



INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

Trabajo Terminal

**Ambiente de análisis de datos mediante Big Data
(Luminus)**

2018-A104

Que para cumplir con la opción de Titulación Curricular en la carrera de:
“Ingeniería en Sistemas Computacionales”

PRESENTAN:

**MONTEÓN VALDÉS RAÚL KEVIN
TOVAR BARBA MAYRA PATRICIA**

DIRECTORES:



BOTELLO CASTILLO ALEJANDRO

VÉLEZ SALDAÑA ULISES

México, D.F. a 20 de Mayo de 2019

Índice general

0.1. Agradecimientos	7
0.1.1. Raúl Kevin Monteón Valdés	7
0.1.2. Mayra Patricia Tovar Barba	7
1. Capítulo 1	9
1.1. Introducción	9
1.2. Planteamiento del problema	9
1.2.1. Problemática	9
1.3. Propuesta de solución	10
1.4. Estado del arte	11
1.5. Justificación	12
1.6. Objetivos	13
1.6.1. General	13
1.6.2. Específicos	13
1.7. Arquitectura de la red distribuida	13
1.8. Alcances y Limitaciones	14
1.8.1. Limitaciones	14
1.8.2. Alcances	15
1.8.3. Prototipo 1	15
1.8.4. Prototipo 3: Algoritmos de minería de datos	16
1.8.5. Prototipo 4: Interfaz de Programación de Aplicaciones	16
1.8.6. Prototipo Adicional: Instalador	16
2. Marco teórico	17
2.1. Big Data	17
2.1.1. Las 3 Vs del Big Data	17
2.1.2. Casos de uso del Big Data	18
2.2. Minería de datos	18
2.3. Árboles de decisión	18
2.4. ID3 (Iterative Dichotomiser 3)	19
2.4.1. Entropía	20
2.4.2. Ejemplo de ejecución del algoritmo ID3	21
2.5. C4.5	25
2.5.1. Ejemplo de ejecución del algoritmo C4.5	26
2.6. Algoritmo KNN (K-Nearest Neighbors)	31
2.6.1. Ejemplo de ejecución del algoritmo KNN	31
2.7. Algoritmo K-Means	33
2.7.1. Ejemplo de ejecución del algoritmo K-Means	33

2.8. Redes Bayesianas	34
2.9. Ambiente de análisis de datos	35
2.10. Clúster	35
2.11. MapReduce	35
2.12. Hadoop	35
2.12.1. Características principales de Hadoop	35
2.12.2. Arquitectura de Hadoop	37
2.12.3. Arquitectura de un clúster Hadoop	38
2.12.4. Distribuciones Comerciales de Hadoop	38
2.13. Apache Spark	38
2.13.1. Características principales de Spark	39
2.13.2. Ventajas de usar Apache Spark	39
2.13.3. Componentes de Apache Spark	40
2.14. Interfaz Web	40
2.15. Interfaz de Programación de Aplicaciones	40
2.16. Framework	41
2.17. Librería	41
3. Evaluación y definición de requerimientos de la red distribuida	43
3.1. Descripción del prototipo	43
3.2. Análisis	43
3.2.1. Análisis de la red distribuida	43
3.2.2. Análisis de los datos	44
3.3. Diseño	45
3.3.1. Diseño de la red distribuida	45
3.3.2. Diseño de propuesta de agrupación de acuerdo a los nodos definidos en la red distribuida	46
3.4. Desarrollo	48
3.5. Pruebas	49
4. Integración de requerimientos de la red distribuida	53
4.1. Descripción del prototipo	53
4.2. Análisis	53
4.2.1. Análisis de la adaptación de los datos a la red distribuida	53
4.3. Diseño	53
4.3.1. Diseño de la red distribuida con nodos de datos	53
4.3.2. Diseño de pruebas de obtención de información	54
4.4. Desarrollo	56
4.4.1. Algoritmo de prueba	57
4.5. Pruebas	58
5. Algoritmos de Minería de Datos	65
5.1. Descripción del prototipo	65
5.2. Análisis	65
5.2.1. Análisis del flujo de datos en las pantallas del sistema	65
5.2.2. Análisis de los algoritmos de minería de datos	65
5.3. Diseño	66
5.3.1. Algoritmo KNN con el uso de MapReduce	66
5.3.2. Algoritmo ID3	72
5.4. Desarrollo	79
5.4.1. Algoritmo KNN	79
5.4.2. Algoritmo ID3	80
5.5. Pruebas	81
5.5.1. Algoritmo KNN	81
5.5.2. Algoritmo ID3	89

6. Interface de programación de aplicaciones de Luminus	101
6.1. Descripción del prototipo	101
6.2. Análisis	101
6.2.1. Casos de uso	101
6.3. CU1 Leer archivo	103
6.3.1. Resumen	103
6.3.2. Descripción	103
6.3.3. Trayectorias del Caso de Uso	103
6.4. CU2 Crear directorio	105
6.4.1. Resumen	105
6.4.2. Descripción	105
6.4.3. Trayectorias del Caso de Uso	105
6.4.4. Puntos de extensión	106
6.5. CU3 Subir un archivo	107
6.5.1. Resumen	107
6.5.2. Descripción	107
6.5.3. Trayectorias del Caso de Uso	107
6.6. CU4 Borrar un archivo o directorio	109
6.6.1. Resumen	109
6.6.2. Descripción	109
6.6.3. Trayectorias del Caso de Uso	109
6.7. CU5 Confirmar existencia de archivo o directorio	110
6.7.1. Resumen	110
6.7.2. Descripción	110
6.7.3. Trayectorias del Caso de Uso	110
6.8. CU6 Obtener Identificador de HDFS	112
6.8.1. Resumen	112
6.8.2. Descripción	112
6.8.3. Trayectorias del Caso de Uso	112
6.9. CU7 Obtener Dirección IP Local	113
6.9.1. Resumen	113
6.9.2. Descripción	113
6.9.3. Trayectorias del Caso de Uso	113
6.10. CU8 Crear configuración	114
6.10.1. Resumen	114
6.10.2. Descripción	114
6.10.3. Trayectorias del Caso de Uso	114
6.11. CU9 Ejecutar Comando	115
6.11.1. Resumen	115
6.11.2. Descripción	115
6.11.3. Trayectorias del Caso de Uso	115
6.12. CU10 Correr KNN	116
6.12.1. Resumen	116
6.12.2. Descripción	116
6.12.3. Trayectorias del Caso de Uso	116
6.13. CU11 Correr ID3	118
6.13.1. Resumen	118
6.13.2. Descripción	118
6.13.3. Trayectorias del Caso de Uso	118
6.14. CU12 Subir Configuración	120
6.14.1. Resumen	120
6.14.2. Descripción	120
6.14.3. Trayectorias del Caso de Uso	120
6.15. CU13 Verificar compatibilidad de datos con algoritmo seleccionado	122

6.15.1. Resumen	122
6.15.2. Descripción	122
6.15.3. Trayectorias del Caso de Uso	122
6.16. Diseño	123
6.16.1. Diagrama de clases	123
6.17. Desarrollo	123
6.17.1. LuminusConfiguration	123
6.17.2. LuminusConfigurationKNN	124
6.17.3. LuminusConfigurationID3	126
6.17.4. HDFSAccess	128
6.17.5. LuminusException	130
6.17.6. MSG1 No existe el archivo o directorio	130
6.17.7. MSG2 El archivo ya existe	130
6.17.8. MSG3 El directorio ya existe	130
6.17.9. MSG4 Selección de algoritmo inválida	130
6.17.10.MSG5 Extensión del archivo properties inválida.	131
6.17.11.MSG6 Ruta de archivo properties inválida	131
6.17.12.TypeManager	131
6.17.13.Pruebas	132
7. Instalador de Luminus	137
7.1. Descripción del prototipo	137
7.2. Análisis	137
7.3. Diseño	138
7.4. Desarrollo	143
7.5. Pruebas	145
8. Conclusiones	157
8.1. Conclusiones	157
9. Trabajo a futuro	159
9.1. Trabajo a futuro	159
10. Anexos	165
10.1. Anexo A: Código JAVA Map-Reduce	165
10.2. Anexo B: Código del instalador	167
10.3. Anexo C: Una manera de presentar a luminus de forma diferente: Sitio web	177

0.1. Agradecimientos

0.1.1. Raúl Kevin Monteón Valdés

En las primeras etapas de nuestra vida, generalmente no nos damos cuenta de lo importante que son las relaciones humanas; de todo lo bueno, y también de lo malo que pueden traernos. Sin embargo, al menos en mi caso, creo que desde hace relativamente poco tiempo, comienzo a ser cada vez más consciente de ello. Y siendo un poco optimista, me centraré en este texto en lo positivo que han traído a mi vida ciertas relaciones humanas. Particularmente, me gustaría resaltar algunas de ellas, ya que si no fuera porque esas valiosas personas estuvieron ahí, formando parte de mi entorno y de mi vida, en este momento muy seguramente no estaría alcanzando este logro: entregar un documento de Trabajo Terminal II para titularme como Ingeniero en Sistemas Computacionales de la Escuela Superior de Cómputo del Instituto Politécnico Nacional. Bastante rimbombante a mi parecer.

Este logro no es solamente mío. Es tuyo también, papá. Siempre ahí por las mañanas, siempre tan recto, tan metódico, tan noble, tan alegre, tan ejemplar. Tan perfecto que no me imagino la vida sin ti y sin tu incondicional apoyo. Me has apoyado en absolutamente todo, y no tengo palabras para agraecerte por estar ahí conmigo.

El logro también es tuyo, mamá. Es difícil siquiera tratar de recordar todo lo que has hecho por mí. Porque son demasiadas cosas. Creo que no hay nada que no hayas hecho por mí. Y lo mejor y más reconfortante para mí, es saber que eres incondicional. Siempre me has impulsado. Me siento muy agradecido contigo por todas esas tardes, mañanas y noches que me dedicaste desde que era un infante, con la finalidad de que explotara mi potencial. Tú y sólo tú, eres la mejor madre del mundo, con tus cualidades y defectos. Con tu inteligencia, perseverancia, firmeza y ternura. Los amo por igual a ti y a mi papá. Me saqué la lotería con ustedes desde que nací. Y eso me convierte en la persona más afortunada del mundo. Sus enseñanzas quedaron en mí para siempre, y sé que seguiré aprendiendo mucho más de los dos. En verdad, gracias.

Por último, quiero agradecer a mi compañera de trabajo terminal: Mayra. ¿Qué te puedo decir, May? De verdad que no pude haber tenido mejor fortuna. Me conoces bien, y te conozco bien. De alguna forma siempre hemos trabajado muy armónicamente. Sin ti este trabajo no tendría pies ni cabeza. Eres una de las personas más inteligentes que conozco, una de las más perseverantes y astutas. Me alegro tanto de haber vivido esta experiencia a tu lado, que ya cada que recuerdo algo sobre el trabajo terminal, al menos en esta recta final, lo hago con una sonrisa. Aún con todas las dificultades y retos que se nos presentaron a lo largo del camino. Siempre tuviste muchísima paciencia conmigo, siempre fuiste demasiado entregada. Siempre diste lo mejor de ti. Ya eres una persona exitosa, y sé que lo serás aún más, porque te lo mereces y tienes todo lo necesario para serlo. Gracias por todo, May.

0.1.2. Mayra Patricia Tovar Barba

Quiero agradecer a mis padres, ya que ellos a pesar de no comprender la complejidad del proyecto estuvieron conmigo para apoyarme en lo que estuvo a su alcance con recursos económicos, permisos, tiempo para realizar el proyecto e incluso frenar algunos planes familiares para darme la oportunidad de terminar mi proyecto incluso limitándome a ellos mismos.

A mis amigos, que siempre han estado conmigo apoyarme tanto personal como profesionalmente no solamente estuvieron a mi lado en el periodo de elaboración del proyecto sino durante toda la carrera y en cada bache que atravesamos y para cada complejidad a la que hice tuve que hacer frente siempre han estado a mi lado. Incluso, hemos perdido la forma de contabilizar todo ese apoyo que nos hemos dado entre nosotros ¿Recuerdan de cuantos jabones se trata? Yo ya perdí la cuenta, creo que es un camión para cada uno de nosotros.

Por otra parte, también me gustaría dar las gracias a Rolando. Muchas veces cuando hasta yo misma perdía el valor del proyecto y el valor de mi propio trabajo y esfuerzo el siempre estuvo ahí para recordarme la importancia de lo que estaba haciendo, el peso que tenía, y la complejidad de mi trabajo.

Cada vez que me topo con dificultades que no sabía como resolver o que empezaban a complicarse demasiado siempre estuviste ahí para darme una mano con el fin de que pudiera lograrlo, algunas veces ayudándome a hallar la solución y en otras tantas solo escuchándome para que pudiera encontrarla por mí misma.

Gracias a Ulises Vélez, ya que él nos mantuvo interesados en el proyecto y trabajando, no nos dejó retrasarnos en nuestros alcances y nos dijo cada vez las palabras que necesitábamos escuchar para seguir adelante, representando con

ello un apoyo para el equipo de trabajo.

No permitió existiera siquiera la posibilidad de quedar fuera del calendario que nosotros mismos establecimos ya que todo el tiempo estuvo programando actividades y revisando que las actividades quedarán completadas en los tiempos estipulados.

Se tuvo mucha visión de su parte para dirigir, ampliar y enriquecer nuestra visión del proyecto. Gracias a Alejandro Botello, por toda su visión y conocimiento en cuanto a base de datos y Big Data, por las ideas que aportó al proyecto y por cada vez que nos orientó para poder alcanzar nuestros objetivos.

Cada una de las veces que creyó en nosotros y que podríamos lograrlo incluso cuando en ocasiones no le presentamos nuestros avances hasta ese momento y en su lugar solo hablábamos de ellos, él sabía que los teníamos y no era necesario tener la evidencia de ello. Cada vez que nos encontramos en un punto donde ya no sabíamos como continuar el nos dio claridad acerca de los siguientes pasos a seguir y de cómo solventar la problemática en la que nos encontrábamos. A mis sinodales, ya que con las ideas y visión que cada uno de ellos tuvo del proyecto se logró enriquecer el mismo. Los comentarios que nos hicieron fueron presentados a nuestros directores con el fin de determinar su viabilidad y realizar los ajustes pertinentes.

Gracias a mis jefes de mi trabajo Antonio y Argenis, ellos fueron comprensivos conmigo en cada una de las etapas involucradas en el proyecto y me permitieron completarlo satisfactoriamente. Me dieron los espacios cada vez que los necesite para acudir a revisiones o entregas, Fueron flexibles en cuanto a mis horarios laborales para permitirme cumplir con estas metas.

Una persona a la que quiero agradecer de manera especial es a mi compañero de trabajo Kevin. Contigo no tendría palabras para darte las gracias por todo lo que has hecho por este proyecto, has pasado por situaciones que podrían detenerte y has seguido adelante siempre firme a pesar de todo.

Yo conozco demasiado de ti, y se lo difícil que puede resultar para ti mantenerte fuerte tan solo para ti, y en incontables ocasiones tuviste la fuerza para soportarnos a los 2, para darnos tranquilidad y fuerzas a ambos. Ni siquiera puedo empezar a describir cómo es que lo hiciste.

Entregaste tu esfuerzo, pasión, ganas, dedicación, que te puedo decir, lo has entregado todo. Estoy enterada de cada una de esas veces en las que te desvelaste tratando de completar alguna funcionalidad, perdiendo tus valiosas horas de sueño porque sin importar lo difícil que fuera tenía que quedar listo.

Yo no puedo terminar de agradecerte por la ocasión en la que te presente el proyecto y dijiste que sí, lo harías conmigo aun cuando ninguno de los 2 sabíamos en lo que nos estábamos metiendo.

Y es que, hemos compartido tanto, y hemos estado de la mano uno con el otro sin importar si lo que tocaba en cada momento era bueno o malo y lo soportamos y vivimos juntos solo tu serías capaz de comprenderlo y relatarlo de principio a fin. Solo contigo puedo compartir historias como: Cloudera, Docker, Código de los algoritmos en la API, Paralelismo en el ID3, Versión de Java, etc jamás las olvidaría.

Tantas personas que terminan mal después de un proyecto de esta magnitud, y como me dijiste recientemente jamás habíamos dedicado tanto trabajo a un solo proyecto. Es un proyecto realmente grande y a pesar de que siento el peso y lo mucho que hemos invertido en el yo puedo decir que volvería a tomar proyectos así de grandes contigo, porque se de lo que somos capaces de lograr juntos.

Solo me resta decir: jamás lo habría podido lograrlo sin ti, Gracias por todo.

Finalmente, se que hubo más personas involucradas en la elaboración de este proyecto y que seguramente se me estará escapando alguna. Pero se que cada persona que convivio conmigo durante este tiempo me aporto de diferentes formas gracias por tanto a cada uno de ustedes.

CAPÍTULO 1

Capítulo 1

1.1. Introducción

En el siglo XXI, el Big Data ha sido un tema que ha dado mucho de qué hablar, su estudio ha ido en constante crecimiento y ha servido de gran manera a empresas dedicadas a las tecnologías de la información a generar conocimiento a partir de cantidades masivas de datos que no tienen una estructura muy bien definida. El propio término Big Data (Grandes Datos en español) lo hace entender, la cantidad de datos que se maneja es demasiado grande para ser siquiera almacenada o procesada de manera tradicional [1].

Big Data provee a quien lo usa conocimiento que de otra forma le habría sido imposible obtener, dicho conocimiento permite hacer análisis de los resultados para con ellos obtener ideas que conduzcan a mejores decisiones y movimientos de negocios estratégicos, por lo anterior, Big Data puede llegar a significar una herramienta de alto valor para las empresas que posean la cantidad de datos suficientes para aplicarlo.

1.2. Planteamiento del problema

En la actualidad, el valor de los datos es muy importante para las empresas; cuando se hace un análisis correcto de la información histórica almacenada, se pueden hacer estadísticas de comportamiento e inclusive predicciones de eventos que vendrán en el futuro, las cuales ayudan a la toma de decisiones [2]. Conforme los datos crecen en el tiempo, también la complejidad en el procesamiento de estos datos aumenta. Una de las tendencias de años recientes, llamada Big Data, utiliza un esquema de cómputo distribuido para aprovechar la infraestructura de cómputo disponible en la empresa, y resuelve de manera cooperativa los problemas de análisis de datos [7].

Si esta tendencia es aplicada de manera efectiva, los datos que se generan, pueden significar que las empresas se muevan mucho más rápidamente, sin problemas y de manera más eficiente, ya que son capaces de proporcionar respuestas a preguntas que de otra forma la empresa ni siquiera sabía de su existencia.

Con una cantidad tan grande de información, los datos pueden ser moldeados o probados de cualquier manera que la empresa considere adecuada. Al hacerlo, las organizaciones son capaces de identificar los problemas de una forma más comprensible [8].

1.2.1. Problemática

A pesar de todas las utilidades que ofrece hacer uso de Big Data a las empresas, también se tiene que considerar que su uso implica una gran inversión de recursos económicos, recursos humanos y tiempo.

Las curvas de aprendizaje de las tecnologías con que se hace uso de Big Data suelen ser pronunciadas, por lo que generalmente conlleva un tiempo amplio a quienes quieren aprender a usarlas. Además de que se requiere de una base de conocimiento en cuestión de bases de datos, sistemas operativos, programación, entre otras áreas.

En el ámbito económico, implementar un ambiente de Big Data suele tener un costo económico muy elevado, ya que normalmente se utilizan servidores dedicados. Tanto costo de los equipos mismos, como de su mantenimiento y el acondicionamiento del lugar donde se pondrán a funcionar, significa un costo elevado.

Por otro lado, considerando estas problemáticas, la implementación y puesta en marcha de un ambiente de Big Data también se puede complicar al momento incluir los datos que la empresa desea analizar. Las razones por las que esto comúnmente pasa, se explican a continuación:

- Datos de gran volumen.^[8]
- Los datos cambian rápidamente por lo que su tiempo de validez es muy corto.^[8]
- Debido a que los datos son muy volátiles para que se obtengan resultados valiosos al aplicar Big Data sobre ellos se requiere un poder de procesamiento alto.
- Es necesario discernir entre qué datos son importantes en el análisis y cuáles no lo son.^[8]

1.3. Propuesta de solución

En este trabajo terminal se propone desarrollar un ambiente de Big Data que pueda adaptarse a diferentes casos de estudio o dicho en otras palabras, grupos de datos diferentes donde se desea aplicar un determinado algoritmo, incluyendo un archivo de configuración que permita realizar las adaptaciones correspondientes para cada uno de estos casos de estudio.

El ambiente de Big Data incluirá la aplicación de los siguientes algoritmos de minería de datos:

- Un algoritmo de reglas de asociación
- Un algoritmo de árboles de decisión

Así como la generación de conocimiento y resultados a partir de la aplicación de estos algoritmos.

Haciendo un análisis de los datos correspondientes al caso de estudio en cuestión se determinará si estos pueden soportar y ser aplicados a el algoritmo de minería de datos que el usuario experto desee aplicar. En caso de que esto no sea posible se notificará de esto al usuario experto para este pueda seleccionar otro algoritmo.

Esta solución podrá implementarse en computadoras tradicionales, reduciendo así el costo económico de su implementación en gran medida, ya que se estaría eliminando la necesidad de comprar equipos especializados y ambientarlos.

Debido a que el ambiente de Big Data del que hablamos requiere hacer uso, de varias computadoras tradicionales y cada una de ellas requiere llevar a cabo un proceso de instalación y configuración para que este pueda funcionar. Se desarrollará un instalador que permita simplificar esta tarea realizando todo este proceso desde un solo nodo que será definido como nodo maestro.

La ejecución y puesta en marcha de los algoritmos de minería de datos se hará desde una Interfaz de Programación de Aplicaciones que podrá ser soportada e incorporada en sistemas que corran haciendo uso del lenguaje de programación JAVA.

1.4. Estado del arte

Actualmente en el mercado existen diferentes alternativas para los usuarios que busquen hacer uso de Big Data. Las alternativas encontradas, podrían dividirse en 2 tipos:

- Software que ya contiene algoritmos de minería de datos implementados.

SOFTWARE	CARACTERÍSTICAS	PRECIO EN EL MERCADO
Knime	<ul style="list-style-type: none"> -Diferentes maneras de presentar los datos como creación de modelos estadísticos y de minería de datos, como árboles de decisión, máquinas de vector soporte, regresiones, etc. -Aplicación de modelos creados con Knime sobre conjuntos nuevos de datos. -Creación de informes -Posible incorporación de nuevos módulos en lenguaje R o Python por el usuario final. <p>Tamaño de instalación: 1.98GB para la versión gratuita [9]</p>	<ul style="list-style-type: none"> Gratuita (Menos funcionalidad)/ De costo Desde \$44,200 pesos al año por un usuario Hasta \$466,410 pesos al año por 5 usuarios
SPSS Modeler IBM	<ul style="list-style-type: none"> -Posible incorporación de nuevos módulos en lenguaje R Python, Hadoop, Spark u otras tecnologías de código abierto por el usuario final. -Diferentes maneras de presentar los datos mediante la creación de modelos. -Acceso y Exportación de datos sin límite de tamaño. - Más de 30 algoritmos de minería de datos implementados -Ejecuta flujos de datos directamente hacia Spark o Hadoop -Soporte técnico <p>Tamaño de instalación: 1.4GB para la versión de prueba [10]</p>	<ul style="list-style-type: none"> Gratuita (Prueba 30 días)/De costo Es necesario contactar directamente con IBM para obtener el precio de la herramienta personalizado para cada empresa
Rapid Miner	<ul style="list-style-type: none"> -Cuenta con gráficos para visualización de información -Comparte reutiliza e implementa modelos predictivos -Ejecuta flujos de datos directamente hacia Hadoop -Presentación de datos desde diferentes representaciones gráficas en tiempo real incluso la creación de árboles de decisión [11] 	<ul style="list-style-type: none"> Desde \$2500.00 pesos al año hasta \$10000.00 pesos al año

- Software de administración de clústers.

SOFTWARE	CARACTERÍSTICAS	PRECIO EN EL MERCADO
Databricks	-Optimizado para lenguaje R. -Muy simple de operar. -Se enfoca en la usabilidad y en procesos intuitivos de navegación. [12]	-Basic - \$0.07 USD por hora de uso de unidad de procesamiento. -Analytics - \$0.2 USD por hora de uso de unidad de procesamiento.[15]
Cloudera Manager	-Plataforma de administración de clústers. -Tiene su propia implementación de búsqueda y ambiente Hadoop con integración y API estándar (CDH). -Cuenta con una interfaz gráfica vía web que opera el stack completo de tecnologías de Big Data. [13]	-Gratuita (Prueba de 60 días) -Essentials \$0.06 USD por hora de uso de unidad de procesamiento -Enterprise Data Hub \$0.87 USD por hora de uso de unidad de procesamiento. [16]
HortonWorks	-Completamente open-source. -Cuenta con un set completo de herramientas necesarias para un ambiente Hadoop. -Se puede lanzar en la nube o en una máquina virtual. -Gran extensibilidad debido a su naturaleza de código abierto. -Utiliza Ambari como interfaz de usuario. [14]	Gratis [17]

Podría decirse que *Luminus* es un híbrido entre las dos clasificaciones anteriormente citadas. Las principales características de *Luminus* son las siguientes:

- Trabaja con datos que se encuentra en distintos equipos de cómputo (Red distribuida).
- El usuario puede seleccionar el algoritmo de minería de datos a utilizar para su funcionamiento dentro de un listado de algoritmos disponibles.
- Se trabaja con archivos CSV O TXT de los datos de entrada.
- Se crea un ambiente que soporta una configuración básica para realizar tareas de Big Data para frameworks como Hadoop y Spark.

1.5. Justificación

El principal motivo de desarrollo de la presente propuesta es el de facilitar a las empresas u organizaciones la implementación de un ambiente de Big Data, dado que se trata de un proceso complicado, tardado y muy costoso. Cada uno de los factores mencionados anteriormente puede ser justificado de la siguiente manera: la complejidad de la implementación de Big Data se puede reducir mediante la preparación previa del software necesario para la infraestructura que tenga actualmente la empresa; de igual forma se pueden reducir los tiempos de implementación mediante la automatización de las configuraciones y pre procesamiento de datos; y finalmente los costos estarán de acuerdo a las computadoras disponibles (al menos 4) que puedan actuar como nodos de procesamiento y almacenamiento de datos que serán procesados.

Esta herramienta estaría dedicada a empresas u organizaciones que busquen implementar un ambiente de Big Data de una manera más rápida, sencilla, eficiente y a un menor costo [20]. Para que con su adecuado uso, cualquiera de estas entidades sea capaz de mejorar, ser más competitiva, y tener un mayor control sobre su propia información, ya que el conocimiento adquirido a partir de la aplicación de técnicas de minería de datos sobre grandes volúmenes de información estaría siendo de vital importancia para que dichas organizaciones planeen y ejecuten estrategias que les permitan alcanzar sus objetivos [19].

Se ha hecho especial hincapié en el sector empresarial en esta sección, ya que, en general, podría decirse que pequeñas y medianas empresas son los principales clientes potenciales. Esto debido a que la mayoría cuenta con cierta infraestructura de datos y con bitácoras que contienen información sobre sus ingresos y egresos. Toda esa información será

la materia prima que será utilizada por la herramienta para la generación de conocimiento.

Si bien, existen algunas implementaciones comerciales como puede verse en la sección **Estado del arte** que realizan la automatización de la implementación de Big Data, son realmente costosas en recursos de cómputo, y el licenciamiento tiene un costo elevado con relación a los usuarios y/o equipos a usar [21]. Algunas implementaciones están basadas en máquinas virtuales, que, si bien, automatizan todo el trabajo, de igual forma resultan costosas con relación al equipo necesario y dinero invertido [18]. Además, el costo de las herramientas comerciales más robustas puede llegar a los cientos de miles de pesos.

1.6. Objetivos

1.6.1. General

Desarrollar un software que permita realizar análisis de datos para la toma de decisiones, mediante la implementación de un ambiente de cómputo distribuido basado en Big Data, y aplicando técnicas de aprendizaje de máquina (machine learning), como reglas de asociación y árboles de decisión, a grandes volúmenes de datos.

1.6.2. Específicos

- Preparación del ambiente de software (instalación, configuración, validación del software del ambiente distribuido) para la implementación de Big Data
- Programación de las rutinas de análisis de datos; algoritmos de aprendizaje de máquina.
- Pruebas y verificación de funcionamiento
- Generación de reportes del análisis de datos.

1.7. Arquitectura de la red distribuida

El siguiente diagrama muestra la arquitectura prevista del sistema:

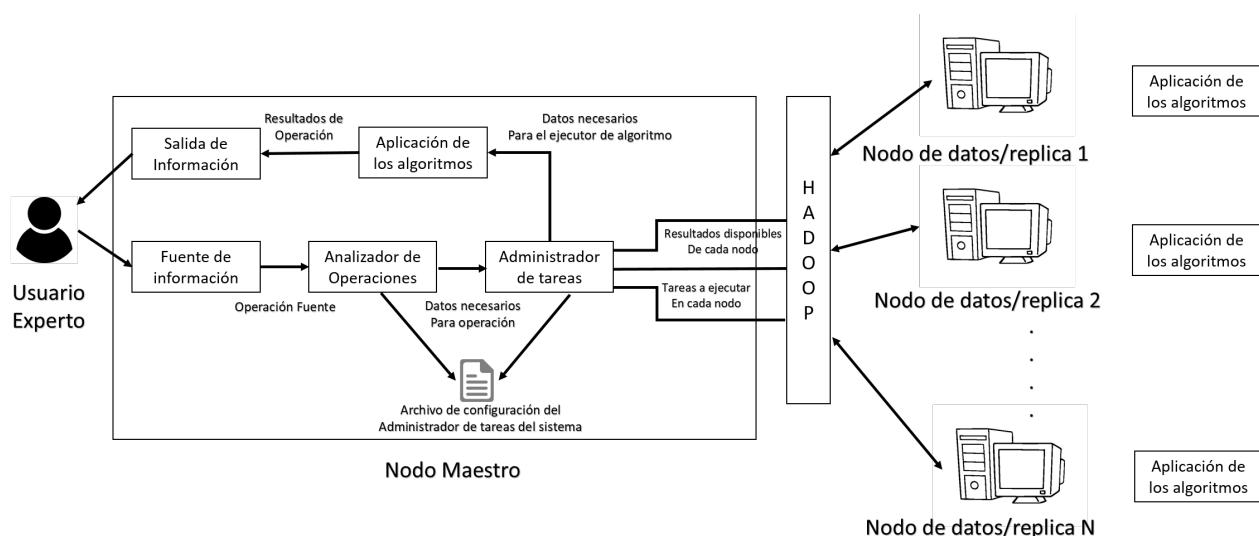


Figura 1.1: Arquitectura de la red distribuida

Con esta arquitectura será posible adaptar el ambiente de Big Data a diversos casos de estudio de las empresas, modificando únicamente archivos que serán destinados para la configuración de dichos casos de estudio.

Esto sin tener que modificar la arquitectura del sistema ni entender su funcionamiento interno. Será posible mantener muy bien separados los bloques de código que realizan cada actividad en el sistema para que en caso de algún fallo sea más fácil detectar en dónde está ocurriendo.

A continuación se explicarán cada uno de los bloques que conforman la arquitectura del sistema.

- **Salida de información:** Es el bloque que se encarga de presentar el resultado final al usuario experto, luego de aplicar los algoritmos de Big Data sobre ellos.
- **Aplicación de los algoritmos:** Es el modulo que permite aplicar los algoritmos de minería de datos sobre los nodos de datos/réplica a los datos. ademas de aplicar la función reduce con los resultados en el nodo maestro. El funcionamiento de la función reduce se explica en la sección del documento [4.3.2](#) dentro del capítulo [Integración de requerimientos de la red distribuida](#).
- **Fuente de información:** Es el modulo que recibe como dato de entrada el algoritmo de minería de datos a ejecutar del usuario conocida como operación fuente.
- **Analizador de operaciones:** Es el modulo encargado de revisar los datos del caso de estudio en cuestión para validar si las configuraciones escritas en el **Archivo de configuración del administrador de tareas** son correctas y suficientes para la ejecución del algoritmo que se seleccione, permite modificar estas configuraciones en caso de que así lo requieran.
Por otro lado, valida que el archivo de entradas pueda ser soportado por el algoritmo seleccionado.
- **Administrador de tareas:** es el encargado de administrar las tareas que llevará a cabo cada nodo de datos/réplica así como también de recibir los resultados de las tareas que estos ejecuten de manera local en el nodo maestro.
- **Archivo de configuración del administrador de tareas :** Este archivo servirá para llevar a cabo el ajuste de los algoritmos de minería de datos a cada caso de estudio que quiera aplicar el usuario experto. sera necesario realizar algunos ajustes en el mismo para que los algoritmos puedan ser ejecutados de manera satisfactoria.

La arquitectura que se presenta por lo tanto, es una arquitectura del tipo maestro-esclavo, la cual es utilizada para el diseño de un cluster de Hadoop como se detalla en la sección [Hadoop](#) del marco teórico.

El cluster de Hadoop es el que se utiliza para ambientes distribuidos que hacen uso de Big Data dentro de su funcionamiento al igual que Luminus.

1.8. Alcances y Limitaciones

1.8.1. Limitaciones

A continuación se definirán los alcances y limitaciones que tiene este trabajo terminal.

Primeramente las limitaciones que se han detectado hasta ahora en cuanto a equipo de computo para poder ejecutar el ambiente de Big Data por parte del usuario experto:

- Se necesitan al menos 2 computadoras para poder tener un cluster distribuido, por lo que, a pesar de que las tecnologías pueden ejecutar en un solo nodo de trabajo, se perderían las ventajas que ofrece el computo distribuido.
- Es necesario que los equipo que se utilicen cuenten con el triple del espacio de almacenamiento del archivo completo correspondiente al caso de estudio cada uno en disco duro, para que además de permitir el almacenamiento de los datos, también sea posible tener espacio para trabajar con ellos.
- Equipos con 1GB de RAM o menos no tienen un funcionamiento apropiado con Apache Hadoop y su rendimiento es inconsistente por lo que, no se recomienda trabajar en equipos de con estas características.

Por otro lado existen otro tipo de limitaciones que se presentan durante el desarrollo de este trabajo terminal:

- Solo se tendrán 2 algoritmos de minería de datos para ejecutar dentro del ambiente de Big Data, para un entorno mas completo seria conveniente incluir mas algoritmos.

- Las pruebas y desarrollo sobre este trabajo terminal en equipos de computo tradicionales serán ejecutadas únicamente con 3 nodos de datos/replica, por lo que, para un mejor desempeño sería conveniente agregar mas nodos de datos/replica al cluster.
- El archivo de datos proveniente del caso de estudio es de 21 GB, una empresa podría manejar archivos de mayor longitud, por lo que las pruebas realizadas podrían no ser significativos para la cantidad de datos que puedan manejar determinadas empresas.
- Las tablas de datos que conformen el caso de estudio que se suban a el repositorio , tendrán que subirse por separado y el manejo de los datos que se podrá ejecutar descartará las relaciones que existan entre las tablas y procesará cada tabla por separado.
- Es posible que no se puedan realizar pruebas o bien suficientes pruebas en equipos de computo especializados para realizar una comparativa del comportamiento que tiene Hadoop, bajo diferentes características computacionales ya que este no es el objetivo del proyecto, sino lo que se busca principalmente establecer el ambiente de análisis de datos.
- El instalador de Luminus tiene muchos casos de errores o casos especiales que no se considerarán y se dejarán a solución del usuario, cuando estos ocurran, ya que por cuestiones de tiempo es imposible englobarlos todos.

Se listan las limitaciones que se han encontrado hasta este momento, sin embargo podrían existir otras limitaciones que no se hayan encontrado y que en su caso deberían tomar su tiempo de análisis para evaluar su impacto en el proyecto.

1.8.2. Alcances

Los alcances fueron definidos al momento de elaborar el cronograma de actividades y estos están establecidos de los prototipos 1 al 5 del cronograma mencionado.

Durante el desarrollo de este trabajo se fusionaron los prototipos 4 y 5 formando en su conjunto un solo prototipo 4, además, este prototipo cambio su definición y en lugar de realizarse el desarrollo de un sitio web para la presentación y puesta en marcha de los algoritmos para el usuario final se realizó una Interfaz de Programación de Aplicaciones que fuera capaz de presentar estos resultados. A continuación, se procede a listar los alcances que se tienen para cada uno de los prototipos planteados:

1.8.3. Prototipo 1

El prototipo 1 tiene como productos esperados:

- Identificar y reconocer las características de los datos que conforman el caso de estudio, así como analizar y definir las posibles agrupaciones que pueden ser generadas a partir de estos datos. Los resultados de este análisis a detalle se encuentran en la sección [Análisis de los datos](#) del capítulo 3.
- Establecer el número de nodos y características que tendrán los nodos así como definir de estas cuales se agregarán a el cluster para realizar las pruebas del caso de estudio de manera distribuida. Buscando que con el número de nodos que se defina se pueda soportar dicho caso de estudio y que además no escapan de las características de computación con las que se cuenta actualmente. Los resultados obtenidos se pueden consultar en las secciones [Análisis de la red distribuida](#) y [Diseño de la red distribuida](#).
- Poner en funcionamiento el cluster con las características que fueron definidas en este mismo Prototipo para Apache Spark.
El desarrollo que implico dicha tarea se presenta a detalle en la sección [Desarrollo](#) mientras que las pruebas que se hicieron para demostrar que este desarrollo es funcional se pueden consultar en la sección [Pruebas](#) del capítulo [Evaluación y definición de requerimientos de la red distribuida](#)

El prototipo 2 tiene como productos esperados:

- Cargar los datos que conforman el caso de estudio de prueba a el clúster en funcionamiento que se explica en la sección [1.8.3](#) en su tercer producto esperado. Con lo cual los datos quedarían almacenados de forma distribuida en el clúster.
Para conocer más detalles de la carga de los datos que se realizó se puede consultar la sección [Desarrollo](#).
- Una vez que los datos son cargados satisfactoriamente a el clúster será necesario comprobar que estos pueden funcionar de manera distribuida y son accesibles, para ello se aplica un algoritmo sencillo sobre los datos. Para conocer más detalles acerca de esta prueba se puede consultar a partir de la sección [Algoritmo de prueba](#) de su correspondiente capítulo.

1.8.4. Prototipo 3: Algoritmos de minería de datos

El prototipo 3 tiene como productos esperados:

- Hacer uso de un algoritmo de cada uno de los tipos de algoritmos que se enlistan a continuación:

- Reglas de asociación
- Árboles de decisión

escrito en java que se adapte al caso de estudio, así como su personalización para que pueda ser adaptado a otros casos de estudio de manera sencilla.

- Elaboración de las pantallas que conformarán el sitio web, diseño, estructura y contenido que se puede ver como un mapa de navegación que aún no está integrado a Luminus por lo tanto, no es funcional.

1.8.5. Prototipo 4: Interfaz de Programación de Aplicaciones

El prototipo 4 tiene como productos esperados:

- Conjunto de clases o métodos que permitan la interacción con el Hadoop Distributed File System para el manejo de archivos y de operaciones que se tengan que ejecutar dentro del mismo.
- Validación de los datos para confirmar que cumplan con las condiciones para ser ejecutados por el algoritmo que el usuario desee y en caso de no cumplirlas no permitir su ejecución.
- Preparación de los algoritmos para que puedan ser soportados y ejecutados desde un programa escrito en JAVA, así como realizar las configuraciones que estos necesiten.

1.8.6. Prototipo Adicional: Instalador

Un instalador suele ser un sistema bastante complejo, ya que realiza las siguientes tareas:

- Instalación de software
- Ejecuta comandos en el shell de la computadora
- Verifica software que ya está instalado en la computadora y generalmente es capaz de cambiar la versión

Este prototipo tiene como productos esperados:

- Que sea capaz de reanudar el proceso de instalación cuando éste haya sido interrumpido. Ya sea por decisión del usuario o por un error en el proceso de instalación. El instalador, debe retomar el proceso a partir del punto en el que fue interrumpido.
- Que pueda agregar tantos nodos a la red como se deseen cuando se inicia la instalación inicial.
- Reduzca la cantidad de pasos a ejecutar por el usuario final para realizar una instalación completa.
- Reduzca el tiempo de instalación, en relación al tiempo que tomaría hacer la misma instalación tomando como referencia el *Manual de instalación de Luminus*
- Incluir todos los pasos del *Manual de instalación de Luminus* dentro del *Instalador*

CAPÍTULO 2

Marco teórico

En esta sección se presentan los conceptos que fundamentan este trabajo. Con el objetivo de dar al lector el conocimiento necesario para comprender los capítulos que integran la documentación de *Luminus*.

En la actualidad, la necesidad de almacenamiento de datos crece a niveles exponenciales. Muchas empresas están optando por implementar algún sistema informático para llevar el almacenamiento y gestión de sus datos. Empresas grandes como Google, Facebook, Youtube y demás empresas que diariamente procesan y almacenan cantidades gigantescas de información, requieren usar técnicas un tanto específicas para poder realizar este procesamiento. Ya que al ser tan grande la cantidad de datos, técnicas convencionales serían de poca o nula utilidad para llevar a cabo esa tarea. Una forma de lograrlo se encuentra en un conjunto de técnicas conocidas como *Big Data*.

2.1. Big Data

Al ser el Big Data un concepto tan importante para este trabajo, se procederá a definirlo, ya que como se menciona en el **Capítulo 1**, *Luminus* realizará procesamiento de grandes cantidades de información.

El término Big Data se refiere a cantidades enormes de información, por ejemplo, la cantidad de información que se produce todos los días con el uso de una red social como Facebook, o la cantidad de datos que producen computadoras y dispositivos electrónicos que se auto monitorean mediante sensores. Esos volúmenes masivos de datos pueden ser utilizados para extraer conocimiento de ellos, y posteriormente atacar problemas que no sería posible resolver sin el Big Data.[24]

2.1.1. Las 3 Vs del Big Data

Al ser el Big Data un concepto emergente y relativamente nuevo, se han tenido ciertas dificultades para definirlo de manera uniforme. Debido a las dimensiones de todo lo que conlleva entender Big Data, resultó conveniente entre los estudiosos del tema definir y acentuar las magnitudes que lo definen. Estas magnitudes se conocen como las 3 Vs del Big Data.

1. **Volumen:** Con Big Data, se tendrán que procesar enormes volúmenes de información. Este punto es importante ya que la necesidad de almacenamiento de datos en el mundo moderno, crece de forma exponencial.[24]
2. **Velocidad:** Usando Big Data, se abre la posibilidad de acceso y flujo de datos a velocidades que no se podrían conseguir de manera convencional.[24]
3. **Variedad:** Procesamiento de datos de naturaleza heterogénea, es decir, múltiples tipos de datos.[24]

2.1.2. Casos de uso del Big Data

- **Desarrollo de productos:** Compañías como Netflix y Procter & Gamble utilizan el Big Data para anticiparse a la demanda de los consumidores de sus productos. Utilizan modelos predictivos para sus nuevos productos clasificando atributos clave de sus anteriores productos modelando las relaciones entre esos atributos.[25]
- **Mantenimiento predictivo:** Se pueden predecir fallas mecánicas en maquinaria que de otra forma quedarían fácilmente ignoradas. Mejorando así ampliamente la calidad y el costo del mantenimiento de dichos equipos.[25]
- **Experiencia de usuario:** El uso de Big Data permite recopilar toda la información sobre el usuario y utilizarla a favor de su experiencia en un sitio o aplicación. Por ejemplo, sus búsquedas frecuentes, los sitios que visita, etc. Para de esta manera empezar a hacerle ofertas o anuncios personalizados, según sus intereses particulares.[25]
- **Machine Learning:** Actualmente el machine learning es un área de mucho auge, ya que permite entrenar a las computadoras mediante conjuntos de datos de entrenamiento en lugar de programarlas. El Big Data facilita esa tarea.[25]

2.2. Minería de datos

La minería de datos es el proceso de generar conocimiento a partir de un conjunto de información de gran tamaño. Para generar ese conocimiento se utilizan diferentes técnicas de análisis que detectan patrones y tendencias en la información que se está procesando. Si se intentara utilizar algún método de análisis tradicional, sería muy complicado o incluso imposible a veces encontrar patrones y tendencias útiles, ya que es muy probable que dentro de los datos existan relaciones muy complejas o simplemente la cantidad de datos sea demasiado grande.[26]

La minería de datos puede utilizarse en escenarios como los que se enuncian a continuación:

- **Pronóstico:** Predicción de datos y eventos que vendrán a futuro a partir del comportamiento de conjuntos de datos que se tienen en el presente. Por ejemplo, predicción de ventas y tendencias de compra.[27]
- **Riesgo y probabilidad:** Es un escenario muy común dentro de los negocios de Business Intelligence. Por ejemplo, se llega a utilizar para encontrar puntos de equilibrio probable para escenarios de riesgo. Elección de los mejores clientes para la distribución de correo directo, determinación del punto de equilibrio probable para los escenarios de riesgo, y asignación de probabilidades a diagnósticos y otros resultados.[27]
- **Recomendaciones:** Muy utilizado en sistemas como MercadoLibre o Amazon, en módulos que toman la información de búsqueda de cada usuario, la procesan y le arrojan recomendaciones de productos o servicios.[27]
- **Búsqueda de secuencias:** Al igual que el escenario de **recomendaciones**, se utiliza mucho en sistemas de venta de artículos por internet. Se analiza el orden de los artículos que se meten a un carrito de compra para poder hacer predicciones útiles y generar conocimiento.[27]
- **Agrupación:** Clasificación de los elementos de un conjunto de información para el posterior análisis de sus afinidades.[27]

2.3. Árboles de decisión

Una de las técnicas de análisis que *Luminus* utilizará para generar conocimiento es el uso de **árboles de decisión**. Un árbol de decisión es un modelo de predicción que apoya al proceso de toma de decisiones. Esta herramienta tiene un campo de aplicación extremadamente amplio, pudiendo ir desde el área de finanzas hasta el área del aprendizaje de máquina. A partir de un conjunto de datos de entrada, se construyen los caminos, dentro del árbol, que llevarán a cada una de las decisiones posibles.[28]

Los árboles de decisión están formados por los siguientes elementos:

1. **Nodos:** Un nodo es un punto del proceso en el que de acuerdo a ciertas condiciones o decisiones se redefine el rumbo del camino. Existen dos tipos de nodo[29]:
 - 1.1. **Nodo de decisión:** Es un nodo en el cual se toma una decisión consciente de acuerdo a las necesidades del problema en cuestión. Estos nodos se representan con un cuadrado.
 - 1.2. **Nodo de incertidumbre:** Es un nodo en el que actúan las probabilidades y la heurística para definir el rumbo que tomará el camino que se hará dentro del árbol. Estos nodos se representan con un círculo
2. **Ramas:** Una rama es una de las respuestas o acciones que se toman a partir de la pregunta o escenario que se presentó en un nodo del cual salió esa rama. Una rama es el camino a otro nodo o escenario resultado de la decisión o evento que definió la rama. Este elemento se representa con una línea[29].
3. **Hojas:** Son escenarios finales, ya clasificados. No tienen ramificaciones, y son el resultado final de seguir un camino de decisiones, acciones y probabilidades. Este elemento se representa con un triángulo[29].

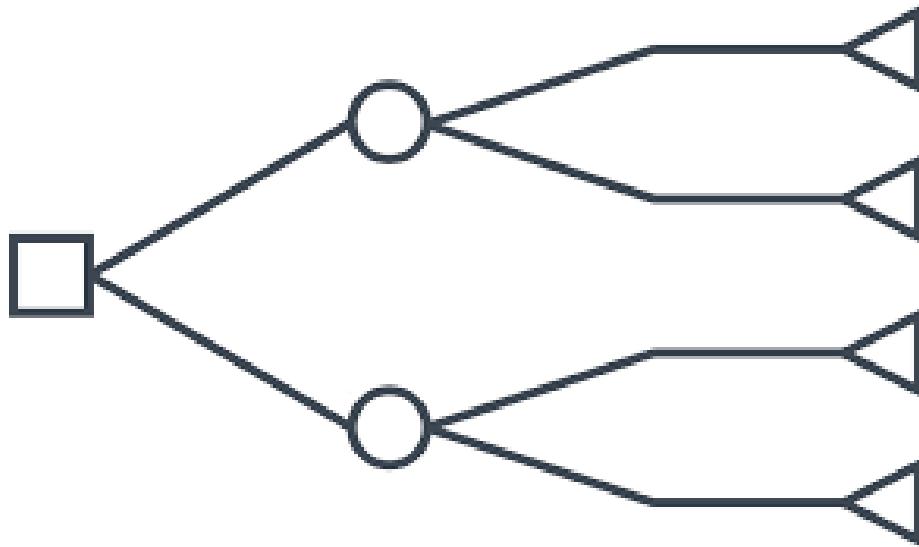


Figura 2.1: Representación general de un árbol de decisión

A continuación se presentan los algoritmos que usan árboles de decisión que tendrá implementados *Luminus* para realizar el análisis de datos.

2.4. ID3 (Iterative Dichotomiser 3)

El algoritmo ID3 es uno de los algoritmos que utilizan árboles de decisión más populares. ID3 genera un árbol a partir de un conjunto de datos llamado **tabla de inducción**. Es útil para hacer toma de decisiones en escenarios binarios, es decir, que tienen 2 posibilidades finales.[30]

Es importante mencionar este algoritmo debido a que es el precursor de otro algoritmo conocido como **C4.5**. Que es un algoritmo que apoya a la toma de decisiones, más avanzado y completo. Por otro lado, en *Luminus* se realizará la implementación de este algoritmo.

El resultado de la ejecución de este algoritmo puede expresarse como un conjunto de reglas **si-entonces**.

2.4.1. Entropía

La entropía es la medida de la aleatoriedad. En otras palabras, la medida e la impredictibilidad. La entropía es más alta cuando todos los eventos posibles en un escenario son igualmente probables, por ejemplo, al tirar una moneda al aire, se tiene 50 % de probabilidad de que caiga cara y 50 % de probabilidad de que caiga cruz, por lo que la entropía es de 1. Este parámetro comienza a decrecer cuando hay una probabilidad o probabilidades que parezcan aplastantes sobre las otras. Las fórmulas para calcular la entropía y la ganancia son las siguientes:

$$\text{Entropia}(S) = \sum_{i=1}^n -p_i \log_2 p_i$$

Figura 2.2: Fórmula para calcular la entropía.

$$\text{Gan Inf}(S, A) = \text{Entropia}(S) - \sum_{v \in V(A)} \frac{|Sv|}{|S|} \text{Entropia}(Sv)$$

Figura 2.3: Fórmula para calcular la ganancia.

2.4.2. Ejemplo de ejecución del algoritmo ID3

Para ilustrar el funcionamiento del algoritmo ID3, utilizaremos la siguiente tabla de inducción que contiene información sobre factores que influyen al tomar la decisión de si un partido de tenis debería o no jugarse:

Día	Aspecto	Temperatura	Humedad	Viento	Decisión
1	Soleado	Caluroso	Alta	Ligero	No
2	Soleado	Caluroso	Alta	Fuerte	No
3	Nublado	Caluroso	Alta	Ligero	Sí
4	Lluvioso	Templado	Alta	Ligero	Sí
5	Lluvioso	Fresco	Normal	Ligero	Sí
6	Lluvioso	Fresco	Normal	Fuerte	No
7	Nublado	Fresco	Normal	Fuerte	Sí
8	Soleado	Templado	Alta	Ligero	No
9	Soleado	Fresco	Normal	Ligero	Sí
10	Lluvioso	Templado	Normal	Ligero	Sí
11	Soleado	Templado	Normal	Fuerte	Sí
12	Nublado	Templado	Alta	Fuerte	Sí
13	Nublado	Caluroso	Normal	Ligero	Sí
14	Lluvioso	Templado	Alta	Fuerte	No

Tabla 2.1: Tabla de inducción para juegos de tenis ID3.

Necesitaremos de la [2.2 fórmula para calcular la entropía](#) y de la [2.3 fórmula para calcular la ganancia](#) para poder proceder con la ejecución del algoritmo.

- Primeramente se calcula la entropía. La columna de **Decisión** consta de 14 instancias e incluye dos posibles valores: **sí** y **no**. Hay 9 decisiones con la etiqueta **sí** y 5 con la etiqueta **no**. Utilizando la fórmula de la entropía, se encuentra que el resultado es de **0.940**.
- Despues de los 5 factores disponibles, se busca el factor más dominante para tomar la decisión. Utilizando la fórmula de la ganancia tomando como parámetros la entropía de la columna **Decisión** y **Viento**.
- El atributo de viento tiene dos posibles valores: **fuerte** y **ligero**. Y al reflejar estos dos posibles valores en la fórmula, ésta se vería más o menos así: $Ganancia(Decisión, Viento) = Entropía(Decisión) - [p(Decisión, Viento = Ligero)*Entropía(Decisión, Viento = Ligero)]-[p(Decisión, Viento = Fuerte)*Entropía(Decisión, Viento = Fuerte)]$.
- Ahora se calcula $(Decisión, Viento = Ligero)$ y $(Decisión, Viento = Fuerte)$ respectivamente.

Día	Aspecto	Temperatura	Humedad	Viento	Decisión
1	Soleado	Caluroso	Alta	Ligero	No
3	Nublado	Caluroso	Alta	Ligero	Sí
4	Lluvioso	Templado	Alta	Ligero	Sí
5	Lluvioso	Fresco	Normal	Ligero	Sí
8	Soleado	Templado	Alta	Ligero	No
9	Soleado	Fresco	Normal	Ligero	Sí
10	Lluvioso	Templado	Normal	Ligero	Sí
13	Nublado	Caluroso	Normal	Ligero	Sí

Tabla 2.2: Tabla de para el factor de decisión *Viento Ligero*.

Dentro de la tabla de viento ligero hay 8 instancias en total, de las cuales 2 tienen la decisión final **no** y 6 tienen la decisión final **sí**. Al introducir estos datos en la [2.2 fórmula de la entropía](#), y obtenemos una entropía de **0.811**.

Día	Aspecto	Temperatura	Humedad	Viento	Decisión
2	Soleado	Caluroso	Alta	Fuerte	No
6	Lluvioso	Fresco	Normal	Fuerte	No
7	Nublado	Fresco	Normal	Fuerte	Sí
11	Soleado	Templado	Normal	Fuerte	Sí
12	Nublado	Templado	Alta	Fuerte	Sí
14	Lluvioso	Templado	Alta	Fuerte	No

Tabla 2.3: Tabla de para el factor de decisión *Viento Fuerte*.

