UNIVERSIDAD AUTÓNOMA DEL ESTADO DE HIDALGO

Instituto de Ciencias Básicas e Ingeniería.
Área Académica de Computación y Electrónica.
Licenciatura en Ciencias Computacionales.

”DESARROLLO DE UN SISTEMA DE RECOMENDACIÓN MUSICAL BASADO EN MÉTODOS DE BÚSQUEDA APROXIMADA DE VECINOS CERCANOS”.

TESIS

QUE PARA OBTENER EL GRADO DE

LICENCIADO EN CIENCIAS COMPUTACIONALES

PRESENTA:

ÁNGEL DAVID FRANCO HERNÁNDEZ

ASESORES:

DRA. ANILU FRANCO ARCEGA
DRA. GUADALUPE CARMONA ARROYO

Mineral de la Reforma, Hidalgo 2025

CONTENIDO

Índice de figuras

ÍNDICE DE ECUACIONES

RESUMEN

En los últimos años se ha presentado un avance considerable en las tecnologías y algoritmos de recomendación de contenido digital, principalmente, en uno de los elementos culturales más populares como lo es la música. Estos algoritmos representan una razón de peso para que un usuario decida permanecer en una plataforma u otra, es por eso que inclusive una mejora minúscula puede significar en un incremento significativo en la satisfacción del usuario.

En el presente proyecto se busca potenciar el algoritmo *Voyager* desarrollado por la empresa *Spotify* con el propósito de mejorar los sistemas de recomendación de la plataforma mediante métodos de Búsqueda de Vecinos más Cercanos. Se investigará de manera profunda los fundamentos del algoritmo, sus unidades de medida y la complejidad computacional que genera para ser usado posteriormente en el sistema.

Utilizando esta herramienta como base se desarrollará un sistema web que permita a los usuarios crear una nueva experiencia de escucha denominada como *Viajes Musicales* a través de las herramientas de desarrollo que Spotify hace públicas, usando como fuente de información la *Spotify Web API*, aprovechando sus alcances y evaluando sus limitaciones para obtener la mayor cantidad de información por segundo a través de un algoritmo de búsqueda exhaustiva de canciones.

Este algoritmo de búsqueda exhaustiva será planteado por el autor como una capa previa de filtración de música a través de características enteramente *cualitativas*, dando así una muestra signficativa de la cual se pueden obtener recomendaciones más fieles a lo que el usuario le gustaría escuchar.

Las aportaciones más importantes de este proyecto se centran en la explicación profunda de los algoritmos de Búsqueda por Vecinos Aproximadamente más Cercanos y por qué son tan eficientes en tareas de recomendación, además de poner en practica el conocimiento en un sistema vinculado a una funcionalidad innovadora, creativa y que mejora la experiencia de escucha musical de los usuarios.

ABSTRACT

Nowadays, there has been a considerable improvement in digital recommendation technologies and algorithms, particularly in one of the most popular cultural elements: music. These algorithms represent a compelling reason for a user to choose one platform over another; therefore, even a tiny improvement can result in a significant increase in user satisfaction.

This project seeks to enhance the *Voyager* algorithm created by *Spotify*, which is dedicated to improving the platform's recommendation technology using Nearest Neighbor Search methods. This is done by first thoroughly investigating the algorithm's fundamentals, its units of measurement, and the computational complexity it generates for use in the system.

Using this tool as a base, it's been sought to develop a web system that allows users to create a new listening experience named as *Musical Journeys* through the developer tools that Spotify opens to developers using *Spotify Web API* as a source of information, taking advantage of its scope and evaluating its limitations to obtain the greatest amount of information per second through an exhaustive song search algorithm.

This exhaustive search algorithm will be proposed by the author as a previous layer of music filtering thorugh entirely qualitative characteristics, thus giving a significant sample from which recommendations more faithful to what the user would like to listen to can be obtained.

The most important contributions of this project focus on the in-depth explanation of Approximate Nearest Neighbor Search algorithms and why they are so efficient in recommendation tasks, as well as putting this knowledge into practice in a system linked to innovative and creative functionality that improves users music listening experience

INTRODUCCIÓN

Actualmente la música es uno de los productos de entretenimiento con más consumidores en todo el mundo, y su crecimiento ha sido exponencial desde que el formato digital se consolidó en la industria. Plataformas de streaming como Spotify ampliaron el acceso a la música, logrando conectar a sectores que no podían permitirse el formato físico con sus canciones favoritas a un precio considerablemente más bajo. Este cambio transformó la relación de las personas con el arte musical, pasando de ser un lujo a convertirse en un consumo cotidiano.

Spotify ha sido una empresa clave en la digitalización de la música desde la década de 2010, sin embargo, con la consolidación del formato digital se volvió determinante para las empresas del mercado seguir innovando para mantenerse por encima de los competidores. Por ello, Spotify se ha decantado por invertir en el desarrollo de herramientas que fomenten la preferencia de los usuarios por su plataforma como su favorita para el consumo musical diario.

Durante años los desarrolladores de diferentes empresas han invertido tiempo y recursos en la investigación de métodos eficientes de recomendación para sistemas con información masiva, sin embargo, se han encontrado con dificultades crecientes relacionadas al consumo de recursos computacionales y al tiempo estimado de respuesta que, al hablar de empresas multinacionales con millones de usuarios, debe ser signficativamente rápido.

Como resultado de estos esfuerzos, el equipo de desarrolladores de Spotify ha diseñado diversas herramientas que mejoran la recomendación de música basadas en algoritmos que emplean técnicas de minería de datos y machine learning, logrando minimizar los obstaculos computacionales inherentes al problema a resolver.

No obstante, a pesar de que Spotify ha desarrollado algoritmos eficientes de búsqueda de recomendaciones, persisten áreas de oportunidad a ser tomadas en cuenta. En este proyecto se identifica una de ellas aún no abordada por la plataforma principal ni por herramientas externas y se propone un sistema desarrollado en el entorno web que soluciona esta problemática a través de una propuesta innovadora y creativa, complementando las investigaciones actuales que Spotify ha logrado en los últimos años.

PLANTEAMIENTO DEL PROBLEMA

En la actualidad, el uso de la plataforma de Spotify le permite al usuario establecer sesiones de escucha de las siguientes maneras:

1. **Elegir manualmente las canciones a escuchar:** Esto implica elegir cada una de las canciones que serán reproducidas en el espacio de tiempo que el usuario decida siguiendo un orden establecido. Este caso de uso se ha exponenciado con el uso de *Listas de Reproducción* [1].
2. **Elegir una canción base como la fuente de recomendaciones:** Sucede cuando el usuario elige una canción que no pertenece a una Lista de Reproducción previamente almacenada, provocando que Spotify ejecute el algoritmo de recomendación tomando de referencia la canción elegida por el usuario. Además, la *Canción base* también es usada cuando la Lista de Reproducción termina de reproducirse totalmente, generando el mismo comportamiento.

Esto indica una problemática emergente: El usuario tiene poco o nulo control de las recomendaciones que la plataforma brinda, teniendo que elegir una sesión de escucha de forma manual o cambiando la *canción base* constantemente. Como resultado, en largas sesiones de escucha, el usuario no sabrá qué canciones elegir o, en el caso de usar el algoritmo de recomendación, es probable que se aburra de escuchar temas similares por largos periodos de tiempo.

Esta situación refleja un área de oportunidad en la que el usuario pueda mantener un control más fino sobre el funcionamiento del algoritmo de recomendación, permitiendo así el diseño de sesiones de escucha prolongadas con mayor diversidad. De esta manera, se plantea la necesidad de explorar métodos alternativos de recomendación que complementen los enfoques actuales empleados por Spotify.

[1] **Lista de Reproducción:** Colección ordenada de canciones que se pueden reproducir de forma consecutiva. Se almacenan para ser usadas en el futuro.

PROPUESTA DE SOLUCIÓN

En el presente proyecto se plantea una solución creativa e innovadora que le brinda al usuario una capa extra de personalización y control del algoritmo de recomendación a través del concepto de *Viajes Musicales*.

En esta tesis se introducen un conjunto de conceptos que servirán como base para el desarrollo del sistema. En particular, se propone el concepto de *Viaje Musical*, que es definido como una sesión de escucha formada por un conjunto de canciones, de las cuales el usuario solo se preocupará por las que se denominan *estaciones*. El concepto de estaciones es definido como aquellas canciones que el usuario elige y que representan la guía que tomarán los algoritmos de recomendación para forjar un camino coherente entre las diferentes estaciones, creando así un viaje musical guiado por el usuario pero enriquecido por el sistema.

Con el objetivo de alcanzar dicho nivel de interacción se propone desarrollar un sistema web en donde el usuario pueda vincular su cuenta de Spotify, elegir su viaje musical y vincular la lista de canciones resultante a su cuenta de forma automática, simplificando al máximo la experiencia del usuario para lograr una herramienta cotidiana y fácil de usar.

Por esta razón se busca desarrollar un algoritmo de recomendación que se inspire en los desarrollados por la plataforma aprovechando los recursos públicos que la empresa facilita como la *Spotify Web API* o el algoritmo de recomendación *Voyager*.

La propuesta no solo busca mejorar las recomendaciones existentes, sino ofrecer al usuario una experiencia activa, flexible y narrativa en su consumo musical. De esta manera, los viajes musicales representan un puente entre el control total del usuario y la automatización de los algoritmos, creando una experiencia de escucha dinámica y personalizada que actualmente no existe en la plataforma.

JUSTIFICACIÓN

En la actualidad, los algoritmos de recomendación representan un elemento clave en la competitividad de los sistemas de entretenimiento digital, ya que permiten ofrecer experiencias personalizadas y mantener la fidelidad de los usuarios. Comprender a fondo las aportaciones previas en este ámbito es esencial para aprovecharlas de manera óptima y generar mejoras que impacten de forma directa en la satisfacción de los usuarios.

Dentro de las técnicas más utilizadas, los algoritmos de Búsqueda de Vecinos Aproximadamente más Cercanos constituyen una herramienta con múltiples aplicaciones, incluyendo el ámbito musical. En particular, el algoritmo *Voyager* fue desarrollado con el objetivo de simplificar la implementación de este tipo de metodologías. Sin embargo, su uso resulta limitado cuando no se entiende a profundidad su funcionamiento, lo cual puede derivar en discrepancias entre los filtros aplicados y los resultados obtenidos. Por esta razón, este proyecto se centra en explicar de manera detallada tanto los fundamentos de la Búsqueda de Vecinos Cercanos como la lógica interna de *Voyager*, con el fin de optimizar su aprovechamiento en sistemas de recomendación.

Además, se propone una nueva capa de *Filtrado por características Cualitativas* que *Voyager* por naturaleza no es capaz de soportar, añadiendo así una innovación técnica enfocada en búsquedas heurísticas adicionales previo al paso de recomendación.

Ambas aportaciones desarrolladas en el contexto musical le permitirán a los usuarios vincularse de forma diferente con las sesiones de música, dándoles un control adicional sobre un algoritmo que, actualmente, está totalmente cerrado al funcionamiento técnico. Este proyecto propone conectar al usuario con su algoritmo y explotar esa interacción de forma innovadora.

Adicionalmente, la implementación de un sistema basado en esta metodología no solo contribuye a fortalecer la investigación en el área, sino que también ofrece a nuevos desarrolladores una base sólida y bien documentada para el diseño de arquitecturas emergentes. De esta manera, el presente trabajo no solo atiende una necesidad técnica, sino que también abre la posibilidad de inspirar nuevas propuestas en el campo de los sistemas de recomendación de cualquier ámbito.

OBJETIVO GENERAL

Desarrollar un sistema web de recomendación musical que le permita al usuario generar sesiones de escucha completas a partir de un conjunto representativo de canciones que servirá como base, las cuales permitirán que el sistema complete la lista mediante recomendaciones personalizadas, implementando algoritmos de búsqueda inteligentes.

OBJETIVOS ESPECÍFICOS

1. Investigar la metodología de Búsqueda de Vecinos Aproximadamente más Cercanos para identificar las deficiencias de los métodos actuales en el entorno de sistemas de recomendación revisando literatura científica y técnica relevante.
2. Implementar el algoritmo *Voyager* con el fin de resolver de forma eficiente el problema de recomendación abordado, configurando sus propiedades de manera justificada y bien documentada.
3. Crear un algoritmo de *Generación de Muestra Significativa* añadiendo una capa de filtrado *Cualitativo* que aporte al problema de recomendación musical.
4. Desarrollar un sistema web dividido en capas que logre implementar los algoritmos de recomendación planteados y, además, obtenga información de la *Spotify Web API* que será usada para las recomendaciones.
5. Utilizar la *Spotify Web API* para facilitar al usuario la posibilidad de vincular su cuenta de Spotify de forma segura y alineada a los estándares de la plataforma.
6. Desarrollar una interface de usuario llamativa que muestre el concepto de *Viaje Musical* y permita obtener una sesión de escucha coherente basada en *Estaciones* haciendo uso de tecnologías web.

Capítulo I

MARCO TEÓRICO

En el presente capítulo se abordarán los temas más importantes que servirán como base para el desarrollo del proyecto. En particular, se explicarán los fundamentos de los sistemas de recomendación, haciendo enfasis en los diferentes enfoques y técnicas más utilizadas en este campo. Además, se presentará una revisión detallada de los algoritmos de Búsqueda de Vecinos, cómo funcionan y las métricas que utilizan con el fin de entender a profundidad los algoritmos actuales como *Voyager*, que será explicado en detalle al final del capítulo.

1.1. FUNDAMENTOS DE LOS SISTEMAS DE RECOMENDACIÓN

Los sistemas de recomendación son **agentes de información personalizada** que brindan sugerencias de elementos que *podrían* ser útiles para un usuario. Los resultados dados por un sistema de recomendación se denominan **Recomendaciones** que representan una opción valiosa que podría ser del interés de una persona o usuario.

1.1.1. CONCEPTOS BÁSICOS DE LOS SISTEMAS DE RECOMENDACIÓN

Los Sistemas de Recomendación se fundamentan mediante entidades que aportan información desde diferentes medios:

- **Usuario:** La entidad a la que se le hacen llegar las recomendaciones es denominada como *usuario*.

- **Items:** Es una pieza de información que refiere a un objeto tangible o digital que el sistema de recomendación sugerirá al usuario en una interacción a través de la Web, email o mensajes de texto (Kotkov et al., 2020).
- **Fuente de Conocimiento:** Determina qué técnica será usada para definir el medio de información y determinar el conocimiento necesario para brindar sugerencias relevantes.

El principio básico que fundamenta a los algoritmos de recomendación es la dependencia significativa que existe entre un *usuario* y los *items* con los que interactúa.

1.1.2. OBJETIVOS DE UN SISTEMA DE RECOMENDACIÓN:

Según (Aggarwal, 2016) más allá de los objetivos mercadológicos que se le atribuyen a los sistemas de recomendación enfocados en mejorar el rendimiento económico de una plataforma, se pueden definir objetivos técnicos y operacionales que mejoran el rendimiento de un sistema en éste ámbito.

