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.

Palabras Clave:: sistemas de recuperación de información, procesamiento del lenguaje natural, búsqueda de texto completo, mapas del conocimiento, sistemas distribuidos.

Abstract:

A proposal is presented for a distributed web application to implement an Information Retrieval System (*information retrieval*) using Natural Language Processing (NLP) techniques, database indexing and visualizations with graphics and graphs of word cooccurrences for the special undergraduate and graduate works done by the students of the Universidad Central de Venezuela that are hosted in the repository www.saber.ucv.ve.

This proposal is mainly based on the search for words or phrases to generate a (*query*), and by means of the technique known as "*full text search*" the works containing such words can be extracted and from there enrich the user's experience with the presentation of the retrieved information.

The application is designed as a distributed system in different containers, each one supporting a process for the operation, having among the main ones the database, the application server and another one with the different processes that are carried out on the texts.

Keywords: information retrieval, natural language processing, full text search, knowledge maps, distributed systems.

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.

Como todos los hombres de la Biblioteca, he viajado en mi juventud; he peregrinado en busca de un libro, acaso del catálogo de catálogos; ahora que mis ojos caso no pueden descifrar lo que escribo, me preparo a morir a unas pocas leguas del hexágono en que nací.
— Jorge Luis Borges, *La Bibioloteca de Babel*, Ficciones

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:

Es conocida la gran disponibilidad de información en distintos formatos que tienen a su disposición los investigadores. Ejemplo de esto son libros en las bibliotecas, o la variedad de documentos que se encuentran accesibles en formatos digitales como artículos publicados en revistas arbitradas, libros, páginas de internet especializadas en algún tema o los repositorios digitales de documentos. Es en esta abundancia donde recaen varios de los problemas a los cuales se enfrenta la labor investigativa, en particular, cuando se realiza la fase exploratoria de selección de aquellos documentos que puedan resultar de mayor relevancia.

Una de las herramientas con que cuentan los investigadores, para la búsqueda de documentos que se encuentran en formato digital, son los Sistemas de Recuperación de Información, donde ante una búsqueda, a la cual denominaremos el ***query***, serán recuperados del conjunto de documentos, conjunto al que denominaremos ***Corpus***, aquellos que cumplan ciertos parámetros(Manning et al., 2008). Los documentos que el Sistema logre recuperar se deben adaptar a las necesidades del investigador y en caso de existir múltiples documentos, el sistema debe poder jerarquizar aquellos que puedan resultar de mayor relevancia.

En el caso de la Universidad Central de Venezuela existe un repositorio digital de documentos denominado SABER.UCV al cual se puede acceder desde la dirección web saber.ucv.ve el cual permite realizar búsquedas de información sobre documentos generados dentro de la comunidad de investigadores de la Universidad.

En este Trabajo de Grado se presenta la propuesta para la **Recuperación, Extracción y Clasificación de Información de SABER UCV**, mediante el desarrollo de un sistema informático que se denomina **Sistema Complementario Saber UCV (SCSU)**, el cual complementa las funcionalidades de búsqueda de información que actualmente tiene el repositorio oficial de la Universidad Central de Venezuela.

La implementación del SCSU hace un aporte a los investigadores que necesitan realizar búsquedas de información sobre el subconjunto de tesis doctorales, de trabajos de grado de maestría y de pregrado, que se encuentran disponibles en SABER.UCV . Adicionalmente, la representación de los resultados de los *querys* son enriquecidas con gráficos interactivos que muestran “mapas del conocimiento” (citar a dueñas).

Gran parte de las funcionalidades que incorpora el SCSU se sustentan en los procesos de extracción y clasificación de información de datos que originalmente no se encuentran estructurados en el

repositorio oficial, como lo son los nombres de las carreras de pregrado o del postrgrado donde fue generada cada una de las investigaciones o el nombre del tutor que acompañó el desarrollo de cada investigación, así que el Sistema acá descrito contribuye efectivamente a que estos datos puedan estar disponibles y afinar con una mayor cantidad de parámetros las búsquedas de información que se deban realizar y se incorporan técnicas de búsqueda indexada mediante un índice inverso que establece los criterios de relevancia al momento de jerarquizar y mostrar los resultados de una búsqueda.