En la tabla de viento fuerte encontramos 6 instancias divididas en 2 partes iguales: 3 tienen la decisión final **sí** y 3 tienen la decisión final **no**. Al sustituir estos datos en la [2.2 fórmula de la entropía](#), obtenemos una entropía de 1.

- Ahora que tenemos esos dos valores, podemos volver a la ecuación de la ganancia. El resultado queda expresado como $Ganancia(Decisión, Viento) = 0.940 - [(8/14)*0.811] - [(6/14)*1] = 0.048$.

- En este punto se ha concluido ya el cálculo para el factor de decisión *Viento*. Ahora se necesita hacer lo mismo para todas las demás columnas/factores.

- A modo de resumen:

1. $Ganancia(Decisión, Aspecto) = 0.246$

2. $Ganancia(Decisión, Temperatura) = 0.029$

3. $Ganancia(Decisión, Humedad) = 0.151$

- Como podemos ver, el factor de decisión *Aspecto* es el que produce el valor de ganancia más alto. Por tal motivo, ese factor de decisión será el nodo raíz del árbol de decisión.

- Ahora debemos probar todos los valores posibles que tiene el factor de decisión *Aspecto*:

- Aspecto Nublado

Día	Aspecto	Temperatura	Humedad	Viento	Decisión
3	Nublado	Caluroso	Alta	Ligero	Sí
7	Nublado	Fresco	Normal	Fuerte	Sí
12	Nublado	Templado	Alta	Fuerte	Sí
13	Nublado	Caluroso	Normal	Ligero	Sí

Tabla 2.4: Tabla de días con aspecto *nublado*.

Podemos observar que siempre que el aspecto del día sea **nublado**, la decisión final sería **sí**.

- Aspecto Soleado

Día	Aspecto	Temperatura	Humedad	Viento	Decisión
1	Soleado	Caluroso	Alta	Ligero	No
2	Soleado	Caluroso	Alta	Fuerte	No
8	Soleado	Templado	Alta	Ligero	No
9	Soleado	Fresco	Normal	Ligero	Sí
11	Soleado	Templado	Normal	Fuerte	Sí

Tabla 2.5: Tabla días con aspecto *soleado*.

En esta tabla tenemos 5 instancias para el aspecto **soleado**. Hay 3 de esas instancias que tienen como decisión final **sí** y 2 que tienen **no**. Por lo que los valores de ganancia del factor de decisión **Aspecto Soleado** con respecto a todos los demás factores de decisión quedarán de la siguiente manera:

1. Ganancia(Aspecto = Soleado, Temperatura) = 0.570
2. Ganancia(Aspecto = Soleado, Humedad) = 0.970
3. Ganancia(Aspecto = Soleado, Viento) = 0.019

En este punto, la humedad es el factor de decisión con mayor ganancia cuando el aspecto del día es soleado. Por lo que debemos probar todos los valores posibles para el factor de decisión *humedad*.

Día	Aspecto	Temperatura	Humedad	Viento	Decisión
1	Soleado	Caluroso	Alta	Ligero	No
2	Soleado	Caluroso	Alta	Fuerte	No
8	Soleado	Templado	Alta	Ligero	No

Tabla 2.6: Tabla días con aspecto *soleado* y humedad *alta*.

La decisión siempre será **no** cuando la humedad sea **alta**.

Día	Aspecto	Temperatura	Humedad	Viento	Decisión
9	Soleado	Fresco	Normal	Ligero	Sí
11	Soleado	Templado	Normal	Fuerte	Sí

Tabla 2.7: Tabla días con aspecto *soleado* y humedad *normal*.

Por otro lado, la decisión siempre será **sí** cuando la humedad es **normal**.

De lo anterior concluimos que deberemos verificar la humedad y decidir si el aspecto del día es **soleado**.

- Aspecto Lluvioso

Día	Aspecto	Temperatura	Humedad	Viento	Decisión
4	Lluvioso	Templado	Alta	Ligero	Sí
5	Lluvioso	Fresco	Normal	Ligero	Sí
6	Lluvioso	Fresco	Normal	Fuerte	No
10	Lluvioso	Templado	Normal	Ligero	Sí
14	Lluvioso	Templado	Alta	Fuerte	No

Tabla 2.8: Tabla de días con aspecto *lluvioso*.

Al evaluar los valores de ganancia de los días con aspecto **lluvioso** con respecto a los demás factores de decisión, se encuentra que el factor que genera una mayor ganancia es el viento. Por lo cual se tienen que checar todos los posibles valores de ese factor de decisión.

Día	Aspecto	Temperatura	Humedad	Viento	Decisión
4	Lluvioso	Templado	Alta	Ligero	Sí
5	Lluvioso	Fresco	Normal	Ligero	Sí
10	Lluvioso	Templado	Normal	Ligero	Sí

Tabla 2.9: Tabla de días con aspecto *lluvioso* y con viento *ligero*.

De la tabla de aspecto **lluvioso** y viento **ligero** podemos deducir que siempre que existan estas dos condiciones al mismo tiempo, la decisión final será **sí**.

Día	Aspecto	Temperatura	Humedad	Viento	Decisión
6	Lluvioso	Fresco	Normal	Fuerte	No
14	Lluvioso	Templado	Alta	Fuerte	No

Tabla 2.10: Tabla de días con aspecto *lluvioso*.

De la tabla de aspecto **lluvioso** y viento **fuerte** podemos deducir que siempre que existan estas dos condiciones al mismo tiempo, la decisión final será **no**.

- Finalmente la construcción de este árbol de decisión es la siguiente:

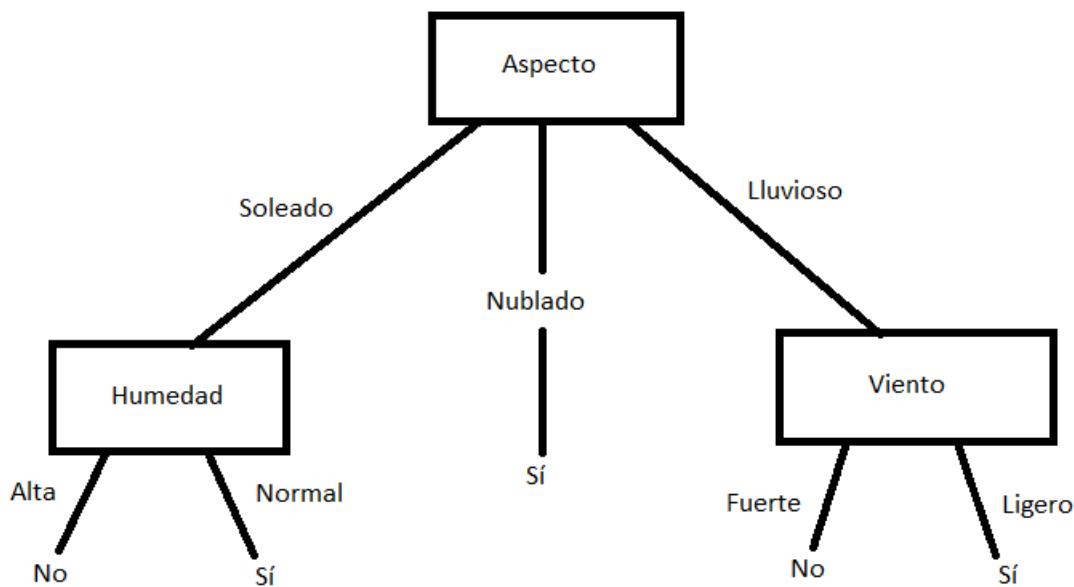


Figura 2.4: Árbol de decisión resultante.

2.5. C4.5

El algoritmo C4.5 es la evolución del algoritmo ID3. Éste genera un árbol de decisión a partir de un conjunto de datos de entrada de manera recursiva, al igual que su precursor [31]. Sin embargo, aunque ID3 y C4.5 son algoritmos muy semejantes, existen ciertas diferencias:

- C4.5 permite trabajar con valores continuos, mientras que ID3 solamente trabaja con valores discretos.
- Los árboles de C4.5 son más compactos, y esto se debe a que cada nodo hoja engloba un conjunto de clases y no una sola clase particular.
- C4.5 es más eficiente computacionalmente hablando.
- C4.5 utiliza un nuevo parámetro llamado **Gain Ratio**, en lugar de la ganancia simple. Y este a su vez requiere de la ganancia y de otro parámetro nuevo llamado **SplitInfo** cuyas fórmulas se enuncian a continuación[31]:

$$GainRatio(A) = \frac{Gain(A)}{SplitInfo(A)}$$

Figura 2.5: Fórmula para obtener el Gain Ratio

$$SplitInfo_A(D) = -\sum_{j=1}^v \frac{|D_j|}{|D|} \times \log_2\left(\frac{|D_j|}{|D|}\right)$$

Figura 2.6: Fórmula para obtener SplitInfo

2.5.1. Ejemplo de ejecución del algoritmo C4.5

Para mostrar la forma en la que se ejecuta el algoritmo C4.5 y los pasos que se deben llevar a cabo, se utilizará la siguiente tabla de inducción:

Día	Aspecto	Temperatura	Humedad	Viento	Decisión
1	Soleado	85	85	Ligero	No
2	Soleado	80	90	Fuerte	No
3	Nublado	83	78	Ligero	Sí
4	Lluvioso	70	96	Ligero	Sí
5	Lluvioso	68	80	Ligero	Sí
6	Lluvioso	65	70	Fuerte	No
7	Nublado	64	65	Fuerte	Sí
8	Soleado	72	95	Ligero	No
9	Soleado	69	70	Ligero	Sí
10	Lluvioso	75	80	Ligero	Sí
11	Soleado	75	70	Fuerte	Sí
12	Nublado	72	90	Fuerte	Sí
13	Nublado	81	75	Ligero	Sí
14	Lluvioso	80	80	Fuerte	No

Tabla 2.11: Tabla de inducción para juegos de tenis C4.5.

- Al igual que con el ejemplo [Ejemplo de ejecución del algoritmo ID3](#), lo primero que se hace es calcular la entropía general. Hay 15 instancias de las cuales 9 tienen una decisión final de **sí** y 5 tienen **no**. Al sustituir los valores correspondientes en la [2.2 fórmula para calcular la entropía](#) el resultado que obtenemos es **0.940**.

- En C4.5 se utilizan **Gain Ratios** (radios de ganancia), mientras que en ID3 se utilizan ganancias.
- Empezaremos por analizar el atributo de **Viento**.

Se tienen 8 instancias de *viento ligero*, dos de ellas concluyen en un **no**, y las otras 6 concluyen en **sí** por lo que:

- Entropía(Decisión, Viento = Ligero) = 0.811

2. Entropía(Decisión, Viento = Fuerte) = 1

3. Ganancia(Decisión, Viento) = 0.049

Existen 6 instancias de viento fuerte por lo que:

1. SplitInfo(Decisión, Viento) = 0.985

2. GainRatio(Decisión, Viento) = Gain(Decisión, Viento) / SplitInfo(Decision, Viento) = 0.049

• Continuamos analizando ahora el atributo **Aspecto**.

Se tienen 5 instancias para el aspecto *soleado*, de las cuales 3 concluyen en **no** y las otras 2 concluyen en **sí**. Calculando sus valores de entropías, ganancia, SplitInfo y GainRatio:

1. Entropía(Decisión, Aspecto = Soleado) = 0.970

2. Entropía(Decisión, Aspecto = Nublado) = 0

3. Entropía(Decisión, Aspecto = Lluvioso) = 0.970

4. Ganancia(Decisión, Aspecto) = 0.246

Hay 5 instancias para *soleado*, 4 instancias para *nublado* y 5 para *lluvioso*, por lo que:

5. SplitInfo(Decisión, Aspecto) = 1.577

6. GainRatio(Decisión, Aspecto) = 0.155

• Procedemos a analizar el atributo de humedad. Cuyo caso es diferente al de los demás atributos, ya que este es un atributo continuo. Necesitamos convertir valoress continuos a valores nominales (como todos los demás atributos). C4.5 propone hacer una división binaria a partir de algún valor que podamos tomar como umbral. El umbral debe ser el valor que mayor ganancia ofrezca para ese atributo. Para esto, primero se deben ordenar las instancias de humedad de menor a mayor.

Día	Humedad	Decisión
7	65	Sí
6	70	No
9	70	Sí
11	70	Sí
13	75	Sí
3	78	Sí
5	80	Sí
10	80	Sí
14	80	No
1	85	No
2	90	No
12	90	Sí
8	95	No
4	96	Sí

Tabla 2.12: Tabla de I atributo *humedad* ordenada de menor a mayor.

Ahora debemos recorrer todos los valores de humedad y separar el conjunto de datos en dos partes. Se calcularán la **Ganancia** y el **GainRatio** y el valor que maximice la ganancia será el umbral.

1. Se propone 65 como umbral.
 - 1.1. Entropía(Decisión, Humedad ≤ 65) = 0
 - 1.2. Entropía(Decisión, Humedad > 65) = 0.961
 - 1.3. Ganancia(Decisión, Humedad $\neq 65$) = 0.048
 - 1.4. SplitInfo(Decisión, Humedad $\neq 65$) = 0.371
 - 1.5. GainRatio(Decisión, Humedad $\neq 65$) = 0.126
2. Se propone 70 como umbral.
 - 2.1. Entropía(Decisión, Humedad ≤ 70) = 0.811
 - 2.2. Entropía(Decisión, Humedad > 70) = 0.970
 - 2.3. Ganancia(Decisión, Humedad $\neq 70$) = 0.014
 - 2.4. SplitInfo(Decisión, Humedad $\neq 70$) = 0.863
 - 2.5. GainRatio(Decisión, Humedad $\neq 70$) = 0.016
3. Se propone 75 como umbral.
 - 3.1. Entropía(Decisión, Humedad ≤ 75) = 0.721
 - 3.2. Entropía(Decisión, Humedad > 75) = 0.991
 - 3.3. Ganancia(Decisión, Humedad $\neq 75$) = 0.045
 - 3.4. SplitInfo(Decisión, Humedad $\neq 75$) = 0.940
 - 3.5. GainRatio(Decisión, Humedad $\neq 75$) = 0.047
4. Se continúa haciendo lo mismo para cada valor nuevo de humedad que se vaya encontrando en la tabla. Resumiendo:
5. Ganancia(Decisión, Humedad $\neq 78$) = 0.090

6. Ganancia(Decisión, Humedad $<>80$) = 0.107

7. Ganancia(Decisión, Humedad $<>85$) = 0.027

8. Ganancia(Decisión, Humedad $<>90$) = 0.016

- Como podemos ver, el valor que maximiza la ganancia es el de 80. Lo que significa que ahora se debe comparar los otros valores nominales con el valor 80 del atributo *humedad* para crear una rama en nuestro árbol. De esta forma podemos resumir los resultados en la siguiente tabla:

Atributo	Ganancia	GainRatio
Viento	0.049	0.049
Aspecto	0.246	0.155
Humedad $<>80$	0.101	0.107

Tabla 2.13: Comparación de las ganancias entre atributos.

- Al igual que en el ejemplo del algoritmo ID3, el atributo **aspecto** es el nodo raíz del árbol de decisión, por lo que ahora se deben analizar todos los posibles valores que puede adquirir dicho atributo.

- Aspecto Soleado.

Día	Aspecto	Temperatura	Humedad	Viento	Decisión
1	Soleado	85	85	Ligero	No
2	Soleado	80	90	Fuerte	No
8	Soleado	72	95	Ligero	No
9	Soleado	69	70	Ligero	Sí
11	Soleado	75	70	Fuerte	Sí

Tabla 2.14: Tabla de inducción para el atributo *Aspecto* con el valor *Soleado*.

Ya hemos dividido la humedad a partir de su punto de umbral que es 80. Podemos observar en la siguiente tabla que la decisión final será **no**, si la humedad es mayor a 80 y el aspecto del día es soleado. La decisión final será **sí**, si la humedad es menor o igual a 80 para un día soleado.

Día	Aspecto	Temperatura	Humedad	Viento	Decisión
3	Nublado	83	78	Ligero	Sí
7	Nublado	64	65	Fuerte	Sí
12	Nublado	72	90	Fuerte	Sí
13	Nublado	81	75	Ligero	Sí

Tabla 2.15: Tabla de inducción para el atributo *Aspecto* con el valor *Nublado*.

Cuando el aspecto del día es **nublado**, no importa ninguna otra condición; ni temperatura, ni humedad. La decisión final siempre será **sí**.

Día	Aspecto	Temperatura	Humedad	Viento	Decisión
4	Lluvioso	70	96	Ligero	Sí
5	Lluvioso	68	80	Ligero	Sí
6	Lluvioso	65	70	Fuerte	No
10	Lluvioso	75	80	Ligero	Sí
14	Lluvioso	80	80	Fuerte	No

Tabla 2.16: Tabla de inducción para el atributo *Aspecto* con el valor *Lluvioso*.

Cuando el aspecto es **lluvioso**, la decisión será **sí** cuando el viento es ligero, y será **no** cuando sea fuerte.

- De todo lo anterior se concluye un árbol de decisión con la siguiente estructura:

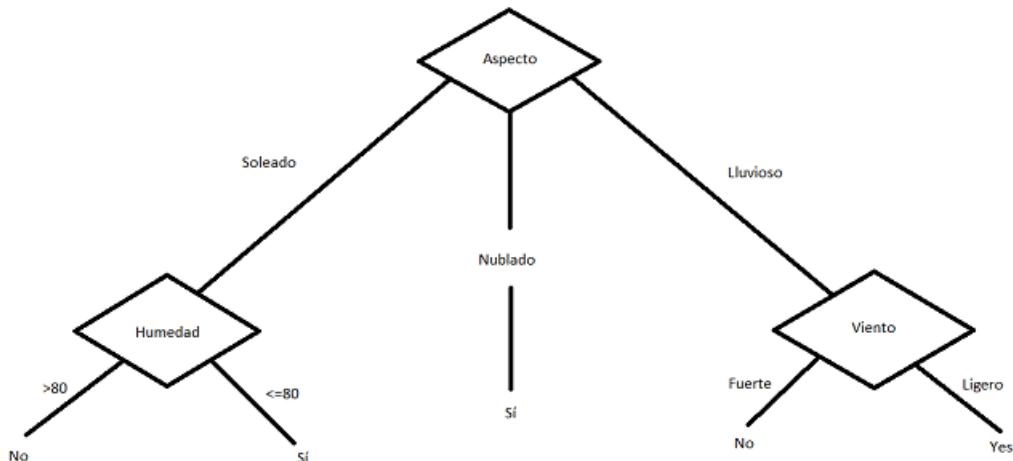


Figura 2.7: Árbol de decisión construido con C4.5.

2.6. Algoritmo KNN (K-Nearest Neighbors)

El algoritmo KNN es un algoritmo de aprendizaje no supervisado en el que se busca clasificar un punto en una categoría con la ayuda de un conjunto de entrenamiento [32]. Las siglas **KNN** en inglés hacen alusión a **K-Nearest Neighbors**, que significa *K* vecinos más cercanos. Este algoritmo calcula distancias euclidianas entre valores que son tomados como puntos. **KNN** se implementó en *Luminus*, ya que para el caso de estudio que se presentará, será necesario conocer cuáles son las tiendas más cercanas a la ubicación desde donde se esté utilizando el sistema.

Los pasos que sigue el algoritmo KNN se enumeran a continuación:

1. Se calcula la similitud entre puntos basándose en una función de distancia.
2. Se encuentran los *K* vecinos más cercanos.

2.6.1. Ejemplo de ejecución del algoritmo KNN

Con base en la siguiente información que relaciona el peso, la altura y la talla de playera de personas, se hará la ejecución del algoritmo KNN.

Altura (cm)	Peso (kg)	Talla
158	58	M
158	59	M
158	63	M
160	59	M
160	60	M
163	60	M
163	61	M
160	64	L
163	64	L
165	61	L
165	62	L
165	65	L
168	62	L
168	63	L
168	66	L
170	63	L
170	64	L
170	68	L

Tabla 2.17: Conjunto de entrenamiento para el algoritmo KNN.

Nuevo dato: Altura = 161 cm, Peso = 61 kg

- Se calcula la distancia Euclídea entre el dato de entrada y cada uno de los datos del conjunto de entrenamiento, utilizando la siguiente fórmula.

$$d_E(P, Q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2} = \sqrt{\sum_{i=1}^n (p_i - q_i)^2}.$$

Figura 2.8: Fórmula para calcular la distancia Euclíadiana.

Obtenemos lo siguiente:

Altura (cm)	Peso (kg)	Talla	Distancia
158	58	M	4.2
158	59	M	3.6
158	63	M	3.6
160	59	M	2.2
160	60	M	1.4
163	60	M	2.2
163	61	M	2.0
160	64	L	3.2
163	64	L	3.6
165	61	L	4.0
165	62	L	4.1
165	65	L	5.7
168	62	L	7.1
168	63	L	7.3
168	66	L	8.6
170	63	L	9.2
170	64	L	9.5
170	68	L	11.4

Tabla 2.18: Tabla del conjunto de entrenamiento con la columna **Distancia** ya calculada.

- Se toma la distancia más pequeña para determinar al vecino más cercano de entre todas las distancias Euclidianas calculadas previamente. Esta distancia es **1.4**.
- Se define K como el número de vecinos más cercanos que queramos conocer. Para este caso definiremos K igual a 5. Por lo que se tomarán los 5 valores más pequeños.

Altura (cm)	Peso (kg)	Talla	Distancia
160	60	M	1.4
163	61	M	2.0
163	60	M	2.2
160	59	M	2.2
160	64	L	3.2

Tabla 2.19: Instancias de la tabla con menor distancia al nuevo dato.

2.7. Algoritmo K-Means

El algoritmo K-Means es un algoritmo de agrupamiento que utiliza aprendizaje no supervisado, el cual es utilizado cuando se tienen datos no etiquetados, es decir sin categorías o grupos definidos. El objetivo de este algoritmo es encontrar grupos en los datos, con el número de grupos se representa la variable K [33]. Es importante definir este algoritmo y su funcionamiento dentro del marco teórico ya que nos da una visión más amplia de como funcionan los algoritmos de clasificación.

El algoritmo arrojará como resultado final lo siguiente:

1. Los centroides de los K grupos.
2. Etiquetas para los datos de entrenamiento. Cada dato es asignado a un solo grupo.

2.7.1. Ejemplo de ejecución del algoritmo K-Means

Para ilustrar el funcionamiento del algoritmo K-means, se utilizará el siguiente conjunto de entrenamiento correspondiente a la puntuación de 7 individuos en 2 pruebas:

Individuo	Prueba 1	Prueba 2
1	1.0	1.0
2	1.5	2.0
3	3.0	4.0
4	5.0	7.0
5	3.5	5.0
6	4.5	5.0
7	3.5	4.5

Tabla 2.20: Conjunto de entrenamiento para el algoritmo K-Means.

- El conjunto de datos se agrupa en dos grupos distintos. Para esto, se toman los valores de la *Prueba 1* y *Prueba 2* de los individuos que tengan estos valores más lejanos, es decir:

Grupo	Individuo	Vector (Centroide)
Grupo 1	1	(1.0, 1.0)
Grupo 2	4	(5.0, 7.0)

Tabla 2.21: Partición inicial.

- Los individuos restantes ahora son examinados secuencialmente y son ubicados en el grupo al que son más cercanos en términos de distancia Euclidiana. El vector es recalculado cada que un nuevo miembro es agregado.

Iteración	Individuo	Vector (Centroide)
1	1	(1.0, 1.0)
2	1, 2	(1.2, 1.5)
3	1, 2, 3	(1.8, 2.3)
4	1, 2, 3	(1.8, 2.3)
5	1, 2, 3	(1.8, 2.3)
6	1, 2, 3	(1.8, 2.3)

Tabla 2.22: Tabla de individuos agregados secuencialmente al **Grupo 1**.

Iteración	Individuo	Vector (Centroide)
1	4	(5.0, 7.0)
2	4	(5.0, 7.0)
3	4	(5.0, 7.0)
4	4, 5	(4.2, 6.0)
5	4, 5, 6	(4.3, 5.7)
6	4, 5, 6, 7	(4.1, 5.4)

Tabla 2.23: Tabla de individuos agregados secuencialmente al **Grupo 2**.

- Ahora la partición inicial ha cambiado, y los dos grupos tienen las siguientes características:

Grupo	Individuo	Vector (Centroide)
Grupo 1	1, 2, 3	(1.8, 2.3)
Grupo 2	4, 5, 6, 7	(4.1, 5.4)

Tabla 2.24: Partición inicial modificada.

- Sin embargo, no podemos asegurarnos de que esas son las clasificaciones correctas para cada dato. Así que comparamos la distancia de cada individuo a su propio grupo y al grupo opuesto:

Individuo	Distancia al Grupo 1	Distancia al Grupo 2
1	1.5	5.4
2	0.4	4.3
3	2.1	1.8
4	5.7	1.8
5	3.2	0.7
6	3.8	0.6
7	2.8	1.1

Tabla 2.25: Cálculo de distancia de cada dato a su grupo y al opuesto.

- Solamente el individuo 3 está más cerca al centroide del grupo opuesto (Grupo 2) que de su propio grupo (Grupo 1). Por lo que el individuo 3 se reubica en el Grupo 2. Resultando la siguiente partición:

Grupo	Individuo	Vector (Centroide)
Grupo 1	1, 2	(1.8, 1.5)
Grupo 2	3, 4, 5, 6, 7	(3.9, 5.1)

Tabla 2.26: Partición final.

- Las iteraciones continuarían hasta que ya no haya más reubicaciones.
- Es probable que el algoritmo no encuentre una solución final.

2.8. Redes Bayesianas

Las redes Bayesianas son un tipo de modelo probabilístico gráfico que usa inferencias Bayesianas para el cómputo de probabilidades. Las redes Bayesianas pretenden modelar dependencia condicional representando esta dependencia mediante un grafo dirigido. A través de esas relaciones, se puede conducir eficientemente la inferencia de variables

aleatorias.[40]

2.9. Ambiente de análisis de datos

Para definir **ambiente de análisis de datos** primero se partirá de la definición de **ambiente de cómputo**.

Un ambiente de cómputo es una colección de computadoras que realizan procesamiento e intercambio de información para resolver diversos tipos de problemas de cómputo.[39]

Partiendo de la definición anteriormente citada, Un ambiente de análisis de datos es una colección de computadoras que tienen como fin principal ejecutar algoritmos de que realicen análisis de conjuntos de datos. *Luminus* es un ambiente de análisis de datos.

2.10. Clúster

Un clúster, en términos de informática, es un conjunto de computadoras interconectadas que pueden ser vistas como un solo sistema. Las computadoras que componen un clúster realizan una tarea en común y son controladas mediante un software específico. Usualmente los clústers son implementados con fines de obtener alto rendimiento en el procesamiento de datos.[38]

Al ser *Luminus* un **Ambiente de análisis de datos** que aplicará técnicas de análisis sobre grandes cantidades de información, se requiere implementarlo en un clúster en el que las computadoras que lo integren trabajen en conjunto para llevar a cabo la ejecución de los algoritmos implementados dentro del sistema y para el almacenamiento distribuido de los datos.

2.11. MapReduce

MapReduce es un paradigma de programación que permite escalabilidad masiva a través de cientos o miles de servidores en un clúster Hadoop. *MapReduce* se refiere a dos tareas distintas y separadas. La primera, *Map*, consiste en realizar mapeos. Ésta convierte un conjunto de datos en otro conjunto de datos diferente en el que los elementos individuales son separados en tuplas. La segunda, *Reduce*, toma los datos que arroja de *Map*, y combina esas tuplas en un conjunto más pequeño de tuplas. MapReduce es el corazón de Hadoop.[34] Los algoritmos implementados dentro de *Luminus* estarán programados con base en este paradigma.

2.12. Hadoop

Una de las capas de la pila de tecnologías que se utilizarán para construir *Luminus* es Hadoop. Hadoop un framework de software que soporta aplicaciones distribuidas bajo una licencia libre. Permite a las aplicaciones trabajar con miles de nodos y petabytes de datos. Hadoop se inspiró en los documentos Google para MapReduce y Google File System (GFS). Hadoop utiliza su propio sistema de archivos HDFS, que divide archivos grandes y los distribuye en diferentes nodos para su procesamiento.[35]

2.12.1. Características principales de Hadoop

- **Procesamiento distribuido:** Hadoop distribuye los datos en los diferentes nodos que formen parte de la arquitectura de clúster que estén haciendo uso de él. Pretende paralelizar tareas de procesamiento de datos[35].

- **Eficiencia:** Mediante la paralelización, se consigue una ganancia considerable en el tiempo de procesamiento de información.[35]
- **Económico:** Propicia un ambiente fácilmente escalable en el que resulta sencillo añadir nodos de manera horizontal conforme se vaya requiriendo.[35]
- **Código abierto:** Es un proyecto de los llamados **open source**.[35]
- **Tolerancia a fallos:** Utiliza replicación de datos haciendo uso de **HDFS (Hadoop Distributed File System)**. Si un nodo falla o cae, hay nodos de respaldo que permiten mantener el ambiente en funcionamiento.[35]

2.12.2. Arquitectura de Hadoop

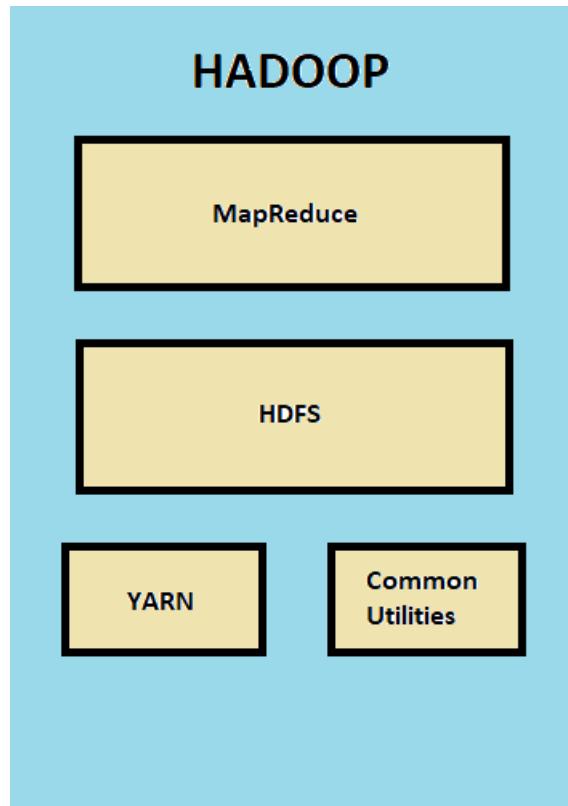


Figura 2.9: Arquitectura básica de Hadoop.

- **MapReduce:** Es el módulo que permite ejecutar cómputo distribuido. Pretende la paralelización de procesos.

- **HDFS (Hadoop Distributed File System:)** Es el sistema de archivos distribuido que utiliza Hadoop. Éste parte los datos en fragmentos y los almacena en los nodos que conforman el clúster donde Hadoop esté ejecutándose.

- **YARN:** Es el gestor de recursos de Hadoop.

- **Common Utilities:** Librerías y código necesario para ejecutar Hadoop.

2.12.3. Arquitectura de un clúster Hadoop

Habitualmente, un clúster Hadoop tiene la estructura **maestro - esclavo**. Lo que significa que hay un nodo **maestro** que estará coordinando la ejecución de tareas y las asignará a los nodos **esclavos**.

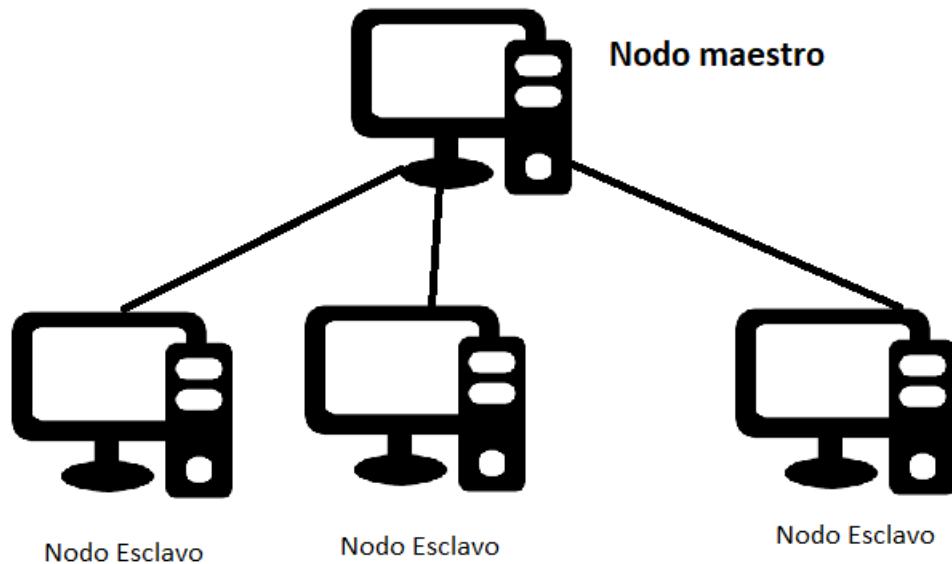


Figura 2.10: Arquitectura de maestros y esclavos.

2.12.4. Distribuciones Comerciales de Hadoop

Durante el proceso de desarrollo de *Luminus* se tenía pensado utilizar alguna distribución comercial de Hadoop, por lo que es importante denotarlas en esta sección, aún cuando al final ninguna de ellas se haya adaptado a las necesidades del proyecto. Existen varias opciones actualmente en el mercado para hacer uso de las capacidades de Hadoop, entre las cuales destacan 3 principalmente[37]:

- **Cloudera:** Fue la primera de todas las distribuciones comerciales de Hadoop. Cuenta con una herramienta llamada Cloudera Manager que permite gestionar el clúster.
- **Hortonworks:** Es la distribución de Hadoop más nueva. Permite instalar el clúster mediante un cliente de virtualización.
- **Microsoft Azure:** Es una distribución desarrollada por Microsoft que permite tener las máquinas del clúster en la nube.

2.13. Apache Spark

Es un motor de análisis de datos unificado para Big Data y Machine Learning. Que se basa en Hadoop Map Reduce. Es una herramienta de código abierto que permite **dividir y ejecutar tareas de manera paralela**. Esta cualidad de Spark de poder paralelizar el trabajo se debe a que éste software en la gran mayoría de los casos es ejecutado en sistemas con arquitectura de clúster.[36] *Luminus* utiliza Spark como motor de análisis de datos, y para la construcción de la red distribuida.

2.13.1. Características principales de Spark

- Integrado con Apache Hadoop.
- Ofrece un desempeño veloz.
- El almacenamiento de datos se administra en memoria. Se reducen mucho los tiempos de ejecución ya que hay muchas menos operaciones de lectura y escritura en disco.
- Puede ejecutar algoritmos escritos en Java, Scala, Python y R.
- Permite procesamiento en tiempo real.

2.13.2. Ventajas de usar Apache Spark

Utilizar Spark para el procesamiento de volúmenes de información de gran tamaño ofrece varios beneficios.[\[36\]](#) Los principales beneficios son los siguientes:

- **Velocidad:** Gran velocidad en el procesamiento de información. Esto se debe a lo ya mencionado anteriormente sobre la gestión de datos desde memoria.
- **Potencia:** Spark nos permite aprovechar el hardware de los equipos que lo estén utilizando.
- **Fácil uso:** A diferencia de Hadoop, que requería de un amplio conocimiento de MapReduce y de Java, Spark permite usar lenguajes de más alto nivel como Python y Scala además de Java.
- **Integración SQL:** Como resultado de contener el módulo Spark SQL, es posible realizar consultas en conjuntos de datos semi-estructurados utilizando lenguaje SQL.
- **Constante mejora del propio sistema:** Al ser un proyecto de código abierto, cada vez hay más personas que contribuyen a la mejora y ampliación de los alcances de Spark.
- **Escalabilidad:** Spark permite que incrementar el tamaño del clúster conforme se vaya necesitando.

2.13.3. Componentes de Apache Spark

Podría decirse que Spark es un conjunto de módulos que nos permiten generar conocimiento usando diferentes técnicas y tecnologías[36].

El diagrama mostrado a continuación ilustra los principales módulos o **componentes** que conforman Apache Spark.

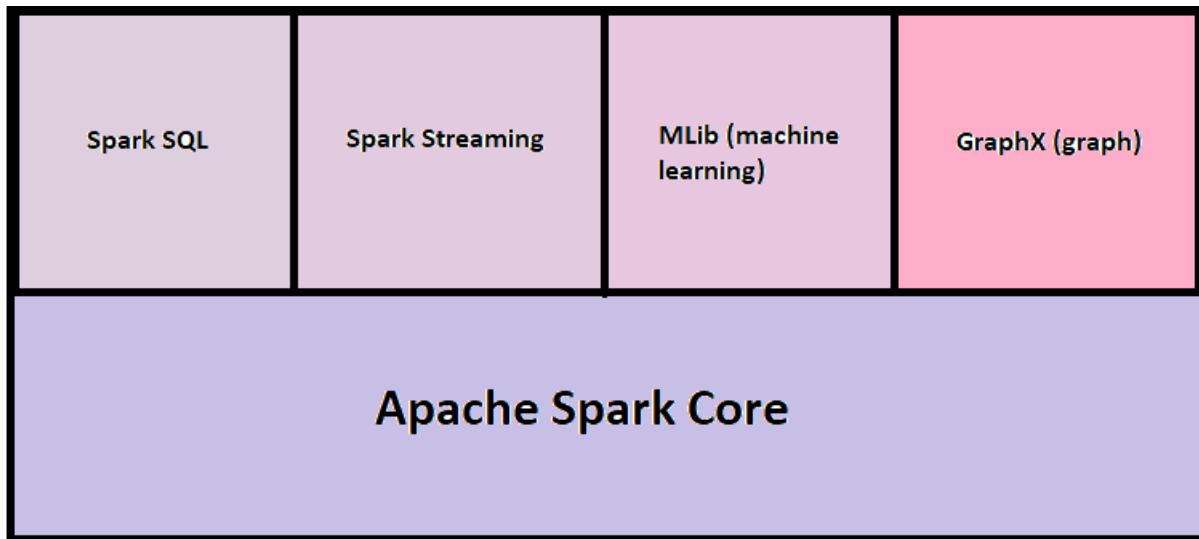


Figura 2.11: Componentes que conforman Apache Spark.

- **Spark Core:** Es el núcleo de Spark. Contiene librerías que se utilizan en todos los demás módulos.
- **Spark SQL:** Módulo para el procesamiento de datos estructurados y semi-estructurados. Esto se conoce como *SchemaRDD*. Este módulo hace posible el uso de lenguaje SQL para hacer consultas.
- **Spark Streaming:** Este módulo hace posible el procesamiento en tiempo real. Hace posible el flujo de gran cantidad de datos a alta velocidad.
- **MLlib:** Este módulo contiene herramientas y algoritmos muy variados para usar de manera fácil, práctica y escalable al *machine learning*.
- **GraphX:** Este módulo permite el procesamiento gráfico. No pinta gráficos, sino que realiza operaciones con grafos.

2.14. Interfaz Web

Una **interfaz de usuario** es el medio por el cual un usuario de un sistema computacional interactúa con el sistema. Se dice que una interfaz es **web** cuando el medio que despliega y renderiza la interfaz es un navegador web. Definir este concepto es importante ya que la interfaz de usuario de *Luminus* será una interfaz web.

2.15. Interfaz de Programación de Aplicaciones

Se trata de un módulo de software que se comunica e interactúa con otro, mediante el uso de comandos, funciones y protocolos las cuales se utilizan para diseñar e integrar software y aplicaciones.[3] El uso de este tipo de

implementaciones permite que se haga uso de cierto código fuente del cual se desconoce su funcionamiento por detrás, esto permite que la implementación de ciertas soluciones de software se simplifique y se logre ahorrar tiempo y dinero para la capa más alta de implementación, es decir, la que hace uso de la Interfaz de Programación de Aplicaciones.

El uso de una Interfaz de Programación de Aplicaciones permite que el usuario de la misma pueda concentrarse únicamente en lo que le interesa perfeccionar o adaptar para las necesidades de su aplicación y olvidarse de ciertas funcionalidades que a pesar de que las utiliza puede prescindir de su administración.

2.16. Framework

Un framework se puede considerar como una aplicación genérica incompleta y configurable a la que podemos añadirle lo que le hace falta para que pueda convertirse en una aplicación completa.^[4] Busca que el desarrollo sea más rápido, que se haga uso del mediante el uso de patrones y además busca que el código fuente pueda ser reutilizado.

Un framework es una abstracción en la que cierto código común provee una funcionalidad genérica que puede ser sobrescrita o especializada de forma selectiva por medio de código con funcionalidad específica provisto por los clientes del framework (desarrolladores de software / programadores ^[5]

2.17. Librería

Una librería representa una colección de implementaciones las cuales son utilizadas para complementar o ampliar la funcionalidad de programas de software y aplicaciones que la importan para hacer uso de ella.

^[6] Las funcionalidades de ambos se conjuntan para proporcionar una funcionalidad más grande e integrada la cual permita que ambos códigos se puedan conjuntar para permitir realizar un conjunto más grande de operaciones y con ello la funcionalidad que tenga el programa que hace uso de ella sea más amplia.

CAPÍTULO 3

Evaluación y definición de requerimientos de la red distribuida

3.1. Descripción del prototipo

El prototipo actual busca definir el número de nodos y las características que estos deben tener para crear un ambiente de Big Data que nos permita realizar las pruebas a nuestro caso de estudio de manera satisfactoria pero que tampoco escape de las capacidades de computación con las que contamos.

Por otro lado, busca definir como serán agrupados los datos que conforman el caso de estudio en los nodos que se establezcan. Para poder conocer con ello la distribución de los datos y el peso en memoria física que se le dará a cada nodo.

Se busca entonces encontrar características en común entre los datos que conforman el caso de estudio para de esta manera clasificarlos y facilitar su consulta al momento de aplicar algoritmos de minería de datos.

3.2. Análisis

3.2.1. Análisis de la red distribuida

Se pretende establecer las características que tendrá el clúster que se aplicará a la red distribuida para poder con esto, generarlo.

A continuación, se presenta el análisis de los factores que se consideran importantes para dicho objetivo:

- Se definió que es necesario realizar una réplica de los datos en los nodos ya que, si la información original no está disponible, en algún espacio en el tiempo se podrá utilizar la información de respaldo que se tiene de los mismos, con lo que se lograría que los datos sean más persistentes.
- Para poder almacenar 2 copias de la misma información la original la réplica, es necesario contar con al menos 2 nodos de datos y un nodo maestro.
- El nodo maestro no puede ser considerado nodo de datos ya que este cumple otra tarea, la cual es administrar y ordenar al resto de los nodos.
- También se definió que para manejar datos de la dimensión de 21GB (tamaño de la base de datos del caso de estudio), nodos de datos de menos de 2GB de capacidad de RAM no son eficientes e incluso llegan a tener problemas con las configuraciones de memoria que se realizan más adelante, por lo que se considera que los nodos de datos deberán trabajar con al menos 2GB de RAM.
- Para poder comunicar el clúster dentro de una red local se utilizan 2 tecnologías Spark y Hadoop además de las dependencias de estas para funcionar como son SSH y Java. Por lo que es necesario considerar el espacio que ocupa la instalación de dichas tecnologías.

3.2.2. Análisis de los datos

La base de datos que se tiene planeado utilizar como caso de estudio tiene un total de 21GB de texto plano de información de productos que se venden en tiendas departamentales en la república mexicana.

Se trata de un archivo de datos de la PROFECO de uso libre el cual tiene los siguientes campos para cada producto.

producto , presentacion , marca , categoria , catalogo , precio , fecharegistro ,
cadenacomercial , giro , nombrecomercial , direccion , estado , municipio ,
latitud , longitud

El archivo contiene toda clase de productos comerciales desde alimentos, electrónica, linea blanca, papelería, etc. Cada producto viene acompañado de la información de la tienda departamental que lo comercializa. como ejemplo del contenido de dicho archivo se muestra los siguientes registros.

crayones caja 12 ceras. jumbo. c.b. 201423 crayola
material escolar utiles escolares 27.5 2011-05-18 00:00:00
abastecedora lumen papelerias abastecedora lumen sucursal villa coapa
cannes no. 6 esq. canal de miramontes distrito federal tlalpan
19.297 -99.1254
crayones caja 12 ceras. tamano regular c.b. 201034 crayola material escolar
utiles escolares 13.9 2011-05-18 00:00:00 abastecedora lumen papelerias
abastecedora lumen sucursal villa coapa cannes no. 6 esq. canal de miramontes
distrito federal tlalpan 19.297 -99.1254
galletas populares paquete 170 gr. marias gamesa galletas pastas y
harinas de trigo mercados 6.5 2011-01-10 00:00:00 walmart
tienda de autoservicio walmart boulevard bernardo quintana no.4113
esquina camino a sa queretaro santiago de queretaro 20.6162 -100.398
galletas populares paquete 170 gr. marias gamesa galletas pastas y
harinas de trigo mercados 6.6 2011-01-10 00:00:00 farmacia guadalajara
tienda de autoservicio farmacia guadalajara sucursal 326 ignacio picazo
no.25 norte casi esquina allende ponien tlaxcala chiautempan 19.3159 -98.1945

Estos ejemplos que fueron tomados estratégicamente para ser analizados como se lista a continuación.

- El archivo contiene información de varios productos que se venden en la misma tienda, conservando entonces la parte de datos referente a la tienda, pero cambiando la parte del producto.
- El archivo contiene información del mismo producto vendido en varias tiendas departamentales en diferentes lugares, en este caso, la información del producto es la misma mientras que la información de la tienda departamental cambia.
- El archivo contiene productos similares que se venden en la misma tienda o bien en otras tiendas que no son precisamente iguales pero que se pueden comparar entre ellos.

Usando el análisis formulado anteriormente se puede proponer diferentes alternativas de agrupación de los datos. Las que se consideraron se listan a continuación.

- Precio del producto
- Categoría del producto
- Tienda departamental

- Zona Geográfica
- Estado de la república donde se encuentra el producto

Por otro lado, se determinó que el sistema de almacenamiento de datos que se va a utilizar en la red distribuida será Hadoop, y revisando su modo de operación y de manejo de archivos en su HDFS este no permite que la forma en la que se agrupan los datos sea definida por el usuario por lo que, los datos serán distribuidos siguiendo la técnica de Hadoop. Es más conveniente utilizar la forma en la que Hadoop distribuye los datos debido a que no es necesario controlar el acceso a los datos de manera manual indicando donde se encuentra cada uno de ellos, sino que Hadoop se encarga de esta tarea, además de ofrecer otros beneficios como la escalabilidad, la tolerancia a fallos, replicación en tiempo real, etc.

A pesar de esto, el análisis realizado para determinar los modos de agrupamiento más favorables no será desperdiciado pues este buscaba simplificar la operación de los algoritmos de minería de datos. y aunque no se aplique directamente sobre la distribución de los repositorios en los nodos, esta será aplicada al momento de ejecutar los algoritmos de minería de datos.

3.3. Diseño

3.3.1. Diseño de la red distribuida

Características de la red distribuida

Las características que tiene la red distribuida se enlistan a continuación: Se diseño una red distribuida que cuenta con 4 nodos en total.

Un nodo que cumple la función de nodo maestro.

Tres nodos que son utilizados como nodos de datos/replica.

Cada nodo usará la siguiente cantidad de memoria RAM:

Nodo Maestro: 6.7 GB.

Nodos Esclavos: 2 GB.

Se usa una conexión de red local para que los nodos puedan comunicarse.

Se asigno una IP estática a cada nodo para evitar problemas con la asignación dinámica de IPs mediante DHCP.

Cada nodo maestro y de datos/replica estará funcionando con base en un sistema operativo Ubuntu 16.04.

Se requiere que cada nodo de datos/replica cuente con 40 GB de almacenamiento en disco duro debido a que:

Se almacenarán 21 GB de información, y 21 GB de réplica en los nodos de información y de réplica respectivamente entre los 3 nodos.

los programas que requieren tener en ejecución los nodos para funcionar correctamente, además de el sistema operativo los cuales ocupan 1.38GB de disco duro.

El espacio reservado libre para que Hadoop haga sus cambios y modificaciones de archivos en tiempo real como este lo requiera.

El espacio en disco libre que necesita el nodo para funcionar y seguir procesando datos.

Sólo los nodos de datos/replica almacenan y procesan información

El nodo maestro administra y ordena. Cada nodo funciona con Apache Spark 2.7 y Hadoop 3.1.1.

Red distribuida en la arquitectura del sistema

El diseño de la arquitectura del sistema y la red distribuida se muestran en la figura 3.1.

Como se puede observar en la arquitectura se tiene un nodo maestro y tres nodos de datos/replica conectados a través de Hadoop.

Se explicará cómo se busca que estos trabajen en conjunto una vez que todos los prototipos se encuentren terminados y como el diseño existente hasta este momento estará afectando dicho comportamiento.

El nodo maestro será el nodo donde el usuario experto instalará el ambiente para hacer uso de big data y configura sus nodos, una vez que la instalación sea exitosa podrá ingresar sus datos, seleccionar los algoritmos que desea aplicar a

los mismos y visualizará los resultados.

Mientras el usuario solicita operaciones directamente desde el nodo maestro los nodos de datos o replica se encontrarán accediendo a los datos que se encuentran alojados en cada uno de ellos y ejecutando las operaciones que se les soliciten sobre los mismos las solicitudes antes mencionadas serán hechas en su totalidad por el nodo maestro.

Posteriormente los resultados parciales obtenidos en cada uno de los nodos serán devueltos al nodo maestro para que este pueda generar el resultado final utilizando los resultados parciales de cada uno de los nodos y mostrarlo al usuario experto.

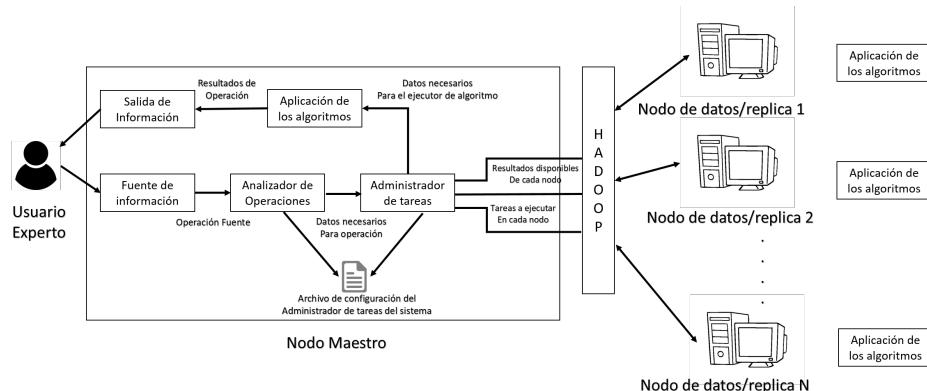


Figura 3.1: Arquitectura del sistema

3.3.2. Diseño de propuesta de agrupación de acuerdo a los nodos definidos en la red distribuida

Al usar una tecnología como Hadoop, este mismo es capaz de distribuir los datos sin que se diseñe una propuesta de agrupación por parte del programador, sino que, Hadoop establece la propia.

Hadoop dentro de su arquitectura cuenta con un bloque conocido como HDFS en el cual es un sistema de archivos distribuido y tolerante a fallos. Funciona sobre el conjunto de los nodos de un clúster de Hadoop, balanceando la carga de archivos entre las máquinas del clúster, de forma equitativa. Gracias a su naturaleza distribuida, proporciona alta disponibilidad y altas prestaciones que le permiten ser capaz de manejar archivos de gran tamaño.

Para realizar esta tarea es necesario proporcionarle el archivo de datos completo a HDFS e indicarle el número de copias que se harán de los datos, el archivo proporcionado será segmentado en bloques de 128 MB por parte de Hadoop como se muestra en la figura 3.2.



Figura 3.2: Generación de bloques de un archivo

El sistema de Hadoop ingresará cada bloque considerando este bloque como bloque de datos en algún nodo y para las réplicas ingresara nuevamente el bloque considerando este bloque como bloque de réplica 1, bloque de réplica 2 , bloque de réplica 3 y continua de la misma forma para la cantidad de bloques de réplica que se hayan indicado. Para cada una de las réplicas es indispensable que no sean asignadas en el mismo nodo que el bloque de datos o alguna otra réplica. No tiene sentido realizar más o igual número de réplicas que de nodos, debido a que las réplicas se realizan para garantizar el acceso a los bloques en todo momento, y si el bloque es accesible en un nodo y la réplica se encuentra en el mismo nunca sería utilizada.

Para ejemplificar la explicación anterior se mostrará la distribución que haría Hadoop para un sistema de 3 nodos con una réplica en la imagen 3.3.

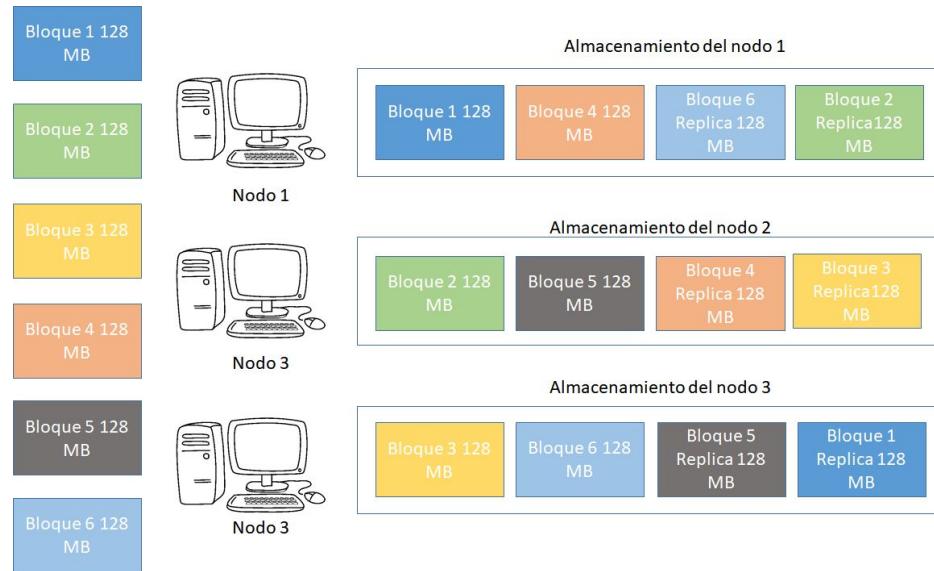


Figura 3.3: Generación de bloques de un archivo

Por lo que podemos ver que en caso de que alguno de los nodos deje de responder en los otros 2 nodos tenemos acceso a todos los datos, ya sea en el bloque de datos o en el bloque de réplica, hecho que no ocurriría en caso de que 2 nodos dejen de responder, el cual se solventa realizando 2 réplicas de los datos.