- **Relevancia:** Un sistema de recomendación siempre debe brindar sugerencias relevantes para el usuario.
- **Novedad:** Las recomendaciones suelen ser más útiles cuando sugieren elementos que el usuario no ha visto antes. Además, recomendar los objetos más populares puede conducir a reducir la diversidad del consumo.
- **Sorpresa:** No basta con que los objetos de recomendación sean nuevos para el usuario, sino que también deben sorprenderlo, dándole la sensación de un *descubrimiento inesperado.*
- **Diversidad de Recomendaciones:** La lista de sugerencias obtenidas deben ser suficientemente diferentes entre sí para no aburrir al usuario, sin embargo, la diversidad de *items* no debe alejarse demasiado de sus preferencias. «*La diversidad tiene el beneficio de asegurar que el usuario no se aburre de constantes recomendaciones de items similares.* » [Aggarwal, 2016]

Existen dos versiones diferentes que definen los resultados esperados de un Sistema de Recomendación, ambos buscan resolver el problema de encontrar items relevantes para los usuarios mediante modelos distintos:

1. **Versión de Predicción:** Se basa en predecir la calificación que el usuario daría a un item. Se asume que existe un conjunto de Datos de Entrenamiento que indica las preferencias del usuario.
2. **Versión de Top - k:** Este enfoque argumenta que muchos de los sistemas no necesitan conocer la calificación estimada del usuario, por el contrario, solo necesitan saber qué recomendar sin necesidad de estimar una calificación exacta. Esto se suele hacer mediante un Top - k elementos que probablemente le gusten al usuario.

1.1.3. PRINCIPALES ENFOQUES DE RECOMENDACIÓN

Para que un sistema de recomendación logre cumplir los objetivos básicos esperados se necesita establecer de forma clara cuál será el filtro para sugerir items al usuario eligiendo diferentes enfoques que se adapten a los resultados que se buscan obtener.

Basándose en (Isinkaye et al., 2015) existen diferentes filtros de recomendación principales, sin embargo, se muestran los más relevantes para el presente proyecto:

ENFOQUE BASADO EN CONTENIDO

Es una técnica de sistema de recomendación que se centra en el análisis de atributos de cada item para generar predicciones sobre las preferencias del usuario. Se apoya de valoraciones previamente vinculadas a un perfil que permiten identificar características clave de los objetos que el usuario ha calificado, principalmente de aquellos que cuentan con una valoración positiva. Por esta razón, la eficacia de este enfoque se basa en la cantidad de información disponible de los items.

Los algoritmos que implementan esta técnica suelen basarse en modelos de análisis estadístico y de *Machine Learning.*

VENTAJAS

- **Capacidad recomendar items nuevos:** Este enfoque permite sugerir elementos que aún no han sido evaluados por los usuarios gracias a la información contenida en sus *metadatos* [1].
- **Alta adaptabilidad:** Se adapta a los cambios de preferencia del usuario en un lapso corto de tiempo.
- **Independencia entre usuarios:** Los usuarios reciben recomendaciones sin la necesidad de compartir su información con externos, priorizando la privacidad y seguridad de los datos.

DESVENTAJAS

- **Dependencia en la profundidad de la información:** Los motores de recomendación basados en esta técnica son dependientes de los metadatos vinculados a los items. Por lo tanto, requieren que los elementos sean profundamente descritos por sus atributos. Esto es definido como ***Análisis limitado al contenido***.
- **Falta de Diversidad:** Es probable que las sugerencias sufran de *sobre-especialización* o, en otras palabras, que carezcan de diversidad debido a que los usuarios están restringidos a obtener recomendaciones similares a los items asociados a sus perfiles.

ENFOQUE COLABORATIVO

A diferencia del enfoque basado en contenido, que se centra en los atributos de los items, el enfoque colaborativo se centra en las similitudes entre lo usuarios para generar recomendaciones.

Este enfoque funciona construyendo una base de datos descrita por una matríz *usuarios - items* que representa la preferencia de un conjunto de usuarios por un item particular. Esto significa que vincula a diferentes perfiles con preferencias e intereses similares para guiar las recomendaciones, formando así un grupo de perfiles que comparten intereses denominado *Vecindario*.

[1] **Metadatos:** Atributos que describen, proveen contexto, indican la calidad, o documentan las características de un objeto. (Greenberg, 2005)

Por lo tanto, un usuario recibe recomendaciones de contenido que no ha evaluado pero que otros usuarios dentro de su *Vecindario* han calificado de forma positiva. Los resultados que se pueden obtener a través de este enfoque son los siguientes:

- **Predicciones:** Es un valor numérico que expresa un puntaje del elemento j para el usuario i, esta predicción se representa con la notación $R_{i,j}$.
- **Rankings:** Es una lista de los Top - N elementos que al usuario le podrían gustar más.

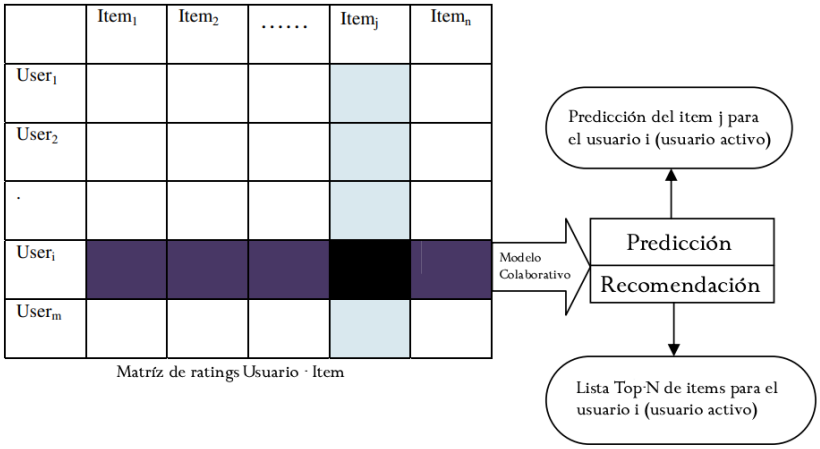

Figura 1: Ejemplo del Filtro Basado en Colaboración (Isinkaye et al., 2015).

En la Figura 1 se muestra un ejemplo de funcionamiento del enfoque colaborativo. Se puede observar que la matríz está compuesta por filas que representan a los usuarios de un mismo *Vecindario*, y por columnas que representan a los *items*. Las intersecciones mostradas de color morado representan la calificación que el usuario ha dado a elementos previos, y la intersección diferenciada de color negro expresa una predicción basada en las calificaciones que los vecinos han asignado a ese item. El diagrama remarca la distinción entre *Predicción* y la *Recomendación* dado que, en el caso de la predicción, la intersección representada en color negro representa el valor $R_{i,j}$, y en el caso de la recomendación representa un *item* dentro de los Top - N items del usuario actual.

VENTAJAS

Este enfoque tiene ventajas fuertemente diferenciadas respecto al enfoque basado en contenido, algunos de sus puntos fuertes son los siguientes:

- **Independencia al Contenido:** Los algoritmos colaborativos pueden funcionar de forma efectiva en dominios en los cuales no existe diversidad de información (o metadatos) para describir los items.
- **Recomendaciones basadas en Serendipia** [2] : Significa que tiene la capacidad de recomendar elementos que son aparentemente ajenos a las preferencias del usuario, pero que para sus *vecinos* son relevantes, conectando al usuario con contenido nuevo.

DESVENTAJAS

- **Problema de *Inicio Frío***: Se refiere a la situación en donde el Sistema de Recomendación no tiene información adecuada acerca del usuario o acerca del item para hacer predicciones relevantes (Burke, 2007). Este problema es crucial ya que este enfoque se vuelve dependiente de conocer con el tiempo las preferencias del usuario que, inicialmente, son desconocidas.
- **Problema del *Esparcimiento de Datos***: Otro de los puntos de los que depende el enfoque colaborativo es en las calificaciones hechas por los usuarios, causando la necesidad de que existan previas valoraciones para los items, de lo contrario, se generan recomendaciones poco relevantes.
- **Escalabilidad**: Un problema bastante común en los algoritmos de recomendación con enfoque colaborativo es la escalabilidad. Cuando la cantidad de usuarios e items crecen rápidamente, la complejidad computacional crece de forma lineal en relacion a estos dos aspectos, dando un costo computacional representado por la siguiente fórmula: $O(items \cdot usuarios)$[3]. Es importante mencionar que un algoritmo lineal no logra ser eficaz cuando la cantidad de información es masiva.

[2] **Serendipia en Sistemas de Recomendación:** *Items* que son relevantes pero inesperados para el usuario. *Items* que son nuevos e inesperados. (Kotkov et al., 2020)

[3] **Notación *BIG O*:** Es un estándar que describe el orden de crecimiento de un algoritmo en función del tamaño del problema. En el presente documento se usará para expresar la complejidad computacional de los algoritmos descritos.

- ***Sinónimos:*** Se refiere a la situación en la que items muy similares aparecen registrados con diferentes nombres o identificadores (Isinkaye et al., 2015). Los sistemas de colaboración no reconocen automáticamente que se trata de equivalencias; por ejemplo, *item 1: Álbum Let It Be - The Beatles*, *item 2: Álbum Let It Be (Remasterizado) - The Beatles*. En este caso, el sistema podría tratarlos como elementos completamente diferentes, a pesar de estar estrechamente relacionados.

ENFOQUE HÍBRIDO

A pesar de que los enfoques de recomendación basados en contenido y colaborativos son eficaces en escenarios particulares, sus desventajas inherentes limitan su escalabilidad en plataformas de uso masivo. El enfoque híbrido busca solventar las desventajas de cada enfoque combinando diferentes técnicas de recomendación con el objetivo de evitar las limitaciones y problemas que los enfoques puros pueden enfrentar.

Los sistemas de recomendación híbridos están fuertemente relacionados al campo del *Análisis de Conjuntos (Ensemble Analysis)* en los cuales se combina el poder de multiples tipos de algoritmos basados en *Machine Learning* para crear modelos más robustos (Aggarwal, 2016).

TÍPOS DE ENFOQUES HÍBRIDOS

- **Enfoque Híbrido basado en Pesos:** Este enfoque combina los resultados de diferentes enfoques de recomendación para crear una lista de recomendaciones que integran las calificaciones de cada técnica mediante una fórmula lineal. Inicialmente todas las técnicas usadas inician con un peso igualado que se modifica a través de la evaluación del usuario.
- **Enfoque Híbrido por Conmutación:** Este enfoque se basa en una heurística definida que selecciona la mejor técnica de recomendación adaptada al momento. Por ejemplo, si se necesita de un algoritmo de recomendación para un usuario nuevo, se puede identificar que usar el enfoque colaborativo no es una buena opción debido al *Inicio Frío*, por lo tanto, se elige el enfoque basado en contenido. El beneficio de esta estrategia es que el sistema es sensible a las fortalezas y debilidades de sus enfoques de recomendación (Isinkaye et al., 2015).

  Sin embargo, esta conmutación de enfoques tiene una desventaja inherente debido al costo computacional que requiere el cambiar de métodos constantemente, principalmente por la diferencia de criterios y parámetros de información que cada enfoque requiere.

- **Enfoque Híbrido Combinado:** Este enfoque combina los resultados de diferentes algoritmos de recomendación, ejecutándolos en paralelo para generar una lista única de recomendaciones. El beneficio principal de esta estrategia es que el rendimiento local de un enfoque específico no suele afectar de manera significativa al rendimiento global del sistema, resultando en una mayor robustez.

1.2. PARADIGMAS Y MÉTRICAS DE EVALUACIÓN PARA UN SISTEMA DE RECOMENDACIÓN:

Los Sistemas de Recomendación deben desarrollarse mediante un enfoque alineado con los objetivos esperados respecto a la interacción con los usuarios. Estos objetivos principales han sido estudiados y seccionados en paradigmas de evaluación, sin embargo, basándose en la investigación de (Aggarwal, 2016) se pueden definir también métricas de evaluación generales para calificar el desempeño de un sistema de recomendación. En esta sección se mencionarán generalidades de los paradigmas de evaluación y se hará un enfasis en las métricas generales.

1.2.1. PARADIGMAS DE EVALUACIÓN:

Existen tres paradigmas de evaluación principales dirigidos a los sistemas de recomendación, donde las principales diferencias se centran en cómo los usuarios interactúan con las métricas de evaluación.

1. **Usuarios de Estudio:** Este paradigma se centra en invitar a los usuarios a un proceso de evaluación para interactuar con el sistema y evaluar su desempeño en contextos específicos. Por lo tanto, las mediciones se obtienen de los comentarios de los usuarios y de las observaciones de interacción obtenidas por el sistema. Esta información se usa para entender cuáles fueron las preferencias del usuario, qué le gustó y qué no le gustó.

   La principal ventaja de este paradigma es que los usuarios dan permisos explícitos para usar su información en función de la evaluación del sistema, sin embargo, las desventajas más significativas radican en los cambios inconscientes del usuario al estar en un escenario de evaluación y en la poca escalabilidad del método. Es probable que no todos los usuarios estén de

acuerdo en participar en un escenario de evaluación, y aquellos que lo hagan podrían no tener preferencias significativas alineadas con la mayoría de la población o, en resumen, se podrían obtener resultados específicos pero no representativos.

Las desventajas mencionadas ocasionan que este paradigma sea usado sólo en entornos académicos o de proyectos que no sean de gran escala.

2. **Evaluación *Online***: Este paradigma de igual manera depende de la interacción del usuario pero en un entorno de uso real, lo que fomenta resultados más naturales debido a que ahora se interactúa con el sistema en una situación cotidiana. Una de las fortalezas más importantes de este paradigma es que funciona combinando diferentes *enfoques de los sistemas de recomendación* para evaluar cuál se comporta mejor en un escenario en particular. Para comparar el rendimiento de diferentes enfoques se usan métricas concretas como la tasa de conversión, CTR, tiempo de interacción, etc... Estos métodos son también conocidos como *Pruebas* A/B y miden el impacto directo del sistema de recomendación hacia el usuario.

   "La idea básica es similar a un apostador (el sistema de recomendación) que debe elegir una máquina tragamonedas (diferentes enfoques de recomendación) en un casino. El apostador sospecha que una de las tragamonedas tiene un porcentaje de retorno más alto (tasa de conversión, CTR, etc...) que las demás. Por lo tanto, el apostador juega con cada tragamonedas de forma individual para observar el porcentaje de retorno de cada máquina. De forma codiciosa el apostador selecciona solo aquella tragamonedas que tenga un rendimiento mas efectivo respecto a las demás" (Aggarwal, 2016).

   Una de las ventajas más grandes de este paradigma es que permite medir de forma algorítmica el rendimiendo de diferentes *enfoques de recomendación* para obtener el mejor rendimiento posible a través de la interacción directa del usuario. Sin embargo, la principal desventaja es que estos sistemas no pueden ser implementados de forma realista a menos de que se tenga una cantidad masiva de usuarios, haciendo que sea muy complejo de usar en las fases iniciales del sistema.

   La necesidad de contar con una cantidad masiva de usuarios surge de los métodos de *Pruebas* A/B que agrupa a los usuarios en muestras aleatorias a las que se les aplican diferentes algoritmos de recomendación y se mide la satisfacción del usuario mediante diferentes métricas. Para que las mediciones retornen resultados relevantes cada grupo debe contar con un gran número de usuarios.

3. **Evaluación *Offline* con *Datasets*[4] Históricos:** En la evaluación Offline se trabaja con datos históricos como, por ejemplo, las calificaciones de un usuario a items del sistema. La principal ventaja de trabajar con *Datasets* de información es que se tiene un conjunto grande de información previamente recopilada y no se tiene la necesidad realizar un seguimiento a miles de usuarios activos para evaluar el sistema. Una vez que un *Dataset* ha sido recolectado se puede usar como principal punto de referencia para comparar diferentes algoritmos.