1.1 Estructura:

En el Capítulo 2 **El Problema** se hace una exposición de cuáles son las funcionalidades de SABER.UCV que pueden ser complementadas por el Sistema Complementario Saber UCV mientras que en 2.2 se establecen las delimitaciones en el proceso de investigación abordado. En 2.3 se justifica la necesidad de realizar este estudio y en 2.4 se presenta la Solución propuesta junto con el 2.5.1 Objetivo General y los 2.5.2 Objetivos Específicos . En 2.6 se enuncia el alcance del proceso de recuperación, extracción y clasificación de Información de SABER UCV y en 2.7 se enumeran los aportes que brinda esta investigación.

En el Capítulo 3 **Marco Teórico** se hace una reseña histórica 3.1 sobre el problema computacional que es la "búsqueda", entendido de manera genérica. En la sección 3.2 "Recuperación de Información" se mencionan los conceptos para comprender los procesos de búsqueda de texto dentro de un Corpus y también se introducen a los "Sistemas" que permiten ejecutar dicha tarea. Luego en 3.2.3 se describen algunos "Modelos de Recuperación de Información" que se usan en la implementación del SCSU. Para representar los resultados de las búsquedas, el Sistema implementado necesita que los textos sean procesados mediante diversos métodos que son descritos en en 3.3 Procesamiento de Texto, como los son el 3.3.1 Procesamiento del Lenguaje Natural y 3.3.2 la Minería de Texto . Seguidamente en 3.4 se conceptualiza a los "Sistemas Distribuidos", motivado a que la implementación realizada del Software se hace mediante este tipo de sistemas.Al finalizar este Capítulo en 3.5 se presentan el Estado del Arte en lo relativo a los procesos de Recuperación de Información.

En el Capítulo **?**(mm) **Marco Metodológico** se presenta la metodología de trabajo Kanban 4.1 que sirvió para realizar la planificación de la investigación así como también el Desarrollo Adaptable de Software –Adaptive Software Development (ASD)- 4.2 que fue la metodología adaptada para realizar los ciclos de desarrollo del SCSU.

En el Capítulo 5 **Desarrollo** se presenta la 5.1 Descripción general de la Solución y en 5.2 se presenta la Arquitectura de la misma. Luego en 5.3 Análisis y Diseño se muestran las consideraciones adoptadas para dicho proceso. En 5.4 Ciclos de Desarrollo se muestran las iteraciones que se hicieron para crear el SCSU; en un primer ciclo para la 5.4.1 Obtención del Conjunto de Datos , en un segundo ciclo 5.4.2 para la Extracción y Clasificación de Información del Conjunto de Datos , en el tercero 5.4.3 para crear el Prototipo de Buscador y en el cuarto 5.4.4 se realizó la Integración de Componentes del Software . En este Capítulo adicionalmente se realizan las distintas 5.5 Pruebas tanto 5.5.1 Funcionales, 5.5.2 de Rendimiento y en 5.5.3 las pruebas de Relevancia sobre la información recuperada al momento de hacer una búsqueda.

Para concluir esta investigación, en el Capítulo 6 **Conclusiones** se exponen las 6.1 Contribuciones

y los 6.2 Trabajos Futuros que se pueden derivar del proceso expuesto a lo largo del contenido de los capítulos anteriormente citados.

Capítulo 2

El Problema:

2.1 El Problema:

2.2 Delimitación del Problema

2.3 Justificación e Importancia:

2.4 Descripción de la Solución:

2.5 Objetivos de la Investigación:

2.5.1 Objetivo General:

2.5.2 Objetivos Específicos:

2.6 Alcance:

2.7 Aporte:

Capítulo 3

Marco teórico-referencial:

En este capítulo se establece el fundamento teórico para los procesos que sustentan la Recuperación, Extracción y Clasificación de Información de SABER UCV.

Se hace una reseña histórica 3.1 sobre los procesos de búsqueda, se examinan conceptos de claves 3.2 como indexación, búsqueda, relevancia y evaluación de resultados. Además, se exploran modelos como el modelo boleano y vectorial junto se introduce el Estado del Arte. Para indicar los métodos con que son 3.3 procesados los textos se muestran las técnicas de 3.3.1 Procesamiento de Lenguaje Natural y de 3.3.2. Adicionalmente al formar parte de este Trabajo el Sistema Complementario Saber UCV que es un software, se exponen definiciones asociadas a los 3.4 Sistemas Distribuidos para que así este análisis profundo siente las bases para el diseño y desarrollo del sistema, asegurando una comprensión sólida de los principios subyacentes

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.