En tiempo de ejecución, para realizar su trabajo Hadoop siempre toma en cuenta primeramente los bloques de datos y en caso de no encontrarlos procede a hacer uso de las réplicas generadas de los mismos.

En el servidor maestro se guarda una información conocida como NameNode que permite conocer dónde se encuentran cada uno de los bloques y sus réplicas, lo que facilita sean encontradas dentro del clúster.

Con lo que podemos concluir que si bien no diseñamos la propuesta de agrupación de los datos comprendemos cómo es que esta funciona.

3.4. Desarrollo

Para que la red distribuida se encuentre en funcionamiento se siguió el Manual de Instalación de Luminus en sus secciones

1 Instalación de Apache Spark en el nodo

- 1.1. Instalación de la paquetería de java
- 1.2. Instalación de la paquetería de SSH
 - 1.2.1. Configuración
 - 1.2.2. Conexión
- 1.3. Instalación de Spark
 - 1.3.1. Configuración maestro

2. Instalación de Apache Spark en los nodos de datos

- 2.1. Instalación de la paquetería de java en los nodos de datos
- 2.2. Instalación de la paquetería de SSH en los nodos de datos
- 2.3. Instalación de Spark en los nodos de datos
 - 2.3.1. Configuración

3. Puesta en funcionamiento

- 3.1. Configuraciones para poner en funcionamiento Apache Spark en la red distribuida

Las cuales nos permitirán instalar Apache Spark para permitir la conexión entre los nodos de datos/replica y el maestro haciendo uso de una red de internet local en la que se encuentren conectados todos los nodos.

Además de contener todas las configuraciones necesarias para tal objetivo.

3.5. Pruebas

Para verificar que la instalación y configuración que se realizó al servidor Apache Spark es correcta será necesario aplicar un algoritmo de prueba.

Un algoritmo de prueba muy común es sparkPi el cual calcula el valor del número pi utilizando todos los nodos de la red distribuida.

Esto nos permitirá comprobar que existe conectividad dentro de la red y que se pueden realizar cálculos con ella.

El comando a ejecutar es el siguiente

```
MASTER=spark:///[ IP del nodo maestro]:7077 run-example SparkPi
```

cómo se puede apreciar en la siguiente imagen

```
root@maestro:/home/maestro# MASTER=spark://192.168.1.88:7077 run-example SparkPiUsing Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
18/10/16 17:59:23 INFO SparkContext: Running Spark version 2.1.0
18/10/16 17:59:24 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
18/10/16 17:59:24 INFO SecurityManager: Changing view acls to: root
18/10/16 17:59:24 INFO SecurityManager: Changing modify acls to: root
18/10/16 17:59:24 INFO SecurityManager: Changing view acls groups to:
18/10/16 17:59:24 INFO SecurityManager: Changing modify acls groups to:
18/10/16 17:59:24 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with view permissions: Set(root); groups with view permissions: Set()
; users with modify permissions: Set(root); groups with modify permissions: Set()
18/10/16 17:59:25 INFO Utils: Successfully started service 'sparkDriver' on port 46871.
18/10/16 17:59:25 INFO SparkEnv: Registering MapOutputTracker
18/10/16 17:59:25 INFO SparkEnv: Registering BlockManagerMaster
18/10/16 17:59:25 INFO BlockManagerMasterEndpoint: Using org.apache.spark.storage.DefaultTopologyMapper for getting topology information
18/10/16 17:59:25 INFO BlockManagerMasterEndpoint: BlockManagerMasterEndpoint up
18/10/16 17:59:25 INFO DiskBlockManager: Created local directory at /tmp/blockmgr-d7ab33be-238f-481a-a2ee-b9a030ea26f4
18/10/16 17:59:25 INFO MemoryStore: MemoryStore started with capacity 366.3 MB
18/10/16 17:59:25 INFO SparkEnv: Registering OutputCommitCoordinator
18/10/16 17:59:25 INFO Utils: Successfully started service 'SparkUI' on port 4040.
18/10/16 17:59:25 INFO SparkUI: Bound SparkUI to 0.0.0.0, and started at http://192.168.1.88:4040
18/10/16 17:59:25 INFO SparkContext: Added JAR file:/opt/spark/examples/jars/scott_2.11-
```

Figura 3.4: Ejecución de algoritmo SparkPi

cuando este termine podremos ver el resultado del valor de PI en la consola

```
90 bytes)
18/10/16 18:31:09 INFO TaskSetManager: Starting task 1.0 in stage 0.0 (TID 1, 192.168.1.88, executor 0, partition 1, PROCESS_LOCAL, 60
90 bytes)
18/10/16 18:31:10 INFO BlockManagerMasterEndpoint: Registering block manager 192.168.1.88:41081 with 366.3 MB RAM, BlockManagerId(0, 1
92.168.1.88, 41081, None)
18/10/16 18:31:10 INFO CoarseGrainedSchedulerBackend$DriverEndpoint: Registered executor NettyRpcEndpointRef(null) (192.168.1.99:57620
) with ID 2
18/10/16 18:31:10 INFO BlockManager: Added broadcast_0_piece0 in memory on 192.168.1.88:41081 (size: 1172.0 B, free: 366.3 MB)
18/10/16 18:31:11 INFO BlockManagerMasterEndpoint: Registering block manager 192.168.1.99:44847 with 413.9 MB RAM, BlockManagerId(2, 1
92.168.1.99, 44847, None)
18/10/16 18:31:11 INFO TaskSetManager: Finished task 0.0 in stage 0.0 (TID 0) in 1661 ms on 192.168.1.88 (executor 0) (1/2)
18/10/16 18:31:11 INFO CoarseGrainedSchedulerBackend$DriverEndpoint: Registered executor NettyRpcEndpointRef(null) (192.168.1.97:35414
) with ID 1
18/10/16 18:31:11 INFO TaskSetManager: Finished task 1.0 in stage 0.0 (TID 1) in 1486 ms on 192.168.1.88 (executor 0) (2/2)
18/10/16 18:31:11 INFO TaskSchedulerImpl: Removed TaskSet 0.0, whose tasks have all completed, from pool
18/10/16 18:31:11 INFO DAGScheduler: ResultStage 0 (reduce at SparkPi.scala:38) finished in 3.398 s
18/10/16 18:31:11 INFO DAGScheduler: Job 0 finished: reduce at SparkPi.scala:38, took 4.423339 s
Pi is roughly 3.141585709278546
18/10/16 18:31:11 INFO SparkUI: Stopped Spark web UI at http://192.168.1.88:4040
18/10/16 18:31:11 INFO StandaloneSchedulerBackend: Shutting down all executors
18/10/16 18:31:11 INFO CoarseGrainedSchedulerBackend$DriverEndpoint: Asking each executor to shut down
18/10/16 18:31:11 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
18/10/16 18:31:11 INFO MemoryStore: MemoryStore cleared
18/10/16 18:31:11 INFO BlockManager: BlockManager stopped
18/10/16 18:31:11 INFO BlockManagerMaster: BlockManagerMaster stopped
18/10/16 18:31:11 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!
18/10/16 18:31:11 INFO SparkContext: Successfully stopped SparkContext
18/10/16 18:31:11 INFO ShutdownHookManager: Shutdown hook called
18/10/16 18:31:11 INFO ShutdownHookManager: Deleting directory /tmp/spark-6b3719b0-b845-424f-9115-be9cb0360bad
```

Figura 3.5: Valor de PI

cabe destacar que debido a que el valor es calculado en tiempo real con los recursos que se tiene en el clúster el valor puede variar cada ejecución, pero será muy aproximado al valor real del número.

Ahora, si accedemos a la interfaz web de Spark podremos ver que un trabajo ha sido completado dentro del clúster,

antes de esta ejecución no se mostraba nada dentro de esta sección.

Spark Master at spark://maestro:7077

Spark 2.1.0

URL: spark://maestro:7077
 REST URL: spark://maestro:6066 (cluster mode)

Alive Workers: 4
 Cores in use: 7 Total, 0 Used
 Memory in use: 9.7 GB Total, 0.0 B Used
 Applications: 0 Running, 1 Completed
 Drivers: 0 Running, 0 Completed
 Status: ALIVE

Workers

Worker Id	Address	State	Cores	Memory
worker-20181016183058-192.168.1.88-46305	192.168.1.88:46305	ALIVE	4 (0 Used)	6.7 GB (0.0 B Used)
worker-20181016183059-192.168.1.97-40057	192.168.1.97:40057	ALIVE	1 (0 Used)	1024.0 MB (0.0 B Used)
worker-20181016183059-192.168.1.99-37930	192.168.1.99:37930	ALIVE	1 (0 Used)	1024.0 MB (0.0 B Used)
worker-20181016183105-192.168.1.98-33185	192.168.1.98:33185	ALIVE	1 (0 Used)	1024.0 MB (0.0 B Used)

Running Applications

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
app-20181016183104-0000	Spark Pi	7	1024.0 MB	2018/10/16 18:31:04	root	FINISHED	7 s

Completed Applications

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
app-20181016183104-0000	Spark Pi	7	1024.0 MB	2018/10/16 18:31:04	root	FINISHED	7 s

Figura 3.6: Aplicación completada en Apache Spark

En caso de entrar a los detalles de la misma podremos observar que todos los trabajadores participaron en dicha aplicación y que se encuentran en el estado muerto debido a que la aplicación a finalizado su ejecución. así como también podemos visualizar los archivos de registro que generaron durante la ejecución de esta aplicación, entre otros detalles. otro punto importante es que también se puede consultar para cada uno de los nodos su participación

Application: Spark Pi

Spark 2.1.0

ID: app-20181016183104-0000
 Name: Spark Pi
 User: root
 Cores: Unlimited (7 granted)
 Executor Limit: Unlimited (4 granted)
 Executor Memory: 1024.0 MB
 Submit Date: Tue Oct 16 18:31:04 CDT 2018
 State: FINISHED

Executor Summary

ExecutorID	Worker	Cores	Memory	State	Logs
2	worker-20181016183059-192.168.1.99-37930	1	1024	KILLED	stdout stderr
1	worker-20181016183059-192.168.1.97-40057	1	1024	KILLED	stdout stderr
3	worker-20181016183105-192.168.1.98-33185	1	1024	KILLED	stdout stderr
0	worker-20181016183058-192.168.1.88-46305	4	1024	KILLED	stdout stderr

Removed Executors

ExecutorID	Worker	Cores	Memory	State	Logs
2	worker-20181016183059-192.168.1.99-37930	1	1024	KILLED	stdout stderr
1	worker-20181016183059-192.168.1.97-40057	1	1024	KILLED	stdout stderr
3	worker-20181016183105-192.168.1.98-33185	1	1024	KILLED	stdout stderr
0	worker-20181016183058-192.168.1.88-46305	4	1024	KILLED	stdout stderr

Figura 3.7: Detalles de la ejecución de la aplicación Spark Pi

en la ejecución de esta aplicación como se muestra en la figura 3.8 para el caso del nodo maestro

Spark 2.1.0 Spark Worker at 192.168.1.88:46305

ID: worker-20181016183058-192.168.1.88-46305
Master URL: spark://maestro:7077
Cores: 4 (0 Used)
Memory: 6.7 GB (0.0 B Used)

[Back to Master](#)

Running Executors (0)

ExecutorID	Cores	State	Memory	Job Details	Logs
Finished Executors (1)					
ExecutorID	Cores	State	Memory	Job Details	Logs
0	4	KILLED	1024.0 MB	ID: app-20181016183104-0000 Name: Spark Pi User: root	stdout stderr

Figura 3.8: Detalles de la ejecución de la aplicación Spark Pi en un nodo

Con lo que podemos concluir:

- El clúster funciona correctamente
- Existe comunicación entre los nodos
- Los nodos de datos/replica son capaces de identificar al nodo maestro y recibir instrucciones de el
- El nodo maestro es capaz de comunicar trabajos a los nodos de datos/replica y de interpretar los resultados de sus trabajos de manera satisfactoria

Por lo tanto, el prototipo uno concluye de manera exitosa

CAPÍTULO 4

Integración de requerimientos de la red distribuida

4.1. Descripción del prototipo

El prototipo actual busca que sea posible integrar los datos del caso de estudio a el cluster definido en el prototipo 1 que se encuentra en el capítulo [Evaluación y definición de requerimientos de la red distribuida](#).

Una vez que los datos sean cargados a los nodos, se busca comprobar que el cluster siga funcionando correctamente y que se puedan hacer consultas sobre los datos que este almacena, comprobando con esto su accesibilidad, disponibilidad y correcta asignación de los mismos a la red.

4.2. Análisis

4.2.1. Análisis de la adaptación de los datos a la red distribuida

El archivo de datos correspondiente al caso de estudio cuenta con 21GB de texto plano. Debido a que se definió que cada nodo de datos/replica cuenta con al menos 40GB de almacenamiento.Los datos podrán ser almacenados en la red distribuida sin causar problemas de almacenamiento ya que estos son soportados por las capacidades definidas en el prototipo 1 que fueron establecidas contemplando esta condición.

Para poder adaptar los datos del caso de estudio a la red distribuida creada con anterioridad se hará uso de Apache Hadoop.

Esto debido a que este cuenta con Hadoop Distributed File System (HDFS). HDFS es un sistema de archivos distribuido y tolerante a fallos. Funciona sobre el conjunto de los nodos de un cluster de Hadoop, balanceando la carga de archivos entre las máquinas del cluster, de forma equitativa. Gracias a su naturaleza distribuida, proporciona alta disponibilidad y altas prestaciones que le permiten ser capaz de manejar archivos de gran tamaño.

por lo que, haciendo uso de este sistema de manejo de archivos y teniendo las capacidades de almacenamiento en los nodos de la red distribuida es posible afirmar que se puede adaptar sin complicaciones el archivo de datos del caso de estudio.

4.3. Diseño

4.3.1. Diseño de la red distribuida con nodos de datos

La red distribuida cuenta con 3 nodos de datos/replica y un nodo maestro, una vez que se apliquen las configuraciones correspondientes a la asignación de los datos en los nodos de datos/replica así como un algoritmo para probar que los datos asignados son accesibles, la red distribuida deberá verse como la que se muestra en la imagen [Diseño de la red distribuida con nodos de datos](#)

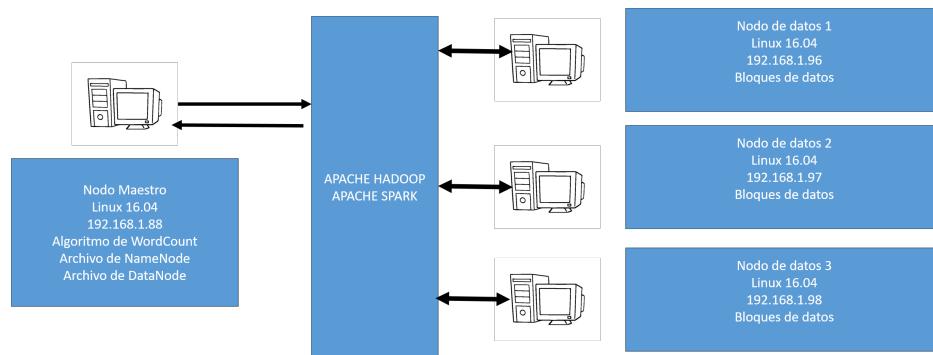


Figura 4.1: Diseño de la red distribuida con nodos de datos

la cual funciona conectando el cluster a una red local que les de una IP a cada uno de ellos la cual será definida como estática y se utilizará para las conexiones de manera distribuida tanto de Apache Hadoop como también de Apache Spark.

Los nodos de datos/replica no se comunicarán entre si al momento de ejecutar operaciones de computo en la red distribuida y para tal motivo utilizarán la conexión con el nodo maestro.

4.3.2. Diseño de pruebas de obtención de información

Para poder conocer si los datos fueron distribuidos correctamente y el cluster se encuentra en funcionamiento es posible utilizar un algoritmo que use los datos que se encuentran en el cluster y al final regrese un resultado.

Los algoritmos de big data que serán necesarios para este trabajo son algoritmos que utilizan como base una tecnología llamada Map Reduce.

Se encontró que existen algoritmos de prueba dentro de Hadoop que pueden ser utilizados para comprobar el correcto funcionamiento de la red. sin embargo, se busca comprender como es que estas pruebas funcionan.

Y a su vez buscar que el ejemplo que sea aplicado sobre los datos del caso de estudio se adapte mejor a estos y ofrezca resultados mas interesantes.

WordCount tradicional

La prueba que se va a realizar es el Contador de palabras.^º "WordCount." este algoritmo lo que hace es contar todas las palabras que se encuentran dentro de un archivo y decir cuantas coincidencias de la misma palabra se encontraron. El algoritmo funciona de la siguiente manera:

- Cada que encuentra una nueva palabra la agrega al listado de palabras con el valor de 1 que significa que solo ha sido encontrada una vez.
- En caso de que la palabra sea encontrada nuevamente, entonces se reemplazara el número asociado a esa palabra por el número que tenia en ese momento + 1.
- Termina cuando llega al final del archivo y por lo tanto todas las palabras nuevas fueron registradas y se conoce cuantas veces aparecen en el archivo.

El procedimiento descrito anteriormente se explicará de igual forma con el diagrama de flujo que se muestra en la Imagen [Diagrama de flujo del algoritmo contador de palabras](#).

Los pasos descritos anteriormente, son los pasos que serían utilizados si se tratara de un algoritmo que se ejecuta sin el uso de Map Reduce.

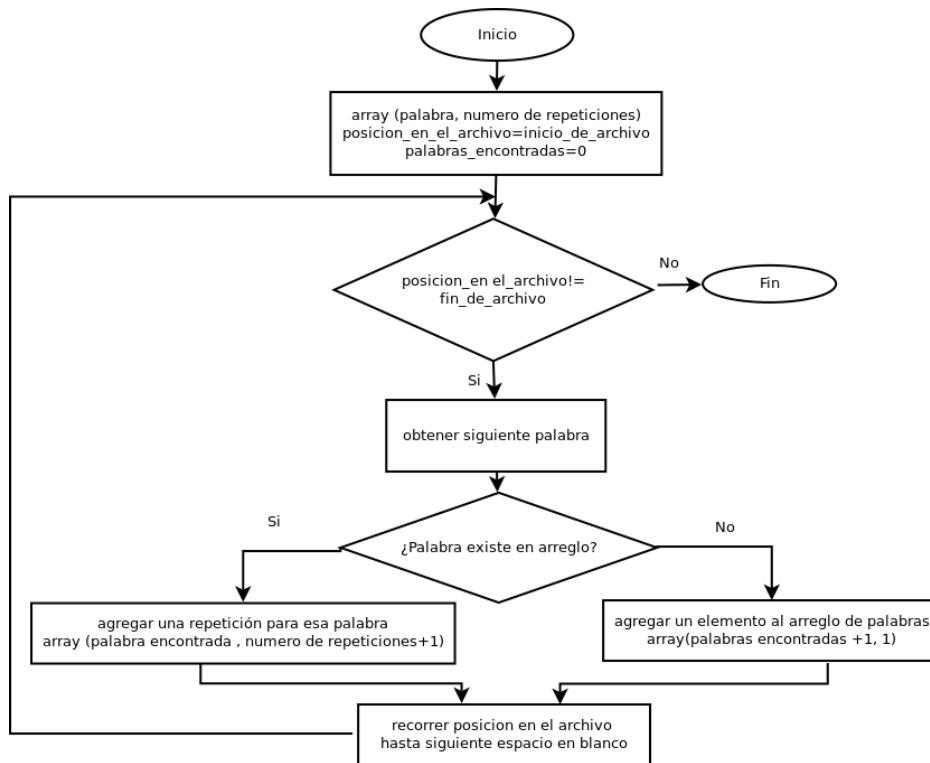


Figura 4.2: Diagrama de flujo del algoritmo contador de palabras

WordCount con el uso de Map Reduce

Map Reduce es una técnica que descompone un trabajo grande en tareas individuales las cuales pueden ser ejecutadas por separado en diferentes computadoras que componen un cluster, y estas al final pueden unir sus resultados individuales para calcular los resultados finales.

Función Map: Toma un conjunto de datos y los convierte en otro conjunto de datos, donde los elementos individuales se dividen en entradas del nuevo conjunto con la estructura (clave-valor).

de tal forma que lo que una función Map en este ejemplo haría sería convertir el conjunto de palabras en un conjunto diferente donde: Clave: toma el valor de la palabra que se encontró en el archivo Valor: asigna el valor de 1 a otras las palabras encontradas en el archivo. Veamos el siguiente conjunto de entrada dentro del archivo:

botella , refresco , Botella , BOTELLA, Fresco , Refresco , botellarefresco .

para este archivo el conjunto de salida de la función map, seria:

(botella ,1) , (refresco ,1) , (Botella ,1) , (BOTELLA,1) ,
(Fresco ,1) ,(Refresco ,1) , (botellarefresco ,1) .

Función Reduce: toma la salida del mapa y combina las entradas para generar un conjunto mas pequeño de datos De tal forma que lo que la función reduce haría en este ejemplo sería buscar las palabras que sean las mismas de las entregadas por cada nodo y agruparlas. veamos el mismo ejemplo, suponiendo que el trabajo fue realizado por 3 nodos de datos, veamos lo que haría la función reduce.

nodo1
(botella ,1) , (refresco ,1) .
nodo2
(Botella ,1) , (BOTELLA,1) .
nodo3
(Fresco ,1) ,(Refresco ,1) , (botellarefresco ,1) .

para estas entradas, el conjunto de salida de la función reduce seria:

(BOTELLA,3) , (REFresco,3) , (BOTELLAREFRESCO ,1) .

Sin embargo, cabe destacar que el sistema de Hadoop no solo trabaja con las funciones map y reduce, sino que tiene otras funciones internas que el sistema controla.

En la imagen [Funcionamiento completo de Hadoop, con todas sus operaciones](#) se puede visualizar el funcionamiento completo que tendría que ejecutarse en el cluster para llevar a cabo este algoritmo. Teniendo como referencia la imagen

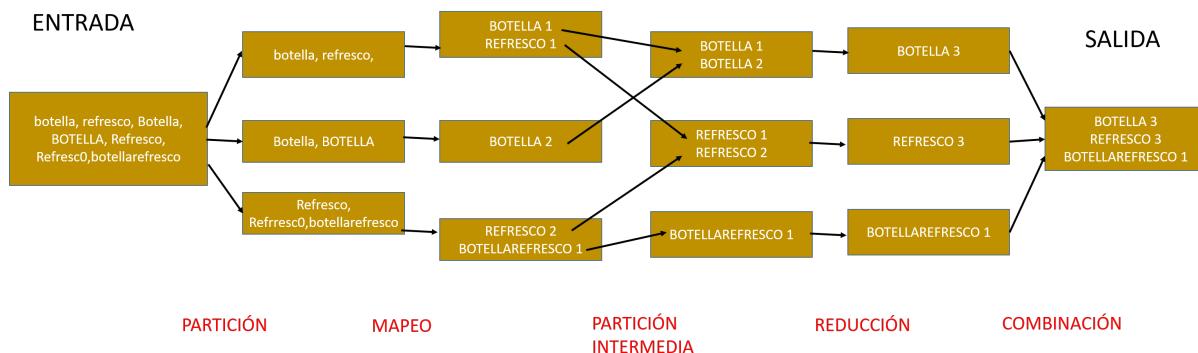


Figura 4.3: Funcionamiento completo de Hadoop, con todas sus operaciones

[Funcionamiento completo de Hadoop, con todas sus operaciones](#) se explicarán los módulos que ejecuta Hadoop de forma interna para una aplicación Map Reduce de forma breve.

- Partición:** Se requiere definir un parámetro de partición, esto para que se puedan reconocer las unidades a analizar y con esto poder asignar tareas a los nodos. el parámetro de partición puede ser cualquier cosa, por ejemplo, dividir por espacio, coma, punto y coma, o incluso por una nueva línea ('n').
- Mapeo:** Se explico anteriormente en [WordCount con el uso de Map Reduce Función Map](#).
- Partición intermedia:** Se busca generar grupos con los datos que después de pasar por el modulo de mapeo y tener la estructura de salida de este modulo tienen la misma CLAVE, al cumplir esta condición se asignan en el mismo grupo. Se generan tantos grupos como claves existan.
- Reducción:** Se explico anteriormente en [WordCount con el uso de Map Reduce Función Reduce](#).
- Combinación:** Todos los datos de salida que arroja la función de reducción son combinados y puestos en un solo grupo para generar un resultado final.

4.4. Desarrollo

Para que Instalar Apache Hadoop en la red distribuida en funcionamiento se siguió el Manual de Instalación de Luminus en sus secciones

- Instalación de Apache Hadoop en el nodo maestro
 - Instalación de Hadoop
 - Configuración
 - Archivos de configuración
- Instalación de Apache Hadoop en los nodos de datos
 - Instalación de Hadoop en los nodos de datos
 - Configuración
- Puesta en funcionamiento
 - Puesta en funcionamiento del Cluster manejado por el Servidor Apache Hadoop

Las cuales nos permitirán instalar Apache Hadoop el cual tendrá al mismo tiempo todas las configuraciones necesarias requeridas para este proyecto, tanto en el nodo maestro como también en los nodos de datos/replica. Dentro del manual de instalación se explica para cada uno de los archivos de configuración de hadoop como es que estos se configuran y cual es el objetivo de cada configuración de manera detallada. Una vez que las configuraciones se hayan ejecutado de manera correcta y el cluster se encuentre en funcionamiento haciendo uso de hadoop, se subirá un archivo a la plataforma, esto se hará siguiendo el manual de instalación de luminus en la sección

6.2. Subir un archivo al HDFS

En el momento de subir el archivo al HDFS se puede comprobar que la red distribuida permite realizar esta operación para lo cual es indispensable comunicarse con todos los nodos que se encuentran dentro de la misma. Esto con el objetivo de realizar el almacenamiento de manera distribuida haciendo uso de todos los nodos de datos/replica. Por lo tanto se sabe que la red distribuida tiene conectividad y se encuentra funcionando de manera apropiada.

4.4.1. Algoritmo de prueba

El siguiente algoritmo escrito en JAVA servirá para comprobar que se pueden realizar operaciones sobre los datos que se encuentran en el cluster.

Como se explico en la sección [Diseño de pruebas de obtención de información](#) de este capítulo, se utilizará el algoritmo wordcount para cumplir con este objetivo. También, se menciono que las únicas secciones que se requiere programar para ejecutar un algoritmo de tipo map reduce es la función map y la función reduce. ya que hadoop se encarga de el resto de funciones.

Por otro lado se explico la forma en que este algoritmo en particular funcionar y un ejemplo de su funcionamiento en la imagen [Funcionamiento completo de Hadoop, con todas sus operaciones](#).

Se decidió que se considere que se encontró una palabra cada que aparezca una "," dentro del archivo como parámetro de la función de partición, esto debido a que cada atributo de la tabla de datos esta separado por una coma y esto nos permitiría comprobar cuantas veces aparece un atributo dentro del archivo en lugar de solamente conocer la repetición de las palabras por separado. Lo cual esta orientado para el archivo de datos del caso de estudio y nos puede dar información de la frecuencia con la que ciertos registros aparecen dentro del mismo.

El código JAVA para tal objetivo se puede visualizar en el [Anexo A: Código JAVA Map-Reduce](#) dentro del cual se pueden destacar algunas observaciones

- Se importan las librerías de Apache Hadoop para que pueda reconocerse que se trata de un programa Map Reduce que se ejecutará sobre este programa.
- Se tiene la clase main la cual principalmente manda a llamar a las clases map y reduce, establece los valores por defecto para empezar a contar, Establece los canales de lectura y escritura para que este pueda acceder a los archivos del cluster a analizar y pueda tambien escribir sus resultados.
- Se tiene la clase map la cual toma la palabra encontrada y le da la estructura (CLAVE, VALOR) , además de pasarlala a mayúsculas para que considere que se trata de la misma palabra cuando tenga las mismas letras sin importar si originalmente estaba escrita en mayúsculas, minúsculas o una combinación de ambas.
- La clase reduce que cuenta cuantas veces se repiten en total las palabras en el grupo que se genero de todas las palabras iguales que se encontraron en cada uno de los nodos. para sacar una suma total para cada palabra del archivo.

Este algoritmo tiene que ser compilado ya sea con un IDE de java o directamente desde Hadoop para generar el jar que posteriormente sera ejecutado por Hadoop.

El código desde consola para generar este JAR es el siguiente.

```
bin/hadoop com.sun.tools.javac.Main [clase_principal_java].java  
jar cf [Nombre_jar].jar [clase_principal_java]*.class
```

4.5. Pruebas

Una vez que se tiene el .JAR a ejecutar se puede hacer uso del siguiente comando para que este entre en funcionamiento dentro de la red distribuida.

```
yarn jar <ruta/a/archivo.jar> <NombreDeClase> <Parametros>
```

Para nuestro ejemplo específico el comando seria:

```
root@maestro:/opt/hadoop/etc/hadoop: yarn jar /home/mayra/Escritorio/MRProgramsDemo.jar  
PackageDemo.WordCount "user/root/productos/inserts.txt" output6
```

output6 sera una carpeta que se creará en el sistema de archivos de HDFS para almacenar los resultados de salida. Esta carpeta puede tener el nombre que se desee y se asigna en esta sección, sin embargo, no puede existir dentro del sistema de archivos al momento de ejecutar este comando. La carpeta se creará en el directorio /user/root/nombreasignado Una vez que se ejecute esta instrucción se procederá a revisar las conexiones con los nodos y hacer los ajustes necesarios para comenzar a ejecutar este algoritmo. esta información de salida y el comienzo de la ejecución del algoritmo con la parte de la función map pueden verse en la imagen [Ejecución de algoritmo map reduce para contar palabras: Comenzando el proceso](#) este algoritmo. esto se hará para todos los porcentajes de la función map, una vez que estos finalicen, comenzará a ejecutar la función reduce y con ella sus porcentajes de progreso como se muestra en la imagen [Ejecución de algoritmo map reduce para contar palabras: Inicio de funcion reduce](#).

```

nodo3:38667          RUNNING      nodo3:8042
          0
nodo2:44891          RUNNING      nodo2:8042
          0
root@maestro:/opt/hadoop/etc/hadoop# yarn jar /home/maestro/Escritorio/MRProgramsDemo.jar PackageDemo.WordCount /user/luminus/datosproductos/all_data.csv /user/root/output6
2018-10-18 03:20:28,566 INFO client.RMProxy: Connecting to ResourceManager at maestro/192.168.1.88:8032
2018-10-18 03:20:29,501 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/root/.staging/job_1539850756835_0001
2018-10-18 03:20:32,024 INFO input.FileInputFormat: Total input files to process : 1
2018-10-18 03:20:33,574 INFO mapreduce.JobSubmitter: number of splits:154
2018-10-18 03:20:33,791 INFO Configuration.deprecation: yarn.resourcemanager.system-metrics-publisher.enabled is deprecated. Instead, use yarn.system-metrics-publisher.enabled
2018-10-18 03:20:34,547 INFO mapreduce.JobSubmitter: Submitting tokens for job : job_1539850756835_0001
2018-10-18 03:20:34,554 INFO mapreduce.JobSubmitter: Executing with tokens: []
2018-10-18 03:20:34,917 INFO conf.Configuration: resource-types.xml not found
2018-10-18 03:20:34,918 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2018-10-18 03:20:35,281 INFO impl.YarnClientImpl: Submitted application application_1539850756835_0001
2018-10-18 03:20:35,328 INFO mapreduce.Job: The url to track the job: http://maestro:8088/proxy/application_1539850756835_0001/
2018-10-18 03:20:35,329 INFO mapreduce.Job: Running job: job_1539850756835_0001
2018-10-18 03:20:50,683 INFO mapreduce.Job: Job job_1539850756835_0001 running in uber mode : false
2018-10-18 03:20:50,694 INFO mapreduce.Job: map 0% reduce 0%
2018-10-18 03:21:21,278 INFO mapreduce.Job: map 1% reduce 0%
2018-10-18 03:21:27,533 INFO mapreduce.Job: map 2% reduce 0%
2018-10-18 03:21:50,770 INFO mapreduce.Job: map 3% reduce 0%
2018-10-18 03:22:20,359 INFO mapreduce.Job: map 4% reduce 0%
2018-10-18 03:22:40,513 INFO mapreduce.Job: map 5% reduce 0%
2018-10-18 03:22:53,605 INFO mapreduce.Job: map 6% reduce 0%
2018-10-18 03:23:50,094 INFO mapreduce.Job: map 7% reduce 0%
2018-10-18 03:23:56,136 INFO mapreduce.Job: map 8% reduce 0%
2018-10-18 03:24:33,320 INFO mapreduce.Job: map 9% reduce 0%
2018-10-18 03:24:52,391 INFO mapreduce.Job: map 10% reduce 0%

```

Figura 4.4: Ejecución de algoritmo map reduce para contar palabras: Comenzando el proceso

```
2018-10-18 04:42:43,728 INFO mapreduce.Job: map 92% reduce 0%
2018-10-18 04:43:05,769 INFO mapreduce.Job: map 93% reduce 0%
2018-10-18 04:43:40,041 INFO mapreduce.Job: map 94% reduce 0%
2018-10-18 04:43:50,060 INFO mapreduce.Job: map 95% reduce 0%
2018-10-18 04:44:43,179 INFO mapreduce.Job: map 96% reduce 0%
2018-10-18 04:44:50,190 INFO mapreduce.Job: map 97% reduce 0%
2018-10-18 04:45:32,416 INFO mapreduce.Job: map 98% reduce 0%
2018-10-18 04:46:29,550 INFO mapreduce.Job: map 99% reduce 0%
2018-10-18 04:46:35,560 INFO mapreduce.Job: map 100% reduce 0%
2018-10-18 04:47:37,928 INFO mapreduce.Job: map 100% reduce 1%
2018-10-18 04:47:50,952 INFO mapreduce.Job: map 100% reduce 2%
2018-10-18 04:48:24,094 INFO mapreduce.Job: map 100% reduce 3%
2018-10-18 04:49:11,337 INFO mapreduce.Job: map 100% reduce 4%
2018-10-18 04:49:35,550 INFO mapreduce.Job: map 100% reduce 5%
2018-10-18 04:50:24,729 INFO mapreduce.Job: map 100% reduce 6%
2018-10-18 04:50:54,842 INFO mapreduce.Job: map 100% reduce 7%
2018-10-18 04:51:43,186 INFO mapreduce.Job: map 100% reduce 8%
2018-10-18 04:52:19,329 INFO mapreduce.Job: map 100% reduce 9%
2018-10-18 04:53:01,520 INFO mapreduce.Job: map 100% reduce 10%
2018-10-18 04:53:46,821 INFO mapreduce.Job: map 100% reduce 11%
2018-10-18 04:54:48,095 INFO mapreduce.Job: map 100% reduce 12%
2018-10-18 04:55:19,227 INFO mapreduce.Job: map 100% reduce 13%
2018-10-18 04:56:33,635 INFO mapreduce.Job: map 100% reduce 14%
2018-10-18 04:57:03,730 INFO mapreduce.Job: map 100% reduce 15%
2018-10-18 04:58:16,120 INFO mapreduce.Job: map 100% reduce 16%
2018-10-18 04:58:58,266 INFO mapreduce.Job: map 100% reduce 17%
2018-10-18 04:59:52,628 INFO mapreduce.Job: map 100% reduce 18%
2018-10-18 05:00:40,852 INFO mapreduce.Job: map 100% reduce 19%
2018-10-18 05:01:41,202 INFO mapreduce.Job: map 100% reduce 20%
2018-10-18 05:02:24,356 INFO mapreduce.Job: map 100% reduce 21%
2018-10-18 05:03:06,483 INFO mapreduce.Job: map 100% reduce 22%
2018-10-18 05:03:50,788 INFO mapreduce.Job: map 100% reduce 23%
2018-10-18 05:04:32,926 INFO mapreduce.Job: map 100% reduce 24%
2018-10-18 05:05:39,345 INFO mapreduce.Job: map 100% reduce 25%
2018-10-18 05:06:21,460 INFO mapreduce.Job: map 100% reduce 26%
2018-10-18 05:07:15,633 INFO mapreduce.Job: map 100% reduce 27%
2018-10-18 05:08:16,934 INFO mapreduce.Job: map 100% reduce 28%
2018-10-18 05:08:47,024 INFO mapreduce.Job: map 100% reduce 29%
2018-10-18 05:09:53,429 INFO mapreduce.Job: map 100% reduce 30%
2018-10-18 05:10:41,565 INFO mapreduce.Job: map 100% reduce 31%
```

Figura 4.5: Ejecución de algoritmo map reduce para contar palabras: Inicio de función reduce

Cuando se complete la función reduce procederá a notificar que el algoritmo fue ejecutado correctamente y mostrará estadísticas de:

- File System Counter
- Job Counter
- MapReduce Framework

Dichas estadísticas y resultados finales para este algoritmo se muestran en las figuras [Ejecución de algoritmo map reduce para contar palabras: Termina la ejecución del algoritmo](#) y [Ejecución de algoritmo map reduce para contar palabras: Estadísticas](#)

```
2018-10-18 05:42:11,402 INFO mapreduce.Job: map 100% reduce 99%
2018-10-18 05:42:29,432 INFO mapreduce.Job: map 100% reduce 100%
2018-10-18 05:43:25,002 INFO mapreduce.Job: Job job_1539850756835_0001 completed successfully
2018-10-18 05:43:27,429 INFO mapreduce.Job: Counters: 57
    File System Counters
        FILE: Number of bytes read=77218820448
        FILE: Number of bytes written=103931282332
        FILE: Number of read operations=0
        FILE: Number of large read operations=0
        FILE: Number of write operations=0
        HDFS: Number of bytes read=20664348064
        HDFS: Number of bytes written=212338004
        HDFS: Number of read operations=467
        HDFS: Number of large read operations=0
        HDFS: Number of write operations=2
    Job Counters
        Killed map tasks=23
        Killed reduce tasks=1
        Launched map tasks=176
        Launched reduce tasks=2
        Other local map tasks=21
        Data-local map tasks=154
        Rack-local map tasks=1
        Total time spent by all maps in occupied slots (ms)=16923807
        Total time spent by all reduces in occupied slots (ms)=1447493
0
        Total time spent by all map tasks (ms)=16923807
        Total time spent by all reduce tasks (ms)=7237465
        Total vcore-milliseconds taken by all map tasks=16923807
        Total vcore-milliseconds taken by all reduce tasks=7237465
        Total megabyte-milliseconds taken by all map tasks=11542036374
        Total megabyte-milliseconds taken by all reduce tasks=98719022
60
    Map-Reduce Framework
        Map input records=62530716
        Map output records=1015403483
        Map output bytes=24648238567
        Map output materialized bytes=26679114886
        Input split bytes=19250
        Combine input records=0
        Combine output records=0
```

Figura 4.6: Ejecución de algoritmo map reduce para contar palabras: Termina la ejecución del algoritmo

```

50
Map-Reduce Framework
  Map input records=62530716
  Map output records=1015403483
  Map output bytes=24648238567
  Map output materialized bytes=26679114886
  Input split bytes=19250
  Combine input records=0
  Combine output records=0
  Reduce input groups=7676839
  Reduce shuffle bytes=26679114886
  Reduce input records=1015403483
  Reduce output records=7676839
  Spilled Records=3946856040
  Shuffled Maps =154
  Failed Shuffles=0
  Merged Map outputs=154
  GC time elapsed (ms)=135866
  CPU time spent (ms)=5140450
  Physical memory (bytes) snapshot=63333994496
  Virtual memory (bytes) snapshot=358570815488
  Total committed heap usage (bytes)=50046279680
  Peak Map Physical memory (bytes)=435355648
  Peak Map Virtual memory (bytes)=2311077888
  Peak Reduce Physical memory (bytes)=1119645696
  Peak Reduce Virtual memory (bytes)=2891509760
Shuffle Errors
  BAD_ID=0

```

Figura 4.7: Ejecución de algoritmo map reduce para contar palabras: Estadísticas

Ahora, para poder visualizar el resultado de este algoritmo se puede hacer un listado de los archivos que contiene el directorio de salida que se creo, para ello se utiliza el comando siguiente

```
root@maestro:/opt/hadoop/etc/hadoop# hdfs dfs -ls /user/root/output6
```

Con lo cual se genera la salida que se despliega en la imagen 4.8 la cual muestra que la operación fue exitosa y las dimensiones generadas para el archivo de salida.

```

root@maestro:/opt/hadoop/etc/hadoop# hdfs dfs -ls /user/root/output6
Found 2 items
-rw-r--r--  2 root supergroup          0 2018-10-18 05:42 /user/root/output6/_SUCCESS
-rw-r--r--  2 root supergroup  212338004 2018-10-18 05:42 /user/root/output6/part-r-00000

```

Figura 4.8: Contenido del directorio de salida

Debido a que se trata de un archivo muy grande, para efectos de simplificación solo se mostrará en este documento un fragmento de dicho archivo que deje ver el trabajo que fue realizado dicho fragmento se muestra en la imagen 4.9. Dentro de esta imagen podemos ver por ejemplo, cuantas veces aparecen determinados precios dentro del archivo, direcciones o bien descripciones de productos, entre otros.

Por lo cual se puede ver que el archivo fue estudiado en su totalidad y que se tienen coincidencias para diferentes entradas que se encontraban dentro del archivo.

Un resultado interesante obtenido que se puede observar en la imagen 4.10 es que, al menos para este archivo la fecha de registro es irrelevante pues para cada articulo se tiene una diferente o bien comparten una misma fecha cuando la hora establecida es 0hrs para una gran cantidad de registros que fueron dados de alta el mismo día, pero en realidad

part-r-00000 (~/Vídeos) - gedit

Abrir ▾ Guardar

```
"1996.65"      20
"19963/ 19934 ASST. MARVEL. SUPER HERO SQUAD. QUINJET"  11
"1997.00"      10
"19977. BEYBLADE. MOBILE BEYSTADIUM"      25
"19979.10"     2
"1998.00"      152
"19980. BEYBLADE. SUPER VORTEX BATTLE SET"      8
"19987. TONKA CHUCK & FRIENDS. ROWDY EL CAMION DE BASURA"      171
"19988.00"     2
"1999.00"      1272
"1999.20"      66
"1999.50"      6
"19990.00"     78
"19998.00"     22
"19998.30"     2
"19999.00"     88
"19999.20"     90
"10. DE MAYO NO. 2 20 DE NOVIEMBRE ESQ. PORFIRIO DIAZ"  24
"10. DE MAYO 200      38266
"10. DE MAYO MZ-C 24-B ESQ. TENANGO DEL VALLE      59318
"10. DE MAYO NO. 18 ESQUINA ALLENDE      30
"10. DE MAYO NO. 18 ESQUINA ALLENDE"      2295
"2 CAJAS CON 14 CAPSULAS DE 20 MG. C/U" 30747
"2 CAJAS CON 30 TABLETAS C/U DE 10 MG." 28985
"2 CAJAS CON 30 TABLETAS C/U DE 20 MG." 28407
"2 CAJAS CON 8 COMPRIMIDOS C/U DE 400 MG.-100 MG."      30791
"2 FCOS AMP DE 20 MG."  11
"2 ORIENTE NO. 225      122
"2 ORIENTE NO. 225"      548
"2 PLIEGOS"      1225
"2 PLIEGOS. CARTULINA"  3771
"2 PONIENTE 704 3
"2 PTE. NO. 704 187
"2.00"      248
"2.03"      2
"2.05"      18
```

Figura 4.9: Contenido del archivo de resultados

no proporciona información real de los artículos y genera mucho ruido en el archivo.

Existen más entradas de fechas que información de los productos, comportamiento que dificultaría el análisis de los datos en un futuro por lo cual este atributo será retirado del archivo correspondiente al caso de estudio. Con lo que

"2015-09-23 17:30:10.397"	1
"2015-09-23 17:30:12.187"	1
"2015-09-23 17:30:15.047"	1
"2015-09-24 00:00:00.000"	87671
"2015-09-25 00:00:00.000"	73318
"2015-09-25 09:36:52.837"	1
"2015-09-25 12:49:06.920"	1
"2015-09-25 12:49:07.767"	1
"2015-09-25 12:49:14.497"	1
"2015-09-25 12:49:28.077"	1
"2015-09-25 12:49:29.027"	1
"2015-09-25 12:49:52.087"	1
"2015-09-25 12:50:07.230"	1
"2015-09-25 12:51:17.713"	1
"2015-09-25 12:51:22.007"	1
"2015-09-25 12:51:23.080"	1
"2015-09-25 12:51:26.690"	1
"2015-09-25 12:52:03.880"	1
"2015-09-25 12:52:28.507"	1
"2015-09-25 12:52:29.143"	1
"2015-09-25 12:53:02.143"	1
"2015-09-25 13:15:48.707"	1
"2015-09-25 13:45:30.567"	1
"2015-09-25 14:48:43.067"	1
"2015-09-25 14:50:48.520"	1
"2015-09-25 14:50:49.510"	1
"2015-09-25 14:50:50.840"	1
"2015-09-25 15:00:21.033"	1
"2015-09-25 15:03:53.300"	1
"2015-09-25 15:47:18.000"	1

Figura 4.10: Fechas de registro en el archivo de resultados

podemos concluir:

- El cluster funciona correctamente
- Existe comunicación entre los nodos
- Los nodos de datos/replica son capaces de identificar al nodo maestro y recibir instrucciones de él
- El nodo maestro es capaz de comunicar trabajos a los nodos de datos/replica y de interpretar los resultados de sus trabajos de manera satisfactoria
- Se tiene el archivo del caso de estudio almacenado de manera distribuida en los nodos
- El archivo del caso de estudio es accesible y se pueden ejecutar operaciones sobre él

Por lo tanto, el prototipo dos concluye de manera exitosa

CAPÍTULO 5

Algoritmos de Minería de Datos

5.1. Descripción del prototipo

En este prototipo, de acuerdo al alcance definido en el cronograma realizado durante la definición del proyecto se planteó hacer el diseño del conjunto de interfaces necesarias que permitieran presentar los resultados del trabajo final haciendo uso de un sitio web.

Por otro lado se estableció que en este prototipo, teniendo una vez configurado el ambiente de Big Data en su totalidad, sería posible implementar los algoritmos de minería de datos que se utilizarían para la operación de este sistema.

De acuerdo a la definición acordada en fases anteriores del proyecto, se seleccionaron 2 algoritmos diferentes: KNN e ID3 para ser desarrollados e implementados en este prototipo.

5.2. Análisis

5.2.1. Análisis del flujo de datos en las pantallas del sistema

En ese momento aun se mantenía la visión de que los resultados de la ejecución de los algoritmos de minería de datos serían reflejados mediante el uso de las pantallas en un sitio web, por lo que se analizó lo que los algoritmos necesitaban para funcionar, y cómo es que estas necesidades ya identificadas podrían ser presentadas e integradas como parte del sitio web.

Con este análisis se propuso un mapa de navegación en el sitio web que pudiera solventar esta necesidad.

Posteriormente se diseñó cada una de las pantallas que se buscaba incluir en el mapa de navegación, sin embargo, al cambiar la forma de presentar los resultados finales se dejó este trabajo de lado.

A pesar de ello, y debido a que este análisis sí fue realizado, se puede encontrar detalladamente en la sección [Anexo C: Una manera de presentar a Luminus de forma diferente: Sitio web](#). Donde se muestra el detalle del mapa de navegación generado, así como las pantallas diseñadas y su propuesta de funcionamiento.

En adelante, en esta sección no se hablará mas de este trabajo ya que no forma parte de la solución final, por lo que en el resto de las secciones de este capítulo solamente se hará énfasis en cómo se llevó a cabo el desarrollo de los algoritmos de minería de datos. Mientras que el detalle del funcionamiento de la interface de programación de aplicaciones de Luminus se encuentra en la sección [Interface de programación de aplicaciones de Luminus](#)

5.2.2. Análisis de los algoritmos de minería de datos

Los algoritmos KNN e ID3 fueron seleccionados para ser implementados ya que se trata de algoritmos comúnmente utilizados para el análisis de grandes volúmenes de datos y son algoritmos conocidos en la industria. Estos algoritmos ya no se encuentran en fase experimental y sus resultados ya han sido comprobados en múltiples ocasiones. Por lo que se sabe que los algoritmos son efectivos y pueden proporcionar resultados valiosos para las empresas que hace uso de

ellos.

Se considera que para proporcionar una muestra de los algoritmos que pueden ser soportados por este paradigma de programación y por esta arquitectura de análisis de datos, los algoritmos seleccionados son una buena opción, además de que se trata de algoritmos son comúnmente utilizados por los usuarios expertos que actualmente hacen uso de Big Data.

Otra particularidad de esta selección es que se trata de 2 tipos de algoritmos distintos:

- Algoritmo de clasificación
- Algoritmo de árboles de decisión.

La teoría del funcionamiento de los algoritmos seleccionados puede ser consultada en las secciones [ID3 \(Iterative Dichotomiser 3\)](#) y [Algoritmo KNN \(K-Nearest Neighbors\)](#) del marco teórico. Esta es explicada de manera tradicional, mostrándose para cada uno de estos algoritmos un ejemplo a resolver haciendo uso del algoritmo de manera teórica. Sin embargo, la implementación en el ambiente de análisis de datos que se tiene no se hace exactamente como se enumera en el marco teórico, sino que en su lugar se hace una implementación del mismo algoritmo haciendo uso del paradigma MapReduce, por lo cual podría llegar a cambiar un poco la forma de poner en marcha el algoritmo, pero en esencia se continúa haciendo uso y basándose en las reglas de operación del algoritmo.

Por lo que comprender de qué trata el algoritmo, qué busca y cómo lo hace, ayudará a hacer más simple que más adelante únicamente se visualice en un paradigma de programación distinto los que estamos acostumbrados.

La forma de implementarlo haciendo uso del paradigma MapReduce se explica a detalle en las siguientes secciones de este capítulo.

5.3. Diseño

5.3.1. Algoritmo KNN con el uso de MapReduce

Como ya se había explicado anteriormente, MapReduce es una técnica que descompone un trabajo grande en tareas individuales, las cuales pueden ser ejecutadas por separado en diferentes computadoras que componen un clúster, y estas al final pueden unir sus resultados individuales para calcular los resultados finales.

Para que un algoritmo en el paradigma MapReduce entre en funcionamiento se requiere llevar a cabo la programación de 2 funciones la Función Map y la función Reduce por lo que se explicará qué tiene que realizar la función Map y posteriormente la parte correspondiente a la función Reduce para la resolución de este algoritmo.

Para ello se utilizará el ejemplo definido en la sección [Algoritmo KNN \(K-Nearest Neighbors\)](#) con el fin de simplificar un poco el proceso de entendimiento de la manera de implementar este algoritmo con el paradigma MapReduce, tomando en cuenta la referencia teórica de lo que busca realizar este algoritmo y cómo es que este lo realiza.

Función Map:

Se lee línea por línea la información que contiene el archivo de entradas y se procesa una a una. Únicamente tomando en cuenta las columnas que serán utilizadas para su evaluación.

Buscando que los datos contenidos dentro de estas columnas puedan ser procesados por el algoritmo KNN. Es decir, que tengan un valor ya sea Double o Entero.

Tomando como referencia el ejemplo de la sección [Algoritmo KNN \(K-Nearest Neighbors\)](#) se utilizará el mismo dato de entrada para iniciar el algoritmo este dato es el siguiente:

Altura (cm)
161 , 61

Para ello, se procede con la evaluación de cada una de las líneas del archivo la información contenida en la primera línea de la tabla de entradas en el ejemplo es la siguiente:

Altura (cm) , Peso (Kg) , Talla
158 , 58 , M

Esta primera línea será utilizada como ejemplo para mostrar la forma en la que este algoritmo opera para cada una de las líneas. Como ya se mencionó solamente pueden ser aceptadas entradas de tipo Número y Flotante, bajo esta

premisa se puede validar que los elementos Altura y Peso pueden ser utilizados en este algoritmo ya que los valores que los identifican son de tipo numérico. Mientras que, para el valor de Talla al tratarse de un valor que no es ni Double ni Entero, no puede ser procesado directamente por el algoritmo.

Así que se tomarán en cuenta los valores únicamente correspondientes a Altura (cm) y Peso (Kg).

El siguiente paso, será calcular la distancia que existe entre el elemento referencia y la entrada de la tabla que se está procesando.

Este cálculo de distancias se efectúa haciendo el cálculo de la distancia euclíadiana entre los 2 elementos, es decir, la fórmula mostrada en la figura 5.1

$$d_E(P, Q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2} = \sqrt{\sum_{i=1}^n (p_i - q_i)^2}.$$

Figura 5.1: Fórmula para calcular la distancia Euclíadiana.

Para el caso del ejemplo se tendría el siguiente cálculo:

$$\sqrt{(161 - 158)^2 + (61 - 58)^2} = \sqrt{3^2 + 3^2} = \sqrt{18} = 4.2$$

Figura 5.2: Fórmula para calcular la distancia Euclíadiana del ejemplo.

Al momento de tener los cálculos de distancia se puede decir que se termina la intervención de la función Map en este algoritmo y se pasa la ejecución a la sección Reduce para que esta continúe calculando los resultados de salida.

Función Reduce:

En esta función se recibe cada una de las líneas que se mandan desde el Map, y se hace un análisis de si este valor ya había sido recibido anteriormente, esto con el objetivo de que en caso de recibir valores duplicados estos sean manejados como un solo elemento y sólo se lleve un conteo que indique el número de veces que este elemento se ha repetido.

Debido a que particularmente en este ejemplo no se tienen elementos duplicados esta funcionalidad no será evidente y no podrá ser demostrada, sin embargo, en la figura 5.7 se muestra esta funcionalidad y se puede visualizar entre los pasos *Mapeo* y *Partición Intermedia* cómo se integran los datos repetidos y se presentan como un solo resultado con un número 2 asociado a él, lo cual indica que se trata de un elemento que se repite 2 veces.

Primeramente, se arroja el archivo de salidas de manera desordenada es decir en el orden en el que se encuentran los renglones en el archivo de entradas. Esta tabla puede verse como se muestra a continuación:

Altura (cm)	Peso (kg)	Talla	Distancia
158	58	M	4.2
158	59	M	3.6
158	63	M	3.6
160	59	M	2.2
160	60	M	1.4
163	60	M	2.2
163	61	M	2.0
160	64	L	3.2
163	64	L	3.6
165	61	L	4.0
165	62	L	4.1
165	65	L	5.7
168	62	L	7.1
168	63	L	7.3
168	66	L	8.6
170	63	L	9.2
170	64	L	9.5
170	68	L	11.4

Tabla 5.1: Tabla de salida no ordenada de Reduce

Mas adelante, la función Reduce también ordena todos los resultados obtenidos para todas las líneas entregadas por Map. en orden ascendente, es decir, se estaría generando la siguiente tabla.

