, distribuir y colocar en producción aplicaciones de software de una forma sencilla, segura y reproducible. También a cada contenedor se le puede realizar una asignación de recursos (memoria, cpu, almacenamiento) que garantice un óptimo funcionamiento de la aplicación que contienen.

Es importante señalar que el uso de esta tecnología añade un entorno de seguridad al estar cada contenedor en una ambiente isolado.

Para cada contenedor es necesario usar una imagen donde se definen las dependencias necesarias para su funcionamiento.

3.9.2 Orquestador:

Al tener diversos contenedores, donde cada uno aloja una aplicación distinta, puede resultar necesario que todos se integren en un sistema. Para que esta integración sea viable es necesario contar con un orquestador. Su uso permitirá lograr altos grados de portabilidad y reproducibilidad,

pudiendo colocarlos en la nube o en centros de datos garantizando que se pueda hacer el *deploy* de forma sencilla y fiel a lo que se implementó en el ambiente de desarrollo.

En el caso de la Solución propuesta se adoptará el uso de Docker Compose como orquestador y en el Capítulo que contiene la Propuesta Técnica 2.2 serán expuestas las funcionalidades de cada contenedor y se apreciará la integración que brindará el orquestador.

3.10 Tendencias actuales Sistemas de Información:

Si bien anteriormente las búsquedas de información dentro de un conjunto de textos se procesaban determinando la aparición o no de palabras, o de frases dentro de un determinado texto, este método ha ido evolucionando para llegar hoy en día a un elevado nivel de abstracción, donde a partir de la necesidad de obtener una determinada información, es decir, de aquello que necesitamos buscar, que antes consistía en hacer *match* con un objeto de información, se ha pasado de los motores de búsqueda (*search engines*) a los motores de respuestas (*answering engines*) (Balog, 2018), donde el sistema ante un determinada consulta del usuario va a retornar una serie de resultados enriquecidos, mostrando la identificación de entidades, hechos y cualquier otro dato estructurado que esté de forma explícita, o incluso implícita, mencionado dentro de los textos que conforman el corpus.

Para lograr la generación de estos resultados se han tenido que conformar las llamadas bases de conocimiento o *knowlodege bases*, que son repositorios donde previo a la búsqueda de la información dentro del sistema, se logra ir estructurando la organización de la información alrededor de objetos o datos específicos que se denominan **entidades**. Estos conceptos y métodos se asocian directamente a los que también se proponen, de manera más amplia, en la denominada *web semántica* [7] .

Como ejemplo de una *knowledge bases* se puede mencionar a DBpedia, que se encuentra en la dirección www.www.dbpedia.org.com y es un proyecto en donde puede acceder a una red global y unificada de grafos de conocimiento, la cual cubre más de 20 idiomas y principalmente genera sus grafos de conocimiento a partir de las publicaciones del Proyecto Wikipedia.

3.10.0.1 Temas en proceso de investigación:

Por estar aún en curso esta investigación a continuación se mencionan algunos temas que se está evaluando incluir en este marco teórico

- Métodos usados para el almacenamiento o la indexación como crear agrupamientos *(clusters)* de aquellos documentos que compartan algunas características, por ejemplo, en la temática que aborden. Otra de las innovaciones que se están añadiendo a los sistemas de recuperación de información, es generar resúmenes con técnicas de procesamiento de lenguaje natural soportadas en el uso de arquitecturas de aprendizaje profundo *(deep learning)* .

[7] Se basa en la idea de añadir metadatos semánticos y ontológicos a la *World Wide Web*. Esas informaciones adicionales —que describen el contenido, el significado y la relación de los datos— se deben proporcionar de manera formal, para que así sea posible evaluarlas automáticamente por máquinas de procesamiento. Tomado de Wikipedia

- Resultados ante una búsqueda personalizados: al existir mecanismos como la sesión de usuario o las *cookies* que guardan información contextual, permitiendo que ante un mismo *query* en distintos equipos, los resultados sean distintos en función de la persona que hizo la búsqueda.
- En cuanto al estado del arte existen distintas librerías y modelos de representación de documentos, palabras y caracteres. Algunos de estos modelos son fastext, word2vec, GPT3. Para este momento aún estamos haciendo la evaluación del uso de ellos para nuestra investigación, por lo cual, sólo los mencionamos referencialmente en este Anteproyecto.

Capítulo 4

Marco Metodológico:

Capítulo 5

Desarrollo de la Solución:

Capítulo 6

Conclusiones:

Bibliografía

Aggarwal, C. C. and Zhai, C., editors (2012). *Mining text data*. Springer, New York Heidelberg. DOI: 10.1007/978-1-4614-3223-4.

Balog, K. (2018). *Entity-Oriented Search*. Number 39 in The Information Retrieval Series. Springer International Publishing : Imprint: Springer, Cham, 1st ed. 2018 edition. DOI: 10.1007/978-3-319-93935-3.

Coulouris, G. F., editor (2012). *Distributed systems: concepts and design*. Addison-Wesley, Boston, 5th ed edition.

Czaja, L. (2018). *Introduction to Distributed Computer Systems: Principles and Features*. Number 27 in Lecture Notes in Networks and Systems. Springer International Publishing : Imprint: Springer, Cham, 1st ed. 2018 edition. DOI: 10.1007/978-3-319-72023-4.

Eisenstein, J. (2019). *Introduction to natural language processing*. Adaptive computation and machine learning. The MIT Press, Cambridge, Massachusetts.

Fredkin, E. (1960). Trie memory. *Communications of the ACM*, 3(9):490–499.

Goodrich, M. T., Tamassia, R., and Goldwasser, M. H. (2013). *Data structures and algorithms in Python*. Wiley, Hoboken, NJ.

Knuth, D. E. (1997). *The art of computer programming*. Addison-Wesley, Reading, Mass, 3rd ed edition.

Kraft, D. H. and Colvin, E. (2017). Fuzzy information retrieval. *Synthesis Lectures on Information Concepts, Retrieval, and Services*, 9(1):i–63.

Mahapatra, A. K. and Biswas, S. (2011). Inverted indexes: Types and techniques. *International Journal of Computer Science Issues*, 8.

Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to information retrieval*. Cambridge University Press, New York. OCLC: ocn190786122.

Segev, E. (2021). *Semantic Network Analysis in Social Sciences*. Routledge.