No obstante con el desarrollo e incremento 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).

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, 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 5.4.2.

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 de cadenas del ADN* de forma parcial o total dentro de una cadena mayor, siendo una versión de estos el algoritmo ***Smith-Waterman*** (Smith and Waterman, 1981). Posteriormente se identificó que este tipo de solución era extrapolable a las subcadenas de texto.

Un algoritmo que se usó para resolver el problema de hacer coincidir una etiqueta de clasificación con los textos del *Corpus* en 5.4.2 fue el algoritmo precitado.

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 del SCSU todos los textos van siendo procesados con distintas técnicas a medida que son insertados en la base de datos.

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) 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).

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, similar a la propuesta del algoritmo ***PageRank***(Brin and Page, 1998), siendo el fin último, la extracción de los documentos que resulten de mayor relevancia para el usuario. Esta aproximación también puede usarse para la detección de comunidades dentro del *Corpus* (Heydari and Teimourpour, 2020).

Incluso es válido incorporar documentos, en los resultados que arroje la búsqueda, que propiamente no coincidan exactamente con los términos buscados sino que contengan palabras 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 el crecimiento exponencial de datos disponibles en formato digital (wor, 2016), potenciada por el uso generalizado de los computadores desde la década de 1980 , abonó las condiciones para la creación de los **Sistemas de Recuperación de Información**.

3.2.1 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) . 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 SCSU está diseñado como un ISRI 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.

3.2.2 Ejemplos de IRS:

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

1. 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.), si se quiere buscar sólo dentro del título de una investigación, o el nombre autor, en el *abstract, o* en las referencias.
2. Portal de la *Asociation Computery Machine* (ACM) alojado en https://dl.acm.org incorpora un motor de búsqueda con particulares características ya que 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.

3.2.3 Modelos de Recuperación de Información:

3.2.3.1 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 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”,

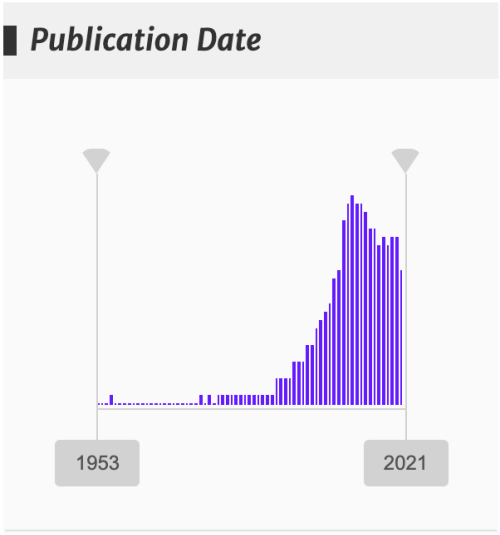

Figura 3.1: Gráfico que acompaña resultados de búsqueda de un término en la biblioteca digital de la Association for Computing Machinery (https://dl.acm.org/)

"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 (palabras únicas contenidas dentro del Corpus), 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 e 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 OR en las búsquedas exactas mencionadas en **??**, se obtiene un gran ***recall*** [1] en los resultados.

Con la propuesta del 5.4.3 Prototipo de Buscador del SCSU se obtiene una versión de recuperación de información que aplica métodos de mayor eficiencia y genera una mayor ***precision*** con un menor ***recall*** , mejorando el desempeño (tiempo) y la relevancia de los resultados.

[1] Precision: la fracción, o porcentaje, de los documentos recuperados que son relevantes en la búsqueda efectuada.

3.2.3.2 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 [2] o el VODKA [3]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)*, donde se flexibiliza la coincidencia entre el texto requerido y el resultado.

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

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

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

uso de tecnologías como *Spark, Hadoop, Apache Storm* entre otras. En el trabajo de (Mahapatra and Biswas, 2011) se encuentran detalles adicionales sobre este tipo de índices.

En **??** se muestra el estado del arte en los Sistemas de Recuperación de Información al incorporar representaciones de embeddings (Reimers and Gurevych, 2019) para los textos y su uso como un Modelo de Recuperación de Información.