Altura (cm)	Peso (kg)	Talla	Distancia
160	60	M	1.4
163	61	M	2.0
160	59	M	2.2
163	60	M	2.2
160	64	L	3.2
158	59	M	3.6
158	63	M	3.6
163	64	L	3.6
165	61	L	4.0
165	62	L	4.1
158	58	M	4.2
165	65	L	5.7
168	62	L	7.1
168	63	L	7.3
168	66	L	8.6
170	63	L	9.2
170	64	L	9.5
170	68	L	11.4

Tabla 5.2: Tabla del conjunto de entrenamiento con la columna distancia ya calculada y ordenada.

Sabiendo el valor del número K, el Reduce toma los primeros K elementos de esta tabla, suponiendo que: K=5 Se generaría la siguiente tabla:

Altura (cm)	Peso (kg)	Talla	Distancia
160	60	M	1.4
163	61	M	2.0
160	59	M	2.2
163	60	M	2.2
160	64	L	3.2

Tabla 5.3: Tabla del conjunto de entrenamiento con la columna distancias ya calculada para K=5

Posteriormente, se procede a intentar clasificar estos elementos dentro de una clase.

Para este ejemplo en particular se considerará la columna *Talla* como la clase tomando esta clase como referencia para los 5 k nodos obtenidos se sabe que existen en el conjunto de k nodos:

- 4 Elementos tipo M
- 1 Elemento tipo L

Finalmente podrían arrojarse los 5 K nodos encontrados y con ellos mencionar que se puede clasificar como un elemento de tipo M, ya que existen más nodos encontrados de tipo M que de tipo L.

Esta vendría siendo la salida final del algoritmo KNN, arrojando como salida los K elementos más cercanos encontrados y la clasificación encontrada para estos elementos.

Diagrama de Flujo

Se elaboró un diagrama de flujo que pretende modelar el funcionamiento de este algoritmo como lo hace en el paradigma MapReduce. El diagrama de flujo generado es el siguiente:

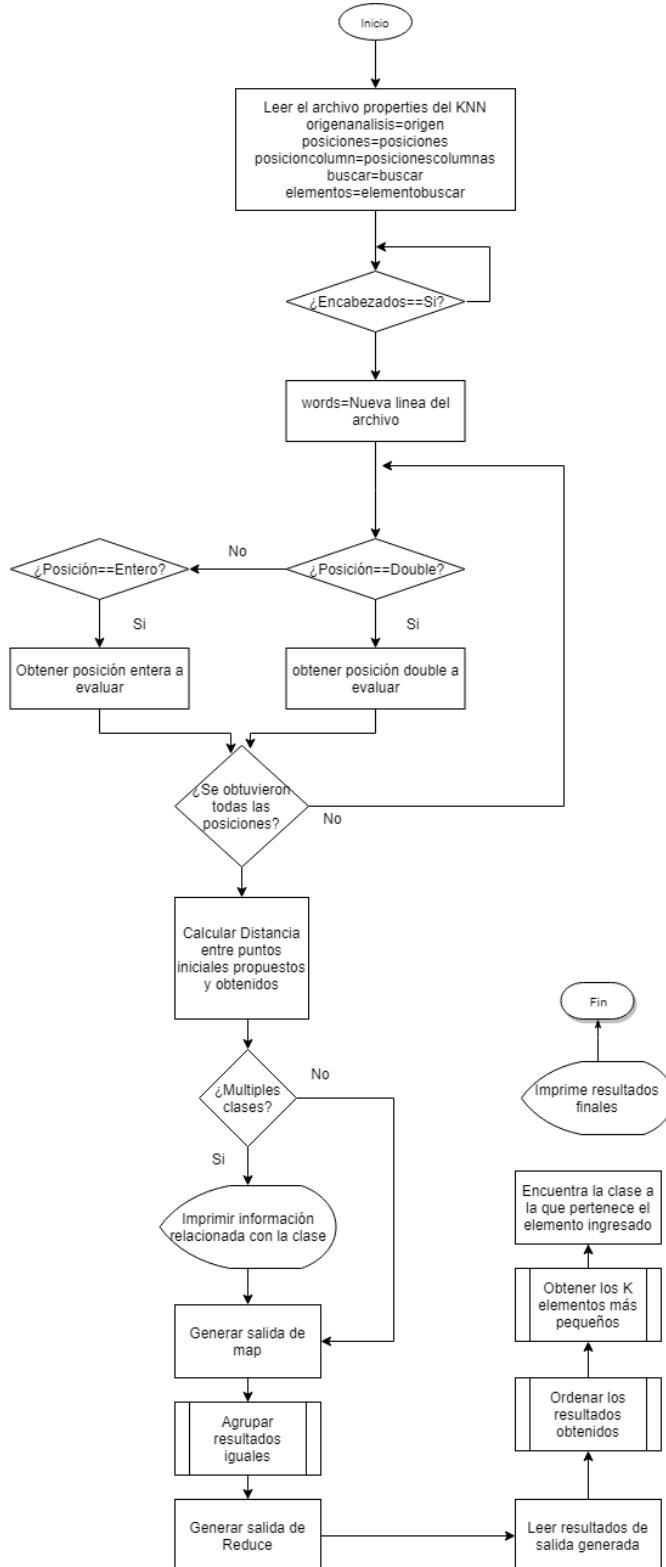


Figura 5.3: Diagrama de Flujo para el algoritmo KNN.

Del diagrama de flujo presentado, se procede a explicar sus pasos:

1. Se lee el archivo "Properties.^{el}" cual almacena la información de los parámetros de configuración que requiere este algoritmo para empezar a funcionar, es decir:

- *posicionescolumnas*: Es el listado de posiciones dentro del archivo donde se encuentran los elementos que conforman la clase a buscar para que los K que se encuentren puedan ser agrupados en clases.

- Buscar: Se trata de una bandera la cual sirve para agregar un elemento en particular que se quiera buscar es decir, de todas las columnas que conforman las clases, se quiere buscar una clase en particular y centrarse únicamente en estos elementos.

Cuando esta bandera se habilita, se tiene este comportamiento, sin embargo, mientras esta bandera no se habilite se consideran todos los elementos del archivo para el análisis.

- Elemento a buscar: Cuando la bandera *Buscar* se habilita, es necesario especificar el elemento que conforma la clase particular a buscar.

Esto se hace dando un elemento de cada una de las columnas especificadas en *posicionescolumnas* de los cuales se desea realizar la búsqueda.

- Origen: Contiene los datos de referencia que se quieren buscar. Podría decirse que se trata de la nueva entrada de conocimiento, estos tienen que ser del mismo tipo de dato que las columnas de interés del usuario ademas de que deben de encontrarse en el mismo orden en que aparecen en el archivo.

- posiciones: Es el listado de los números de columnas que contienen la información que se desea tomar en cuenta para el análisis KNN de la totalidad del archivo.

El resto de columnas que contenga el archivo y que no se encuentren listadas en esta sección simplemente serían ignoradas durante el proceso de análisis.

Es importante considerar que las columnas que se escojan para el análisis sean de tipo Double o Entero ya que en caso de no serlo, la ejecución de este algoritmo fallaría.

- K: es el número de k vecinos que se solicita encontrar para esta prueba

2. El programa valida si se trata de un archivo que contenga encabezados o no, en caso de contenerlos, se omite una linea de lectura para no tomarlos en cuenta para el análisis y continuar con la siguiente.

3. Cada que se lee una linea de datos valida se proceden a realizar lo siguiente:

- Se revisa que de los datos de la linea que se van a considerar para el análisis todos sean del tipo de dato esperado y en caso de serlo, se obtiene su valor; en caso de no serlo, para al menos uno de ellos se omite la evaluación de esa linea.

- Cuando se tienen todas las posiciones, se calcula la distancia que existe entre el elemento a evaluar con el elemento obtenido en la entrada del archivo.

- en caso de encontrarse que la linea leída no es un resultado concluyente ya que pertenece a múltiples clases se imprime la información obtenida para estas múltiples clases, en caso contrario simplemente se arroja la salida

- se compara la salida con salidas otorgadas anteriormente para comprobar que no se trate de una salida repetida ya que en caso de serlo se tiene que integrar a la contabilización de esa salida y no ser tratada como una nueva.

4. Cuando se tienen todas las salidas es decir, cuando se terminó de leer el archivo, se procede a ordenarlas de la menor distancia a la mayor distancia.

5. Teniendo esta información ya se puede conocer cuáles conforman los K vecinos mas cercanos y con ello otorgar el resultado final.

5.3.2. Algoritmo ID3

Cada algoritmo MapReduce comienza su trabajo iniciando la actividad de un Job cada Job ejecuta las 5 tareas de el paradigma MapReduce, que como se sabe, solo se permite al programador tener injerencia en 2 de ellas sin embargo, una vez ejecutándose las 5 tareas se finaliza el programa.

En el caso particular del algoritmo ID3 se trata de un algoritmo que por definición se trata de un algoritmo recursivo. Existen varias consideraciones a tomar en cuenta estas consideraciones se listan a continuación, a su vez como último punto listado se presenta la solución encontrada e implementada tomando en cuenta las consideraciones anteriores listadas en los puntos previos:

- La forma de ejecución de un algoritmo basado en este paradigma consiste en realizar una ejecución particionada del archivo, esta partición es definida por el programador dentro de dicho algoritmo, pero sin importar el criterio de partición que se seleccione definitivamente no se puede realizar un tratamiento de todos los datos en una sola exhibición por lo que se requiere procesar estas entradas de manera independiente una a una por la función Map. Posteriormente, la función Reduce puede hacer algunas operaciones con las salidas, sin embargo no puede regresar a ejecutar nuevamente la función map.
Esto provocaría que se complicaran las cosas ya que al tratarse de un algoritmo recursivo es indispensable que este pueda repetir todos sus pasos cada vez que se manda a llamar, por lo que esta particularidad no haría posible ejecutar este algoritmo de manera inmediata.
- Otra propuesta fue iniciar un nuevo job cada vez que se requiriera de una llamada recursiva. Al intentar esto, ocurrió un problema ya que cada trabajo era independiente del trabajo anterior, por lo que los trabajos comenzaban a ejecutarse de manera paralela provocando que las ejecuciones se efectuarán de forma desincronizada y en ocasiones no fuera posible acceder a los resultados de trabajos que tenían que haber sido ejecutados antes, inclusive, se interrumpían las ejecuciones entre sí.
- Una tercera propuesta fue controlar la ejecución de cada uno de estos job de tal forma que no se comenzará la ejecución de uno nuevo hasta que el anterior no hubiera terminado, en ese sentido, se ejecuta cada uno de los trabajos de manera secuencial y no se interrumpen en sus tareas.
Lo cual proporciona los resultados esperados, con esto se logró emular la funcionalidad de una operación recursiva pero desde la función principal del programa y no directamente dentro del código MapReduce.

En el entendido de que cada MapReduce inicia un nuevo job se explicará el diseño que se realizó dentro de cada una de las funciones involucradas primero se hará para la función map, posteriormente se procederá a explicar la operación de la función reduce y por último se explicará cómo se hace uso de la recursividad desde la función principal.

Para ello se tomará en cuenta el ejemplo mostrado en la sección [Ejemplo de ejecución del algoritmo ID3](#) del marco teórico.

Función Map:

La función Map en este algoritmo buscará primeramente tomar el listado de entradas y almacenarlas de una manera que sea sencilla de entender.

Para ser manipulada posteriormente, se convertirá el archivo de entradas con una estructura plana en una estructura, como la que se muestra a continuación:

Nombre columna			
Elemento 1	Número de apariciones	Salida 1	Numero de apariciones correspondientes a esta salida
		Salida 2	Numero de apariciones correspondientes a esta salida
Elemento 2	Número de apariciones	Salida 1	Numero de apariciones correspondientes a esta salida
		Salida 2	Numero de apariciones correspondientes a esta salida
Elemento 3	Número de apariciones	Salida 1	Numero de apariciones correspondientes a esta salida
		Salida 2	Numero de apariciones correspondientes a esta salida

Figura 5.4: Tabla construcción ID3 sin valores.

Donde se almacena:

-Para cada una de las columnas incluidas en el archivo de entradas de el algoritmo ID3 un listado de los elementos diferentes que la conforman.

-Para cada elemento se incluye el número de veces que este elemento aparece en el archivo de entradas.

De la totalidad de entradas encontradas se contabiliza cuántas de ellas pertenecen a la primera clase, y cuántas a la segunda clase.

La figura 5.5 muestra la estructura descrita con datos reales del ejemplo que se está revisando.

General			
Soleado	5	N	3
Nublado	4	P	2
lluvioso	5	N	2
		P	3

Figura 5.5: Tabla construcción ID3 con valores de la columna Soleado.

Esta tabla se va a actualizando cada vez que la función Map lee una nueva linea del archivo y al finalizar la lectura de todas las líneas se tiene una sola estructura que almacena la información de toda la tabla y que se fue refrescando a cada iteración. Por lo que se puede decir que los datos que en esta tabla se almacenan no son confiables hasta que se recorre el archivo de entradas en su totalidad, en otras palabras, al terminar la ejecución de la función Map.

Una vez que esta tabla es construida en su totalidad, se procede al tratamiento de la misma.

La cual se hace haciendo el cálculo de la entropía general de la tabla de entradas esto se hace tomando los valores de la columna de salidas.

Luego de la entropía específica de cada uno de los diferentes elementos que se encuentran en cada columna.

Con el cálculo de estos valores se procede a realizar el cálculo de la entropía general de cada fila y posteriormente con este valor encontrar la ganancia correspondiente a cada fila.

La ganancia mayor calculada con todas las filas, será la que se tome en cuenta para llevar a cabo la siguiente iteración. La forma en que se hacen estos cálculos es la misma manera que como se hace tradicionalmente se puede saber esto consultando la sección [Ejemplo de ejecución del algoritmo ID3](#).

Una parte importante que mencionar es que si resulta que todos los elementos que se están evaluando terminan en un mismo resultado de salidas, eso significa que no se requiere realizar más evaluaciones, por lo que se llegó a un resultado final. En este caso no se realizan los cálculos de entropía y ganancia mencionados en los párrafos anteriores y en su lugar se devuelve un fin de recursión, esto buscando que la recursividad se detenga.

Función Reduce:

La función *Reduce* en este problema sirve para presentar los resultados calculados en el map y agruparlos de tal forma que puedan ser claramente presentados.

Por otro lado también permite preparar la siguiente corrida del algoritmo en caso de que requiera aplicar la recursividad.

Función Principal Se considera que esta es la parte mas importante a tomar en cuenta para comprender cómo es que funciona esta implementación por lo que esta parte se procederá a explicar a detalle.

La función Main: Únicamente obtiene las dimensiones del archivo buscando conocer el número de columnas.

Posteriormente invoca a la función recursiva que es la que hace todo el trabajo. La cual se explicará mas adelante una vez que la función recursiva termina se escribe el resultado final obtenido durante toda la ejecución del algoritmo en el archivo de salida.

La función Recursiva:

esta función tiene una estructura que de manera simplificada se puede escribir como sigue:

Función recursiva:

Número_de_iteraciones=numero_de_palabras:

Se escriben resultados

Termina iteracion por exceso de intentos y porque ya se probó con todas las columnas

```
Se_llego_a_caso_base:  
    Se escriben resultados  
    Termina iteración por llegar a solución  
Ninguno_de_los_anteriores:  
    Se crea subtabla  
    Se invoca nuevamente a función recursiva
```

Como se puede ver, se tienen 3 condiciones, dos de ellas son condiciones de paro y la tercera es una condición de recursividad. A continuación se realiza la evaluación de lo que hace cada una de ellas.

El algoritmo busca establecer las condiciones que se tienen que tomar en cuenta para hacer una clasificación, por lo tanto, cuando tiene todas estas condiciones completas el algoritmo termina.

No sin antes escribir los resultados encontrados, es decir, que columnas se tienen que tomar en cuenta para llevar a cabo esta clasificación.

- Condición de paro 1: Cuando se encuentra que todos los elementos contenidos en la columna correspondiente a las clases son iguales significa que el elemento ha sido clasificado.
En este caso en particular no es necesario realizar nuevamente el cálculo de las entropías y las ganancias, por lo que solo es necesario establecer que se trata de un caso base, esto se hace indicando que la siguiente columna a evaluar es -1 y cuando se encuentra este resultado únicamente se detienen las invocaciones a la función recursiva para la tabla que se está evaluando.
- Condición de paro 2:
Cuando el contador de iteraciones llega a ser igual al número de columnas contenidas en el archivo significa que ya fueron evaluadas todas las columnas posibles y con ello todos los datos disponibles han sido tomados en cuenta.
En este caso se tiene una tabla de solo un renglón ya que para este punto se han descartado todas las columnas. y al tratarse de una sola posibilidad a pesar de no haberse podido simplificar su clasificación.
El renglón está clasificado para el peor posible caso. En esta situación se escribe la clasificación a la que se llegó y se detienen las llamadas recursivas.
- Condición de recursividad: Cuando ninguna de las 2 anteriores se cumple significa que no se ha logrado llegar a una solución por lo que es necesario seguir iterando.
Esto quiere decir que se volverá a invocar a otro job que ejecute nuevamente las tareas MapReduce. únicamente cambiando la tabla de entradas. e incrementando el número de veces que se ha realizado la recursividad, esta segunda con el objetivo de llevar un control de la primera condición de paro la cual controla el número de iteraciones.

Subtabla: Esta función permite crear tablas más pequeñas de acuerdo a los resultados obtenidos durante el análisis, para que estas nuevas tablas sean las que se manden a evaluar en la siguiente iteración de MapReduce. Por ejemplo, para el caso de la tabla de entradas vista en el marco teórico que se muestra a continuación:

y posterior de eso, después de el primer cálculo se detecta que la mayor ganancia se encuentra en la columna **Aspecto**. por lo que en este caso, existen 3 valores diferentes en esta columna:

- Soleado
- Nublado
- Lluviosos

Esta función entonces tendría que crear las siguientes 3 subtablas para mandarlas a ejecución, en el siguiente ciclo, las cuales serían:

Día	Aspecto	Temperatura	Humedad	Viento	Decisión
1	Soleado	Caluroso	Alta	Ligero	No
2	Soleado	Caluroso	Alta	Fuerte	No
8	Soleado	Templado	Alta	Ligero	No
9	Soleado	Fresco	Normal	Ligero	Sí
11	Soleado	Templado	Normal	Fuerte	Sí

Tabla 5.5: Tabla 1.

Día	Aspecto	Temperatura	Humedad	Viento	Decisión
3	Nublado	Caluroso	Alta	Ligero	Sí
7	Nublado	Fresco	Normal	Fuerte	Sí
12	Nublado	Templado	Alta	Fuerte	Sí
13	Nublado	Caluroso	Normal	Ligero	Sí

Tabla 5.6: Tabla 2

Día	Aspecto	Temperatura	Humedad	Viento	Decisión
4	Lluvioso	Templado	Alta	Ligero	Sí
5	Lluvioso	Fresco	Normal	Ligero	Sí
6	Lluvioso	Fresco	Normal	Fuerte	No
10	Lluvioso	Templado	Normal	Ligero	Sí
14	Lluvioso	Templado	Alta	Fuerte	No

Tabla 5.7: Tabla 3

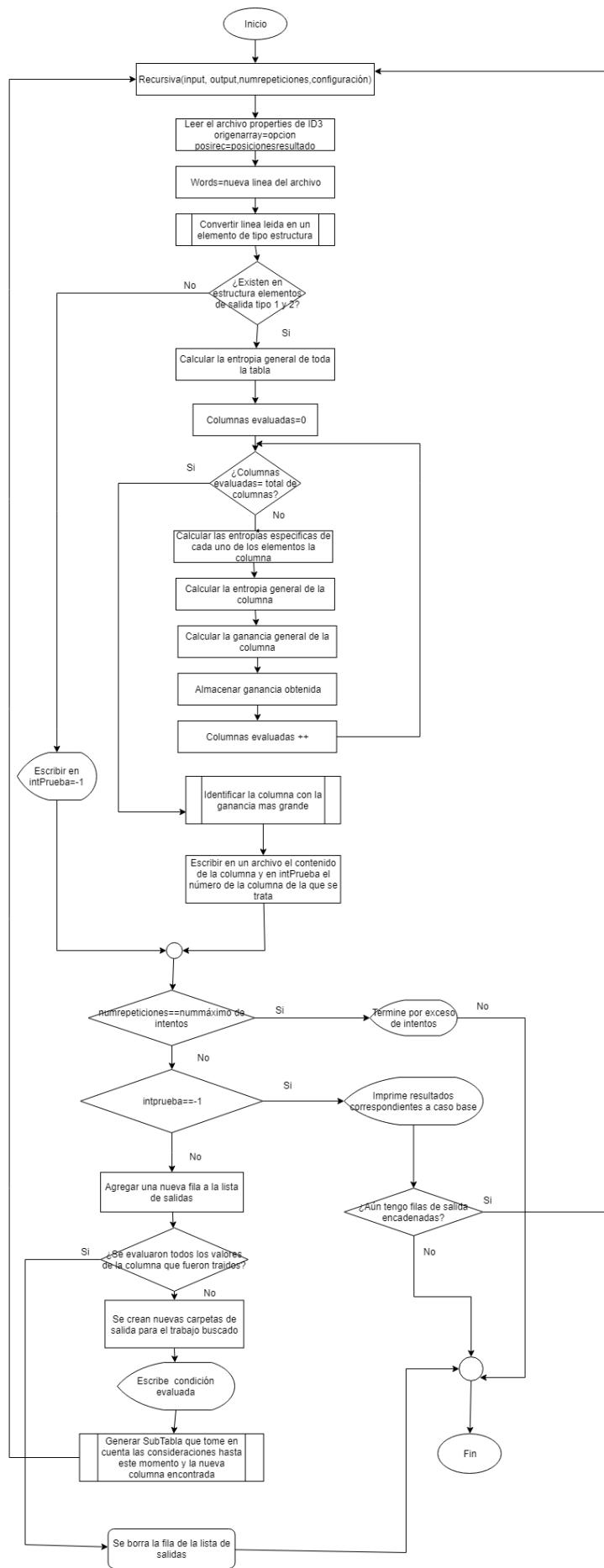
Como se puede ver, lo que hace esta función es crear las tablas para cada uno de los valores contenidos en la columna que posee la mayor ganancia una vez que la tabla ha sido construida se vuelve a invocar a MapReduce las veces que sean necesario y las funciones recursiva junto con subtabla se encargan de la administración de las llamadas recursivas.

Día	Aspecto	Temperatura	Humedad	Viento	Decisión
1	Soleado	Caluroso	Alta	Ligero	No
2	Soleado	Caluroso	Alta	Fuerte	No
3	Nublado	Caluroso	Alta	Ligero	Sí
4	Lluvioso	Templado	Alta	Ligero	Sí
5	Lluvioso	Fresco	Normal	Ligero	Sí
6	Lluvioso	Fresco	Normal	Fuerte	No
7	Nublado	Fresco	Normal	Fuerte	Sí
8	Soleado	Templado	Alta	Ligero	No
9	Soleado	Fresco	Normal	Ligero	Sí
10	Lluvioso	Templado	Normal	Ligero	Sí
11	Soleado	Templado	Normal	Fuerte	Sí
12	Nublado	Templado	Alta	Fuerte	Sí
13	Nublado	Caluroso	Normal	Ligero	Sí
14	Lluvioso	Templado	Alta	Fuerte	No

Tabla 5.4: Tabla de inducción para juegos de tenis ID3.

Diagrama de Flujo

Se elaboró el siguiente diagrama de flujo el cual pretende modelar el funcionamiento del algoritmo ID3 como se hace en el paradigma MapReduce, el diagrama de flujo generado es el siguiente:



Del diagrama de flujo presentado, se procede a explicar sus pasos:

1. El primer paso, es la invocación de la función recursiva para que este ejecute en el primer caso para la tabla original y en lo subsecuente para las tablas generadas.
2. Se lee el archivo "Properties.el" cual almacena la información de los parámetros de configuración que requiere este algoritmo para funcionar:
 - origenarray: contiene las opciones de resultados que se pueden presentar durante la ejecución del algoritmo. En otras palabras se trata de el conjunto de opciones de salida que se tienen. únicamente pueden ser 2 ya que se trata de un algoritmo de salida de tipo binario y vienen separadas por comas.
 - posirec: Contiene la posición en el archivo en la que se encuentran los resultados. Es un elemento de tipo numérico y empieza a contar de 0 en adelante.
3. Se lee una nueva linea el archivo que contiene la tabla para comenzar a trabajar con ella.
4. Para la linea leída se hace el tratamiento correspondiente para que se convierta en un elemento de tipo estructura como el que se explicaba anteriormente, esto con la finalidad de facilitar su tratamiento.
5. Se valida que la estructura tenga 2 diferentes tipos de salida, es decir, que esta no haya sido clasificada aún. Si se encuentra que se trata de un elemento ya clasificado. entonces se indica que no hay ninguna nueva columna a evaluar esto asignándole al valor de la nueva columna a evaluar -1 y con esto, haciendo posible que la función recursiva se detenga en este punto.
En caso de que aun no se haya llegado a la solución, entonces se procede a realizar todos los cálculos que se enlistan en los siguientes puntos.
6. Lo primero es calcular la entropia, general de toda la tabla.
7. Posteriormente, se va columna por columna en la tabla, calculando las entropias de cada elemento en cada una de las columnas.
8. Teniendo todas las entropias correspondientes a una columna, se calcula la entropia general de la columna.
9. Una vez teniendo la entropia general de la columna con ella se puede calcular la ganancia de esta columna.
10. Cuando se tienen todas las entropias y ganancias correspondientes, es necesario comparar todas las ganancias de columnas obtenidas.
Con el fin de obtener la mas grande y conservarla como la columna sobre la que se van a ejecutar los cálculos.
Cuando se conoce esta columna se procede a establecer en un archivo todos los elementos contenidos dentro de la columna y establecer el numero de la columna de la que se trata.
11. Cuando se hace esto entonces se vuelve a la función recursiva y se realizan las validaciones correspondientes.
 - En el caso de que se haya alcanzado el numero máximo de intentos, es decir, la condición 1 entonces se establece esto como salida, se escribe el resultado obtenido y se rompe la ejecución de la función recursiva.
 - En caso de que se trate de un caso base sera necesario escribir el resultado obtenido para posteriormente romper la ejecución de la función recursiva.
 - En caso de que no sea ninguna de las anteriores, se procede a preparar la ejecución para que se haga nuevamente.
Esto se hace tomando en cuenta que para cada columna se tiene mas de un elemento, entonces, cuando se ejecuta el ultimo de ellos, quiere decir que ya se termino la ejecución de esta columna, lo que que será necesario eliminarla de la lista de salidas.
En caso de que no se trate de esta situación entonces se prepara el ambiente necesario para la ejecución de un nuevo job. esto se consigue destinando una nueva carpeta de salida, ya que, cada job requiere una

independiente para su ejecución.

Posteriormente, se escriben los resultados parciales obtenidos hasta este momento durante la ejecución del algoritmo.

Y por ultimo es necesario generar la subtabla con la información recuperada de lo que debe contener la nueva tabla a ser ejecutada.

5.4. Desarrollo

5.4.1. Algoritmo KNN

El algoritmo KNN escrito en JAVA servirá para realizar operaciones de clasificación sobre los datos que se almacenen en el HDFS por parte de los usuarios finales.

Este algoritmo, al igual que todos los algoritmos que sean programados con el paradigma Map Reduce. únicamente se requiere programar sus secciones de Map y de Reduce, mientras que para el resto de secciones no contempladas este paradigma las auto completa para poder funcionar.

Sin embargo para que esta funcionalidad pueda operar de manera correcta es necesario que se siga el paradigma de programación Map Reduce, ya que en caso de no hacerlo como se indica el resto de funciones intermedias no podrían integrarse como se espera.

Se leerán y procesarán archivos de texto con formato CSV, entradas que permitirá el algoritmo.

Con lo que, este algoritmo podrá procesar archivos de datos que tengan una fila de datos de encabezado y al mismo tiempo podría hacerlo si este archivo no la tiene.

Para el correcto funcionamiento de este algoritmo se pueden hacer las siguientes consideraciones:

- Es importante invocar un algoritmo de ordenamiento que permita ordenar los resultados obtenidos, esto para obtener un método de ordenamiento optimo que permita optimizar la solución y como , se sabe que se trabajaría mayormente con grandes cantidades de datos, se busca que este algoritmo sea lo mas optimo posible.
- Es necesario realizar la importación de las librerías de Map y de Reduce, ya que, se trata de librerías requeridas por Hadoop para que este algoritmo pueda funcionar.
- Se utilizan archivos de salida adicionales para poder gestionar el resto de salidas y poder ver los resultados en diferentes puntos del proceso, lo cual podría tener gran valor para el usuario final.
- Se agrego una salida en formato de datos de Excel esta con el objetivo de que el manejo de datos sea mas fácil para el usuario final, y se puedan utilizar las funcionalidad de Excel para manipular los resultados de salida.
- Se permite personalizar desde el archivo de configuraciones las rutas donde el usuario desea escribir las salidas, para que este pueda tener control de las diferentes ejecución que haga y donde estas se están almacenando.
- En ocasiones los archivos de datos tienen registros repetidos que no vale la pena contabilizar como mas de un vecino, ya que se trata del mismo.en este caso se realiza una agrupación para que se consideren uno mismo y solamente se le notifica al usuario cuantas veces se encontró este vecino en particular a lo largo del archivo.

En la imagen [Funcionamiento completo de algoritmo KNN con todas sus operaciones en Hadoop](#) se puede visualizar el funcionamiento completo que tendría que ejecutarse en el cluster para llevar a cabo este algoritmo.

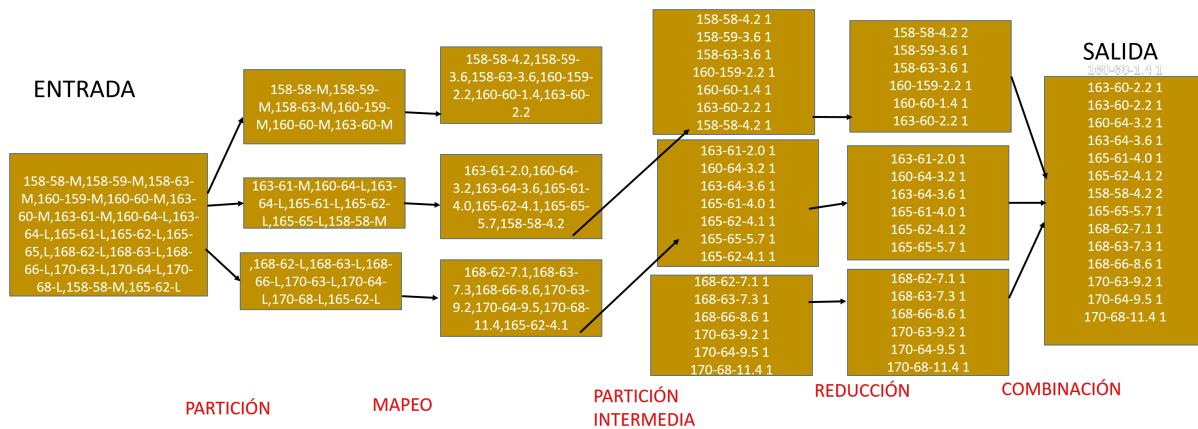


Figura 5.7: Funcionamiento completo de algoritmo KNN con todas sus operaciones en Hadoop.

Se explicarán los módulos que forman parte de la ejecución de el algoritmo KNN haciendo uso del paradigma MapReduce.

1. **Partición:** Se requiere definir un parametro de partición, esto para que se puedan reconocer las unidades a analizar y con esto poder asignar tareas a los nodos. el parámetro de partición puede ser cualquier cosa, por ejemplo, dividir por espacio, coma, punto y coma, o incluso por una nueva línea (' n'). En este caso, se realiza por una nueva línea del archivo es decir (' n').
2. **Mapeo:** La explicación de esta parte ya se realizo anteriormente en la sección [Algoritmo KNN con el uso de MapReduce Función Map](#).
3. **Partición intermedia:** Se busca generar grupos con los datos que después de pasar por el módulo de mapeo y tener la estructura de salida de este modulo tienen la misma CLAVE, al cumplir esta condición se asignan en el mismo grupo. Se generan tantos grupos como claves existan.
4. **Reducción:** Se explicó anteriormente en [Algoritmo KNN con el uso de MapReduce Función Reduce](#).
5. **Combinación:** Todos los datos de salida que arrojó la función de reducción son combinados y puestos en un sólo grupo para generar un resultado final.

Es necesario generar un JAR para la ejecución de este algoritmo, esto se hace ya sea desde un IDE de Java, una instalación de Java en alguna de las maquinas, como la que efectúa el [Instalador de Luminus](#) o bien desde el mismo Hadoop, esto podría hacerse ejecutándose el siguiente comando:

```
bin/hadoop com.sun.tools.javac.Main [clase_principal_java].java
jar cf [Nombre_jar].jar [clase_principal_java]*.class
```

5.4.2. Algoritmo ID3

El algoritmo ID3 escrito en Java permite realizar operaciones soportadas por los arboles de decisión sobre los datos que se almacenen en el HDFS por parte de los usuarios finales.

Únicamente es necesario realizar la programación correspondiente a la sección Map y a la sección Reduce de acuerdo a las consideraciones de programación propuestas durante la sección de diseño.

A continuación se listan las consideraciones que se tienen que tomar en cuenta para el correcto funcionamiento de este algoritmo:

- Este algoritmo soporta ejecuciones para cualquier entrada de datos en formato tabla siempre y cuando se cumpla la condición de que, la columna que se desee utilizar como columna de salida, solamente tenga 2 posibles opciones de resultado, lo cual permitiría hacer la clasificación de los elementos.

- Este algoritmo acepta archivos de tipo .txt y de tipo .csv es decir archivos de texto plano con la única obligatoriedad de que los separadores que existan entre las columnas que conforman el archivo sean , coma. ya que dentro del código se utiliza este carácter como separador.
- Se requiere que la carpeta donde se van a escribir los resultados parciales a partir de la segunda ejecución no exista y en caso de existir que no contenga archivos propios de una ejecución anterior, ya que en caso de hacerlo este intentará sobre escribirlos y se generará un error en tiempo de ejecución.
- El archivo properties contiene las configuraciones de ordenes de ejecución para cambiar y manipular entre ejecución y ejecución.
- Debido a que el programa levanta múltiples jobs para realizar un mismo trabajo, no hay una forma certera de conocer el progreso en la ejecución del algoritmo ya que el contado de Map y Reduce se estarán reiniciando cada que se levante un nuevo job. Por lo que, cuando ambos se vean muy cercanos al 100 % no es una señal de que el algoritmo está por terminar, sino mas bien que una de las ejecuciones esta por hacerlo.

Este algoritmo se programó haciendo además la implementación de una clase que se llama Estructura, la cual contiene la definición que representa a una entrada dentro del archivo para facilitar su tratamiento.

Esta clase es independiente de las clases que son utilizadas para la propia corrida del algoritmo, y es de utilidad ya que permite simplificar el manejo de los datos, en diferentes puntos del algoritmo.

Estructurarlos de una manera cómoda simplifica el tratamiento de los datos y puede hacer mas simple su manejo en diferentes puntos de la de la ejecución.

Por lo que respecta a lo propio del algoritmo se siguió lo estipulado en el diseño para el desarrollo del mismo , y se tiene una programación que empata por completo con el diseño que se propuso para su funcionamiento.

Para que el código fuente pueda ser funcional y pueda ser ejecutado desde Hadoop, es necesario que este se encuentre compilado como un JAR, para realizar esta tarea, es posible utilizar el siguiente comando.

```
bin/hadoop com.sun.tools.javac.Main [clase_principal_java].java  
jar cf [Nombre_jar].jar [clase_principal_java]*.class
```

Cuando se tenga al código fuente representado como un JAR, entonces se podrá ejecutar como se indica en la sección de pruebas para obtener los resultados esperados para los casos de prueba que tenga el usuario.

5.5. Pruebas

5.5.1. Algoritmo KNN

Prueba 1

A continuación se muestra una prueba de la ejecución de este algoritmo para comprobar su correcto funcionamiento y que esta operando como debería hacerlo:

Para ello y con el objetivo de que sea mas claro, se utilizará el mismo ejemplo que se viene manejando durante todo el capítulo y también en el marco teórico.

Esto porque, es un ejemplo que ya se conoce y que facilitará el entendimiento de la prueba.

Como primer instancia es necesario crear el archivo de configuraciones para esta ejecución el cual, deberá contener la siguiente información:

```
posicionescolumnas=1
```

```
buscar=N
```

```
elementoabuscar=
```

```
origen= 161,61
```

posiciones=1,2

K=5

Con esta configuración indicaremos:

1. Se quiere agrupar tomando en cuenta el contenido de la columna 1 es decir la que contiene información relacionada con el peso
2. No se desea buscar ningún elemento de peso en específico por lo que se dice que no y no se indica en elemento a buscar.
3. El elemento a tomar de referencia o bien, el que se quiere comparar esta representado por los valores 158, 58
4. Las posiciones en el archivo que se desea utilizar para realizar esta evaluación son las posiciones 1 y 2 es decir las que contienen la Altura (cm) y el Peso en (kg).
5. Se desea encontrar los 5 vecinos mas cercanos. Por otro lado, otro archivo importante a construir antes de comenzar las pruebas es el archivo de entradas.

Este archivo contiene la tabla con la información de entradas con la que se alimentará el algoritmo, para este ejemplo particular, el archivo tendrá el contenido siguiente:

```
Altura,Peso,Talla
160,60,M
163,61,M
160,59,M
163,60,M
160,64,L
158,59,M
158,63,M
163,64,L
165,61,L
165,62,L
158,58,M
165,65,L
168,62,L
168,63,L
168,66,L
170,63,L
170,64,L
170,68,L
```

Una vez que se han creado estos 2 archivos es necesario que estos se suban al HDFS ya que serán necesarios en el momento que el algoritmo se ejecute.

Para comenzar el proceso, primeramente se ejecuta el siguiente comando:

```
root@maestro:/opt/hadoop/etc/hadoop: yarn jar /home/mayra/Escritorio/MRProgramsDemo.jar
PackageDemo.WordCount "user/luminus/documentoknn1/tabla.txt" "user/luminus/documentoknn1/output"
"user/luminus/documentoknn1/configuracion.properties"
```

como puede visualizarse en la imagen siguiente.

```
root@Pony:/opt/hadoop/bin# ./yarn jar /home/mayra/Escritorio/MRProgramsDemo.jar PackageDemo.WordCount /user/luminus/documentoknn1/tabla.txt /user/luminus/documentoknn1/output
2019-05-01 15:08:44,911 INFO client.RMProxy: Connecting to ResourceManager at Pony/192.168.1.72:8032
2019-05-01 15:08:46,969 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/root/.staging/job_1556738561291_0006
2019-05-01 15:08:47,378 INFO input.FileInputFormat: Total input files to process : 1
2019-05-01 15:08:47,922 INFO mapreduce.JobSubmitter: number of splits:1
2019-05-01 15:08:48,058 INFO Configuration.deprecation: yarn.resourcemanager.system-metrics-publisher.enabled is deprecated. Instead, use yarn.system-metrics-publisher.enabled
2019-05-01 15:08:48,595 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1556738561291_0006
2019-05-01 15:08:48,604 INFO mapreduce.JobSubmitter: Executing with tokens: []
2019-05-01 15:08:49,697 INFO conf.Configuration: resource-types.xml not found
2019-05-01 15:08:49,698 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2019-05-01 15:08:50,030 INFO impl.YarnClientImpl: Submitted application application_1556738561291_0006
2019-05-01 15:08:50,309 INFO mapreduce.Job: The url to track the job: http://Pony:8088/proxy/application_1556738561291_0006/
2019-05-01 15:08:50,311 INFO mapreduce.Job: Running job: job_1556738561291_0006
```

Figura 5.8: Funcionamiento completo de algoritmo KNN con todas sus operaciones en Hadoop.

Con esto empezará la ejecución del algoritmo y solo será cuestión de esperar a que se finalice su ejecución de manera correcta esto ocurre cuando el archivo termina de escribir todos los archivos correspondientes en el directorio como se muestra en la imagen 5.9

```
Input split bytes=118
Combine input records=0
Combine output records=0
Reduce input groups=18
Reduce shuffle bytes=897
Reduce input records=18
Reduce output records=18
Spilled Records=36
Shuffled Maps =1
Failed Shuffles=0
Merged Map outputs=1
GC time elapsed (ms)=762
CPU time spent (ms)=4780
Physical memory (bytes) snapshot=332619776
Virtual memory (bytes) snapshot=4120244224
Total committed heap usage (bytes)=138625024
Peak Map Physical memory (bytes)=215040000
Peak Map Virtual memory (bytes)=2056044544
Peak Reduce Physical memory (bytes)=117579776
Peak Reduce Virtual memory (bytes)=2064199680
Shuffle Errors
BAD_ID=0
CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0
File Input Format Counters
Bytes Read=183
File Output Format Counters
Bytes Written=819
root@Pony:/opt/hadoop/bin#
```

Figura 5.9: Termino de ejecución de algoritmo knn.

Cuando esta ejecución finalice, se podrán ver las salidas generadas por Hadoop directamente en el HDFS. se presenta

una captura de pantalla de la información desplegada en el directorio indicado durante la ejecución en la figura 5.10

Browse Directory

/user/luminus/documentoknn1/output									Go!			
Show 25 entries		Search:										
	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name				
<input type="checkbox"/>	-rw-r--r--	root	supergroup	0 B	May 01 15:10	1	128 MB	_SUCCESS				
<input type="checkbox"/>	-rw-r--r--	root	supergroup	819 B	May 01 15:10	1	128 MB	part-r-00000				

Showing 1 to 2 of 2 entries

Previous 1 Next

Hadoop, 2018.

Figura 5.10: Directorio de salida del HDFS.

como siguiente paso, se procede a visualizar el contenido del archivo llamado *part-r-00000* el cual contiene la información de salida correspondiente a esta ejecución. El contenido de este archivo se muestra en la figura 5.11

```
1.4142135623730951, 160.0 60.0 clase: 160 1
11.40175425099138, 170.0 68.0 clase: 170 1
2.0, 163.0 61.0 clase: 163 1
2.23606797749979, 160.0 59.0 clase: 160 1
2.23606797749979, 163.0 60.0 clase: 163 1
3.1622776601683795, 160.0 64.0 clase: 160 1
3.605551275463989, 158.0 59.0 clase: 158 1
3.605551275463989, 158.0 63.0 clase: 158 1
3.605551275463989, 163.0 64.0 clase: 163 1
4.0, 165.0 61.0 clase: 165 1
4.123105625617661, 165.0 62.0 clase: 165 1
4.242640687119285, 158.0 58.0 clase: 158 1
5.656854249492381, 165.0 65.0 clase: 165 1
7.0710678118654755, 168.0 62.0 clase: 168 1
7.280109889280518, 168.0 63.0 clase: 168 1
8.602325267042627, 168.0 66.0 clase: 168 1
9.219544457292887, 170.0 63.0 clase: 170 1
9.486832980505138, 170.0 64.0 clase: 170 1
```

Figura 5.11: Archivo de salidas part-r-00000.

Dentro de este archivo se puede visualizar el cálculo de distancias que se hizo para cada uno de los puntos dados, con respecto al punto.

Altura (cm), Peso (Kg), Talla
158 , 58 , M

para cada uno de ellos se especifica:

- El valor de distancia calculado entre cada uno de los puntos del archivo y el punto de referencia especificado.
- Los valores de Altura y Peso que se encontraron en cada renglón del archivo, que fueron utilizados en ese renglón específico para hacer el cálculo de distancias
- La clase a la que pertenece, ese elemento en particular, la cual puede ser escogida entre una o más columnas del archivo. En este caso se selecciono la clase Altura como se puede ver en el primer archivo de properties llenado, al inicio de la sección [Prueba 1](#). Donde se especifica que posicionescolumnas=1 es decir que se tomará como referencia la columna 1 que corresponde a Altura.
- Un contador que indica el numero de veces que se repite cada elemento dentro del archivo, como en este caso todos son diferentes, cada uno de ellos viene etiquetado con el valor de 1.

Como se mencionaba anteriormente, esta solo representa la primera salida, ya que se generan 2 archivos de salidas mas como se muestra en la figura [5.12](#), los cuales se explican a continuación. Esto es importante ya que, así, el usuario experto final podrá tener acceso a la información en diferentes puntos del proceso y utilizarla según le sea de utilidad.

May 01 15:08	<u>1</u>	128 MB	salidak.txt
May 01 15:08	<u>1</u>	128 MB	salidaordenadacompleta.txt 

Figura 5.12: Archivos adicionales de salidas.

El segundo archivo de salidas, es constituido por una salida ordenada del archivo, es decir, la misma información desplegada en [5.11](#) pero en este caso en orden ascendente de acuerdo a la distancia como se puede visualizar en la figura [5.13](#)

```
1.4142135623730951, 160.0 60.0 clase: 160 1
2.0, 163.0 61.0 clase: 163 1
2.23606797749979, 160.0 59.0 clase: 160 1
2.23606797749979, 163.0 60.0 clase: 163 1
3.1622776601683795, 160.0 64.0 clase: 160 1
3.605551275463989, 158.0 59.0 clase: 158 1
3.605551275463989, 158.0 63.0 clase: 158 1
3.605551275463989, 163.0 64.0 clase: 163 1
4.0, 165.0 61.0 clase: 165 1
4.123105625617661, 165.0 62.0 clase: 165 1
4.242640687119285, 158.0 58.0 clase: 158 1
5.656854249492381, 165.0 65.0 clase: 165 1
7.0710678118654755, 168.0 62.0 clase: 168 1
7.280109889280518, 168.0 63.0 clase: 168 1
8.602325267042627, 168.0 66.0 clase: 168 1
9.219544457292887, 170.0 63.0 clase: 170 1
9.486832980505138, 170.0 64.0 clase: 170 1
11.40175425099138, 170.0 68.0 clase: 170 1
```

Figura 5.13: Archivo Salida Ordenada Completa.

En este archivo se puede visualizar la misma información que se presentaba anteriormente en el archivo [5.11](#) pero los datos están ordenados en forma ascendente por la distancia calculada. lo cual ya nos empieza a proporcionar una

idea de cuales podrían ser los vecinos mas cercanos.

El ultimo archivo representa la salida final, como tal este únicamente muestra la cantidad de vecinos especificados y además de acuerdo a la clase seleccionada intenta clasificarlos siempre y cuando en el número de k seleccionado haya alguna clase para la que este número de elementos tenga más salidas. Lo anterior se muestra en la figura 5.14

```
se solicito un k=5 se listan estos k nodos
1.4142135623730951, 160.0 60.0 clase: 160 1
2.0, 163.0 61.0 clase: 163 1
2.23606797749979, 160.0 59.0 clase: 160 1
2.23606797749979, 163.0 60.0 clase: 163 1
3.1622776601683795, 160.0 64.0 clase: 160 1
pertenece a 160 la cual forma parte de 3 k cercanos
```

Figura 5.14: Archivo Salida k.

En este archivo, como se especificó un numero de vecinos igual a 5 entonces se muestran únicamente las primeras 5 entradas del archivo 5.13 y ademas se busca la clase que como se puede ver se repite la clase **160** 3 veces en las 5 salidas por lo que se escribe que esta es la clase a la que pertenece con una cantidad de 3k vecinos cercanos.

Prueba 2

A continuación se muestra otra ejecución de este mismo algoritmo con diferentes configuraciones de properties para poder ver otra funcionalidad que puede ser aplicada al momento de hacer uso de este algoritmo.

El archivo de configuraciones escogido para esta ejecución, tiene la siguiente información:

```
posicionescolumnas=2,3
```

```
buscar=S
```

```
elementoabuscar=62,L
```

```
origen= 161,61
```

```
posiciones=2,3
```

```
K=5
```

Con estas configuraciones se indica:

- Posicionescolumnas: Las columnas donde se encuentran los elementos que conforman la clase son la 2 y la 3. Es decir, para este ejemplo en particular, Peso y Talla
- Buscar: Al momento de indicar buscar en el valor de S significa que se tiene un determinado elemento en el valor de posiciones columnas el cual se desea buscar.
Es decir, para valores diferentes de esta clase no se realizaría el análisis y se omitiría esa fila en el archivo
- Elemento a Buscar: A continuación se muestra el elemento en particular que se desea buscar dentro del archivo. En este caso se indica que se trata del elemento 62,L.
Cabe destacar que los elementos deben ser indicados tal cual se muestran en el archivo de datos de coma a coma, con los espacios y todo lo que venga incluido entre ellos. En caso de no realizarse de esta manera no se encontrarán coincidencias para estos elementos.

El archivo de entradas tiene la misma información para ser ocupada durante el proceso de análisis.

Pero al momento de cambiar el archivo de configuración los resultados que podremos obtener serán diferentes.

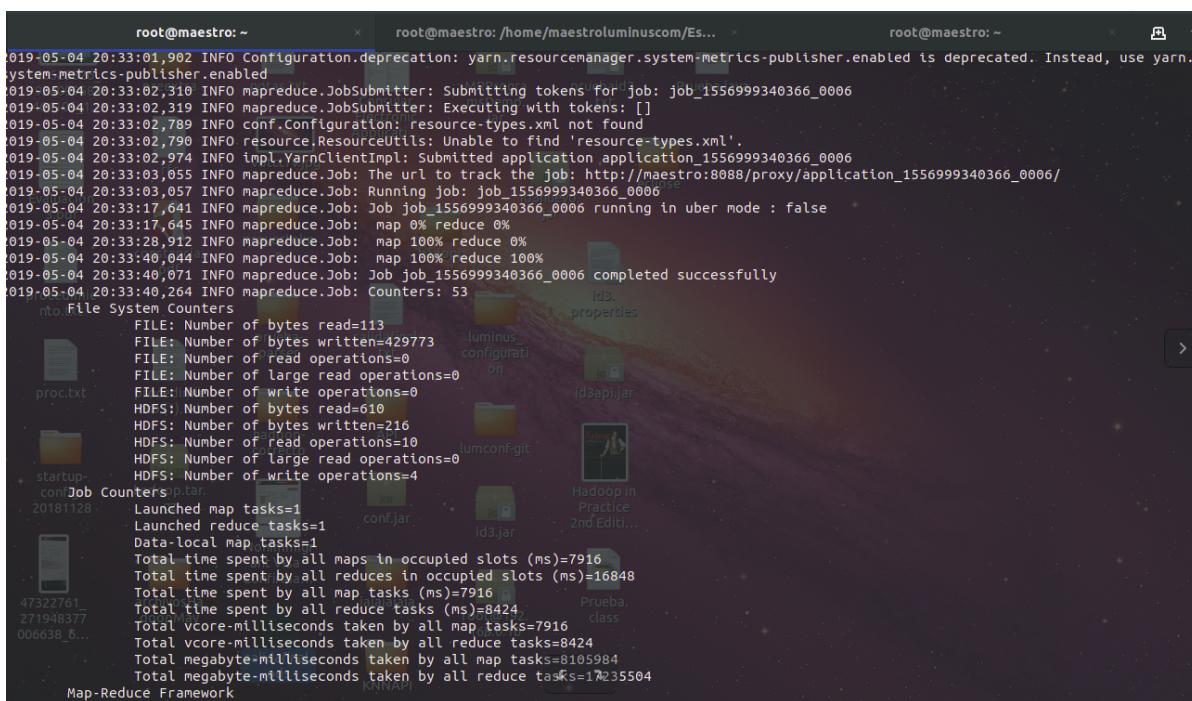
Altura,Peso,Talla

160,60,M
 163,61,M
 160,59,M
 163,60,M
 160,64,L
 158,59,M
 158,63,M
 163,64,L
 165,61,L
 165,62,L
 158,58,M
 165,65,L
 168,62,L
 168,63,L
 168,66,L
 170,63,L
 170,64,L
 170,68,L

A continuación se procede con la ejecución del algoritmo:
 esta se realiza con el comando siguiente:

```
root@maestro:/opt/hadoop/etc/hadoop: yarn jar /home/mayra/Escritorio/MRProgramsDemo.jar
PackageDemo.WordCount "user/luminus/documentoknn1/tabla.txt" "user/luminus/documentoknn1/output1" "user/lum
```

La ejecución de este algoritmo se puede ver en la figura 5.15

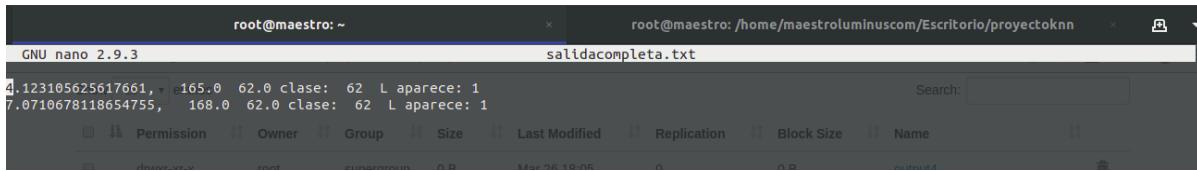


```
root@maestro: ~          root@maestro:/home/maestroluminuscom/Esc...          root@maestro: ~
2019-05-04 20:33:01,902 INFO Configuration.deprecation: yarn.resourcemanager.system-metrics-publisher.enabled is deprecated. Instead, use yarn.
system-metrics-publisher.enabled
2019-05-04 20:33:02,310 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1556999340366_0006
2019-05-04 20:33:02,319 INFO mapreduce.JobSubmitter: Executing with tokens: []
2019-05-04 20:33:02,789 INFO conf.Configuration: resource-types.xml not found
2019-05-04 20:33:02,790 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2019-05-04 20:33:02,974 INFO impl.YarnClientImpl: Submitted application application_1556999340366_0006
2019-05-04 20:33:03,055 INFO mapreduce.Job: The url to track the job: http://maestro:8088/proxy/application_1556999340366_0006/
2019-05-04 20:33:03,057 INFO mapreduce.Job: Running job: job_1556999340366_0006
2019-05-04 20:33:17,641 INFO mapreduce.Job: Job job_1556999340366_0006 running in uber mode : false
2019-05-04 20:33:17,645 INFO mapreduce.Job: map 0% reduce 0%
2019-05-04 20:33:28,912 INFO mapreduce.Job: map 100% reduce 100%
2019-05-04 20:33:40,071 INFO mapreduce.Job: Job job_1556999340366_0006 completed successfully
2019-05-04 20:33:40,264 INFO mapreduce.Job: Counters: 53
  File System Counters
    FILE: Number of bytes read=113
    FILE: Number of bytes written=429773
    FILE: Number of read operations=0
    FILE: Number of large read operations=0
    FILE: Number of write operations=0
    HDFS: Number of bytes read=610
    HDFS: Number of bytes written=216
    HDFS: Number of read operations=10
    HDFS: Number of large read operations=0
    HDFS: Number of write operations=4
  Launched map tasks=1
  Launched reduce tasks=1
  Data-local map tasks=1
  Total time spent by all maps in occupied slots (ms)=7916
  Total time spent by all reduces in occupied slots (ms)=16848
  Total time spent by all map tasks (ms)=7916
  Total time spent by all reduce tasks (ms)=8424
  Total vcore-milliseconds taken by all map tasks=7916
  Total vcore-milliseconds taken by all reduce tasks=8424
  Total megabyte-milliseconds taken by all map tasks=8105984
  Total megabyte-milliseconds taken by all reduce tasks=17235504
Map-Reduce Framework
```

Figura 5.15: Ejecución Algoritmo KNN.