   Además, se pueden utilizar diversos datasets que favorezcan la *generalidad* del Sistema de Recomendación.

   Sin embargo, la principal desventaja de la *Evaluación Offline* es que no logra predecir cómo reaccionará el usuario al sistema en el futuro. A pesar de estas áreas de oportunidad, éste paradigma sigue siendo uno de los más usados y aceptados para la evaluación de sistemas de recomendación, esto se debe a que las estadísticas que se pueden obtener de estos métodos son robustas, confiables y fáciles de entender.

1.2.2. MÉTRICAS GENERALES DE EVALUACIÓN:

Además de los paradigmas de evaluación que definen metodologías específicas para realizar pruebas a los usuarios, existen métricas cuantificables que añaden una capa extra de complejidad y robustez a los sistemas de recomendación. Estas metricas cumplen la función de asegurar resultados satisfactorios para el usuario mediante características cuantificables y subjetivas, entendiendo que el gusto y las preferencias pueden ser complejas de representar mediante mediciones numéricas.

EXACTITUD

La exactitud es una de las métricas fundamentales con las que se evalúan los sistemas de recomendación. Se basan en *ratings* que se definen como *çantidades numéricas que deben ser estimadas"* (Aggarwal, 2016). Sin embargo, como ha sido mencionado anteriormente, algunos enfoques de recomendación no se centran en predecir calificaciones sino que reunen el top-k de los *items* más favorables.

[4]**Dataset:** Es una colección de elementos relacionados organizados y con un formato para cumplir un propósito particular (Chapman et al., 2019).

La medición por exactitud tiene diversos componentes que aseguran su efectividad:

- ***Métricas de Efectividad:*** Las Métricas de Efectividad son usadas para evaluar tanto la efectividad de predicción para estimar la calificación de un usuario específico respecto a un item o la efectividad del top - k sugerido por el sistema de recomendación (Aggarwal, 2016).

  Existen dos métodos diferentes para medir la efectividad de las predicciones y de los top - k:

  - ***Efectividad de Predicción:*** Se centra en medir el *Error Específico de Entrada* [5] que se define mediante la siguiente fórmula:

$$e_{uj} = \hat{r}_{uj} - r_{uj} \tag{I.1}$$

    Donde:

    u representa al usuario.

    j representa al item.

    $\hat{r}_{uj}$ representa la predicción de calificación.

    r_{uj} representa la calificación real del usuario.

    Este cálculo de e_{uj} puede ser aprovechado de diferentes maneras para hacer un calculo sobre el conjunto de errores de predicción E y los rankings vinculados a cada $e \in E$ [6].

    Para sintetizar, se dice que el conjunto E es definido por:

$$E = \{\, e_{uj} \mid (u,j) \in D \,\} \tag{I.2}$$

    Recordando que e_{uj} representa el *Error específico de Entrada* del usuario u para el item j. La fórmula indica que a cada par $(u,\ j)$ perteneciente a todos los posibles pares en el conjunto D (o, en otras palabras, el conjunto de items evaluados por el usuario) tiene asignado un valor e_{uj}.

    Este valor de *Error específico de entrada* puede indicar la calidad de un sistema de recomendación si se usa en conjunto con métricas de efectividad.

[5] **Error Específico de Entrada:** Diferencia entre la calificación que el sistema de recomendación predice y la calificación real que el usuario dio.

[6] **Pertenencia** $\in$: Éste símbolo indica que un elemento cualquiera u *pertenece* a un conjunto R, por lo tanto $u \in R$

Mean Squared Error (MSE):

El Error Cuadrático Medio es una métrica utilizada para evaluar la diferencia entre las calificaciones predichas por un modelo y las calificaciones reales del usuario. Funciona calculando el promedio de los errores elevados al cuadrado. Un valor MSE más bajo indica que el modelo tiene un mejor rendimiento, ya que sus predicciones están más cerca de los valores reales.

La forma del MSE es:

$$MSE = \frac{\sum_{(u,j) \in E} e_{uj}^2}{\mid E \mid} \tag{I.3}$$

Que se puede resumir en la sumatoria de todos los $e_{uj}^2 \in E$ divididos entre la cantidad de elementos en E para obtener un promedio, haciendo así que un valor alto de e_{uj}, al ser elevado al cuadrado, penalice de forma drástica los errores del sistema.

Root Mean Squared Error (RMSE): Como su nombre lo índica, esta métrica se basa en el MSE con la ligera diferencia de aplicar raíz cuadrada de la siguiente manera:

$$RMSE = \sqrt{\frac{\sum_{(u,j) \in E} e_{uj}^2}{|E|}} \tag{I.4}$$

En ésta definición se puede observar que es similar a MSE, la única diferencia radica en la escala de los resultados. RMSE mantiene las mismas unidades que la variable original y suele representar el formato estándar de error (Hodson, 2022).

◇ **Efectividad de Ranking:** Para los sistemas enfocados en *rankings* se pueden usar mediciones diferentes, algunas de ellas son las siguientes.

Evaluando Rankings vía Correlación:

Cuando un sistema de recomendación retorna un top - k de recomendaciones, en general, es deseable que los items más altos en el top sean más deseados por el usuario que los items más bajos. Por lo tanto, la idea central de la *Evaluación de Rankings vía Correlación* se centra en identificar qué tan correcto fue el desempeño del ranking retornado por el sistema de recomendación en relación con el ranking decidido por el usuario de forma manual.

Si se quisiera definir formalmente, se podría decir que ésta evaluación se centra en evaluar la similitud entre:

$$\begin{aligned} R_j &= \{i_{j1}, i_{j2}, i_{j3}, ..., i_{jn}\} \\ R_u &= \{i_{u1}, i_{u2}, i_{u3}, ..., i_{un}\} \end{aligned} \tag{I.5}$$

Donde:

j representa al sistema de recomendación.

u representa al usuario.

R_j representa el *Ranking* recomendado por el sistema.

R_u representa el *Ranking* elegido por el usuario.

i representa un item dentro de un *Ranking*.

Se debe tomar en cuenta que el análisis de correlación entre rankings no solo toma en cuenta que $i \in R$ tomando en cuenta ambos R_u y R_j, sino que también toma en cuenta el *orden* en que se encuentran los items.

Un detalle importante a tener encuenta es que las *calificaciones* suelen elegirse en una escala discreta, y podrían existir muchos empates en la verdad del mundo real (*ground truth*) (Aggarwal, 2016). Por lo tanto, tomando en cuenta que un usuario elige una calificación dentro de un conjunto discreto [7], por ejemplo, calificaciónes enteras del 1 al 5, o calificaciones basadas en estrellas, es probable que las valoraciones del usuario caigan en empates para diferentes items. Esta característica complica el cálculo de correlación, obligando al sistema a tener cuidado con la naturaleza poco fina de las calificaciones del usuario.

Para obtener el valor de similitud se usan las métricas *Spearman Rank Correlation*, *Kendall Rank Correlation* para obtener una medición exacta de la similitud entre R_u y R_j.

[7] **Conjuntos Discretos:** Un conjunto discreto es un conjunto formado por números en el cual entre cada número y el siguiente no hay ningún otro. (Perez García, 2021)

COBERTURA

Como se ha mencionado anteriormente, los sistemas de recomendación podrían tener dificultades en encontrar resultados relevantes cuando no se tiene la suficiente información de los *items* y de los *usuarios*, por ejemplo, cuando un usuario nuevo interactúa con la plataforma, aunque el sistema de recomendación tenga un excelente rendimiento en exactitud, es probable que no brinde resultados relevantes debido a la falta de información.

."Esta limitación de los sistemas de recomendación es consecuencia del hecho de que las matrices de calificación suelen ser escasas." (Aggarwal, 2016)

Para medir qué tan propenso es el sistema a caer en esta dificultad se usa el concepto de *Cobertura*, que mide la proporción de items que el sistema puede recomendar a un usuario. Sin embargo, generalmente durante la evaluación de sistemas de recomendación, es común que favorecer la *exactitud* tenga como consecuencia reducir la *cobertura* debido a que la *exactitud* favorece los items más parecidos a las preferencias del usuario, y la *cobertura* se centra en brindar diversidad de items. Es por esto que, en general, la mejor decisión sea obtener un balance entre ambas métricas sin buscar maximizar alguna de las dos.

CONFIABILIDAD Y CERTEZA

La estimación de calificaciones es un proceso inexacto que depende directamente de ciertos algoritmos que pueden proveer diferentes predicciones. Por lo tanto, es común que el usuario pierda confianza en el sistema al no saber qué tan acertadas serán las recomendaciones.

Para medir esta dificultad se usan métricas de confiabilidad que tratan de medir qué tan confiable es la recomendación que el sistema ha brindado.

."En general, los sistemas de recomendación que pueden recomendar de forma acertada elementos con un menor intervalo de confianza son más deseables debido a que refuerzan la confiabilidad del usuario por el sistema". (Aggarwal, 2016)

Por ejemplo, si el sistema obtiene un intervalo de confiabilidad de 4.8 a 5.0 significa que tiene un rendimiento certero debido a que la distancia entre intervalos es pequeña. *."El método más simple para obtener intervalos de confianza es asumir que la cantidad de interés es tiene una distribución gaussiana caracterizada por su forma de campana.*."(Ricci et al., 2010)

NOVEDAD

La *Novedad* en un sistema de recomendación se define como la probabilidad de que el sistema sugiera un item que el usuario no ha visto antes. Generalmente, es más importante que un usuario brinde información al sistema sobre elementos totalmente nuevos, con el objetivo de brindar una nueva perspectiva de sus preferencias al sistema. Inclusive podría ser más importante la calificación del usuario respecto a items que no ha visto antes en comparación a calificar items faltantes que son fieles al gusto del usuario.

En algunos enfoques de recomendación como el *basado en contenido* es común que las sugerencias sean excesivamente apegadas al gusto del usuario, causando que éste pierda confianza en el sistema y lo considere repetitivo.

"La forma más natural de medir la novedad en un sistema es a través de experimentaciones Online en donde el usuario es cuestionado sobre su conocimiento previo de los items recomendados" (Aggarwal, 2016).

Sin embargo, como se mencionó anteriormente en el paradigma *Online*, éste suele ser poco escalable en sistemas grandes. Por lo tanto, generalmente es mejor opción combinar un enfoque *Online* y *Offline* para realizar las mediciones de novedad en un sistema.

SERENDIPIA

La *Serendipia* es la medición del nivel de sorpresa de recomendaciones exitosas (Aggarwal, 2016).

Las ventajas de la Serendipia respecto a la Novedad es que, además de recomendar items nuevos para el usuario, también busca dar recomendaciones *no obvias* que lo sorprendan. Entonces, se puede decir que la serendipia se compone de relevancia, novedad y sorpresa.

Según (Kotkov et al., 2020), una recomendación es *Relevante* para el usuario si éste ha expresado preferencia (o lo hará) por el item. Dependiendo del escenario, el método de expresar preferencia podría variar. En adición, la *Sorpresa* es lograda cuando el sistema recomienda un item que es inesperado para el usuario y, además, se estima que no lo habría descubierto por sí mismo.

DIVERSIDAD

La *Diversidad* implica que el conjunto de recomendaciones debe ser suficientemente diverso para que, si al usuario no le gusta el $item_i$, la probabilidad de que le guste otro item dentro del conjunto siga siendo alta, logrando que el conjunto no esté compuesto únicamente de items del mismo tipo y evitando la fatiga del usuario.

Debido a esto la *Diversidad* está estrechamente relacionada con la *Serendipia* y la *Novedad*, ya que un conjunto ampliamente diverso incrementa el porcentaje de items nuevos e inesperados para el usuario.

Este aspecto se suele medir usando una representación vectorial del conjunto de recomendaciones (este concepto se explicará más adelante), definiendo así a la *Diversidad* como el **promedio** de la similitud entre cada par de items (en su representación vectorial). Si el valor del promedio es pequeño se considera una mayor diversidad.

1.3. INTRODUCCIÓN A LA BÚSQUEDA DE VECINOS CERCANOS

Los métodos de Vecinos Cercanos se apoyan en fundamentos matemáticos que deben ser abordados previamente para comprender su funcionamiento de manera profunda. Por esta razón, previo a la explicación de los Métodos de Vecinos Cercanos, se hará una breve introducción a las formas *Representación de Datos* y los métodos de *Medición de Distancias* generalmente usados para implementar ésta técnica.

REPRESENTACIÓN DE LOS DATOS

A lo largo de este documento se ha hecho uso del concepto de *item* que representa un elemento de recomendación hecho por el sistema. Estos items pertenecen a un conjunto de elementos con los que interactúa el usuario y a los que se les pueden describir mediante sus *metadatos*.

Generalmente, cuando el usuario interactúa con un sistema de recomendación, los items se muestran de forma amigable mediante interfaces que facilitan su interacción. Sin embargo, de forma interna, los algoritmos de recomendación

requieren de estructuras de datos que permiten hacer operaciones entre items sin perder sus características únicas. En el caso de los *Métodos de Vecinos Cercanos*, la forma más eficiente de representar la información de cada item es a través de **vectores**.

VECTORES

Un ***Vector***, desde un punto de vista geométrico, es un segmento de recta dirigido que corresponde a un desplazamiento entre un punto A y un punto B (Poole, 2007).

Sin embargo, desde el punto de vista algebraico, un vector es definido como un elemento dentro de un *espacio vectorial*.

Los vectores se pueden representar de las siguientes maneras:

$$\vec{AB} = (x_1, x_2, \cdots, x_n) \tag{I.6}$$

$$\vec{AB} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \tag{I.7}$$

Donde la **Ecuación I.6** define la representación de un vector mediante tuplas y la **Ecuación I.7** define su representación mediante matrices. Además, es importante puntualizar que cada x_i dentro de un vector es un número real, es decir, $x_i \in \mathbb{R}$. Por lo tanto, el vector $\vec{AB}$ pertenece al espacio $\mathbb{R}^n$, donde n representa las **dimensiones** del vector.

En el contexto de los sistemas de recomendación, los vectores se utilizan para representar de forma algebraica la información numérica de un item. Esto se hace con el propósito de realizar diferentes operaciones algebraicas entre los items para lograr calculos complejos. Además, cuando se habla de vectores que representan items con información extensa se pierde la posibilidad de visualizarlos mediante un plano debido a que las dimensiones suelen crecer en grandes cantidades.

MEDICIÓN DE DISTANCIAS

Uno de los cálculos más importantes para los sistemas de recomendación es la similitud entre items. Uno de los beneficios de trabajar con vectores es que brindan la posibilidad de usar calculos de medición de distancias que representan de forma numérica la diferencia entre un par de items. Sin embargo, existen diferentes tipos de medición de distancia entre vectores, principalmente la *Distancia Euclidiana*, *Distancia del Coseno* y *Producto Interno*.