3.2.4 Re Ordenamiento (re-ranking):

Es una técnica utilizada para mejorar la precisión y lograr extraer los documentos que tengan mayor relevancia en los resultados de una búsqueda. Cuando los usuarios realizan el query a menudo se encuentran con una gran cantidad de documentos que coinciden con sus consultas. Sin embargo, no todos estos documentos son igualmente relevantes para el usuario. Por lo tanto, el re-ranking implica reorganizar los resultados de búsqueda originales para que los documentos más relevantes aparezcan en las primeras posiciones, mejorando así la experiencia del usuario.

3.2.4.1 Learning to Rank (LTR):

Los algoritmos de aprendizaje para la clasificación (LTR, por sus siglas en inglés) son comúnmente utilizados para el re-ranking. Estos algoritmos utilizan técnicas de aprendizaje automático para modelar la relevancia de los documentos basándose en características específicas. Los atributos pueden incluir la frecuencia de palabras clave, la proximidad de términos en el documento y otros factores que indican la relevancia. Los modelos LTR pueden ser entrenados con conjuntos de datos que contienen consultas y documentos etiquetados con su relevancia, y luego aplicados para re-ordenar los resultados de búsqueda en función de las características aprendidas.

3.2.4.2 BM25:

Es un algoritmo que contiene una función de puntuación utilizada para calcular la relevancia de un documento con respecto a una consulta de búsqueda y es una mejora del modelo probabilístico Okapi BM, y ha demostrado ser efectivo en la práctica para clasificar documentos según su relevancia con las consultas de los usuarios. Se basa en la frecuencia de los términos de búsqueda y la longitud del documento. A diferencia de los modelos clásicos como TF-IDF, BM25 ajusta la importancia de la frecuencia del término y la longitud del documento mediante una fórmula matemática compleja (Zhai and Massung, 2016), lo que lo hace más eficaz para lidiar con variaciones en la longitud del documento y mejorar la precisión en los resultados de búsqueda.

3.3 Procesamientos de texto:

En esta sección mostramos métodos de manipulación y tratamiento de los textos. Lo primero que se indica es que hasta el año 2016 eran escasas las herramientas computacionales para el procesamiento de los textos **??**nlproc) en el idioma español. 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.

Frameworks para tareas de procesamientos de texto se basan en los proyectos de Universal Dependencies (de Marneffe et al., 2021), como el coreNLP de la Universidad de Stanford (Manning et al., 2014) que fue uno de los primeros sistemas en incluir procesamientos para el idioma español, sin incluir todas las utilidades sí disponibles para el idioma inglés como el identificación de parte del discurso *(Part of Speech Tagging),* ni el análisis morfológico (*Morphological Analysis)* (Straka and Straková, 2017) o el reconocimiento de entidades nombradas (*Named Entity Recognigtion)*, sino algunas pocas como el tokenizador 3.3.1.1 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 [4] . También la Universidad Politécnica de Cataluña creó la herramienta FreeLing [5] que permitió realizar algunas de las funcionalidades mencionadas en el párrafo anterior. No obstante su integración en cadenas de trabajo y la actualización de sus modelos de entrenamiento, presentan rezagos en comparación a otros modelos que actualmente se están usando basados en el uso del aprendizaje mediante redes neuronales (Chen and Manning, 2014) y que serán indicados con mayor detalle en 3.5 .

3.3.1 Procesamiento del Lenguaje Natural (Natural Language Processing- NLP):

Son las técnicas computacionales desarrolladas para permitir al computador representar e interactuar de una forma más efectiva con el "significado" de los textos . Al aplicar la tokenización 3.3.1.1 , el Etiquetado de Partes del Discurso 3.3.1.2, el stemming 3.3.1.3, la lematización 3.3.1.4 , entre otros métodos, se desea obtener un Corpus Anotado (Desagulier, 2017). Los métodos que se detallan a continuación fueron aplicados sobre el Corpus del SCSU.

3.3.1.1 Tokenizador:

Básicamente es separar el documento en palabras, o unidades semánticas que tengan algún signficado a las cuales se le llaman *tokens* (Straka and Straková, 2017). 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, por ejemplo, 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.

[4] **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)

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

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

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

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.

3.3.1.3 Stemming:

Stemming es un algoritmo que persigue encontrar la raíz de una palabra, teniendo como el de mayor uso el Algoritmo de Porter (Willett, 2006). 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** 3.2.3.2 son las raíces las que se guardarán y no propiamente la palabra que aparece en el texto.