De primera instancia, como únicamente son evaluadas las entradas que contengan la información estipulada en el archivo de configuraciones, se tomarán en cuenta una cantidad inferior de entradas.

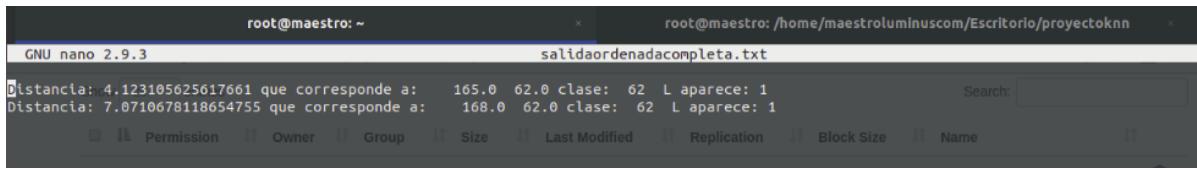
Por lo que el archivo de salidas completo se ve como se muestra en la figura 5.16



```
root@maestro: ~                               root@maestro: /home/maestroluminuscom/Escritorio/proyectoKnn
GNU nano 2.9.3                                     salidaCompleta.txt
4.123105625617661, 165.0 62.0 clase: 62 L aparece: 1
7.0710678118654755, 168.0 62.0 clase: 62 L aparece: 1
[REDACTED]
```

Figura 5.16: Salida completa algoritmo KNN.

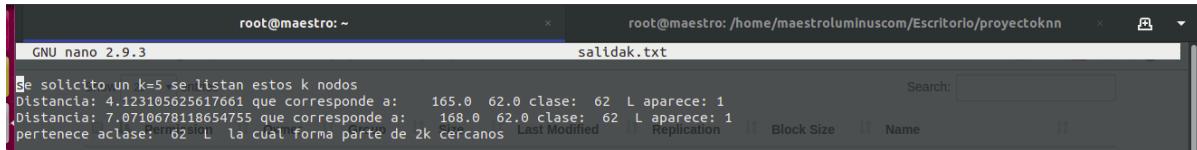
Tomando entonces, esta salida como referencia, la salida ordenada completa se puede visualizar en la figura 5.17



```
root@maestro: ~                               root@maestro: /home/maestroluminuscom/Escritorio/proyectoKnn
GNU nano 2.9.3                                     salidaOrdenadaCompleta.txt
Distancia: 4.123105625617661 que corresponde a: 165.0 62.0 clase: 62 L aparece: 1
Distancia: 7.0710678118654755 que corresponde a: 168.0 62.0 clase: 62 L aparece: 1
[REDACTED]
```

Figura 5.17: Salida Ordenada Completa KNN.

Una vez que se hace este ordenamiento, la siguiente salida es la salida K y se pidió una K igual a 5, pero como únicamente se pudieron obtener 2 k durante esta ejecución el algoritmo es persistente y de todas formas arroja la salida, únicamente mostrando las salidas que encontró, lo cual se puede ver en la figura 5.18.



```
root@maestro: ~                               root@maestro: /home/maestroluminuscom/Escritorio/proyectoKnn
GNU nano 2.9.3                                     salidaK.txt
Se solicito un k=5 se listan estos k nodos
Distancia: 4.123105625617661 que corresponde a: 165.0 62.0 clase: 62 L aparece: 1
Distancia: 7.0710678118654755 que corresponde a: 168.0 62.0 clase: 62 L aparece: 1
[REDACTED]
```

Figura 5.18: Salida K algoritmo KNN

Como se puede observar, debido a que la clase de las salidas es la misma, se clasifica el elemento en esta clase, y a pesar de que se pide un número de 5 nodos cercanos únicamente se muestran 2.

Por otro lado, se genera esta salida también en tipo excel, la cual se muestra a continuación en la siguiente figura 5.19:

final.ods (solo lectura) - LibreOffice Calc														
A1	Distancia	C	D	E	F	G	H	I	J	K	L	M		
Este documento se abrió en modo de solo lectura.														
1	Distancia	Ubicacion												
2	4.12310562561766	165.0	62.0	clase: 62	L	aparece: 1								
3	7.07106781186548	168.0	62.0	clase: 62	L	aparece: 1								
4														
5														
6														
7														
8														
9														
10														
11														
12														
13														
14														
15														
16														
17														
18														
19														
20														
21														
22														
23														
24														
25														
26														
27														
28														
29														
30														
31														
32														
33														

Figura 5.19: Salida en formato Excel Algoritmo KNN.

Con esto, se podrían dar por concluidas las pruebas del algoritmo KNN, con lo que se puede decir que este opera de manera correcta con y sin el uso de la característica extra de agrupación por clase y las pruebas han resultado exitosas.

5.5.2. Algoritmo ID3

A continuación se muestra una prueba de la ejecución de este algoritmo para comprobar su correcto funcionamiento y que esta operando como debería hacerlo:

Para ello y con el objetivo de que sea mas claro, se utilizará el mismo ejemplo que fue revisado en el marco teórico del presente documento, el cual puede ser visualizado en [Ejemplo de ejecución del algoritmo ID3](#).

Esto debido a que se trata de un ejemplo el cual, ya es conocido y se conoce la forma en la que debe operar y los resultados esperados por lo que en este sentido facilitará el entendimiento de la prueba.

Como primer instancia es necesario crear el archivo de configuraciones para esta ejecución, el cual deberá contener la siguiente información:

```
posicionresultado=4
```

```
opciones=N,P
```

```
nombres=General,Temperatura,Humedad,Viento,Clase
```

Además, se procede a crear el archivo de entradas el cual contiene la información que será necesaria para la ejecución del algoritmo este archivo de entradas es el siguiente:

```
Soleado,caliente,alta,no,N
```

```
soleado,caliente,alta,si,N
```

```
nublado,caliente,alta,no,P
```

```
lluvioso,templada,alta,no,P
```

```

lluvioso,fria,normal,no,P
lluvioso,fria,normal,si,N
nublado,fria,normal,si,P
soleado,templada,alta,no,N
soleado,fria,normal,no,P
lluvioso,templada,normal,no,P
soleado,templada,normal,si,P
nublado,templada,alta,si,P
nublado,caliente,normal,no,P
lluvioso,templada,alta,si,N

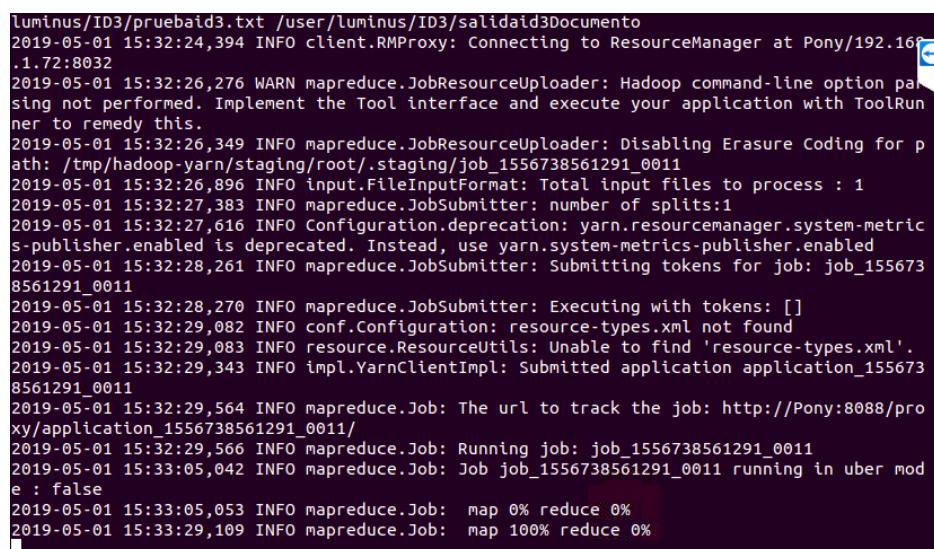
```

Una vez listas estas configuraciones iniciales se procede a iniciar la ejecución de este algoritmo esto haciendo uso de la siguiente instrucción de línea de comandos.

```
root@maestro:/opt/hadoop/etc/hadoop: yarn jar /home/mayra/Escritorio/ID3.jar
id3.id3algoritm "user/luminus/ID3/pruebaid3.txt" "user/luminus/ID3/salidaid3Documento" "user/luminus/ID3/i
```

La segunda especificada es la ruta dentro de la cual se escribirá la primera salida de la ejecución.

Es importante resaltar que una vez que termine de ejecutar el primer job se buscará la siguiente carpeta donde continuará escribiendo los resultados, por lo que se tiene que destinar una carpeta donde se escribirán el resto de salidas. la información relacionada con esta carpeta será introducida desde la interface de programación de aplicaciones de Luminus que hará uso de este algoritmo, y la escribirá directamente sobre el archivo properties. En la figura 5.20 se puede apreciar la puesta en marcha y el inicio de ejecución de este algoritmo.



```

luminus/ID3/pruebaid3.txt /user/luminus/ID3/salidaid3Documento
2019-05-01 15:32:24,394 INFO client.RMProxy: Connecting to ResourceManager at Pony/192.168.1.72:8032
2019-05-01 15:32:26,276 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.
2019-05-01 15:32:26,349 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/root/.staging/job_1556738561291_0011
2019-05-01 15:32:26,896 INFO input.FileInputFormat: Total input files to process : 1
2019-05-01 15:32:27,383 INFO mapreduce.JobSubmitter: number of splits:1
2019-05-01 15:32:27,616 INFO Configuration.deprecation: yarn.resourcemanager.system-metrics-publisher.enabled is deprecated. Instead, use yarn.system-metrics-publisher.enabled
2019-05-01 15:32:28,261 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1556738561291_0011
2019-05-01 15:32:28,270 INFO mapreduce.JobSubmitter: Executing with tokens: []
2019-05-01 15:32:29,082 INFO conf.Configuration: resource-types.xml not found
2019-05-01 15:32:29,083 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2019-05-01 15:32:29,343 INFO impl.YarnClientImpl: Submitted application application_1556738561291_0011
2019-05-01 15:32:29,564 INFO mapreduce.Job: The url to track the job: http://Pony:8088/proxy/application_1556738561291_0011/
2019-05-01 15:32:29,566 INFO mapreduce.Job: Running job: job_1556738561291_0011
2019-05-01 15:33:05,042 INFO mapreduce.Job: Job job_1556738561291_0011 running in uber mode : false
2019-05-01 15:33:05,053 INFO mapreduce.Job: map 0% reduce 0%
2019-05-01 15:33:29,109 INFO mapreduce.Job: map 100% reduce 0%

```

Figura 5.20: Primera corrida del algoritmo ID3.

Posteriormente, cuando esta primera ejecución finalice nos dará un resultado parcial de la salida de la primera ejecución la cual para este tipo de corrida puede ser visualizada desde pantalla, como se muestra en la figura 5.23 los

cálculos intermedios y las escrituras de la salida de este job pueden ser visualizadas en las imágenes 5.21 y 5.22.

```

2019-05-01 15:33:29,109 INFO mapreduce.Job: map 100% reduce 0%
2019-05-01 15:33:52,000 INFO mapreduce.Job: map 100% reduce 100%
2019-05-01 15:33:53,091 INFO mapreduce.Job: Job job_1556738561291_0011 completed successfully
2019-05-01 15:33:53,618 INFO mapreduce.Job: Counters: 53
  File System Counters
    FILE: Number of bytes read=904
    FILE: Number of bytes written=430279
    FILE: Number of read operations=0
    FILE: Number of large read operations=0
    FILE: Number of write operations=0
    HDFS: Number of bytes read=588
    HDFS: Number of bytes written=821
    HDFS: Number of read operations=10
    HDFS: Number of large read operations=0
    HDFS: Number of write operations=4
  Job Counters
    Launched map tasks=1
    Launched reduce tasks=1
    Rack-local map tasks=1
    Total time spent by all maps in occupied slots (ms)=123918
    Total time spent by all reduces in occupied slots (ms)=114792
    Total time spent by all map tasks (ms)=20653
    Total time spent by all reduce tasks (ms)=19132
    Total vcore-milliseconds taken by all map tasks=20653
    Total vcore-milliseconds taken by all reduce tasks=19132
    Total megabyte-milliseconds taken by all map tasks=7930752
    Total megabyte-milliseconds taken by all reduce tasks=7346688
  Map-Reduce Framework
    Map input records=14
    Map output records=15
    Map output bytes=866

```

Figura 5.21: Pasos intermedios de la ejecución.

```

Map output bytes=866
Map output materialized bytes=904
Input split bytes=112
Combine input records=0
Combine output records=0
Reduce input groups=2
Reduce shuffle bytes=904
Reduce input records=15
Reduce output records=2
Spilled Records=30
Shuffled Maps =1
Failed Shuffles=0
Merged Map outputs=1
GC time elapsed (ms)=689
CPU time spent (ms)=5440
Physical memory (bytes) snapshot=339243008
Virtual memory (bytes) snapshot=4122796032
Total committed heap usage (bytes)=139640832
Peak Map Physical memory (bytes)=221372416
Peak Map Virtual memory (bytes)=2059370496
Peak Reduce Physical memory (bytes)=117870592
Peak Reduce Virtual memory (bytes)=2063425536
Shuffle Errors
  BAD_ID=0
  CONNECTION=0
  IO_ERROR=0
  WRONG_LENGTH=0
  WRONG_MAP=0
  WRONG_REDUCE=0
File Input Format Counters
  Bytes Read=393
File Output Format Counters
  Bytes Written=866

```

Figura 5.22: Pasos intermedios de la ejecución.

```

Merged Map outputs=1
GC time elapsed (ms)=742
CPU time spent (ms)=5670
Physical memory (bytes) snapshot=337752064
Virtual memory (bytes) snapshot=4122198016
Total committed heap usage (bytes)=139726848
Peak Map Physical memory (bytes)=219783168
Peak Map Virtual memory (bytes)=2058153984
Peak Reduce Physical memory (bytes)=117968896
Peak Reduce Virtual memory (bytes)=2064044032
Shuffle Errors
BAD_ID=0
CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0
File Input Format Counters
    Bytes Read=393
File Output Format Counters
    Bytes Written=795
acabo de agregar la fila0
la construccion aqui esif (General ==soleado)

traigo en posicion0
traigo en palabra buscada soleado
para[soleado]
voy a escribirsoleado,caliente,alta,no,N
voy a escribirsoleado,caliente,alta,si,N
voy a escribirsoleado,templada,alta,no,N
voy a escribirsoleado,fria,normal,no,P
voy a escribirsoleado,templada,normal,si,P
2019-05-01 15:33:54,516 INFO client.RMProxy: Connecting to ResourceManager at Pony/192.168

```

Figura 5.23: Primera corrida del algoritmo ID3.

por lo que una vez que se obtiene el primer resultado parcial se procede a la ejecución de los subsecuentes jobs de trabajo, estos se irán levantando uno a uno conforme vaya finalizando el anterior, cada vez que uno de estos finalice será posible visualizar su salida intermedia y con esto permitir que el job subsecuente inicie su ejecución estos resultados parciales de los que se habla se pueden visualizar en las figuras 5.24,5.25,5.26,5.27,5.28y 5.29. para cada una de ellas es posible visualizar la salida intermedia que se tenía en ese momento de la ejecución, la nueva inicialización del nuevo job para continuar y el detalle en el pie de imagen de que combinación de columnas se estaba evaluando en ese momento. Con lo que con las figuras presentadas se puede dar seguimiento a la evolución de la ejecución y como es que esta ejecución se va comportando con respecto a la recursividad.

```

BAD_ID=0
CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0
File Input Format Counters
    Bytes Read=135
File Output Format Counters
    Bytes Written=612
acabo de agregar la fila2
la construccion aqui esif (General ==soleado)
    if (Humedad ==normal)

traigo en posicion2
traigo en palabra buscada normal
para[soleado, normal]
voy a escribirsoleado,fria,normal,no,P
voy a escribirsoleado,tempizada,normal,si,P
2019-05-01 15:35:27,459 INFO client.RMProxy: Connecting to ResourceManager at Pony/192.168
.1.72:8032
2019-05-01 15:35:27,495 WARN mapreduce.JobResourceUploader: Hadoop command-line option par
sing not performed. Implement the Tool interface and execute your application with ToolRun
ner to remedy this.
2019-05-01 15:35:27,524 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for p
ath: /tmp/hadoop-yarn/staging/root/.staging/job_1556738561291_0013
2019-05-01 15:35:27,723 INFO input.FileInputFormat: Total input files to process : 1
2019-05-01 15:35:28,036 INFO mapreduce.JobSubmitter: number of splits:1
2019-05-01 15:35:28,062 INFO Configuration.deprecation: yarn.resourcemanager.system-metric
s-publisher.enabled is deprecated. Instead, use yarn.system-metrics-publisher.enabled
2019-05-01 15:35:28,668 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_155673
8561291_0013
2019-05-01 15:35:28,668 INFO mapreduce.JobSubmitter: Executing with tokens: []

```

Figura 5.24: Salidas intermedias para Soleado-Normal.

```

Peak Reduce Virtual memory (bytes)=2063622144
Shuffle Errors
BAD_ID=0
CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0
File Input Format Counters
    Bytes Read=54
File Output Format Counters
    Bytes Written=21
la construccion en salida esif (General ==soleado)
    if (Humedad ==normal)
Clase = P

la construccion aqui esif (General ==soleado)
    if (Humedad ==normal)
Clase = P
    if (Humedad ==alta)

traigo en posicion2
traigo en palabra buscada alta
para[soleado, alta]
voy a escribirsoleado,caliente,alta,no,N
voy a escribirsoleado,caliente,alta,si,N
voy a escribirsoleado,tempizada,alta,no,N
2019-05-01 15:37:01,564 INFO client.RMProxy: Connecting to ResourceManager at Pony/192.168
.1.72:8032
2019-05-01 15:37:01,589 WARN mapreduce.JobResourceUploader: Hadoop command-line option par
sing not performed. Implement the Tool interface and execute your application with ToolRun
ner to remedy this.

```

Figura 5.25: Salidas intermedias para Soleado-Alta.

```

IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0
File Input Format Counters
    Bytes Read=81
File Output Format Counters
    Bytes Written=21
la construccion en salida esif (General ==soleado)
    if (Humedad ==normal)
Clase = P
    if (Humedad ==alta)
Clase = N

acabo de borrar la fila1
la construccion aqui esif (General ==soleado)
    if (Humedad ==normal)
Clase = P
    if (Humedad ==alta)
Clase = N
    if (General ==lluvioso)

traigo en posicion0
traigo en palabra buscada lluvioso
para[lluvioso, alta]
voy a escribirlluvioso,templada,alta,no,P
voy a escribirlluvioso,fria,normal,no,P
voy a escribirlluvioso,fria,normal,si,N
voy a escribirlluvioso,templada,normal,no,P
voy a escribirlluvioso,templada,alta,si,N
2019-05-01 15:38:32,500 INFO client.RMProxy: Connecting to ResourceManager at Pony/192.168
.1.72:8032

```

Figura 5.26: Salidas intermedias para Lluvioso.

```

CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0
File Input Format Counters
    Bytes Read=138
File Output Format Counters
    Bytes Written=612
acabo de agregar la fila3
la construccion aqui esif (General ==soleado)
    if (Humedad ==normal)
Clase = P
    if (Humedad ==alta)
Clase = N
    if (General ==lluvioso)
        if (Viento ==no)

traigo en posicion3
traigo en palabra buscada no
para[lluvioso, no, ]
voy a escribirlluvioso,templada,alta,no,P
voy a escribirlluvioso,fria,normal,no,P
voy a escribirlluvioso,templada,normal,no,P
2019-05-01 15:40:01,789 INFO client.RMProxy: Connecting to ResourceManager at Pony/192.168
.1.72:8032
2019-05-01 15:40:01,826 WARN mapreduce.JobResourceUploader: Hadoop command-line option par
sing not performed. Implement the Tool interface and execute your application with ToolRun
ner to remedy this.
2019-05-01 15:40:01,859 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for p
ath: /tmp/hadoop-yarn/staging/root/.staging/job_1556738561291_0016
2019-05-01 15:40:02,012 INFO input.FileInputFormat: Total input files to process : 1

```

Figura 5.27: Salidas intermedias para Lluvioso-No.

```

        WRONG_LENGTH=0
        WRONG_MAP=0
        WRONG_REDUCE=0
    File Input Format Counters
        Bytes Read=84
    File Output Format Counters
        Bytes Written=21
la construccion en salida esif (General ==soleado)
    if (Humedad ==normal)
Clase = P
    if (Humedad ==alta)
Clase = N
    if (General ==lluvioso)
    if (Viento ==no)
Clase = P

la construccion aqui esif (General ==soleado)
    if (Humedad ==normal)
Clase = P
    if (Humedad ==alta)
Clase = N
    if (General ==lluvioso)
    if (Viento ==no)
Clase = P
    if (Viento ==si)

traigo en posicion3
traigo en palabra buscada si
para[lluvioso, si, ]
voy a escribirlluvioso,fria,normal,si,N
voy a escribirlluvioso,tempizada,alta,si,N
2019-05-01 15:41:32,417 INFO client.RMProxy: Connecting to ResourceManager at Pony/192.168
1.72:8032

```

Figura 5.28: Salidas intermedias para Lluvioso-Si.

```

        WRONG_LENGTH=0
        WRONG_MAP=0
        WRONG_REDUCE=0
    File Input Format Counters
        Bytes Read=84
    File Output Format Counters
        Bytes Written=21
la construccion en salida esif (General ==soleado)
    if (Humedad ==normal)
Clase = P
    if (Humedad ==alta)
Clase = N
    if (General ==lluvioso)
    if (Viento ==no)
Clase = P

la construccion aqui esif (General ==soleado)
    if (Humedad ==normal)
Clase = P
    if (Humedad ==alta)
Clase = N
    if (General ==lluvioso)
    if (Viento ==no)
Clase = P
    if (Viento ==si)

traigo en posicion3
traigo en palabra buscada si
para[lluvioso, si, ]
voy a escribirlluvioso,fria,normal,si,N
voy a escribirlluvioso,tempizada,alta,si,N
2019-05-01 15:41:32,417 INFO client.RMProxy: Connecting to ResourceManager at Pony/192.168
1.72:8032

```

Figura 5.29: Salidas intermedias para Nublado.

Al finalizar todos los puntos intermedios, finalmente se puede llegar a la última ejecución la cual finaliza el proceso y termina, esto es cuando termina de realizar todas las evaluaciones de todas las columnas obtenidas durante la función recursiva.

Debido a que cada que llega una nueva columna se va a agregando a un arreglo de columnas, y cada que una columna termina de ser evaluada en su totalidad esta se elimina del arreglo de columnas, cuando la última fila de este arreglo es eliminada es decir, la fila 0. Quiere decir que la ejecución termino exitosamente como se puede observar en la figura

5.30 donde se puede ver que ya se imprime la salida final para este ejemplo y además se puede visualizar como se elimina la fila 0.

```

Virtual memory (bytes) snapshot=4121772032
Total committed heap usage (bytes)=139714560
Peak Map Physical memory (bytes)=209952768
Peak Map Virtual memory (bytes)=2058153984
Peak Reduce Physical memory (bytes)=118099968
Peak Reduce Virtual memory (bytes)=2063618048
Shuffle Errors
BAD_ID=0
CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0
File Input Format Counters
Bytes Read=108
File Output Format Counters
Bytes Written=21
la construccion en salida esif (General ==soleado)
    if (Humedad ==normal)
Clase = P
    if (Humedad ==alta)
Clase = N
        if (General ==lluvioso)
            if (Viento ==no)
Clase = P
            if (Viento ==si)
Clase = N
        if (General ==nublado)
Clase = P
acabo de borrar la fila0

```

Figura 5.30: Finaliza Ejecución de algoritmo ID3

Cuando la ejecución termina, entonces se puede visualizar el archivo de salidas establecido para la ejecución en el HDFS como se muestra en la figura 5.31, cabe señalar, que este archivo únicamente contiene la información de la primera ejecución de MapReduce debido a que cada ejecución requiere un nuevo directorio de salidas.

□	drwxr-xr-x	root	supergroup	0 B	May 01	0	0 B	salidaID3Documento	●
---	------------	------	------------	-----	--------	---	-----	--------------------	---

Figura 5.31: Visualización de archivo de salidas de linea de comandos(Primera ejecución en el HDFS)

Sin embargo, si se procede a visualizar el interior de este directorio en el archivo *part-00000* se encuentra la información contenida en la figura 5.32 la cual contiene toda la información relacionada con la primera corrida, se puede visualizar los cálculos de las entropías y las ganancias realizados y finalmente se puede conocer la fila que se determinó con la mayor ganancia y los elemento que esta fila contiene.

```

la entropia general del problema es:0.9402859586706309
la entropia especifica0.9709505944546687
la entropia especifica0.9709505944546687
la entropia especificaNaN
la entropia fila0.6935361388961919
la ganancia fila0.24646386110380802
la entropia especifica0.8112781244591328
la entropia especifica1.0
la entropia especifica0.9182958340544894
la entropia fila0.9110633930116763
la ganancia fila0.028936606988323677
la entropia especifica0.5916727785823274
la entropia especifica0.9852281360342515
la entropia fila0.7884504573082894
la ganancia fila0.1515495426917105
la entropia especifica0.8112781244591328
la entropia especifica1.0
la entropia fila0.8921589282623617
la ganancia fila0.0478410717376383
la fila con la mayor ganancia es0
Datos:soleado,lluvioso,nublado,0          0

```

Figura 5.32: Salida del archivo Part-00000

Sin embargo, esta no constituye la totalidad de la ejecución del algoritmo ya que como se mencionaba cada ejecución genera su propia carpeta de salida.

Para este caso en particular, la ruta donde estos directorios fueron escritos es:

/user/luminus/ID3/pruebasjob

La información contenida en este directorio puede ser visualizada en la figura 5.33

Browse Directory								
<input type="text" value="/user/luminus/ID3/pruebasjob"/> <input type="button" value="Go!"/> <input type="button" value="New"/> <input type="button" value="Upload"/> <input type="button" value="Download"/>								
Show <input type="button" value="25"/> entries <input type="text" value="Search:"/> <input type="button" value="Search"/>								
□	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
□	drwxr-xr-x	root	supergroup	0 B	May 01 15:35	0	0 B	carpetaj0
□	drwxr-xr-x	root	supergroup	0 B	May 01 15:36	0	0 B	carpetaj1
□	drwxr-xr-x	root	supergroup	0 B	May 01 15:38	0	0 B	carpetaj2
□	drwxr-xr-x	root	supergroup	0 B	May 01 15:39	0	0 B	carpetaj3
□	drwxr-xr-x	root	supergroup	0 B	May 01 15:41	0	0 B	carpetaj4
□	drwxr-xr-x	root	supergroup	0 B	May 01 15:42	0	0 B	carpetaj5
□	drwxr-xr-x	root	supergroup	0 B	May 01 15:44	0	0 B	carpetaj6

Figura 5.33: Directorio de salidas de HDFS para trabajos posteriores

Siguiendo la misma lógica de funcionamiento esperada para el primer directorio, se procede a visualizar la información de salida contenida en cada uno de los directorios y se dará una breve información del porque se presenta esta información.

La primera salida se muestra en la figura 5.34 esta corresponde al cálculo de ganancias y entropias que dan como resultado la fila con mayor ganancia 2.

Esto ocurre cuando se ejecuta el algoritmo tomando en cuenta únicamente las entradas de la tabla donde en la columna

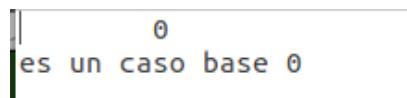
0 poseen el valor de soleado.

Como se requiere aproximar un poco más y probar con una nueva fila, se realiza entonces este cálculo.

```
|      0
la entropia general del problema es:0.9709505944546687
la entropia especifica0.9709505944546687
la entropia fila0.9709505944546686
la ganancia fila-0.030950594454668634
la entropia especificaNaN
la entropia especificaNaN
la entropia especifica1.0
la entropia fila0.4
la ganancia fila0.5399999999999999
la entropia especificaNaN
la entropia especificaNaN
la entropia fila0.0
la ganancia fila0.94
la entropia especifica0.9182958340544894
la entropia especifica1.0
la entropia fila0.9509775004326937
la ganancia fila-0.010977500432693743
la fila con la mayor ganancia es2
Datos:normal,alta,2      0
```

Figura 5.34: Salida de carpetaj0

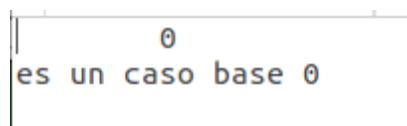
La salida subsecuente se muestra en la figura 5.35. Y debido a que se establece que Soleado y Normal, esto es todo lo que se necesita para hacer una clasificación entonces se llega a un caso base y únicamente se indica en este archivo.



```
|      0
es un caso base 0
```

Figura 5.35: Salida de carpetaj1

La siguiente salida se muestra en la figura 5.36. Esta también constituye un caso base y esto se debe a una situación muy parecida a la de la figura 5.35 ya que en este caso Soleado y Alta también constituyen un caso base.



```
|      0
es un caso base 0
```

Figura 5.36: Salida de carpetaj2

Para la siguiente ejecución se procede a trabajar con Lluvioso, el cual únicamente de esta forma no genera un caso pase por lo que se procede a realizar nuevamente los cálculos pertinentes para obtener una nueva columna a evaluar, de acuerdo a la figura 5.37 se puede ver que se obtiene que la fila con mayor ganancia es la 3 y que los valores que esta fila contiene es NO y SI.

```

0
la entropia general del problema es:0.9709505944546687
la entropia especifica0.9709505944546687
la entropia fila0.9709505944546686
la ganancia fila-0.03095059445466834
la entropia especifica1.0
la entropia especifica0.9182958340544894
la entropia fila0.9509775004326937
la ganancia fila-0.010977500432693743
la entropia especifica0.9182958340544894
la entropia especifica1.0
la entropia fila0.9509775004326937
la ganancia fila-0.010977500432693743
la entropia especificaNaN
la entropia especificaNaN
la entropia fila0.0
la ganancia fila0.94
la fila con la mayor ganancia es3
Datos:no,si,3 0

```

Figura 5.37: Salida de carpetaj3

Debido a que la respuesta final para la siguiente evaluación es Lluvioso- No, entonces no se requiere realizar nuevamente el cálculo de ganancias y entropías por lo que el siguiente archivo el cual se muestra en la figura 5.38 representa un caso base.

```

| 0
es un caso base 0

```

Figura 5.38: Salida de carpetaj4

Un caso muy similar ocurre con la siguiente entrada y el siguiente archivo el cual se muestra en la figura 5.39 debido a que Lluvioso- Si representa un caso base, la información descrita en este archivo describe ese detalle.

```

| 0
es un caso base 0

```

Figura 5.39: Salida de carpetaj5

Una vez que se termina de efectuar la evaluación correspondiente a la columna 3, esta se retira del listado de columnas y se regresa a la columna 0, debido a que en la columna 0 aun se encuentra el valor de Nublado, se procede a realizar la evaluación de este valor, sin embargo desde este momento ya se trata de un elemento clasificado por lo que, se trata de un caso base como se muestra en la figura 5.40.

```

| 0
es un caso base 0

```

Figura 5.40: Salida de carpetaj6

Como último paso se procede a presentar en la figura 5.41 el archivo final de salidas para la ejecución de este algoritmo.

```
if (General ==soleado)
    if (Humedad ==normal)
        Clase = P
    if (Humedad ==alta)
        Clase = N
if (General ==lluvioso)
    if (Viento ==no)
        Clase = P
    if (Viento ==si)
        Clase = N
if (General ==nublado)
    Clase = P
```

Figura 5.41: Salida final del algoritmo ID3 para este ejemplo

La cual contiene la información, que se despliega desde línea de comandos presentada en la figura 5.30 pero ya en un archivo permanente para consulta y uso por parte del usuario final. Se encuentra con las identaciones correspondientes para que se comprenda a que nivel corresponde cada columna evaluada. Con esto se puede afirmar que los cálculos se llevaron a cabo de manera exitosa y que el algoritmo ID3 funciona correctamente.

CAPÍTULO 6

Interface de programación de aplicaciones de Luminus

6.1. Descripción del prototipo

Esta interfaz de programación de aplicaciones provee acceso al sistema de archivos distribuído de *Hadoop*, tanto para operaciones de lectura como de escritura. Esta herramienta permite la interacción del programador con la configuración y ejecución de los algoritmos de minería de datos implementados en *Luminus*. Así mismo, permite a cualquier programador Java añadir sus propios algoritmos programados con el paradigma MapReduce.

6.2. Análisis

Inicialmente se tenía contemplado hacer uso de un sitio web para presentar los resultados de la ejecución de los algoritmos de minería de datos que contiene *Luminus*. Sin embargo, durante fases posteriores del proyecto, se tomó la decisión de que sería de mayor utilidad presentarlos por medio de una interfaz de programación de aplicaciones (API). De esta forma *Luminus* se vuelve una herramienta reutilizable y mucho más versátil.

6.2.1. Casos de uso

A continuación se muestran los casos de uso encontrados para este aplicativo

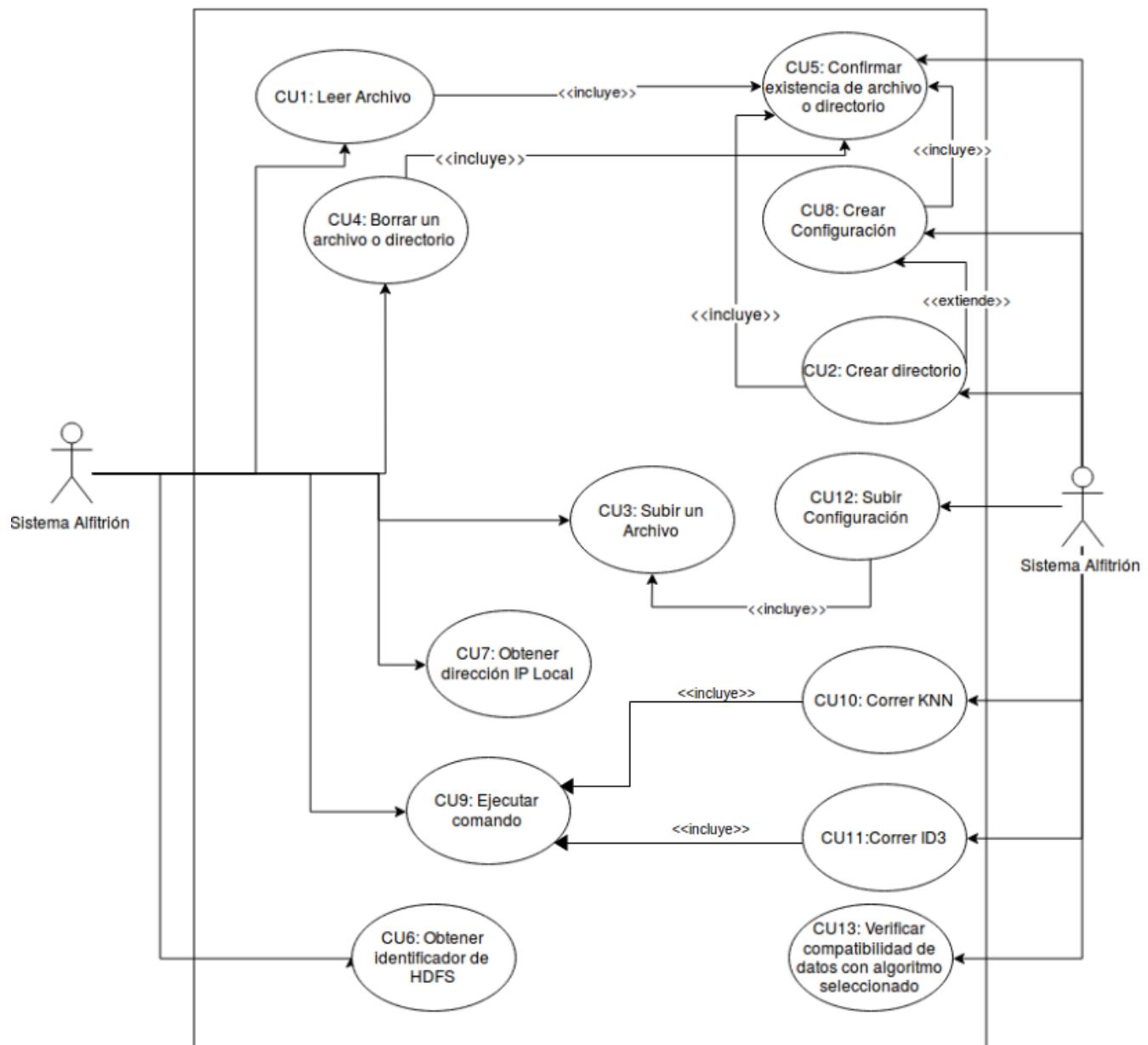


Figura 6.1: Diagrama de casos de uso

6.3. CU1 Leer archivo

6.3.1. Resumen



Lee un archivo del sistema de archivos distribuidos a un stream especificado.

6.3.2. Descripción

Caso de Uso:	CU1 Leer archivo
Versión:	1.0
Autor:	Mayra Patricia Tovar Barba y Kevin Raúl Monteón Valdes.
Supervisa:	Ulises Vélez Saldaña y Alejandro Botello Castillo.
Actor:	Usuario Experto, Sistema.
Propósito:	Obtener el contenido de un archivo del HDFS.
Entradas:	Ruta de archivo, Stream de salida, Identificador de HDFS, Dirección IP del Nodo Maestro.
Origen:	Código que implementa la API.
Salidas:	Ninguna.
Destino:	Stream de salida.
Precondiciones:	Debe existir el archivo de la ruta especificada. El HDFS es accesible por la red.
Postcondiciones:	Los bytes del archivo habrán sido enviado a través del stream de salida.
Errores:	LuminusException(1) MSG1 No existe el archivo o directorio , IOException.
Tipo:	Caso de uso primario.
Observaciones:	

6.3.3. Trayectorias del Caso de Uso

Trayectoria principal

- 1 ⚙ Invoca el método `getDataFromFile`.
 - 2 ⚙ Envía [Ruta del Archivo].
 - 3 ⚙ Envía [Stream de Salida].
 - 4 ⚙ Envía [Identificador de HDFS].
 - 5 ⚙ Envía [Dirección IP del Nodo Maestro].
 - 6 ○ Realiza una conexión al nodo maestro del HDFS por medio de la [Dirección IP del Nodo Maestro].
 - 7 ○ Verifica que la [Ruta del Archivo] exista. **[Trayectoria A]**
 - 8 ○ Copia en el [Stream de Salida] la información contenida en el archivo. **[Trayectoria B]**
 - 9 ○ Cierra la conexión al nodo maestro del HDFS creada en el paso 6.
 - 10 Termina el caso de uso.
- - - - *Fin del caso de uso.*

Trayectoria alternativa A:

Condición: Archivo o directorio no encontrado

- A-1** ○ Manda el Mensaje **MSG1 No existe el archivo o directorio**.
 - A-2** Termina el caso de uso.
- - - - *Fin de la trayectoria.*

Trayectoria alternativa B:

Condición: *Error al copiar*

B-1  Manda el Mensaje con la información proporcionada por Java acerca del error de escritura o lectura.

B-2 Termina el caso de uso.

- - - - *Fin de la trayectoria.*

6.4. CU2 Crear directorio

6.4.1. Resumen

Crea un directorio en el sistema de archivos distribuido de Hadoop (HDFS).



6.4.2. Descripción

Caso de Uso:	CU2 Crear directorio
Versión:	0.1
Autor:	Mayra Patricia Tovar Barba y Kevin Raúl Monteón Valdes
Supervisa:	Ulises Vélez Saldaña y Alejandro Botello Castillo
Actor:	Usuario Experto, Sistema.
Propósito:	Crear un directorio en el HDFS.
Entradas:	Ruta de directorio, Identificador de HDFS, Dirección IP del Nodo Maestro.
Origen:	Código fuente
Salidas:	Nuevo directorio en el HDFS.
Destino:	Ninguno.
Precondiciones:	No debe existir el directorio en la [Ruta del directorio] especificada. El HDFS es accesible por la red.
Postcondiciones:	Se crea un nuevo directorio en la [Ruta del directorio] especificada.
Errores:	LuminusException(3) MSG3 El directorio ya existe
Tipo:	Caso de uso primario.
Observaciones:	

6.4.3. Trayectorias del Caso de Uso

Trayectoria principal

- 1 ⚡ Invoca al método makeHDFSDirectory
 - 2 ⚡ Envía [Ruta de directorio].
 - 3 ⚡ Envía [Identificador de HDFS].
 - 4 ⚡ Envía [Dirección IP del Nodo Maestro].
 - 5 ○ Realiza una conexión al nodo maestro del HDFS por medio del [Identificador de HDFS]
 - 6 ○ Verifica que la [Ruta del directorio] no exista **[Trayectoria A]**
 - 7 ○ Crea un nuevo directorio en la [Ruta del directorio] especificada.
 - 8 ○ Cierra la conexión al nodo maestro del HDFS creada en el paso 5
 - 9 Termina el caso de uso.
- - - - *Fin del caso de uso.*

Trayectoria alternativa A:

Condición: Directorio ya existe

- A-1** ○ Manda el Mensaje MSG3 El directorio ya existe.

- A-2** Termina el caso de uso.

- - - - *Fin de la trayectoria.*

6.4.4. Puntos de extensión

PE El directorio que se desea crear corresponde a un directorio de configuración : Del paso 7 al paso 9. .

Región: .

Extiende a: CU8 Crear Configuración. .

6.5. CU3 Subir un archivo

6.5.1. Resumen



Copia un archivo local al sistema de archivos distribuido de Hadoop (HDFS).

6.5.2. Descripción

Caso de Uso:	CU3 Subir un archivo
Versión:	0.1
Autor:	Mayra Patricia Tovar Barba y Kevin Raúl Monteón Valdes
Supervisa:	Ulises Vélez Saldaña y Alejandro Botello Castillo.
Actor:	Usuario Experto, Sistema.
Propósito:	Sube un archivo al HDFS.
Entradas:	Ruta de archivo local, Ruta de archivo remoto, Identificador de HDFS, Bandera de Sobrescribir.
Origen:	Código fuente
Salidas:	Nuevo archivo en el HDFS.
Destino:	Ninguno.
Precondiciones:	El HDFS es accesible por la red.
Postcondiciones:	Se crea o actualiza un archivo en la [Ruta de archivo remoto] especificada.
Errores:	LuminusException(3)MSG3 El directorio ya existe
Tipo:	Caso de uso primario.
Observaciones:	

6.5.3. Trayectorias del Caso de Uso

Trayectoria principal

- 1 ⚙ Invoca el método uploadFileToHDFS
- 2 ⚙ Envía [Ruta de archivo local].
- 3 ⚙ Envía [Ruta de archivo remoto].
- 4 ⚙ Envía [Identificador de HDFS].
- 5 ⚙ Envía [Bandera de Sobrescribir].
- 6 ○ Realiza una conexión al nodo maestro del HDFS por medio del [Identificador de HDFS].
- 7 ○ Verifica que el archivo en la [Ruta de archivo remoto] no exista. [Trayectoria A] [Trayectoria B]
- 8 ○ Copia el archivo local de la [Ruta de archivo local] al archivo remoto de la [Ruta de archivo remoto].
- 9 ○ Cierra la conexión al nodo maestro del HDFS creada en el paso 6.
- 10 Termina el caso de uso.
- - - Fin del caso de uso.

Trayectoria alternativa A:

Condición: Sobrescribir archivo

- A-1 ○ Verifica que la [Bandera de Sobrescribir] esta activada.
 - A-2 Reanuda el caso de uso CU3 en el paso 8.
- - - Fin de la trayectoria.

Trayectoria alternativa B:

Condición: No sobrescribir archivo

B-1  Verifica que la [Bandera de Sobrescribir] esta desactivada.

B-2  Manda el Mensaje **MSG3** El directorio ya existe.

B-3 Termina el caso de uso.

- - - - *Fin de la trayectoria.*

6.6. CU4 Borrar un archivo o directorio

6.6.1. Resumen



Elimina un archivo o directorio del sistema de archivos distribuido (HDFS).

6.6.2. Descripción

Caso de Uso:	CU4 Borrar un archivo o directorio
Versión:	0.1
Autor:	Mayra Patricia Tovar Barba y Kevin Raúl Monteón Valdes
Supervisa:	Ulises Vélez Saldaña y Alejandro Botello Castillo.
Actor:	Usuario Experto, Sistema.
Propósito:	Elimina un archivo o directorio del HDFS.
Entradas:	Ruta de archivo o directorio, Identificador de HDFS
Origen:	Código fuente.
Salidas:	El archivo borrado ya no existe más en el HDFS.
Destino:	Ninguno.
Precondiciones:	El archivo o directorio en la [Ruta de archivo o directorio] debe existir en el HDFS. El HDFS es accesible por la red.
Postcondiciones:	Se elimina el archivo o directorio en la [Ruta de archivo o directorio].
Errores:	LuminusException(1)MSG1 No existe el archivo o directorio.
Tipo:	Caso de uso primario.
Observaciones:	

6.6.3. Trayectorias del Caso de Uso

Trayectoria principal

- 1 ⚙ Invoca el método deleteFromHDFS
 - 2 ⚙ Envía [Ruta de archivo o directorio].
 - 3 ⚙ Envía [Identificador de HDFS].
 - 4 ○ Realiza una conexión al nodo maestro del HDFS por medio del [Identificador de HDFS]
 - 5 ○ Verifica que el archivo o directorio en la[Ruta de archivo o directorio] exista. **[Trayectoria A]**
 - 6 ○ Elimina el archivo o directorio de la [Ruta de archivo o directorio].
 - 7 ○ Cierra la conexión al nodo maestro del HDFS creada en el paso 4.
 - 8 Termina el caso de uso.
- - - - *Fin del caso de uso.*

Trayectoria alternativa A:

Condición: No existe el archivo o directorio

- A-1** ○ Manda el Mensaje MSG1 No existe el archivo o directorio.
- A-2** Termina el caso de uso.
- - - - *Fin de la trayectoria.*

6.7. CU5 Confirmar existencia de archivo o directorio

6.7.1. Resumen

Confirma la existencia de un archivo o un directorio en un sistema de archivos distribuidos



6.7.2. Descripción

Caso de Uso:	CU5 Confirmar existencia de archivo o directorio
Versión:	0.1
Autor:	Mayra Patricia Tovar Barba y Kevin Raúl Monteón Valdes
Supervisa:	Ulises Vélez Saldaña y Alejandro Botello Castillo.
Actor:	Usuario Experto, Sistema.
Propósito:	Confirma la existencia de un archivo o directorio.
Entradas:	Ruta de archivo o directorio, Identificador de HDFS
Origen:	Código que implementa la API.
Salidas:	Existe, No Existe
Destino:	Ninguno.
Precondiciones:	El HDFS es accesible por la red.
Postcondiciones:	Se confirmo si el archivo o directorio en la [Ruta de archivo o directorio] existe.
Errores:	IllegalArgumentException, IOException.
Tipo:	Caso de uso primario.
Observaciones:	

6.7.3. Trayectorias del Caso de Uso

Trayectoria principal

- 1 ⚙ Invoca el método existsInHDFS
 - 2 ⚙ Envía [Ruta de archivo o directorio].
 - 3 ⚙ Envía [Identificador de HDFS].[Trayectoria C]
 - 4 ○ Realiza una conexión al nodo maestro del HDFS por medio del [Identificador de HDFS]
 - 5 ○ Consulta al archivo o directorio en la[Ruta de archivo o directorio]. [Trayectoria A] [Trayectoria B]
- - - - Fin del caso de uso.

Trayectoria alternativa A:

Condición: No existe el archivo o directorio

- A-1** ○ Se devuelve el valor booleano *false*.
 - A-2** ○ Cierra la conexión al nodo maestro del HDFS creada en el paso 4.
 - A-3** Termina el caso de uso.
- - - - Fin de la trayectoria.

Trayectoria alternativa B:

Condición: Existe el archivo o directorio

- B-1** ○ Se devuelve el valor booleano *true*.
- B-2** ○ Cierra la conexión al nodo maestro del HDFS creada en el paso 4.

B-3 Termina el caso de uso.

- - - - *Fin de la trayectoria.*

Trayectoria alternativa C:

Condición: *Los argumentos recibidos no son válidos*

C-1  Se devuelve el mensaje con la información proporcionada por Java para notificar que los argumentos recibidos no son válidos

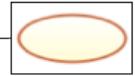
C-2 Termina el caso de uso.

- - - - *Fin de la trayectoria.*

6.8. CU6 Obtener Identificador de HDFS

6.8.1. Resumen

Obtiene el Identificador de HDFS a partir de una Dirección IP de Nodo Maestro.



6.8.2. Descripción

Caso de Uso:	CU6 Obtener Identificador de HDFS
Versión:	0.1
Autor:	Mayra Patricia Tovar Barba y Kevin Raúl Monteón Valdes
Supervisa:	Ulises Vélez Saldaña y Alejandro Botello Castillo.
Actor:	Usuario Experto, Sistema.
Propósito:	Recupera el Identificador de HDFS.
Entradas:	Dirección IP de Nodo Maestro.
Origen:	Código que implementa la API
Salidas:	Ninguna.
Destino:	Ninguno.
Precondiciones:	El nodo maestro es accesible por la red.
Postcondiciones:	Se obtiene un identificador del HDFS.
Errores:	Ninguno.
Tipo:	Caso de uso primario.
Observaciones:	

6.8.3. Trayectorias del Caso de Uso

Trayectoria principal

- 1 ⚡ Invoca el método getFileSystem
 - 2 ⚡ Envía [Dirección IP del nodo maestro].
 - 3 ○ Crea una conexión al nodo maestro del HDFS.
 - 4 ○ Obtiene un objeto identificador del sistema de archivo.
 - 5 ○ Retorna el identificador del sistema de archivos de tipo FileSystem obtenido en el paso 4.
 - 6 ○ Cierra la conexión al nodo maestro del HDFS creada en el paso 3.
- - - - *Fin del caso de uso.*

Trayectoria alternativa A:

Condición: No existe el archivo o directorio

- A-1** ○ Manda el Mensaje no existe MSG1 No existe el archivo o directorio.
- A-2** Termina el caso de uso.
- - - - *Fin de la trayectoria.*

6.9. CU7 Obtener Dirección IP Local

6.9.1. Resumen

Obtiene la Dirección IP Local asignada a la interfaz de red por defecto.



6.9.2. Descripción

Caso de Uso:	CU7 Obtener Dirección IP Local
Versión:	0.1
Autor:	Mayra Patricia Tovar Barba y Kevin Raúl Monteón Valdes
Supervisa:	Ulises Vélez Saldaña y Alejandro Botello Castillo.
Actor:	Usuario Experto, Sistema.
Propósito:	Recupera la Dirección IP Local.
Entradas:	Ninguna.
Origen:	Código fuente.
Salidas:	Cadena que contiene [Dirección IP Local].
Destino:	Ninguno.
Precondiciones:	Ninguna.
Postcondiciones:	Se conoce la [Dirección IP Local].
Errores:	UnknownHostException.
Tipo:	Caso de uso primario.
Observaciones:	

6.9.3. Trayectorias del Caso de Uso

Trayectoria principal

- 1 ⚖ Invoca el método getLocalNetAddress
- 2 ○ Recupera la [Dirección IP Local] **[Trayectoria A]**
- 3 ○ Retorna una cadena con el valor de la [Dirección IP Local]
- 4 Termina el caso de uso.
- - - *Fin del caso de uso.*

Trayectoria alternativa A:

Condición: Error al Recuperar Dirección IP Local

- A-1** ○ Manda el Mensaje con la información proporcionada por Java acerca del error de recuperación de la [Dirección IP Local].
- A-2** Termina el caso de uso.
- - - *Fin de la trayectoria.*

6.10. CU8 Crear configuración

6.10.1. Resumen

Crea un objeto de configuración cuyas propiedades son guardadas en un archivo dentro del sistema de archivos local.

6.10.2. Descripción

Caso de Uso:	CU8 Crear configuración
Versión:	0.1
Autor:	Mayra Patricia Tovar Barba y Kevin Raúl Monteón Valdes
Supervisa:	Ulises Vélez Saldaña y Alejandro Botello Castillo.
Actor:	Usuario Experto, Sistema.
Propósito:	Generar un objeto de configuración de tipo LuminusConfiguration.
Entradas:	Nombre del archivo de configuración.
Origen:	Código fuente.
Salidas:	Archivo con configuración.
Destino:	Ninguno.
Precondiciones:	Se tiene que haber creado una instancia de tipo LuminusConfiguration con la configuración deseada.
Postcondiciones:	Las propiedades del archivo de configuracion generadas son escritas en un archivo dentro del sistema de archivos local.
Errores:	LuminusException(5) MSG5 Extensión del archivo de configuración inválida.
Tipo:	Caso de uso primario.
Observaciones:	

6.10.3. Trayectorias del Caso de Uso

Trayectoria principal

- 1  Invoca el método createConfiguration.
 - 2  Envía el nombre del archivo de configuración.
 - 3  Valida que el nombre del archivo contenga la extensión ".properties". **[Trayectoria A]**
 - 4  Crea el archivo "configuration.properties" localmente y genera su contenido basado en variables previamente definidas.
 - 5 Termina el caso de uso.
- - - - *Fin del caso de uso.*

Trayectoria alternativa A:

Condición: *Extensión del archivo inválida.*

- A-1**  Se arroja una excepción de tipo LuminusException(5) **MSG5 Extensión del archivo de configuración inválida.**
- A-2** Termina el caso de uso.
- - - - *Fin de la trayectoria.*

6.11. CU9 Ejecutar Comando

6.11.1. Resumen

Ejecutar un comando en el shell de ubuntu.