DISTANCIA EUCLIDIANA

Según (Ricci et al., 2010) una de las mediciones de distancia más comunes es la distancia euclideana, que se define mediante la siguiente fórmula:

$$d(\vec{x},\vec{y}) = \sqrt{\sum_{k=1}^{n}(x_k - y_k)^2} \tag{I.8}$$

Donde $\vec{x},\vec{y} \in \mathbb{R}^n$ o, en otras palabras, el vector x y el vector y pertenecen al mismo campo dimensional $\mathbb{R}^n$. Para expresarlo de manera gráfica, se usará un ejemplo representativo definiendo los siguientes vectores:

$$\begin{gathered}\vec{x} = \begin{bmatrix}3\\5\end{bmatrix},\ \vec{y} = \begin{bmatrix}6\\2\end{bmatrix},\ (\vec{x},\vec{y}) \in \mathbb{R}^2 \\ d(\vec{x},\vec{y}) = \sqrt{(3-6)^2 + (5-2)^2} = \sqrt{18} \approx 4{,}24\end{gathered} \tag{I.9}$$

Como se puede ver en la **Figura** 2, el resultado de la **Ecuación I.9** fue aproximadamente igual, por lo tanto, la distancia euclidiana se puede ver graficamente mediante ejemplos representativos como la distancia entre los puntos destino entre dos vectores. Es importante recordar que en el contexto de los sistemas de recomendación el ejemplo dado es un caso excepcional debido a que un item suele estar descrito por más de dos dimensiones.

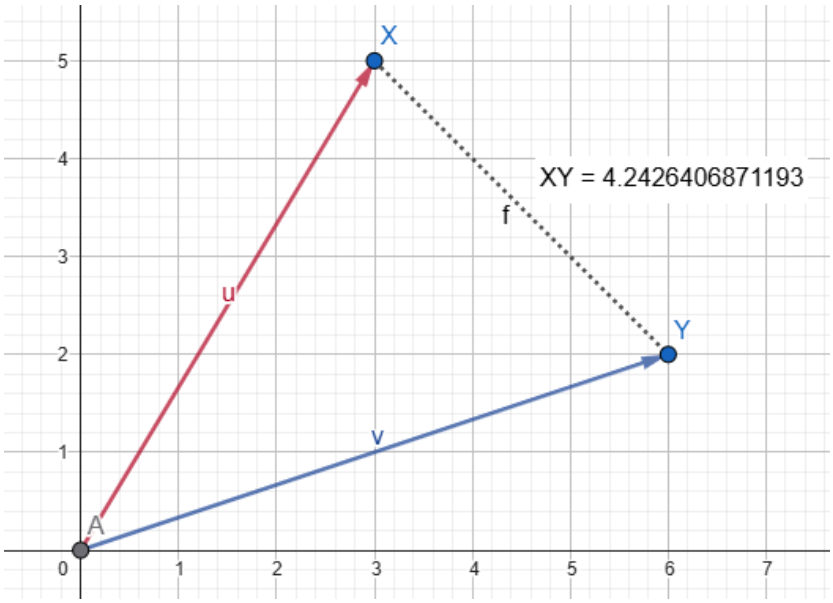

Figura 2: Representación gráfica de la Distancia Euclidiana entre dos vectores.

DISTANCIA DE COSENO

Siguiendo a (Ricci et al., 2010), la Distancia del Coseno es otra de las medidas de distancia más comunes cuando un item es considerado un vector. Esta distancia se centra en calcular el coseno del ángulo a través de la **Similaridad** definida como:

$$cos(x,y) = \frac{(\vec{x} \cdot \vec{y})}{|| \vec{x} || \, || \vec{y} ||} \tag{I.10}$$

La fórmula de la Similaridad del Coseno utiliza dos cálculos algebráicos comunes en los vectores, el ***Producto Punto*** y el calculo de ***Norma o Magnitud***.

Citando a (Poole, 2007), el *Producto Punto* se define como:

$$\vec{u} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix}, \quad \vec{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} \tag{I.11}$$

$$(u \cdot v) = u_1 v_1 + u_2 v_2 + \ldots + u_n v_n$$

Y la *Norma o Magnitud* de un vector se define como el escalar no negativo $|| \vec{v} ||$ definido por:

$$\vec{V} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} \tag{I.12}$$

$$|| \vec{V} || = \sqrt{\vec{V} \cdot \vec{V}} = \sqrt{v_1^2 + v_2^2 + \ldots + v_n^2}$$

Una vez entendiendo cómo calcular la similaridad, se calcula la distancia del coseno:

$$d(\vec{x}, \vec{y}) = 1 - cos(\vec{x}, \vec{y}) \tag{I.13}$$

Para ilustrar la distancia del coseno mediante un ejemplo visual se definen los siguientes vectores:

$$\vec{x} = \begin{bmatrix} 3 \\ 5 \end{bmatrix}, \vec{y} = \begin{bmatrix} 6 \\ 2 \end{bmatrix}$$

$$\begin{aligned} ||\vec{x}|| &= \sqrt{3^2 + 5^2} = \sqrt{34} \approx 5{,}83 \\ ||\vec{y}|| &= \sqrt{6^2 + 2^2} = \sqrt{40} \approx 6{,}32 \\ (\vec{x} \cdot \vec{y}) &= (3 \cdot 6) + (5 \cdot 2) = 28 \end{aligned} \tag{I.14}$$

$$cos(\vec{x}, \vec{y}) = \frac{28}{5{,}83 \cdot 6{,}32} \approx 0{,}759$$

$$d(\vec{x}, \vec{y}) \approx 1 - 0{,}759 \approx 0{,}241$$

Para entender las equivalencias entre la **Figura** 3 y la **Ecuación I.14** es importante mencionar que el calculo geométrico de la Similitud del Coseno se puede realizar mediante el calculo de $cos(\alpha)$ donde α representa el ángulo de separación entre $\vec{x}, \vec{y}$. Entendiendo esto, se puede visualizar que ambos resultados coinciden y representan la distancia del Coseno.

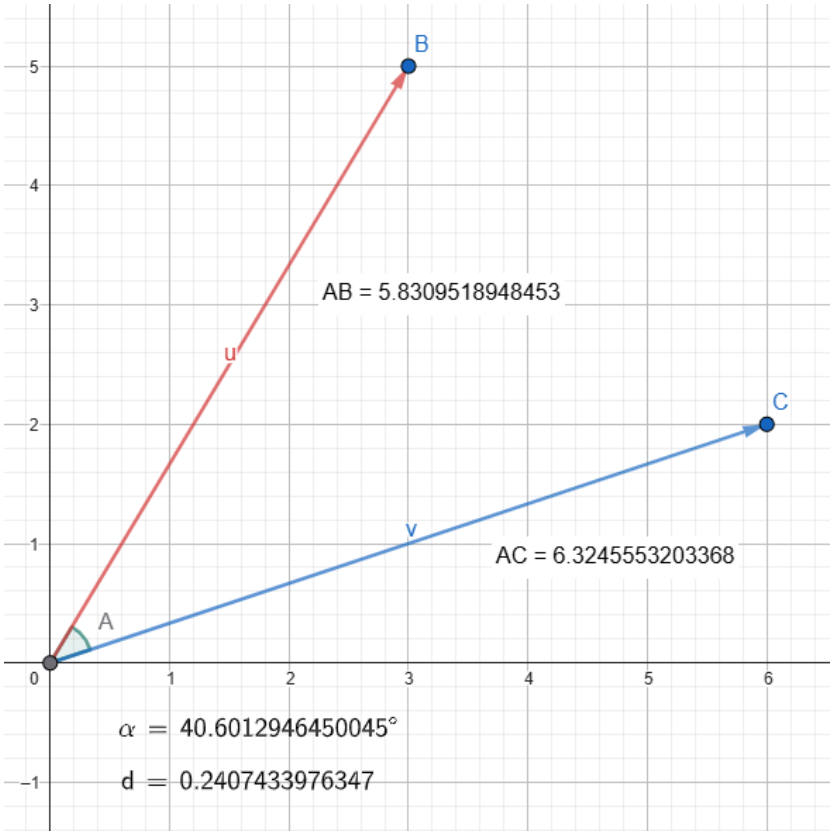

Figura 3: Representación gráfica de la Distancia de Coseno entre dos vectores.

PRODUCTO INTERNO

Según (Nicholson, 2024), el *Producto Interno* es un concepto extenso dentro del Álgebra Lineal, sin embargo, en el contexto de los Sistemas de Recomendación el concepto se puede reducir a una extensión del *Producto Punto* de dos vectores **siempre y cuando** todos los vectores cumplan con la propiedad de pertenecer al espacio de $\mathbb{R}^n$ que, en los sistemas de recomendación, se cumple en la mayoría de casos. El *Producto Interno* mide la similitud entre dos vectores y se puede calcular de la misma manera descrita en la **Ecuación I.11**

Para enteder cómo funciona el *Producto Punto* se usará nuevamente una representación geométrica mediante los siguientes vectores:

$$\vec{x} = \begin{bmatrix} 2 \\ 2 \end{bmatrix}, \ \vec{y} = \begin{bmatrix} -2 \\ -1 \end{bmatrix}, \ (\vec{x}, \vec{y}) \in \mathbb{R}^2 \tag{I.15}$$

$$(\vec{x} \cdot \vec{y}) = (2 \cdot (-2)) + (2 \cdot (-1)) = -4 - 2 = -6$$

En la **Ecuación I.15** el resultado dado es negativo debido a que los vectores apuntan en sentidos contrarios como se puede ver en la **Figura** 4. Si el resultado obtenido fuera 0 significa que los vectores son perpendiculares y en el caso de ser positivos significa que los vectores apuntan en direcciones parecidas. Entre más alto el valor del *Producto Interno* más alta la similitud de los vectores. Se debe tomar en cuenta que el valor obtenido de esta métrica no está normalizado y es afectado por la *magnitud* de los vectores.

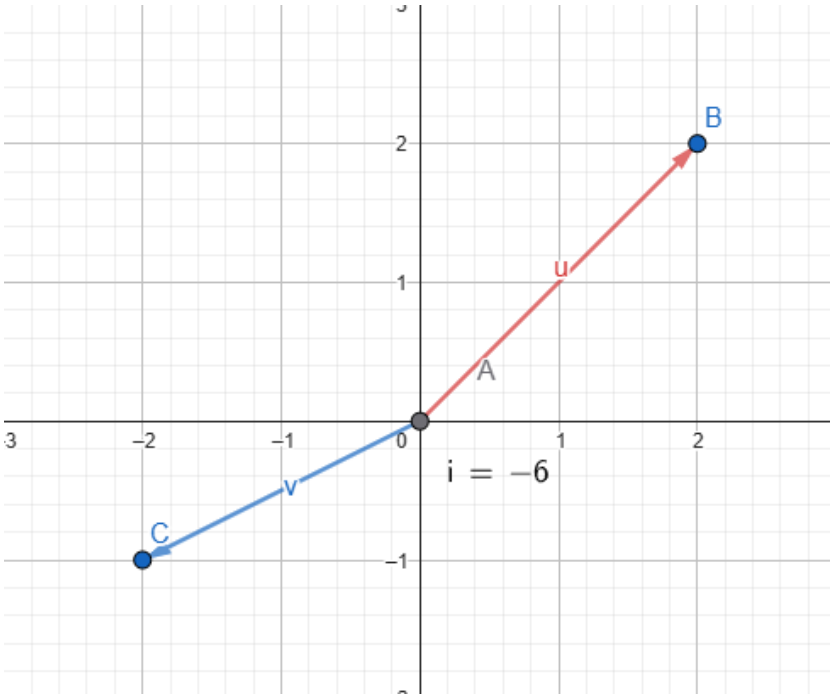

Figura 4: Representación gráfica del Producto Punto entre dos vectores.

1.4. BÚSQUEDA DE VECINOS CERCANOS

Uno de los métodos de búsqueda de recomendaciones más populares es el método de búsqueda de vecinos más cercanos más comunmente denominado como K-Nearest Neighbors (k-NN). Este término es usado debido a que esta búsqueda se centra en encontrar los k puntos más cercanos (o también denominados como *Vecinos*) a partir de un registro de entrenamiento(Ricci et al., 2010).

Se puede definir a la *Búsqueda de Vecinos más cercanos* de la siguiente manera:

Definición 1.4.1. Dado un punto petición q al que se le quiere conocer su clase l, y un conjunto de entrenamiento $X = \{\{x_1, l_1\} \ldots \{x_n\}\}$, donde x_j es el j-ésimo elemento y l_j representa la etiqueta de la clase, los métodos *k-NN* encuentran el subconjunto $Y = \{\{y_1, l_1\} \ldots \{y_k\}\}$ tal que $Y \in X$ y, además, $\sum_1^k d(q, y_k)$ [8] se minimiza. El subconjunto Y contiene los k puntos en X que son los más cercanos al punto petición q (Ricci et al., 2010).

Antes de implementar un método de *Vecinos más Cercanos* es importante definir una *funcion de similaridad*, según (Aggarwal, 2016) la más común es la *Distancia de Coseno*. Esta función de similaridad, independientemente de la elegida, cumple la función de predecir la preferencia del usuario respecto a un item que no ha evaluado.

Una vez elegida la función de similaridad, el siguiente paso es definir el valor de k que, en el punto de vista de (Ricci et al., 2010) es el reto más importante al usar los métodos de vecinos cercanos, debido a que un valor pequeño de k incrementa la cantidad de *ruido*[9] en los resultados, sin embargo, si k es muy grande, el vecindario podría incluir muchos puntos de diversas clases.

En la **Figura** 5 se muestra un ejemplo representativo de la *Búsqueda de Vecinos Cercanos* que representa a dos grupos o clasificaciones, uno representado por el color rojo y otro por el color azul, y una petición representada como un triangulo. En la imágen de la izquierda se muestran los items sin clasificar, y en la imágen de la derecha se visualiza el valor de k a través de círculos punteados, el circulo más pequeño representa $k = 1$ y el circulo más grande $k = 7$

[8] Representa la sumatoria de distancias entre q y cada elemento $y_k \in Y$, pudiendo calcularse ya sea con distancia euclidiana, distancia de coseno o producto interno.

[9] **Ruido**: Es definido como un error aleatorio o una varianza en una variable medida (Alasadi & Bhaya, 2017).

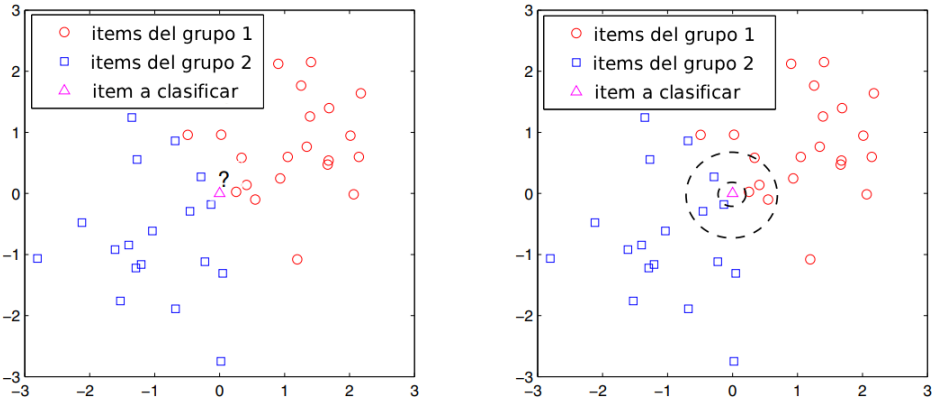

Figura 5: Ejemplo de Búsqueda de Vecinos Cercanos (Ricci et al., 2010).

A través de esta descripción del funcionamiento de los algoritmos *k-NN* se puede intuir que estos métodos son más simples a comparación de otros relacionados con *Machine Learning* y, además, están estrechamente relacionados con el concepto de *Vecindario* mencionado en la sección de *Filtros Colaborativos de un Sistema de Recomendación*. Los algoritmos *k-NN* suenan ideales debido a que no necesitan mantener un modelo de aprendizaje, pueden ser sencillos de implementar y, además, cuentan con las ventajas de los *Filtros Colaborativos*, sin embargo, su simpleza es computacionalmente costosa.