3.3.1.4 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 (de Marneffe et al., 2021). Los lemas, o lexemas, constituyen la parte principal de la palabra, la que transmite el significado. Los morfemas son el elemento variable de la palabra y son los que se busca desechar en el proceso de lematización.

Tener presente que hay que revisar las menciones a la lematización y explicar que va en encontrar los lexemas, o que es un proceso similar.

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 en esta tarea en varios de los modelos de aprendizaje automático preentrenados para realizarla, no obstante no se disponen datos puntuales para la precisión que se alcanza en el idioma español.

3.3.2 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 (Aggarwal, 2018a).

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) y ha ganado atención en recientes años motivado a las grandes cantidades de textos digitales que están disponibles. Los procesamientos inherentes al NLP mencionados anteriormente son insumo para la minería de texto.

Algunos de los métodos que pertenecen a la Minería de Texto son:

3.3.2.1 Term-Document Matrix:

Una vez que se tiene conformado un Corpus, se procede 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 . Este procedimiento es similar al que fue revisado en 3.2.3.1.

Uno de los problemas que presenta la matriz obtenida es la alta dimensionalidad y lo dispersa que es, llegando a estar conformada en un 98% por ceros, que indican la ausencia de la aparición de una palabra en un determinado documento.

Para mejorar un tanto este tipo de representación del Corpus, se aplican otras técnicas, que en principio puedan colaborar a reducir la dimensionalidad, por medio de simplificar los atributos, es decir, disminuyendo el vocabulario aplicando el stemming 3.3.1.3.

3.3.2.2 Coocurrencia de Palabras:

En esta investigación se usará un método denominado "Coocurrencia de Palabras" para la detección de patrones en los textos y se hará la representación de aparición de las coocurrencias mediante grafos.

El método se explica en que se evaluan las palabras que coocurren, es decir, aquellas que forman parte del conjunto de palabras obtenidas de la intersección de los documentos que conforman el *corpus,* o del subconjunto de documentos recuperados mediante un determinado *query*.

También se puede establecer el nivel al que se quiere determinar la coocurrencia, por ejemplo, las palabras que coocurren una seguida de otra en los textos, o las que coocurren dentro de la misma oración, o dentro de un párrafo o dentro de todo el texto de cada documento.

Para la representación visual se usan los grafos donde cada palabra representa 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) 3.3.1.2 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 (*topicos*) 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 textos 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.

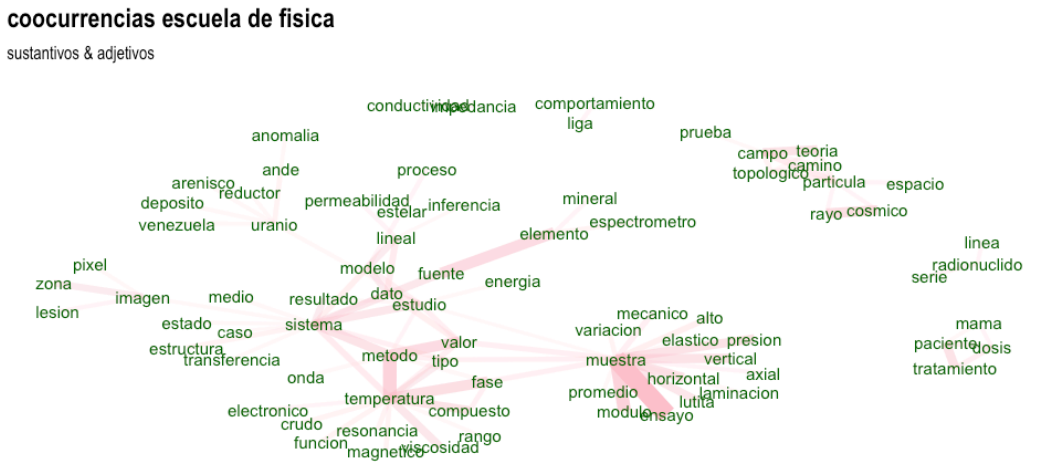

Figura 3.2: Coocurrencia de Palabras

La representación gráfica y el método de extracción de sustantivos y adjetivos, resulta similar a la propuesta metodológica realizada por (Dueñas et al., 2011) para crear Mapas del Conocimiento con “las palabras claves obtenidas a través de búsquedas recurrentes y relacionadas”. En esta Investigación se simplificará la obtención y representación de los Mapas del Conocimiento, asumiendo que las palabras claves son los sustantivos adjetivizados, equivalente a visualizar las personas, cosas o ideas que se mencionan y que son modificados por los adjetivos, al cambiar sus propiedades o atributos; seleccionando aquellas palabras que muestran una mayor aparición en el query realizado y que se interconectan mediante arcos.