6.11.2. Descripción

Caso de Uso:	CU9 Ejecutar Comando
Versión:	0.1
Autor:	Mayra Patricia Tovar Barba y Kevin Raúl Monteón Valdes
Supervisa:	Ulises Vélez Saldaña y Alejandro Botello Castillo.
Actor:	Usuario Experto, Sistema.
Propósito:	Enviar comando a proceso en ejecucion.
Entradas:	Comando en forma de cadena de caracteres.
Origen:	Código fuente.
Salidas:	De acuerdo al comando.
Destino:	Ninguno.
Precondiciones:	Debe existir el proceso.
Postcondiciones:	El proceso recibe el comando y lo ejecuta apropiadamente.
Errores:	IOException.
Tipo:	Caso de uso primario.
Observaciones:	

6.11.3. Trayectorias del Caso de Uso

Trayectoria principal

- 1 ⚙ Invoca el método executeCommand.
- 2 ⚙ Envía el comando.
- 3 💼 Obtiene la instancia del proceso en ejecución.
- 4 💼 Utiliza el método del proceso para comunicación de comandos.
- 5 💼 Muestra salida del comando si hay alguna.
- 6 💼 Muestra errores del comando si hay alguno.
- 7 Termina el caso de uso.
- - - *Fin del caso de uso.*

6.12. CU10 Correr KNN

6.12.1. Resumen

Ejecuta el algoritmo KNN sobre los datos de entrada especificados por el usuario experto. Escribe una salida en la ruta especificada por el usuario experto.

6.12.2. Descripción

Caso de Uso:	CU10 Correr KNN
Versión:	0.1
Autor:	Mayra Patricia Tovar Barba y Kevin Raúl Monteón Valdes
Supervisa:	Ulises Vélez Saldaña y Alejandro Botello Castillo.
Actor:	Usuario Experto, Sistema.
Propósito:	Ejecutar algoritmo de KNN.
Entradas:	Ruta del archivo de datos de entrada en el HDFS, ruta del directorio de salida del algoritmo en el HDFS.
Origen:	Código fuente.
Salidas:	Archivo de resultados de KNN.
Destino:	Ninguno.
Precondiciones:	Debe existir una configuración valida en el HDFS.
Postcondiciones:	Se genera un archivo con los resultados de la ejecución en el HDFS.
Errores:	LuminusException(1) MSG1 El archivo o directorio no existe, LuminusException(3) MSG3 El directorio ya existe
Tipo:	Caso de uso primario.
Observaciones:	

6.12.3. Trayectorias del Caso de Uso

Trayectoria principal

- 1 ⚙ Invoca el método runKNN.
 - 2 ⚙ Envía la ruta del archivo de datos de entrada en el HDFS.
 - 3 ⚙ Envía la ruta del directorio de salida en el HDFS.
 - 4 ○ Verifica que la ruta del archivo de datos de entrada no exista en el HDFS. [Trayectoria A]
 - 5 ○ Verifica que la ruta del directorio de salida no exista en el HDFS. [Trayectoria B]
 - 6 ○ Invoca al método executeCommand y le envía las rutas de entrada y salida.
 - 7 ○ Escribe la salida en el directorio de salida especificado.
 - 8 Termina el caso de uso.
- - - - *Fin del caso de uso.*

Trayectoria alternativa A:

Condición: El archivo de datos de entrada no se encuentra en el HDFS.

- A-1** ○ Se arroja una excepción de tipo LuminusException(1) MSG1 El archivo o directorio no existe.

- A-2** Termina el caso de uso.

- - - - *Fin de la trayectoria.*

Trayectoria alternativa B:

Condición: *El directorio de salida del algoritmo ya existe.*

B-1  Se arroja una excepción de tipo LuminusException(3) **MSG3 El directorio ya existe.**

B-2 Termina el caso de uso.

- - - - *Fin de la trayectoria.*

6.13. CU11 Correr ID3

6.13.1. Resumen

Ejecuta el algoritmo ID3 sobre los datos de entrada especificados por el usuario experto. Escribe una salida en la ruta especificada por el usuario experto.

6.13.2. Descripción

Caso de Uso:	CU11 Correr ID3
Versión:	0.1
Autor:	Mayra Patricia Tovar Barba y Kevin Raúl Monteón Valdes
Supervisa:	Ulises Vélez Saldaña y Alejandro Botello Castillo.
Actor:	Usuario Experto, Sistema.
Propósito:	Ejecutar algoritmo de ID3.
Entradas:	Ruta del archivo de datos de entrada en el HDFS, ruta del directorio de salida del algoritmo en el HDFS.
Origen:	Código fuente.
Salidas:	Archivo de resultados de ID3.
Destino:	Ninguno.
Precondiciones:	Debe existir una configuración valida en el HDFS.
Postcondiciones:	Se genera un archivo con los resultados de la ejecución en el HDFS.
Errores:	LuminusException(1) MSG1 El archivo o directorio no existe, LuminusException(3) MSG3 El directorio ya existe
Tipo:	Caso de uso primario.
Observaciones:	

6.13.3. Trayectorias del Caso de Uso

Trayectoria principal

- 1 ⚙ Invoca el método runID3.
 - 2 ⚙ Envía la ruta del archivo de datos de entrada en el HDFS.
 - 3 ⚙ Envía la ruta del directorio de salida en el HDFS.
 - 4 ○ Verifica que la ruta del archivo de datos de entrada no exista en el HDFS. [Trayectoria A]
 - 5 ○ Verifica que la ruta del directorio de salida no exista en el HDFS. [Trayectoria B]
 - 6 ○ Invoca al método executeCommand y le envía las rutas de entrada y salida.
 - 7 ○ Escribe la salida en el directorio de salida especificado.
 - 8 Termina el caso de uso.
- - - - *Fin del caso de uso.*

Trayectoria alternativa A:

Condición: El archivo de datos de entrada no se encuentra en el HDFS.

- A-1** ○ Se arroja una excepción de tipo LuminusException(1) MSG1 El archivo o directorio no existe.

- A-2** Termina el caso de uso.

- - - - *Fin de la trayectoria.*

Trayectoria alternativa B:

Condición: *El directorio de salida del algoritmo ya existe.*

B-1  Se arroja una excepción de tipo LuminusException(3) **MSG3 El directorio ya existe.**

B-2 Termina el caso de uso.

- - - - *Fin de la trayectoria.*

6.14. CU12 Subir Configuración

6.14.1. Resumen

Sube el archivo de configuración generado localmente al sistema de archivos distribuidos.



6.14.2. Descripción

Caso de Uso:	CU12 Subir Configuración
Versión:	0.1
Autor:	Mayra Patricia Tovar Barba y Kevin Raúl Monteón Valdes
Supervisa:	Ulises Vélez Saldaña y Alejandro Botello Castillo.
Actor:	Usuario Experto, Sistema.
Propósito:	Subir configuración a HDFS.
Entradas:	Ninguna.
Origen:	Código fuente.
Salidas:	Ninguna.
Destino:	Ninguno.
Precondiciones:	Debe haberse creado ya la configuración haciendo uso del caso de uso CU8 Crear Configuración
Postcondiciones:	Se genera una copia del archivo de configuración en el sistema de archivos distribuido.
Errores:	LuminusException(6) MSG6 Ruta del archivo de configuración no valida.
Tipo:	Caso de uso primario.
Observaciones:	

6.14.3. Trayectorias del Caso de Uso

Trayectoria principal

- 1  Invoca el método uploadConfigurations.
- 2  Envía la ruta donde se guardará el archivo de configuración en el HDFS.
- 3  Valida que la ruta esté correctamente escrita. **[Trayectoria A]**
- 4  Verifica si la ruta existe. **[Trayectoria B]**
- 5  Sube el archivo de configuración a la ruta especificada en el HDFS.
- 6  Invoca al método CU3 Subir un archivo con argumentos del archivo de configuración.
- 7 Termina el caso de uso.
---- *Fin del caso de uso.*

Trayectoria alternativa A:

Condición: La ruta del archivo de configuración está correctamente escrita.

- A-1**  Arroja una excepción de tipo LuminusException(6) MSG3 Ruta del archivo de configuración inválida.
- A-2** Termina el caso de uso.
---- *Fin de la trayectoria.*

Trayectoria alternativa B:

Condición: La ruta del archivo de configuración no existe.

- B-1**  Crea el directorio.

B-2 Continúa la ejecución en el paso 5

B-3 Termina el caso de uso.

- - - - *Fin de la trayectoria.*

6.15. CU13 Verificar compatibilidad de datos con algoritmo seleccionado

6.15.1. Resumen

Verifica que los datos incluidos en el caso de estudio del usuario final sean soportados por el algoritmo que el usuario seleccione.

Esto con el objetivo de no permitir la ejecución de un determinado algoritmo hasta que se sepa que soportará los datos de entrada.

6.15.2. Descripción

Caso de Uso:	CU13 Verificar compatibilidad de datos con algoritmo seleccionado
Versión:	0.1
Autor:	Mayra Patricia Tovar Barba y Kevin Raúl Monteón Valdes
Supervisa:	Ulises Vélez Saldaña y Alejandro Botello Castillo.
Actor:	Usuario Experto, Sistema.
Propósito:	Validar el objeto de configuración.
Entradas:	Ninguna.
Origen:	Código fuente.
Salidas:	Valido o Invalido.
Destino:	Código fuente.
Precondiciones:	El archivo properties debe haber sido creado previamente con el CU8 Crear Configuración
Postcondiciones:	Se conoce la validez del archivo de configuración para el algoritmo seleccionado.
Errores:	IOException, LuminusException(4) MSG4 Selección de algoritmo invalida.
Tipo:	Caso de uso primario.
Observaciones:	

6.15.3. Trayectorias del Caso de Uso

Trayectoria principal

- 1  Invoca el método verifyTypes() especificando el algoritmo que se quiere ejecutar.
 - 2  Valida que el algoritmo ingresado sea igual a KNN. [\[Trayectoria A\]](#) [\[Trayectoria B\]](#)
 - 3  Lee el archivo properties.
 - 4  Extrae el valor de las posiciones de las columnas que contienen los datos que alimentarán al algoritmo.
 - 5  Lee el archivo de datos de entrada.
 - 6  Extrae los datos contenidos en las posiciones que se extrajeron en el paso 4.
 - 7  Verifica que todos los datos contenidos en las columnas sean de tipo numérico. (Flotantes, Dobles o Enteros). [\[Trayectoria C\]](#)
 - 8  Retorna el valor booleano true.
 - 9 Termina el caso de uso.
- - - *Fin del caso de uso.*

Trayectoria alternativa A:

Condición: Se seleccionó el algoritmo ID3.

- A-1  Retorna el valor booleano false.

A-2 Termina el caso de uso.

- - - - *Fin de la trayectoria.*

Trayectoria alternativa B:

Condición: Selección inválida de algoritmo.

B-1 ○ Se lanza una excepción de tipo LuminusException(4) **MSG4 Selección de algoritmo Invalida.**

B-2 Termina el caso de uso.

- - - - *Fin de la trayectoria.*

Trayectoria alternativa C:

Condición: Un valor dentro del archivo de datos de entrada no es numérico.

C-1 ○ Retorna el valor booleano *false*.

C-2 Termina el caso de uso.

- - - - *Fin de la trayectoria.*

6.16. Diseño

Dentro del conjunto de clases que conforman la API, se tienen dos clases centrales: la clase *Access* y la clase *LuminusConfiguration*. *Access* es una clase estática cuyos métodos son utilizados en todas las demás clases de la API. Esta clase provee acceso al sistema de archivos distribuido de Hadoop. Permite hacer lectura y escritura sobre ese sistema de archivos. La clase *LuminusConfiguration* permite la interacción con los algoritmos de minería de datos que se implementaron en *Luminus*. Las clases que conforman la API son las siguientes:

- *LuminusConfiguration*
- *LuminusConfigurationKNN*
- *LuminusConfigurationID3*
- *TypeManager*
- *LuminusException*
- *HDFSAccess*

6.16.1. Diagrama de clases

6.17. Desarrollo

El código fuente de la API fue escrito en lenguaje Java. La funcionalidad a detalle de cada clase se enumera a continuación. Para manejar las dependencias del proyecto, se utilizó Maven.

6.17.1. *LuminusConfiguration*

Clase padre genérica de tipo *LuminusConfiguration* que provee la interacción con la configuración de los algoritmos que podrá ejecutar *Luminus*.

Paquete: com.luminus.configuration

Campos:

- **Nombre:** propertiesFileName
- **Alcance:** protected
- **Tipo de dato:** String

- **Descripción:** Nombre del archivo de configuración con extensión .properties.
- **Nombre:** propertiesFilePathInHDFS
- **Alcance:** protected
- **Tipo de dato:** String
- **Descripción:** Ruta en el HDFS donde estará guardado el archivo de configuración con extensión .properties.

Constructores:

- **Nombre:** LuminusConfiguration
- **Alcance:** public
- **Parámetros:**
- **Descripción:** Constructor por default de la clase LuminusConfiguration.

Métodos

- **Nombre:** uploadConfigurations
- **Alcance:** public
- **Parámetros:** String propertiesPathInHDFS
- **Tipo de retorno:** void
- **Descripción:** Sube el archivo properties al HDFS en una ruta específica.
- **Excepciones:** LuminusException

6.17.2. LuminusConfigurationKNN

Clase hija que hereda de la clase LuminusConfiguration que provee la interacción con la configuración del algoritmo KNN y su ejecución.

Paquete: com.luminus.configuration

Campos:

- **Nombre:** k
- **Alcance:** private
- **Tipo de dato:** Integer
- **Descripción:** Número de vecinos al dato centroide que se van a considerar como resultados de la ejecución del algoritmo KNN.
- **Nombre:** columnPositions
- **Alcance:** private
- **Tipo de dato:** Integer
- **Descripción:** Número de la columna que contiene el elemento a buscar. Este campo se ignora si el campo search está configurado como *false*.
- **Nombre:** search

- **Alcance:** private
 - **Tipo de dato:** Boolean
 - **Descripción:** Indica si el elemento a buscar se tomará en cuenta para filtrar la búsqueda y hacerla un poco más selectiva.
-
- **Nombre:** searchElement
 - **Alcance:** private
 - **Tipo de dato:** String
 - **Descripción:** Elemento a buscar que servirá como filtro para restringir aún más el número de comparaciones que se harán con el elemento origen.
-
- **Nombre:** origin
 - **Alcance:** private
 - **Tipo de dato:** ArrayList<Double>
 - **Descripción:** Coordenadas que indican cuál es el elemento origen o centroide con el que se van a comparar los demás elementos del archivo de entradas del algoritmo.
-
- **Nombre:** positions
 - **Alcance:** private
 - **Tipo de dato:** ArrayList<Integer>
 - **Descripción:** Indica el número de las columnas que contienen los datos del elemento origen.
-
- **Nombre:** completeOrderedOutputPath
 - **Alcance:** private
 - **Tipo de dato:** String
 - **Descripción:** Ruta donde se escribirá la salida ordenada completa del algoritmo KNN.
-
- **Nombre:** completeOutputPath
 - **Alcance:** private
 - **Tipo de dato:** String
 - **Descripción:** Ruta donde se escribirá la salida completa sin ordenar del algoritmo KNN.
-
- **Nombre:** kOutputPath
 - **Alcance:** private
 - **Tipo de dato:** String
 - **Descripción:** Ruta donde se escribirá la salida interpretada del algoritmo KNN.

- **Nombre:** spreadsheetOutputPath
- **Alcance:** private
- **Tipo de dato:** String
- **Descripción:** Ruta donde se escribirá la salida en forma de hoja de cálculo del algoritmo KNN.

Constructores:

- **Nombre:** LuminusConfigurationKNN
- **Alcance:** public
- **Parámetros:** Integer k, Integer columnPositions, Boolean search, String searchElement, ArrayList<Double> origin, ArrayList<Integer> positions, String completeOrderedOutputPath, String completeOutputPath, String kOutputPath, String spreadsheetOutputPath

Métodos:

- **Nombre:** createConfiguration
- **Alcance:** public
- **Parámetros:** String fileName
- **Tipo de retorno:** void
- **Descripción:** Crea el archivo properties. Verifica que el nombre del archivo que se le pasó como parámetro tenga la extensión properties.
- **Excepciones:** LuminusException
- **Nombre:** run
- **Alcance:** public
- **Parámetros:** String inputInHDFSPath, String outputInHDFSPath
- **Tipo de retorno:** void
- **Descripción:** Ejecuta el algoritmo KNN con base en las configuraciones previamente realizadas.

6.17.3. LuminusConfigurationID3

Clase hija que hereda de la clase LuminusConfiguration. Provee interacción con la configuración del algoritmo ID3 y con su ejecución. **Paquete:** com.luminus.configuration **Campos:**

- **Nombre:** resultPosition
- **Alcance:** private
- **Tipo de dato:** Integer
- **Descripción:** Número de la columna del archivo de entrada que contiene las clases finales.
- **Nombre:** classes
- **Alcance:** private
- **Tipo de dato:** ArrayList<String>
- **Descripción:** Conjunto de clases finales.

- **Nombre:** headers
- **Alcance:** private
- **Tipo de dato:** ArrayList<String>
- **Descripción:** Encabezados de las columnas del archivo de datos de entrada.

- **Nombre:** outputPath
- **Alcance:** private
- **Tipo de dato:** String
- **Descripción:** Ruta de salida del algoritmo ID3.

- **Nombre:** intermediatePath
- **Alcance:** private
- **Tipo de dato:** String
- **Descripción:** Ruta donde se escribirán los archivos intermedios durante la ejecución del algoritmo ID3.

- **Nombre:** testFolderPath
- **Alcance:** private
- **Tipo de dato:** String
- **Descripción:** Ruta donde se ubicarán las carpetas de prueba durante la ejecución del algoritmo ID3.

- **Nombre:** jFolderPath
- **Alcance:** private
- **Tipo de dato:** String
- **Descripción:** Ruta donde se ubicarán las carpetas de salidas intermedias durante la ejecución del algoritmo ID3.

Constructores:

- **Nombre:** LuminusConfigurationID3
- **Alcance:** public
- **Parámetros:** Integer resultPosition, ArrayList<String> classes, ArrayList<String> headers, String outputPath, String testFolderPath, String jFolderPath, String intermediatePath

Métodos

- **Nombre:** run
- **Alcance:** public
- **Parámetros:** String inputHDFSPath, String outputHDFSPath
- **Tipo de retorno:** void

- **Descripción:** Ejecuta el algoritmo ID3 configurado previamente.
- **Nombre:** createConfiguration
- **Alcance:** public
- **Parámetros:** *String* fileName
- **Tipo de retorno:** void
- **Descripción:** Verifica que el nombre del archivo tenga la extensión .properties. Posteriormente crea el archivo de configuración con todos los parámetros necesarios para la ejecución del algoritmo ID3.

6.17.4. HDFSAccess

Esta clase contiene una clase estática anidada dentro, la clase Access. Esta última provee acceso al sistema de archivos de Hadoop.

Paquete: com.luminus.access

Clases anidadas:

- **Nombre:** Access
- **Alcance:** public static
- **Descripción:** Clase estática que provee acceso al sistema de archivos de Hadoop. Es decir, permite realizar operaciones de lectura y escritura sobre el HDFS.

Constructores de la clase Access Esta clase no tiene constructores debido a que es una clase estática y no necesita ser instanciada para poder hacer uso de sus métodos.

Métodos

- **Nombre:** getDataFromFile
 - **Alcance:** public
 - **Parámetros:** *String* path, *OutputStream* outputStream, *FileSystem* fileSystem, *String* localNetAddress
 - **Tipo de retorno:** void
 - **Descripción:** Obtiene el contenido de un archivo del HDFS y lo escribe haciendo uso de un OutputStream.
 - **Excepciones:** IOException, LuminusException
- **Nombre:** downloadFile
 - **Alcance:** public
 - **Parámetros:** *String* localPath, *String* hdfsPath, *FileSystem* fileSystem, *Boolean* overwrite
 - **Tipo de retorno:** void
 - **Descripción:** Descarga un documento del HDFS y lo escribe en una ruta local.
 - **Excepciones:** IllegalArgumentException, IOException
- **Nombre:** makeHDFSDirectory
 - **Alcance:** public
 - **Parámetros:** *String* path, *FileSystem* fileSystem, *String* localNetAddress

- **Tipo de retorno:** void
 - **Descripción:** Crea un directorio en el HDFS.
 - **Excepciones:** Exception, LuminusException
-
- **Nombre:** uploadFileToHDFS
 - **Alcance:** public
 - **Parámetros:** *String localPath, String hdfsPath, FileSystem fileSystem, Boolean overwrite*
 - **Tipo de retorno:** void
 - **Descripción:** Sube un archivo local al HDFS.
 - **Excepciones:** IOException, LuminusException
-
- **Nombre:** deleteFromHDFS
 - **Alcance:** public
 - **Parámetros:** *String path, FileSystem fileSystem*
 - **Tipo de retorno:** void
 - **Descripción:** Elimina un archivo o directorio del HDFS.
 - **Excepciones:** IllegalArgumentException, IOException, LuminusException
-
- **Nombre:** existsInHDFS
 - **Alcance:** public
 - **Parámetros:** *String path, FileSystem fileSystem*
 - **Tipo de retorno:** void
 - **Descripción:** Verifica si un archivo o directorio existe en el HDFS.
 - **Excepciones:** IllegalArgumentException, IOException
-
- **Nombre:** getLocalNetAddress
 - **Alcance:** public
 - **Parámetros:**
 - **Tipo de retorno:** String
 - **Descripción:** Obtiene la dirección IP local.
-
- **Nombre:** createLocalFile
 - **Alcance:** public
 - **Parámetros:** *String fileName*
 - **Tipo de retorno:** void

- **Descripción:** Crea un archivo local.

- **Nombre:** deleteLocalFile
- **Alcance:** public
- **Parámetros:** *String* fileName
- **Tipo de retorno:** void
- **Descripción:** Borra un archivo local.

6.17.5. LuminusException

Una clase creada para el manejo de excepciones que pueden generarse en tiempo de ejecución al implementar *Luminus*.

Paquete: com.luminus.exception

Campos

- **Nombre:** ex
- **Alcance:** private
- **Tipo de Dato:** String
- **Descripción:** Cadena que guarda el motivo explícito de la excepción.

Constructores

- **Nombre:** LuminusException
- **Alcance:** public
- **Parámetros:** *Integer* reason

Mensajes que arroja la clase LuminusException

6.17.6. MSG1 No existe el archivo o directorio

Tipo: Error

Redacción: No such file or directory.

6.17.7. MSG2 El archivo ya existe

Tipo: Error

Redacción: File already exists.

6.17.8. MSG3 El directorio ya existe

Tipo: Error

Redacción: Directory already exists.

6.17.9. MSG4 Selección de algoritmo inválida

Tipo: Error

Redacción: Invalid algorithm selection.

6.17.10. MSG5 Extensión del archivo properties inválida.

Tipo: Error

Redacción: Invalid properties file extension.

6.17.11. MSG6 Ruta de archivo properties inválida

Tipo: Error

Redacción: Invalid properties path.

6.17.12. TypeManager

Esta clase se encarga de verificar si los datos contenidos en el archivo de entrada es el adecuado para poder ejecutar satisfactoriamente cada algoritmo.

Paquete: com.luminus.configuration

Campos

- **Nombre:** inputFileReader
 - **Alcance:** private
 - **Tipo de dato:** BufferedReader
 - **Descripción:** Lector para acceder a los datos de entrada del algoritmo en cuestión.

- **Nombre:** propertiesFileReader
 - **Alcance:** private
 - **Tipo de dato:** Properties
 - **Descripción:** Lector que permite acceder a los properties del archivo de configuración con extensión .properties.

- **Nombre:** inputFilePath
 - **Alcance:** private
 - **Tipo de dato:** String
 - **Descripción:** Ruta local del archivo de entrada que va a alimentar al algoritmo en cuestión.

- **Nombre:** propertiesFilePath
- **Alcance:** private
- **Tipo de dato:** String
- **Descripción:** Ruta local del archivo de configuración con extensión .properties.

Constructores

- **Nombre:** TypeManager
- **Alcance:** public
- **Parámetros:** *String* inputFilePath, *propertiesFilePath*

Métodos

- **Nombre:** verifyTypes
 - **Alcance:** public
 - **Parámetros:** *String* alg
 - **Tipo de retorno:** Boolean
 - **Descripción:** Selecciona un método de verificación para el algoritmo seleccionado y regresa un booleano indicando si es posible o no ejecutar el algoritmo.
 - **Excepciones:** LuminusException
- **Nombre:** matchColumnWithType
 - **Alcance:** public
 - **Parámetros:** *Integer[]* columns, *String* alg
 - **Tipo de retorno:** Boolean
 - **Descripción:** Verifica que los datos de entrada para un algoritmo seleccionado cumplan con las características necesarias para poder ser ejecutados satisfactoriamente según el algoritmo seleccionado.
- **Nombre:** findColumns
 - **Alcance:** public
 - **Parámetros:** *String* property
 - **Tipo de retorno:** Integer[]
 - **Descripción:** Busca en el archivo de configuración el property que contiene las columnas que se van a evaluar y las regresa en un arreglo de enteros.

6.17.13. Pruebas

Para comprobar el correcto funcionamiento de la API y debido a que esta se ejecuta desde otro código fuente escrito en lenguaje JAVA.

Se escribió un código fuente de ejemplo para demostrar la forma en la que se tendría que hacer uso de los casos de uso listados.

Los casos de uso incluidos en la API al ejecutarse no muestran ningún mensaje en terminal que describa su proceso, exceptuando los mensajes de error.

Al ejecutar los casos de uso, se genera la misma salida que la que generan los algoritmos al ser ejecutados de forma independiente por lo que no seria de gran valor agregar volver a explicar estas salidas en esta sección. en caso de querer conocerlas, esto pueden ser consultadas en la sección [Pruebas](#) del capítulo [Algoritmos de Minería de Datos](#).

Manejo de Archivos

Es necesario subir en el HDFS los datos de entrada a un determinado algoritmo, por ejemplo el archivo que contiene los datos asociados al caso de estudio.

Para ello se requiere crear el archivo correspondiente y subirlo al HDFS esto con el objetivo de que una vez que comience la ejecución del algoritmo, estos datos ya se encuentren al alcance de HADOOP como se muestra en la figura 6.2.

Este método es de utilidad exclusivamente para cuando el archivo que se desea subir ya existe en el directorio local y desea subirse este mismo archivo de datos al directorio del HDFS.

```
try {
    knn.uploadFileToHDFS("/home/maestroluminuscom/Descargas/configuracion.properties",
                         "/user/luminus/configuracion.properties", true);
} catch (IOException e) {
    e.printStackTrace();
}
```

Figura 6.2: Subir un archivo previamente creado al HDFS

En el caso de que se desee crear un archivo o directorio dentro del HDFS sin que este tenga contenido únicamente buscando que este exista y sin la necesidad de crear su equivalente en el equipo local se pueden utilizar las instrucciones mostradas en la figura ??.

```
try {
    knn.createConfiguration("maestro.properties");
    knn.uploadConfigurations("/nndende/dsds/");
} catch (LuminusException e) {
    e.printStackTrace();
}
```

Figura 6.3: Crear archivos en blanco dentro del HDFS

Algo importante a tomar en cuenta al momento de realizar estas tareas es que puede que en ocasiones el archivo que se pretende crear o subir al HDFS ya existiera con anterioridad.

Haciendo uso de esta función, es posible conocer si el archivo o directorio por el que se pregunta ya se encuentra en el HDFS, en caso de existir se devuelve el valor de True y en caso contrario se devuelve el valor False.

```
try {
    System.out.println(
        Access.existsInHDFS("/user/luminus/proc.txt", Access.getFileSystem(Access.getLocalNetAddress())));
} catch (IllegalArgumentException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
```

Figura 6.4: Validar si un archivo o directorio existe dentro del HDFS

Al finalizar la ejecución de un determinado algoritmo es probable que se requiera obtener una copia del archivo de salidas de manera local, por lo que en este caso se puede utilizar el método que se muestra en la figura 6.5.

Haciendo uso de este método se permite obtener la información contenida dentro del archivo seleccionado.

```

try {
    knn.getDataFromFile("/user/luminus/datos.txt", System.out);
} catch (IOException e) {
    e.printStackTrace();
}

```

Figura 6.5: Validar si un archivo o directorio existe dentro del HDFS

Algoritmo KNN

Lo primero que se tiene que realizar es la construcción de los elementos que se pasarán a la API como configuraciones para que esta los utilice en su manejo.

El número de elementos diferentes que se necesiten de entrada serán definidos propiamente por cada algoritmo que se quiera ejecutar, para los algoritmos establecidos hasta ahora se pueden revisar sus especificaciones en la sección [Diseño](#) del prototipo [Algoritmos de Minería de Datos](#)

En el caso particular del ejemplo se trata de la ejecución del algoritmo KNN, por lo que los datos establecidos en el código fuente del usuario corresponden directamente con este algoritmo.

Cabe señalar que estos datos pueden ser construidos de diferentes maneras por el usuario experto y la forma en la que estos se obtengan no tiene injerencia directa con la API.

Lo que resulta importante es que una vez que se tenga la totalidad de estos, se tienen que cargar a el método Luminus Configuration KNN.

```

public class App {
    public static void main(String[] args) {
        Integer k = 3;
        Integer columnas = 13;
        Boolean buscar = false;
        ArrayList<Double> origen = new ArrayList<Double>();
        origen.add(14.2);
        origen.add(13.7);
        ArrayList<Integer> posiciones = new ArrayList<Integer>();
        posiciones.add(14);
        posiciones.add(15);
        String elemento = "\\"Tlaxcala"";
        String salidaCompleta = "/user/luminus/salidacompleta.txt";
        String salidaOrdenadaCompleta = "/user/luminus/salidaordenadacompleta.txt";
        String salidaK = "/user/luminus/salidak.txt";
        String salidaExcel = "/user/luminus/salidafinal.ods";
        LuminusConfigurationKNN knn = new LuminusConfigurationKNN(k, columnas, buscar, elemento, origen, posiciones,
            salidaOrdenadaCompleta, salidaCompleta, salidaK, salidaExcel);
    }
}

```

Figura 6.6: Definición de elementos necesarios para el funcionamiento del algoritmo KNN

Conociendo entonces, los datos de entrada que requieren modificación para la ejecución de este algoritmo en particular que se muestra en la figura 6.6, y ademas de la información de manejo de archivos descrita en la sección [Manejo de Archivos](#), se puede entender con ello la construcción que se realiza en la figura 6.8.

Se presentan 2 nuevas Invocaciones de clases de la API la primera de ellas será útil para revisar si efectivamente el archivo de datos del caso de estudio puede operar con el algoritmo KNN de acuerdo a las columnas seleccionada, en caso de que este no pueda realizarlo regresara un valor false que no permitirá continuar con la ejecución. La segunda y ultima invocación a clase de la API que se presenta en esta sección permite visualizar una operación de tipo run, esta permite que, una vez que se tienen todas las configuraciones pertenecientes a los algoritmos y ya este listo para ponerse en marcha, esta inicie.

Con ello, al finalizar su ejecución se puede tener en el HDFS los resultados de la ejecución.

Estas configuraciones representan todo lo necesario para que un algoritmo de KNN pueda iniciar su operación.

```

try {
    knn.createConfiguration("maestro.properties");
    TypeManager manager = new TypeManager("/home/maestroluminuscom/Escritorio/proyectoknn/tabla.txt",
                                         "/home/maestroluminuscom/Descargas/configuracion.properties");
    if (manager.verifyTypes("KNN")) {
        knn.uploadConfigurations("/user/luminus/configuracion.properties");
        knn.uploadFileToHDFS("/home/maestroluminuscom/Escritorio/proyectoknn/tabla.txt",
                             "/user/luminus/tabla.txt", true);
        knn.run("/user/luminus/tabla.txt", "/user/luminus/salidaluminus");
    }
} catch (LuminusException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}

```

Figura 6.7: Ejecución knn

Algoritmo ID3

De una forma muy similar a como ocurre con el algoritmo KNN este algoritmo, tiene que elaborar sus propias configuraciones de entrada, ya que en caso de no realizarlo, El algoritmo de minería de datos a ejecutar no tendría la información suficiente para comenzar su proceso.

De esta manera, nuevamente conociendo cuales son las configuraciones para que este algoritmo opere, únicamente es necesario obtener los datos solicitados por parte del usuario experto.

Esta información puede ser obtenida de acuerdo a los medios que el programador de la herramienta que haga uso de Luminus, considere, sin embargo en la figura ?? se puede visualizar la forma en la que esta lo realiza para un determinado caso de uso que puede funcionar con el algoritmo ID3.

Es importante que una vez que se tengan todos los parámetros de configuración distintos estos se manden a la clase LuminusConfigurationID3 para que este tenga conocimiento de esta información para poder comenzar a preparar la ejecución del algoritmo con los datos ingresados.

```

Integer posicionResultado = 4;
ArrayList<String> opciones = new ArrayList<String>();
opciones.add("N");
opciones.add("P");
ArrayList<String> nombres = new ArrayList<String>();
nombres.add("General");
nombres.add("Temperatura");
nombres.add("Humedad");
nombres.add("Viento");
nombres.add("Clase");
String pathSalida = "/user/luminus/ID3/salida.txt";
String pathIntermedio = "/user/luminus/ID3/pruebasitnermedias/intermedio.txt";
String pathCarpetapru = "/user/luminus/ID3/pruebasitnermedias/prueba.txt";
String pathCarpetaj = "/user/luminus/ID3/pruebasitnermedias/pruebasjob";
LuminusConfigurationID3 id3 = new LuminusConfigurationID3(posicionResultado, opciones, nombres, pathSalida,
                                                             pathIntermedio, pathCarpetapru, pathCarpetaj);
try {
    id3.createConfiguration("id3nuevo.properties");
    id3.uploadConfigurations("/user/luminus/ID3JA/");
} catch (Exception e) {
    e.printStackTrace();
}

```

Figura 6.8: Definición de elementos necesarios para el funcionamiento del algoritmo ID3

De una forma muy similar a la forma de funcionamiento del algoritmo KNN, una vez que se tiene la totalidad de configuraciones realizadas.

Es necesario realizar el manejo de archivos correspondiente con respecto al HDFS para poder comenzar la ejecución. para revisar mas a detalle como realizar estas operaciones se puede revisar la sección [Manejo de Archivos](#). una vez que se tenga preparado este ambiente, solo es necesario iniciar la ejecución del algoritmo lo cual se realiza haciendo uso del método ID3.run.

Al finalizar la ejecución de este se tendrán los archivos de salida correctamente escritos con la información relacionada al algoritmo.

Finalmente, se pueden eliminar los archivos que se crean para los resultados intermedios en caso de que el usuario experto no tenga interés en ellos, mientras que, si desea conocer los resultados obtenidos a cada iteración puede ser información de utilidad para este usuario. Por esta razón se deja a criterio del usuario si desea realizar su eliminación y en que momento desea hacerlo.

```
try {
    id3.createConfiguration("id3nuevo.properties");
    id3.uploadConfigurations("/user/luminus/ID3Lum/");
    id3.uploadFileToHDFS("/home/maestroluminuscom/Escritorio/proyid3/pruebaid3.txt", "/user/luminus/ID3Lum/",
        true);
    id3.run("/user/luminus/ID3Lum/pruebaid3.txt", "/user/luminus/ID3Lum/salidaID3");
    id3.deleteFromHDFS("/user/luminus/ID3/pruebasitnermedias/pruebasjob");
} catch (LuminusException e) {
    e.printStackTrace();
} catch (IllegalArgumentException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
```

Figura 6.9: Ejecución ID3

Instalador de Luminus

7.1. Descripción del prototipo

Uno de los objetivos de la realización de este trabajo terminal es permitir al usuario empezar a hacer uso de Big Data de una manera sencilla y sin demasiadas complicaciones. Por lo que se llevo a cabo el desarrollo de un instalador que simplifique el proceso de puesta en marcha del ambiente de análisis de datos.

Se trata de una herramienta que asista al usuario en disminuir el número de pasos necesarios para instalar un ambiente que permita llevar a cabo la realización de Big Data sobre de el.

Podría decirse que la utilidad de este prototipo se refleja al comparar el número de pasos que se enuncian en el manual de instalación de *Luminus*, contra el número de pasos que se siguen al utilizar el instalador mismos que serán comparados dentro de esta sección.

7.2. Análisis

La manera en que se pretende que el instalador simplifique el proceso de instalación es automatizando algunas tareas que de otra forma el usuario tendría que realizar una a una, existiendo así la posibilidad de que éste mismo cometa algún error u omita algún paso, y por lo tanto, el ambiente de análisis de datos no pueda ponerse en funcionamiento. Existen diferentes tecnologías para solventar la instalación y configuración de un conjunto previamente determinado de tecnologías, por ejemplo, Docker.

Sin embargo, para el uso de Big Data se utilizan tecnologías como Apache Hadoop y Apache Spark, mismas que para estas tecnologías, la instalación y puesta en marcha con Docker se encuentran en fase experimental lo que significa que no es consistente y generaría muchos problemas al momento de querer utilizarlo [22].

Para solventar esto, se decidió hacer uso del Shell Script de Unix/Linux el cual es simplemente un programa que lee los comandos que se teclean y los convierte en una forma mas entendible para el sistema Unix/Linux. También incluye algunas sentencias básicas de programación que permiten: tomar decisiones, realizar ciclos y almacenar valores en variables [23].

Por lo que, dadas las características de Shell a pesar de que este es la forma mas rudimentaria de realizar esta tarea, es la que de acuerdo a nuestra investigación ofrece mejores resultados para las tecnologías involucradas.

Utilizar Shell Script por otro lado, implica un conjunto de complicaciones ya que todo lo que el script haga, tendrá que ser programado, y esto implica un gran número de validaciones que de utilizar otras tecnologías no se tendrían que considerar, ya que estas serían solventadas por la propia tecnología y su robustez.

7.3. Diseño

La funcionalidad completa del instalador originalmente estaba planteada para contemplar todos los prototipos del Trabajo Terminal, sin embargo, al cambiar la definición del funcionamiento que tendrían los prototipos 3,4 y 5 se encontró que incluirlos en el instalador, no sería lo mas apropiado de acuerdo a la manera en la que estos funcionan y como esta es independiente en cierto sentido del instalador.

Inicialmente, cuando se plantearon estos prototipos de manera general su funcionamiento en conjunto estaba pensado para ser incorporado como un sitio web, por lo que el instalador podría poner los archivos correspondientes de funcionamiento en las carpetas destinadas para el sitio web, con esto, al momento de iniciar Hadoop, ya se podría también entrar en la pestaña *Luminus* de la pantalla de inicio de Hadoop y con esto comenzar a ejecutar las tareas propias de *Luminus* permitiendo de esta forma cargar archivos y ejecutar algoritmos.

Sin embargo, se cambio la forma de Presentar los resultados a una API (Interfaz de Programación de Aplicaciones) por lo que, la forma en la que se incorporen los prototipos 3 , 4 y 5 dependerá de donde se desee que estos sean implementados sobre algún programa Java del usuario final, por lo que, además, no siempre estarían en el mismo equipo que la computadora maestro en el sistema distribuido, con lo que se puede afirmar que, esta administración sería independiente.

E incluso bajo esta nueva definición existe la posibilidad de visualizar la totalidad del trabajo como 2 productos separados pero complementarios. Es decir, se puede requerir al instalador, únicamente para iniciar y configurar una red distribuida con las características para soportar Big Data; y por otro lado o en adición requerir, las configuraciones que tiene la API para soportar los algoritmos sobre la instalación previamente configurada que vendría a ser complementaria pero no forzosa.

Con ello satisfacer necesidades para diferentes grupos de usuarios considerando a los que no desea consumir los algoritmos de minería de datos, o bien, ser complementario, en caso de que el usuario tenga interés en ambos componentes. Por lo que, debido a lo anterior el instalador se elaboro, termino y perfecciono para contemplar la instalación de los prototipos [Evaluación y definición de requerimientos de la red distribuida](#) y [Integración de requerimientos de la red distribuida](#).

En el diagrama de flujo que se muestra en la figura 7.1 se puede apreciar el diseño del instalador.

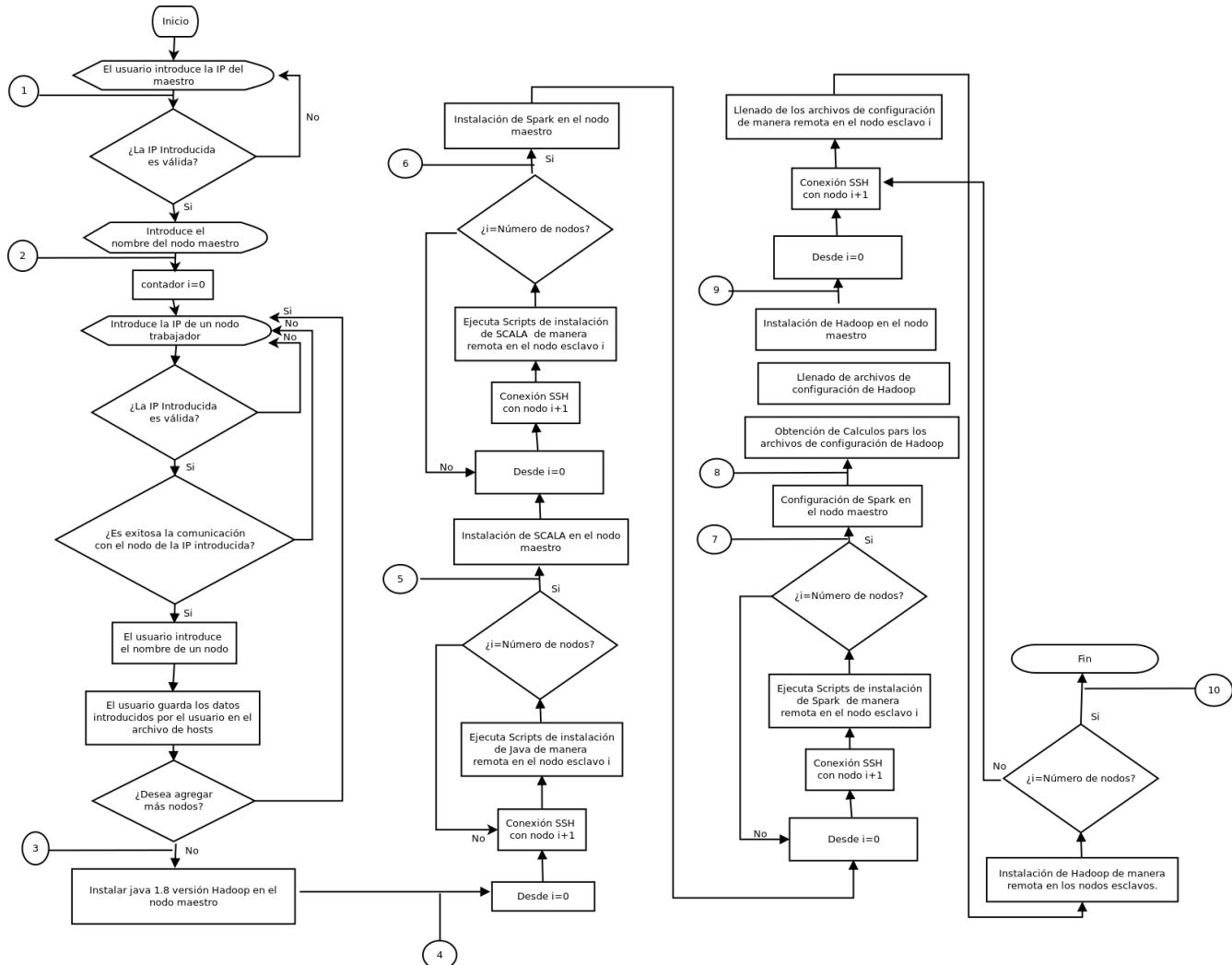


Figura 7.1: Diagrama de flujo del instalador.

El listado de pasos del instalador tal como se puede observar en el diagrama de flujo, si se compará contra el indice del Manual de Instalación de *Luminus*, se podría confirmar que estos en general siguen el mismo proceso y secuencia. Si además se evalúá su flujo paso a paso, sería posible ver que el *Instalador* sigue el mismo flujo que el *Manual de Instalación de Luminus* en cada uno de los pasos, exceptuando que, para algunos de ellos es necesario realizar algunas adecuaciones y consideraciones para que puedan ser ejecutadas desde código y no directamente por el usuario. Una vez establecido que el listado de pasos es el mismo, faltaría conocer cuales son las consideraciones adicionales que se tienen que tomar en cuenta el instalador para poder operar en relación a hacerlo de manera manual las cuales se encuentran listadas en la sección 7.3.

En Resumen, ambos medios hacen lo mismo, ejecutando los mismos pasos solo que uno de ellos lo realiza de manera automática y el otro requiere intervención del usuario en cada uno de sus pasos. La principal diferencia entre ambos es que el usuario en este instalador no puede manejar las excepciones de manera directa y se busca que el instalador pueda arrojar el menor numero de errores posible y considerar la mayor cantidad de casos,de manera que a medida de nuestras posibilidades, estos puedan ser manejados y resueltos desde el propio código del instalador.

El *Manual de instalación de Luminus* tiene documentados cada uno de los pasos necesarios para el funcionamiento describiendo:

- Que hace cada uno de los pasos en la instalación
- Como lo hace
- Como se configura
- Que significa cada uno de los parámetros de entrada
- Entre otros

Es posible que en caso de existir, dudas, interés o algún problema con el *Instalador de Luminus* se recurra a el *Manual de instalación de Luminus* a consultar la documentación de los pasos que hace.

Por otro lado, también es de utilidad en caso de desear conocer el detalle de cuales son los pasos para instalar alguna de las paqueterías que son instaladas por el instalador, o bien, lo que se requiere para replicar su instalación en los nodos de datos/replica, toda esta información se encuentra en el *Manual de instalación de Luminus*. por lo que puede ser utilizado como referencia.

Una de las consideraciones importantes que este instalador toma en cuenta es que todos los pasos puedan ser ejecutados desde el nodo maestro, para que con esto, sea mas sencillo para el usuario final, en lugar de realizar cada uno de los pasos en los diferentes equipos de computo involucrados esto lo hace haciendo uso de un programa llamado SSH el cual le permite realizar todas las conexiones pertinentes entre los diferentes equipos de computo involucrados.

Esto con el objetivo de que el nodo maestro pueda comunicarse remotamente a los nodos de datos/replica y realizar las instalaciones y configuraciones pertinentes.

Las entradas y salidas del diagrama de flujo con respecto a la intervención del usuario en el manual de instalación actual de Luminus, disminuyen demasiado lo cual será detallado y revisado a profundidad en la sección 7.4.

Explicación del funcionamiento del instalador

A continuación se da una descripción a gran escala de como el instalador lleva a cabo su trabajo en cada uno de los pasos, considerando de manera general, un paso como los que vienen indicados en el diagrama,con un circulo el cual contiene el numero de paso.

1. Este paso consiste en comenzar el instalador y llevar a cabo la primera tarea para que inicie su ejecución la cual seria, solicitar la dirección IP de la computadora que tendrá el papel de nodo maestro.
Como tal no es un paso de ejecución o completa ninguna tarea en particular, sin embargo, sirve para marcar el inicio de la ejecución del instalador, e identificar en caso de fallo que ya se había intentado realizar la instalación anteriormente.
2. Este paso, permite introducir los datos que identifican al nodo maestro, por lo que, una vez completado este paso se tiene conocimiento de la IP del nodo maestro así como del nombre que se le quiere dar como identificador a este nodo.

La información de la IP introducida pasa a través de una expresión regular, la cual, permite validar que efectivamente tiene las características de una dirección IP, por ejemplo: 4 octetos, caracteres numéricos, octetos separados por punto, valor de un octeto no mayor a 255, etc. en caso de que los datos recibidos no sean aprobados en esta validación se solicita al usuario que la ingrese nuevamente.

Si la IP introducida es correcta, se valida que al realizar un ping exista una respuesta, esto para verificar que efectivamente se trate de una IP accesible y asignada a algún equipo de computo dentro de la red.

Una vez hechas ambas validaciones se solicita un nombre de nodo maestro, para identificar al nodo que se acaba de agregar, a continuación se permite al usuario continuar con el instalador y lo posiciona en el punto 2.

3. El siguiente paso, consiste en realizar una operación muy parecida a la que se lleva a cabo en el punto 2, pero en lugar de ser para el nodo maestro se ejecuta para los nodos de datos/replica por lo que, al poder existir más de un nodo de datos/replica esta sección es capaz de repetirse hasta que el usuario haya terminado de ingresar todos los nodos de datos/replica que deseé.

La información de la IP introducida para cada uno de los nodos de datos/replica pasa a través de una expresión regular, la cual, permite validar que efectivamente tiene las características de una dirección IP, por ejemplo: 4 octetos, caracteres numéricos, octetos separados por punto, valor de un octeto no mayor a 255, etc. en caso de que los datos recibidos no sean aprobados en esta validación se solicita al usuario que la ingrese nuevamente.

Si la IP introducida es correcta, se valida que al realizar un Ping exista una respuesta, esto para verificar que efectivamente se trate de una IP accesible y asignada a algún equipo de computo dentro de la red.

Una vez hechas ambas validaciones se permite al usuario indicar el identificador que desea dar a este nodo de datos/replica, y posterior a ello tiene la opción de seleccionar si quiere ingresar un nuevo nodo de datos/replica. En caso de indicar que desea hacerlo entonces esta sección se repetirá para el nuevo nodo, cuando el usuario ya no quiera agregar mas nodos de datos/replica se le permitirá continuar con el instalador y lo posiciona en el punto 3.

4. Cuando ya se tienen todas las configuraciones necesarias para iniciar la instalación de los paquetes necesarios, el primero en instalarse es JAVA instalándolo de primera instancia en el nodo maestro.

Debido a que muchos usuarios tienen otra instalación de JAVA en sus equipos de computo requerida para otras aplicaciones instalar la versión que se necesita para el instalador, tiene que hacerse de forma independiente, creando una nueva carpeta para su instalación.

Por lo que la versión de JAVA requerida, no sería la principal del equipo y tampoco empataría con otras versiones disponibles esto quiere decir que no se intentaría actualizar las versiones que se tengan en la maquina.

Consultando otras herramientas que también ofrecen soluciones de Big Data como *Cloudera*, la solución mencionada anteriormente es la solución que se implementa para esta problemática. por lo que, tomando como referencia el análisis hecho y el conocimiento de que se trata de una posibilidad ya probada por otros sistemas se recurrió a hacerlo de esta misma forma.

Cuando la versión de JAVA se encuentre correctamente instalada en el nodo maestro, se coloca la instalación en el punto 4.

5. Posteriormente se procede a instalar de manera distribuida para cada uno de los nodos que se tienen la paquetería de JAVA, utilizando la misma estrategia propuesta para el nodo maestro, haciendo uso de una carpeta adicional para poder diferenciar esta instalación de otras que pudiera tener el equipo de computo.

Esta instalación se lleva a cabo comunicándose con el nodo de replica/datos haciendo uso de SSH y con esto, hace el envío de lo requerido para ejecutar la instalación y posteriormente se envían las instrucciones en forma de pasos para la terminal para que esta los ejecute y al finalizar JAVA quede instalado, desde cada uno de los nodos de datos/replica indicados en el paso 2 cuando son invocados para instalación.

Una vez que esta instalación se concluye de forma satisfactoria se coloca el instalador en el punto 5.

6. Otra tecnología importante a considerar durante la instalación es SCALA el cual es un lenguaje de programación moderno multi-paradigma sobre el que pueden ser implementadas muchas operaciones de minería de datos y algoritmos de análisis de datos de gran volumen.

Esta instalación primero se efectúa en el nodo maestro y posteriormente es replicada haciendo uso de SSH a cada uno de los nodos de datos/replica.

En lo que respecta a esta instalación en particular solo fue necesario pasar cada uno de los pasos descritos en el manual de instalación a el instalador sin tener que hacer cambios en la lógica de funcionamiento, ni considerar

excepciones.

Una vez que esta instalación se concluye de forma satisfactoria se coloca el instalador en el punto 6.

7. Lo siguiente que hace el instalador en este punto es: Instalar Apache Spark tanto en el nodo maestro como en los nodos de datos/replica.

Para llevar a cabo esta instalación Apache Spark requiere que un par de archivos de configuración sean modificados, por lo que el instalador crea archivos "Plantilla" Los cuales se crean escribiendo los datos correspondientes a la instalación que se está efectuando. Es importante que se haga de esta forma, ya que, cada instalación llena estos archivos con información diferente.

La información que sea escrita en los archivos antes mencionados depende directamente de la información que fue introducida en los pasos 1 y 2 como identificadores de los nodos ya que, Apache Spark debe ser capaz de identificarlos y para ello se utilizan los identificadores proporcionados.

Una vez que estos archivos son generados de forma satisfactoria se coloca el instalador en el punto 7.

8. Los archivos "Plantilla" que se generan así como el archivo de instalación de Apache Spark es replicado hacia los nodos de datos/replica, para que estos puedan concretar sus propias instalaciones, y los archivos "Plantilla" generados reemplazan al archivo original que se encuentra en el directorio sin ningún tipo de configuración estos cambios se realizan tanto para el nodo maestro como para los nodos de datos/replica.

Por otro lado, también es necesario modificar los archivos de variable de entorno, para que el nodo maestro sea capaz de reconocer a Apache Spark como una variable de entorno y con esto tener acceso a él desde cualquier punto del sistema operativo entre otras ventajas que son ofrecidas por esta configuración.

una vez hecho esto, se tiene todo lo necesario para poder iniciar Apache Spark únicamente haciendo uso del comando *start-all.sh*.

Cuando estas configuraciones hayan sido efectuadas de manera exitosa, se coloca el instalador en el punto 8.

9. La última tecnología considerada por el instalador es Apache Hadoop, esta es una tecnología que con el proceso de instalación resulta la más pesada de todas las anteriormente contempladas.

Como primer punto es importante instalarlo de manera satisfactoria en el nodo maestro, para ello, al igual que en Apache Spark se requiere el manejo de archivos "Plantilla", ya que se hacen bastantes configuraciones en diferentes archivos.