"El reto principal relacionado al uso de este enfoque es la alta complejidad computacional. Se debe notar que el vecino más cercano de cada item $x_j \in X$ *debe ser determinado, y el tiempo requerido por cada calculo es lineal al tamaño de* X*. Por lo tanto, la complejidad computacional se describe como* $| x_j | \times | X |$*"* (Aggarwal, 2016).

Es importante recordar que $| x_j |$ representa la *dimensión* del vector x_j previamente definida como n. Por lo tanto, la fórmula de complejidad computacional mencionada por (Aggarwal, 2016) expresa que el tiempo necesario para calcular los vecinos cercanos **por item** depende del tamaño del vector x_j multiplicado por el tamaño total del conjunto X o, en otras palabras, la cantidad total de items en el sistema.

A pesar de este problema, el enfoque *k-NN* brinda resultados muy efectivos, incluso convirtiéndose en uno de los métodos principales del *enfoque colaborativo*, gracias a esto se le han hecho mejoras que aumentan su rendimiento y lo hacen viable en sistemas con cantidades masivas de información.

1.5. BÚSQUEDA APROXIMADA DE VECINOS CERCANOS

La *Búsqueda Aproximada de Vecinos Cercanos (ANN)* es una de las propuestas más avanzadas de optimización para los enfoques *k-NN* redefiniendo el objetivo principal para obtener resultados con tasa de efectividad similar pero con un costo computacional más bajo.

Definición 1.5.1. Sea d_k la distancia del **verdadero** k-ésimo vecino más cercano, el algoritmo *ANN* devuelve un vecino cuya distancia d' cumple con:

$$d' \leq (1+\varepsilon) \cdot d_k \tag{I.16}$$

Donde $(1+\varepsilon)$ representa el grado de proximidad respecto a los vecinos cercanos, por lo tanto, si $\varepsilon = 0$ entonces nos referimos al método *k-NN* debido a que no existe tolerancia de proximidad y, por el contrario, si $\varepsilon = 0{,}1$ entonces el vecino resultante puede estar hasta $10\,\%$ más lejos que el verdadero k-ésimo vecino más cercano (Liu et al., 2004).

Este enfoque de proximidad ha sido estudiado con el fin de lograr que la *Búsqueda de Vecinos Cercanos* se vuelva computacionalmente viable en sistemas de alta dimensionalidad, y gracias a estas investigaciones se han formulado diferentes metodologías y algoritmos que enfocan la proximidad de maneras diferentes. En la **Figura** 6 se muestra una gráfica que compara el rendimiento de los algoritmos basados en *ANN* según la cantidad de peticiones por segundo realizadas y la **Sensibilidad o Recall** de los resultados obtenidos. Además, a través de dicha gráfica se puede observar que existe un campo diverso de algoritmos que aprovechan este concepto, sin embargo, cada algoritmo se basa en diferentes enfoques de *ANN* como *Enfoques basados en hashing*, *Enfoques basados en árboles* o *Enfoques basados en Grafos*.

En el ámbito de los sistemas de recomendación enfocados en contenido multimedia, los enfoques más importantes de *ANN* son los que están basados en árboles y grafos.

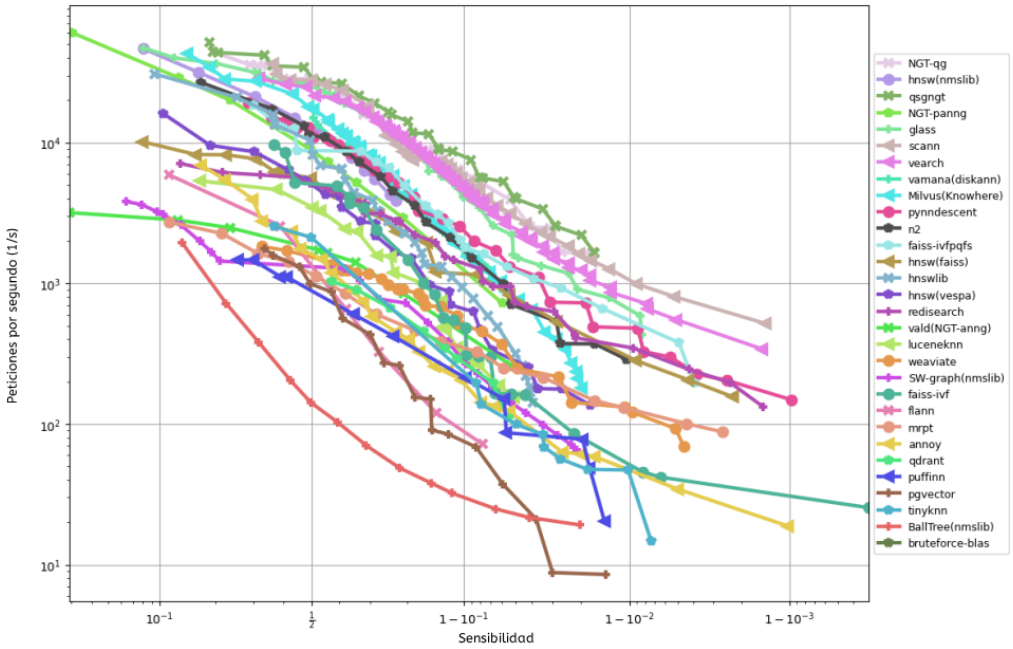

Figura 6: Rendimiento de los algoritmos *ANN* más usados en la actualidad (Aumüller & Ceccarello, 2023).

Según (Team, 2025), los sistemas que se benefician del uso de *ANN* son los que cumplen con las siguientes características:

- **Conjuntos de Datos Masivos:** Es el caso de uso esencial de *ANN*, de lo contrario, un enfoque de Vecinos Cercanos sería inviable.
- **Datos con alta dimensionalidad:** Recordando que el crecimiento de la dimensionalidad hace que la complejidad de *k-NN* explote. Por lo tanto, *ANN* logra reducir la dimensionalidad mediante técnicas de efectividad y optimizan el costo computacional.
- **Aplicaciones en Tiempo Real:** La velocidad de *ANN* logra respuestas en tiempo real que sería imposible lograr con los métodos de vecinos cercanos tradicionales.
- **Apertura a Aproximaciones:** Si el sistema puede tolerar resultados aproximados, entonces usar *ANN* es ideal para mejorar el rendimiento, de lo contrario, un enfoque de vecinos cercanos se vuelve inviable.

1.6. SISTEMAS DE RECOMENDACIÓN SENSIBLES AL CONTEXTO

La forma en que se ha abordado el problema principal de los sistemas de recomendación ha evolucionado a lo largo del tiempo, sin embargo, la mayoría de investigaciones se enfocan en la interacción *item - usuario* o *usuario - item* y no se suele tomar en cuenta ningún tipo de contexto adicional que podría alterar la calidad de las recomendaciones.

Sin embargo, con la evolución de los Sistemas de Recomendación, ha surgido la necesidad de categorizar a los sistemas que son **sensibles al contexto** para encontrar métodos y enfoques que logren brindar al sistema la información necesaria para entender las necesidades y preferencias del usuario. Estos sistemas de recomendación sensibles al contexto se definen formalmente como *Context-Aware recommender systems (CARS)*.

Definición 1.6.1. En los sistemas de recomendación, el **Contexto** se puede definir como cualquier información que puede ser usada para caracterizar la situación de una entidad, donde una entidad puede representar a una persona, un lugar o un objeto computacional. Esta información adicional puede ayudar al sistema a ser más preciso. (Mateos & Bellong, 2024).

Los *CARS* son aquellos sistemas que buscan entender las preferencias del usuario incorporando información contextual en el proceso de recomendación. Esto implica que una calificación dada por el usuario ahora es afectada también por el contexto en el que se realizó. Por lo tanto, según (Ricci et al., 2010), una *calificación* ahora se puede definir de la siguiente manera:

$$Usuario \times Item \times Contexto \rightarrow Calificacion \tag{I.17}$$

Es importante mencionar que el Contexto puede definir diferentes aspectos del contexto del sistema ya sea tiempo, localización, compañía, propósito, y muchos más.

1.6.1. PREFILTRADO CONTEXTUAL

En el *Prefiltrado Contextual* el contexto específico determinará qué información es la más relevante respecto a las preferencias del usuario, obteniendo un subconjunto de elementos que servirá como base para obtener recomendaciones mediante los algorítmos de recomendación clásicos. El prefiltrado de items mediante contexto aumenta la diversidad de recomendaciones gracias a la división de datos en diferentes escenarios contextuales, sin embargo, tratar de recrear un contexto específico puede llevar a un problema de **escasez de datos**, obteniendo una recomendación altamente limitada (Mateos & Bellong, 2024).

En general, cuando se busca desarrollar un sistema de recomendación, usar un prefiltrado contextual puede resultar en desventajas importantes que afectan al desempeño del sistema, principalmente si se busca un enfoque generalizable a distintos dominios. A pesar de sus limitaciones en escenarios generales, el prefiltrado sigue siendo útil cuando se diseña en campos específicos.

1.6.2. HEURÍSTICAS DE PREFILTRADO

Para obtener un subconjunto relevante de items se suelen usar diversas técnicas de prefiltrado, una de ellas es el uso de **Heurísticas**.

Definición 1.6.2. Las **Heurísticas** son criterios, métodos o principios para decidir entre diversas ramas de acción a aquella que prometa ser la más efectiva para lograr cierto propósito. Una heurística podría ser un atajo usado para guiar cierta acción (Pearl, 1984).

El objetivo de una Heurística no es garantizar la mejor solución a un problema, sino proponer una solución **suficientemente buena** en un tiempo razonable. En el caso de los sistemas de recomendación sensibles al contexto, las heurísticas permiten seleccionar de manera rápida un subconjunto de ítems relevantes para un usuario bajo un contexto particular, obteniendo una muestra significativa sobre el cual podrían operar los algorítmos de recomendación tradicionales.

Definición 1.6.3. Los **Algorítmos Heurísticos** son una clase de algorítmos matemáticos que utilizan diversas técnicas heurísticas para la resolución de problemas complejos (Proy de Santiago et al., 2024).

Las técnicas heurísticas más importantes para los algorítmos propuestos en el presente proyecto son las *heurísticas voraces* y las *heurísticas tabú*.

Antes de explicar de manera detallada estas heurísticas, es importante comprender el concepto de **espacio de estados** y **Funciones Heurísticas**.

Definición 1.6.4. El **espacio de estados** es el conjunto de todas las posibles configuraciones o situaciones que un sistema o problema puede alcanzar. Cada **estado** representa una situación particular del sistema, y una **transición** representa una acción o movimiento permitido con un costo asociado (Heusner et al., 2017).

En ciertos algorítmos el *espacio de estados* es un conjunto predefinido que representa el espacio total con el que se trabajará, sin embargo, igualmente existen **modelos generativos de espacio de estados** que generan estados bajo demanda en lugar de tenerlos definidos desde el inicio. Los **modelos generativos** brindan la posibilidad de explorar **espacios grandes** sin necesidad de procesar toda la información. Estos algoritmos se apoyan en la estimación de costos o distancias entre estados para decidir cuál estado expandir en cada paso.

Definición 1.6.5. Las **Funciones Heurísticas** son funciones que estiman el **costo** o **distancia** desde el estado inicial hasta el estado final (Heusner et al., 2017).

Las funciones heurísticas nos permiten estimar el costo o distancia entre estados mediante cálculos o procedimientos algorítmicos, sin embargo, en muchas ocasiones son llamadas *cajas negras* ya que, en ciertas situaciones, no importa cómo calcula la heurística, solo importa la información que retorna.

HEURÍSTICAS VORACES (GREEDY)

Los *Algorítmos Voraces* representan una estrategia para generar soluciones *suboptimas* mediante ramas de decisiones irrevocables. Cada una de estas decisiones representa la mejor opción en el momento en que se toman (Khuller et al., 2018).

En el ámbito de la algorítmia, los algorítmos voraces son usados para encontrar respuestas **máximas** o **mínimas** a un problema. Sin embargo, en sistemas complejos, también suelen ser usados para obtener soluciones **aproximadas** encontrando respuestas que podrían ser *subóptimas* pero cumpliendo con un mínimo de rendimiento.

A continuación se describen algunos algorítmos clásicos que utilizan heurísticas voraces para la exploración del espacio de estados.

ALGORÍTMO VORÁZ DE BÚSQUEDA POR NIVELES

El *Algorítmo Voráz de Búsqueda a Profundiad* o *(GBFS)* es un algorítmo de búsqueda informada y voráz que intenta alcanzar la meta priorizando los estados que parecen más cercanos según la heurística, aunque no garantiza el costo mínimo total. Recibe como entrada un *espacio de estados* estático o generativo y devuelve un plan que alcanza la meta sin embargo, existen casos *insolubles* que indican que no se puede llegar a la meta.

Según (Heusner et al., 2017) la *GBFS* se basa en la suposición de que los estados con menor **valor heurístico** (costo) forman parte del camino más barato hacia la meta, por esta razón se considera un *algoritmo voráz*. Este algoritmo no necesita conocer todo el espacio de estados para funcionar debido a que únicamente opera en los estados que ya han sido generados. En cada paso el estado actual se expande al que tiene el *menor valor heurístico* entre todos los estados que han sido generados que no han sido visitados previamente, esto se repite hasta generar el estado destino.

Este criterio *voráz* es la razón por la que este algoritmo no genera las soluciones más cortas debido a que en la mayoría de casos una decisión local no es suficiente para predecir el mejor resultado global, inclusive, no hay garantías de que el resultado tenga un mínimo de calidad. Debido a la inexactitud de la heurística, *GBFS* puede quedarse atrapado en mesetas donde muchos estados tienen el mismo valor heurístico, o en mínimos locales donde la heurística guía hacia caminos no óptimos, lo que puede aumentar el tiempo de búsqueda o producir soluciones subóptimas.

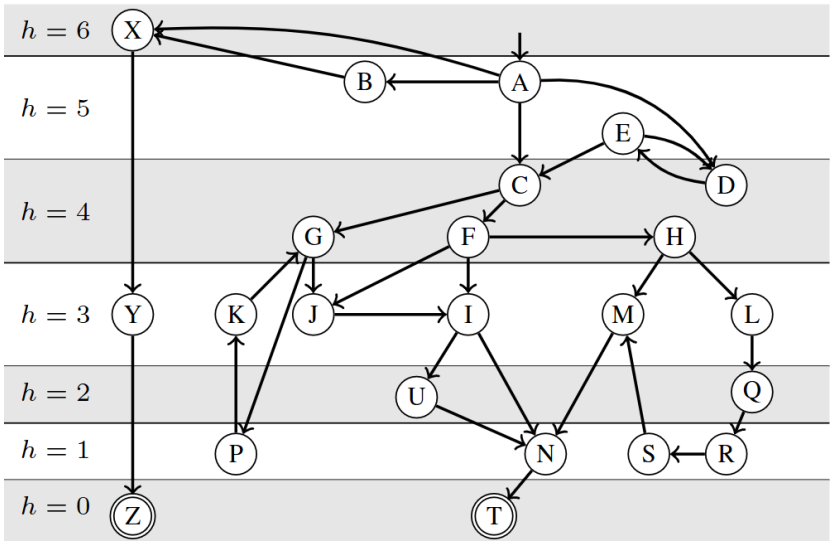

Figura 7: Ejemplo de GBFS (Heusner et al., 2017).