3.3.3 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 (Aggarwal, 2018b) . 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.

Al hacer la comparación de un documento i del Corpus que contiene n documentos, en un proceso iterativo con otra cantidad de (*n-1)* documentos, se obtendrán (*n*-1) índices de similitud. Aquel que obtenga un mayor valor se puede inferir que presenta una mayor similitud con el documento *i.*

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,* similar a lo que se explicó en 3.3.2.1 donde un documento i es el vector correspondiente a una fila de la matriz y la cantidad de dimensiones que presenta es equivalente al tamaño del vocabulario.

Recientemente se han creado formas más complejas para la representación de los documentos, como lo son los *words embeddings* que son obtenidos mediante el entrenamiento de redes neuronales de aprendizaje profundo lo cual será expuesto a continuación en 3.5.

3.4 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.

3.4.1 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 (Nüst et al., 2020).

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 previamente se definen las dependencias (sistema operativo, librerías, lenguajes) necesarias para su funcionamiento.

3.4.2 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 (Cook, 2017). 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 5.4.4 serán expuestas las funcionalidades de cada contenedor y se apreciará la integración que brindará el orquestador.

3.5 Estado del Arte:

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.

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)* .

Capítulo 4

Capítulo Marco Metodológico:

En este capítulo, se presenta el enfoque metodológico adoptado para este estudio. La metodología Kanban 4.1 se empleó para gestionar el proceso general, mientras que la metodología de Desarrollo Adaptable de Software 4.2 guió la creación del software, permitiendo una implementación eficiente y adaptable a las necesidades cambiantes del proyecto **Recuperación, Extracción y Clasificación de Información de SABER UCV**.

4.1 Metodología de Trabajo:

Para el desarrollo del Sistema se adoptó la metodología Kanban por considerarse idónea ante los retos propuestos. En ella se fomenta una cultura de mejora continua, incremental, al identificar cuellos de botella, la limitación del trabajo en curso y la mejora continua para aumentar la eficiencia y la productividad. Gracias a su enfoque basado en la colaboración, flexibilidad y respuesta rápida a los cambios puede considerarse que está dentro de las metodologías "Ágiles" y se basa en la visualización del flujo de trabajo mediante un tablero, que en el caso de los proyectos de desarrollo de software, facilita la toma de decisiones informadas y promueve la transparencia al proporcionar una visualización clara de las tareas y actividades involucradas (Stephens, 2015).

En el tablero que se aprecia en la figura 4.1 se puede visualizar y facilitar la gestión del flujo de trabajo, desde la concepción de una idea, hasta su implementación y entrega. Durante el desarrollo de este Sistema se necesitaba contar con la flexibilidad que ofrece esta metodología, ya que en ella no se tienen que definir roles específicos en el equipo desarrollador, ni tampoco se querían definir períodos fijos para alguna fase en particular sino más bien para el desarrollo general.

4.2 Desarrollo Adaptable de Software:

El **Adaptive Software Development (ASD)** (Highsmith, 2000) es una metodología ágil de desarrollo de software que se centra en la adaptabilidad y capacidad para adaptarse a los cambios, proporcionando una retroalimentación temprana y frecuente, dando la flexibilidad para abordar los desafíos cambiantes del desarrollo de software. A diferencia de los enfoques tradicionales, ASD reconoce la naturaleza impredecible del desarrollo de software y se adapta continuamente para