Las configuraciones requeridas, dependen de la información propia del equipo de computo que alojará la instalación, por lo que es necesario, correr un script que obtiene información del sistema operativo la procesa y obtiene la que es importante para los archivos de configuración.

además se requieren otro tipo de configuraciones, por ejemplo datos proporcionados en los pasos 1 y ?? y otro conjunto de configuraciones de como se espera que sea el funcionamiento Hadoop por ejemplo, delimitadores dentro de un conjunto permitido de valores para que el ambiente pueda ser adaptado a diferentes necesidades.

Cuando se tienen todos los datos necesarios para la configuración de los archivos, Se construyen los archivos de configuración "Plantilla" con los datos recabados en las secciones correspondientes.

Una vez que se tienen los archivos plantilla antes mencionados, se reemplaza en el nodo maestro los archivos de configuración por los archivos "Plantilla".

también se agrega la información de Apache Hadoop a las variables de entorno para este pueda ser accedido desde cualquier punto dentro del sistema operativo.

Cuando estas configuraciones en el nodo maestro hayan sido efectuadas de manera exitosa, se coloca el instalador en el punto 9.

10. Lo siguiente que hace el instalador en este punto es: Instalar Apache Hadoop en los nodos de datos/replica.

Para ello es necesario replicar los archivos de configuración "Plantilla" generados a cada uno de los nodos de datos/replica dentro de la red distribuida.

Realizar la instalación de Apache Hadoop en cada uno de los nodos de datos/replica y reemplazar los archivos de configuración por defecto con los archivos de "Plantilla" que se tienen para realizar estas configuraciones.

Una vez que esta instalación se concluye de forma satisfactoria en los nodos de datos/replica, se coloca el instalador en el punto 10.

es necesario ponerlo en este punto, ya que, si alguien desea volver a iniciar el instalador, con esta información el instalador será capaz de saber que ya había terminado de ejecutar todas las tareas y no quedaban tareas restantes, por lo que se regresará la solicitud notificando que esta instalación ya había terminado con éxito anteriormente.

Este es el ultimo paso del instalador y una vez que este termina entonces termina la ejecución del instalador.

Cada uno de los pasos, descritos anteriormente, al ser completado, marca un estatus de progreso en la ejecución del instalador, este permite que , si el instalador o el proceso de instalación presenta alguna falla o problemática que no le permita continuar con su ejecución, será posible volver a iniciar el instalador y que este reconozca el ultimo punto donde estuvo trabajando para continuar a partir de ese punto y re intentar la instalación. Pero sin comenzar desde el principio y conservando los pasos que resultaron exitosos.

7.4. Desarrollo

El código fuente que se escribió en Shell Script contiene los siguientes pasos que se enuncian en el manual de instalación de *Luminus*.

- 1 Instalación de Apache Spark en el nodo maestro
 - 1.1. Instalación de la paquetería de java
 - 1.3. Instalación de Spark
 - 1.3.1. Configuración maestro
2. Instalación de Apache Spark en los nodos de datos
 - 2.1. Instalación de la paquetería de java en los nodos de datos
 - 2.3. Instalación de Spark en los nodos de datos
 - 2.3.1. Configuración
3. Puesta en funcionamiento
 - 3.1. Configuraciones para poner en funcionamiento Apache Spark en la red distribuida
 - 3.2. Puesta en funcionamiento del cluster manejado por el Servidor Apache Spark
4. Instalación de Apache Hadoop en el nodo maestro
 - 4.1. Instalación de Hadoop
 - 4.1.1. Configuración
 - 4.1.2. Archivos de configuración
5. Instalación de Apache Hadoop en los nodos de datos
 - 5.1. Instalación de Hadoop en los nodos de datos
 - 5.1.1. Configuración

Algunos pasos enunciados en el manual de instalación de *Luminus*, no se encuentran detallados en esta parte, por lo que a continuación se procederá a explicar cuales son estos puntos y la razón por la cual no forman parte del instalador. Estos pasos, tendrían que ser resueltos de manera manual por el usuario final durante el proceso de instalación.

- 1 Instalación de Apache Spark en el nodo maestro
 - 1.2. Instalación de la paquetería de SSH
 - 1.2.1. Configuración
 - 1.2.2. Conexión
2. Instalación de Apache Spark en los nodos de datos
 - 2.2 Instalación de la paquetería SSH en los nodos de datos
6. Puesta en funcionamiento
 - 6.1. Puesta en funcionamiento del Cluster manejado por el Servidor Apache Hadoop
 - 6.2. Subir un archivo al HDFS

Instalación de la paquetería de SSH

Para el instalador, es muy importante que tanto el nodo maestro como los nodos de datos/replica tengan instalado y configurado correctamente SSH ya que en caso de no ser así, no se podría establecer la conexión entre los nodos y por lo tanto sería imposible realizar la instalación de manera distribuida a cada uno de los nodos.

Por lo que se requiere que se encuentre correctamente instalado, el servicio se encuentre iniciado al momento y que además se permitan conexiones de tipo root en cada uno de los nodos.

Por otro lado, se requiere la creación de una llave keygen en el servidor SSH para que las contraseñas que se utilicen

para establecer las conexiones sean seguras y se encuentren cifradas esto con el fin de proteger los datos internos de la empresa y la información que viaja a través de la red, permitiendo cambiar entre los algoritmos de cifrado a utilizar y la longitud con la que las llaves son generadas.

como segundo punto, simplifica el numero de pasos del instalador, ya que de esta manera, no se necesita introducir las contraseñas de root para establecer las conexiones al momento de llevar a cabo el proceso de instalación.

Puesta en funcionamiento

Una vez que se terminan de hacer todas las configuraciones básicas consideradas en el proyecto y se tienen todos los programas necesarios para el mismo, el instalador termina, y con esto es suficiente para que la plataforma de Big Data, este en operación, sin embargo, puede presentarse el caso en que un usuario final requiera realizar mas configuraciones específicas.

En este caso, el usuario, deberá realizar estas configuraciones manualmente antes de poner en funcionamiento, la totalidad del sistema. por lo que, las configuraciones de esta sección tendrían que ser hechas por el usuario, para considerar estas excepciones.

Además de que solo se trata únicamente de 2 pasos los cuales además solo pueden ser hechos en una única ocasión luego de la instalación, y al tratarse de algo tan delicado, por eso lo dejamos como un paso independiente que se considera no es apropiado automatizar.

Consideraciones adicionales

A continuación se muestra una tabla, con el listado del número de pasos que implica hacer la instalación mediante el *Manual de instalación de Luminus* de estas tecnologías, con respecto al número de pasos que implica hacerlo mediante el instalador.

Tarea	Número de pasos en el manual de instalación	Número de pasos en el instalador
Instalación de la paquetería java en el Nodo Maestro	2 pasos	—
Instalación de la paquetería SSH en el Nodo Maestro	9 pasos	9 pasos
Instalación de Spark en el Nodo Maestro	7 pasos	—
Instalación de la paquetería java en los nodos de datos	2 pasos	—
Instalación de la paquetería SSH en los nodos de datos	4 pasos	4 pasos
Instalación de Spark en los nodos de datos	7 pasos	—
Instalación de Apache Hadoop en el nodo maestro	20 pasos	—
Instalación de Apache Hadoop en los nodos de datos	5 pasos	—
Puesta en funcionamiento (Pasos necesarios)	2 pasos	2 pasos
Pasos adicionales del instalador	—	5 pasos
TOTAL	58 pasos	20 pasos

Como se puede notar, el número de pasos necesarios disminuyo considerablemente y esto debido a que algunas de las secciones necesarias de llevar a cabo en el *Manual de Instalación de Luminus* haciendo uso del instalador ya no son necesarias.

Por otro lado, muchos de los pasos enlistados para la configuración de SSH, puede que algún usuario ya los tenga realizados de instalaciones o configuraciones previas, con lo que en este caso solo sería necesario llevar a cabo **7 pasos** para tener un ambiente de Big Data, funcionando en su totalidad.

Pero, incluso no existiendo este caso ideal, únicamente seria necesario ejecutar el **34 % de los pasos originales**.

El código fuente actual de este desarrollo se puede consultar en el anexo [10.2](#).

Con esto, se logro que el usuario final realice menos pasos de los que tendrían que llevarse a cabo si se siguiera el *Manual de Instalación de Luminus* directamente.

Manejo de versionamiento de Software involucrado

Debido a que las versiones de los software a utilizar pueden estar en constante cambio, y que en el momento de ocurrir una actualización de versión en alguno de los software requeridos por el instalador podrían ocurrir fallas en el instalador al no soportar los cambios realizados.

Por lo que, se propone utilizar versiones estáticas de el software requerido por el instalador, para que de esta manera se pueda garantizar, actualizar las versiones del instalador en paralelo con las actualizaciones que se realicen sobre la paquetería necesaria para su operación.

De esta forma, se puede garantizar, que el instalador sea capaz de soportar la versión que por el mismo descargue, y no por una actualización del repositorio origen este deje de funcionar de manera correcta o incluso, no existiría la necesidad de buscar entre diferentes servidores para localizar donde se encuentre disponible el archivo que se esta buscando, ya que la información que cada uno de estos servidores independientes tiene disponible no es gestionada por *Luminus*.

Al momento de descargar el instalador desde el servidor que lo aloje, se descargaran también los paquetes requeridos de las tecnologías a utilizar de esta versión.

Dichos paquetes se irán actualizando en el servidor de versiones de *Luminus* para que, con ello, para cada nueva versión que se genere de *Luminus* se actualice también la paquetería necesaria.

Por ejemplo, para la primera versión de este instalador se utilizarían las siguientes versiones de los paquetes a instalar:

- Java Open JDK 1.8
- Scala 2.6.11
- Spark 2.7
- Hadoop 3.1.1

7.5. Pruebas

Se presenta a continuación una prueba demostrativa del funcionamiento del instalador y como este es operado por el usuario final: Que es lo que el ve, en que momento se despliegan los pasos, cuales son las tareas que tiene que llevar a cabo, etc.

Para la documentación de esta prueba se procede a listar paso por paso lo que el instalador va haciendo, además se agrega evidencia de como esto se visualiza en las salidas arrojadas por el instalador.

El instalador cuenta con 10 pasos importantes durante su ejecución de acuerdo con el diagrama de flujo que se muestra en la imagen 7.1 y con la descripción de los pasos del instalador que se realiza en la sección [Explicación del funcionamiento del instalador](#) se tomarán como referencia estos pasos para clasificar cada uno de los elementos del instalador, por lo que en esta sección no se hará mucha referencia a la manera en que estos operan ya que esta información puede ser encontrada en la sección 7.3, sino mas bien en como el usuario participa en cada una de estas secciones y como es que estas secciones se visualizan para el usuario final.

Paso 1 y Paso 2

Para comenzar, el instalador será ejecutado solamente desde la máquina que ocupe el rol de nodo maestro de la red distribuida haciendo uso de su terminal con privilegios de super usuario y ejecutando el siguiente comando.

`./lum-ins.sh`

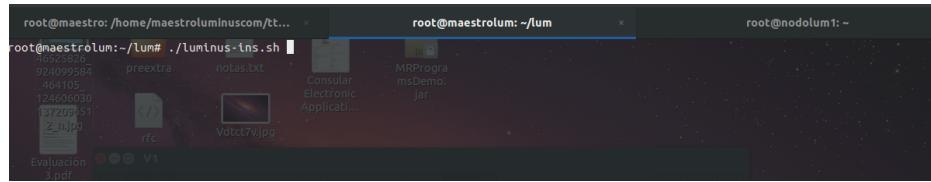


Figura 7.2: Ejecución del instalador.

Este preguntará al usuario por la IP del nodo maestro, Luego preguntará por las IPs de los nodos que conformarán la red distribuida.

Cuando una dirección IP es introducida, el instalador valida que efectivamente se trate de un formato de dirección IP valida, revisando sus características como cadena haciendo uso de una expresión regular, en caso de no tratarse de una dirección IP valida, entonces, se indicara el error como se muestra en la figura 7.6 para que el usuario pueda intentar nuevamente introducirla y no se almacena ningún tipo de información referente a este host erroneo.

Además, el instalador hará un ping para comprobar si hay conexión con el nodo cuya IP se desea agregar a la red distribuida. Si hay respuesta por parte del nodo en cuestión, se pregunta el nombre con el que se desea almacenar este nodo y se procede a almacenar estos datos en un archivo.

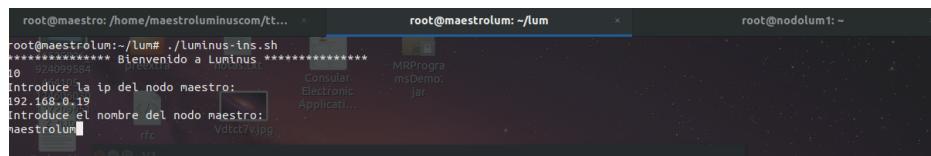


Figura 7.3: IP del Maestro.

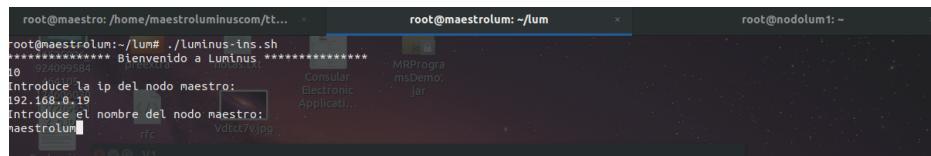


Figura 7.4: IP del nodo datos/replica.

finalmente, una vez que el nodo es agregado, se le pregunta al usuario si desea agregar otro nodo como se muestra en la figura 7.5. Si su respuesta es afirmativa, este proceso se repite.

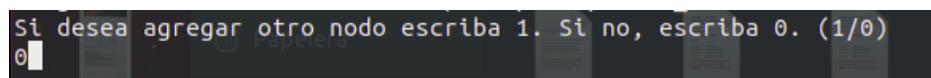


Figura 7.5: Nuevo nodo

El instalador, es capaz de detectar cuando la IP introducida no corresponde a un Host que se encuentre dentro de la red local y notifica al usuario experto de esto, como se muestra en la figura 7.6 además de no realizar ningún tipo de almacenamiento de información referente a este host, erroneo. En esta situación, el usuario puede volver a introducir los datos del host una vez que este los verifique, o bien, introducir los datos de algún otro host luego de comprobar que el host introducido no es accesible desde el nodo maestro y que los datos introducidos por el fueron correctos.

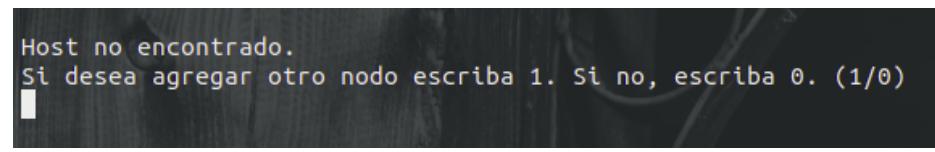


Figura 7.6: El usuario introdujo la IP de un host no encontrado.

Paso 3

Posteriormente, cuando ya no se deseen agregar más nodos, el instalador comenzará su trabajo llevando a cabo la instalación de los paquetes necesarios, en cada uno de los nodos que conforman la red distribuida.

En este punto el usuario experto, deja de tener mucha injerencia en el proceso de instalación y solo tiene que esperar que los pasos vayan siendo completados de manera exitosa.

Por lo que ya no se verá tanta participación por parte del usuario experto a partir de este paso en el instalador y los posteriores.

Lo primero que se instala es la Paquetería de java comenzando por hacerlo desde el nodo maestro. Como se muestra en la imagen 7.7 es necesario ejecutar la descompresión del archivo correspondiente a este paquete que viene con el instalador, en el nodo maestro para posteriormente, proceder a realizar la instalación del mismo.

como se menciona en la sección [Explicación del funcionamiento del instalador](#) se busca que, incluso en caso de que alguno de los equipos de computo involucrados en la red distribuida tengan instalada alguna versión de Java, no se tenga ningún tipo de conflicto con ella, y en su lugar, puedan vivir ambas instalaciones en el equipo de computo.

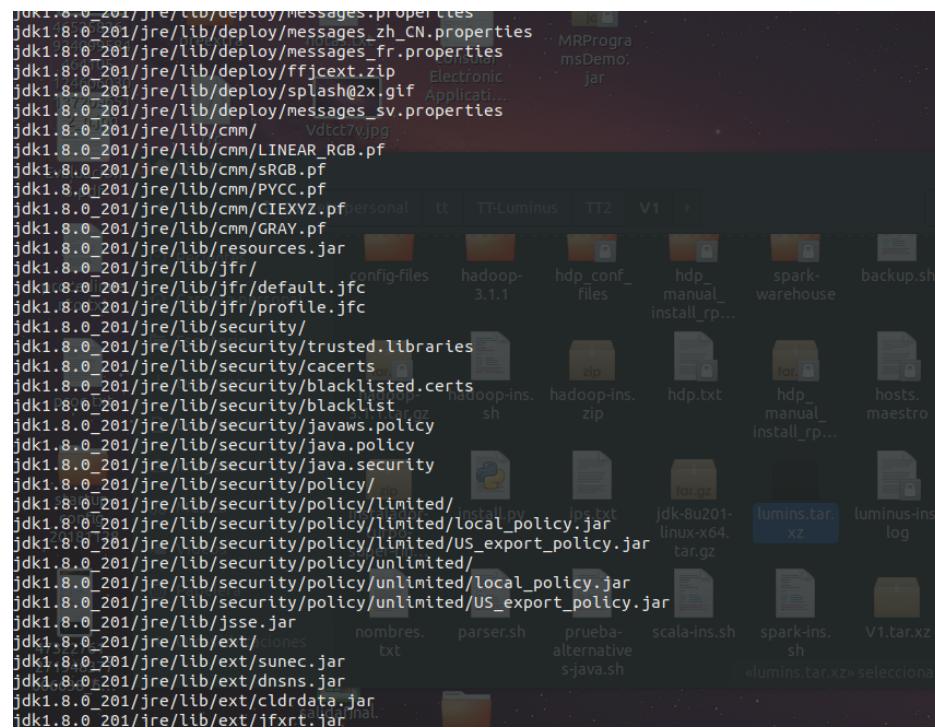


Figura 7.7: Instalación de Java en el Nodo Maestro.

Paso 4

Una vez que la paquetería de JAVA esta correctamente instalada en el nodo maestro, se procede a instalarla en cada uno de los nodos de datos/replica que fueron configurados.

Es necesario hacer un copiado mediante el uso de SSH de los archivos como se puede apreciar en la figura 7.8. y

cuando estos archivos ya se encuentran en cada uno de los nodos de datos/replica, entonces se procede a hacer la descompresión de estos de la misma forma en que lo efectuá en el nodo maestro, y puede ser observado en la imagen 7.7.

Cuando los archivos se encuentran descomprimidos, se hace la instalación de java en cada uno de los nodos, esto se hace de la misma forma que en el nodo maestro. la cual es: crear una carpeta que contenga la versión de JAVA que se pretende instalar, para que esta versión pueda vivir en el equipo de computo,ademas de que si se encuentra alguna otra versión de JAVA instalada en el equipo de computo no se genere ningún conflicto con ella y ambas puedan funcionar estar en funcionamiento dentro del equipo de computo.



Figura 7.8: Copiado de los archivos correspondientes de java.

Paso 5

Posterior a la instalación de Java se procede a realizar la instalación de Scala, la cual se lleva a cabo de una forma muy similar a como se hizo la instalación de Java.

Es necesario descargar el archivo comprimido de Scala desde la liga que lo contiene, para esta versión del instalador en el nodo maestro como se muestra en la figura 7.9.



Figura 7.9: Descarga de Scala en el nodo Maestro

Posteriormente se procede a descomprimir los archivos de Scala descargados al nodo maestro y realizando la instalación de los mismos como se muestra en la figura 7.10 . Para esta instalación no es necesario realizar configuraciones adicionales o algún tipo de adecuación y se siguen los mismos pasos de el *Manual de instalación de Luminus*

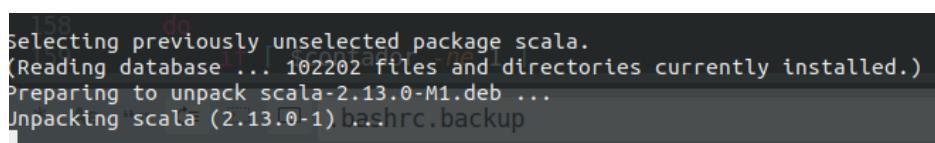


Figura 7.10: Instalación de Scala

Paso 6

Cuando la instalación esta completa en el nodo maestro, se envía el archivo .deb a cada uno de los nodos de datos/replica haciendo uso de SSH como se muestra en la figura 7.11.

Una vez que se encuentran en el nodo de datos/replica se procede a hacer la descompresión de este archivo y posteriormente a realizar la instalación como se muestra en la figura 7.10

```
20550K . . . . . then . . . . . 15% 2.56M 44s
20600K . . . . . scp jdk-8u201-linux-x64.tar.gz root@192.168.1.101:/usr/lib/jvm/java-8-openjdk-amd64/. 15% 2.71M 44s
20650K . . . . . ssh -n root@192.168.1.101 "rm -rf /usr/lib/jvm/java-8-openjdk-amd64/. & sleep 1" 15% 2.48M 44s
20700K . . . . . ssh -n root@192.168.1.101 "mkdir -p /usr/lib/jvm/java-8-openjdk-amd64/. & sleep 1" 15% 2.57M 44s
20750K . . . . . ssh -n root@192.168.1.101 "tar -xvf /home/jdk/jdk-8u201-linux-x64.tar.gz -C /usr/lib/jvm/java-8-openjdk-amd64/. & sleep 1" 15% 2.62M 44s
20800K . . . . . ssh -n root@192.168.1.101 "tar -xvf /home/jdk/jdk-8u201-linux-x64.tar.gz -C /usr/lib/jvm/java-8-openjdk-amd64/. & sleep 1" 15% 2.10M 44s
20850K . . . . . ssh -n root@192.168.1.101 "mv jdk1.8.0_201 /usr/lib/jvm/java-8-openjdk-amd64/. & sleep 1" 15% 2.38M 44s
20900K . . . . . ssh -n root@192.168.1.101 "update-alternatives --config java & sleep 1" 15% 2.83M 44s
20950K . . . . . ssh -n root@192.168.1.101 "fi & sleep 1" 15% 2.61M 44s
21000K . . . . . ssh -n root@192.168.1.101 "contador=$((contador + 1)) & sleep 1" 15% 2.52M 44s
21050K . . . . . ssh -n root@192.168.1.101 "done > ips.txt & sleep 1" 15% 2.43M 44s
21100K . . . . . ssh -n root@192.168.1.101 "rm -f /tmp/basinc & sleep 1" 15% 2.32M 44s
21150K . . . . . ssh -n root@192.168.1.101 "cp /tmp/basinc backup & sleep 1" 15% 2.91M 44s
21200K . . . . . ssh -n root@192.168.1.101 "rm -f /tmp/basinc & sleep 1" 15% 2.10M 44s
21250K . . . . . ssh -n root@192.168.1.101 "rm -f /tmp/basinc & sleep 1" 16% 2.57M 44s
21300K . . . . . ssh -n root@192.168.1.101 "rm -f /tmp/basinc & sleep 1" 16% 2.61M 44s
21350K . . . . . ssh -n root@192.168.1.101 "rm -f /tmp/basinc & sleep 1" 16% 2.53M 44s
21400K . . . . . ssh -n root@192.168.1.101 "echo "5" > /tmp/basinc & sleep 1" 16% 2.57M 44s
21450K . . . . . ssh -n root@192.168.1.101 "#installar scala & sleep 1" 16% 2.02M 44s
21500K . . . . . ssh -n root@192.168.1.101 "#installar scala & sleep 1" 16% 3.99M 44s
21550K . . . . . ssh -n root@192.168.1.101 "#installar scala & sleep 1" 16% 2.41M 44s
21600K . . . . . ssh -n root@192.168.1.101 "#installar scala & sleep 1" 16% 2.06M 44s
21650K . . . . . ssh -n root@192.168.1.101 "#installar scala & sleep 1" 16% 2.60M 44s
21700K . . . . . ssh -n root@192.168.1.101 "#installar scala & sleep 1" 16% 2.36M 44s
21750K . . . . . ssh -n root@192.168.1.101 "function instalar_scala() { & sleep 1" 16% 2.26M 44s
21800K . . . . . ssh -n root@192.168.1.101 "cd ~ & ./scala-ins.sh & sleep 1" 16% 3.67M 44s
21850K . . . . . ssh -n root@192.168.1.101 "contador=1 & sleep 1" 16% 2.24M 44s
21900K . . . . . ssh -n root@192.168.1.101 "while read line & sleep 1" 16% 2.30M 44s
21950K . . . . . ssh -n root@192.168.1.101 "do uu & sleep 1" 16% 2.48M 44s
22000K . . . . . ssh -n root@192.168.1.101 "if [[ $contador -ne 1 ]] & sleep 1" 16% 2.58M 44s
22050K . . . . . ssh -n root@192.168.1.101 "then uu & sleep 1" 16% 2.67M 44s
22100K . . . . . ssh -n root@192.168.1.101 "else uu & sleep 1" 16% 2.68M 44s
22150K . . . . . ssh -n root@192.168.1.101 "fi & sleep 1" 16% 2.50M 44s
```

Figura 7.11: Descarga de Scala en los nodos de datos/replica.

Paso 7

La siguiente tecnologia a instalar es Apache Spark, en este caso, de la misma manera que se realizo con las tecnologias anteriores se realiza la descarga de los paquetes necesarios en el nodo maestro tal como se muestra en la figura 7.12



Figura 7.12: Descarga de Apache Spark en el nodo maestro.

Una vez que se tienen los paquetes necesarios, estos se descomprimen en el nodo maestro como se muestra en la figura 7.13.

```
spark-2.1.0-bin-hadoop2.7/conf/docker.properties.template
spark-2.1.0-bin-hadoop2.7/conf/slaves.template
spark-2.1.0-bin-hadoop2.7/conf/spark-defaults.conf.template
spark-2.1.0-bin-hadoop2.7/LICENSE
spark-2.1.0-bin-hadoop2.7/bin/...@line "tar -xvf /home/jdk-8u201-linux-x64.tar.gz"
spark-2.1.0-bin-hadoop2.7/bin/...@line "mv jdk1.8.0_201 /usr/lib/jvm/java-8-oracle"
spark-2.1.0-bin-hadoop2.7/bin/spark-shell
spark-2.1.0-bin-hadoop2.7/bin/spark-submit.cmd
spark-2.1.0-bin-hadoop2.7/bin/spark-shell2.cmd
spark-2.1.0-bin-hadoop2.7/bin/pyspark2.cmd
spark-2.1.0-bin-hadoop2.7/bin/sparkR.cmd
spark-2.1.0-bin-hadoop2.7/bin/spark-class2.cmd
spark-2.1.0-bin-hadoop2.7/bin/run-example.cmd
spark-2.1.0-bin-hadoop2.7/bin/spark-submit2.cmd
spark-2.1.0-bin-hadoop2.7/bin/spark-class
spark-2.1.0-bin-hadoop2.7/bin/spark-submit
spark-2.1.0-bin-hadoop2.7/bin/spark-class2.cmd
spark-2.1.0-bin-hadoop2.7/bin/find-spark-home
spark-2.1.0-bin-hadoop2.7/bin/run-example
spark-2.1.0-bin-hadoop2.7/bin/beeline
spark-2.1.0-bin-hadoop2.7/bin/pyspark2.cmd
spark-2.1.0-bin-hadoop2.7/bin/spark-shell.cmd
spark-2.1.0-bin-hadoop2.7/bin/spark-class.cmd
spark-2.1.0-bin-hadoop2.7/bin/pyspark.cmd
spark-2.1.0-bin-hadoop2.7/bin/sparkR
spark-2.1.0-bin-hadoop2.7/bin/beeline.cmd
spark-2.1.0-bin-hadoop2.7/bin/sparkR2.cmd
spark-2.1.0-bin-hadoop2.7/bin/load-spark-env.sh
spark-2.1.0-bin-hadoop2.7/bin/load-spark-env.cmd
spark-2.1.0-bin-hadoop2.7/yarn/
spark-2.1.0-bin-hadoop2.7/yarn/spark-2.1.0-yarn-shuffle.jar
spark-2.1.0-bin-hadoop2.7/README.mdackup
```

Figura 7.13: Instalación de Spark en el nodo Maestro.

Paso 8

Cuando la descompresión esta completada en el nodo maestro se procede a elaborar los archivos de configuración "Plantilla" para reemplazar a los archivos de configuración iniciales. Una vez que Apache Spark queda correctamente instalado y configurado en el nodo maestro se procede a realizar lo mismo para los nodos de datos/replica. en este punto, se envía a los nodos esclavos tanto el .deb con los datos del comprimido de Apache Spark como los archivos de configuración generados, este paso se muestra en la figura 7.14

```

12700K ..... F.. $contador...na.1.j..... 6% 2.36M 70s
12750K ..... chen..... 6% 2.46M 70s
12800K ..... scp /jdk-8u201-linux-x64.tar.gz root@$line..... 6% 2.00M 70s
12850K ..... ssh -n root@$line "rm -rf /usr/lib/ivm" 6% 3.83M 70s
12900K ..... ssh -n root@$line "mkdir /usr/lib/ivm" 6% 2.43M 70s
12950K ..... ssh -n root@$line "tar -xvf /home/jdk-8u201-linux-x64.tar.gz -C /usr/lib/ivm" 6% 2.56M 70s
13000K ..... ssh -n root@$line "tar -xvf /home/jdk-8u201-linux-x64.tar.gz -C /usr/lib/ivm" 6% 2.14M 70s
13050K ..... ssh -n root@$line "mv jdk1.8.0_201 /usr/lib/ivm/jdk" 6% 2.50M 70s
13100K ..... ssh -n root@$line "update-alternatives --set java" 6% 2.75M 70s
13150K ..... ssh -n root@$line "update-alternatives --set javac" 6% 2.46M 70s
13200K ..... [1]..... 6% 2.25M 70s
13250K ..... $contador."$((contador+1))."..... 6% 2.13M 70s
13300K ..... echo $ip>/etc/hosts..... 6% 3.93M 70s
13350K ..... cp ./bastidor/backup/* /basirrc..... 7% 2.51M 70s
13400K ..... cp ./bastidor/backup/* /basirrc..... 7% 2.37M 70s
13450K ..... cp ./bastidor/backup/* /basirrc..... 7% 2.18M 70s
13500K ..... # Se escribe un 5 en el log de progreso de instalacion..... 7% 2.91M 70s
13550K ..... [1]..... 7% 2.30M 70s
13600K ..... echo "5" > /var/log/terminus-inst.log..... 7% 2.68M 70s
13650K ..... #Instalar scata..... 7% 2.78M 70s
13700K ..... [1]..... 7% 2.49M 70s
13750K ..... [1]..... 7% 2.41M 70s
13800K ..... [1]..... 7% 2.44M 70s
13850K ..... [1]..... 7% 2.11M 70s
13900K ..... function instalar_scata(){..... 7% 3.17M 70s
13950K ..... ./scala-inst.sh..... 7% 2.62M 70s
14000K ..... ./scala-inst.sh..... 7% 2.56M 70s
14050K ..... contador=1..... 7% 2.10M 70s
14100K ..... while read linea..... 7% 2.87M 70s
14150K ..... do..... 7% 2.65M 70s
14200K ..... [1]..... 7% 2.43M 70s
14250K ..... [1]..... 7% 2.60M 70s
14300K ..... [1]..... 7% 2.20M 70s
14350K ..... [1]..... 7% 659K 70s
14400K ..... [1]..... 7% 659K 70s

```

Figura 7.14: Descargar Spark en los nodos esclavos.

Al finalizar esta transferencia se procede a descomprimir los archivos de configuración enviados, reemplazar los archivos plantilla con los archivos de configuración y a realizar la instalación pertinente para cada uno de los nodos de datos.

cuando este proceso termina, es suficiente con que se ejecute un paso manual adicional posterior a que este paso resulte exitoso en el instalador esto con el objetivo de iniciar las dependencias desde la consulta con privilegios de super usuario el nodo maestro.

El paso es el siguiente:

```
./start-all.sh
```

Lo cual iniciará todas las dependencias de Apache Spark como se muestra en la imagen 7.15

```

root@maestro:/etc/ssh# start-all.sh
starting org.apache.spark.deploy.master.Master, logging to /opt/spark//logs/spark-root-org.apache.spark.deploy.master.Master-1-maestro.out
root@maestro's password: nodo1: starting org.apache.spark.deploy.worker.Worker,
logging to /opt/spark/logs/spark-root-org.apache.spark.deploy.worker.Worker-1-nodo1.out
nodo2: starting org.apache.spark.deploy.worker.Worker, logging to /opt/spark/logs/spark-root-org.apache.spark.deploy.worker.Worker-1-nodo2.out
nodo3: starting org.apache.spark.deploy.worker.Worker, logging to /opt/spark/logs/spark-root-org.apache.spark.deploy.worker.Worker-1-nodo3.out
maestro: starting org.apache.spark.deploy.worker.Worker, logging to /opt/spark/logs/spark-root-org.apache.spark.deploy.worker.Worker-1-maestro.out

```

Figura 7.15: Inicio de Apache Spark

Cuando se inicia Apache Spark en todos los nodos configurados mediante el paso mencionado ya se puede acceder

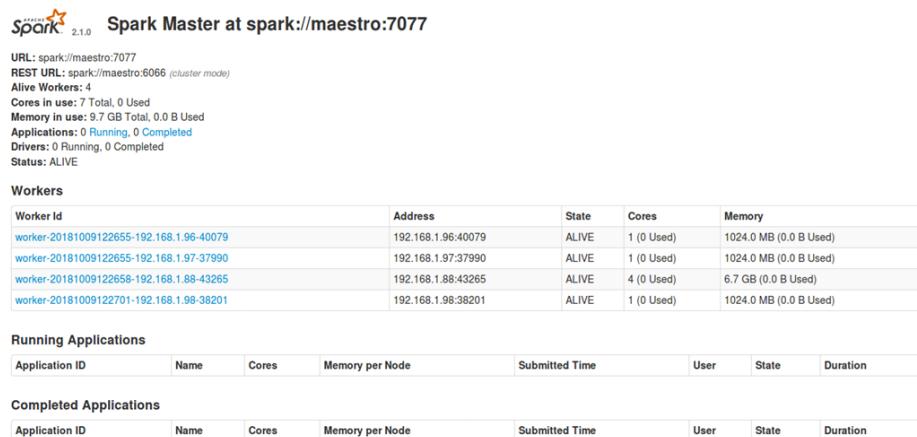
a su vista de configuración de esta herramienta desde la que se puede controlar su funcionalidad. Esto, se puede hacer accediendo a la dirección ip del nodo maestro en el puerto 8080 con el siguiente comando desde un navegador web :

[Direccion IP Maestro]:8080

si se desea acceder a esta vista desde el propio nodo maestro se puede reemplazar la dirección ip con la instrucción de hacerlo en local host como se muestra a continuación:

localhost:8080

La salida entonces para esta dirección web será como se muestra en la imagen 7.16 donde podremos ver toda la configuración de todos los nodos en la red distribuida para Apache Spark.



The screenshot shows the Apache Spark 2.1.0 master interface. At the top, it displays the Spark logo and the text "Spark Master at spark://maestro:7077". Below this, there is a summary of cluster resources:

- URL: spark://maestro:7077
- REST URL: spark://maestro:6066 (cluster mode)
- Alive Workers: 4
- Cores in use: 7 Total, 0 Used
- Memory in use: 9.7 GB Total, 0.0 B Used
- Applications: 0 Running, 0 Completed
- Drivers: 0 Running, 0 Completed
- Status: ALIVE

Below the summary, there are three tables:

- Workers**: A table listing four workers with their addresses, states, cores, and memory usage.
- Running Applications**: An empty table with columns for Application ID, Name, Cores, Memory per Node, Submitted Time, User, State, and Duration.
- Completed Applications**: An empty table with columns for Application ID, Name, Cores, Memory per Node, Submitted Time, User, State, and Duration.

Figura 7.16: Pantalla de configuración de Apache Spark

Paso 9

Otra tecnología que se instala es Apache Hadoop, para ello es necesario nuevamente, descargar los paquetes necesarios para efectuar esta instalación y posteriormente descomprimirlos dentro del nodo maestro, el proceso de descompresión puede ser observado en la figura 7.17.

```

hadoop-3.1.1/share/hadoop/tools/sources/hadoop-archive-logs-3.1.1-test-sources.jar
hadoop-3.1.1/share/hadoop/tools/sources/hadoop-extras-3.1.1-sources.jar
hadoop-3.1.1/share/hadoop/tools/sources/hadoop-archive-logs-3.1.1-sources.jarx64.tar.gz"
hadoop-3.1.1/share/hadoop/tools/sources/hadoop-extras-3.1.1-test-sources.jar
hadoop-3.1.1/share/hadoop/tools/sources/hadoop-distcp-3.1.1-sources.jar
hadoop-3.1.1/share/hadoop/tools/sources/hadoop-gridmix-3.1.1-sources.jar
hadoop-3.1.1/share/hadoop/tools/sources/hadoop-rumen-3.1.1-test-sources.jar
hadoop-3.1.1/share/hadoop/tools/sources/hadoop-datajoin-3.1.1-test-sources.jar
hadoop-3.1.1/share/hadoop/tools/sources/hadoop-streaming-3.1.1-sources.jar
hadoop-3.1.1/share/hadoop/tools/sources/hadoop-distcp-3.1.1-test-sources.jar
hadoop-3.1.1/share/hadoop/tools/sources/hadoop-gridmix-3.1.1-test-sources.jar
hadoop-3.1.1/share/hadoop/tools/sources/hadoop-sls-3.1.1-test-sources.jar
hadoop-3.1.1/share/hadoop/tools/sources/hadoop-archives-3.1.1-test-sources.jar
hadoop-3.1.1/share/hadoop/tools/sources/hadoop-resourceestimator-3.1.1-test-sources.jar
hadoop-3.1.1/share/hadoop/tools/sources/hadoop-rumen-3.1.1-sources.jar
hadoop-3.1.1/share/hadoop/tools/lib/
hadoop-3.1.1/share/hadoop/tools/lib/hadoop-datajoin-3.1.1.jar
hadoop-3.1.1/share/hadoop/tools/lib/hadoop-azure-datalake-3.1.1.jar
hadoop-3.1.1/share/hadoop/tools/lib/hadoop-sls-3.1.1.jar
hadoop-3.1.1/share/hadoop/tools/lib/hadoop-resourceestimator-3.1.1.jar
hadoop-3.1.1/share/hadoop/tools/lib/kafka-clients-0.8.2.1.jar
hadoop-3.1.1/share/hadoop/tools/lib/hadoop-fsizing-3.1.1.jar
hadoop-3.1.1/share/hadoop/tools/lib/hadoop-rumen-3.1.1.jar
hadoop-3.1.1/share/hadoop/tools/lib/azure-keyvault-core-1.0.0.jar
hadoop-3.1.1/share/hadoop/tools/lib/lz4-1.2.0.jar
hadoop-3.1.1/share/hadoop/tools/lib/hadoop-aws-3.1.1.jar
hadoop-3.1.1/share/hadoop/tools/lib/hadoop-openstack-3.1.1.jar
hadoop-3.1.1/share/hadoop/tools/lib/netty-buffer-4.1.17.Final.jar
hadoop-3.1.1/share/hadoop/tools/lib/netty-codec-4.1.17.Final.jar
hadoop-3.1.1/share/hadoop/tools/lib/aws-sdk-bundle-1.11.271.jar

```

Figura 7.17: Instalación de Hadoop en el nodo maestro

una vez que se termina la descompresión se procede a escribir los archivos de configuración "Plantilla" necesarios para estas tecnologías, estos se obtienen de: los datos proporcionados en los pasos 1 y 2 de este manual de instalación, algunos datos por defecto que se establecieron al definir el instalador y que tienen que ser replicados a los archivos de configuración cada vez que se haga una instalación de este tipo y algunos datos particulares de las maquinas donde se pretende establecer el ambiente de Big Data, para ello, se corre otro Script que permite obtener esta información para poder escribirla sobre a los archivos de configuración "Plantilla".

una vez que los archivos de configuración "Plantilla" generados, estan listos, se procede a realizar el copiado de los mismos con las configuraciones establecidas reemplazando los archivos de configuración por defecto, este proceso de copiado se muestra en la figura 7.18.

core-site-config.xml	867	59.1KB/s	00:00
hadoop-env-config.sh	16KB	8.0MB/s	00:00
hdfs-site-config.xml	1124	71.7KB/s	00:00
mapred-site-config.xml	1620	31.2KB/s	00:00
yarn-site-config.xml	1443	1.8KB/s	00:00
workers	23	0.0KB/s	00:00

Figura 7.18: Instalación de Hadoop en el nodo maestro

Paso 10

El siguiente paso, es instalar Apache Hadoop en los nodos de datos/replica, para ello es necesario pasar tanto los archivos binarios de instalación como los archivos de configuración generados como se muestra en la figura 7.19

core-site-config.xml	867	59.1KB/s	00:00
hadoop-env-config.sh	16KB	8.0MB/s	00:00
hdfs-site-config.xml	1124	71.7KB/s	00:00
mapred-site-config.xml	1620	31.2KB/s	00:00
yarn-site-config.xml	1443	1.8KB/s	00:00
workers	23	0.0KB/s	00:00

Figura 7.19: Instalación de Hadoop en los nodos de datos/replica

Una vez hecho esto, se procede a realizar la descompresión de los mismos y posteriormente a sobre escribir los archivos de configuración por defecto de Apache Hadoop por los archivos de configuración "Plantilla" recibidos del nodo maestro.

Con esto se terminaría el proceso de instalación y estaría todo listo para las tecnologías contempladas.

Al finalizar el instalador se deja a decisión y consideración del usuario realizar cambios sobre los archivos en caso de que requiera efectuar algún cambio antes de comenzar con su ambiente.

Cuando termine de hacer los cambios pertinentes, tendrá que ejecutar el siguiente comando en una terminal del nodo maestro con privilegios de super usuario:

```
hdfs namenode -format
```

Lo que formateará toda la información contenida en el HDFS y con esto lo dejará disponible para ser utilizado como se puede ver en la figura 7.20.

Es importante mencionar que esta tarea solo puede ser ejecutada una vez antes de comenzar el ambiente de Big Data.

Figura 7.20: Limpieza del HDFS

Cuando el formateo esta terminado, entonces se procede a iniciar Apache Hadoop para que empiece su funcionamiento dentro de el ambiente distribuido, esto se puede hacer ejecutando el siguiente comando.

start-all.sh

Con lo que comenzará a levantar e inicializar todas las dependencias necesarias para el funcionamiento de este framework como se puede ver en la figura 7.21.

```
root@maestrolum:/opt/hadoop/sbin# ./start-all.sh
Starting namenodes on [maestro]
Starting datanodes 9866 (192.168.0.19:9866) http://maestrolum:9866
Starting secondary namenodes [maestrolum]
Starting resourcemanager 192.168.0.20:9866 http://node01:9864
Starting nodemanagers
```

Figura 7.21: Inicio de Apache Hadoop

Cuando se inicia Apache Hadoop en todos los nodos configurados mediante el instalador ya se puede acceder a su vista de configuración de esta herramienta desde la que se puede controlar su funcionalidad.

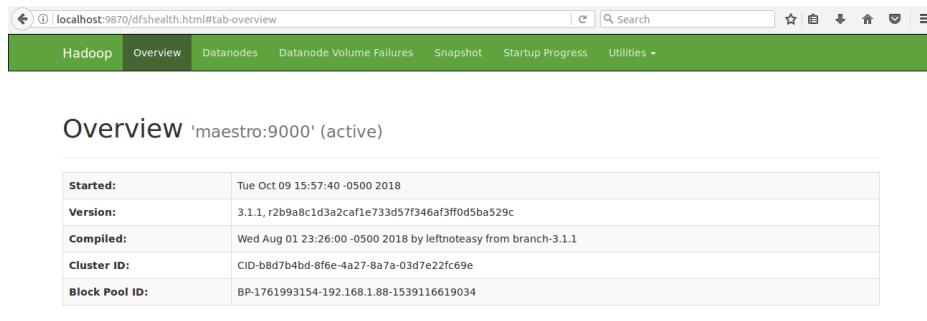
Esto, se puede hacer accediendo a la dirección ip del nodo maestro en el puerto 9870 con el siguiente comando desde un navegador web :

[Direccion IP Maestro]:9870

si se desea acceder a esta vista desde el propio nodo maestro se puede reemplazar la dirección ip con la instrucción de hacerlo en local host como se muestra a continuación:

localhost:9870

La salida entonces para esta dirección web será como se muestra en la imagen 7.22 donde podremos ver toda la configuración de todos los nodos en la red distribuida para Apache Hadoop.



The screenshot shows a web browser window with the URL `localhost:9870/dfshealth.html#tab-overview`. The title bar says "localhost:9870/dfshealth.html#tab-overview". The top navigation bar has tabs: Hadoop, Overview, Datanodes, Datanode Volume Failures, Snapshot, Startup Progress, and Utilities. The "Overview" tab is selected. Below the tabs is a section titled "Overview 'maestro:9000' (active)". Underneath is a table with the following data:

Started:	Tue Oct 09 15:57:40 -0500 2018
Version:	3.1.1, r2b9a8c1d3a2cafce733d57f346af3ff0d5ba529c
Compiled:	Wed Aug 01 23:26:00 -0500 2018 by leftnoteeasy from branch-3.1.1
Cluster ID:	CID-b8d7b4bd-8f6e-4a27-8a7a-03d7e22fc69e
Block Pool ID:	BP-1761993154-192.168.1.88-1539116619034

Figura 7.22: Pantalla de configuración de Apache Hadoop

En este punto se da por terminado, las actividades y alcances del instalador, una vez terminado el paso 10 y comprobado que todo funciona correctamente, se puede afirmar que ya se tiene un ambiente de Big Data correctamente instalado y configurado ejecutándose en los equipos que forman parte de la red distribuida.

CAPÍTULO 8

Conclusiones

8.1. Conclusiones

Puede decirse con toda certeza que los objetivos que se establecieron al inicio del proyecto se cumplieron, esto indica, que se pudo completar lo que se esperaba realizar durante el desarrollo de este proyecto.

Se realizaron algunos cambios en cuanto a las tecnologías propuestas para ser utilizadas y los prototipos que se establecieron en un principio, sin embargo, para cada cambio realizado se busco primeramente que este siguiera cumpliendo los objetivos que fueron planteados en un inicio. Con la finalidad de que ningún cambio nos alejará de dichos objetivos. Sobre la marcha del presente trabajo terminal, se detectó que era necesario realizar un instalador que fuera capaz de simplificar la preparación del ambiente de análisis de datos *Luminus* al usuario experto.

Esta necesidad fue detectada debido a que el número de pasos necesario para tener un ambiente de análisis de datos que pueda empezar a operar era grande. Por lo que, se consideró simplificar este proceso para el usuario experto con el objetivo de que pudiera centrarse en la utilización del ambiente de minería de datos, y no hacer tan engorroso su proceso de instalación.

El instalador desarrollado cuenta con múltiples validaciones que se hicieron para englobar la mayor cantidad de errores posibles e instala todas las tecnologías necesarias para tener en funcionamiento un ambiente de análisis de datos.

Por lo que, se puede concluir que el instalador fue una implementación que logro cumplir el objetivo para el que se propuso, además de que se ha comprobado su correcto funcionamiento ya que hasta ahora se han realizado múltiples instalaciones.

Por otro lado, también se interactuo con el ambiente de análisis de datos haciendo las instalaciones correspondientes en equipos de cómputo tradicionales y se aplicaron algoritmos de minería de datos sobre esta instalación con la finalidad de mostrar que con equipos de cómputo tradicionales es posible hacer uso de Big Data y que estos son capaces de soportar estas operaciones. Por lo que se puede decir, que en un momento dado no es absolutamente necesario para una empresa invertir en equipo de computo especializado para el manejo de Big Data.

Los algoritmos de minería de datos implementados se encuentran en funcionamiento y se obtienen los resultados esperados por los mismos, con lo que de acuerdo a las pruebas se pudo comprobar su correcta implementación y la demostración de que en caso de programarse nuevos algoritmos de minería de datos este ambiente serio capaz de soportarlos para su ejecución.

No se pretende limitar a que únicamente opere con los algoritmos implementados sino la intención de esta implementación es demostrar que para diferentes algoritmos propuestos el ambiente puede realizar los cálculos pertinentes.

Esto podría abrir las puertas a que muchas empresas comiencen a utilizar a estas técnicas para el análisis de sus datos sin tener que realizar una gran inversión inicial, por lo que, podría constituir la posibilidad de crecimiento en el uso de Big Data en la industria.

CAPÍTULO 9

Trabajo a futuro

9.1. Trabajo a futuro

Pequeñas mejoras e implementaciones que se pueden hacer sobre lo ya propuesto

En este trabajo a pesar de que se intentó cubrir la mayor cantidad de puntos posibles quedaron abiertos otros puntos a desarrollar. Algunos de los puntos que se sugieren para trabajar posteriormente se listan a continuación:

- Llevar a cabo un análisis completo de cuales son las características mas apropiadas y que impliquen el menor costo posible en la adquisición y mantenimiento de las computadoras que trabajaran en la red distribuida. Con ello ofrecer una propuesta a las empresas, para que consigan equipos de computo con estas características y con ello obtener el mejor rendimiento optimizando su inversión
- Hacer una implementación diferente del instalador de *luminus* haciendo uso incluso de otra tecnología que permita hacer validaciones de errores de forma completa y contemplando una mayor cantidad de casos para que este pueda ser mas efectivo.
Podría tratarse de un instalador con interfaz gráfica el cual podría resultar mas atractivo para un conjunto de usuarios.
- Realizar el desarrollo y adaptación a Luminus de otros algoritmos de minería de datos ya sea de arboles de decisión o de reglas de asociación o bien crear un nuevo conjunto de algoritmos que sean soportados por la plataforma, además de los 2 ya establecidos.
Para que el análisis que se realice a los datos pueda llegar a ser mas completo y existan más opciones para el usuario experto de algoritmos a aplicar a su caso de estudio.
- Hacer un tratamiento de los datos de entrada para los algoritmos con el fin de que se analicen características como la precisión y el muestreo de los datos. para tener un análisis mas completo de los datos con los que se alimentan los algoritmos.
- Crear una librería de transformación y limpieza para que los datos de entrada sean coherentes y bien formados. Por ejemplo: Si se tiene un campo llamado Edad el cual almacena la edad de los colaboradores de una empresa en un tipo de dato entero.
Supongamos que uno de ellos tiene almacenado el valor 3000 por mencionar algun numero.
En este caso este número no es un valor válido para un campo que almacena edades de personas y el algoritmo no sería capaz de detectarlo ya que cumple con la premisa de tratarse de un valor entero. Introducirlo con este error a la ejecución del algoritmo puede entorpecer los resultados que este arroje como salidas. Por lo que, detectar este tipo de inconsistencias es lo que se buscaría con esta librería.

- Graficación de los resultados de salida de los algoritmos para que se puedan ver las salidas de forma mas clara y visual para el usuario final.

El algoritmo KNN podría dibujar las distancias espaciales con respecto a los vecinos cercanos encontrados.

El algoritmo ID3 podría dibujar el árbol de salida gráfico con sus ramas y hojas.

- Realizar una adaptación al algoritmo KNN para que sea capaz de soportar datos ponderados por el usuario.

Es decir, para datos que no son nominales pero que sin embargo no son muy cambiantes y tienen un numero finito de entradas diferentes, estas puedan ser definidas como una equivalencia antes de comenzar la ejecución del algoritmo y con ello considerar estos campos dentro de la ejecución del algoritmo.

Por ejemplo, para un dato que habla de las etapas de vida de una persona:

y este únicamente almacena los siguientes valores: Bebe,Infante,Niño,Adolescente,Adulto Joven,Adulto Maduro, Anciano

estos podrían ser ponderados de la forma en que el usuario elija para cada ejecución para el caso del ejemplo podrían ser agregados valores numéricos como se muestra a continuación.

Bebe 1,Infante 2,Niño 3,Adolescente 4,Adulto Joven 5,Adulto Maduro 6, Anciano 7.

siendo estos valores de ponderación responsabilidad directa de quien los asigne, pero que permiten, tener mas opciones para elegir al momento de seleccionar los atributos que pueden ser utilizados por el algoritmo.

- Realizar los ajustes para que se soporten tipos diferentes de entradas de datos, por ejemplo: JSON, XML , Bases de datos.

ya que al momento únicamente se permite trabajar con archivos CSV, Lo cual puede llegar a ser muy limitado al momento de escoger las fuentes de datos disponibles para alimentar el algoritmo.

- Realizar una implementación que permita obtener automáticamente datos de una fuente externa y los actualice en tiempo real dentro del repositorio de datos. Esto para empresas que poseen datos cambiantes y requieren actualización constante de los mismos.

Cabe señalar que además de estas propuestas existen muchas otras que pueden adaptarse a este proyecto y ampliarlo para diferentes finalidades. Las posibilidades son prácticamente infinitas y dependen directamente de las necesidades específicas de cada usuario o empresa.

Las propuestas que aquí se señalan son ideas que se encontraron durante el desarrollo del proyecto y que se considera pueden contribuir en gran medida al mismo tomando en cuenta los objetivos que se establecieron.

Se tiene además una propuesta general que puede ser aplicada para varios de los puntos anteriormente mencionados y además se trata de una forma diferente de visualizar el proyecto esta se puede consultar en el anexo ?? . Se tiene un previo desarrollo de esta propuesta, por esta razón a pesar de no estar terminada no se incluye directamente dentro de la sección trabajo a futuro y en su lugar se detallan los avances logrados y la propuesta para seguir trabajando en ellos.