En la **Figura** 7 se define el estado inical el nodo A y el nodo objetivo Z. Cada nivel horizontal indica el cálculo h de la distancia mediante la *función heurística*. Además, se puede ver que existen diferentes caminos para llegar al nodo Z y *GBFS* podría retornar un camino menos óptimo como el más prometedor.

ALGORITMO DE BÚSQUEDA A*

El algoritmo *A** es un algoritmo que aprovecha una *función heurística* para guiar su búsqueda hacia el nodo objetivo. Este algoritmo combina los mejores aspectos del *Algoritmo Dijkstra*[10] y *GBFS* (Kumar, 2024). Esta combinación de algorítmos se puede definir de manera formal mediante la combinación de funciones $g(n)$ que representa el costo o distancia para alcanzar el nodo n desde el nodo inicial, y $h(n)$ el costo de llegar desde un nodo n hasta el nodo destino:

$$f(n) = g(n) + h(n) \tag{I.18}$$

Así, $f(n)$ representa el costo estimado de la solución más barata que pase por el nodo n. Por lo tanto, si se busca encontrar la solución más barata, la estrategia es priorizar el nodo con el valor mínimo de $g(n) + h(n)$. Esta combinación es razonable y, junto a una buena *función de heurística*, la *búsqueda A** garantiza soluciones óptimas (Russell & Norvig, 2009).

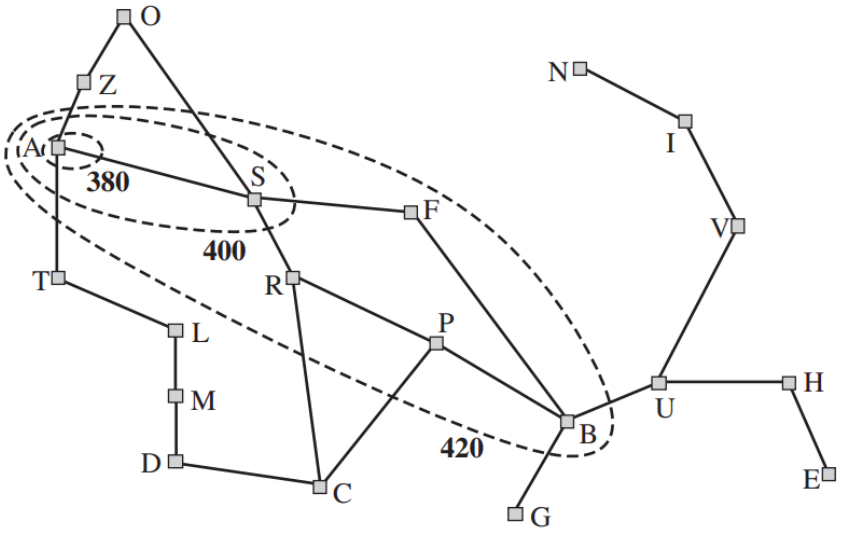

Figura 8: Ejemplo del Algoritmo A* (Russell & Norvig, 2009).

[10]**Algoritmo Dijkstra:** Algoritmo que encuentra el camino más corto a todos los nodos desde un único nodo orígen (Javaid, 2013).

En la **Figura** 8 se representa un problema clásico de inteligencia artíficial usado para explicar el algoritmo *A** llamado *El Mapa de Rumanía*. Cada nodo representa una ciudad de Rumanía y las conexiones entre nodos representan las carreteras entre ciudades. El problema pide encontrar la ruta más corta desde *Arad* hasta *Bucarest*. En los contornos se muestran los valores $f(x)$ de los nodos que representan el camino más corto desde el inicio.

HEURÍSTICAS DE BÚSQUEDA TABÚ

En base a (Gendreau, 2003), la *Búsqueda Tabú* es un método de optimización matemática que mejora el rendimiento de una búsqueda local mediante el uso de estructuras de memoria que describen las soluciones procesadas anteriormente. Además, se considera un *algoritmo metaheurístico* [11] usado para resolver problemas de *optimización combinatoria*.

El término *tabú* proviene de la idea focal de este algoritmo que busca censurar una solución potencial que ya ha sido procesada con el objetivo de no regresar al mismo estado de manera repetida. La *Búsqueda Tabú* utiliza un enfoque de búsqueda local de vecindarios para moverse de una solución x a una solución x' dentro del vecindario de x hasta que un criterio de parada se cumpla (Cvijović & Klinowski, 2002).

Cuando se utiliza un algoritmo de búsqueda *ingenuo* es común que los estados se estanquen en óptimos locales que no representan la mejor solución posible, sin embargo, en la *búsqueda tabú* se modifica la estructura del vecindario para prohibir revisitar estados y fomentar la diversidad de soluciones procesadas. Para lograr una modificación correcta de vecindarios se utilizan estructuras de memoria que permitan la prohibición de estados ya visitados, una de las estructuras más importantes es la **Lista Tabú**.

Definición 1.6.6. La **Lista Tabú** es una estructura de memoria a corto plazo que contiene soluciones que han sido visitadas en un pasado reciente (con un lapso de n iteraciones) (Gendreau, 2003).

Sin embargo, existen también variaciones de la *Lista Tabú* que prohiben soluciones con ciertos atributos no deseados o que previenen movimientos futuros. Esta variación puede ser más efectiva en ciertos dominios pero conlleva el riesgo de censurar más de un estado óptimo que no ha sido procesado anteriormente.

[11] **Algoritmo Metaheurístico:** Se definen como algoritmos de optimización usados para enfrentar problemas complejos que no se pueden resolver con métodos tradicionales. En general, están inspirados en fenómenos naturales (Almufti et al., 2023).

Ejemplo 1.6.1. Basándose en (Gendreau, 2003), el problema clásico de optimización **Traveling Salesman Problem (TSP)** es un ejemplo claro del uso eficiente de *búsquedas tabú*. El problema plantea un conjunto de ciudades que un vendedor debe visitar en su totalidad, el objetivo es encontrar el orden de visita que minimice la distancia total recorrida. Actualmente no existe un algoritmo eficiente que encuentre soluciones óptimas en todos los casos posibles, por lo tanto, la *búsqueda tabú* es un método heurístico que busca aproximarse a la solución óptima tanto como sea posible. La *búsqueda tabú* representa cada estado como una ruta completa de ciudades (independientemente de la distancia recorrida entre ellas), el **vecindario** representa a las variantes más cercanas de cada ruta.

En la **lista tabú** se almacenan soluciones procesadas anteriormente dentro de un rango de iteraciones n para evitar ciclos en soluciones subóptimas, posteriormente el algoritmo se mueve en el campo del **vecindario** hasta cumplir una condición de parada. El resultado final es el mejor estado encontrado durante la ejecución del algoritmo.

A pesar de que la *búsqueda tabú* optimiza problemas combinatorios con altos costos computacionales, es importante enfatizar que al ser una *heurística* únicamente brinda soluciones aproximadas.

Capítulo II

MARCO METODOLÓGICO

En el presente capítulo se describirán cuatro metodologías enfocadas en optimizar el desarrollo total del sistema mediante un enfoque específico con el objetivo de comparar su funcionamiento en el desarrollo del Sistema de Recomendación planteado.

2.1. CROSS INDUSTRY STANDARD PROCESS FOR DATA MINING (CRISP-DM)

Según (Wirth & Hipp, 2000) la metodología *CRISP-DM* proveé una visión general del ciclo de vida de un proyecto basado en *Minería de Datos*. Este ciclo de vida se descompone en seis fases: *Entendimiento del Negocio*, *Entendimiento de los Datos*, *Preparación de los Datos*, *Modelado*, *Evaluación* y *Despliegue*. Estas fases siguen un proceso secuencial no estricto y se puede adaptar a las necesidades del proyecto, sin embargo, existe una secuencia frecuente basada en las dependencias de cada fase.

En el contexto del proyecto actual, las metodologías basadas en Minería de Datos deben ser tomadas en cuenta debido a que los Sistemas de Recomendación forman parte del ámbito de dicha disciplina y, *CRISP-DM* ha sido diseñado para estructurar proyectos de Minería de Datos sin establecer un objetivo en específico (clasificación, clustering, regresión, recomendación).

En la figura **Figura** 1 se puede visualizar que las fases de *CRISP-DM* son cíclicas debido a la naturaleza de la *Minería de Datos*, ya que estos se retroalimentan de sus propios resultados, ya sean errores o aciertos.

Cada una de las fases que componen a la metodología *CRISP-DM* cumplen con un propósito específico, a continuación se describe la función de cada una de las fases basándose en (Schröer et al., 2021) y (Wirth & Hipp, 2000).

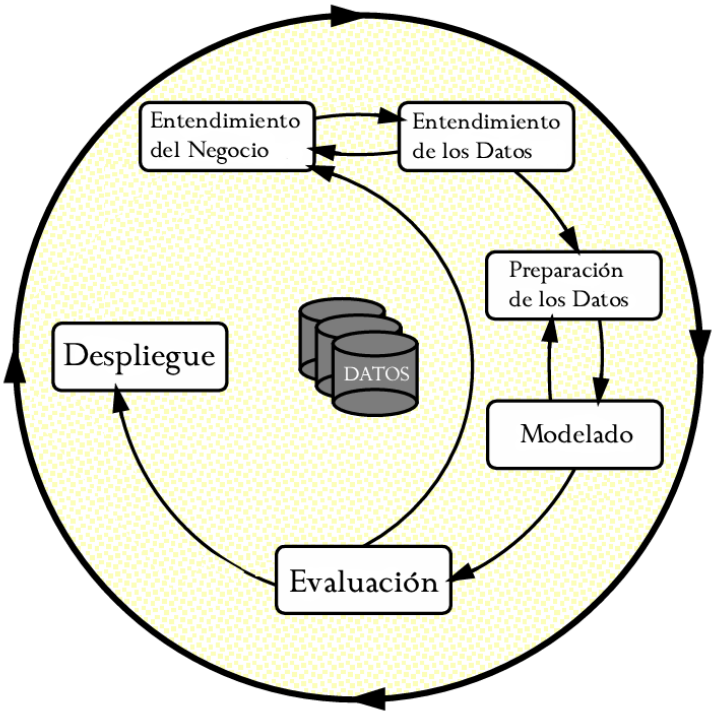

Figura 1: Fases de la Metodología CRISP-DM (Wirth & Hipp, 2000).

FASE 1: ENTENDIMIENTO DEL NEGOCIO

La fase inicial denominada como *Entendimiento del Negocio* se enfoca en entender los objetivos y requerimientos del proyecto, convirtiéndo este conocimiento en la definición de un problema a resolver mediante *Minería de Datos* y en un plan de proyecto preliminar diseñado para cumplir dichos objetivos.

La situación del negocio debe ser evaluada para obtener una visión general de los recursos necesarios. Uno de los puntos más importantes de esta fase es la determinación del **Objetivo de la Minería de Datos**. El primer paso es definir el tipo de Minería de Datos que se desarrollará (en este caso, Recomendación) y el criterio de éxito.

FASE 2: ENTENDIMIENTO DE LOS DATOS

La fase de *Entendimiento de los Datos* empieza con una colección de datos inicial y es seguida de actividades que fomenten el entendimiento de los datos para identificar problemas de calidad, descubrir perspectivas iniciales o detectar subcolecciones para identificar información escondida.

FASE 3: PREPARACIÓN DE LOS DATOS

La fase de *Preparación de los Datos* se enfoca en construir el *Dataset* final en base a la primera recolección de datos. Estas tareas de preparación se deben ejecutar múltiples veces. Estas actividades incluyen tareas de selección de tablas, limpieza de datos, construcción de nuevos atributos y transformación de datos para herramientas de modelado.

Además, se debe realizar un proceso de selección de datos definiendo criteros de inclusión y exclusión. Los datos de mala calidad deben ser procesados mediante una tarea de limpieza de datos.

FASE 4: MODELADO

Esta fase consiste en seleccionar una técnica de minería de datos dependiendo el problema del negocio y los datos disponibles. En general, se pueden usar todas las técnicas siempre y cuando se justifiquen a través de criterios de evaluación, o de lo contrario, es recomendable evaluar el rendimiento de diferentes técnicas y determinar a los que lograron un mejor rendimiento.

FASE 5: EVALUACIÓN

En este momento del desarrollo ya se tiene uno o más modelos que aparentan buena calidad desde un punto de vista analítico. Sin embargo, antes de proceder al despliegue es importante evaluar profundamente al sistema, principalmente identificando si existe algún objetivo del negocio que no ha sido cumplido satisfactoriamente. Además de evaluar al sistema a profundidad, también se debe evaluar el proceso de desarrollo llevado a cabo.

FASE 6: DESPLIEGUE

La creación del modelo generalmente no representa el final del proyecto, por el contrario, la experiencia obtenida en el uso de diferentes modelos de Minería de Datos debe ser organizada y reportada de forma detallada para elaborar manuales de usuario. Sin embargo, esta fase puede ser tan simple como generar un reporte de usuario o tan complejo como aplicar diferentes procesos de Minería de Datos.

En muchos casos, el usuario final será el encargado del despliegue y no el analísta de datos.

2.2. METODOLOGÍA SCRUM

Basándose en (Pressman, 2010), la *Metodología SCRUM* es un método de desarrollo ágil de software centrado en guiar actividades de desarrollo dentro de un proceso de análisis a través de fases secuenciales estructurales: requerimientos, análisis, diseño, evolución y entrega. La *SCRUM* acentúa el uso de un conjunto de patrones de proceso del software que han demostrado ser eficaces para proyectos con plazos de entrega **muy apretados**, requerimientos cambiantes y negocios críticos.

En proyectos relacionados con los *Sistemas de Recomendación* la metodología *SCRUM* es útil por su enfoque iterativo, incremental y adaptativo.

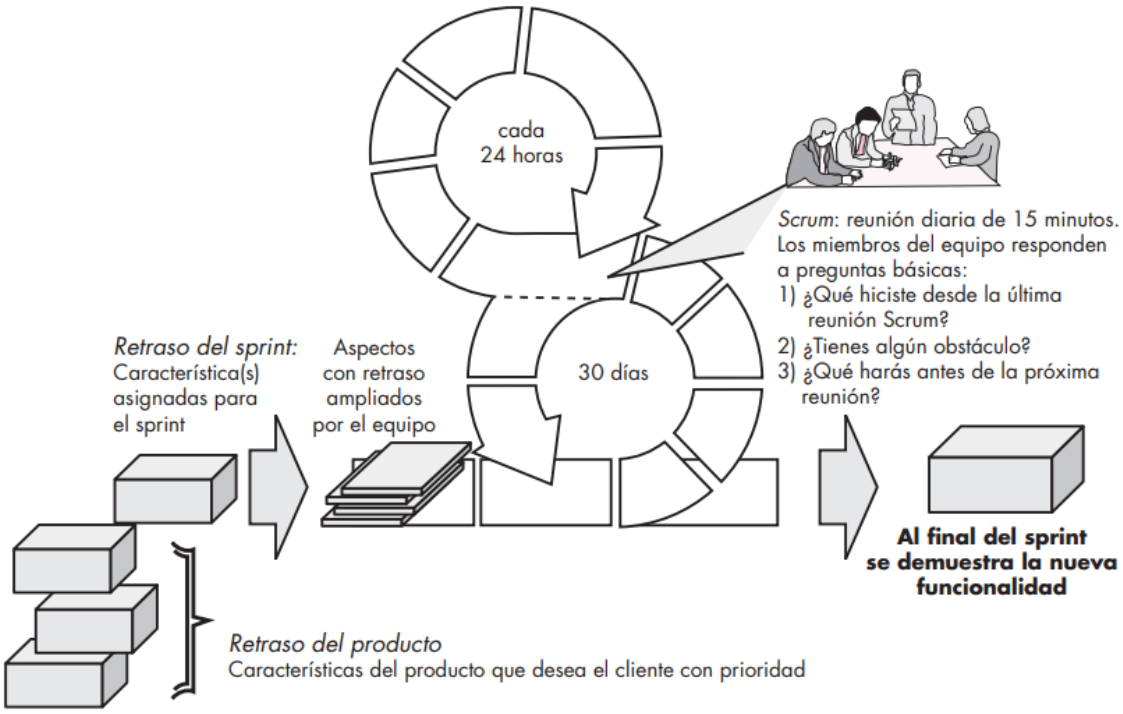

Figura 2: Descripción del ciclo de vida SCRUM (Pressman, 2010).