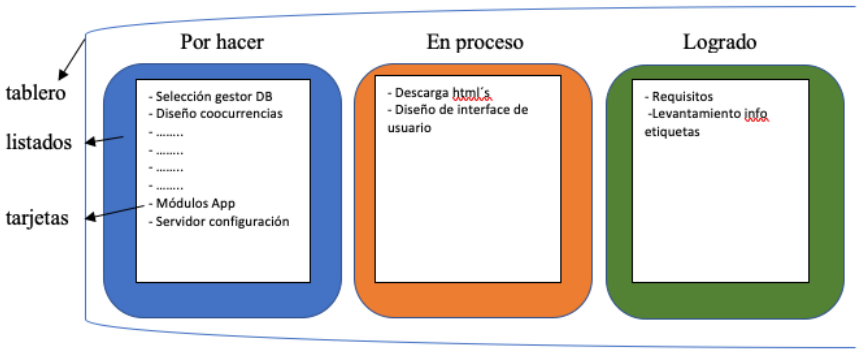

Figura 4.1: Representación de un tablero según la metodología Kanban

satisfacer las necesidades del cliente en un entorno dinámico y complejo, haciendo énfasis en el principio "Entregar el proyecto que se necesita al final, no el proyecto que se pidió al principio".

4.2.1 Características:

Colaboración y Comunicación Constante: fomenta la colaboración cercana entre los equipos de desarrollo y los stakeholders. La comunicación constante permite una comprensión profunda de los requisitos del cliente y facilita ajustes rápidos según las necesidades cambiantes.

Iteraciones Incrementales: divide el proyecto en iteraciones cortas y manejables. Cada iteración produce un incremento funcional del software, lo que permite obtener retroalimentación temprana que permite corregir errores y ajustar el rumbo del proyecto antes de que los problemas se vuelvan críticos.

Flexibilidad y Adaptabilidad: reconoce que los requisitos del proyecto pueden cambiar con el tiempo. Por lo tanto, se adapta fácilmente a los cambios, permitiendo una rápida reevaluación y ajuste de las estrategias y metas del proyecto asegurando que el producto final esté alineado de manera óptima con las necesidades y expectativas del cliente, incluso en un entorno de desarrollo volátil.

4.2.2 Ciclo:

El desarrollo adaptable de software se basa en un proceso dinámico e iterativo donde cada ciclo contiene las siguientes fases: Especular-Colaborar-Aprender. Cada ciclo se enfoca en el aprendizaje continuo y la colaboración intensiva entre desarrolladores y clientes, fundamental para enfrentar las cambiantes dinámicas empresariales.

4.2.2.1 Especulación:

Este componente ofrece un espacio para la exploración y la comprensión de la incertidumbre. Permite desviarse del plan inicial sin temor, transformando los errores en oportunidades de aprendizaje. Aceptar que no se sabe todo impulsa la disposición para aprender y experimentar.

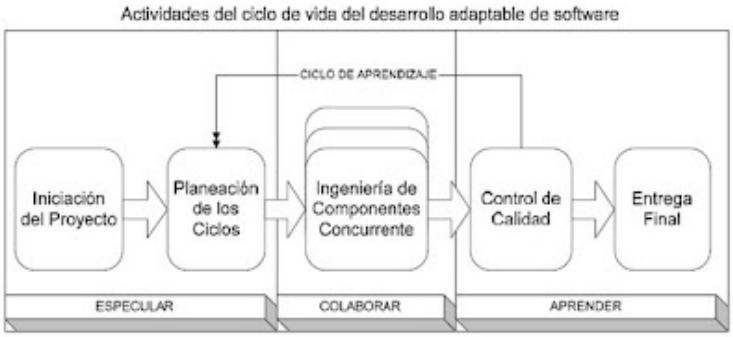

Figura 4.2: Ciclo ASD

4.2.2.2 Colaboración:

Las aplicaciones complejas requieren la recopilación y el análisis de grandes volúmenes de información y ejecución de tareas. Este proceso es inmanejable para un individuo. En entornos dinámicos, donde fluyen grandes cantidades de datos, es esencial la colaboración. Un solo individuo o un pequeño grupo no puede abarcar todo el conocimiento necesario.

4.2.2.3 Aprendizaje:

La evaluación continua del conocimiento a través de retroalimentaciones y reuniones grupales al final de cada ciclo iterativo es esencial. Este enfoque difiere de la evaluación al final del proyecto. Evaluar constantemente permite enfrentar y resolver de manera efectiva los cambios constantes del proyecto y su adaptación.