Bibliografía

- [1] M. Mittal, V.E. Balas, D.J. Hemanth (2018) *Data Intensive Computing Applications for Big Data*, 2018.
- [2] Manieviesa, P, 2017 *El Big Data en las empresas*,Septiembre 21, 2017 Pymerang.com. Sitio web:<http://www.pymerang.com/direccion-de-negocios/821-el-big-data-en-las-empresas>.
- [3] Luján Mora, S, 2002 *Programación de aplicaciones web*,1st ed. San Vicente (Alicante): Editorial Club Universitario, p.Indice de acronimos. Sitio Web: <http://rua.ua.es/dspace/handle/10045/16995>
- [4] Gutiérrez ,Javier, S/A ¿Qué es un framework web? Marzo 12, 2019 lsu.es Sitio Web: www.lsu.us.es/~javierj/investigacion_ficheros/Framework.pdf
- [5] Gutierrez,D, 2010 *Frameworks y Componentes* Marzo 12, 2019 codecompiling Sitio Web: http://www.codecompiling.net/files/slides/IS_clase_10_frameworks_componentes.pdf
- [6] Techopedia, 2019 *Software Library* Marzo 13, 2019 techopedia Sitio Web: <https://www.techopedia.com/definition/3828/software-library>
- [7] Joyanes, L (2016) «Big Data, Análisis de grandes volúmenes de datos en organizaciones», *AlfaOmega*,
- [8] PowerData, S/A *Big Data: ¿En qué consiste? Su importancia, desafíos y gobernabilidad*,Septiembre 14, 2017 Pymerang.com. Sitio web: <http://www.powerdata.es/big-data>.
- [9] KNIME, (2017) *open for innovation KNIME*, Octubre 6, 2017, de KNIME Sitio web: <https://www.knime.com/>
- [10] IBM,(2017) *IBM SPSS Modeler*, Octubre 6, 2017,de IBM Sitio web. <https://www.ibm.com/es-es/marketplace/spss-modeler/purchase#product-header-top>
- [11] QlikSense,(2017) *Simple to use. More power than you can imagine*, Septiembre 14,2017, de QlikSense. Sitio web: <http://www.qlik.com/>
- [12] Databricks, (2018) *Databricks Unified Analytics Platform*, Agosto 3, 2018, de Databricks. Sitio web: <https://databricks.com/product/unified-analytics-platform>
- [13] Cloudera, (2018), *About Cloudera*, Enero 23, 2018, de Cloudera. Sitio web: <https://www.cloudera.com/more/about.html>
- [14] Hortonworks, (2018), *Hortonworks Data Platform*, Junio 14, 2018, de Hortonworks. Sitio web: <https://es.hortonworks.com/products/data-platforms/hdp/>
- [15] Databricks, (2018), *Launch cloud-optimized Apache Spark™ clusters in minutes*, Septiembre 20, 2018, de Databricks. Sitio web: <https://databricks.com/try-databricks>
- [16] Cloudera, (2018), *Download Cloudera*, Septiembre 10, 2018, de Cloudera. Sitio web: <https://www.cloudera.com/downloads.html>

- [17] Hortonworks, (2018), *Descargas de las plataformas de datos conectadas de Hortonworks*, Agosto 1, 2018, de Hortonworks. Sitio web: <https://es.hortonworks.com/downloads/>
- [18] Data Science,(2017) *Does a big data virtual machine machine help in analyzing large files*, Septiembre 14, 2017, de Data Science StackExchange. Sitio web <https://datascience.stackexchange.com/questions/14993/does-a-big-data-virtual-machine-machine-help-in-analyzing-large-file>
- [19] Srinivasan, V. (2016) «The Intelligent Enterprise in the Era of Big Data.», Canada: John Wiley & Sons.
- [20] Manchón M.,(2017) *Las empresas y el reto del big data: sólo usan el 1 % de los datos.*, Junio 6, 2017, de EDeconomíaDigital Sitio web: https://www.economiadigital.es/directivos-y-empresas/empresas-big-data-uno-por-ciento_408504_102.html
- [21] García, S,(2017) *Big Data: realidades, retos y limitaciones*, Julio 16, 2017, de Excelsior Sitio web: <http://www.excelsior.com.mx/opinion/opinion-del-experto-nacional/2017/07/16/1175906>
- [22] The Apache Software Foundation (2018) *Launching Applications Using Docker Containers*, Marzo 16, 2018, de Apache Hadoop. Sitio web: <https://hadoop.apache.org/docs/r3.0.1/hadoop-yarn/hadoop-yarn-site/DockerContainers.html>
- [23] Santos A *Programación Shell*, Septiembre 12, 2018, de freeshel.de. Sitio web: www.freeshell.de/rasoda/programacion/guia-shell.pdf
- [24] Oracle (2017) *The Definition of Big Data*, Marzo 22, 2017, de Oracle. Sitio web: <https://www.oracle.com/big-data/guide/what-is-big-data.html>
- [25] Oracle (2017) *Big Data Use Cases*, Abril 14, 2017, de Oracle. Sitio web: <https://www.oracle.com/big-data/guide/big-data-use-cases.html>
- [26] Sinnexus (2016) *Datamining (Minería de datos)*, Marzo 6, 2016 de Sinnexus. Sitio web: https://www.sinnexus.com/business_intelligence/datamining.aspx
- [27] Microsoft (2017) *Conceptos de minería de datos*, Julio 22, 2017, de Microsoft. Sitio web: <https://docs.microsoft.com/es-es/sql/analysis-services/data-mining/data-mining-concepts?view=sql-server-2017>
- [28] GestiónDeOperaciones (2016) *Árbol de Decisión (Qué es y para qué sirve)*, Marzo 17, 2016 de www.gestiondeoperaciones.net. Sitio web: <https://www.gestiondeoperaciones.net/procesos/arbol-de-decision/>
- [29] LucidChart (2016) *Qué es un diagrama de árbol de decisión*, Diciembre 12, 2016, de LucidChart. Sitio web: <https://www.lucidchart.com/pages/es/qu%C3%A9-es-un-diagrama-de-%C3%A1rbol-de-decisi%C3%B3n>
- [30] Sefik Ilkin Serengil (2017) *A Step by Step ID3 Decision Tree Example*, Noviembre 20, 2017, de Sefik Ilkin Serengil. Sitio web: <https://sefiks.com/2017/11/20/a-step-by-step-id3-decision-tree-example/>
- [31] Sefik Ilkin Serengil (2018) *A Step By Step C4.5 Decision Tree Example*, Mayo 13, 2018, de Sefik Ilkin Serengil. Sitio web: <https://sefiks.com/2018/05/13/a-step-by-step-c4-5-decision-tree-example/>
- [32] Deepanshu Bhalla (2016) *K NEAREST NEIGHBOR : STEP BY STEP TUTORIAL*, Septiembre 30, 2016, de Deepanshu Bhalla. Sitio web: <https://www.listendata.com/2017/12/k-nearest-neighbor-step-by-step-tutorial.html>
- [33] Andrea Trevino (2016) *Introduction to K-means Clustering*, Diciembre 6, 2016, de Oracle. Sitio web: <https://www.datascience.com/blog/k-means-clustering>
- [34] IBM, (2018) *Apache MapReduce*, Febrero 8, 2018, de IBM. Sitio web: <https://www.ibm.com/analytics/hadoop/mapreduce>
- [35] Abraham Requena Mesa (2017) *¿Qué es Hadoop?*, Septiembre 28, 2017, de OpenWebinars. Sitio web: <https://openwebinars.net/blog/que-es-hadoop/>
- [36] Abraham Requena Mesa (2018) *Qué es Apache Spark*, Julio 02, 2018 de OpenWebinars. Sitio web: <https://openwebinars.net/blog/que-es-apache-spark/>

-
- [37] David Loshin (2017) *Explorando distribuciones Hadoop para gestionar big data*, Mayo 30, 2017, de SearchDataCenter. Sitio web: <https://searchdatacenter.techtarget.com/es/cronica/Explorando-distribuciones-Hadoop-para-gestionar-big-data>
 - [38] Microsoft (2018) *What is a server cluster?*, Mayo 28, 2018, de Microsoft. Sitio web: [https://technet.microsoft.com/pt-pt/library/cc785197\(v=ws.10\).aspx](https://technet.microsoft.com/pt-pt/library/cc785197(v=ws.10).aspx)
 - [39] BTECH (2010) *What is Computing Environment?*, Julio 12, 2010, de BTECH. Sitio web: <http://btechsmartclass.com/CP/computing-environments.htm>
 - [40] Devin Soni (2018) *Introduction to Bayesian Networks*, Junio 8, 2018, de Towards Data Science. Sitio web: <https://towardsdatascience.com/introduction-to-bayesian-networks-81031eeed94e>

CAPÍTULO 10

Anexos

10.1. Anexo A: Código JAVA Map-Reduce

```
package PackageDemo;

import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;
public class WordCount {
    public static void main(String [] args) throws Exception
    {
        Configuration c=new Configuration();
```

```
String [] files=new GenericOptionsParser(c, args).getRemainingArgs();  
  
Path input=new Path(files[0]);  
  
Path output=new Path(files[1]);  
  
Job j=new Job(c, "wordcount");  
  
j.setJarByClass(WordCount.class);  
  
j.setMapperClass(MapForWordCount.class);  
  
j.setReducerClass(ReduceForWordCount.class);  
  
j.setOutputKeyClass(Text.class);  
  
j.setOutputValueClass(IntWritable.class);  
  
FileInputFormat.addInputPath(j, input);  
  
FileOutputFormat.setOutputPath(j, output);  
  
System.exit(j.waitForCompletion(true)?0:1);  
  
}  
  
public static class MapForWordCount extends Mapper<LongWritable, Text, Text, IntWritable>{  
  
    public void map(LongWritable key, Text value, Context con) throws  
IOException, InterruptedException  
  
    {  
  
        String line = value.toString();  
  
        String [] words=line.split(",");  
  
        for(String word: words)  
  
        {  
  
            Text outputKey = new Text(word.toUpperCase().trim());  
  
            IntWritable outputValue = new IntWritable(1);  
  
            con.write(outputKey, outputValue);  
  
        }  
  
    }  
  
}
```

```
public static class ReduceForWordCount extends Reducer<Text,
  IntWritable, Text, IntWritable>

{

  public void reduce(Text word, Iterable<IntWritable> values, Context
  con) throws IOException, InterruptedException

  {

    int sum = 0;

    for(IntWritable value : values)

    {

      sum += value.get();

    }

    con.write(word, new IntWritable(sum));

  }

}

}
```

10.2. Anexo B: Código del instalador

Código de la clase principal luminus-ins.sh

```
#!/bin/bash
# Instalador general de luminus version 4.

# El instalador pide al usuario introducir las ips de los nodos incluida la del master.
# Las ips son almacenadas en un archivo de texto.

# Funcion que valida el formato de las ips mediante el uso de expresiones regulares.
function validar_ip() {
  local valida=0
  local ip=$1
  if [[ $ip =~ ^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}$ ]]; then
    OIFS=$IFS
    IFS='.'
    ip=($ip)
    IFS=$OIFS
    if [[ ${ip[0]} -le 255 && ${ip[1]} -le 255 \
          && ${ip[2]} -le 255 && ${ip[3]} -le 255 ]]
    then
      valida=1
      echo $valida
    else
      echo $valida
    fi
  else
    echo $valida
  fi
}
```

```
        fi
    else
        echo $valida
    fi
}

# Lectura de la ip del nodo maestro
function pedir_ip_maestro() {
    # Se escribe un 1 en el log del progreso de instalacion.
    # El 1 corresponde a que solamente se inicio pero no se completo el primer paso.
    echo "1" > luminus-ins.log
    cp /etc/hosts.back /etc/hosts
    confirmacionIPMaestro=0
    while [ $confirmacionIPMaestro -eq 0 ]
    do
        echo "Introduce la ip del nodo maestro:"
        read ipMaestro
        if [ $(validar_ip $ipMaestro) -eq 1 ]
        then
            echo "Introduce el nombre del nodo maestro:"
            read nodoMaestro
            echo $nodoMaestro > nombres.txt
            echo $ipMaestro > ips.txt
            echo "$ipMaestro $nodoMaestro maestro" >> /etc/hosts
            echo "$ipMaestro $nodoMaestro maestro" > hosts.maestro
            cp /etc/hosts /etc/hosts.back
            cp ~/.bashrc ~/bashrc.back
            ssh-keygen -f "/root/.ssh/known_hosts" -R $ipMaestro
            confirmacionIPMaestro=1
        else
            echo "Introduzca una IP valida."
        fi
    done
    # Se escribe un 2 en el log de progreso de instalacion.
    # El 2 corresponde a que se introdujo de manera satisfactoria la IP del nodo maestro.
    echo "2" > luminus-ins.log
    pedir_ips_esclavos
}

# Lectura de las ips de los nodos esclavos.
# Escritura de las ips en el archivo hosts del nodo maestro.
# Adicion de la ip del nodo maestro al archivo hosts de los nodos esclavos.
function pedir_ips_esclavos() {
    confirmacion=1
    confirmacionIP=0
    contador=1
    while [ $confirmacion -eq 1 ]
    do
        while [ $confirmacionIP -eq 0 ]
        do
            echo "Introduce la ip del nodo $contador:"
            read ipNodo
            if [ $(validar_ip $ipNodo) -eq 1 ]
```

```

        then
            ping -c1 -qq $ipNodo
            if [ $? -ne 0 ]
            then
                echo "Host no encontrado."
            else
                echo "Host encontrado!"
                echo "Introduzca el nombre del host:"
                read nodo
                echo "$ipNodo $nodo" >> /etc/hosts
                ssh-copy-id -i ~/.ssh/id_rsa.pub root@$ipNodo
                ssh root@$ipNodo "cp /etc/hosts.back /etc/hosts"
                scp hosts.maestro root@$ipNodo:/etc/
                ssh root@$ipNodo "cat /etc/hosts.maestro >>
                /etc/hosts"
                ssh root@$ipNodo "rm /etc/hosts.maestro"
                ssh root@$ipNodo
                "cp /etc/hosts /etc/hosts.back"
                ssh root@$ipNodo "cp ~/bashrc
                ~/bashrc.back"
                ssh-keygen -f "/root/.ssh/known_hosts" -R
                $ipNodo
                echo $nodo >> nombres.txt
                echo $ipNodo >> ips.txt
                contador=$((contador + 1))
                confirmacionIP=1
            fi
        else
            echo "Introduzca una IP valida."
        fi
    done
    echo "Si desea agregar otro nodo escriba 1. Si no, escriba 0. (1/0)"
    read confirmacion
done
# Numero de replicacion para el hdfs segun el numero de nodos esclavos
cp config-files/hdfs-site-config.back.xml config-files/hdfs-site-config.xml
numNodos=$((contador - 1))
echo $numNodos >> config-files/hdfs-site-config.xml
echo "</value> </property> </configuration>" >>
config-files/hdfs-site-config.xml
# Se escribe un 3 en el log de progreso de instalacion.
# El 3 corresponde a que se introdujeron de manera satisfactoria las IPs
de los esclavos.
echo "3" > luminus-ins.log
instalar_java_maestro
}

# Instalacion de java 1.8 especifico para hadoop
function instalar_java_maestro() {
    rm -rf /usr/lib/jvm/java-8-oracle-hadoop
    mkdir /usr/lib/jvm
    tar -xvf jdk-8u201-linux-x64.tar.gz
    mv jdk1.8.0_201/ /usr/lib/jvm/java-8-oracle-hadoop
    update-alternatives --install /usr/bin/java java
}

```

```
/usr/lib/jvm/java-8-oracle-hadoop/bin/java 1061
# Se escribe un 4 en el log de progreso de instalacion.
# El 4 corresponde a la instalacion de java de manera satisfactoria en el
# nodo maestro.
echo "4" > luminus-ins.log
instalar_java_esclavos
}

function instalar_java_esclavos() {
    contador=1
    while read line
    do
        if [ $contador -ne 1 ]
        then
            scp jdk-8u201-linux-x64.tar.gz root@$line:/home/
            ssh -n root@$line "rm -rf /usr/lib/jvm/java-8-oracle-hadoop"
            ssh -n root@$line "mkdir /usr/lib/jvm/"
            ssh -n root@$line "tar -xvf /home/jdk-8u201-linux-x64.tar.gz"
            ssh -n root@$line "mv jdk1.8.0_201/
/usr/lib/jvm/java-8-oracle-hadoop"
            ssh -n root@$line "update-alternatives --install /usr/bin/java java
/usr/lib/jvm/java-8-oracle-hadoop/bin/java 1061"
        fi
        contador=$((contador + 1))
    done < ips.txt
    cp ~/.bashrc.backup ~/.bashrc

    # Se escribe un 5 en el log de progreso de instalacion.
    # El 5 corresponde a la instalacion de java de manera satisfactoria en los
    # esclavos.
    echo "5" > luminus-ins.log
    #instalar_scala
}

#Instalacion de Scala en todos los nodos.
function instalar_scala() {
    ./scala-ins.sh
    contador=1
    while read line
    do
        if [ $contador -ne 1 ]
        then
            echo $line
            ssh -qq root@$line < scala-ins.sh
        fi
        contador=$((contador + 1))
    done < ips.txt

    # Se escribe un 6 en el log de progreso de instalacion.
    # El 6 corresponde a la instalacion de scala de manera satisfactoria en el
    # maestro y los esclavos.
    echo "6" > luminus-ins.log
    instalar_spark
}
```

```
# Instalacion de spark en todos los nodos.
function instalar_spark() {
    ./spark-ins.sh
    contador=1
    while read line
    do
        if [ $contador -ne 1 ]
        then
            echo $line
            ssh -qq root@$line < spark-ins.sh
        fi
        contador=$((contador + 1))
    done < ips.txt

    # Se escribe un 7 en el log de progreso de instalacion.
    # El 7 corresponde a la instalacion de Spark de manera satisfactoria en el
    # maestro y los esclavos.
    echo "7" > luminus-ins.log
    configurar_spark_maestro
}

# Configuracion de Spark en el nodo maestro.
function configurar_spark_maestro() {
    # Copia de la plantilla de esclavos de Spark.
    cp /opt/spark/conf/slaves.template /opt/spark/conf/slaves

    # Se agregan los esclavos y el maestro a la lista de nodos de la red.
    cat ips.txt > /opt/spark/conf/slaves

    # Se copia la plantilla de spark-env
    cp /opt/spark/conf/spark-env.sh.template /opt/spark/conf/spark-env.sh

    # Se agrega el maestro al archivo spark-env.sh
    echo "export SPARK_MASTER_HOST=$ipMaestro" >> /opt/spark/conf/spark-env.sh

    # Se escribe un 8 en el log de progreso de instalacion.
    # El 8 corresponde a la configuracion de Spark de manera satisfactoria en el
    # maestro.
    echo "8" > luminus-ins.log
    instalar_hadoop_maestro
}

# Instalacion de Hadoop.
function instalar_hadoop_maestro() {
    # wget http://mirrors.sonic.net/apache/hadoop/common/
    wget http://mirrors.sonic.net/apache/hadoop/common/
    n/hadoop-3.1.1/hadoop-3.1.1.tar.gz
    rm -rf /opt/hadoop
    mkdir /opt/hadoop
    tar -xzvf hadoop-3.1.1.tar.gz
    mv hadoop-3.1.1/* /opt/hadoop/

    # Archivo de configuracion de las variables de entorno
    cat config-files/config-bash.txt >> ~/.bashrc
```

```
source ~/.bashrc

# Modificacion del arhcivo de configuracion hadoop-env.sh
# Respaldo del archivo hadoop-env.sh
cp /opt/hadoop/etc/hadoop/hadoop-env.sh
/opt/hadoop/etc/hadoop/hadoop-env.backup.sh
# Limpieza del archivo
# echo "" > /opt/hadoop/etc/hadoop/hadoop-env.sh
# Sustitucion por el archivo con la variable de entorno JAVA_HOME con su valor
correcto
cp config-files/hadoop-env-config.sh /opt/hadoop/etc/hadoop/hadoop-env.sh

# Modificacion del archivo de configuracion core-site.xml
# Respaldo del archivo core-site.xml
cp /opt/hadoop/etc/hadoop/core-site.xml
/opt/hadoop/etc/hadoop/core-site.backup.xml
# Limpieza del archivo
# echo "" > /opt/hadoop/etc/hadoop/core-site.xml
# Sustitucion por el archivo con la configuracion correcta
cp config-files/core-site-config.xml
/opt/hadoop/etc/hadoop/core-site.xml

# Modificacion del archivo de configuracion hdfs-site.xml
# Respaldo del archivo hdfs-site.xml
cp /opt/hadoop/etc/hadoop/hdfs-site.xml
/opt/hadoop/etc/hadoop/hdfs-site.backup.xml
# Limpieza del archivo
# echo "" > /opt/hadoop/etc/hadoop/hdfs-site.xml

# Sustitucion por el archivo con la configuracion correspondiente
cp config-files/hdfs-site-config.xml
/opt/hadoop/etc/hadoop/hdfs-site.xml

# Configuracion de los archivos mapred y yarn.
./parser.sh

# Modificacion del archivo de configuracion mapred-site.xml
# Respaldo del archivo mapred-site.xml
cp /opt/hadoop/etc/hadoop/mapred-site.xml
/opt/hadoop/etc/hadoop/mapred-site.backup.xml
# Limpieza del archivo
# echo "" > /opt/hadoop/etc/hadoop/mapred-site.xml
# Sustitucion por el archivo con la configuracion correspondiente
cp config-files/mapred-site-config.xml
/opt/hadoop/etc/hadoop/mapred-site.xml

# Modificacion del archivo de configuracion yarn-site.xml
# Respaldo del archivo yarn-site.xml
cp /opt/hadoop/etc/hadoop/yarn-site.xml
/opt/hadoop/etc/hadoop/yarn-site.backup.xml
# Limpieza del archivo
# echo "" > /opt/hadoop/etc/hadoop/yarn-site.xml
# Sustitucion por el archivo con la configuracion correspondiente
cp config-files/yarn-site-config.xml /opt/hadoop/etc/hadoop/yarn-site.xml
```

```
# Configuracion del archivo workers
contador=1
while read line
do
    if [ $contador -ne 1 ]
    then
        echo $line >> /opt/hadoop/etc/hadoop/workers
    fi
    contador=$((contador + 1))
done < ips.txt

# Se escribe un 9 en el log de progreso de instalacion.
# El 9 corresponde a la configuracion de Hadoop de manera satisfactoria
#en el maestro.
echo "9" > luminus-ins.log
instalar_hadoop_esclavos
}

function instalar_hadoop_esclavos() {
    # Copiado de los archivos de configuracion de hadoop del nodo maestro
    #a los nodos esclavos
    contador=1
    while read line
    do
        if [ $contador -ne 1 ]
        then
            echo "Instalando Hadoop en nodo $contador ($line)..."
            scp hadoop-3.1.1.tar.gz root@$line:/home/
            ssh -qq root@$line < hadoop-ins.sh
            scp config-files/core-site-config.xml
            root@$line:/opt/hadoop/etc/hadoop/core-site.xml
            scp config-files/hadoop-env-config.sh
            root@$line:/opt/hadoop/etc/hadoop/hadoop-env.sh
            scp config-files/hdfs-site-config.xml
            root@$line:/opt/hadoop/etc/hadoop/hdfs-site.xml
            scp config-files/mapred-site-config.xml
            root@$line:/opt/hadoop/etc/hadoop/mapred-site.xml
            scp config-files/yarn-site-config.xml
            root@$line:/opt/hadoop/etc/hadoop/yarn-site.xml
            scp /opt/hadoop/etc/hadoop/workers
            root@$line:/opt/hadoop/etc/hadoop/workers
            contador=$((contador + 1))
        fi
        contador=$((contador + 1))
    done < ips.txt

    # Se escribe un 10 en el log de progreso de instalacion.
    # El 10 corresponde a la configuracion e instalacion de Hadoop en los nodos
    #esclavos
    # de manera satisfactoria en el maestro.
    echo "10" > luminus-ins.log
}
```

```
echo "***** Bienvenido a Luminus *****"

pedir_ip_maestro
instalar_hadoop_maestro

# # Ciclo para recolectar la informacion del archivo log
contador=1
while read line
do
    if [ $contador -eq 1 ]
    then
        line=$line
        break
    fi
done < luminus-ins.log

echo $line

# # Switch para saber en que paso iniciara el instalador
if [ $line -eq '1' ] || [ $line -eq '10' ]
then
#     pedir_ip_maestro
fi
if [ $line -eq '2' ]
then
    echo "REANJUNDANDO INSTALACION..."
#     pedir_ips_esclavos
fi
if [ $line -eq '3' ]
then
    echo "REANJUNDANDO INSTALACION..."
#     instalar_java_maestro
fi
if [ $line -eq '4' ]
then
    echo "REANJUNDANDO INSTALACION..."
#     instalar_java_esclavos
fi
if [ $line -eq '5' ]
then
    echo "REANJUNDANDO INSTALACION..."
#     instalar_scala
fi
if [ $line -eq '6' ]
then
    echo "REANJUNDANDO INSTALACION..."
    instalar_spark
fi
if [ $line -eq '7' ]
then
    echo "REANJUNDANDO INSTALACION..."
#     configurar_spark_maestro
fi
if [ $line -eq '8' ]
```

```
then
    echo "REANJUNDANDO INSTALACION . . ."
#       instalar_hadoop_maestro
fi
if [ $line -eq '9' ]
then
    echo "REANJUNDANDO INSTALACION . . ."
#       instalar_hadoop_esclavos
fi
```

Código de scala-ins.sh

```
# Instalacion de Scala
wget https://downloads.lightbend.com/scala/2.13.0-M1/scala-2.13.0-M1.deb
dpkg -i scala-2.13.0-M1.deb
scala -version
```

Código de spark-ins.sh

```
# Instalacion de Spark
rm spark-2.1.0-bin-hadoop2.7.tgz
wget http://d3kbcqa49mib13.cloudfront.net/spark-2.1.0-bin-hadoop2.7.tgz
mkdir /opt/spark
tar -xzvf spark-2.1.0-bin-hadoop2.7.tgz
cp -a spark-2.1.0-bin-hadoop2.7/. /opt/spark/
rm -rf spark-2.1.0-bin-hadoop2.7
echo 'export SPARK_HOME=/opt/spark/' >> ~/.bashrc
echo 'export PATH="/opt/spark/bin/:/opt/spark/sbin:$PATH"' >> ~/.bashrc
source ~/.bashrc
```

Código de hadoop-ins.sh

```
# Inicia instalacion de hadoop
# Descarga , descompresion y movimiento del
# contenido de hadoop a su carpeta
```

```
# wget http://mirrors.sonic.net/apache/hadoop/common/hadoop-3.1.1/hadoop-3.1.1.tar.gz
rm -rf /opt/hadoop
mkdir /opt/hadoop
tar -xzvf /home/hadoop-3.1.1.tar.gz
mv hadoop-3.1.1/* /opt/hadoop/
```

Código de backup.sh

```
# Se ejecuta en el nodo maestro .
cp /etc/hosts /etc/hosts.back
cp ~/.bashrc ~/bashrc.back
```

```
# Se ejecuta en cada uno de los nodos .
```

```
contador=1
while read line
do
    if [ contador -ne 1 ]
        ssh root@$line "cp /etc/hosts /etc/hosts.back"
        ssh root@$line "cp ~/bashrc ~/bashrc.back"
    then
        contador=$((contador + 1))
    fi
done
```

Código de parser.sh

```
#!/usr/bin/env bash
function parsear_conjunto_valores_hdp() {
rm -rf valores_hdp.txt
while IFS='=' read -ra line; do
    for i in "${line[@]}"; do
        while IFS=' ' read -ra value; do
            for j in "${value[1]}"; do
                echo $value >> valores_hdp.txt
            done
        done <<< $i
    done
done <<< $(echo $(python hdp_conf_files/scripts/yarn-utils.py -c 1 -m 1 -d 1 false))
}
function generar_property() {
    echo "<property>" >> $3
    echo "<name>$1</name>" >> $3
    echo "<value>$(sed $2'q;d' valores_hdp.txt)</value>" >> $3
    echo "</property>" >> $3
}

# Metodo que construye el archivo mapred-site-config.xml.
function generar_mapred_site_xml() {
    cp config-files/mapred-site-config.back.xml config-files/mapred-site-config.xml
    # Property para el valor yarn.app.mapreduce.am.command-opts
    # generar_property
    #'yarn.app.mapreduce.am.command-opts' '23'
    #'config-files/mapred-site-config.xml'
    # Property para el valor mapred.map.memory.mb
    generar_property 'mapreduce.map.memory.mb'
    '18' 'config-files/mapred-site-config.xml'
    # Property para el valor mapreduce.reduce.memory.mb
    generar_property 'mapreduce.reduce.memory.mb' '20'
    'config-files/mapred-site-config.xml'
    # Property para el valor yarn.app.mapreduce.am.resource.mb
    generar_property 'yarn.app.mapreduce.am.resource.mb' '22'
    'config-files/mapred-site-config.xml'
    # Property para el valor mapreduce.map.java.opts
    # generar_property 'mapreduce.map.java.opts' '19'
    #'config-files/mapred-site-config.xml'
    # Property para el valor mapreduce.java.opts
    # generar_property #'mapreduce.reduce.java.opts' '21'
    #'config-files/mapred-site-config.xml'
    # Property para el valor mapreduce.task.io.sort.mb
    # generar_property
    #'mapreduce.task.io.sort.mb' '24' 'config-files/mapred-site-config.xml'

    echo '</configuration>' >> config-files/mapred-site-config.xml
}

# Metodo que construye el archivo yarn-site-config.xml
function generar_yarn_site_xml() {
    cp config-files/yarn-site-config.back.xml config-files/yarn-site-config.xml
    # Property para el valor yarn.nodemanager.resource.memory-mb
```

```
generar_property  
'yarn.nodemanager.resource.memory-mb' '17'  
'config-files/yarn-site-config.xml'  
# Property para el valor yarn.scheduler.maximum.allocation-mb  
generar_property 'yarn.scheduler.maximum.allocation-mb' '16'  
'config-files/yarn-site-config.xml'  
# Property para el valor yarn.scheduler.minimum.allocation-mb  
generar_property 'yarn.scheduler.minimum.allocation-mb' '15'  
'config-files/yarn-site-config.xml'  
# Property para el valor yarn.app.mapreduce.am.command-opts  
# generar_property 'yarn.app.mapreduce.am.command-opts' '23'  
'config-files/yarn-site-config.xml'  
echo '</configuration >' >> config-files/yarn-site-config.xml  
}  
  
parsear_conjunto_valores_hdp  
generar_yarn_site_xml  
generar_mapred_site_xml
```

10.3. Anexo C: Una manera de presentar a luminus de forma diferente: Sitio web

Como se planteo anteriormente, inicialmente el proyecto estaba pensado para ser desarrollado como un sitio web, sin embargo, durante el desarrollo se opto por la opción de darle un comportamiento de API. esto con el fin de proporcionar una funcionalidad diferente.

Por lo que se puede afirmar que la API no es excluyente de el desarrollo del sitio web. y la implementación mas recomendable dependería de cada usuario final y sus necesidades. Podría trabajarse en la implementación de este sitio web para cubrir una mayor cantidad de usuarios y adaptarse de mejor manera a sus necesidades.

En este sentido, debido a que, ya se había pensado en este desarrollo se tiene una propuesta de como podrían diseñarse las pantallas correspondientes al sitio web por lo menos para el algoritmo KNN además de el mapa de navegación de las mismas. mismas que podrían ser usadas como referencia, o bien, ser utilizadas como tal.

El mapa de navegación es el siguiente:

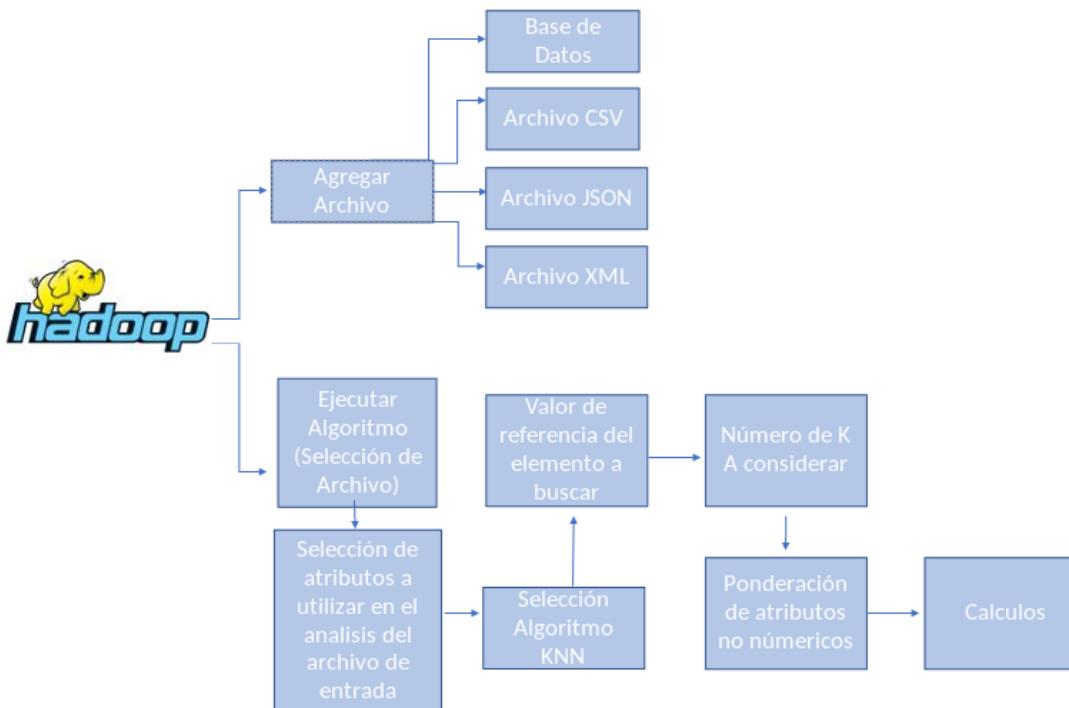


Figura 10.1: Mapa de Navegación

Se procede a explicar cada una de las pantallas que se listan en el mapa de navegación en el orden en que se mandarían a llamar, esto con el fin de que sea claro el proceso de ejecución y funcionamiento de las mismas, además de mencionar para cada una de ellas su objetivo la propuesta de visualización.

El diseño de pantallas propuesto, vendría directamente de las vistas que ya tiene implementadas Hadoop para su funcionamiento, es decir, se agregaría un elemento al menú de opciones de Hadoop. esto para simplificar el entendimiento de las pantallas, y como todas las tareas se realizarían de manera local no sería necesario colgarlas de ningún servicio. En este ejemplo el botón que permitiría llevar a cabo esta tarea sería **Funcionamiento Luminus**, como se muestra en la siguiente figura:



Figura 10.2: Funcionamiento Luminus

El cual, al hacer clic sobre de el, debería redirigir a las pantallas que se muestran a continuación:
Agregar Archivo



Figura 10.3: Agregar Archivo

Esta pantalla se propone que el usuario seleccione el tipo de archivo que desea subir al HDFS. desde su directorio local, o en su defecto desde una base de datos las configuraciones de conexión a la base de datos serian hechas en pantallas posteriores.

Es necesario que el usuario agregue al menos un archivo de datos al HDFS antes de empezar a aplicar algoritmos ya que en caso de no existir ningún archivo sobre el HDFS, no se tendrían datos sobre los cuales trabajar.

Además de que es importante señalar que, los archivos que el usuario tenga en su directorio local no pueden ser leídos o manipulados por LUMINUS, ni por los algoritmos de programación, por lo que este paso es muy importante.

Con esto, se estaría garantizando que los archivos se encuentren sobre la red distribuida.

[**Agregar Archivo CSV**](#)



Figura 10.4: Agregar Archivo CSV

En el caso que el archivo que deseé subir el usuario sea de tipo CSV que son los que actualmente son soportados por la plataforma, no se tendría que realizar ningún ajuste o mejora a los algoritmos.

Como parte de la funcionalidad de esta pantalla se propone realizar un análisis detallado a los datos contenidos en el archivo para que, con el análisis que se haga, se le puede decir al usuario que tipo de dato se cree que contiene cada una de las columnas de su archivo, además de permitirle la opción de cambiar el tipo de dato en caso de que el tipo de dato encontrado al ser validado por el usuario este encuentre que es incorrecto. por otro lado, se propone mostrar un ejemplo de un dato contenido dentro de la columna, esto para que si el usuario no recuerda exactamente que contiene una columna, pueda darse una idea y con esto seleccionar si el tipo de dato que se encontró para esa columna efectivamente es el correcto y en caso de no serlo pueda tener los elementos para cambiarlo con toda tranquilidad.

Para los archivos CSV que no contienen encabezado con nombre de los atributos, se mostrara esta columna vacía y se dejará al usuario que indique su nombre.

Una vez terminado el proceso podrá hacer clic en uno de los dos botones listados; ya sea en **[Aceptar y Terminar]** o bien en **[Regresar]**. y esto lo determina, el hecho de que desee o no, guardar el archivo encontrado con los datos que se encuentran actualmente en la tabla mostrada en la pantalla.

En caso de que se presione **[Aceptar y Terminar]** esta pantalla debería validar que los tipos de dato seleccionados por el usuario puedan ser aplicados para todos los elementos que contiene cada uno de los atributos y si no lo es, mostrar uno de los elementos del atributo que no aplica para el tipo de dato introducido por el usuario, para que entonces el usuario final pueda volver a hacer la selección.

En caso de que todos los tipos de dato sean correctos, se debería hacer una carga de este archivo al HDFS, agregando en caso de que no lo tuviera y hayan sido introducido por el usuario los atributos correspondientes.

Los tipos de datos, seleccionados por el usuario, serían utilizados para alimentar los algoritmos soportados por *Luminus* con los datos que cada uno de ellos soporta. y se tendría que llevar un control de estos tipos de datos, como el programador de este sitio web lo prefiera siempre y cuando no los pierda de vista para tomarlos en cuenta al momento de invocar a los programas.

Agregar XML o Agregar JSON

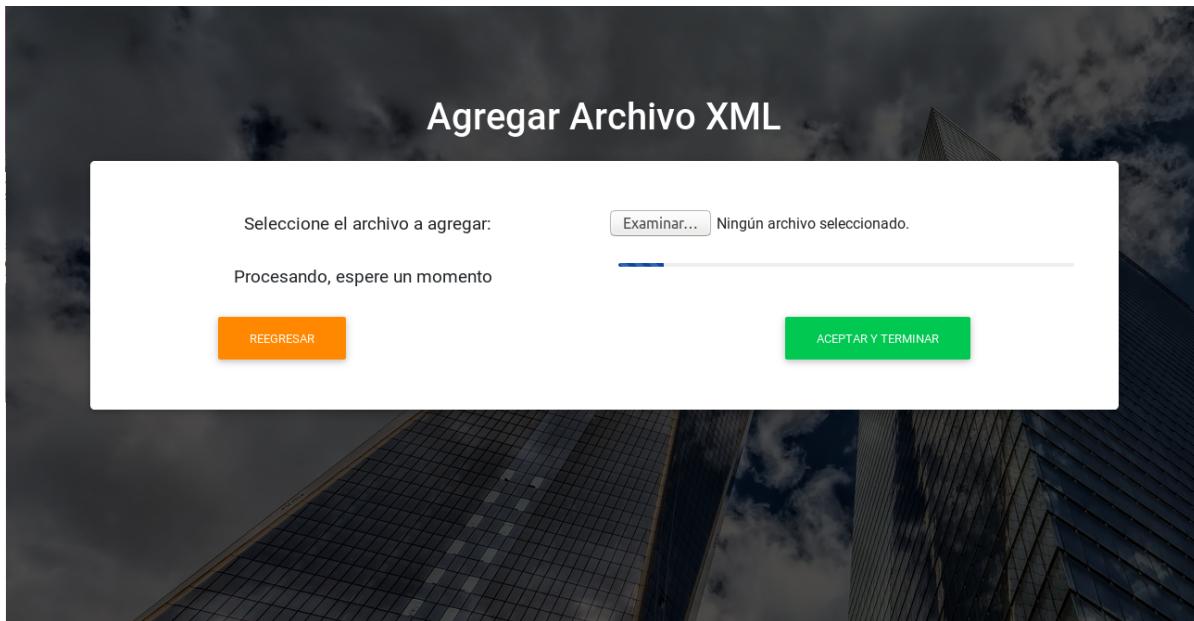


Figura 10.5: Agregar Archivo XML

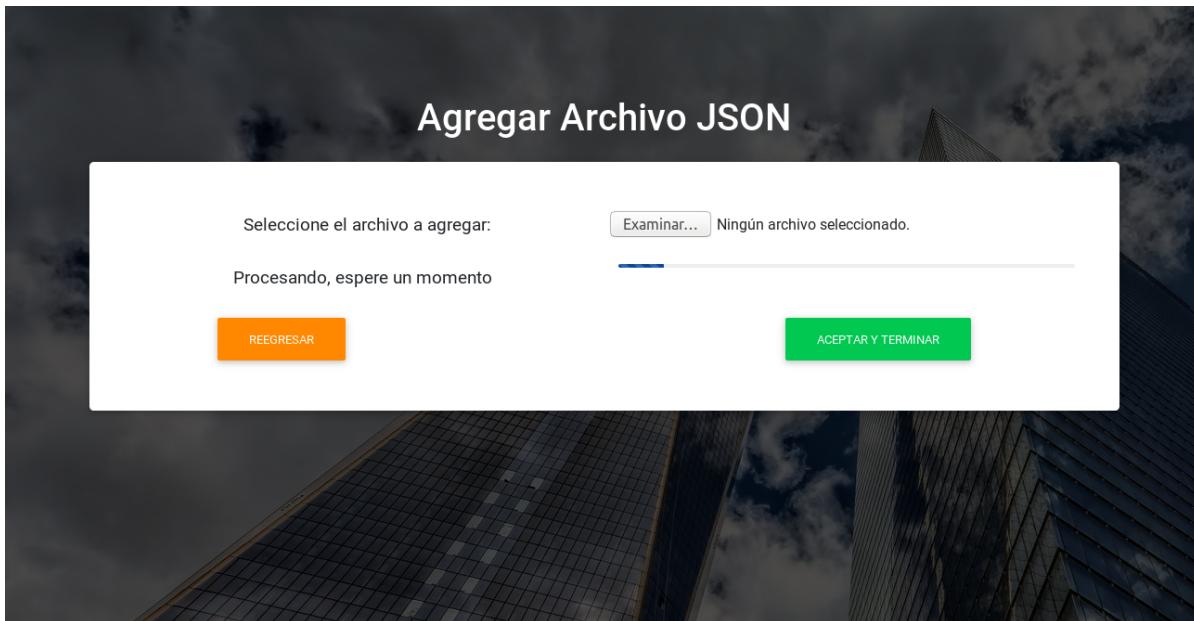


Figura 10.6: Agregar Archivo JSON

Estos tipos de archivo, deben ser cargado desde el directorio local del usuario del sitio web, y no debería requerir ningún tipo de configuración adicional ya que como tal, estos tipos de archivo están diseñados de tan forma que la información que se desea conocer del archivo ya está soportada en la propia estructura del mismo, por lo que no habría que pedir o validar detalles adicionales.

Sin embargo, este tipo de archivos aun no son soportados por los algoritmos implementados en *Luminus*, por lo que, sería necesario realizar alguna de las siguientes opciones:

- Adaptar los algoritmos que actualmente se tienen implementados en *Luminus* para que sean capaces de trabajar

con archivos XML y JSON

- Hacer un tratamiento de los datos para que a pesar de que su origen sea un tipo de archivo distinto (XML, JSON) puedan ser tratados como un CSV y soportados por los algoritmos.

Sin embargo, sin importar la solución que se escoja será necesario realizar la subida de los archivos al directorio del HDFS.

Agregar Base de Datos



Figura 10.7: Agregar Tabla o Vista de una Base de Datos

Algunas veces los usuarios, tendrán sus datos en bases de datos relacionales convencionales, ya que es una manera muy común que utilizan las empresas para el tratamiento de sus datos, en este caso, se puede destinar una tabla o una vista para poder trabajar con los datos que estas nos arrojan.

La vista, en su caso, le permitiría al usuario final, incluir solo algunos campos de una determinada tabla, o bien datos de varias tablas haciendo las uniones entre las tablas que este necesite.

Por lo que una vez que el usuario tuviera lista su vista o bien su tabla, podrá en esta pantalla ingresar las credenciales de su base de datos, las cuales son:

- Dirección IP

- Puerto donde corre la base de datos
- Nombre de usuario
- Clave de este usuario dentro de la base de datos

Bases de datos de diferentes manejadores, pueden establecer una conexión únicamente con estos 4 datos por lo que se considera que estos podrían ser suficientes para testear una conexión.

Antes de realizar el Testeo, será necesario que el usuario indique cual es el nombre de la tabla o vista con la que desea conectarse, después de esto, se puede verificar que la conexión a la base de datos es valida y continuar con el copiado de la tabla o de la vista dentro de el HDFS.

La implementación para base de datos, no es soportada actualmente por los algoritmos de clasificación programados, por lo que se tendría que implementar alguna de las siguientes soluciones:

- Adaptar los algoritmos de *Luminus* para que sean capaces de trabajar con archivos originarios de una base de datos ya sea una tabla o una vista.
- Hacer un tratamiento de los datos para que a pesar de que su origen sea un tipo de archivo distinto (Base de datos) puedan ser tratados como un CSV y soportados por los algoritmos.

En este caso, es necesario considerar, cuantos tipos de bases de datos diferentes pueden ser soportados por los algoritmos debido a que cada base de datos que se maneje tiene diferentes formas de exportar sus datos, y este comportamiento deberá ser considerado al momento de manejarlos.

Ejecutar algoritmo (Selección de Archivo)



Figura 10.8: Algoritmos Propuestos

En esta pantalla se pretende presentar todos los tipos de algoritmos que pueden ser soportados por la aplicación, por el momento ID3 y KNN, en caso de que, se desarrollen mas algoritmos podrían ser listados desde esta misma pantalla.

Sería necesario que el usuario seleccionará del listado mostrado el algoritmo que desea ejecutar.

Algoritmo KNN

A continuación se muestra un ejemplo de como podría operar el algoritmo KNN en esta implementación. no se tiene una propuesta para el algoritmo ID3 pero la lógica que estos debería ser la misma.

Si se alcanza a visualizar la forma en que se pretende presentar este algoritmo, podría diseñarse una solución para cada uno de los algoritmos que sean implementados.

Ejecutar Algoritmo



Figura 10.9: Ejecutar Algoritmo

En este caso, se buscará en el HDFS, el archivo que contiene los datos que se desean aplicar al algoritmo KNN, ya que, el HDFS puede contener mas de un archivo de datos. pero será necesario distinguir cual de ellos sera utilizado para cada ejecución. Una vez que este se seleccione se presionaría **Continuar** .

Selección de atributos a utilizar en el análisis del archivo de entrada



Figura 10.10: Selección de atributos

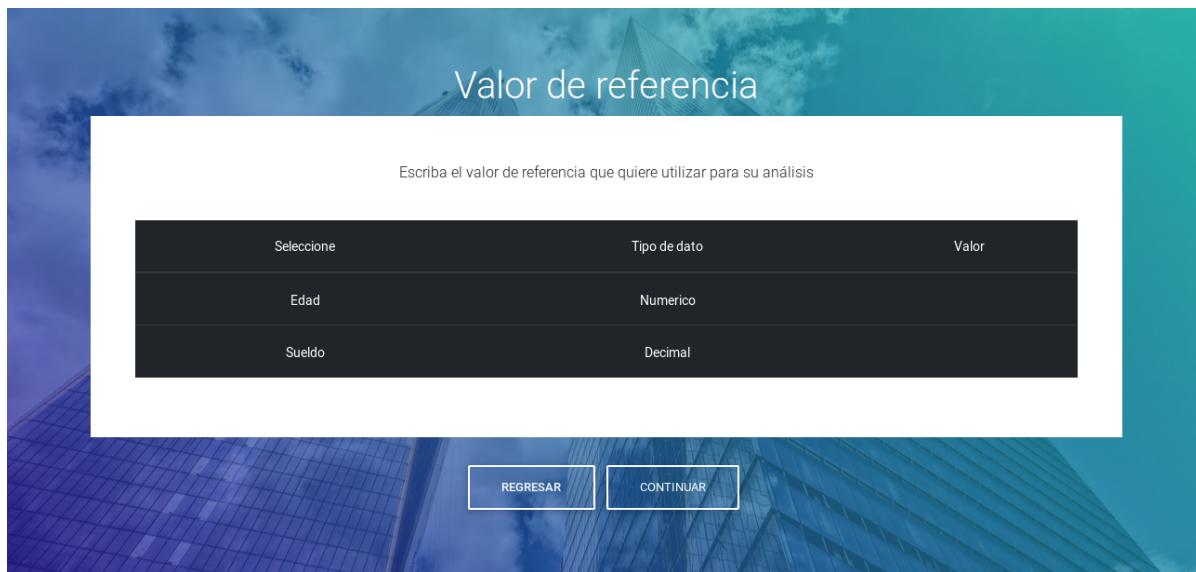
En esta pantalla, se muestran los atributos contenidos dentro del archivo de configuraciones, para que el usuario seleccione los que desea utilizar como parte del procesamiento.

Se propone restringir a que no sea posible seleccionar mas de 5 atributos, ya que entre mas atributos se incluyan en

el algoritmo mas complejo se vuelve su procesamiento, sin embargo, podría no ser restringido y seguiría operando de manera correcta.

Nuevamente, si el usuario desea cambiar el tipo de dato, debería ser permitido. ya que quizá deseé tratar el mismo dato de diferentes maneras para diferentes algoritmos. Pero sería recomendable mantener los que seleccionó anteriormente, para que solo requiera hacer los cambios para los atributos que tengan modificaciones y el resto de ellos no tenga que volver a introducirlos cada que quiera ejecutar un algoritmo.

Valor de referencia del elemento a buscar



The screenshot shows a user interface titled "Valor de referencia" (Reference Value) against a background of a modern skyscraper. The interface includes a text input field and a table for selecting a reference value.

Seleccione	Tipo de dato	Valor
Edad	Numerico	
Sueldo	Decimal	

Below the table are two buttons: "REGRESAR" (Return) and "CONTINUAR" (Continue).

Escriba el valor de referencia que quiere utilizar para su análisis

Figura 10.11: Valor de Referencia

Para todos los datos seleccionados en la pantalla anterior se tiene que dar un valor de referencia para el análisis, este servirá para encontrar el número de vecinos más cercanos. el valor que se ingrese debe ser del mismo tipo de dato, seleccionado en la pantalla anterior.

Número de K a considerar



Valor de referencia

Ingrese el número de K con el que desea alimentar el algoritmo, para un número de K más grande el algoritmo puede tomar mas tiempo

Los valores recomendados estan entre 1 y 11, valores mayores a 11 pierden significado en los resultados.

Se recomienda usar un número impar

Seleccione
 Valor de K

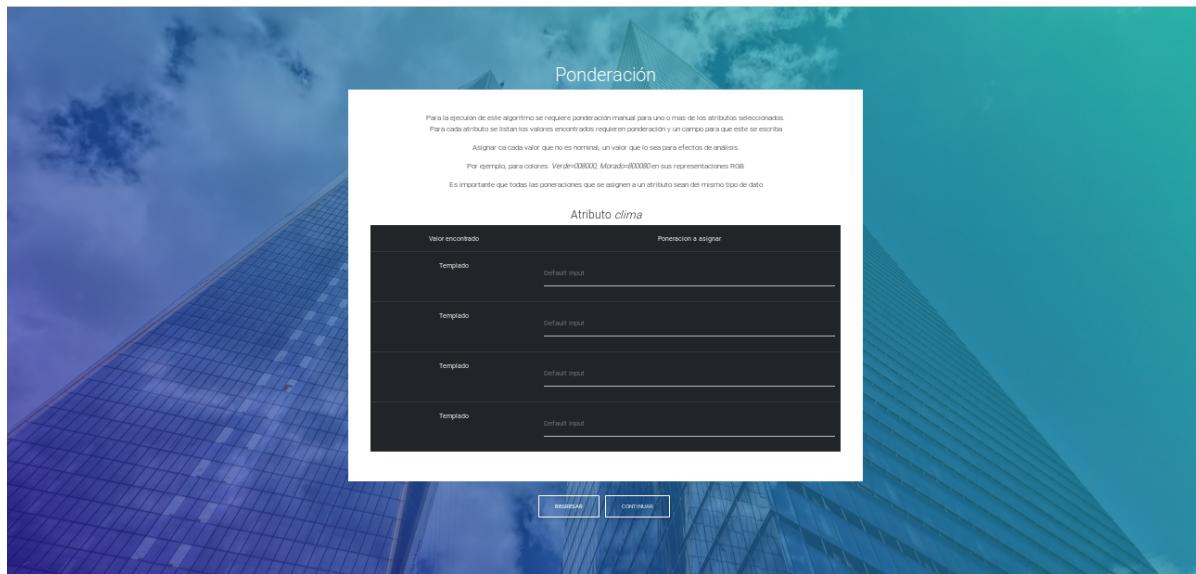
[REGRESAR](#)
[EJECUTAR](#)

Figura 10.12: Valor K

En este caso, para obtener los valores se selecciona un numero de K valido, se recomienda seleccionar este valor entre 0 y 11. para hacer mas simple el análisis. ya que valores de k superiores a 11 la información empieza a perder valor.

esta pantalla únicamente permite seleccionar el valor mas apropiado para esta variable.

Ponderación de atributos no numéricos



Ponderación

Para la ejecución de este algoritmo se requiere ponderación manual para uno o más de los atributos seleccionados. Para cada atributo se instan los valores encontrados requieren ponderación y un campo para que este sea escrito. Agregar co cada valor que no es nómico, un valor que lo sea para efectos de análisis.

Por ejemplo, para colores: Verde#000000 Morado#0000ff en sus representaciones RGB

Es importante que todas las ponderaciones que se asignen a un atributo sean del mismo tipo de dato

Atributo clima	Ponderación a asignar
Valor encontrado: Templado	Ponderación a asignar: Default Input
Templado	Default Input
Templado	Default Input
Templado	Default Input

[REGRESAR](#)
[CONTINUAR](#)

Figura 10.13: Ponderación

Un elemento que se puede agregar a las consideraciones hechas es la ponderación de atributos no numéricos. Se trata de valores, que no pueden ser trabajados de manera inmediata por los algoritmos, por lo que hay que asignarles un valor de manera manual a cada uno de los elementos del atributo. Esto es viable mientras que, el número de elementos dentro de un atributo no haya crecido demasiado, ya que para atributos con muchas opciones esto se volvería muy pesado.

Sería necesario establecer un número máximo de valores diferentes permitidos para que, atributos que tengan mas elementos que ese numero establecido no puedan ser configurados de esa forma.

Todas las ponderaciones que se establezcan, deben de ser del mismo tipo de dato, y debe de configurarse el archivo de tal forma que cuando se pase a los algoritmos pueda leer los datos ponderados para que el algoritmo que se pretende utilizar funcione de manera correcta.

Al inicio de la sección Trabajo a Futuro se detalla un poco mas acerca de que significa hacer las ponderaciones y se detalla un ejemplo en caso de que se requiera mas información al respecto.

Ejecución del algoritmo



Figura 10.14: Ejecución del algoritmo

Cuando se realicen todas las configuraciones necesarias, entonces se procede a ejecutar el algoritmo pasandole las configuraciones establecidas y se devuelve al usuario un archivo de salidas en el formato mas recomendable para el algoritmo que se este ejecutando. con los resultados obtenidos de la ejecución del algoritmo.