En la **Figura** 2 se muestra el ciclo de vida de un sistema basado en la metodología *SCRUM*, además, se puede ver que se incorpora un conjunto de patrones del proceso que ponen énfasis en las prioridades del proyecto, las unidades de trabajo agrupadas, la comunicación y la retroalimentación frecuente con el cliente.

Siguiendo a (Pressman, 2010), algunos de los conceptos más importantes para entender la metodología *SCRUM* son los siguientes:

Definición 2.2.1. El **Retraso** se define como la lista de prioridades de los requerimientos o características del proyecto que dan al cliente un valor del negocio.

Definición 2.2.2. Los **Sprints** consisten en unidades de trabajo que se necesitan para alcanzar un requerimiento definido en el retraso que debe ajustarse en una caja de tiempo.

Además de estos conceptos, existen roles, artefactos y ceremonias que describen el proceso de desarrollo mediante la metodología *SCRUM*, basándose en (Sachdeva, 2016) se definen de manera general los conceptos representados en **Figura** 3.

Figura 3: Conceptos Fundamentales de la Metodología SCRUM (Sachdeva, 2016)

ROLES

Una ventaja importante de la Metodología *SCRUM* es que, en comparación con otras metodologías de Ingeniería de Software, no requieren de un lider de proyecto. En compensación, se dividen las responsabilidades en tres roles importantes.

- **Dueño del Producto:** Es el responsable de visión general del producto. Reune y prioriza los requerimientos del producto.
- **Scrum Master:** El Scrum Master es el responsable de que las reglas de la metodología se sigan de manera apropiada, además de tomar control de problemas emergentes.
- **Equipo:** Es un conjunto organizado de personas responsables de la creación y de la calidad del producto. Este equipo debe incluir los roles de testers, diseñadores y *dev ops*.

ARTEFACTOS

En *SCRUM* los **artefactos** son los elementos clave de información que se utilizan para asegurar la transparencia, inspección y adaptación en el desarrollo del producto.

- **Cartera de Productos:** Enlista los requerimientos del producto que deben ser desarrollados. Es la lista principal de todas las funcionalidades deseadas en el producto final.
- **Plan de Lanzamiento:** Plan que describe los objetivos esperados en el lanzamiento, los elementos prioritarios pertenecientes a la cartera de productos y las funcionalidades que el lanzamiento agregará. Además, establece una posible fecha de entrega y el costo esperado.
- **Pila de Sprint:** Consiste de un conjunto de tareas derivadas de la Cartera de Productos segmentando los requerimientos en tareas específicas. Esta pila de sprint esta diseñada para ser modificada únicamente por el equipo de trabajo.
- **Gráfico de Trabajo Pendiente:** Durante un sprint los elementos visuales pueden ser de gran ayuda para el equipo de desarrollo. Este gráfico se centra en la cantidad de trabajo pendiente en la Pila de Sprint y el tiempo estimado restante para la entrega.

CEREMONIAS

Las ceremonias son reuniones que buscan dar una estructura clara a cada sprint.

- **Planeación de Sprint:** Esta reunión planea una nueva iteración. En general, se dividen en dos partes: Determinar qué se va a hacer en el siguiente Sprint y cómo será construido el producto.
- **Revisión de Sprint:** Toma lugar al final de un Sprint y tiene como objetivo que el equipo presente las funcionalidades desarrolladas al dueño del producto.
- **Retrospectiva del Sprint:** Se centra en discutir los logros y las áreas de oportunidad durante el proceso de desarrollo.
- **Reunión Diaria de SCRUM:** Esta enfocada en que los integrantes del equipo expliquen en qué han estado trabajando desde la última reunión diaria y lo que planean hacer hasta la siguiente reunión.

Según (Pressman, 2010), una de las características más importantes de *SCRUM* es que supone la existencia del caos. Esta metodología permite que un equipo de software trabaje con éxito en un mundo en el que es imposible eliminar la *incertidumbre*.

2.3. PROGRAMACIÓN EXTREMA (XP)

Citando a (Solis, 2003), la *Programación Extrema* es una metodología de desarrollo de software que se basa en la simplicidad, la comunicación y el reciclado continuo de código. Además, siguiendo a (Pressman, 2010), se dice que *XP* es el enfoque más utilizado del desarrollo de software ágil.

Los objetivos de *XP* se centran en la satisfacción del cliente, enfocandose en entregar un producto alineado con las necesidades del cliente.

VALORES XP

En (Pressman, 2010) se define un conjunto de cinco valores que establecen el fundamento para los desarrollos basados en *XP*: comunicación, simplicidad, retroalimentación, valentía y respeto.

- Para lograr la **Comunicación** entre los integrantes del equipo del desarrollo y agentes externos, se pone el énfasis en la colaboración estrecha entre los clientes y los desarrolladores para comunicar conceptos importantes y retroalimentación continua.
- La **Simplicidad** se logra restringiendo a los desarrolladores para que diseñen sólo para las necesidades inmediatas, en lugar de considerar las del futuro. Si se necesita mejorar el diseño, se realizarán las mejoras en el futuro.
- Para fomentar la **Retroalimentación** se definen tres fuentes esenciales: El software implementado, el cliente y otros miembros del equipo. El software brinda retroalimentación mediante *priebas unitarias*.
- Seguir la metodología *XP* requiere de **Valentía o Disciplina**. Un equipo *XP* debe tener la valentía de diseñar centrándose en el hoy y reconocer que los requerimientos futuros podrían cambiar mucho y deberán modificar el diseño y código implementado.
- Para apegarse a cada una de las reglas *XP* se requiere **Respeto** entreo los miembros del equipo e indirectamente para el software en sí mismo.

FASES DE LA PROGRAMACIÓN EXTREMA

La *Programación Extrema* engloba un conjunto de reglas y prácticas que ocurren en el contexto de cuatro actividades estructurales representadas en la **Figura** 4. A continuación se describen de manera general las fases inherentes a *XP*.

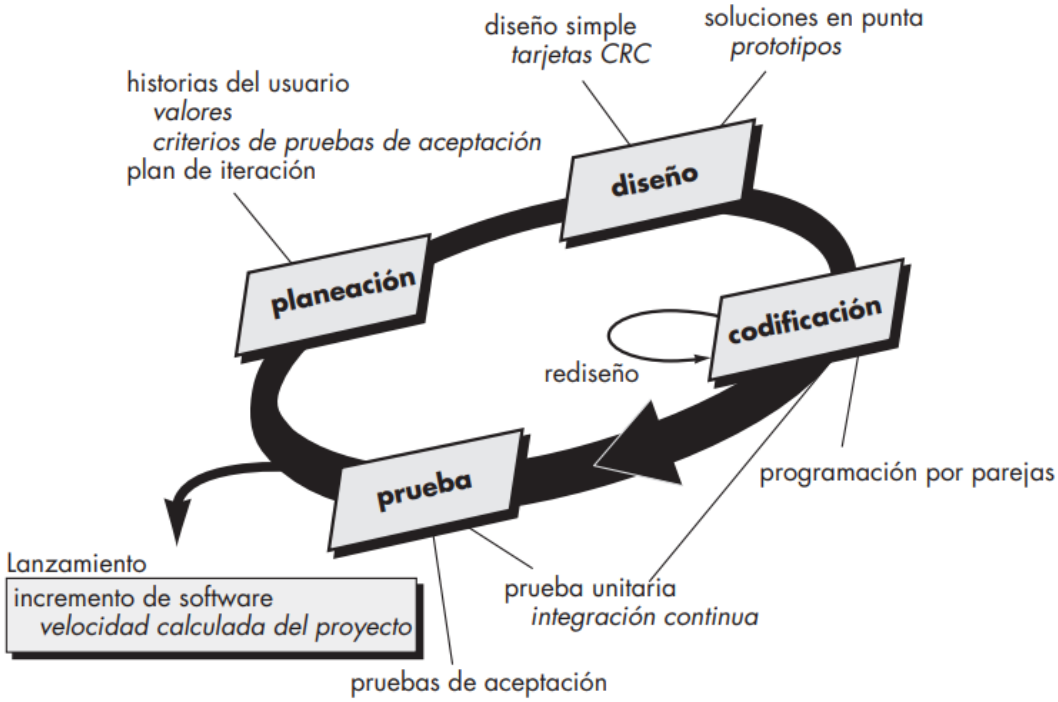

Figura 4: Fases de la Programación Extrema (Pressman, 2010).

PLANEACIÓN

Según (Solis, 2003), la planificación está planteada como un permanente diálogo entre la parte empresarial (lo deseable) y la técnica (lo posible). Esta actividad comienza recabando los requerimientos para que el equipo de trabajo entiendan el contexto del negocio para el software.

Basándose en (Pressman, 2010), la planeación permite escuchar y recabar *historias de usuario* que describen la salida necesaria, características y funcionalidad del software que se va a elaborar. Cada historia es descrita por el cliente. Es importante mencionar que en cualquier momento es posible escribir nuevas historias.

Además, el cliente (o persona del negocio) debe determinar lo siguiente:

- **Ámbito:** ¿Qué es lo que el software debe de resolver para que genere valor?

- **Prioridad:** ¿Qué debe ser hecho en primer lugar?
- **Composición de Versiones:** ¿Cuánto es necesario hacer para saber si el negocio va mejor con el software que sin él?
- **Estimaciones:** ¿Cuánto tiempo lleva iplementar una característica?
- **Procesos:** ¿Cómo se organiza el trabajo y el equipo?

A medida que avanza el trabajo, el cliente puede agregar historias, cambiar el valor de una ya existente, descomponerlas o eliminarlas (Pressman, 2010).

DISEÑO

El diseño en *XP* se basa en el principio de *'Mantenlo Sencillo'* que se basa en las siguientes reglas (Solis, 2003):

1. Funciona con todas las pruebas.
2. No tiene lógica duplicada.
3. Manifiesta cada intención importante para los programadores.
4. Tiene el menor número de clases y métodos.

Según (Pressman, 2010), un concepto central en *XP* es que el diseño ocurre tanto antes como después de que comienza la codificación. Rediseñar significa que el diseño se hace de manera continua conforme se construye el sistema.

CODIFICACIÓN

La codificación en *XP* se centra en historias validadas por pruebas unitarias, siendo fiel al diseño fundamental sin agregar nada extraño. El planteamiento de codificación más relevante es intentar implementar funcionalidades de la manera más simple posible. Después de implementar nuevas características, es necesario preguntarse cómo hacer el programa más simple sin perder funcionalidad (Solis, 2003).

PRUEBAS

Según (Pressman, 2010), las pruebas que se crean deben implementarse con el uso de una estructura que permita la automatización. No debe existir ninguna característica en el programa que no haya sido probada, los programadores escriben pruebas para verificar el correcto funcionamiento del programa, los clientes realizan pruebas funcionales. El resultado deriva a un programa más seguro que conforme pasa el tiempo es capaz de aceptar nuevos cambios.

La metodología *XP* parece arriesgada y voráz porque apuesta por velocidad, simpleza y cambios continuos, sin emabargo, en entornos reales ese riesgo se disuelve con prácticas técnicas estrictas que protegen el desarrollo.

Falta una última metodología.

Falta Marco Tecnológico.

Falta estado del arte.

Falta introducción al desarrollo.

Capítulo III

DESARROLLO DEL PROYECTO

3.1. ALGORITMO DE PREFILTRADO

El Sistema de Recomendación Musical tiene la característica particular de ser un sistema sensible al contexto. Aplicar un algoritmo de recomendación basado en las características cuantitativas de las canciones podría no ser suficiente para generar recomendaciones acertadas. Además, el uso de *datasets* masivos podría resultar en una desventaja más que en una ventaja, ya que en este problema en particular, una cantidad masiva de canciones podría resultar en un conjunto altamente ruidoso y poco relevante para el usuario final. Un ejemplo claro de este problema se puede ilustrar con el siguiente escenario:

Ejemplo 3.1.1. El usuario selecciona la canción *Let It Be - The Beatles* e indica al sistema que quiere escuchar canciones similares durante 20 minutos. El sistema, utilizando un *Dataset* masivo de canciones, aplica el algoritmo *ANN* y obtiene una playlist de canciones muy parecidas a la canción seleccionada por el usuario. Sin embargo, debido a la limitación de *ANN* de considerar únicamente las características cuantitativas, el sistema genera una playlist que incluye canciones muy similares en términos de características técnicas, pero muchas de ellas están en chino, japonés, o son de géneros musicales que el usuario no disfruta.

Este ejemplo ilustra cómo un enfoque basado únicamente en características cuantitativas puede resultar en recomendaciones que para el usuario podrían ser *carentes de sentido*. Para mitigar este problema, se propone la creación de un algoritmo de prefiltrado que tome en cuenta las características cualitativas de las canciones, además de su contexto inherente de tiempo, cultura y género musical.

Para lograr este objetivo, se propone agregar al sistema un servicio de recomendación que divide sus funciones en dos capas: una capa de prefiltrado y una capa de recomendación basada en *ANN*. La capa de prefiltrado se encarga de aprovechar la información disponible en la *Spotify Web API* para generar un conjunto de canciones al que se le denominará como *Muestra Significativa*.

La *Muestra Significativa* debe cumplir con ciertos criterios para que los resultados finales obtenidos justifiquen su uso. Estos criterios están basados en las características deseables de los sistemas de recomendación que favorecen la satisfacción del usuario final. Los criterios son los siguientes:

- **Tamaño Controlado:** Una muestra significativa debe tener un tamaño controlado para evitar tener un conjunto de canciones excesivamente pequeño que impida al algoritmo *ANN* encontrar canciones similares, o un conjunto excesivamente grande que reintroduzca el problema del ruido y la irrelevancia.
- **Relevancia Contextual:** La muestra debe estar alineada con el contexto de la canción original seleccionada por el usuario.
- **Diversidad:** No basta con que las canciones sean similares a la original, además la muestra debe incluir diversidad de géneros y artistas para enriquecer la experiencia del usuario.
- **Serendipia:** La diversidad en sí misma favorece la serendipia en las recomendaciones, sin embargo, una característica adicional que la muestra debe tomar en cuenta es la popularidad de las canciones. Así como es importante recomendar canciones populares, también es importante incluir canciones menos conocidas que podrían sorprender gratamente al usuario.