Para el desarrollo del SCSU se hizo un proceso iterativo donde en cada ciclo se abordó cada una de las fases descritas y así se fueron añadiendo funcionalidades al Sistema, y también en otros casos desechándolas, ya que no se adaptaban de forma correcta a los objetivos propuestos. Como esta metodología se enfoca en construir el software en pequeñas y frecuentes partes incrementales, añadiendo funcionalidades con cada iteración, es válido asumir que este modelo es representativo para la aproximación de desarrollo realizada.

La literatura en este tema siempre especifica a un cliente del que hay que obtener retroalimentación temprana, para así adaptar el producto a medida que evolucionan. Esto fue lo que se hizo en reuniones continuas en la materia Tópicos Especiales en Sistemas de Información y Gerencia que representó a la unidad requirente (cliente) y así se fueron evaluando los requisitos y se formularon las correspondientes hipótesis ,se observó y se midió el desempeño, por ejemplo, en los modelos de aprendizaje automático preentrenados usados para los procesamiento de los textos.

Capítulo 5

Desarrollo de la Solución:

Este es el capítulo del desarrollo

5.1 Descripción general de la Solución:

5.2 Arquitectura de la Solución:

5.3 Análisis y diseño:

5.4 Ciclos de Desarrollo:

5.4.1 Obtención del Conjunto de Datos:

Recuperación

5.4.2 Extracción y Clasificación de Información del Conjunto de Datos:

Extracción y clasificación

5.4.3 Prototipo de buscador:

5.4.4 Integración de componentes del software:

5.5 Pruebas:

5.5.1 Funcionales:

5.5.2 Rendimiento:

5.5.3 Relevancia:

Capítulo 6

Conclusiones:

Este es el capítulo de las Conclusiones

6.1 Contribución:

6.2 Trabajos Futuros:

Bibliografía

(2016). *World Development Report 2016: Digital Dividends*. World Bank, Washington, DC. OCLC: ocn915488955.

Aggarwal, C. C. (2018a). *Machine Learning for Text*. Springer International Publishing : Imprint: Springer, Cham, 1st ed. 2018 edition. DOI: 10.1007/978-3-319-73531-3.

Aggarwal, C. C. (2018b). *Machine Learning for Text*. Springer International Publishing : Imprint: Springer, Cham, 1st ed. 2018 edition. DOI: 10.1007/978-3-319-73531-3.

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.

Brin, S. and Page, L. (1998). The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1-7):107–117.

Chen, D. and Manning, C. D. (2014). A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 740–750.

Cook, J. (2017). *Docker for Data Science*. Apress.

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.

de Marneffe, M.-C., Manning, C. D., Nivre, J., and Zeman, D. (2021). Universal dependencies. *Computational Linguistics*, pages 1–54.

Desagulier, G. (2017). *Corpus Linguistics and Statistics with R*. Springer International Publishing.

Dueñas, M., Rojas, D., and Morales, M. (2011). Propuesta metodológica para realizar mapas de conocimiento. *Revista Facultad de Ciencias Económicas*, 20(1):77–90.

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.

Heydari, M. and Teimourpour, B. (2020). Analysis of researchgate, a community detection approach. In *2020 6th International Conference on Web Research (ICWR)*, pages 319–324. IEEE.

Highsmith, J. A. (2000). *Adaptive software development: a collaborative approach to managing complex systems*. Dorset House Pub, New York.

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., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S., and McClosky, D. (2014). The Stanford CoreNLP natural language processing toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60, Baltimore, Maryland. Association for Computational Linguistics.

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

Nüst, D., Sochat, V., Marwick, B., Eglen, S. J., Head, T., Hirst, T., and Evans, B. D. (2020). Ten simple rules for writing dockerfiles for reproducible data science. *PLOS Computational Biology*, 16(11):e1008316.

Reimers, N. and Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks.

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

Smith, T. and Waterman, M. (1981). Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197.

Stephens, R. (2015). *Beginning software engineering*. Wrox, A Wiley Brand, Indianapolis, IN.

Straka, M. and Straková, J. (2017). Tokenizing, pos tagging, lemmatizing and parsing ud 2.0 with udpipe. *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*.

Willett, P. (2006). The porter stemming algorithm: then and now. *Program*, 40(3):219–223.

Zhai, C. and Massung, S. (2016). *Text data management and analysis: a practical introduction to information retrieval and text mining*. Number #12 in ACM Books. ACM Books, New York, first edition edition. OCLC: ocn957355971.