Para asegurar que la muestra significativa cumple con estos criterios, el algoritmo de prefiltrado necesita de heurísticas bien definidas que guíen la búsqueda y selección de canciones a favor de la calidad sobre la cantidad.

En la actualidad existen algoritmos que se podrían utilizar como base para el desarrollo de este prefiltrado, como el algoritmo *GBFS* que introduce la idea de generación dinámica de un arbol de búsqueda basado en la similitud de características. Esto es importante debido a las limitaciones de la *Spotify Web API* que no facilita a los desarrolladores un *Dataset* completo de canciones. Además, otro de los algoritmos altamente relacionados al concepto de búsqueda de elementos similares es el algoritmo *A**, que introduce la idea de una función heurística que guía la búsqueda hacia los nodos más prometedores.

Ambos algoritmos pueden servir como base teórica para fundamentar la creación de un algoritmo dinámico de generación de *Muestra Significativa* que genere un conjunto de canciones partiendo de una canción raíz que se expande a mediante un conjunto de heurísticas que aseguren la calidad de las canciones seleccionadas.

Sin embargo, si no se tiene control de la dirección en la que se expande el árbol de búsqueda, es posible que se generen conjuntos de canciones que no cumplan

con los criterios establecidos. Por esta razón, se propone además usar conceptos de las *Búsquedas Tabú* para generar diversas ramas de expansión que fomenten la diversidad y la serendipia en la muestra significativa.

Además de todas las características mencionadas, una propiedad importante en el Sistema de Recomendación Musical debe ser la capacidad de brindar diferentes respuestas ante un mismo estímulo. Esto se traduce en que el usuario pueda obtener diferentes listas de reproducción al seleccionar la misma canción en diferentes ocasiones. Esta propiedad es importante para mantener el interés del usuario y evitar la monotonía en las recomendaciones. Para lograr esto, el algoritmo de prefiltrado debe agregar una capa controlada de aleatoriedad que favorezca la generación de diferentes muestras en cada ejecución.

El algoritmo propuesto añade valor en el dominio musical que los algoritmos *GBFS*, *A** y *Búsqueda Tabú* no consideran de manera explícita. La combinación de estos enfoques heurísticos permite crear un algoritmo de prefiltrado robusto, novedoso y adaptado a las necesidades específicas del Sistema de Recomendación Musical.

DESCRIPCIÓN DEL ALGORITMO DE PREFILTRADO

En esta sección se describirá de manera específica el funcionamiento del algoritmo de prefiltrado propuesto. El algoritmo se puede describir a través de las siguientes características:

ENTRADAS

El algoritmo debe recibir una *canción raíz* que servirá como punto de partida para la generación de la *Muestra Significativa*. Esta canción es seleccionada por el usuario en la aplicación web y es recibida por la capa central del sistema. Esta capa debe realizar una petición a la *Spotify Web API* para obtener los metadatos de la canción, que incluyen características tanto cuantitativas, cualitativas y técnicas. Sin embargo, para simplificar el desarrollo del algoritmo, la capa central del sistema se encargará de limpiar los datos para obtener la siguiente estructura:

En la Table III.1 se muestra la estructura curada de una canción obtenida de la *Spotify Web API*. Esta estructura busca minimizar la cantidad de información que se transportará entre capas para reducir tiempos de respuesta y simplificar el manejo de los datos. Esto se puede visualizar en el atributo *album*, que en la *Spotify Web API* es un objeto complejo que contiene mucha información que no es relevante para el algoritmo de prefiltrado. Esto se justificará en las etapas posteriores.

Atributo	Tipo de Dato	Ejemplo
ID-Track	*string*	2up3OPMp9Tb4dAKM2erWXQ
name	*string*	Let It Be
artist	*Artist*	{The Beatles, ...}
album	*string*	Let It Be - Remastered
popularity	*int* [0-100]	90

Cuadro III.1: Estructura curada de una canción obtenida de la *Spotify Web API*.

Sin embargo, una estructura que es necesario mantener es el atributo *artist* debido a características que serán mencionadas más adelante. La definición curada de un artista se puede observar en la Table III.2.

Atributo	Tipo de Dato	Ejemplo
ID-Artist	*string*	2up3OPMp9Tb4dAKM2erWXQ
name	*string*	The Beatles
genres	*string[]*	[Rock, Brit Pop, ...]
popularity	*int* [0-100]	90
related-artists	*Artist[0-2]*	[The Rolling Stones, ...]

Cuadro III.2: Estructura curada de un artista obtenida de la *Spotify Web API*.

VARIABLES Y ESTRUCTURAS AUXILIARES

Para lograr que el algoritmo funcione correctamente es necesario utilizar variables y estructuras de datos que controlen el flujo y expansión de la muestra significativa. En la Table III.3 se mencionan las variables más importantes, su función y la importancia que tienen en el flujo del algoritmo.

MUESTRA

La variable *sample* representa un conjunto de tracks con el mismo formato mostrado en la Table III.1. Esta estructura de datos representa la *muestra significativa* final que será procesada posteriormente mediante el algoritmo de recomendación elegido, por lo tanto, es importante analizar su construcción de manera detallada.

Variable	Tipo	Descripción
sample	Track[]	Arreglo dinámico que contendrá la *Muestra Significativa* de canciones.
size	int	Tamaño ideal de la muestra, calculado dinámicamente a partir de la duración de escucha deseada.
queue_API	queue	Cola de peticiones a la *Spotify Web API* para expandir el grafo de búsqueda.
tabu	RuleSet	Conjunto de restricciones que controlan la diversidad de la muestra (por artista, género, etc.).
score	float	Valor heurístico asignado a cada canción para medir su relevancia.

Cuadro III.3: Variables y estructuras auxiliares del algoritmo de prefiltrado.

El tamaño de este arreglo será determinado mediante un valor máximo denominado como *size*. Esta variable debe ser determinada mediante un cálculo arbitrario que determine el tamaño ideal de la muestra, para lograr esto se necesitan saber dos datos importantes: La duración que el usuario ha elegido para el bloque de escucha actual y el promedio de duración estimado de una canción.

Con esta información podremos obtener un estimado de cuántas canciones debe tener ésta fracción de la lista de reproducción mediante la siguiente fórmula:

$$total = \lceil \frac{duracion}{promedio} \rceil^{1} \tag{III.1}$$

Este *total* representa la duración ideal de cada bloque de reproducción resultante de las recomendaciones (aunque la duración final podría variar por unos pocos minutos). Sin embargo, una muestra de este tamaño no es suficiente, por esta razón se ha decidido agregar una variable λ que sirve como múltiplo que aumenta el tamaño de la muestra a través de una proporción arbitraria. Por lo tanto, la fórmula de *size* se denomina de la siguiente manera:

$$size = total \cdot \lambda \tag{III.2}$$

[1] La necesidad de redondear el valor total se debe a que los algoritmos planteados trabajan mediante un número de canciones exactas.

El valor de λ será elegido arbitrariamente al inicio de la experimentación y se ajustará según los resultados obtenidos. Si bien no existe una proporción ideal, se puede ajustar para que los resultados presentados al usuario sean aceptables en la mayoría de casos.

COLA DE PETICIONES

Una de las principales dificultades que se presentan en el desarrollo de este algoritmo es la cantidad de peticiones que se van a hacer a la *Spotify Web API*, lo que podría ocacionar errores de sobrecarga o latencia. Para solventar estos obstáculos es necesario explicar la filosofía que el algoritmo usará para obtener la información.

La filosofía de obtención de datos se basa en una idea “greedy” o voráz de obtener la máxima cantidad de información en la menor cantidad de peticiones posible. Para lograr esto, la capa central del sistema debe tener un control robusto de las peticiones que realizará a la API de Spotify. Con el objetivo de lograr este control se plantea el uso de una *Cola de Peticiones* que será formada por el algoritmo de prefiltrado. Esta estructura guardará dinámicamente peticiones en una cola que almacenará instancias de la clase *Request* definida en la Table III.4.

Atributo	Tipo de Dato	Ejemplo
ID-Request	*string*	2up3OPMp9Tb4dAKM2erWXQ
type	*Enum[Artist,Album,Playlist...]*	Artist
id-object	*string*	3an1OJMwx7Ua9asmk91QIP
params	*string[]*	{maxsize: 20}

Cuadro III.4: Estructura de una Petición en el algoritmo de Prefiltrado.

La forma en que la *Cola de Peticiones* se llenará es a través de las canciones procesadas en tiempo real. Por ejemplo, en la primera iteración del algoritmo la única canción en la cola de peticiones será la canción raíz. Esas peticiones realizadas retornarán un conjunto de canciones nuevas que serán procesadas individualmente almacenando nuevas peticiones en la cola. De esta manera se logra que la central haga peticiones agrupadas aprovechando los *endpoints* brindados por la *Spotify Web API* y obteniendo grandes cantidades de información rapidamente.

ESTRUCTURA TABÚ

Bibliografía

Aggarwal, C. C. (2016). An Introduction to Recommender Systems. En *Recommender Systems: The Textbook* (pp. 1-28). Springer International Publishing. https://doi.org/10.1007/978-3-319-29659-3_1

Alasadi, S. A., & Bhaya, W. S. (2017). Review of data preprocessing techniques in data mining. *Journal of Engineering and Applied Sciences*, *12*(16), 4102-4107.

Almufti, S. M., Shaban, A. A., Ali, Z. A., Ali, R. I., & Fuente, J. D. (2023). Overview of metaheuristic algorithms. *Polaris Global Journal of Scholarly Research and Trends*, *2*(2), 10-32.

Aumüller, M., & Ceccarello, M. (2023). Recent Approaches and Trends in Approximate Nearest Neighbor Search. {*IEEE*} *Data Engineering Bulletin*.

Burke, R. (2007). Hybrid Web Recommender Systems. En P. Brusilovsky, A. Kobsa & W. Nejdl (Eds.), *The Adaptive Web: Methods and Strategies of Web Personalization* (pp. 377-408). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-72079-9_12

Chapman, A., Simperl, E., Koesten, L., Konstantinidis, G., Ibáñez, L.-D., Kacprzak, E., & Groth, P. (2019). Dataset search: a survey. *The VLDB Journal*, *29*(1), 251-272. https://doi.org/10.1007/s00778-019-00564-x

Cvijović, D., & Klinowski, J. (2002). Taboo Search: An Approach to the Multiple-Minima Problem for Continuous Functions. En P. M. Pardalos & H. E. Romeijn (Eds.), *Handbook of Global Optimization: Volume 2* (pp. 387-406). Springer US. https://doi.org/10.1007/978-1-4757-5362-2_11

Gendreau, M. (2003). An Introduction to Tabu Search. En F. Glover & G. A. Kochenberger (Eds.), *Handbook of Metaheuristics* (pp. 37-54). Springer US. https://doi.org/10.1007/0-306-48056-5_2

Greenberg, J. (2005). Understanding Metadata and Metadata Schemes. *Cataloging & Classification Quarterly*, *40*(3-4), 17-36. https://doi.org/10.1300/J104v40n03_02

Heusner, M., Keller, T., & Helmert, M. (2017). Understanding the search behaviour of greedy best-first search. *Proceedings of the International Symposium on Combinatorial Search*, *8*(1), 47-55.

Hodson, T. O. (2022). Root-mean-square error (RMSE) or mean absolute error (MAE): when to use them or not. *Geoscientific Model Development*, *15*(14), 5481-5487. https://doi.org/10.5194/gmd-15-5481-2022

Isinkaye, F., Folajimi, Y., & Ojokoh, B. (2015). Recommendation systems: Principles, methods and evaluation. *Egyptian Informatics Journal*, *16*(3), 261-273. https://doi.org/https://doi.org/10.1016/j.eij.2015.06.005

Javaid, A. (2013). Understanding Dijkstra's algorithm. *Available at SSRN 2340905*.

Khuller, S., Raghavachari, B., & Young, N. E. (2018). Greedy methods. En *Handbook of approximation algorithms and metaheuristics* (pp. 55-69). Chapman; Hall/CRC.

Kotkov, D., Veijalainen, J., & Wang, S. (2020). How does serendipity affect diversity in recommender systems? A serendipity-oriented greedy algorithm. *Computing*, *102*(2), 393-411. https://doi.org/10.1007/s00607-018-0687-5

Kumar, R. (2024). El Algoritmo A*: Guía Completa.

Liu, T., Moore, A., Yang, K., & Gray, A. (2004). An Investigation of Practical Approximate Nearest Neighbor Algorithms. En L. Saul, Y. Weiss & L. Bottou (Eds.), *Advances in Neural Information Processing Systems* (Vol. 17). MIT Press. https://proceedings.neurips.cc/paper_files/paper/2004/file/1102a326d5f7c9e04fc3c89d0ede88c9-Paper.pdf

Mateos, P., & Bellong, A. (2024). A systematic literature review of recent advances on context-aware recommender systems. *Artificial Intelligence Review*, *58*(1), 20.

Nicholson, W. K. (2024, enero). 10.1: Inner Products and Norms. https://math.libretexts.org/Bookshelves/Linear_Algebra/Linear_Algebra_with_Applications_(Nicholson)/10%3A_Inner_Product_Spaces/10.01%3A_Inner_Products_and_Norms

Pearl, J. (1984). *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley Longman Publishing Co., Inc.

Perez García, J. I. (2021). MATEMÁTICA DISCRETA Conjuntos discretos. Conjuntos numerables. Lógica proposicional y lógica de predicados, la inducción matemática, Álgebra relacional, Álgebra de Boole, Teoría de grafos, grafos orientados; Cálculo combinatorio: Arreglos, permutaciones y combinaciones, Aplicaciones. https://repositorio.une.edu.pe/handle/20.500.14039/8990

Poole, D. (2007). *Álgebra lineal: una introducción moderna*. Cengage Learning Latin America. https://books.google.com.mx/books?id=fAPctkEifCMC

Pressman, R. S. (2010). *Ingeniería del Software: Un enfoque práctico* (7.ª ed.) [Traducido al español]. McGraw-Hill Interamericana de España S.L. https://books.google.com.cu/books?id=deuwcQAACAAJ

Proy de Santiago, P., et al. (2024). Algoritmos heurísticos.

Ricci, F., Rokach, L., Shapira, B., & Kantor, P. B. (2010). *Recommender Systems Handbook* (1st). Springer-Verlag. https://doi.org/https://doi.org/10.1007/978-0-387-85820-3

Russell, S., & Norvig, P. (2009). *Artificial Intelligence: A Modern Approach* (3rd). Prentice Hall Press.

Sachdeva, S. (2016). Scrum methodology. *Int. J. Eng. Comput. Sci*, *5*(16792), 16792-16800.

Schröer, C., Kruse, F., & Gómez, J. M. (2021). A systematic literature review on applying CRISP-DM process model. *Procedia computer science*, *181*, 526-534.

Solis, M. C. (2003). Una explicación de la programación extrema (XP). *V Encuentro usuarios xBase*, *1*.

Team, E. P. (2025, septiembre). Understanding the approximate nearest neighbor (ANN) algorithm. https://www.elastic.co/blog/understanding-ann#when-to-use-approximate-nearest-neighbor-search

Wirth, R., & Hipp, J. (2000). CRISP-DM: Towards a standard process model for data mining. *Proceedings of the 4th international conference on the practical applications of knowledge discovery and data mining*, *1*, 29-39.